PACKED COMPUTATION OF EXACT MEANING REPRESENTATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Iddo Lev

June 2007

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Stanley Peters) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Christopher D. Manning)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Michael R. Genesereth)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Richard Crouch)

Approved for the University Committee on Graduate Studies.

Abstract

An important goal in Natural Language Understanding (NLU) is improving accuracy in NLU tasks. Accuracy is paramount in "exact NLU" applications, such as solving word problems (logic puzzles, math/physics/chemistry questions), understanding regulatory texts and controlled language, and NL interfaces to databases. These applications require exact meaning representations that incorporate knowledge of structural semantics, i.e. how function words (quantifiers, connectives, comparatives, etc.) and sentence structure affect the meaning of sentences. Exact meaning representations allow computers to capture and integrate information that appears throughout documents and to draw appropriate inferences from it. Even in other NLU applications such as question answering, exact meaning representations can help improve accuracy of understanding function words as well as in information integration.

Three major questions pertaining to exact meaning representations are:

1. How can the representations be calculated given one syntactic analysis of a sentence?

2. How can all meaning representations for an ambiguous sentence be calculated efficiently given a packed syntactic analysis (parse forest)?

3. How can semantic coverage be extended to complex linguistic constructions?

This dissertation addresses these questions. I show how the syntax-semantics interface can be specified more naturally than in traditional approaches by using the framework of Glue Semantics (linear logic). I then develop a novel algorithm that efficiently computes a packed meaning representation given a packed syntactic analysis – this combines Glue Semantics with Palo Alto Research Center's "choice-space packing" framework for

ambiguity management. Finally, I extend the coverage of semantic analysis to complex linguistic constructions, including comparatives, reciprocals, and *same* and *different*, where the mapping from syntax to semantics is convoluted.

Therefore, it is now possible for computers to translate a richer array of sentences in "exact NLU" texts to exact meaning representations, and to do so efficiently. Thanks to the flexible framework here, it will now be feasible to extend linguistic coverage to more complex constructions. This work can also help with resolution of NL ambiguities: exact meaning representations can be integrated with domain knowledge to reduce ambiguities, and their packed computation makes it possible to do this more efficiently using both logical and statistical methods.

Preface

The question of how a computer can "understand" natural language texts, in the sense that it can correctly perform tasks based on the information conveyed by the texts, has been studied in the field of Natural Language Processing (NLP) for a long time. Many sub-questions need to be addressed, and in particular:

1. How should the meaning of natural language texts be represented?

2. How are the representations related to the text's syntactic structures?

3. How can the representations be effectively computed from the input text? In particular, what should be done about natural language ambiguities?

4. How can computers perform reasoning by using the information conveyed by the meaning representations?

The first two questions have also been studied by linguists (semanticists) for a long time. The four questions are important because it would be immensely useful if a computer could "understand" at least some aspects of a text's content well enough that it could draw correct conclusions from it, answer questions about it, and integrate the information with other knowledge it has. This would be useful in many applications, including question answering, dialogue systems, and machine translation.

The traditional approach to the second question, starting with Montague (1974), is to have a simple one-to-one mapping between syntactic and semantic rules. While this idea works well for artificial languages (in logic and computer science), it is too simplistic for natural language. First, there isn't always a simple one-to-one match between parts of syntactic structures and parts of the corresponding semantic structures. Second, natural

language has semantic ambiguities, which require one syntactic structure to be related to more than one meaning representation.

This led researchers to propose various extensions to the simple syntax-semantics scheme, but these brought only limited success. Thus, type-shifting rules (e.g. Emms (1992); Hendriks (1993)) are complex, and many such rules are needed to deal with various possibilities for quantifier positions and scopings. Storage methods (Cooper, 1983; Keller, 1988) and QLFs (Hobbs and Shieber, 1987; Moran and Pereira, 1992) are ad-hoc constructions – the manipulations of pieces of formulas are not justified by an underlying model-theoretic semantics (they do not have clear nontrivial identity criteria). Underspecified representations (Blackburn and Bos, 2005a; Copestake et al., 2005; Egg et al., 2001) have the same problem, and are also complicated to work with because the engineer needs to manipulate pieces of constraint graphs rather than simple meaning terms. And Categorial Grammar (Morrill, 1994; Carpenter, 1998; Steedman, 2000) requires the semanticist to worry about surface word order, instead of leaving this to the exclusive purview of the syntax. Moreover, none of these methods provides a good answer to the question: how can the packed syntactic representations (parse forest), which a chart-parser outputs, be translated in a packed way to a packed semantic representation, which efficiently represents all syntactic and semantic ambiguities.

A further problem is that the semantic coverage of existing systems is quite limited. Although there are analyses of more complex constructions in the linguistic literature, they are not presented in a unified way in one place. A main challenge that one is confronted with when trying to specify how a meaning representation should be calculated for a NL sentence is that one is usually interested in information beyond the literal meaning of the text. But even for the literal meaning, sometimes there seems to be too big of a mismatch between the material that is explicit in the text and the required meaning representations.

These difficulties led some people in NLP to see it as too difficult at this time to work on extending the linguistic coverage of precise translations from syntax to exact meaning representations. Instead, they resort to rather crude approximations, such as using structural matching between the syntactic structures of a sentence and a query in lieu of semantic inference between them. This may be useful in some applications, but it does not address the real issues of semantic representation and reasoning. Exact meaning representations are paramount in exact NLU applications, such as solving word problems (logic puzzles, math/physics/chemistry questions), understanding regulatory texts and controlled

language, and NL interfaces to databases. In these applications, even an analysis that is only slightly wrong may lead to completely wrong final results. This is because answers to queries do not appear explicitly in the text and must be inferred from it by understanding and combining the information that appears throughout the text. This information therefore needs to be represented with high quality. Even in other NLP applications that can tolerate some error, exact meaning representations can help improve accuracy of understanding functional words, and help with information integration, which must rely on the content and not only the syntax of the texts.

Therefore, this dissertation addresses the first three questions above, and makes important progress towards solving the existing obstacles.

The problem of structural mismatch between syntax and semantics is solved here by using a more sophisticated framework for specifying the syntax-semantics interface, along the lines of more modern constraint-based linguistic formalisms (such as HPSG and LFG). This framework is called Glue Semantics (GS). Instead of using a simple scheme of combination (only lambda application) and complicating the semantic entries of words with higher-order expressions or pieces of underspecified representations, GS uses a more powerful combination scheme (based on linear logic), and simpler meaning terms. The framework allows the engineer to specify, in a simple way, constraints on how the pieces may combine. In this way, the pieces combine with each other as in a jigsaw-puzzle, except there may be more than one solution (this allows us to model semantic ambiguities). Although the constraints are based on information in the syntactic structures, the constraints can look at non-local information there, and are not forced to adhere to the hierarchy of the parse trees. This flexibility provides a more "industrial strength" tool that can rely on the output (F-structures) of a linguistically-sophisticated, hand-written, broad-coverage grammar of natural language in the XLE system developed at PARC (Palo Alto Research Center).

In this dissertation, I provide a new introduction to GS, which unlike previous introductions, does not rely on prior knowledge of LFG (Lexical Functional Grammar), and does not require the reader to first understand linear logic. I also show how to actually implement a GS specification that takes the syntactic output produced by the XLE and creates appropriate statements in GS that can then combine together to form meaning representations.

To deal with both semantic and syntactic ambiguities, this dissertation uses a general framework for ambiguity management called "choice-space packing", which was also developed as part of the XLE. The basic idea is that no single stage in the linguistic processing pipeline has complete information to allow it to select the one correct analysis in context. An analysis which is locally most likely (e.g. in the syntax stage) may later turn out to be globally incorrect, while a locally less likely analysis is the globally correct one. Therefore, all analyses should be computed together in an efficient, packed way – this is a generalization of the idea of packing in chart parsers for context-free grammars. The question of how to extend this idea of packing to the next stage of semantic composition, encompassing both syntactic and semantic ambiguities, was an open problem. In particular, it was unknown how to integrate choice-space packing with the framework of Glue Semantics.

In this work, I provide a novel algorithm that solves this problem, and this is a major result of this dissertation. To develop this algorithm, I had to reformulate an existing basic algorithm for (unpacked) Glue Semantics (Hepple, 1996), and to carefully extend the definitions of all the operations in the reformulated algorithm in order to relativize them to the choice-space framework. I also provide mathematical proofs for the correctness of both the basic algorithm (that was never proved correct), and for my packed algorithm.

There is no one single solution to the issue of broad semantic coverage, because there are different kinds of semantic phenomena and each poses its own challenges. It is beyond the scope of this dissertation to address the non-literal meaning of texts (such as metonymy and user intensions). For the literal meaning, a main challenge is that when looking at NL sentences, it is not always clear what part of the computation of meaning representations should be done by the semantic composition module and what should be left to other modules (such as: anaphora and ellipsis resolution). Sometimes, attempts were made to do too much in the semantic composition module, and this was either impossible or led to a high cost. This problem was exacerbated because the syntax-semantics mapping scheme was not flexible and powerful enough to be practical for creating a broad enough coverage of semantic phenomena.

This dissertation makes important steps towards a solution for several semantic constructions. Several factors contribute to the solution. First, the flexible GS framework in combination with the broad-coverage precise grammar of the XLE make it more feasible than previous work to develop and write the specification for the syntax-semantics interface. In particular, I show how reciprocal expressions do not require any special additional

mechanisms. Second, I follow the approach of "separation of powers", where the semantic composition module is not given more load that it can naturally handle, and some of the work is done by additional modules. Thus, anaphora resolution is not done by the composition process but by a separate module that uses DRS-like structures. Furthermore, the specification is written in such a way that DRSs appear only when they are needed rather than permeating all entries. In the case of numerical comparatives, gradable comparatives, and *same/different*, I carefully delineate what should be done by compositional semantics and what should be done by an independently-motivated module for ellipsis resolution. This allows the semantic composition to be simpler and not to try to copy abilities that are also needed in the ellipsis module. Third, I arrive at these conclusions by examining a broad range of cases rather than just a few examples.

As a result of my work, it is now possible for computers to translate a richer array of sentences in "exact NLU" texts to exact meaning representations, and to do so efficiently in the face of ambiguities. Thanks to the flexible framework here, and the initial work on semantic coverage, it will now be feasible to extend linguistic coverage to more complex constructions. This work can also help with resolution of NL ambiguities: the exact meaning representations of textual information can be integrated with domain knowledge to reduce ambiguities, and their packed computation makes it possible to do this more efficiently using both logical and statistical methods.

In this dissertation, I assume the reader is familiar with first-order logic, basic concepts in NLP such as parsing, and has a basic knowledge of linguistic semantics at the level of an introductory course. Readers unfamiliar with any of these may find it useful to consult (Gamut, 1991a) for logic, (Allen, 1995) for NLP, and (de Swart, 1998; Gamut, 1991b) for semantics. Although I do not assume familiarity with (Blackburn and Bos, 2005a,b) for the most part, these two books provide very useful background to the research direction and some of the methods employed here.

Acknowledgements

I would like to express my deepest thanks to my adviser, Stanley Peters. I am very grateful to him for giving me complete freedom to pursue my research interests, and for being patient with me when I wasn't sure what my research direction was precisely. I am glad that I had an adviser with such broad and deep knowledge and experience in diverse fields that are relevant to my research, including linguistics, computational linguistics, and logic. I learned a lot from him both during the seminars he taught and in numerous one-on-one conversations. Stanley has a talent to bring out the general issues and big questions that underly the particular details being discussed, and he helped me clarify to myself what I am really after in my research and why, which are questions that I struggled with for a long time. I thank Stanley for his encouragement, constructive advice, guidance, and financial support over the last five years.

I am very grateful for the opportunity that I had during summer 2006 to do an internship with the Natural Language Theory and Technology group at PARC (Palo Alto Research Center), where I did the bulk of the work on combining glue semantics and choice-space packing that is reported in this dissertation. I am especially thankful to Dick Crouch, who was my adviser at PARC, for many meetings I had with him during my years at Stanford in which I learned a lot from him about glue semantics and packing, and for his support and encouragement for my work. Thanks also to the other members of the group for their help, especially Valeria de Paiva, Tracy King, Cleo Condoravdi, and John Maxwell, and thanks to the other interns Rowan Nairn and Joshua Johanson who made the experience of working at PARC even more fun.

I also thank Chris Manning for coming up with the idea of building a system for solving GRE logic puzzles. Once I heard about this, I immediately knew that this is the research project I was looking for all along. I am grateful for the collaboration on the logic puzzles

project during 2003-4 that I had with him and his NLP group, especially Bill MacCartney, Roger Levy, and Kristina Toutanova, for their hospitality in the NLP lunches, and for useful discussions about my research.

I was glad to be a teaching assistant in Mike Genesereth's computational logic course. I thank him and his research group, especially Tim Hinrichs, as well as Selene Makarios, for useful feedback and help regarding the automated reasoning aspects of my project.

Thanks also to the chair of my defense committee, Johan van Benthem, whom I first met when I attended his fascinating class "Logic, language, and information."

For helpful discussion about issues relevant to the thesis, whether in person or over email, I thank Avery Andrews, Johan Bos, Ralph Campbell, Rui Chaves, Iván García Álvarez, and Ivan Sag.

My time at Stanford would not have been as pleasant without the friends I met here: Bill, David, Dmitry, Eran, Florian, Ira, Jessica, Mihaela, Nicolas, Paul, Sam, Uriel, and Yonatan.

Many thanks to my parents Malka and Dan and my brother Ron for their love, support, and encouragement over the years. Finally, last but not least, I give immense thanks to Beth, who gives me endless love and support every day.

Contents

List of Figures

Part I

Preliminaries

Chapter 1

Introduction

This dissertation investigates a framework for specifying how a computer can calculate exact meaning representations given a syntactic analysis of a text, and provides an algorithm by which the computer can calculate all meaning representations compactly given a packed syntactic analysis. The dissertation further provides computationally-grounded, linguistic, structural semantic analyses of many complex natural language constructions, and shows that they can be specified naturally using the framework. The framework is called Glue Semantics.

This chapter addresses the following questions:

- Why should people who are developing means for Natural Language Processing (NLP) care about exact meaning representations and structural semantics? (sections 1.1 and 1.2)
- Why use a new framework for specifying structural semantics? (section 1.3.2)
- Why do we need packed meaning representations? (section 1.3.3)
- Why should NLP care about linguistic analysis of natural language constructions? (section 1.3.4)

I address these questions because they provide important motivation for the work presented here.

1.1 Background: Exact NLU

1.1.1 What Is It?

A central problem when designing Natural Language Understanding (NLU) systems is how to improve their accuracy as far as the information needs of the user are concerned. Let us take this problem to its extreme for a moment.

There are some NLU tasks where complete accuracy is all-important. They require exact calculation of NL meaning and consequences, while results that are "almost correct" ("only slightly wrong") are not good enough. Many of these tasks have exactly one correct answer that everyone agrees on. Most importantly, the NL input texts were originally written with a back-end unambiguous formalization in mind, which is intended to be fed into a rigorously-defined mathematical procedure for solving some type of problem. Thus, although in the general case of NLU there is no guarantee that the NL input can be precisely formalized or that the task has a precise result that all humans could agree upon, these things are in fact guaranteed by definition in the context of exact NLU. I will now give a few examples of such applications.

1.1.2 Examples of Exact NLU Applications

Controlled Language

A controlled language is a restricted subset of NL which is so precisely delineated that it effectively becomes a formal language which nevertheless looks (almost) like NL.[1] Such languages have been used in manuals and other technical texts in various areas of industry (such as the aerospace industry[2]) and in other specification tasks (see e.g. (Nelken and Francez, 1996) and (Fuchs et al., 2006)[3]). The motivation for these applications is that a controlled NL maintains the complete precision and predictability of a formal language while making it easier and more natural for humans to learn it (compared to a formal language). Here is an example text written in Attempto Controlled English, specifying the operation of an ATM (from (Fuchs, 2005)):

(1) Every customer has at least 2 cards and their associated codes. If a customer C approaches an automatic teller and she inserts her own card that is valid carefully into the slot and types

[1] See e.g. http://www.ics.mq.edu.au/∼rolfs/controlled-natural-languages/.
[2] See http://www.boeing.com/phantom/sechecker/se.html.
[3] See also http://www.ifi.unizh.ch/attempto/.

> the correct code of the card then the automatic teller accepts the card and displays a message "Card accepted" and C is happy. No card that does not have a correct code is accepted. It is not the case that a customer's card is valid, and is expired or is cancelled.

This text is quite complex and sounds almost completely natural. The ACE engine automatically translates it and similar texts to correct first-order logic (FOL) representations of the specifications conveyed by the texts. One way to demonstrate correctness is to show that NL sentences (in the controlled language) which humans judge to follow from the text are in fact translated to FOL formulas that logically follow from the text's representation, whereas sentences that are judged not to follow are translated to formulas that do not logically follow from the text.

General NL is inherently imprecise, so how is the computer able to accurately understand such texts? This is possible thanks to two things: First, the human who wrote this text had a precise FOL end-formalization in mind. Second, the allowed NL input is restricted to avoid imprecision and to transcend particular contexts. Moreover, potential ambiguities are usually handled in such systems by fiat: When the computer encounters a NL construction which is ambiguous in general, a specific one of its possible interpretations is always selected, and the definition of the controlled language declares in advance which one it is. For example, PP attachment ambiguity is always resolved by attaching the PP to the preceding NP rather than the VP. If the user wants to convey the alternative choice, she needs to use a paraphrase (e.g. putting the phrases in a different order or using an explicit relative clause), or to use some artificial means such as a comma. This practice makes the NL input at times a little less than completely natural, but it is an effective compromise.

The research direction I want to pursue shares with controlled languages the goal of precisely understanding the user input (by this I mean: correctly perform tasks based on the information conveyed by the texts). However, I do not want to place any a-priori restrictions directly on the possible range of input NL constructs or what they mean. Instead, the restriction is placed on the application as a whole – it should have a precise back-end formalization. This overall restriction ensures that, by definition, what each input sentence means and what should be done with it under difference circumstances is completely clear *when the context of the application is taken into account*. This restriction may indirectly lead to some restrictions on the range of allowed NL phenomena in the input, but this need not be the case.

Despite the differences, research on controlled languages still aims to gradually expand the range of linguistic constructions the system can handle. In that respect, the work of this dissertation can directly feed into that endeavor.

Word Problems

There are many types of problems in physics and mathematics that have well-understood and mathematically-precise procedures for solving them. These are so precisely defined that it is possible to write computer programs that carry out these calculations correctly. The input to these programs usually needs to be expressed in a formalized language.

Students who are taught these procedures are given practice problems to improve and test their skills of using these procedures. The practice problems are usually given not in a formalized way but using a NL description of the problem. The writers of the text start from a formalized problem and then convert it to a NL description. There is usually more than one NL text that can describe the same formalized problem. The student's task is to understand the NL description and rediscover the formalized problem, and then apply the appropriate procedure to solve it. When the student reads the NL text, he relies on the assumptions that there is a precise end-formalization from which the NL text was created, and there is exactly one correct answer to the word problem.

Given these assumptions, solving physics, chemistry, math, or logic problems from their NL descriptions is a central example of exact NLU applications. Even if the NL input text contains parts that, taken out of context, pose serious difficulties for automated understanding because they are imprecise, ambiguous, and depend on a complex psychological model of the writer's intentions, all these issues can be set aside when the text is understood within the context of the word problem solving application.

An example of a logic puzzle is shown in Figure 1.1. Such logic puzzles appeared in LSAT and GRE exams. A precise formalization of the third constraint in the puzzle could be expressed using the following FOL formula:

$$(2)\ \forall x.[(room(x) \wedge \textit{exhibited-in}(E,x) \wedge \textit{exhibited-in}(F,x)) \rightarrow \\ \neg\exists y.sculpture(y) \wedge y \neq E \wedge y \neq F \wedge \textit{exhibited-in}(y,x)]$$

This dissertation is part of a larger project that aims to create a system that can solve such puzzles given their textual descriptions.[4] In (Lev, 2006), I proposed the creation of

[4]For more information, please see the project's website: http://www-csli.stanford.edu/∼iddolev/pulc.

Preamble: Six sculptures – C, D, E, F, G, and H – are to be exhibited in rooms 1, 2, and 3 of an art gallery. The exhibition must conform to the following conditions:

(1) Sculptures C and E may not be exhibited in the same room.
(2) Sculptures D and G must be exhibited in the same room.
(3) If sculptures E and F are exhibited in the same room, no other sculpture may be exhibited in that room.
(4) At least one sculpture must be exhibited in each room, and no more than three sculptures may be exhibited in any room.

Question 1: If sculpture D is exhibited in room 3 and sculptures E and F are exhibited in room 1, which of the following may be true?

(A) Sculpture C is exhibited in room 1.
(B) No more than 2 sculptures are exhibited in room 3.
(C) Sculptures F and H are exhibited in the same room.
(D) Three sculptures are exhibited in room 2.
(E) Sculpture G is exhibited in room 2.

Question 2: If sculpture G is exhibited in room 1, which of the following may NOT be a complete list of the sculpture(s) exhibited in room 2?

(A) Sculpture C (B) Sculptures E and H (C)...

Adapted from Weber (1999).

Figure 1.1: Example of a logic puzzle text

a test-suite of logic puzzle texts. The advantage of this test-suite is that in contrast to other word problems which may require complex knowledge of the subject matter, solving GRE/LSAT logic puzzles relies on very simple procedures (essentially, finite constraint-satisfaction). Thus, we can focus our research efforts on the NL understanding part without also having to worry about a difficult back-end computation (as in dialogue systems, for example). Furthermore, while solving math and physics problems requires access to a rich ontology of concepts and a related lexicon, the body of lexical and world knowledge (or commonsense knowledge) required for solving such logic puzzles is quite small.[5] The major research challenge is that logic puzzles require an accurate understanding and merging of the logical constraints expressed *throughout* the text – even a small misunderstanding usually leads to completely wrong answers.

If the domain knowledge required for solving a class of word problems can be precisely formalized, the knowledge and methods developed for solving logic puzzles would be useful in a NL front-end that translates the word problems to formalized representations. A good

For preliminary work on a system for solving logic puzzles, see (Lev et al., 2004).

[5]This is not true for more general logic puzzles, such as in (Smullyan, 1978), which, among other things, require meta-reasoning (reasoning about the truth or falsity of statements), reasoning about agents' beliefs, world knowledge about things such as what humans can infer from what they see when they stand in a circle with other humans, and complex formalizations as in (McCarthy, 1981).

1. Ginger, over the course of an average work-week, wanted to see how much she spent on lunch daily. On Monday and Thursday, she spent $5.43 total. On Tuesday and Wednesday, she spent $3.54 on each day. On Friday, she spent $7.89 on lunch. What was her average daily cost?

(A) $3.19 (B) $3.75 ...

2. During a 5-day festival, the number of visitors tripled each day. If the festival opened on a Thursday with 345 visitors, what was the number of visitors on that Sunday?

(A) 345 (B) 1,035 ...

Adapted from http://www.testprepreview.com.

Figure 1.2: Examples of simple math question texts

example is math questions such as those on SAT exams – sample questions are shown in Figure 1.2. Although solving such questions requires some domain knowledge of math, this knowledge is simple enough that it does not pose a large AI problem.[6]

Recently, a knowledge-based system developed in project HALO[7] was able to get a reasonably high score on a chemistry AP test after a chemistry textbook and the test were encoded in a knowledge representation language. A natural next step to work on is to try giving the computer the ability to read and understand the NL texts directly and translate them to the knowledge representations.

Computational Law

A real-world application that has some similarity to solving logic puzzles is understanding regulatory texts, such as a website that describes which collection of courses a college student must take in order to fulfill the requirements of a study program. Here is an example:[8]

(3) A candidate is required to complete a program of 45 units. At least 36 of these must be graded units, passed with an average 3.0 (B) grade point average (GPA) or higher. The 45 units may include no more than 21 units of courses from those listed below in Requirements 1 and 2.[9]

[6]Solving math puzzles given their English descriptions was one of the tasks that was suggested in discussions of the next DARPA Grand Challenge. But I think that logic puzzles have more interesting NL phenomena than math questions (and their texts are usually longer).

[7]http://www.projecthalo.com/

[8]For a general motivation for the field of computational law, see http://complaw.stanford.edu.

[9]From: http://cs.stanford.edu/Degrees/mscs/degree.php.

As with logic puzzles, regulatory texts describe general rules and conditions that must be met. The computer should be able to answer questions about these rules as well as questions based on an additional description of a particular real or hypothetical situation (e.g. check whether the set of courses a student has taken conforms to the regulations). Also similarly to logic puzzles, answers to such questions rarely if ever appear explicitly in the text, and must be *inferred* from it. Most legal texts cannot be formalized precisely because they are written to be deliberately vague and because of other reasons, but some of them can, and they are good candidates for exact NLU.

NL Interface to Databases and Knowledge Bases

Instead of querying a database using a formal language like SQL, it would be very useful for users if the computer could understand NL questions and commands such as:

(4) a. Which department has the largest number of employees?
b. How many employees were paid a salary higher than $70,000 over the last two years?
c. Update: John Smith works in the personnel department.

There are two approaches to designing a NL interface to a database (NLIDB) (see (Androutsopoulos et al., 1995) for a survey). One approach is to direct the user to ask any question on a certain topic, and then try to use several existing databases that may have relevant information to answer the question. In this case, there is no guarantee that the conceptual understanding of the domain that the user has in mind aligns well with the conceptual view that exists implicitly in the design of the database.

Another approach is to instruct the user that the interface is essentially a database interface language such as SQL, except that it is expressed in NL (see e.g. (Nelken and Francez, 2000; Androutsopoulos, 2002)). In particular, the user should only expect the system to understand very straightforward requests, just as those possible in SQL. The only thing the user should not need to worry about in advance are potential NL ambiguities because they should be resolvable either through interaction with the user or automatically based on a combination of the data in the database and assumptions about the context of the application (as with any other exact NLU applications).

To demonstrate this, consider the question "Did at least two supervisors attend every repair session?". This question has a potential scope ambiguity between the two readings:

(5) a. $\textit{at-least}[2](\textit{supervisor}, \lambda x.\textit{every}(\textit{repair-session}, \lambda y.\textit{attend}(x, y)))$
b. $\textit{every}(\textit{repair-session}, \lambda y.\textit{at-least}[2](\textit{supervisor}, \lambda x.\textit{attend}(x, y)))$

However, if the application is supplied with domain knowledge that says it is impossible for one supervisor to attend all repair sessions, then the only relevant reading is (5)b. If in fact the answer to that query is *true* in a given database, the system could answer "yes" even though the answer to the query (5)a is *false* in that database. If such knowledge is unavailable and the system cannot disambiguate the question, it should say so to the user. In effect, this is treating NL as a formal language except that in some contexts, an expression may contain an actual ambiguity and the system has to report this (just as a compiler could report an unresolvable ambiguity of a polymorphic operator in a particular context of use).

A similar application but more complex is NL interfaces to knowledge bases (KBs). There are many KBs that were designed to model a particular domain such as intelligence analysis. Existing interfaces to such KBs involve rigid forms, which can only address simple queries, or complex formal languages, which require expertise on behalf of the user. It would be helpful to connect the formalized knowledge with a NL front-end. This is more complex than NLIDBs because chains of inference may be involved. As with the exact NLU view of NLIDBs above, the user should be instructed not to expect any more than the formal interface provides.

1.1.3 Why Exact NLU?

Working on exact NLU applications is a useful endeavor for two reasons.

The scientific perspective is that no system exists today that can read an unseen word problem text in a certain domain and solve it correctly, and it would be interesting to investigate how to construct such a system. Similarly with the other applications. If the computer did manage to understand these complex texts and correctly solve the problems they describe, that would necessarily mean that we have discovered something non-trivial about NL, and have learned things that have implications for linguistics and psychology.

The technological perspective is that, as we have seen above, there are in fact useful real world applications that can be classified as exact NLU or a closely related rubric. The practical motivation for exact NLU is that not compromising precision for breadth of coverage is essential for system reliability, which is required for specification languages and

certain NL interfaces. Here is a relevant quote from (Popescu et al., 2003):

> NLIs are only usable if they map NL questions to SQL queries correctly. As Schneiderman and Norman have argued, people are unwilling to trade reliable and predictable user interfaces for intelligent but unreliable ones.
>
> ... To satisfy users, NLIs can only misinterpret their questions very rarely if at all. Imagine a mouse that appropriately responds to a 'click' most of the time, but periodically whisks the user to an apparently random location. We posit that users would have an even worse reaction if they were told that a restaurant was open on a Sunday, but it turned out to be closed. If the NLI does not understand a user, it can indicate so and attempt to engage in a clarification dialog, but to actively misunderstand the user, form an inappropriate SQL query, and provide the user with an incorrect answer, would erode the user's trust and render the NLI unusable.

Exact NLU tasks are a special subset of the space of all NLU applications, but if we can figure out how to do them, we will have learned valuable lessons and developed useful tools that could help us improve the accuracy of other NLP applications as well, in which less-than-perfect levels of accuracy are acceptable.

1.1.4 What Is Needed for Exact NLU?

Because of the high accuracy required in exact NLU tasks, simple approximations ("shallow" approaches) won't work. Thus, "bag of words" approaches are wholly unsuitable. Even structural matching between syntactic analyses of text and questions, which is popular in information extraction and question-answering systems (such as (Pasca and Harabagiu, 2001)), will provide very little. This is because the answer to a query does not appear explicitly in the text and needs to be *logically inferred* from it. In order to do that, the computer needs to combine information/meaning that appears *throughout* the text, and not simply match the query to each text sentence separately. To extract the information from the text reliably, the computer needs to rely on high-quality meaning representations and linguistic knowledge.

Constructing a system for an exact NLU task requires answering at least the following questions:

1. What **semantic representation language** should we use to express the meaning of the NL texts? The representations should be sophisticated enough so that they could support the inferences that are needed in the application. For that purpose, they should have a well-defined interpretation which allows us to understand them and predict what inferences they support (see e.g. (McDermott, 1978) for more on this point).

2. What do particular NL sentences **mean**, and how should this meaning be represented using the semantic representations? The answer is sometimes not clear if a sentence is taken out of context (e.g. how many readings does the sentence "four boys lifted five pianos" have?), and a linguistic semantic investigation is required.

3. How can the representations be **calculated** from NL texts? To answer this, we need to know how the morphology and syntax of the input texts can be calculated. We also need to know what are the algorithms and linguistic knowledge that comprise the **syntax-semantics interface**, which specifies how the semantic representations are calculated from the syntactic analysis of the input NL text.

4. How should **NL ambiguities** be handled? This includes: how should the representations be calculated efficiently, without an exponential explosion, and how should the sentences be disambiguated?

5. How can the representations **support inference**?

6. NL texts usually do not supply explicitly all the **domain and world knowledge** that is needed in an application, because they were written for human consumption and so they assume the reader has this knowledge. How can this knowledge be acquired?

1.2 Structural Semantics

In NLP, a lot is known about how to calculate morphological and syntactic analyses of NL input, and to some extent, how to deal with lexical semantics. Much less is known about the next step of structural processing, namely structural semantics. However, this is one of the crucial pieces of knowledge that is needed in exact NLU applications. I now explain what this knowledge is, why it is needed and useful, and what its limitations are.

1.2.1 What Is Structural Semantics?

Roughly speaking, the branch of linguistics called *structural semantics*[10] deals with the literal meanings of sentences after these meanings are abstracted away from the concepts that non-function words express. It is similar to the semantics of a logical language, where what the symbols in the non-logical vocabulary stand for is irrelevant for defining the semantics of the language abstractly, and the only thing that matters is the syntax of the language and the meaning of the logical connectives. Structural semantics explains how the structural meaning of a sentence is related to the syntactic structure of the sentence and the meaning of functional terms (the counterpart of logical connectives in a formal language), which include morphological markers, quantifiers, determiners, logical connectives, modals, auxiliary words, pronouns and relative pronouns, some prepositions, and ellipsis markers.

As a simple illustration, the inference in (6), which is based only on structural semantics, does not require knowing the meaning of the words *John*, *pencheon*, and *sopomous*. It only requires knowing that they are a proper name, a noun, and an adjective, respectively, as well as the meaning of the functional words *every*, *is*, and *a*. Additionally, one needs to have an accurate knowledge of morphology and syntax as prerequisites for the structural semantics stage.

(6) Every pencheon is sopomous.
John is a pencheon.
⇒ John is sopomous.

In contrast, the inference in (7) is not *merely* a structural semantic inference because it depends on the meaning of the words *man* and *human* and on lexical/world knowledge about the connection between those meanings. However, if this knowledge is given to us in explicit form, as in (8) for instance, then the pattern in (7) can still provide a useful test for structural semantic knowledge.[11]

[10] The terms *formal semantics* or *compositional semantics* are commonly used instead. I prefer the term *structural semantics* as it distinguishes this branch of semantics from lexical semantics, which may also be formalized, and which also addresses issues of composition, e.g. noun-noun compounds or the alteration in meaning that verbs and nouns undergo when they are combined in a metonymy.

[11] An NL sentence expressing background knowledge may itself introduce ambiguities and require more world knowledge in order to understand it. Since the goal here is to test only the understanding of (7), the background knowledge in (8) is formalized in an unambiguous logical language rather than given as an NL sentence.

(7) Every human likes watermelons.
⇒ Every man likes watermelons.

(8) $\forall x.[man(x) \rightarrow human(x)]$

The inference in (9)a and the difference between (9)a and (9)b depend not only on the meaning of the words but also on linguistic knowledge about argument realization – how thematic roles are connected to syntactic roles (see e.g. (Levin and Hovav, 2005)). This knowledge is intimately connected to issues of knowledge representation and the ontology, and is not considered part of what I mean here by structural semantics.

(9) a. John loaded the wagon with hay.
⇒ The wagon became full [at that time].

b. John loaded hay on the wagon.
$\not\Rightarrow$ The wagon became full [at that time].

Finally, structural semantics does not encompass world knowledge of the kind necessary for drawing the inference in (10).

(10) John ate a steak in a restaurant.
⇒ John probably talked to a waiter before he ate the steak.

1.2.2 Why Is Structural Semantics Important for NLP?

Structural Semantics knowledge is essential for exact NLU applications. To get at the exact truth conditions that are expressed in exact NLU texts, it is essential to understand the meaning of functional words, and how they specify the logical combinations of the other pieces of the sentence. For example, understanding (3) requires knowledge of quantifiers and numeric expressions in their various manifestations (including e.g. "X or higher"), generic as opposed to existential quantification, modals ("must", "may"), and more. Queries and commands as in (4) require understanding comparative and superlative constructions, as well as being aware of scope ambiguities of semantic operators (is the $70,000 salary in each year or total over the two years?).

Another reason why it is essential to have an accurate representation of the information in each sentence is that this information needs to be combined correctly with the information in other sentences, as well as with background information about the domain and

the world. We need exact inference procedures to combine these pieces of information and correctly infer consequences from them, as the answers to questions almost never appear explicitly in the text.

Structural semantic knowledge is largely domain-independent, and so it possesses a level of generality higher than other kinds of knowledge. Thus, once it is developed, it could help improve the accuracy of many NLP applications. It could be used for different purposes with little customization (the main customization might be adding frequencies of various phenomena, which may differ across domains). Here are some examples of NLP applications that could benefit from structural semantics:

Question Answering: The matching between queries and texts that existing question-answering systems (such as (Pasca and Harabagiu, 2001)) calculate is based mainly on word alterations and matching syntactic structures. The quality of this matching would be improved if it also relied on knowledge of structural semantics. This knowledge would be used to help capture and represent more precisely the meaning and information that are actually conveyed by the texts and to perform higher-level reasoning. While arguably such knowledge may not be needed for answering simple who-did-what-to-whom factoid questions whose answers appear explicitly in the text, it is necessary for being able to answer questions that rely on *combining* the information that is conveyed by several sentences. For example:

(11) T: [Some NL texts talking about U.S. presidents, e.g.:]
George Walker Bush (born July 6, 1946) is the 43rd President of the United States, inaugurated on January 20, 2001. He was re-elected in 2004 and is currently serving his second term.[12]
Q: Was the third president of the United States who got re-elected a Democrat or a Republican?

It is unlikely that the answer appears explicitly in some text as "The third president of the United States who got re-elected was a Democrat" or some variation of it. Rather, the computer must understand what *third* means and how it combines with the meaning of the relative clause to constrain the choice of president, as well as understand the meaning

[12]From http://en.wikipedia.org/wiki/George_W._Bush.

of the connective *or*. It also has to calculate the answer by combining various pieces of information conveyed throughout the given texts.

In the RTE challenge (Dagan et al., 2005),[13] the computer needs to decide whether a given sentence called "hypothesis" can be inferred from a given short text. Knowledge of structural semantics would be useful for many RTE-like questions. For example, in (12), the computer needs to know what *more than* means (the answer would be *Does not follow* if *80 kilometers* were replaced with *120 kilometers*). In (13), the computer needs to know how to instantiate a statement quantified by *each* (here, with the year 2005), as well as to know about implications between numeric quantifiers. Answering (14) correctly requires understanding conditionals and modality.[14]

(12) T: In any case, the fact that this week Michael Melvill, a 63-year-old civilian pilot, guided a tiny rocket-ship more than 100 kilometers above the Earth and then glided it safely back to Earth, is a cause for celebration.
H: A tiny rocket-ship was guided more than 80 kilometers above the Earth. [Follows]

(13) T: Each year, 26,000 people are killed or mutilated by landmines of which 8,000 are children.
H: In 2005, at least two thousand children were injured by landmines. [Follows]

(14) T: Things would be different if Microsoft was located in Georgia.
H: Microsoft's corporate headquarters are not located in Georgia. [Follows]

NLP systems that go beyond syntactic and lexical knowledge to actually knowing what functional words mean and how they tie together different pieces of information would enjoy a competitive edge over similar systems that do not use such knowledge.

Knowledge Acquisition from NL Texts: The more structural language knowledge a computer has, semantics included, the better able it is to automatically acquire factual knowledge correctly from crawling texts on the internet. Possessing only knowledge of syntax allows only rudimentary acquisition of simple patterns of facts that appear explicitly in the text; but having more sophisticated semantic representations and inference could allow the computer to combine separate pieces of information that appear throughout the texts. This is precisely the utility of semantic representations, that they capture the

[13] See also http://www.pascal-network.org/Challenges/RTE/Introduction/.
[14] These examples are taken or adapted from the first RTE test-suite.

content of a text, and make it possible to relate the pieces of information to each other, merge them, compute entailments between them, etc. These are not possible if one relies only on the *form*, i.e. syntax, of the texts.

Finally, of all the kinds of semantic knowledge needed in a sophisticated NL understanding system (including lexical knowledge and world-knowledge in ontology and facts), the body of structural semantic knowledge has the smallest size. A rough estimate is that English has about 400 functional words (and about 400 grammar rules), in contrast to about 45,000 verb frames (verbs with syntactic and thematic roles), and at least 100,000 nouns. Therefore, we have a good chance to capture all or almost all of the structural semantic knowledge through a concentrated collaborative effort in a reasonable amount of time (of course that project extends much beyond the scope of one dissertation).

1.2.3 Limitations

For almost all NLU applications, structural semantics knowledge is not enough by itself as the semantic component. It obviously does not deal with the meaning of non-functional words at all, which is the realm of lexical semantics. This was already discussed for (7) and (9). Moreover, in general NL texts, very little can be inferred by using only structural meaning representations that capture the information explicitly stated in the text. This is because NL texts are written for humans, and so they assume a huge amount of world knowledge that computers lack.

Even in simple domains such as logic puzzles, where most of the information required for solving the task is in fact captured by the direct literal meaning representations, this lack of knowledge sometimes happens. For example, in Question 2 of Figure 1.1, the term "complete list" appears. The structural semantics level cannot say anything about the meaning of this term except that it refers to some object which is a "list" and is "complete". But it is necessary to further flesh out the truth conditions of this term, namely that it refers to an object g such that an element has the property "sculpture exhibited in room 2" iff that element is a member of g. Also, the puzzle does not state explicitly that no sculpture may be exhibited in more than one room. Without this piece of knowledge, the explicit conditions in the text are insufficient to yield exactly one answer for each multiple choice question (for question 1, answers B, D and E would all be correct).

Human readers know this implicit piece of world knowledge, but it has to be given to a computer in explicit form.

It is possible to solve these problems, at least in principle, by supplying the computer with additional information that could be combined with the information in the literal meaning representations. But there are some fundamental limitations to the structural semantic meaning representations themselves. The first limitation is context dependence on pragmatic factors. Natural language is very flexible in terms of zoning in on one situation that is a subset of the larger ongoing discourse, and doing so implicitly. For example, it is highly unlikely that the sentence "Most students liked the class" claims something about the majority of all students in the world, and it is very likely it claims something about most members of the set of students in a particular group, probably the class itself that is mentioned in the sentence. However, figuring this out is not part of the "pure" truth conditions of *most*, but is a contextual/pragmatic issue.

The most fundamental difficulty is that meaning representations based on structural semantics capture the direct literal meaning of the text and ignores all issues of "language use". The way that humans use NL goes beyond the literal meaning, and there are various ways in which this happens, including:

1. Hyperbole: exaggeration. For example, the sentence "All the students shouted at each other" might be used even if only almost all of them, or just many of them, did.

2. Irony: the use of words to convey a meaning that is the opposite of its literal meaning.

3. Metonymy: using one object to stand for another. For example, (15)a is a shorthand way of saying the precise statement (15)b (this example is taken from (Clark et al., 2005)).

   (15) a. The acid on the left is stronger than the acid on the right.

   b. The acid denoted by the formula on the left side of the equation of the reaction is stronger than the acid denoted by the formula on the right side of the equation of the reaction.

4. Metaphor: interaction of seemingly unrelated concepts. For example, the *literal* meaning of the sentence "Our relationship is not going anywhere" does not make sense because a relationship is an abstract concept and so it cannot move to a different physical location. The intended meaning of the sentence is that in terms of the

conceptual domain of relationships, the state of our relationship is that it is not developing (like it should). The sentence expresses this thought by borrowing a term from the conceptual domain of (physical) movement, namely not going anywhere. (Lakoff and Johnson, 1980).

5. Speech acts: A NL statement could be used to perform an action, or express the speaker's intention, in a way that goes beyond the literal meaning of the statement. Depending on the context, a person uttering "I'm cold" may be implying he wants the situation to change and therefore his utterance conveys a request to another person to close the window. (Austin, 1962; Searle, 1969).

6. Implicatures: The author or speaker may intend to implicate an additional meaning beyond the literal meaning. For example, "I may or may not go there" is a tautology as far as truth conditions goes. But since it is usually useless to state tautologies (since they do not add information), the hearer can infer that the intended message here is "I do not know whether I will go there". Figuring out what the intended meaning is requires identifying adherence to or violation of conversational maxims (Grice, 1975).

The problem that these ways of using language pose for structural semantics is that the direct model-theoretic denotation of the terms in the sentence is different from the intended meaning. Nevertheless, getting the literal meaning right is an important first step towards understanding the intended meaning. Once the computer has the literal meaning, it might be able to reason *about* the meaning representation formula itself in order to flesh it out or change it, rather than simply reasoning *from* it. While this is sometimes necessary in exact NLU applications, it tends to be less of an issue there compared to other NLP applications.

Despite its limitations, structural semantics is a necessary component in any system that does interesting inference based on the *meaning* of NL input and the *information* conveyed by texts. Without this knowledge, one is limited to just the meaning of words and only very rudimentary combinations of them.

1.3 Dissertation Plan

1.3.1 General Picture

Generally speaking, there are four main parts to an exact NLU application, although they do not necessarily correspond to four distinct components in the system:

1. Task-independent linguistic analysis of the NL input.
2. Task-dependent resolution and adaptation of that analysis to the application.
3. Back-end task solver (e.g. puzzle solver).
4. Ambiguity management.

The reason for aiming at a task-independent linguistic analysis of the NL input is that we want the algorithms and knowledge that produce such an analysis to be portable to many other applications. If, instead, assumptions about the particular application crept into those components, it would be difficult if not impossible to port them to other applications.

Ambiguity management is not an independent component or stage in the system, but rather a theme that permeates all stages of processing. Disambiguation requires, in the general case, information from all stages, and the system may need to iterate through the stages several times until it can zone in on the correct interpretation of the text.

In this dissertation, I will mainly focus on items 1 and 4 above, and in particular, the structural semantic analysis and its interaction with ambiguity management. Before we proceed to Part II, **chapter 2** will provide a basic picture of how the computer can translate NL texts to exact meaning representations and reason with them. Part II of this dissertation will focus on questions 3 and 4 from section 1.1.4, i.e. how to specify the syntax-semantics interface, and how to handle ambiguities efficiently at the level of semantic representations and their calculation. Part III of the dissertation elaborates on question 3, as well as questions 1 and 2 from section 1.1.4, by providing a computationally-grounded linguistic analysis of many complex linguistic constructions that appear in exact NLU texts such as logic puzzles. Part IV includes **chapter 15**, where I briefly touch on items 2 and 3 above in the context of an application for solving logic puzzles, and **chapter 16**, where I summarize the main results of the dissertation and point to future work.

There has been some progress in recent decades in the research on computing semantic representations from syntactic structures. The kind of work that is widely considered to be the state-of-the-art in the area of computational structural semantics today is along the lines of (Bos, 2004; Blackburn and Bos, 2005a,b; Fuchs et al., 2006; Egg et al., 2001; Copestake et al., 2005). Much work remains to be done on making the formalisms convenient and practical for human knowledge writers, making the algorithms for dealing with ambiguities efficient, and extending the coverage of analysis to more advanced linguistic constructions. I will address these three issues in turn.

1.3.2 The Syntax-Semantics Interface

The traditional approach to specifying the (structural) syntax-semantics interface, starting with Montague (1974) (see also e.g. (de Swart, 1998)) adheres to a very simple scheme: The meaning of a syntactic constituent is calculated by using only lambda-application to combine the meanings of the constituent's immediate sub-parts. This idea works for simple sentences but runs into difficulties when quantifiers and other more complex linguistic constructions are involved. Consequently, the rules mapping syntax to semantics become complicated, e.g. using type-raising operators, as we shall see in section 3.1.3.

In contrast, Glue Semantics (Dalrymple, 2001) provides a more powerful and flexible scheme of combination. While allowed combinations are restricted by the syntactic structures, the order of their computation need not necessarily adhere to the hierarchical structure of the syntax. This allows for a simpler specification of the syntax-semantics connection. The price is that the scheme of combination is more complex – it does not rely on a simple lambda-application principle but on a more powerful logical proof system (implicational linear logic). However, the price is worth paying because it needs to be paid only once, and the framework makes it easier for the linguist or grammar engineer to specify the syntax-semantics connection. Therefore, I will use this framework in this dissertation. **Chapter 3** will provide a new introduction to this topic, which, unlike previous published work, does not rely on any knowledge of Lexical Functional Grammar (LFG) or linear logic.

A computational implementation of glue semantics has been created by the NLTT group at the Palo Alto Research Center (PARC) during the 1990s.[15] This was made into

[15]Participants included: Ash Asudeh, Dick Crouch, Mary Dalrymple, John Fry, Vineet Gupta, Angie Hinrichs, John Lamping, Jonathan Reichenthal, Vijay Saraswat, and Martin van der Berg.

a component of the XLE. (The XLE is a large computational infrastructure for analyzing and generating NL texts, which has been developed by the NLTT group since the early 1990's.[16]) The glue semantics component takes as input syntactic analyses in LFG (Bresnan, 2001; Dalrymple, 2001), and by following glue semantics specifications it creates semantic representations. The implementation is based on the algorithm of Hepple (1996) with some optimizations. Some additional experimental implementations were based on Proof Nets and on the ideas in (Gupta and Lamping, 1998).

In **chapter 4**, I will explain how to use XLE's rewriting system to transform XLE's F-structure output to glue semantic premises. In **chapter 5**, I will discuss Hepple's algorithm and provide a proof for its correctness (which has not been shown before). I will also discuss some issues of optimization.

1.3.3 Ambiguity Management

Packing and Free Choice Space

NL sentences may be ambiguous if they are taken out of context, and sometimes even when the context is taken into account. For example, the sentence "Flying planes can be dangerous" has a syntactic ambiguity, and in general, we cannot disambiguate it just by looking at the sentence itself. We may need to take the larger context and information into account, as the following example demonstrates:

(16) a. Flying planes can be dangerous. Therefore, only licensed pilots are allowed to do it.
b. Flying planes can be dangerous. Therefore, some people are afraid to ride in them.

The simplest strategy to deal with this issue is to enumerate all the possible interpretations of the text, translate each one to FOL, and see which interpretation leads to a correct result (in the puzzle application, which interpretation leads to exactly one correct answer to each multiple-choice question). The problem with this strategy is that it may lead to an exponential explosion in the number of possible interpretations, because ambiguities multiply within and across levels of analysis in the system (morphology, syntax, semantics, etc.).

Another strategy is to choose at each level of analysis the locally most likely analysis based on overall statistics on NL. For example, there have been attempts to resolve scope

[16] http://www2.parc.com/isl/groups/nltt/xle/

ambiguities based on general statistics using features such as the quantifier name, its position in the sentence, active/passive voice, etc. (Higgins and Sadock, 2003; Andrew and MacCartney, 2004). This can provide a useful heuristic for *ranking* possible scopings. However, such general statistics may lead to wrong results. Higgins and Sadock's system achieves an accuracy of about 77% on WSJ sentences. Even when considering sentences from GRE/LSAT logic puzzle texts with two interacting quantifiers where the two scopings are not logically equivalent, Andrew and MacCartney's system achieves an accuracy of only 94%. In a precise understanding application, the ultimate arbitrator on ambiguity resolution is the meaning of the entire text in the context of the application rather than local decisions. A careful consideration of the logical content of sentences is needed in order to guarantee the correctness of the reasoning.

Therefore, in this dissertation I will follow the strategy for ambiguity management that is employed in the XLE. The XLE's general philosophy of handling NL ambiguities says that no component of the system has complete information locally to choose the correct analysis and completely disambiguate the input. Instead, all possible analyses are represented and calculated efficiently in a *packed* form. To demonstrate this idea schematically, consider the ambiguous sentence:

(17) The sheep ate the cabbage.

The first noun can be singular or plural, and the second can be count or mass noun. Thus we get four possibilities:

(18) The sheep-sg ate the cabbage-ct.
The sheep-sg ate the cabbage-ms.
The sheep-pl ate the cabbage-ct.
The sheep-pl ate the cabbage-ms.

Analyses can multiply exponentially in this way. Instead, we can factor our common parts and get a packed representation:

(19) The sheep-$\left\{ \begin{array}{c} \text{sg} \\ \text{pl} \end{array} \right\}$ ate the cabbage-$\left\{ \begin{array}{c} \text{ct} \\ \text{ms} \end{array} \right\}$.

The important thing about packing is that all analyses are encoded without loss of information, and common parts are both represented and computed just once.

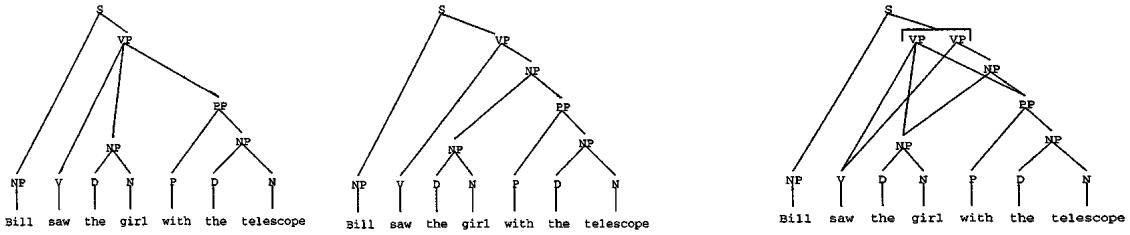


Figure 1.3: Explicit and packed syntactic structures

The management of alternatives is done with the help of a *free choice space* (Maxwell and Kaplan, 1991). This will be explained in more detail in section 4.2.1. Briefly, choice space packing is a generalization of the idea underlying a classic chart parser for a context-free grammar (CFG), which runs in $O(n^3)$ time and compactly calculates and represents in a chart-forest an exponential number of analyses by factoring out shared parts. Also, each analysis can be retrieved in linear time from the chart-forest (see e.g. (Allen, 1995)). The right side of Figure 1.3 shows an example of a chart forest, i.e. a packed form of the two syntactic structures on the left. In the XLE, this idea is generalized to other levels of linguistic analysis: Morphology (Kaplan et al., 2004b), C(onstituent)-structure and F(unctional)-structure (in syntax) (Maxwell and Kaplan, 1993, 1996), and knowledge representation (Crouch, 2005). Thus, ambiguities are not necessarily resolved locally but are pushed through the stages until either a later stage has sufficient knowledge to resolve them, or packed reasoning is performed.[17] The XLE can also use statistical methods to filter out some of the locally least-likely analyses in each component (Kaplan et al., 2004a).

Packed Glue Computation

Although Hepple's algorithm for calculating glue derivations that was mentioned above was implemented at PARC, an open question remained: how should this algorithm be integrated with the choice-space framework? Although some attempts were made to answer this question, until now no satisfactory solution was found. Currently, therefore, the packed syntactic analysis first needs to be unpacked, and then for each possible syntactic analysis, the algorithm runs a separate glue semantics calculation which yields a separate

[17] Some applications may be able to avoid disambiguation altogether. In machine translation, for instance, it may be possible to find a translation in the target language that preserves the ambiguities in the sentence being translated.

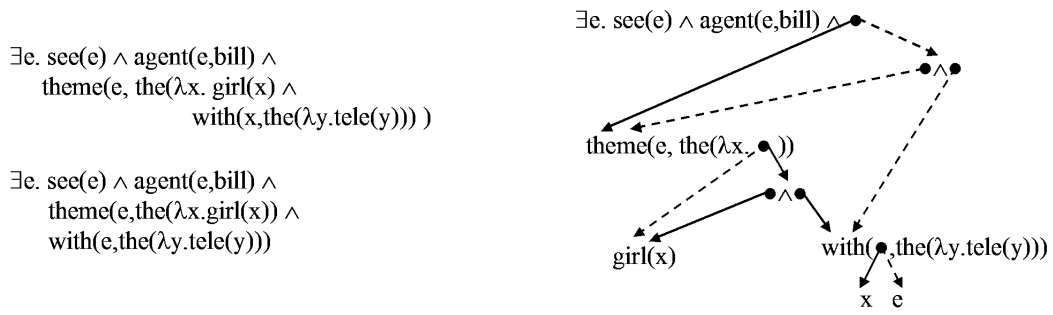


Figure 1.4: Explicit and packed semantic representations

semantic representation. Representations can multiply also as a result of scope ambiguities of quantifiers and other operators. This is inefficient and the number of results could explode.

A central achievement of this dissertation is showing how to solve this problem. I present a component that integrates the computation of glue semantics with the choice-space framework. The component's output is a packed semantic representation. For example, the two possible semantic representations for "Bill saw the girl with the telescope" are shown on the left of Figure 1.4. Since they share a lot of material in common, we would like to obtain the packed representation on the right instead. The solid arrows correspond to the interpretation where the girl is with the telescope, while the dashed arrows correspond to the other interpretation. Parts that are common to the two semantic structures appear just once, and most of the differences are just in the pointers, similarly to the packing in Figure 1.3.

Packing should also be obtained when there are scope ambiguities. For example, a sentence such as "Every boy didn't see most films" theoretically has six readings that result from all possible permutations of the three operators (the negation and two quantifiers). All six readings share these operators and the main verb, and the only differences between them are in the pointers that connect the different parts. This should be reflected in the packed form.

These packed structures bear some similarity to underspecified semantic representations (Bos, 1996; Egg et al., 2001; Copestake et al., 2005). While these latter methods share the goal of compactly representing alternative semantic representations, they were developed in quite ad-hoc ways by complicating the syntax-semantics specification with extra overhead to manage ambiguities and pieces of underspecified structures. Instead, the goal here

is to have the packed semantic representations fall out naturally from the interaction between glue semantics and the choice-space framework. Just as with the packed rewriting system in the XLE (see (Crouch, 2005) and the review in section 4.2.3 here), the grammar writer should be allowed to think (almost) only about unambiguous inputs, leaving it to the choice-space mechanism to *automatically* take care of managing the ambiguities and keeping representations in a packed form as in Figure 1.4.[18] Section 7.7 will compare the underspecification and the packing approaches in more detail.

The computation of glue premises from F-structures in **chapter 4** will automatically take care of handling a morphosyntactically ambiguous but packed input. Chapters 6 and 7 will present a novel algorithm that combines glue semantics computation with an efficient handling of ambiguities using the choice-space framework. In **chapter 6**, I will discuss how to deal efficiently with internal ambiguities, i.e. those arising from different possible orders of modifiers. In that chapter, I will assume that the input is a simple and unambiguous set of glue premises, although it may lead to more than one semantic result. In **chapter 7**, I will explain how the glue computation can be extended to deal efficiently with external ambiguities, i.e. those arising from a packed set of input glue premises.

More Packed Linguistic Analysis

Ideally, we should also calculate anaphoric connections, ellipsis reconstructions, and plurality possibilities in a packed way. However, it is not a trivial matter to take an algorithm that works on unpacked structures and then adapt it to packed structures. We will see in chapters 6-7 that creating a packed version of the algorithm that computes semantic representations given syntactic structures requires some ingenuity and trial-and-error. I will only hint at how packed anaphora resolution could be done in section 8.5.3, and I will use an underspecified form for plurality ambiguities in chapter 12.

Packed Reasoning

Ideally, the goal of compactly representing and computing all possible analyses should be carried forward to the next stage: automated reasoning. This has been partially done at PARC in the system they built for the RTE challenge mentioned in section 1.2.2. The meaning of a text is represented as a set of skolemized facts (derived from the semantic

[18] A related proto-attempt to achieve this goal is reported in (Lev, 2005a), but it uses quasi-logical forms (Alshawi, 1992) and hole semantics (Bos, 1996) rather than glue semantics.

representations), relativized to various contexts in the choice-space if there is an ambiguity in the input. Then, a hypothesis is said to be entailed from a text if its facts match a subset of the facts in the text (unless negation is involved). The gain here is that if a conclusion depends only on facts that are in the top context of the choice-space, then no matter how many ambiguities there are (i.e. how many subcontexts the choice-space has), the calculation needs to be done only once. See (Crouch, 2005; Bobrow et al., 2005; Nairn et al., 2006) for more details.

This is a specialized kind of reasoning, but it is not hard to imagine how a general theorem prover could be extended to work with packed FOL input. Each formula will reside in one part of the choice space. Whenever the theorem prover combines two facts to produce a new fact, the new fact resides in the context obtained from the intersection of the contexts of its parents. Also, whenever a fact resides in more than one context, the choice-space management should automatically move it to the context which is the disjunction of the individual contexts. The merit of doing all this depends on whether independencies between choices hold by-and-large also in the automated reasoning stage. In the worst case, many dependencies would lead to an exponential blowup of the choice-space.

1.3.4 Linguistic Coverage

As the examples in section 1.1.2 show, the computer needs to know about functional semantic constructions, such as quantifiers, numbers and plurals, modifications, connectives, comparative and superlative constructions, conditionals, modals, anaphora, and more. It is therefore important to work on extending the coverage of the syntax-semantics specification. Without further progress on these constructions, we would not be able to feed the back-end inference component of a NLU application with appropriate semantic representations.

Part III of this dissertation is devoted to extending the coverage of the glue semantics specification to additional linguistic constructions that are necessary for understanding logic puzzle texts and texts for other exact NLU applications. I will discuss the interaction of anaphora with glue semantics (**chapter 8**), and then cover ellipsis (**chapter 9**), numerical comparatives (**chapter 10**), gradable comparatives (**chapter 11**), plurals (**chapter 12**), overt reciprocals (**chapter 13**), and covert reciprocals including in uses of *same* and *different* (**chapter 14**). The topics are covered in this order because each chapter relies

on certain issues discussed in preceding chapters. The introduction to Part III will provide more information about the choice of these topics and their main issues.

Machine Learning?

This knowledge acquisition process here requires not only intense manual labor but also a high degree of knowledge and expertise in structural semantics. A natural question to ask is: Could we acquire this knowledge automatically using machine learning (ML) techniques?

First, in exact NLU applications that cannot tolerate errors, we need completely accurate knowledge. It is doubtful that ML techniques could match the level of quality that a human knowledge engineer could provide.

What about other NLP applications that can tolerate lower levels of accuracy? Even here, the answer is we cannot do it at the present moment. The main reason is that automatic acquisition of knowledge presupposes we know what kinds of structures the computer should acquire, i.e. how to define the search space of representations, as well as how to define what good solutions are. However, we (humans) are not yet sure ourselves how complex meanings should be represented. Therefore, there is still much linguistic and computational linguistic research that needs to be done before we can know how to do automatic acquisition of complex semantic knowledge. Even in syntax, unsupervised learning methods have thus far not been as successful as handwritten grammars or supervised learning from treebanks.

One could ask whether it is possible to do in structural semantics what people have done since the 1990s in syntax, starting with the Penn Treebank. Is it possible to create a "Semantic Treebank," i.e. a corpus in which sentences are annotated not only with syntactic structures but also with semantic representations of their meanings? Such a treebank could help support progress in statistical semantics that would parallel the progress that the Penn Treebank allowed in statistical syntax.

The creation of the syntactic Penn Treebank required a 300-page annotation manual that essentially codified a syntactic grammar. Creating this manual was possible thanks to a more-or-less broad consensus that existed in the field regarding what syntactic representations should be used (at least for some core phenomena in English). This consensus was reached only after several decades of research on syntax in theoretical linguistics and on rule-based parsers in NLP.

A similar kind of consensus in computational structural semantics is a necessary prerequisite for a Semantic Treebank. However, the levels of consensus, understanding, and coverage in structural semantics in linguistics and computational linguistics today are less than they were for syntax in the 1990s. Many issues still remain unresolved, and more progress on them is needed before good annotation schemes can be developed. In fact, even in syntax, there is still much room for improvement in the Penn Treebank, for example in the internal structure of noun phrases, and the treatment of ellipsis.

Above all, there is a fundamental difficulty in semantics that is absent from syntax. Semantic structures are much more complex than syntactic parse trees. More crucially, while there should be just one correct syntactic structure for the intended reading of a sentence, in semantics the intended truth conditions can be expressed by many different logically-equivalent formulas. Although they are equivalent as far as their meaning, the *shapes* of these formulas can be radically different from each other, and only few of them are amenable to a compositional computation from the syntax. However, a human annotator may not come up with one of those few versions.

Here is a simple illustration. The formula in (20)b correctly captures the truth conditions of (20)a. However, the shape of (20)b really ignores the *each other* phrase in the sentence (20)a, and so (20)b cannot be broken down into pieces that come from the different parts of the sentence (20)a in a way that is *generalizable to and consistent with* other sentences, such as (20)c-d. The formula in (20)e is logically equivalent to (20)b (assuming exactly one book was read), but its structure is consistent with the structure of (20)d, in contrast to (20)b. (I will discuss sentences such as (20)a,c in detail in chapters 13 and 14.)

(20) a. John and Mary read the same book as each other.

b. $\exists x.book(x) \wedge read(john, x) \wedge read(mary, x)$

c. John, Mary, and Bill like each other.

d. $\textsc{recip}(\{john, mary, bill\}, \lambda x \lambda y.like(x, y))$

e. $\textsc{recip}(\{john, mary\}, \lambda x \lambda y.read(x, the(\lambda z.book(z) \wedge read(y, z))))$

The obstacle that this issue poses is the following. If the sentence (20)c is annotated by (20)d and (20)a by (20)e, one could have a reasonable hope that a computer could learn automatically how to break these formulas into constituents that come from the different parts of the sentence in a way that is consistent for both sentences and is generalizable to unseen sentences. However, given a sentence such as (20)a, a human annotator is likely to

write the truth conditions using (20)b rather than (20)e. This would make it extremely hard for the computer to induce correctly a consistent syntax-semantics interface that could generalize correctly to other cases.

This example demonstrates that the knowledge acquisition process here requires a linguistically sophisticated and comprehensive manual data analysis of many cases. Although the manual effort is intensive, there is abundant work in NLP on non-automatic acquisition of knowledge. Such efforts include: dictionaries, WordNet, VerbNet, FrameNet, PropBank, the Penn treebank, and other annotated corpora. In linguistically-based NLP, we can point to the Lingo project[19] and the XLE that were mentioned above. Furthermore, once the effort gets underway, it becomes easier. I am hopeful that my efforts in this dissertation, and especially in chapter 4 and Part III of this dissertation, will make it easier to extend the linguistic coverage of the syntax-semantics interface.

[19] http://lingo.stanford.edu/.

Chapter 2

A Basic Framework for Semantics and Inference

In the rest of this dissertation, I will be using meaning representations that are supposed to support exact inference. This chapter provides some relevant background about these concepts.

2.1 Abstract Definition of the Task

All the exact NLU applications mentioned in the previous chapter can be seen as instances of the following abstract task definition. The computer is given as input a NL text T and a NL sentence S. The computer needs to produce an output A, which states whether or not S is a conclusion from T *in the context of a specific application* (solving logic/physics/math puzzles, understanding regulatory texts, etc.) Instead of S, we may give the computer a NL question Q and ask it to produce as output a NL answer A. We would then check whether A is a correct answer to the question Q in light of the information given in T, again in the context of a specific task. We call the pair (T, S) or (T, Q) a *query*. (In the case of a NL interface to a database, T is not an NL text but the database).

If Q is a yes/no question, the answer may be *Yes* or *No*, and if Q is a wh-question, the answer may be a value. Other possible answers are:

- *Unknown*: the computer *knows* that the answer cannot be determined from the text. For example, when T is "John likes Mary" and Q is "Does Mary like John?"

- *AssumptionFailure*: the computer *knows* that the question relies on a false assumption. For example, a person asking "Does John realize that Mary likes him?" presupposes that Mary likes John. If this question is given to the computer together with a text that does not support this presupposition, such as the text "John likes Mary", the proper answer is "AssumptionFailure: the fact 'Mary likes John' is not supported by the text". An answer *No* might be misleading as it might lead the user to conclude that the sentence "John does not realize that Mary likes him" is supported by the text, which is not the case.

- *Ambiguous*: the computer *knows* that there is more than one correct answer to the query due to some ambiguity in the text or the question. For example, if the text includes "John saw her duck" and the question is "Did John see a duck?", the answer may be *Yes* or *Unknown*, depending on the syntactic analysis of the text (i.e. whether *her* is analyzed as an accusative pronoun or a possessive pronoun). If the intended analysis is not clear from the text in the context of the task, the answer should be *Ambiguous*. In addition, it would be useful if the computer pointed out the ambiguities and even gave all possible answers.

- *DontUnderstand*: the computer *knows* that understanding the text or the question is beyond the scope of its abilities. An example is when T is "John considers Mary a nice woman" and the computer's knowledge of language does not include knowledge of how to understand reduced clauses. In addition, it would be useful if the computer specified what parts of the input it did and did not understand.

- *Fail*: the computer could not reach any of the definite answers mentioned above because of lack of computational resources or some other problem.

An important constraint we impose in the task definition of an exact NLU application is that the computer must produce an answer with certainty and not by guessing. As the list of possible answers above shows, we want the computer to *know* when it cannot answer a query as a result of it not having the necessary knowledge to do so. As section 1.1.3 explained, the motivation for this restriction is reliability, which is needed in exact NLU applications.

In application domains other than exact NLU, one could refine this inventory of answers even further to add degrees of plausibility and confidence.

This classification of answers is much richer than the binary classification of *follows* vs. *does not follow* in the RTE task (Dagan et al., 2005).

2.2 The Logical Perspective

An exact NLU task relies on a richer notion of *consequence* compared to standard logical consequence. However, since the latter is simpler and most work in semantics is built on it, let us review it first. Classical logic and its computation include the following components:

1. **Formal language:** A definition of the lexicon and syntax of the formal language. The lexicon consists of terms: variables, constants, and terms composed of a n-ary function symbol applied on n terms. The syntax says that an atomic formula is composed of a n-ary predicate symbol applied on n terms, and that if φ and ψ are formulas then so are $\varphi \rightarrow \psi$, $\varphi \wedge \psi$, $\forall x.\varphi$ (where x is a variable), and so on.

2. **Model-theoretic semantics:** (MTS) – defined compositionally on the formal language. Assigns to each expression in the language a certain mathematical object through the mapping relation $[\![\cdot]\!]^g_M$, where M is a model and g an assignment function. Thus, for a variable x, $[\![x]\!]^g_M = g(x)$, for an atomic formula $[\![P(t_1, \ldots, t_n)]\!]^g_M = true$ iff $\langle [\![t_1]\!]^g_M, \ldots, [\![t_1]\!]^g_M \rangle \in I_M(P)$, for a conjunction $[\![\varphi \wedge \psi]\!]^g_M = true$ iff $[\![\varphi]\!]^g_M = true$ and $[\![\psi]\!]^g_M = true$, and so forth.

   The MTS induces a **(semantic) consequence relation:** The relation $\models$ is defined between formulas based on their MTS: $\varphi \models \psi$ iff: for all M and g, if $[\![\varphi]\!]^g_M = true$ then $[\![\psi]\!]^g_M = true$. For a set of formulas Γ, $\Gamma \models \psi$ iff: for all M and g, if $[\![\varphi]\!]^g_M = true$ for all $\varphi \in \Gamma$ then $[\![\psi]\!]^g_M = true$.

3. **Proof system:** A system C of axioms and rules that allows to demonstrate finite proofs of entailment. If there is a proof of ψ from Γ we write $\Gamma \vdash^C \psi$. The system should be sound w.r.t. the semantics, i.e. if $\Gamma \vdash^C \psi$ then $\Gamma \models \psi$, and ideally, it should also be complete, i.e. if $\Gamma \models \psi$ then there is a proof of that, i.e. $\Gamma \vdash^C \psi$ (for FOL there are sound and complete proof systems, but completeness may not hold for some logics). There may be more than one sound and complete system C for a given semantics.

4. **Automated Prover:** A computer program that gets pairs of Γ and ψ and attempts to find whether or not $\Gamma \vdash^C \psi$. Ideally, at least if $\Gamma \vdash^C \psi$ then the computer will eventually terminate and show the proof. If $\Gamma \nvdash^C \psi$ then the computer may not terminate because FOL is only semi-decidable. However, in some cases the computer may terminate and demonstrate a countermodel, i.e. a (finite) model of $\Gamma \cup \{\neg\psi\}$. For certain classes of Γ, ψ, termination can be guaranteed.

In other logics, things may be more complicated. For example, in modal logic, a model M is a Kripke structure, and $[\![\cdot]\!]$ is relative not just to M and g but also to a world w.

The rest of this chapter reviews a basic (and simplistic) logical-computational framework for doing logical inference from sentences in a small fragment of English. In order to clarify the basic issues involved, I will start with a few simplifying assumptions: Each NL sentence has just one reading, and we handle only very simple linguistic constructions and issues. Later sections will relax these assumptions.

2.3 A Basic Fragment: L_0^{NL}

For now, we have a very small fragment of (simplified) English, L_0^{NL}, to be defined below. We are given a set T of sentences in that fragment and a query sentence Q, and we need to determine whether Q follows from T, T refutes Q, or neither is the case. If a text T entails a query Q, we write this $T \models_0^{NL} Q$ (the fragment is small enough that there is consensus about judgments of entailments). Our goal is to devise a system that decides this relation (the fragment is small enough that entailment is decidable).

As an illustration for such a simple fragment, consider the following grammar and lexicon (ignoring agreement etc.):

(21) S → NP (Neg) VP
NP → Det N'
N' → N
N' → Adj N'
N' → N RelC
Relc → who VP
NP → PN

VP → IV

VP → TV NP

VP → is a N

VP → is Adj IV → walk, arrive, sleep, sneeze, ...

TV → see, love, hate, ...

PN → John, Mary, Bill, ...

Det → every, some, no

N → boy, girl, man, woman, ...

Adj → French, red, married, ...

Neg → doesn't

This grammar is unambiguous, i.e. every sentence it accepts has exactly one syntactic structure. Since there is no ambiguity, there is a function σ that maps each sentence in L_0^{NL} to its syntactic structure. Although in reality there may be scope ambiguities of quantifiers as well as negation in the NL fragment, for now let us stipulate that always a canonical scoping is intended, i.e. the one determined by the linear order of quantifiers in the sentence.

2.4 Direct Model-Theoretic Semantics for L_0^{NL}

Following the essential ideas in (Montague, 1974) and subsequent work, we can construct a *mathematical model* $\mathcal{M}_0$ for the meaning of sentences in L_0^{NL}. The meaning of a sentence S is modelled in $\mathcal{M}_0$ by a function $\|S\|$ from the class of first-order (FO) models to $\{true, false\}$. Thus, for any FO model M, it should be that $\|S\|^M = true$ iff S is intuitively judged to be true in that model. We say that $T \models^{\mathcal{M}_0} Q$ iff for any FO model M, if $\|S\|^M = true$ for all $S \in T$ then $\|Q\|^M = true$.

A FO model M is a pair $M = \langle D_M, I_M \rangle$ of a non-empty domain D_M and an interpretation function I_M. That function is taken here to be from words of the language L_0^{NL} to appropriate elements over D_M. Thus, in any model M we have $I_M(\text{“John”}) \in D_M$, $I_M(\text{“man”}) \subseteq D_M$, and $I_M(\text{“every”}) = \{\langle A, B\rangle \in \wp(D_M) \times \wp(D_M) \mid A \subseteq B\}$.

This forms the basis of the definition of $\|\cdot\|^M$. Thanks to the simplicity of L_0^{NL}, each node X in a syntactic structure can be assigned an appropriate meaning in a model M, which we will denote using $\|X\|^M$. (Note that since each sentence S has just one syntactic structure $\sigma(S)$, we can define $\|S\|^M = \|\sigma(S)\|^M$.) The interpretation $\|\cdot\|^M$ is defined for

words based on I_M and is extended to larger syntactic constituents through a one-to-one correspondence of the familiar sort between syntax rules and semantic interpretation rules. The definition of this mapping includes:

(22)
- If w is a word then $\|w\|^M = I_M(w)$.
- $\| \, [_{NP[quant+]} \; X_{quant} \; Y_{N'}] \, \|^M = \{A \subseteq D_M \mid \langle \|Y\|^M, A\rangle \in \|X\|^M\}$
- $\| \, [_{S} \; X_{NP[quant-]} \; Y_{VP}] \, \|^M = true$ iff $\|X\|^M \in \|Y\|^M$
- $\| \, [_{S} \; X_{NP[quant+]} \; Y_{VP}] \, \|^M = true$ iff $\|Y\|^M \in \|X\|^M$
- $\| \, [_{VP} \; X_{TV} \; Y_{NP[quant-]}] \, \|^M = \{a \in D_M \mid \langle a, \|Y\|^M\rangle \in \|X\|^M\}$
- $\| \, [_{VP} \; X_{TV} \; Y_{NP[quant+]}] \, \|^M = \{a \in D_M \mid \{b \in D_M \mid \langle a, b\rangle \in \|X\|^M\} \in \|Y\|^M\}$
- $\| \, [_{N'} \; X_{N'} \; Y_{RelC}] \, \|^M = \|X\|^M \cap \|Y\|^M$

Since L_0^{NL} is a restricted enough subset of English, it is possible to find, using this setup, a set of semantic rules, which guarantees that for all T and Q of L_0^{NL} we have $T \models_0^{NL} Q$ iff $T \models^{\mathcal{M}_0} Q$. How can we be sure of that? This is an *empirical* question. The answer cannot be proven mathematically, but rather the model presented here serves as a theory that predicts whether or not a given text entails a given query. The theory's predictions about an expression's meaning should be intuitive. The theory can be tested by examining its predictions on a large enough sample of texts and queries from L_0^{NL}.

Given T and Q, how can we (humans) check whether $T \models^{\mathcal{M}_0} Q$? We perform mathematical reasoning based on the statements in (22). For example, if we have:

(23) $T =$ "Every man is mortal. Socrates is a man."
$Q =$ "Socrates is mortal."

then we can reason as follows:

(24) Suppose that $\|T\|^M = true$. Then $\|$"every man is mortal"$\| = true$. So $\|$"mortal"$\|^M \in \|$"every man"$\|^M$. Also, $\|$"every man"$\|^M = \{A \subseteq D_M \mid \langle \|$"man"$\|^M, A\rangle \in \|$"every"$\|^M\}$. Hence $\langle \|$"man"$\|^M, \|$"mortal"$\|^M\rangle \in \|$"every"$\|^M$. So $\langle I_M($"man"$), I_M($"mortal"$)\rangle \in I_M($"every"$)$. By definition of $I_M($"every"$)$, it follows that $I_M($"man"$) \subseteq I_M($"mortal"$)$. ...

If Q does not follow from T, we would construct a counter model M in which $\|T\|^M = true$ and $\|Q\|^M = false$.

2.5 Direct Translation from NL to FOL

Now we have a way to assign model-theoretic semantics to sentences of L_0^{NL}. This allows us to talk about what sentences mean and test entailments. This is the chief concern of the work in linguistic semantics. However, the interest in this dissertation is to compute entailments. A computer cannot manipulate abstract mathematical entities over $\mathcal{M}_0$ but only representations of such entities that are written in some formal language.

We now turn to define such a representation language. It is customary to say in the linguistic literature, following Montague (1974), that this move is done merely for convenience: It is easier to write expressions in a formal language than definitions in a meta-language as those above, but the intermediate representation level is dispensable. But from a computational perspective, the representations are indispensable. What we still need to make sure, though, is that it is clear what they mean.

One option is to choose the language of FOL. This is possible here because the expressive power of L_0^{NL} is not greater than that of FOL: For each sentence S in L_0^{NL} we can find a closed FOL formula $\tau_0^{\text{FOL}}(S)$ that captures its meaning in the sense that for all sentences $S_1, \ldots, S_n, Q$ in L_0^{NL} we have $S_1, \ldots, S_n \models_0^{NL} Q$ iff $\tau_0^{\text{FOL}}(S_1), \ldots, \tau_0^{\text{FOL}}(S_n) \models^{FOL} \tau_0^{\text{FOL}}(Q)$.

One way to define the translation τ_0^{FOL} is to assign to each node in the syntactic structure an expression in the language of λ-FOL, i.e. FOL extended with λ-expressions. Then, for a sentence S in L_0^{NL}, $\tau_0^{\text{FOL}}(S) = \tau_0^{\text{FOL}}(\sigma(S))$. The clauses of the definition of τ_0^{FOL} mirror the semantic rules given above, including the following (in (25), I abbreviate τ_0^{FOL} as τ):

(25)
- If w is a word not of category Det then $\tau(w) = \textit{string-to-symbol}(w)$. Also:
  $\tau(\text{“every”}) = \lambda P \lambda Q.\forall z.[P@z \rightarrow Q@z]$
  $\tau(\text{“some”}) = \lambda P \lambda Q.\exists z.[P@z \wedge Q@z]$
  $\tau(\text{“no”}) = \lambda P \lambda Q.\neg\exists z.[P@z \wedge Q@z]$
- $\tau([_{NP[quant+]}\ X_{quant}\ Y_{N'}]) = \tau(X)@\tau(Y)$
- $\tau([_{S}\ X_{NP[quant-]}\ Y_{VP}]) = \tau(Y)@\tau(X)$
- $\tau([_{S}\ X_{NP[quant+]}\ Y_{VP}]) = \tau(X)@\tau(Y)$
- $\tau([_{VP}\ X_{TV}\ Y_{NP[quant-]}]) = \lambda z.\tau(X)(z, \tau(Y))$
- $\tau([_{VP}\ X_{TV}\ Y_{NP[quant+]}]) = \lambda z_1.\tau(Y)@(\lambda z_2.\tau(X)(z_1, z_2))$
- $\tau([_{N'}\ X_{N'}\ Y_{RelC}]) = \lambda z.\tau(X)@z \wedge \tau(Y)@z$
- where P, Q, z, z_1, z_2 are fresh variables (in each application of τ).

If γ is a simple symbol and x is a variable then $\gamma@x$ is $\gamma(x)$. Otherwise, @ stands for λ-application including β-reduction. Since we always use fresh variables, there is no need to worry about accidental binding of variables.

For example, this translation yields:

(26) a. $\tau_0^{\text{FOL}}(\text{No man walks}) = \neg\exists x.[man(x) \wedge walk(x)]$

b. $\tau_0^{\text{FOL}}(\text{Mary sees every French man}) = \forall x.[man(x) \wedge french(x) \rightarrow see(mary, x)]$

This kind of translation is given, for example, in (Blackburn and Bos, 2005a).

We want to show that for all pairs of a text T and a query Q in L_0^{NL}, $T \models_0^{NL} Q$ iff $\tau_0^{\text{FOL}}(T) \models^{FOL} \tau_0^{\text{FOL}}(Q)$. This is again an empirical question, as it was with $\models^{\mathcal{M}_0}$. However, we can also ask a mathematical question, which can be proven precisely. The question is whether (27) is true:

(27) For all T and Q in L_0^{NL}, $\tau_0^{\text{FOL}}(T) \models_0^{NL} \tau_0^{\text{FOL}}(Q)$ iff $T \models^{\mathcal{M}_0} Q$.

Assuming that $\mathcal{M}_0$ is a correct model for $\models_0^{NL}$, (27) entails that τ_0^{FOL} is a faithful translation of L_0^{NL} to FOL.

How can (27) be proved? The easiest way is to provide a model-theoretic semantics for λ-FOL, and then use it to prove inductively that for every node X in a syntactic structure, the meaning of the λ-FOL associated with X is the same as $\|X\|$ according to (22). This claim would then be true in particular for syntactic structures of whole sentences, and from this, (27) immediately follows.

There is another independent motivation for providing a model-theoretic semantics for λ-FOL. Whenever one uses a formal language, it is important to be clear about what this language means. If it is not clear what the language means then we cannot be sure whether syntactic transformations we perform on formulas are valid in the sense that they are justifiable according to our intuitive understanding of the formulas. The case in point here is β-reduction in λ-FOL: how can we be sure the resulting expressions make sense? Knowing what a formal language means is especially important when it is used to express the meaning of NL expressions. The formal language should be a precise tool for expressing those meanings. See (McDermott, 1978) for a relevant discussion.

Here, FOL is our meaning representation language, and every sentence in L_0^{NL} is translated by τ_0^{FOL} to a closed sentence of FOL, without any lambda expressions from λ-FOL.

The model-theoretic semantics of FOL is well-known. Still, it is also important to provide model-theoretic semantics for the intermediate expressions in λ-FOL for the reasons mentioned above. An expression in λ-FOL that is associated via τ_0^{FOL} with a node X should *represent* the mathematical entity $\|X\|$. These representations allow the computer to reason about these abstract entities even though it does not have direct access to them.

2.6 Modularity

Instead of providing a model-theoretic semantics for λ-FOL, I will use a more general meaning representation language (MRL), called L_0^{MR}, and provide a model-theoretic semantics for it. I will also define a faithful translation τ_0 from L_0^{NL} to L_0^{MR}. All this will be done in section 2.7.

The are several reasons for switching here from λ-FOL to L_0^{MR}. First, FOL by itself is not powerful enough to *directly* represent the meaning of many NL phenomena. For example, τ_0^{FOL} cannot be extended to directly translate the quantifier *most* and other proportional quantifiers (*half of the*, *two thirds of the*, etc.) to FOL,[1] as well as numerical comparatives (because they talk about the sizes of sets), plurals, intensionality, and so on. It is only possible to give an *indirect* translation to FOL if we also rely on a sufficiently large set of additional axioms.

Should we simply use another logical language for our meaning representations? The problem with choosing any particular language is that the language might be sufficient for NL phenomena we have considered, but it might not be generalizable or extendible to additional NL phenomena. Committing now to a particular meaning language might force us later to revise all the work we will have done by the time we encounter that next phenomenon. In particular, the specification of the syntax-semantics interface may need to be revised each time we extend the fragment.

The way I want to deal with this problem is to use a meaning language whose most important criterion is the shape of its formulas. That shape should be as close as possible to the surface structure of the NL sentences and be amenable to compositional computation from it, while at the same time, be semantically intuitive. For example, the shape of formulas in FOL is quite different from the surface form of NL. The sentences (28)a and (28)b are the same in every respect except for the quantifier they use. This similarity is

[1] See chapter 13 of (Peters and Westerståhl, 2006), and also section 10.10.2 of this dissertation.

lost in the FOL formulas (28)c and (28)d. Furthermore, the FOL quantifiers $\forall$ and $\exists$ are type $\langle 1 \rangle$ quantifiers in that they accept a predicate of individuals (the structure of the formulas is $Qx.\varphi$), whereas the NL quantifiers *every* and *some* are type $\langle 1, 1 \rangle$ quantifiers that accept two predicates (Keenan and Westerståhl, 1997). The MRL should reflect this, and so (28)e-f are the appropriate formulas.

(28) a. Every man left.
b. Some man left.
c. $\forall x.[man(x) \rightarrow left(x)]$
d. $\exists x.[man(x) \wedge left(x)]$
e. $every(man, left)$
f. $some(man, left)$

In this way, if we extend the coverage of our analysis to additional NL phenomena, there should be very little, if any, modifications to the shape of the language and to the syntax-semantics interface. This goal will sometimes lead us to introduce several operators that may seem redundant because they are definable in terms of more basic operators, but are in fact necessary in order to simplify the syntax-to-semantics mapping (see for example section 10.3.2).

In addition, the meaning language should have a well-defined MTS. However, the MTS should be defined in such a way that it could be revised whenever the definition of the language is extended, without requiring much change to the shape of language expressions we previously used. For example, the denotation of formulas in L_0^{MR} below is a truth value, but it can be easily changed to be a proposition (i.e., an intension which may have different truth values in different worlds or contexts) with very minimal change to the shape of the formulas. The goal here will sometimes be achieved by treating some expressions in the meaning language as a shorthand or as macros that can be expanded to more basic expressions.

Because of these requirements, the MRL will be a higher-order typed logical language that will become increasingly complex as we incorporate more NL phenomena into it. That means it will become increasingly harder to program the computer to do automated reasoning with it because reasoning tools for such languages are not well-developed yet. The solution is to introduce another modularity between the meaning language and an inference language, which I will designate using L^{IMP} for "implementation language."

This language should be tailored to the particular application one is interested in, and should support efficient reasoning for it. This separation allows the MRL to be general for NL and to be usable in many different domains and applications. In contrast, if we tried to directly translate the NL to L^{IMP}, it would be difficult because the shape of L^{IMP} formulas might be very different from NL surface structure and not easily amenable to compositional computation; and furthermore, our work would not be portable to other domains. For example, some logic puzzles can be represented as constraint satisfaction problems (CSPs) over finite universes, but we certainly do not want to directly translate NL sentences to such representations. I will continue to discuss these issues in section 2.8.

2.7 Translation from NL to a MRL

2.7.1 The Language L_0^{MR}

The main difference between L_0^{MR} and FOL is that in the former we have lambda terms that represent constructed predicates (or equivalently, their set extensions). As usual in the semantics literature, we use e for the type of individuals, and t for truth values ($true$ and $false$). In the following definition of the language, I abbreviate $e \to t$ as et.

(29)
- terms:
  1. *individual terms*:
     - If w is a word of L_0^{NL} of category PN then *string-to-symbol*(w) is an *individual constant* (of type e).
     - `x`, `y`, `z`, $\texttt{x}_1$, $\texttt{x}_2$, ... are *individual variables* (of type e).
  2. *predicate terms*:
     - If w is a word of L_0^{NL} of category N, V, Adj, Det, then *string-to-symbol*(w) is a *predicate constant* of the appropriate type (e.g. the type of `man` is et and the type of `every` is $et \to et \to t$).
     - `X`, `Y`, `Z`, $\texttt{X}_1$, $\texttt{X}_2$, ... are *predicate variables* (of type et).
     - If $x_1, \ldots, x_n$ are variables that occur free in the formula φ, and x_i is of type $\tau_i \in \{e, et\}$, then $\lambda x_1 \ldots \lambda x_n.\varphi$ is a *lambda term* of arity n (and of type $\tau_1 \to \ldots \to \tau_n \to t$).
- formulas:
  - If γ is a predicate term of arity n and $t_1, \ldots, t_k$ for $1 \leq k \leq n$ are terms of the appropriate types for the first k arguments of γ, then $\gamma(t_1, \ldots, t_k)$ is a formula if

$k = n$ and a predicate term if $k < n$.

– If φ and ψ are formulas then so are $\neg\varphi$, φ `&` ψ, and φ `v` ψ.

Here are example formulas for NL sentences:

(30) a. Some man walks.

`some(man,walk)`

b. Mary sees every French man.

`every(`λ`z.man(z) & french(z),` λ`z.see(mary,z))`

2.7.2 Model-Theoretic Semantics

The model-theoretic semantics is defined as follows. A FO model M for L_0^{MR} is defined in the same way as models in $\mathcal{M}_0$ above, except that the interpretation function I_M maps symbols of L_0^{MR}, rather than words of L_0^{NL}, to appropriate mathematical objects. In particular, $I_M(\texttt{john}) \in D_M$ and $I_M(\texttt{every}) = \{\langle A, B\rangle \in \wp(D_M) \times \wp(D_M) \mid A \subseteq B\}$. Here we also need to use assignments which map individual variables to elements of D_M. The meaning of expressions is defined as expected:

(31)

- individual terms:
  - If c is a constant then $[\![c]\!]^{M,g} = I_M(c) \in D_M$
  - If x is an individual variable then $[\![x]\!]^{M,g} = g(x) \in D_M$
- predicate terms:
  - If x is a predicate variable then $[\![x]\!]^{M,g} = g(x) \subseteq D_M$
  - If p is a predicate symbol of arity n, and its ith argument is of type $\tau_i \in \{e, et\}$, then $[\![p]\!]^{M,g} = I_M(p) \subseteq (D_M)^{\tau_1} \times \ldots \times (D_M)^{\tau_n}$, where $(D_M)^e = D_M$ and $(D_M)^{et} = \wp(D_M)$
  - If $x_1, \ldots, x_n$ are variables that occur free in the formula φ, and x_i is of type $\tau_i \in \{e, et\}$, then $[\![\lambda x_1 \ldots \lambda x_n.\varphi]\!]^{M,g} = \{\langle a_1, \ldots, a_n\rangle \in (D_M)^{\tau_1} \times \ldots \times (D_M)^{\tau_n} \mid [\![\varphi]\!]^{M,g[x_1 \to a_1, \ldots, x_n \to a_n]} = true\}$.
- formulas:
  - If γ is a predicate term of arity n and $t_1, \ldots, t_n$ are terms that are appropriate as arguments of γ, then $[\![\gamma(t_1, \ldots, t_n)]\!]^{M,g} = true$ iff $\langle [\![t_1]\!]^{M,g}, \ldots, [\![t_n]\!]^{M,g}\rangle \in [\![\gamma]\!]^{M,g}$
  - If γ is a predicate term of arity n and $t_1, \ldots, t_k$, $1 \leq k < n$, are terms that are appropriate as the first k arguments of γ, then $[\![\gamma(t_1, \ldots, t_k)]\!]^{M,g} = \{\langle a_{k+1}, \ldots, a_n\rangle \in (D_M)^{\tau_{k+1}} \times \ldots \times (D_M)^{\tau_n} \mid \langle [\![t_1]\!]^{M,g}, \ldots, [\![t_k]\!]^{M,g}, a_{k+1}, \ldots, a_n\rangle \in [\![\gamma]\!]^{M,g}\}$.

– If φ and ψ are formulas then so are $\neg\varphi$, φ & ψ, and φ v ψ, and
$[\![\varphi \texttt{ \& } \psi]\!]^{M,g} = true$ iff $[\![\varphi]\!]^{M,g} = true$ and $[\![\psi]\!]^{M,g} = true$, etc.

For $\varphi, \psi \in L_0^{MR}$ without free variables, we say $\varphi \models_0^{MR} \psi$ iff for all models M, if $[\![\varphi]\!]^M = true$ then $[\![\psi]\!]^M = true$, and we extend this to a set of formulas on the left as usual.

2.7.3 Calculating the Representations

Now we want to define a translation τ_0 from syntactic structures of L_0^{NL} to L_0^{MR} such that each node X in a syntactic structure receives a faithful translation, so that $\tau_0(X)$ represents $\|X\|$. The translation rules are very similar to those in (25) except we do not give explicit clauses to the quantifiers (since they are all covered by the second line in the definition here):

(32)
- If w is a word then $\tau(w) =$ *string-to-symbol*(w).
- $\tau([_{NP[quant+]}\ X_{quant}\ Y_{N'}]) = \lambda\mathtt{Z}.\ \tau(X)(\ \tau(Y)\ ,\mathtt{Z})$
- $\tau([_S\ X_{NP[quant-]}\ Y_{VP}]) = \tau(Y)(\ \tau(X)\ )$
- $\tau([_S\ X_{NP[quant+]}\ Y_{VP}]) = \tau(X)(\ \tau(Y)\ )$
- $\tau([_{VP}\ X_{TV}\ Y_{NP[quant-]}]) = \lambda\mathtt{z}.\tau(X)(\ \mathtt{z},\ \tau(Y)\ )$
- $\tau([_{VP}\ X_{TV}\ Y_{NP[quant+]}]) = \lambda\mathtt{z}_1.\tau(Y)(\lambda\mathtt{z}_2.\tau(X)(\mathtt{z}_1,\mathtt{z}_2))$
- $\tau([_{N'}\ X_{N'}\ Y_{RelC}]) = \lambda\mathtt{z}.\ \tau(X)(\mathtt{z})\ \texttt{\&}\ \tau(Y)(\mathtt{z})$
- where $\mathtt{Z}$, $\mathtt{z}$, $\mathtt{z}_1$, $\mathtt{z}_2$ are fresh variables

Now we need to show:

(33) a. For all syntactic nodes X and for all models M, $\|X\|^M = [\![\tau_0(X)]\!]^{\tau_0(M)}$
(where $\tau_0(M) = \langle D_M, \lambda w.I_M($*symbol-to-string*$(w))\rangle$).

b. For all T and Q of L_0^{NL}, $T \models^{\mathcal{M}_0} Q$ iff $\tau_0(T) \models_0^{MR} \tau_0(Q)$.

The second claim follows from the first. The first claim is proved by induction on syntactic structures, following the definitions in (22) and (32). For example, one part of the proof is:

(34) We prove that $[\![\tau_0([_{N'}\ X_{N'}\ Y_{RelC}])]\!]^M = \|\ [_{N'}\ X_{N'}\ Y_{RelC}]\ \|^M$ as follows:

$[\![\tau_0([_{N'}\ X_{N'}\ Y_{RelC}])]\!]^M = [\![\lambda \mathtt{z}.\ \tau_0(X)(\mathtt{z})\ \&\ \tau_0(Y)(\mathtt{z})]\!]^M =$
$\{a \in D_M \mid [\![\tau_0(X)(\mathtt{z})\ \&\ \tau_0(Y)(\mathtt{z})]\!]^{M,[\mathtt{z}\to a]} = true\} =$
$\{a \in D_M \mid [\![\tau_0(X)(\mathtt{z})]\!]^{M,[\mathtt{z}\to a]} = true \wedge [\![\tau_0(Y)(\mathtt{z})]\!]^{M,[\mathtt{z}\to a]} = true\} =$
$\{a \in D_M \mid [\![\mathtt{z}]\!]^{M,[\mathtt{z}\to a]} \in [\![\tau_0(X)]\!]^M \wedge [\![\mathtt{z}]\!]^{M,[\mathtt{z}\to a]} \in [\![\tau_0(Y)]\!]^M\} =$
$\{a \in D_M \mid a \in [\![\tau_0(X)]\!]^M \wedge a \in [\![\tau_0(Y)]\!]^M\} =$
$\{a \in D_M \mid a \in [\![\tau_0(X)]\!]^M\} \cap \{a \in D_M \mid a \in [\![\tau_0(Y)]\!]^M\} =$
$[\![\tau_0(X)]\!]^M \cap [\![\tau_0(Y)]\!]^M =$ (by induction hypothesis)
$\|X\|^M \cap \|Y\|^M = \|\ [_{N'}\ X_{N'}\ Y_{RelC}]\ \|^M.$

In section 3.1.3, we will see that this simple way of assigning a denotation directly to each node of the syntactic structure by combining the denotations of its daughter nodes does not generalize well to more complex NL constructions. We will then use a more powerful framework for specifying the mapping from syntax to semantics.

2.8 Proof System and Computation

2.8.1 General Discussion

Now that we have representations in L_0^{MR} of the mathematical objects which model the meanings of L_0^{NL} expressions, a computer can (in principle) perform reasoning with them. Our two goals now are:

1. Find a proof system $\mathcal{C}_0^{MR}$ for L_0^{MR}, with a provability relation $\vdash_0^{MR}$, which is sound and complete w.r.t. the semantics of L_0^{MR}, i.e. for all sets of formulas Γ and formulas φ in L_0^{MR} we have $\Gamma \vdash_0^{MR} \varphi$ iff $\Gamma \models_0^{MR} \varphi$.

2. Based on this proof system, build an efficient computer program that finds these proofs and decides the provability relation $\vdash_0^{MR}$.

Generally speaking, we might encounter two problems at this point. The first problem is that even if we can fine a sound and complete proof system for our meaning representation language, L^{MR}, entailment in that system might be undecidable. This is the case if L^{MR} is FOL or any stronger logic. The second problem is that L^{MR} might not even have a sound and complete proof system.

Nevertheless, note that we do not necessarily need to worry about the entire language L^{MR}. This is because the expressions we get when translating NL sentences to L^{MR} might

actually all fall in a subset of L^{MR}, for which the two problems go away. As long as we know that the NL entailment relation $\models^{NL}$ that we are modelling is decidable on the queries from L^{NL} that interest us, that means we *can* find some model of it and a computer program that decides it.

For example, although FOL is undecidable, if a set $\mathcal{S}$ of FOL queries is such that for every query $\langle \Gamma, \varphi \rangle$ in the set (where Γ is finite), either $\Gamma \models^{FOL} \varphi$ or $\Gamma \cup \{\neg\varphi\}$ is *finitely* satisfiable, then both the set $\{\langle \Gamma, \varphi \rangle \in \mathcal{S} \mid \Gamma \models^{FOL} \varphi\}$ and the set $\{\langle \Gamma, \varphi \rangle \in \mathcal{S} \mid \Gamma \not\models^{FOL} \varphi\}$ are decidable. For each query $\langle \Gamma, \varphi \rangle$, simply run in parallel a theorem prover on $\langle \Gamma, \varphi \rangle$ and a model builder on $\Gamma \cup \{\neg\varphi\}$, and one of them is guaranteed to terminate with an answer. In particular, if we consider the language L_0^{NL} and the translation τ_0^{FOL}, then the set of queries $\{\langle T, Q \rangle \mid T, Q \text{ are in } L_0^{NL} \text{ and } \tau_0^{\text{FOL}}(T) \models^{FOL} \tau_0^{\text{FOL}}(Q)\}$ is decidable (for a relevant result, see (Pratt-Hartmann, 2003)).

One way to take advantage of the point that the representations resulting from translating NL all fall in a subset of L^{MR} is to translate those representations to yet another language, L^{IMP}, that is more tailored (both in proof system and in computer program) to the class of entailment questions we actually get. For example, many logic puzzles can be represented as constraint satisfaction problems (CSPs) over finite universes. We would need to prove that the translation from L^{MR} to L^{IMP}, call it π_0, is faithful in the following sense:

(35) For all T and Q in L_0^{NL}, $\tau_0(T) \models_0^{MR} \tau_0(Q)$ iff $\pi_0(\tau_0(T)) \models_0^{IMP} \pi_0(\tau_0(Q))$.

One might then ask: if the NL queries can be represented in L^{IMP}, why not translate L^{NL} directly to that? I have already addressed this question in section 2.6. One of the reasons was that the *shape* of expressions in L^{IMP} might be superficially very different from the surface structure of NL, so defining a direct translation from NL to L^{IMP} might be very hard. For example, although the expressions resulting from the τ_0 translation above are structurally similar to the NL expressions, performing β-conversions on them makes the resulting expressions less similar to the NL expressions. Translating those to a finite CSP language would yield expressions that are even more different than the surface NL. In contrast, a central goal in designing the MRL is that the shape of expressions will mirror as much as possible the NL surface structure.

2.8.2 Proof System and Computation for L_0^{NL}

To illustrate, if we were only interested in L_0^{NL}, we could choose L^{IMP} to be FOL. For each sentence S of L_0^{NL}, we translate $\tau_0(S) \in L_0^{MR}$ to a formula of L^{IMP}. The translation has two steps:

1. Recursively traversing the formulas in L_0^{MR}, and converting the quantifier symbols, such as `every`, to an equivalent λ-FOL representation:

   $\mu_0(\texttt{every(}\varphi\texttt{,}\psi\texttt{)}) = \forall \texttt{z.[}\mu_0(\varphi)\texttt{(z)} \rightarrow \mu_0(\psi)\texttt{(z)]}$
   $\mu_0(\texttt{some(}\varphi\texttt{,}\psi\texttt{)}) = \exists \texttt{z.[}\mu_0(\varphi)\texttt{(z) \& }\mu_0(\psi)\texttt{(z)]}$
   $\mu_0(\texttt{no(}\varphi\texttt{,}\psi\texttt{)}) = \neg\exists \texttt{z.[}\mu_0(\varphi)\texttt{(z) \& }\mu_0(\psi)\texttt{(z)]}$

   We need to prove that the transformation preserves meaning. We can extend L_0^{MR} to include $\forall$ and $\exists$, give them the usual meaning, and prove the claim based on the meaning assigned to the quantifier symbols by the interpretation function I_M.

2. Performing β-reduction (R_β):

   (36) $(\lambda x.\varphi)(t) \Rightarrow_\beta \varphi[t/x]$

   We need to prove that this transformation preserves meaning, i.e. that for all $\varphi \in L_0^{MR}$, $[\![\varphi]\!]^{M,g} = [\![R_\beta(\varphi)]\!]^{M,g}$. This is done by induction on the structure of formulas. In particular:

   $[\![(\lambda x.\varphi)(t)]\!]^{M,g} = true$ iff
   $[\![t]\!]^{M,g} \in [\![\lambda x.\varphi]\!]^{M,g}$ iff
   $[\![t]\!]^{M,g} \in \{a \in D_M \mid [\![\varphi]\!]^{M,g[x\rightarrow a]} = true\}$ iff
   $[\![\varphi]\!]^{M,g[x\rightarrow a']} = true$ where $a' = [\![t]\!]^{M,g}$ iff (by a lemma that needs to be proved)
   $[\![\varphi[t/x]]\!]^{M} = true$ iff
   $[\![R_\beta((\lambda x.\varphi)(t))]\!]^{M} = true$

The resulting formulas are in FOL proper. To see this, note that μ_0 gets rid of all lambda expressions that are arguments to quantifiers. The β-conversion gets rid of all other lambda expressions.

To summarize how this helps us decide the relation $\models_0^{NL}$, we have: for all texts T and queries Q in L_0^{NL}, $T \models_0^{NL} Q$ iff $\tau_0(T) \models_0^{MR} \tau_0(Q)$ (this is an empirical observation). The latter is true iff $R_\beta(\mu_0(\tau_0(T))) \models^{FOL} R_\beta(\mu_0(\tau_0(Q)))$ (by the explanations given above),

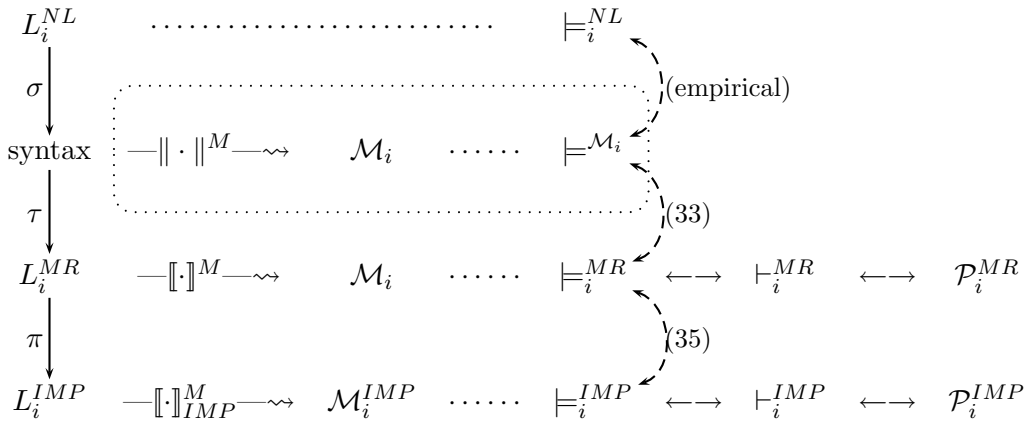


Figure 2.1: The Basic Framework

thus showing that (35) holds here for $\pi_0 = R_\beta \circ \mu_0$. So now we can use FOL reasoners to decide on L_0^{NL} queries (as $\models_0^{NL}$ is decidable).

2.9 Summary of the Basic Framework

Figure 2.1 shows a picture of the framework so far. The i indicates that this is a fragment of NL. The plan is then to gradually extend the fragment. Once we have all the components in place for L_i^{NL}, we move on to a larger fragment $L_{i+1}^{NL} \supset L_i^{NL}$, investigate $\models_{i+1}^{NL}$, and extend the other components. The dashed arrows indicate correspondence. The dotted area exists only for simple fragments where a rule-to-rule translation is possible. Otherwise, we have a direct correspondence between $\models^{NL}$ and $\models^{MR}$, which is "empirical." The arrows between $\models$ and $\vdash$ indicate that the proof system is sound and (hopefully) complete w.r.t. the semantics. The arrows between $\vdash$ and $\mathcal{P}$ indicate that the program that finds proofs needs to be sound (if it finds a proof, this is indeed a proof) and (hopefully) complete (if there is a proof, it finds it). Both kinds of soundness and completeness are really required only for the subset of L^{MR} and L^{IMP} that are obtained by translation from L^{NL}.

2.10 Extensions

The basic framework above covers only very simple sentences and issues. NL has many aspects that go beyond the straightforward approach presented above.

2.10.1 Syntax-Semantics Interface

The syntax of a formal language is deliberately designed to correlate straightforwardly with its semantics. This allows a very simple compositional definition of the MTS using a rule-by-rule translation, i.e. for each syntax rule in the definition of the formal language there is one corresponding MTS rule that specifies the meaning of the expression as a simple combination of the meanings of the expression's immediate parts. Also, the language is designed so that possible ambiguities are resolved by using explicit syntactic devices (such as adding parentheses).

The situation in NL is much more complex. As an example, imagine a formal language with a statement that looks like this:

(37) $R(a, (\forall x))$

The quantifiers appear in the formula *in situ*, but as far as the meaning, it should scope over the predication:

(38) $\forall x.R(a, x)$

Formal languages are not designed in this way, but NL has this:

(39) John saw every boy.
$\forall x.boy(x) \rightarrow saw(john, x)$

There is also the question which of the quantifiers scopes above the other in:

(40) $R((\forall x), (\exists y))$

It seems useless to define a formal language where this is not completely apparent from the formula. In contrast, this is common in NL, as the following classical examples demonstrate:

(41) In this country, a woman gives birth every 15 minutes. Our job is to find that woman, and stop her. (Groucho Marx)

(42) Every man likes a woman.
$\forall x.man(x) \rightarrow \exists y.woman(y) \wedge like(x, y)$
$\exists y.woman(y) \wedge \forall x.man(x) \rightarrow like(x, y)$

Quantifiers can have scope interactions with other operators such as negation. They may also float far above their surface position:

(43) John believes he saw each man.
(one reading:) $\forall x.man(x) \rightarrow bel(john, see(john, x))$

Because of these issues and others, the rule-by-rule definition of the syntax-to-semantics mapping in the basic framework above cannot scale up to realistic fragments of NL. I will discuss this issue in the next chapter. The τ in figure 2.1 should be replaced with the kind of glue semantics specification that we will see in subsequent chapters.

2.10.2 Ambiguity

Because NL sentences are ambiguous, the relation $\models^{NL}$ is not well-defined. A decision about entailment depends on the particular *reading* one is interested in for each sentence. The $\tau \circ \sigma$ in Figure 2.1 is really a one-to-many mapping. I will show how the glue semantics specification takes care of this.

Context-Dependence

What can we say about the question $T \models^{NL} Q$ in the face of ambiguities? Usually, we are interested in entailment between the readings of T and Q that are the preferred ones in some context such as a computer application. The application context hopefully supplies enough information and domain knowledge for the computer to be able to choose the correct reading and decide on entailment. Finding out the correct reading may itself require reasoning, so this is an iterative process. In some cases, no resolution can be found because the query is inherently ambiguous, and one option is to ask the user for a clarification.

Context-Independent Entailment

If one still insists on thinking about context-independent entailment, there are a few options. The strictest criterion for deciding whether T entails Q is to require all interpretations of T to entail all interpretations of Q. The defined entailment relation $\models_\forall$ is weak in the sense that not much is entailed. In particular, an ambiguous sentence does not entail itself:

If a sentence S is ambiguous, and has at least two non-equivalent readings φ and ψ, then either $\varphi \not\models^{MR} \psi$ or $\psi \not\models^{MR} \varphi$. Hence $S \not\models_{\forall} S$.

If instead we require that some interpretation of T entails some interpretation of Q, we get a relation $\models_{\exists}$ that is too strong:

(44) Every man saw some woman. $\models_{\exists}$ All men saw the same woman.

A third possibility is to require each interpretation of T to entail at least one interpretation of Q. Under this entailment relation $\models_{\forall\exists}$, every sentence does entail itself. This seems the most useful entailment relation of the three. Here is an example of an ambiguous text that nonetheless entails an ambiguous query under $\models_{\forall\exists}$:

(45) At least three boys visited every room. Every boy is a human. Some rooms are red.
$\models_{\forall\exists}$ At least three humans visited every red room.

Entailments based on monotonicity properties of quantifiers are (usually) not affected by scope ambiguity.

Packed Inference

Both context-dependent and independent inferences seem to require investigating all readings separately. Is it possible to do better and find cases of entailment by working directly on packed semantic representations?

There are certain proposals for performing specific kinds of inferences. For example, Winter (2005) shows steps towards calculating entailments based on monotonicity properties of quantifiers without disambiguating a sentence. Koller and Thater (2006) show an algorithm that takes an underspecified semantic representation and computes one with fewer mutually equivalent readings. I already mentioned at the end of section 1.3.3 the work that was done at PARC on packed inference. The most general view is to do inference under all contexts in the choice space efficiently, getting an answer which says under which contexts the premises entail the conclusion. This kind of packed inference, however, is not yet available at this time.

2.10.3 Linguistic Coverage

The framework here covers only very simple sentences. The linguistic coverage needs to be gradually extended.

Morphology and Syntax

NL morphology and syntax are more complex than in formal languages. For example, words may be obtained from simpler words by adding prefixes and suffixes. In contrast to formal languages, the syntax of NL requires more than a simple context-free grammar. For example, it requires features to describe agreement in person and number between different parts (46), and it has long-distance dependencies such as in relative clauses (47).

(46) John likes/*like Mary.
John and Bill like/*likes Mary.

(47) The man that [John introduced Mary to] arrived.

Since there is already a very good theoretical and computational understanding of these topics, I will have almost nothing to say about them in this dissertation. I will simply use the XLE system for these matters.

Events

A verb may take a varying number of arguments, and additionally, an unbounded number of further VP modifiers. To deal with this variation in a uniform way, we would like to represent events explicitly using event variables, so that the event variable could be further modified, following a neo-Davidsonian approach. I will address this in section 4.4.

Contextual Parameters

A formal language may introduce a variable which can be reused several times in a local scope, but that scope is fixed and clearly delineated. In contrast, NL introduces elements into the discourse, which may then be referred back to by anaphoric expressions, as the subscripts indicate:

(48) [A man]$_i$ sent a present to Susan$_j$. He$_i$ likes her$_j$.

If we tried to directly represent this in FOL, we would run into problems:

(49) $\exists x.man(x) \wedge \exists y.present(y) \wedge send(x, y, susan)$
$like(z, w)$

What tells us that w should refer to Susan and could be replaced with *susan*? Worse, changing z to x yields a formula with a free variable x that is not bound by the $\exists$ quantifier in the previous formula. Also, unlike statements in FOL, we cannot treat sentences in NL as an unordered set, because their order does matter:

(50) * He_i likes her_j. $[\text{A man}]_i$ sent a present to Susan_j.

I will address these issues in chapter 8. Other contextual parameters include *indexicals*, which are expressions whose referent depends on the circumstances of utterance, e.g. *I*, *you*, *now*, *yesterday*, *here*. These parameters are sometimes not mentioned explicitly, as when a sequence of sentences in a story indicates a gradual progression of time. The domain of quantification of a quantifier can also depend on the context, as was mentioned in section 1.2.3.

Reduction (Ellipsis and Pro-Form)

In a formal language, we do not get formulas such as (51)a, with the intended interpretation (51)b.

(51) a. $P(c) \wedge Q(c) \wedge also(d)$
b. $P(c) \wedge Q(c) \wedge P(d) \wedge Q(d)$

In contrast, NL has a phenomenon of *reduction* where part of a sentence is not expressed explicitly because it is very similar to another part of the sentence or discourse. The part which is not expressed is either replaced by a short (one or two words long) *pro-form*, as in (78)a, or altogether omitted or *elided*, as in (78)b.

(52) a. If Zena plays the piano, then Xara *does so*, too.
b. If owls are not in the forest then sparrows are $\triangle$.

One challenge is identifying correctly the sentence's syntactic structure: some of its parts are missing and the remaining sentence is grammatically "defective" in some way. Also, inference from a reduced sentence must proceed as if the missing material was there explicitly, and so it has to be "reconstructed" in some way and become an explicit part of the semantic representation. In puzzle texts, the reduction's antecedent that contains the missing material is usually within the same sentence. Here are some more interesting examples from puzzle texts:

(53) a. In each case, the wife is listed first, the husband $_{\triangle}$ second.
(i.e.: the husband is listed second)

b. Olga's aisle is numbered higher than either $_{\triangle}$ of Kurt's aisles $_{\triangle}$, and $_{\triangle}$ lower than at least one of Larisa's $_{\triangle}$.
= Olga's aisle is numbered higher than either [one] of Kurt's aisles [is numbered], and [Olga's aisle is numbered] lower than at least one of Larisa's [aisles] [is numbered].

c. An animal shelter houses eighteen puppies in three kennels, six puppies per kennel.
= six puppies [are housed] per [each] kennel.

Reduction is particularly common in comparative constructions, which are themselves common in puzzle texts. For example, (54)b is a reduced form of (54)a, and (54)c is a reduced form of (54)b.

(54) a. Ella lifted two more boxes than Victor lifted.
b. Ella lifted two more boxes than Victor did.
c. Ella lifted two more boxes than Victor.

I will mention ellipsis again in chapter 9, and comparative constructions in chapters 10, 11, and 14.

Intensionality

In FOL, the concepts "mutually entailing sentences" and "the same proposition" are the same. In other words, if $\varphi \vDash \psi$ and $\psi \vDash \varphi$ then any occurrence of φ in a larger formula can be replaced with ψ while preserving truth values. For example:

(55) If $\varphi \vDash \psi$ and $\psi \vDash \varphi$ then:
$\varphi \wedge \tau \vDash \psi \wedge \tau$ and $\psi \wedge \tau \vDash \varphi \wedge \tau$
$\varphi \rightarrow \tau \vDash \psi \rightarrow \tau$ and $\psi \rightarrow \tau \vDash \varphi \rightarrow \tau$
etc.

This is not so in NL. For example, the following two sentences are mutually entailing, i.e. if one is true in a certain situation then so is the other:

(56) It is Fred who likes Mary.
It is Mary whom Fred likes.

However, they express different propositions. This can be seen when the sentences are embedded in a larger sentence, where each is not freely substitutable for the other. The following two sentences are not mutually entailing:

(57) Bill explained to me why it is Fred who likes Mary.
Bill explained to me why it is Mary whom Fred likes.

A similar thing occurs with objects. In FOL, if $c = d$ then we can replace any occurrence of c with that of d and preserve truth conditions: $c = d \vDash \psi \leftrightarrow \psi[d/c]$. In contrast, even if we know that (58)a and (58)b are true, we cannot freely substitute "the evening star" for "the morning star" in (58)b to get (58)c. Similarly, while (58)d is true a-priori, (58)a was a major astronomical discovery.

(58) a. The morning star is the evening star.
b. John believes that the morning star is Venus.
c. John believes that the evening star is Venus.
d. The morning star is the morning star.

Dealing with intensionality requires using a weaker logical system that does not always allows substitutivity of equals. We would need to revise the MTS definition here, and interpret NL sentences (clauses) as propositions (intensional objects) rather than truth values. A proposition can have different truth values in different situations.

Starting in the next chapter, some example sentences will have embedded clauses, for instance: "John thinks that Bill left". Although this requires revising the MTS here, this is not critical for the purpose of discussing the syntax-semantics interface. Therefore, I will usually gloss over this point, and continue using the sorts e and t.

2.10.4 Lexical Semantics and World Knowledge

NL usually assumes a large body of background world knowledge, without which, very few conclusions could be drawn from the text (this was already mentioned in section 1.2.3). Therefore, if Γ encodes the information explicit in the text and φ encodes a query, we are usually not interested in knowing whether $\Gamma \vDash \varphi$ (this usually does not hold) but instead we want to know whether $\Gamma \cup K \vDash \varphi$, where K encodes relevant world knowledge. Lexical semantic knowledge could be handled in a similar way. See also sections 15.2.2 and 15.2.3.

2.10.5 Non-Binary Truth Conditions

The view of truth in the basic framework above is extremely simple: every sentence is either completely true or completely false, and there is complete knowledge about it. NL is more complex in various ways.

Presuppositions

In some programming languages, certain operations come with expectations about their input. For example, a division operation expects its second argument to be non-zero, and a dereferencing operator expects its pointer argument to be non-null. If these expectations are not met, the expression cannot return a normal value, and an exception is raised that needs to be treated in a different manner.

Similarly, some expressions in NL fail to have a denotation if some condition, a presupposition, is not met. As I discussed in section 2.1, a query cannot be answered by either *Yes* or *No* if it relies on an unmet presupposition. For various approaches to handle this issue, see (Beaver, 1997).

Gradability

Statements in many logics are usually either true or false with no shades of gray in between (exceptions include some multi-valued logics). In NL, some statements cannot be assigned a definite binary truth value. For example, for some values of X we would be willing to answer the question in (59) with *Yes*, and for some values of X with *No*, but for intermediate values the answer is unclear.

(59) John is X years old.
Is John old?

To address this, the MTS of our meaning representation language could be extended to use a range $[0, 1]$ of truth values rather than the binary ones. Work on Fuzzy Logic[2] may be relevant here.

[2]See http://plato.stanford.edu/entries/logic-fuzzy/.

Uncertainty

In most NL texts, one usually wants the computer to jump to conclusions that may be drawn from the information in the text with high likelihood even if not definitely. The computer would also need to be able to retract such conclusions if further information contradicts them. For example, when reading the short story: "John went to a restaurant. He had a steak.", one can conclude with high likelihood that John ordered the steak. This conclusion could be cancelled if the story continued with: "He didn't order it but stole it from the kitchen when no one noticed." Relevant work includes non-monotonic reasoning (Gabbay et al., 1994) and probabilistic reasoning (see (Russel and Norvig, 2003) for relevant references).

In exact NLU applications, however, most if not all conclusions relevant to the application can be inferred logically with complete certainty from the texts.

2.10.6 Language Use

The limitations mentioned in section 1.2.3 entail that for general NL texts, one cannot have a NL entailment relation that is based directly on the literal meaning of the text. Before one can get to calculating consequences, one first needs to reconstruct the intended meaning of the text from the literal meaning and the context. This is a very hard problem in general, but is probably manageable in exact NLU applications because they are designed in such a way that the intention of the text's author is made quite clear.

Part II

Semantic Composition and Packed Computation

Chapter 3

Semantic Composition

In this chapter, I discuss problems with the traditional approach for specifying the structural syntax-semantics interface, namely the one-to-one correspondence between syntactic and semantic structures. I then discuss an alternative approach, Glue Semantics, and provide a new introduction to this topic, which unlike existing introductions, does not rely on any knowledge of Lexical Functional Grammar (LFG), and postpones discussion of linear logic to the end rather than starting with it.

3.1 Background

3.1.1 The Syntax-Semantic Interface

The syntax-semantics interface is the relation between the syntactic structures of a sentence (and its parts) on the one hand and the meaning of the sentence (and its parts) on the other hand. The meaning can be represented by using expressions in a formal language, as was explained in the previous chapter.

This interface can be decomposed into two quite different kinds of relations. The *lexical* interface specifies the relation between the meaning of words on the one hand and their corresponding functor-argument constellations in the syntactic structure on the other hand. For example, this interface can specify that the verb "give", in active mode, has two syntactic constellations: one where the *theme* is related to the direct object and the *recipient* to the oblique[to] role (as in "John gave a book to Mary"), and the other where the *recipient* is related to the direct object and the *theme* to the indirect object (as in

"John gave Mary a book"). The interface specifies this particular set of constellations for "give" as a result of principles that relate the meaning of different classes of events and thematic roles to syntactic roles. (See e.g. (Levin and Hovav, 2005)).

The *sentential* interface specifies the relation between the meaning of a sentence, the meaning of its words, and the sentence's syntactic structure. For example, for the sentence (60)a with the syntactic structure (60)b, the interface can specify that the meaning of the sentence is the truth value *true* (in a model) when the meanings of the words (in that model) are the object **john**, the object **mary**, and the binary relation **saw**, respectively, such that $\langle \mathbf{john}, \mathbf{mary} \rangle \in \mathbf{saw}$.

(60) a. John saw Mary.

b. [$_{\text{S}}$ [$_{\text{NP}}$ John] [$_{\text{VP}}$ [$_{\text{V}}$ saw] [$_{\text{NP}}$ Mary]]]

From now on, we will focus only on the sentential syntax-semantics interface, and we will drop the qualifier *sentential*.

As explained in the previous chapter, the meaning of a sentence can be represented using expressions in a meaning representation language.

3.1.2 The Principle of Compositionality

The traditional approach to specifying the syntax-semantics interface, starting with Montague's work (Montague, 1974; Dowty et al., 1980) (see also e.g. (de Swart, 1998)) adheres to a very simple scheme called *the principle of strict compositionality* (or "direct compositionality" (Jacobson, 1999)). This principle consists of two parts:

1. Every constituent of the surface syntactic structure of a sentence is assigned a well-formed semantic denotation. A "well-formed denotation" (or: "healthy model-theoretic object" in the words of Jacobson (1999)) is a mathematical object that does not depend on any representational devices such as variables. The expression we write in the semantic representation language to name that mathematical object is just a convenience, and it must not contain any *unbound* variables.

2. If a syntactic constituent C was created by a syntactic rule R then the denotation of C is specified by a function f_R that takes as input only the denotations of C's immediate sub-constituents.

In short form, the principle says: "The meaning of a syntactic constituent is a function of the meaning of its immediate parts." The principle is usually implemented by relying on a third part:

3. The functions f_R are (almost) always simple lambda-application of the denotation of one sub-constituent to the denotations of its sibling(s).

This is essentially what we used in sections 2.4 and 2.7.3. For the sake of discussion and to illustrate the principle again, consider the following simple grammar:

(61) R_1: S $\rightarrow$ NP VP
R_2: NP $\rightarrow$ John, Mary, Bill, ...
R_3: VP $\rightarrow$ IV
R_4: VP $\rightarrow$ TV NP
R_5: IV $\rightarrow$ walked, slept, ...
R_6: TV $\rightarrow$ saw, liked, ...

We can assign meanings to the words as follows:

(62) $m(\text{John}) = john$
$m(\text{walked}) = \lambda x.walk(x)$
$m(\text{saw}) = \lambda y \lambda x.see(x, y)$

Here $john$ is a constant in the MRL which denotes a member of the domain of a model. The term $\lambda x.walk(x)$, or simply $walk$, denotes a unary predicate, and $\lambda y \lambda x.see(x, y)$, or simply see, denotes a binary predicate.[1]

Now, according to the principle of compositionality, the meaning of a sentence with parts NP and VP is a function of the meanings of these parts, and similarly with the other rules. The appropriate functions for the grammar here are as follows (we write f_i for f_{R_i}):

(63) $f_1(\alpha, \beta) = apply(\beta, \alpha)$
$f_2(\alpha) = \alpha$
$f_3(\alpha) = \alpha$
$f_4(\alpha, \beta) = apply(\alpha, \beta)$
$f_5(\alpha) = \alpha$
$f_6(\alpha) = \alpha$

[1] As is customary in the semantics literature, we often ignore tense and events when they are not crucial for the point being discussed. Thus, *see* rather than *saw*.

Notice that all of these are just the identity function or lambda-application. With these we can get:

(64) $m([_S\ [_{NP}\ \text{John}]\ [_{VP}\ [_V\ \text{saw}]\ [_{NP}\ \text{Mary}]]]) =_{f_1}$
$apply(m([_{VP}\ [_V\ \text{saw}]\ [_{NP}\ \text{Mary}]]), m([_{NP}\ \text{John}])) =_{f_4,f_2}$
$apply(apply(m([_V\ \text{saw}]), m([_{NP}\ \text{Mary}])), m(\text{John})) =_{f_6,f_2,(62)}$
$apply(apply(m(\text{saw}), m(\text{Mary})), john) =_{(62)}$
$apply(apply(\lambda y \lambda x.see(x,y), mary), john) =$
$apply(\lambda x.see(x, mary), john) =$
$see(john, mary)$

To remind, the last expression denotes (in a model M) a truth value: *true* in case that $\langle [\![john]\!]_M, [\![mary]\!]_M \rangle \in [\![like]\!]_M$, and *false* otherwise.

3.1.3 The Principle Meets Reality

The principle works well for simple examples such as the one above. However, it quickly runs into difficulties in slightly more complex cases. This is because a surface syntactic structure and a corresponding semantic representation do not neatly parallel each other. This was already mentioned in section 2.10.1, and we can see this issue again in Figure 3.1. The *past* and *not* operators there are far removed from each other in one possible semantic structure even though they are adjacent in the tree. Also, although "see every show" is a VP constituent where the verb C-commands the NP, it is actually "every show" that needs to take scope above the verb, with possibly additional intervening material. Finally, there may be more than one semantic structure for one surface syntactic structure due to scope ambiguities.

Despite these discrepancies between syntactic and semantic structures, the traditional approach insists on adhering to the principle of compositionality. This comes with a price that needs to be payed in exchange for bridging over the mismatch between the principle and the facts. The price is fleshed out in the form of (semantically justifiable but) sometimes quite complex type-shifting (or raising or lifting) operations that are designed to step around the mismatch. The common case is when there is a mismatch between the types of the meanings of two sibling constituents: neither can be applied on the other by simple lambda-application. In such a case, one might try raising the type of one of them so that it can apply on the other and produce an appropriate result.

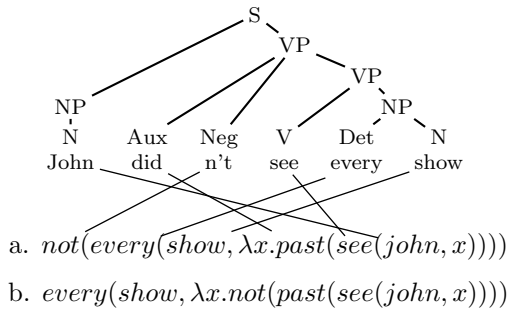


Figure 3.1: A syntax tree and corresponding semantic representations

The first example of this can be seen in Montague's approach for dealing with the following problem. In the sentence "John arrived", one would initially assume that the meaning of "John" is an individual $john$ of type e, and the meaning of "arrived" is a predicate $arrive$ of type $e \to t$, so the composition rule for a sentence says to apply the VP's meaning on the NP's meaning (we write $N{:}\gamma$ for a syntactic node N with meaning γ):

(65) S:$apply(\beta, \alpha)$ $\to$ NP:α VP:β

However, in the sentence "Everyone arrived", the NP is a generalized quantifier of type $(e \to t) \to t$, so it is the one who needs to take the VP's meaning as its argument:

(66) S:$apply(\alpha, \beta)$ $\to$ NP:α VP:β

But this is in conflict with (65).

Montague's solution is to always raise the type of expressions to the "worst possible case" in order to preserve the order of application. Thus, the meaning of the NP "John" is raised to the type $(e \to t) \to t$ as well, and it becomes:

(67) "John" : $\lambda P.P(john)$.

Now, instead of the intuitive application of $arrive : e \to t$ on $john : e$, we get the type-shifted version where $\lambda P.P(john) : (e \to t) \to t$ applies on $arrived : e \to t$. The result is $(\lambda P.P(john))(arrive)$, and it needs a further β-reduction to obtain the normal form $arrive(john)$.

The situation gets even worse if we consider transitive verbs. For example, in "Everyone saw someone", there are two quantifiers of type $(e \to t) \to t$, and one binary relation of

type $e \to e \to t$. Neither the verb nor the quantifiers can apply on the others. To deal with this problem, one proposal (see e.g. Emms (1992)) is to use a rule that raises the type of a quantifier, making it accept a binary relation rather than a property. Thus, the rule in (68)a would first apply on the object NP to raise its type so that it could take the binary relation and the subject as its arguments. But this gives only the object-narrow-scope reading. To get the object-wide-scope reading, another type-raising rule is needed for the object NP, shown in (68)b. If we have n quantifiers in the sentence, we will have to posit $n!$ type-raising rules to produce all the possible scopings of quantifiers.

(68) a. $Q : (e \to t) \to t$
$\Rightarrow \lambda R \lambda x.Q(\lambda y.R(x, y)) : (e \to e \to t) \to e \to t$
b. $Q : (e \to t) \to t$
$\Rightarrow \lambda R \lambda Q'.Q(\lambda y.Q'(\lambda x.R(x, y))) : (e \to e \to t) \to ((e \to t) \to t) \to t$

Another idea (see e.g. Hendriks (1993)) is to type-lift the verb rather than the quantifiers. To get the object-narrow-scope reading, the verb undergoes "argument raising" so that it can accept a quantifier as an argument, as in (69)a. To get the object-wide-scope reading, the predicate is type-lifted even more, to accept both quantifiers as arguments, as shown in (69)b, and so forth.

(69) a. $\lambda y \lambda x.R(x, y) : e \to e \to t$
$\Rightarrow \lambda Q \lambda x.Q(\lambda y.R(x, y)) : ((e \to t) \to t) \to e \to t$
b. $\lambda y \lambda x.R(x, y) : e \to e \to t$
$\Rightarrow \lambda Q \lambda Q'.Q(\lambda y.Q'(\lambda x.R(x, y))) : ((e \to t) \to t) \to ((e \to t) \to t) \to t$

But now, what happens in the following sentences?

(70) a. John and Mary like each other.
b. John and Mary introduced each other to Bill.

I will discuss reciprocals extensively in this dissertation in chapter 13. We will see that the desired semantic representations for these sentences are:

(71) $\textsc{recip}(john \oplus mary, \lambda x \lambda y.like(x, y))$
$\textsc{recip}(john \oplus mary, \lambda x \lambda y.introduce\text{-}to(x, y, bill))$

Here, RECIP is an operator that takes a plurality A and a binary predicate R, and asserts that for each pair of distinct elements from A, R holds on this pair.[2] Now it is the meaning of the NP "each other" in (70) that must take the two other elements – the plurality and the binary relation – as its arguments. None of the rules in (68) and (69) will work correctly because they expect the object NP to be a quantifier of type $(e \to t) \to t$. Furthermore, notice that in (70)b and (71)b, we must first "skip over" the reciprocal NP and combine *introduce-to* with *bill* in order to create the binary predicate $\lambda x \lambda y.\textit{introduce-to}(x, y, \textit{bill})$ which is the argument to RECIP.

We could try to rectify this problem by assuming more type-shifting rules. But this will not help in the long-run as new problems will keep popping up. For example, in (72)b-c, *each* floats to a different place from its logical place. A similar thing happens in (73).[3]

(72) a. Each of the children drew a picture.
b. The children each drew a picture.
c. The children drew a picture each.

(73) a. An occasional sailor walked by.
= Occasionally, a sailor walked by.
b. John drank a quiet cup of tea.
= Quietly, John drank a cup of tea.

In (73)a, the meaning of *occasional* wants to float to the sentence level, as the difference between (74)a and (74)b shows.

(74) a. A brave sailor walked by.
$a(\lambda x.\textit{brave}(x) \wedge \textit{sailor}(x), \lambda y.\textit{walk-by}(y))$
b. An occasional sailor walked by.
$\textit{occasionally}(a(\lambda x.\textit{sailor}(x), \lambda y.\textit{walk-by}(y)))$

In (74)a, it is reasonable to assign to the N' *brave sailor* the meaning $\lambda x.\textit{brave}(x) \wedge \textit{sailor}(x)$, but in (74)b there is no reasonable meaning to assign to the N'. If we insist on doing this anyway, we get a type-raised meaning for the N' that should take as its arguments the meanings of the determiner and the VP:

[2] More complex readings are possible (Dalrymple et al., 1998).
[3] This example is from (Barker, 2004a).

(75) $\lambda Q \lambda R.occasionally(Q(\lambda x.sailor(x), R))$.

Let us take a step back and think about what is going on here. It looks like the research paradigm of compositionality that was initially successful gradually encountered more and more difficulties, and there seems to be no end to the growth in complexity required to account for the observed phenomena. As time goes by and the paradigm is stretched to its limits, one feels increasingly confined by it. Inventing more and more type-shifting rules seems like dealing with the symptoms rather than the real cause of the problem. And the cause is that the principle of compositionality is just too naive. As we said above, the fact is that syntactic and semantic structures do not neatly parallel each other.

When a paradigm increasingly requires elevated complexity to deal with new cases, alternatives to long-held, obvious-seeming assumptions are eventually explored. The approach that I will present next frees us from the limitations of compositionality. In section 3.5.1, I will return to discuss this principle and get a deeper insight into its limitations.

3.2 Basics of Glue Semantics

As we saw, if we insist on adhering to the principle of compositionality, we must pay by complicating the specification of the rules of composition. There is a more flexible and powerful way for specifying the syntax-semantics interface, called Glue Semantics (GS) (Dalrymple, 2001; Asudeh et al., 2002; Dalrymple, 1999).[4] Most people are unfamiliar with GS, partly because the existing literature requires the reader to first learn about Lexical Functional Grammar (Bresnan, 2001) and linear logic[5] (which is the underlying logic of the framework as we shall see in section 3.4). However, the central issue of GS is not LFG or linear logic but rather the fact that GS is a convenient formalism for specifying the syntax-semantics interface. Therefore, one aim of this introduction is to present the important ideas of GS in an intuitive way while keeping technicalities to a minimum. To make this point clear, I do not discuss LFG until the next chapter, and linear logic will be mentioned only briefly in section 3.4 and then in more detail in chapter 5.

[4] An up-to-date bibliography of glue semantics literature can be found in http://www-csli.stanford.edu/~iddolev/glue_bibliography.html or through the LFG home page http://www.essex.ac.uk/linguistics/LFG/. Glue Semantics fits in the long tradition of work on Categorial Grammar, and even the term 'glue' itself comes from the CG tradition. For a bit more background history, see (van Benthem, 2003) and http://let.uvt.nl/general/people/rmuskens/ndttg/.

[5] See http://en.wikipedia.org/wiki/Linear_logic for various sources.

3.2.1 Word Meanings and Constraints on Combinations

I will first explain glue semantics by examples and then in abstract terms. Consider again the sentence "John saw Mary." The semantic expressions for the words in this sentence are *john* of type e, *mary* of type e, and *see* of type $e \to e \to t$. This already tells us that *john* cannot apply on *mary* because the former is not a functor from type e to something else; nor can it apply on *see*. However, nothing yet prevents us from applying the binary relation on the other two terms in the wrong order thus obtaining the wrong representation $see(mary, john)$. To prevent this undesirable result, we add constraints on how the semantic elements may combine. The most abstract way of doing so is to label each element in the type expressions, and to add equations on these labels:

(76)
$$\begin{array}{ll} john : e_2 & e_4 = e_2 \\ mary : e_3 & e_5 = e_3 \\ see : e_4 \to e_5 \to t_1 & \end{array}$$

As it turns out, it is the labels that are important rather than the type decorations. So from now on we will write this as follows:

(77)
$$\begin{array}{ll} john : \mathbf{2}^e & \mathbf{4}^e = \mathbf{2}^e \\ mary : \mathbf{3}^e & \mathbf{5}^e = \mathbf{3}^e \\ see : \mathbf{4}^e \to \mathbf{5}^e \to \mathbf{1}^t & \end{array}$$

An expression "$\psi : A$" is a "Glue Semantics statement," where ψ is an expression in the meaning representation language, and A is the corresponding type expression. Type expressions can function as a sort of "handle" on meaning expressions, and they allows us to place constraints on how the meaning expressions may combine. By combining the constraints and the statements in (77) we obtain:

(78)
$$\begin{array}{l} john : \mathbf{2}^e \\ mary : \mathbf{3}^e \\ see : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t \end{array}$$

Now these statements may combine only in one way by using the usual λ-application rule while taking the right-hand-side categories into account:

(79)
$$\frac{\psi : A \qquad \delta : A \to B}{\delta(\psi) : B}\ \text{APP}$$

This says: if δ is a functor of category $A \rightarrow B$, i.e. it expects an argument of category A and returns a result of category B, and if ψ is of category A, then you can apply δ on ψ to get $\delta(\psi)$ of category B. Using this rule on the elements in (78), we can get the correct result $see(john)(mary) : \mathbf{1}^t$ as shown in (80) (we can also write the result as $see(john, mary) : \mathbf{1}^t$). Thanks to the constraints, we cannot get the wrong result $see(mary, john) : \mathbf{1}^t$.

$$(80)\quad \dfrac{\dfrac{john : \mathbf{2}^e \qquad see : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t}{see(john) : \mathbf{3}^e \rightarrow \mathbf{1}^t}\,\text{APP} \qquad mary : \mathbf{3}^e}{see(john)(mary) : \mathbf{1}^t}\,\text{APP}$$

3.2.2 The Constraints Come from the Syntax

Of course, the crucial question is: Where do the constraints in (77) come from? The answer is that the constraints come from schemes in the lexicon which are instantiated in a particular sentence using labels from the sentence's syntactic structure.

There are many ways of doing this, depending on the particular syntactic theory chosen. The simplest way to demonstrate the idea is to consider the simplified syntactic structure of the sentence "John saw Mary" in Figure 3.2. Some of the nodes in the tree have been labelled. In the lexicon, we would like the GS statements to look like this:

(81)
proper name n	$n : \square^e$
transitive verb m	$m : \square^e \rightarrow \square^e \rightarrow \square^t$

In each box we want to have some instruction on how to find some label from the syntactic strucutre. The total effect of these instructions should be that in a particular sentence, the resulting glue statements would combine correctly with each other. We can do so with the following instructions:

(82)
proper name n	n : [label of my NP node]^e
transitive verb m	m : [label of the NP to the left of my S node]$^e \rightarrow$ [label of the NP to my right]$^e \rightarrow$ [label of my S node]^t

When these general lexical schemes are instantiated on the particular syntactic structure in Figure 3.2, we get the desired statements in (78).

The instructions in (82) are too restrictive. They should not be stated in terms of configurations in the syntax tree but rather in terms of more abstract roles, such as the

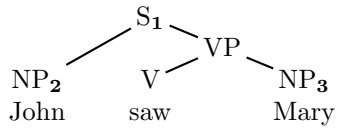


Figure 3.2: A simple syntax tree

syntactic subject and object of the sentence, or even in terms of thematic roles. Thus, the lexicon may specify (for verbs of a certain class which take the *agent* and *theme* roles):

(83) transitive verb m $\quad m$: [label of the subtree of my *agent* role]$^e \rightarrow$
[label of the subtree of my *theme* role]$^e \rightarrow$
[label of my S node]^t

The *lexical* syntax-semantics interface would then find these labels by knowing about the connection between the thematic roles *agent* and *theme* and the corresponding syntactic roles *subj* and *obj*, and the place of those latter roles in the syntactic structure.

3.2.3 An Abstract View

Now that we have seen an example, we can give an abstract description of glue semantics. The idea is to view the syntax-semantics interface (SSI) in more general terms than the classical approach. The SSI is seen as a ternary relation between the functor-argument structure of a sentence, the functor-argument structure of its words, and the sentence's syntactic structure. By "functor-argument structure" we mean the level of linguistic representation that specifies the number and types of arguments of a linguistic functor, as well as which functors apply on which arguments. A schematic diagram of this idea is shown in Figure 3.3.[6]

Every word may contribute one or more statements, each consisting of a meaning term and a glue category. The labels in the category are related to particular places in the syntactic structure. Syntactic rules and parts of syntactic structures may also contribute glue statements. The sharing of labels between glue categories is what constrains the possible ways in which meaning terms may combine, according to rules of combination – we have seen the APP rule in (79) above, and there will be more rules below.

Thus, glue semantics provides a more powerful and flexible scheme of combination than traditional approaches. Allowed combinations are restricted by the syntactic structures but

[6]This picture is only for illustration. Although the syntactic structure here is a tree, in theories like LFG and HPSG the syntactic structure would also include the feature structures. See chapter 4.

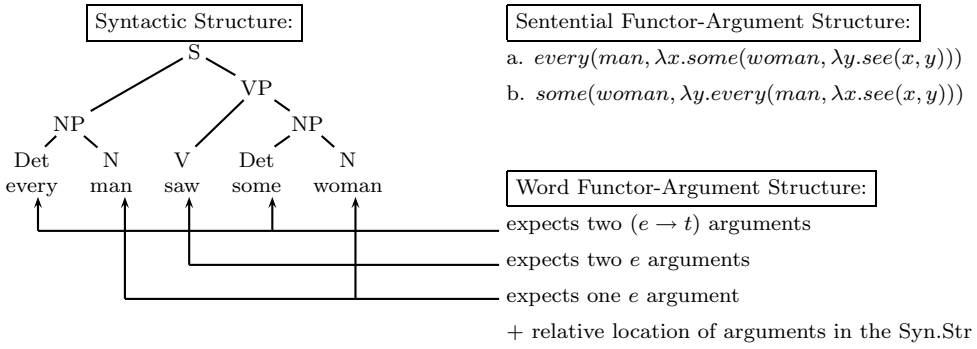


Figure 3.3: A schematic view of the sentential syntax-semantics interface

the order of their computation need not necessarily adhere to the hierarchical structure of the syntax. Thus, whereas it was crucial in (62) to define the meaning of "saw" as $\lambda y \lambda x.see(x, y)$ rather than $\lambda x \lambda y.see(x, y)$, because according to the syntactic structure, the meaning of the transitive verb must first combine with the meaning of the object NP and only then with that of the subject NP, in Glue Semantics that order is possible, but also the one shown in (80). This flexibility allows for a simpler specification of the syntax-semantics connection (this will become apparent when we consider more complex cases below).

The price for this power is that the scheme of combination is more complex – it does not rely on a simple lambda-application principle but on a more powerful logical proof system (which will be discussed in section 3.4). However, the price is worth paying because it needs to be paid only once, and the framework makes it easier for the linguist or grammar engineer to specify the syntax-semantics connection.

To sum up, instead of traditional compositionality, we adopt a new principle which we may express as follows:

> The meaning of a sentence is obtained from the meanings of its words while respecting the structure of these meanings and the syntactic constraints on how these meanings may combine.

3.2.4 Double Independence

An important feature of glue semantics is that it is an *abstract framework* for specifying the syntax-semantics interface. That means it does not depend on either a particular syntactic

theory or a particular semantic representation language.

On the syntax side, glue semantics specifications have been shown for LFG (Dalrymple, 2001), HPSG (Asudeh and Crouch, 2001) (and see also (Lev, 2005b)), LTAG (Frank and van Genabith, 2001), and Categorial Grammar (Asudeh et al., 2002). All those versions share the same glue semantic statements, the only difference between them is the instructions in lexical schema regarding how to find locations in the syntactic structures (i.e. how to fill the boxes such as those in (81). So as not to be bound by a specific syntactic theory, in this chapter and in some parts of this dissertation I will present glue semantics specifications for simplified syntactic structures. I will also discuss some issues in my implementation of glue semantics, which relies on the LFG grammar in the XLE system.

On the semantics side, glue semantics has been shown for a simple (Montague-like) semantic representation language (Dalrymple, 1999, 2001), for an extension which includes event variables (Fry, 1999a), and for DRT (van Genabith and Crouch, 1999; Kokkonidis, 2005). I will discuss all of these in this dissertation. I will start with the simple MRL, and show how it is possible to gradually extend it.

3.2.5 A Note about the Logic

We defined above the contribution of "saw" to be $see : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$. According to the APP rule, this expression must first combine with a value of category $\mathbf{2}^e$ and only then with a value of category $\mathbf{3}^e$. But what we really want is simply for the lexical entry of "saw" to say that the first argument comes from the sentence's subject and the second from the object, without constraining the order in which these arguments are used. The restricted order imposed by APP is merely a consequence of the fact that we write logical formulas linearly rather than in some parallel fashion.

One way to improve this is to switch to a notation that makes this parallelism explicit. But here we will stick with the linear notation and introduce a rule of variable exchange. In its simplest form, it looks like this:

$$(84)\quad \frac{\psi : A \rightarrow B \rightarrow C}{\lambda y \lambda x.\psi(x,y) : B \rightarrow A \rightarrow C}\ \text{EXCH}$$

More generally, we may exchange the order of more than two variables (also when they are nested in arguments).

The rule EXCH and the other rules introduced below are justifiable as theorems in linear logic which is the underling logic of glue semantics. In my opinion, people who are introduced to GS for the first time may find it easier to understand and use these rules rather than the more general logic. So I will present the rest of this chapter in this manner, and only in section 3.4 will I present the general logic and discuss it further.

We will also employ the familiar α-renaming, β-reduction, and η-reduction rules from the lambda-calculus. These operations preserve the meaning and do not change the right-hand-side category. From now on, we may freely apply these rules without mentioning them.

(85) $\dfrac{\lambda x.\psi : A}{\lambda y.\psi[y/x] : A}\ \alpha\text{-RENAMING}$ (y fresh)

(86) $\dfrac{(\lambda x.\psi)\varphi : A}{\psi[\varphi'/x] : A}\ \beta\text{-REDUCTION}$ (φ' is like φ except bound variables are possibly renamed to avoid accidental binding)

(87) $\dfrac{\lambda x.(\psi(x)) : A}{\psi : A}\ \eta\text{-REDUCTION}$

3.2.6 A Note about Old vs. New Glue Semantics

Older work in glue semantics ((Dalrymple, 1999) and before) was developed in the old version of GS, "old glue", whereas Dalrymple (2001) and newer work is presented in "new glue". The present paper uses new glue, but I mention old glue here so as to allow the reader to understand the older literature, as well as to make this point clear to people who are only familiar with old glue.

In old glue, the semantic material and the categories are mixed together in the glue semantics statements. Recasting our examples above in an old glue style, we would get:

(88) "John" $\mathbf{2}^e \rightsquigarrow john$
"Mary" $\mathbf{3}^e \rightsquigarrow mary$
"saw" $\forall X, Y.\ [\mathbf{2}^e \rightsquigarrow X \wedge \mathbf{3}^e \rightsquigarrow Y] \rightarrow \mathbf{1}^t \rightsquigarrow see(X, Y)$

In the old glue language, categories were directly connected to their meaning terms via the $\rightsquigarrow$ operator. The form above for "saw" says: if X is the meaning of $\mathbf{2}^e$ and Y is the meaning of $\mathbf{3}^e$ then $see(X, Y)$ is the meaning of $\mathbf{1}^t$. The conclusion from these three statements would be $\mathbf{1}^t \rightsquigarrow see(john, mary)$.

In chapter 7 of (Dalrymple, 1999), new glue was introduced, where the expressions in the meaning language are separated from the control mechanism (the right-hand-side

typed-category expressions) that specifies how the meanings may combine. This separation provides a cleaner theory and makes it easier to see what the semantic operators are and what the restrictions on their combination are. It is also easier then to see the connection to other semantic theories, especially Categorial Grammar (see section 3.5.2). The results developed in old glue can generally be translated to new glue (basically, everything on the right of the $\rightsquigarrow$ connectives should participate in the meaning term, and everything on the left should participate in the glue category).

3.3 Scope Flexibility

3.3.1 Basic Scope Ambiguity

A central phenomenon where semantic structure does not parallel syntactic structure is the ability of quantifiers and other operators to take scope over the material in which they are embedded. The familiar example is (89), and the two possible semantic structures are shown in (90). The syntactic parts of the sentence (89) are labelled (just as they were in Figure 3.2). The glue statements in (90) indicate on their right-hand-side that the meaning expressions correspond to the entire sentence. The basic type expressions of the semantic entities are as expected, and an initial assignment of labels to them is shown in (91).

(89) [[Every man]$_2$ saw [some woman]$_3$]$_1$.

(90) $every(man, \lambda x.some(woman, \lambda y.see(x, y)) : \mathbf{1}^t$
$some(woman, \lambda y.every(man, \lambda x.see(x, y)) : \mathbf{1}^t$

(91) $man : \mathbf{4}^e \rightarrow \mathbf{5}^t$ $\quad$ $every : (\mathbf{4}^e \rightarrow \mathbf{5}^t) \rightarrow (\mathbf{2}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$
$woman : \mathbf{6}^e \rightarrow \mathbf{7}^t$ $\quad$ $some : (\mathbf{6}^e \rightarrow \mathbf{7}^t) \rightarrow (\mathbf{3}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$
$see : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$

How may the statements in (91) combine? Obviously, *man* must combine with *every*, and *woman* with *some*, using the APP rule, to yield:

(92) $every(man) : (\mathbf{2}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$
$some(woman) : (\mathbf{3}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$

(Here, $every(man)$ means $\lambda S.every(man, S)$, and similarly with the other expression.) Using the familiar rule of *function composition*:

$$\dfrac{\dfrac{see : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t \qquad some(woman) : (\mathbf{3}^e \to \mathbf{1}^t) \to \mathbf{1}^t}{\lambda x.some(woman, \lambda y.see(x,y)) : \mathbf{2}^e \to \mathbf{1}^t}\,\text{COMP} \qquad every(man)}{every(man, \lambda x.some(woman, \lambda y.see(x,y))) : \mathbf{1}^t}\,\text{APP}$$

$$\dfrac{\dfrac{\dfrac{see : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t}{\lambda y \lambda x.see(x,y) : \mathbf{3}^e \to \mathbf{2}^e \to \mathbf{1}^t}\,\text{EXCH} \qquad every(man) : (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{1}^t}{\lambda y.every(man, \lambda x.see(x,y)) : \mathbf{3}^e \to \mathbf{1}^t}\,\text{COMP} \qquad some(woman)}{some(woman, \lambda y.every(man, \lambda x.see(x,y))) : \mathbf{1}^t}\,\text{APP}$$

Figure 3.4: Two derivations for "Every man saw some woman."

(93) $$\dfrac{\delta : A \to B \qquad \gamma : B \to C}{\lambda x.\gamma(\delta(x)) : A \to C}\,\text{COMP}$$

Together with APP and EXCH, there are exactly two results of category $\mathbf{1}^t$ that can be obtained by using each statement of (91) exactly once. This is shown in Figure 3.4.

The benefit here is that we have one simple statement for each semantic element in (91). This is all we need to state, and the combination apparatus takes care of the rest, i.e. to combine them appropriately. We do not need to use complicated type-shifting rules.

3.3.2 Nested Scope Ambiguity

How are the lexicon instructions defined for nouns and quantifiers? As we shall see below, to account for scope ambiguities of quantifiers around relational nouns of type $e \to e \to t$, we need direct access to both e parts of that type expression. Therefore, we also need access to the e part of the $e \to t$ type expression of a regular noun. However, this e part does not naturally correspond to any part of the syntactic structure. In such cases, we may introduce sub-labels which exist only in the glue statements but not the syntactic structure. Thus, the lexical schema for a noun is:

(94) a noun with meaning m:
$m : {l_v}^e \to {l_r}^t$ where l is the label of my NP

where l_v and l_r are sub-labels of l, distinct from l. This definition follows glue semantics tradition, where the 'v' stands for 'variable' and the 'r' stands for 'restrictor' (or 'result'), since the meaning of the noun (plus its modifiers) serves as the restrictor of a quantifier.[7]

[7] We could instead assign a label k to the N', separate from the NP's label, and use ${k_v}^e$ and ${k_r}^t$. However, in LFG, both N' and NP nodes in the syntactic tree are mapped to the same F-structure node – see chapter 4. So we will stick with glue semantics tradition here.

Thus, in (91), we would change $\mathbf{4}^e \rightarrow \mathbf{5}^t$ to $\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$, and $\mathbf{6}^e \rightarrow \mathbf{7}^t$ to $\mathbf{3}_v^e \rightarrow \mathbf{3}_r^t$.

The schema for a quantifier is:

(95) a quantifier with meaning Q:
$Q : ({l_v}^e \rightarrow {l_r}^t) \rightarrow (l^e \rightarrow H^t) \rightarrow H^t$ where l is the label of my NP

The first part ${l_v}^e \rightarrow l^t$ of this statement expects the meaning of the noun (plus its modifiers). The second part has the shape of a NP generalized quantifier of type $(e \rightarrow t) \rightarrow t$ (as in (92)) which says: if there is a predicate that needs me (of type $l^e \rightarrow H^t$), I can take *it* as my argument and return the result H^t instead of it.

The reason we use a variable H^t rather than an instruction for a specific label is that quantifiers may land on more than one label, as in the case of a nested scope ambiguity shown in (96)-(97). Here, "mayor" is a relative noun that requires an oblique[of] argument, and the scope of *a* may be either the entire sentence or the predicate *mayor*. Thus we need the variable H^t whereas an instruction such as "label of the lowest sentence I appear in" would be too restrictive.[8] (In (97), we abbreviate the VP "indicted for accepting bribes" as *indicted*, and we omit the adjective *big* – both are not important for the issue of scope ambiguity here).

(96) [[Every mayor of [a big city]$_3$]$_2$ was indicted for accepting bribes]$_1$.

(97) $every(\lambda x.a(city, \lambda y.mayor\text{-}of(x,y)), \lambda x.indicted(x)) : \mathbf{1}^t$
$a(city, \lambda y.every(\lambda x.mayor\text{-}of(x,y), \lambda x.indicted(x))) : \mathbf{1}^t$

(98)

[indicted]	$indicted : \mathbf{2}^e \rightarrow \mathbf{1}^t$
[every]	$every : (\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t) \rightarrow (\mathbf{2}^e \rightarrow H^t) \rightarrow H^t$
[mayor-of]	$mayor\text{-}of : \mathbf{2}_v^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{2}_r^t$
[city]	$city : \mathbf{3}_v^e \rightarrow \mathbf{3}_r^t$
[a]	$a : (\mathbf{3}_v^e \rightarrow \mathbf{3}_r^t) \rightarrow (\mathbf{3}^e \rightarrow G^t) \rightarrow G^t$

The new element here is the relational noun "mayor". Its meaning is the binary relation *mayor-of* of type $e \rightarrow e \rightarrow t$. Its category is almost like the category of a simple noun except that it expects an additional argument whose category comes from the argument NP "a city". The general lexicon instruction for a relative noun with meaning m:

[8] But see the end of this section for a revised proposal.

(99) $m : l_v{}^e \to k \to l_v{}^t$ where l is the label of my NP
and k is the label of my oblique[of] argument

There are only two ways in which the five statements of (98) may each be used once and together combine to produce a result of the desired category $\mathbf{1}^t$. Both ways combine **[a]** and **[city]** to get:

(100) **[a-city]** $a(city) : (\mathbf{3}^e \to G^t) \to G^t$

If we instantiate $G^t = \mathbf{2}_r{}^t$, we can only get the city-narrow-scope result:

$$\dfrac{\textbf{[indicted]} \quad \dfrac{\dfrac{\textbf{[mayor-of]} \quad \textbf{[a-city]}(G^t = \mathbf{2}_r{}^t)}{\textbf{[mayor-of-a-city]}}\,\text{COMP} \qquad \textbf{[every]}(H^t = \mathbf{1}^t)}{\textbf{[every-mayor-of-a-city]}}\,\text{APP}}{every(\lambda x.a(city, \lambda y.mayor\text{-}of(x, y)), indicted) : \mathbf{1}^t}\,\text{APP}$$

Here we used the abbreviations:

[mayor-of-a-city]	$\lambda x.a(city, \lambda y.mayor\text{-}of(x, y)) : \mathbf{2}_v{}^e \to \mathbf{2}_r{}^t$
[every-mayor-of-a-city]	$every(\lambda x.a(city, \lambda y.mayor\text{-}of(x, y))) : (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{1}^t$

If instead we instantiate $G^t = \mathbf{1}^t$, we can get only the city-wide-scope result:

$$\dfrac{\dfrac{\textbf{[indicted]} \quad \dfrac{\dfrac{\textbf{[mayor-of]}}{\textbf{[mayor-of]}'}\,\text{EXCH} \qquad \textbf{[every]}(H^t = \mathbf{1}^t)}{\textbf{[every-mayor-of]}}\,\text{COMP}}{\textbf{[every-mayor-of]}'}\,\text{EXCH}}{\textbf{[every-mayor-of-indicted]}}\,\text{APP} \qquad \textbf{[a-city]}(G^t = \mathbf{1}^t)}{a(city, \lambda y.every(\lambda x.mayor\text{-}of(x, y), indicted))}\,\text{APP}$$

Here we used the abbreviations:

[mayor-of]′	$\lambda y \lambda x.mayor\text{-}of(x, y) : \mathbf{3}^e \to \mathbf{2}_v{}^e \to \mathbf{2}_r{}^t$
[every-mayor-of]	$\lambda y \lambda S.every(\lambda x.mayor\text{-}of(x, y), S) : \mathbf{3}^e \to (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{1}^t$
[every-mayor-of]′	$\lambda S \lambda y.every(\lambda x.mayor\text{-}of(x, y), S) : (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{3}^e \to \mathbf{1}^t$
[every-mayor-of-indicted]	$\lambda y.every(\lambda x.mayor\text{-}of(x, y), indicted) : \mathbf{3}^e \to \mathbf{1}^t$

Notice the flexibility of the approach: **[every-mayor-of-indicted]** provides the semantics for the entire sentence (96) except for the deeply-embedded NP "a city", which remains as the "hole" $\mathbf{3}^e$.

A note about variable categories: Why did we not explore the option of instantiating H^t as $\mathbf{2}_r{}^t$ or $\mathbf{3}_r{}^t$ or instantiating G^t as $\mathbf{3}_r{}^t$? The reader is invited to verify that under such instantiations there would be no way in which all the statements could combine into one

complete derivation. We will be able to prove this precisely once we discuss the underlying logic of glue semantics in section 3.4.

It seems a good idea, therefore, to not use variable categories and instead use normal category descriptions whose specification may identify more than one possible label. In fact, not any possible label of type t can be a legal value for the scope category of a quantifier, but only the labels of clauses that contain the quantifier. So we can change (95) to:

(101) $m : ({l_v}^e \rightarrow {l_r}^t) \rightarrow (l^e \rightarrow h^t) \rightarrow h^t$
where l is the label of my NP
and h is the label of my scope clause s such that l is subordinated to s

This proposal deviates from the tradition in the glue semantics literature of using variable categories in the analysis of quantifiers. However, the glue semantics literature that handles anaphora resolution by glue derivations uses a similar device to the one I propose here to deal with antecedent ambiguities. We will see this in section 8.2. That the two kinds of ambiguity can be treated in the same fashion has not been previously recognized in the literature. In fact, Kokkonidis (2006) has a long discussion explaining why variable categories should be seen as first order predicates with a variable index rather than as second order typed variables (essentially, he advocates using $e(\mathbf{2})$, $t(\mathbf{1})$, and $t(h)$ rather than $\mathbf{2}^e$, $\mathbf{1}^t$, and H^t). But his point becomes unnecessary if variable categories are not used, as I propose here. We will see in section 4.5.3 how this idea is implemented in a glue specification.

3.3.3 Other Scope-Taking Operators

Negation and modals also participate in scope ambiguities:

(102) [[John]$_2$ didn't see [every show]$_3$]$_1$
$not(every(show, \lambda x.see(john, x)))$
$every(show, \lambda x.not(see(john, x)))$

(103) a. You may not smoke. $not(may(smoke(you)))$
b. You may not succeed. $may(not(succeed(you)))$

Therefore, we let the lexical entry of negation or a modal m be:

(104) $m : l^t \rightarrow l^t$ where l is the label of my clause

For example, in the sentence (102), we have:

(105) $see : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$ $\quad not : \mathbf{1}^t \rightarrow \mathbf{1}^t$
$john : \mathbf{2}^e$ $\quad every(show) : (\mathbf{3}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$

The first reading of (102) is obtained by first combining the elements except for the negation to form $every(show, \lambda x.see(john, x)) : \mathbf{1}^t$ and then applying *not* on it. The second reading is obtained by first combining *not* with *see* to get $\lambda x \lambda y.not(see(x, y)) : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$, and then combining the rest. In contrast to quantifiers that may float outside the clause they appear in, as was shown in section 3.3.2, negation and modals do not have this ability, and therefore their lexical entry (104) has instructions for a particular label rather than having a variable as in (95) or an indeterminate specification as in (101).

We can also easily write a glue specification for floating adjectives, like those we saw in (73). We do not need to insist that the N' has a meaning that is obtained from both *occasional* and *sailor*. Simply, for a floating adjective m, we have the lexical entry:

(106) $m{+}ly : l^t \rightarrow l^t$ where l is the label of my scope clause

Thus, we get the following:

(107) [[An occasional sailor]$_2$ walked by]$_1$

$walk\text{-}by : \mathbf{2}^e \rightarrow \mathbf{1}^t$ $\quad sailor : \mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$
$occasionally : \mathbf{1}^t \rightarrow \mathbf{1}^t$ $\quad a : (\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t) \rightarrow (\mathbf{2}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$
Result $\Rightarrow$ $occasionally(a(sailor, walk\text{-}by)) : \mathbf{1}^t$

3.3.4 Scoping Constraints

The lexical entries presented here are very flexible, but they overgenerate. While quantifiers may escape their clause in NL, they may not escape it too far away, and they may not escape scope islands such as relative clauses and some cases of coordination. Moreover, even for those cases that are possible, some are more plausible than others.

This situation is similar to syntactic ambiguity. For those ambiguities, one assumes that the grammar can generate all possibilities while an additional module selects the most probable syntactic structure in a given context (e.g. based on general statistical preferences in language as well as based on additional semantic and other world knowledge). All the

exponentially many syntactic parses can be efficiently represented in a compact chart forest, and the selection module can efficiently extract the one or few most probable parses from that chart.

Similarly with scope ambiguities, we assume that the syntax-semantics interface can generate all the possibilities. Additional hard constraints and soft preferences on scope may be expressed by adding a "scope label" to each scope-taking operator (cf. (Crouch and van Genabith, 1999)). For example, we may label "not" with s_1 and "every" with s_2 in (105). To indicate that the narrow-scope of "every" is the preferred reading, or merely to select that reading from the possible ones, we can write: $s_2 \succ s_1$. One can then use this statement to constrain glue semantics derivations by requiring them to use s_1 before using s_2. More details will be given in sections 4.6.2 and 6.7.2.

3.4 Logic and Computability

3.4.1 Underlying Logic

Instead of introducing explicitly the rules we saw above (EXCH and COMP), we could use a general implicational logic, and derive those rules as theorems in the logic. In addition to the APP rule, the implicational logic has the abstraction rule:

(108)
$$\frac{\begin{array}{c}[x : A]_i \\ \vdots \\ \psi : B\end{array}}{\lambda x.\psi : A \rightarrow B}\ \text{ABS}_i$$

This rule says: if you can prove $\psi : B$ based on the assumption $[x : A]_i$ then you can discharge the assumption and conclude $\lambda x.\psi : A \rightarrow B$. The intuitive explanation is that if we can calculate a ψ of category B given an arbitrary x of category A, then we know that $\lambda x.\psi$ is a function from A to B. Using this rule, we can justify the special rules we used before. The proof for EXCH is shown in (109), and the proof for COMP is shown in (110). These proofs can be generalized for extended versions of the rules (exchanging the arguments of n-ary predicates, also if they are nested, and composing larger predicates, or more predicates).

$$\dfrac{\dfrac{\dfrac{[y:B]_2 \quad \dfrac{[x:A]_1 \quad \psi : A \to B \to C}{\psi(x) : B \to C}\,\text{APP}}{\psi(x,y) : C}\,\text{APP}}{\lambda x.\psi(x,y) : A \to C}\,\text{ABS}_1}{\lambda y \lambda x.\psi(x,y) : B \to A \to C}\,\text{ABS}_2 \tag{109}$$

$$\dfrac{\dfrac{[x:A]_1 \quad \delta : A \to B}{\delta(x) : B}\,\text{APP} \qquad \gamma : B \to C}{\dfrac{\gamma(\delta(x)) : C}{\lambda x.\gamma(\delta(x)) : A \to C}\,\text{ABS}_1}\,\text{APP} \tag{110}$$

Why use special rules rather than prove everything in the underlying logic? The answer is that we can distinguish between a "user friendly" version of the logic and an "implementation level" version. I think that in some cases, the user friendly version with the special rules is easier for doing derivations by hand than the implementation level, because it keeps all the semantic expressions in closed form, i.e. without free variables and without needing to use the abstraction rule as in the implementation level. For example, if we are only allowed to use APP and ABS, the second proof of Figure 3.4 becomes the one shown in (111).

$$\dfrac{\dfrac{\dfrac{\dfrac{[y:\mathbf{3}^e]_2 \quad \dfrac{[x:\mathbf{2}^e]_1 \quad see : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t}{see(x) : \mathbf{3}^e \to \mathbf{1}^t}\,\text{APP}}{see(x,y) : \mathbf{1}^t}\,\text{APP}}{\lambda x.see(x,y) : \mathbf{2}^e \to \mathbf{1}^t}\,\text{ABS}_1 \qquad every(man)}{every(man, \lambda x.see(x,y)) : \mathbf{1}^t}\,\text{APP}}{\lambda y.every(man, \lambda x.see(x,y)) : \mathbf{3}^e \to \mathbf{1}^t}\,\text{ABS}_2 \qquad some(woman)}{some(woman, \lambda y.every(man, \lambda x.see(x,y))) : \mathbf{1}^t}\,\text{APP} \tag{111}$$

The rules EXCH, COMP, etc. also provide a specific repertoire of combinators that are actually useful in practice for linguistic phenomena whereas the general logic may not provide such guidance.[9] However, for a computer program that automatically finds proofs in the logic, it might be more efficient to use the implementation level, and it might be easier to implement, so I will use it in the algorithms of chapters 5-7.

3.4.2 Resource Sensitivity

The underlying logic we use is the implicational fragment of linear logic. The difference between this logic and the implicational fragment of classical logic is that each assumption

[9]This idea of restricting the logical power is similar to the difference between Type-Logical Categorial Grammar (e.g. Carpenter (1998)) and Combinatory Categorial Grammar (e.g. Steedman (2000)). While the former is based on a general logical system, the latter uses only certain combinators that are actually useful for natural language, but all of whom are justifiable as lemmas in the more powerful logic.

must be used exactly once in a valid derivation. Thus we have:

(112) $A,\ A \to B \vdash B$ whereas $C,\ A,\ A \to B \nvdash B$ (not even when $C = A$)
$A,\ A \to (A \to B) \nvdash B$

This is desirable because we usually want to use the contribution of each word at least once (not ignore words) and not more than once (not re-use the semantics of a word). For example, if we didn't use resource sensitivity, we could prove $every(man, laughed)$ from the premises in (113) while ignoring the contribution of "tall", and we could also use this contribution more than once to prove $every(tall(tall(man)), laughed)$.

(113) [[Every tall man]$_2$ laughed]$_1$
$laughed : \mathbf{2}^e \to \mathbf{1}^t$ $tall : (\mathbf{2}_v^e \to \mathbf{2}_r^t) \to \mathbf{2}_v^e \to \mathbf{2}_r^t$
$every : (\mathbf{2}_v^e \to \mathbf{2}_r^t) \to (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{1}^t$ $man : \mathbf{2}_v^e \to \mathbf{2}_r^t$

Note that the abstraction rule in (108) is marked with the index of the assumption. In contrast to classical logic, each assumption must be marked with a *unique* index. So while the following proof is valid in classical logic, it is *not* in linear logic:

(114)
$$\dfrac{[x : A]_1 \qquad \dfrac{[x : A]_1 \quad \psi : A \to (A \to B)}{\psi(x) : A \to B}\ \text{APP}}{\dfrac{\psi(x, x) : B}{\lambda x.\psi(x, x) : A \to B}\ \text{ABS}_1}\ \text{APP}$$

In this sense, linear logic is less like a familiar reasoning system and more like a calculus for combining pieces of a jigsaw puzzle, except there may be more than one way to combine all the pieces.

Some cases have been explored in the glue semantics literature where it seems we might want to deviate from using each word contribution exactly once. One case is doing anaphora resolution within glue derivations. Since a discourse element may be referred to later by more than one anaphoric expression, it seems we want to use the contribution of the element more than once (see section 8.2). This is a case of *resource deficit*. The dual case of *resource surplus* might arise in resumptive pronouns (Asudeh, 2004). However, these minor exceptions can be handled by adding GS statements that express this exception explicitly. For more on this issue, see "Issues in Resource Management" in Asudeh et al. (2002).

From a computational perspective, resource sensitivity has nice properties. When the computer searches for a proof from a given set of glue premises, it knows it found one when each premise was used exactly once. We will see how this works out in chapter 5.

3.5 Comparison to Other Approaches

3.5.1 The Traditional Approach, Again

As we said regarding Figure 3.1, a surface syntactic structure and a corresponding semantic representation do not neatly parallel each other. As we have seen, this poses no problem for the constraint-based glue semantics approach. We can now gain a deeper understanding of the problems that this mismatch causes in the traditional approach that was discussed in section 3.1.

The type-shifting rules in section 3.1.3 are more complicated than the straightforward glue semantics derivations we saw above. The complications arise for the following reasons.

Adhering to a Fixed Order of Combination: First, the classical approach adheres to the idea that elements in a syntactic construction must be combined in a fixed order – either always VP on NP or vice versa. Recall the uniform rule (66), repeated here:

(115) S:$apply(\alpha, \beta)$ $\rightarrow$ NP:α VP:β

This is needed because an NP might be a generalized quantifier that takes the VP's meaning as its argument:

(116) $$\frac{everyone : (e \rightarrow t) \rightarrow t \quad left : e \rightarrow t}{everyone(left) : t}\ \text{APP}$$

However, for this to work with the subject NP "John", its meaning $john : e$ must first be type-raised to $\lambda P.P(john) : (e \rightarrow t) \rightarrow t$ before it can apply on the meaning of the VP of type $e \rightarrow t$ (as we saw in (67)). Then a further β-conversion is required to get the final semantic form:

(117) $$\frac{\dfrac{\dfrac{john : e}{\lambda P.P(john) : (e \rightarrow t) \rightarrow t}\ \text{TYPE-LIFT} \qquad left : e \rightarrow t}{(\lambda P.P(john))(left) : t}\ \text{APP}}{left(john) : t}\ \beta\text{-REDUCTION}$$

In contrast, in glue semantics there is no a-priori requirement to combine elements in a certain order, and any two elements are allowed to combine as long as they can do so according to their category expressions:

(118) $$\frac{john : \mathbf{2}^e \quad left : \mathbf{2}^e \rightarrow \mathbf{1}^t}{left(john) : \mathbf{1}^t}\ \text{APP}$$

$$(119)\quad \frac{everyone : (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{1}^t \quad left : \mathbf{2}^e \to \mathbf{1}^t}{everyone(left) : \mathbf{1}^t}\ \text{APP}$$

Still, if one really wanted, for some reason, to derive the meaning of a sentence by following the particular hierarchy of the syntactic structure, one could do that too in glue semantics. In fact, the type-raising rule which changes an individual of type e to a generalized quantifier of type $(e \to t) \to t$ is a theorem of linear logic:

$$(120)\quad \frac{\dfrac{john : \mathbf{1}^e \quad [P : \mathbf{1}^e \to \mathbf{2}^t]_1}{P(john) : \mathbf{2}^t}\ \text{APP}}{\lambda P.P(john) : (\mathbf{1}^e \to \mathbf{2}^t) \to \mathbf{2}^t}\ \text{ABS}_1$$

This can now be used to combine with "left", as was done in (117):

$$(121)\quad \frac{\dfrac{\dfrac{john : \mathbf{1}^e \quad [P : \mathbf{1}^e \to \mathbf{2}^t]_1}{P(john) : \mathbf{2}^t}\ \text{APP}}{\lambda P.P(john) : (\mathbf{1}^e \to \mathbf{2}^t) \to \mathbf{2}^t}\ \text{ABS}_1 \qquad left : \mathbf{2}^e \to \mathbf{1}^t}{(\lambda P.P(john))(left) : \mathbf{1}^t}\ \text{APP}$$

Now, the final term $(\lambda P.P(john))(laughed)$ must still undergo another β-conversion to obtain the normal form $left(john)$.

As we shall see in chapter 5, this β-conversion corresponds to normalizing linear logic proofs to the simplest form without detours. Here, the detour in (121) would be normalized to get the proof in (118). From both a theoretical and a practical point of view, allowing (118) directly is preferable to always *requiring* (121). Since GS is not *forced* to adhere to a fixed order of application based on constituent structure, we get a more simple and elegant account.

Another instance of this point was mentioned at the end of section 3.1 in (Francez and Pratt, 1997), where the meaning representation is obtained by combining the verb's meaning with various verb arguments as well as temporal and aspectual modifiers. If glue semantics is used then it does not matter whether one uses $\lambda x\lambda y\lambda z.F(x,y,z) : A \to B \to C \to D$ or $\lambda y\lambda z\lambda x.F(x,y,z) : B \to C \to A \to D$ or any other permutation because the underlying linear logic apparatus takes care to combine these correctly with their arguments, and the results will be exactly the same (after $\beta\eta$-reduction). In contrast, in the traditional scheme of the syntax-semantics interface, one has to choose a particular order for the lambda variables. This imposes an order of functional application, which occasionally has to be overruled. Since Pratt and Francez (2001) used the traditional scheme, they had to introduce to the λ-calculus certain non-standard operations in order to overcome this inflexibility (these operations are similar to the "extended functional composition" operator we will see in (399)).

Adhering to One Rule of Combination: Second, the classical approach adheres to a very simple scheme of combination: The meaning of a syntactic constituent is calculated by using only lambda-application to combine the meanings of the constituent's immediate sub-parts. Forcing simplicity here necessarily makes the underlying complexity of NL pop up somewhere else: the meaning elements themselves become more complicated as a result of type-shifting rules.

In contrast, glue semantics adheres to the basic meaning-types of words (e.g. an individual is of type e and not $(e \to t) \to t$, and a binary relation is of type $e \to e \to t$ and not a type-raised version of it). This is made possible by using a more powerful scheme of combination: either a repertoire of additional combination rules such as EXCH and COMP (in the "user friendly" level), or a powerful logical proof system (in the "implementation" level). The price for this more powerful scheme of combination is worth paying because it needs to be paid only once, and the complexities are shifted from the meanings and meaning entries to the scheme of combination. This makes it easier for the linguist or grammar engineer to specify the syntax-semantics connection.

Adhering to a Denotation for Each Constituent: Third, the classical approach insists on providing a denotation for each syntactic phrase. Yet upon further reflection, this does not really make much sense. For one, if

(122) $\lambda Q.some(person, \lambda x.Q(\lambda y.saw(x, y)))$

is one meaning of the VP "saw someone" (where "someone" is to take scope over the sentence's subject quantifier), then this meaning has a big hole *in its middle*, namely Q. Therefore, this meaning of the VP is a function of the meaning of its parts only in a very uninteresting sense that owes its merit to a technical trick. (Also, due to scope ambiguities, we might need to assume $n!$ different meanings for a VP in a sentence with n quantifiers). We saw another good example of this point in (73)-(75), where the N' "occasional sailor" did not have a nice intuitive denotation.

Here is an additional demonstration of why (122) is a "technical trick". Because "someone" appears inside the VP with "saw", the traditional approach insists on "directly" combining their meanings. Since this does not make much sense when "someone" takes scope outside the VP, forcing this "direct" combination yields the strange "meaning" (122).

Now, there is nothing special about the order of combination that is imposed by the surface syntactic structure – we could just as well insist on "directly" combining the meanings of *any* two words ψ and ϕ regardless of where they appear in the sentence. It is always possible to combine their meanings "directly" by type-raising one of them using the operator $\lambda X \lambda Y \lambda R.R(X,Y)$. Applying this operator on ψ gives its "type-raised" version $\delta = \lambda Y \lambda R.R(\psi, Y)$. If the sentence's meaning is τ, we can abstract away ψ and ϕ from τ to get σ. Then, τ can be obtained by applying δ on ϕ and on σ. For example, here is how we can "directly" combine the meanings of "John" and "woman" or of "John" and "every" in the following sentence:

(123) John saw every woman.

$\tau = every(woman, \lambda z.saw(john, z))$

composition #1:	composition #2:
$\psi = john$	$\psi = john$
$\phi = woman$	$\phi = every$
$\sigma = \lambda X \lambda Y.every(Y, \lambda z.saw(X, z))$	$\sigma = \lambda X \lambda Y.Y(woman, \lambda z.saw(X, z))$
$\delta = \lambda Y \lambda R.R(john, Y)$	$\delta = \lambda Y.\lambda R.R(john, Y)$
$\tau = \delta(\phi)(\sigma)$	$\tau = \delta(\phi)(\sigma)$

In composition #1:

$\tau = \delta(\phi)(\sigma) = (\lambda Y \lambda R.R(john, Y))(woman)(\lambda X \lambda Y.every(Y, \lambda z.saw(X, z))) =$
$(\lambda R.R(john, woman))(\lambda X \lambda Y.every(Y, \lambda z.saw(X, z))) =$
$(\lambda X \lambda Y.every(Y, \lambda z.saw(X, z)))(john, woman) =$
$every(woman, \lambda z.saw(john, z))$

In composition #2:

$\tau = \delta(\phi)(\sigma) = (\lambda Y \lambda R.R(john, Y))(every)(\lambda X \lambda Y.Y(woman, \lambda z.saw(X, z))) =$
$(\lambda R.R(john, every))(\lambda X \lambda Y.Y(woman, \lambda z.saw(X, z))) =$
$(\lambda X \lambda Y.Y(woman, \lambda z.saw(X, z)))(john, every) =$
$every(woman, \lambda z.saw(john, z))$

As this example shows, the order of derivation imposed by the syntactic surface structure is just one of many possible orders of derivation, and the question is why is it privileged? Glue semantics does not see it as privileged. Instead, GS follows modern constraint-based grammatical formalisms (such as HPSG and LFG) by letting each component of the grammar incrementally add constraints to describe a linguistic construction. Although

the grammar's components are interconnected, there is no need to adhere to an unnatural one-to-one correspondence between them, nor a need for a directional or derivational specification of how the structures are obtained. Therefore, in glue semantics we do not need to use non-intuitive meanings like (122) and (75).

Another way to see this point is to notice that glue semantics is modular in the sense that it separates the meaning expressions from the constraints on their combinations (i.e. the left-hand-side vs. the right-hand-side of glue statements), and it does not directly rely on the hierarchical structure of the syntax to guide combination. In contrast, expressions such as (69) are extra-complex because they interleave those two aspects together in one formula.

As a final point of discussion, one potential objection to glue semantics is that the description of some linguistic phenomena could benefit from the more restrictive principle of compositionality. For example, some theories attempt to show that certain predictions about possible scope ambiguities or anaphoric bindings follow from a strict compositionality approach. In my opinion, however, linguistic theories as well as their computational implementation should be kept modular because humans need to understand, debug, maintain, and extend them. Scope and anaphoric predictions should be made by independent modules rather than as side effects in one big complicated module, or else that module becomes unwieldy to manage (and it always seems possible to find exceptions to its side-effect predictions). Nonetheless, the independent modules can have access to information in the semantic composition module and use it in their computations.

3.5.2 Categorial Grammar

Categorial Grammar (CG) (Morrill, 1994; Carpenter, 1998; Steedman, 2000; van Benthem, 2003) is a family of formalisms in natural language syntax. In a CG, a sentence is grammatical iff its lexical items lead to a derivation of category s using certain logical rules. A basic example of a CG derivation is the following:

$$(124)\quad \frac{\text{John} : np \qquad \dfrac{\text{saw} : (np\backslash s)/np \quad \text{Mary} : np}{\text{saw Mary} : np\backslash s}\,/\,\text{ELIM}}{\text{John saw Mary} : s}\,\backslash\,\text{ELIM}$$

An expression of category $np\backslash s$ expects an expression of category np to its left and after application produces an expression of category s. Similarly with s/np which expects a np

to its right. Thus, $(np\backslash s)/np$ is the category of a transitive verb that expects its object NP to its right and yields a "verb phrase" which expects the subject NP to its left, to yield a sentence.

Categorial grammar has been popular among semanticists because of its type-theoretical elegance. Every linguistic category corresponds (through the Curry-Howard isomorphism – see section 5.1.2) to some type. Thus, $type(np) = e$, $type(s) = t$, $type(s/np) = type(np\backslash s) = e \rightarrow t$, and more generally, $type(A/B) = type(B\backslash A) = type(B) \rightarrow type(A)$. If lexical items are assigned meanings of the appropriate types, we get a meaning for each step in the derivation, and in particular, to the entire sentence. For example:

$$(125)\quad \dfrac{john : np \qquad \dfrac{\lambda y\lambda x.saw(x,y) : (np\backslash s)/np \quad mary : np}{\lambda x.saw(x,mary) : np\backslash s}\,/\,\textsc{elim}}{saw(john,mary) : s}\,\backslash\,\textsc{elim}$$

As pointed out in (Dalrymple, 1999, chapter 7), there is a strong similarity between the categories on the right-hand-side of semantic contributions and syntactic categories in Categorial Grammar. Thus, (125) is similar to the derivation (80) except that the basic entry for *saw* has the variable order reversed:

(126) a. $\lambda y\lambda x.saw(y,x) : \mathbf{3}^e \rightarrow \mathbf{2}^e \rightarrow \mathbf{1}^t$ (= (78) after EXCH applied on it)

b. $\lambda y\lambda x.saw(x,y) : (np\backslash s)/np$ (CG)

The left-hand-side in both CG and Glue is just the semantic functor itself, and often it is the same in both, e.g. (126)a and (126)b. The semantic information carried in categories in both CG and GS is also often the same, e.g. the semantic type in (126) is $e \rightarrow e \rightarrow t$. However, the combinatorial information in the categories is quite different. In CG, the categories specify where to look for their arguments *in the surface sentence*: s/np and $np\backslash s$ are of the same semantic type $e \rightarrow t$ but the former can only combine with an np to its right, while the latter, to its left. In contrast, glue categories in themselves are decoupled from the sentence and the syntactic structures, and equations between those categories serve to constrain the way that the left-hand-side semantic terms may combine. Therefore, glue statements are compatible with any syntactic framework that can supply such equations. If we use LFG as the syntactic framework, the equations are specified in the lexical entries in terms of places in the f-structure (and sometimes specified in the c-structure rules), as we will see in the next chapter. If we use HPSG as the syntactic framework, the equations are obtained through structure-sharing, as was shown in (Asudeh

and Crouch, 2001). A Glue Semantics version for LTAG was shown in LTAG (Frank and van Genabith, 2001), and even a version for Categorial Grammar itself exists (Asudeh et al., 2002).

There are several advantages to the GS approach compared to CG:

Modularity – Separation of Semantics from Syntax: Many people interested in the syntax-semantics interface started working in CG because of the nice coupling of syntax, types, and semantics. Glue semantics can be seen as a generalization of CG which allows the same nice properties except that it frees the syntactic side from being tied to surface word order, and allows the use of more powerful syntactic theories.

For example, LFG's separation of F-structure from C-structure has several advantages over CG. For one, there are some syntactic restrictions (e.g. in binding theory) which are difficult to express on the linear order of the surface string and easier to express in terms of a hierarchical syntactic analysis in the F-structure. Also, it is more difficult for CG to deal with free word-order languages because it depends on the surface order of words, while frameworks like LFG are more powerful in this respect. Finally, LFG allows the use of very similar F-structures across languages while restricting the differences to the C-structures. Since GS statements in the LFG version are mainly based on F-structures, there is consequently much less work that needs to be done when porting a GS specification to other languages, in comparison to CG, where most of the glue entries need to be rewritten to account for the different word order.

Elegant treatment of non-contiguous components: It is much easier in GS to deal with the construction of semantic functors whose material does not follow surface word order, or comes from non-contiguous parts of the sentence.

The logic underlying CG is *non-commutative* linear logic. Like linear logic, it is sensitive to the number of occurrences of a formula, but in addition, the order of formulas matters, and implication is sensitive to that order. Thus:

(127)
$$\begin{array}{lll} A,\ A\backslash B\ \vdash B & \text{whereas} & A\backslash B,\ A\ \nvdash B \\ B/A,\ A\ \vdash B & & A,\ B/A\ \nvdash B \end{array}$$

This logic is used in CG because the grammar is supposed to account for word order (in addition to semantic composition).

As we have seen above, syntactic structures do not neatly parallel corresponding semantic structures, and so a mapping from syntax-to-semantics that depends too closely on the syntax is going to run into complexities. In CG, the mismatch is even greater because the syntax itself intimately depends on surface word order.

For example, in order to deal with quantifiers, Carpenter (1998), following Moortgat (1990), needs to introduce a new operator $\Uparrow$ for creating categories. A category $B{\Uparrow}A$ is assigned to expressions that act locally as a B but take their semantic scope over an embedding expression of a category A. Thus, a generalized quantifier will be given category $np{\Uparrow}s$ because it acts like a noun phrase *in situ* but scopes semantically to an embedding sentence. The type of $B{\Uparrow}A$ is $((type(B) \rightarrow type(A)) \rightarrow type(A)$. For this operator to behave correctly, a new rule needs to be added to the logical system:

(128)

$$\dfrac{\begin{array}{c}\vdots\\ \dfrac{\alpha : B{\Uparrow}A}{x : B}\,{\Uparrow}E_i \\ \vdots \\ \beta : A\end{array}}{\dfrac{\alpha(\lambda x.\beta) : A}{}\,D_i}$$

provided x is fresh
and none of the assumptions in the derivation of
$\alpha : B{\Uparrow}A$ are available for discharge in the derivation of
$\beta : A$ from $x : B$ and the surrounding context.

For example, when $B = np$ and $A = s$, this rule says that once a quantifier $np{\Uparrow}s$ is derived, it can be turned into a variable that participates in a derivation of a sentence. At that point, that variable can be abstracted over to yield a property, and the quantifier α can scope over that property.

This new operator is introduced because the non-commutativity constraints of CG need to be relaxed. In contrast, treating quantifiers in glue semantics is done straightforwardly by assigning appropriate constraints on categories in right-hand-side type expressions, and crucially, no additional proof rules are required. This is possible because GS is based on the simpler, commutative linear logic, while accounting for word order is left as the responsibility of the syntactic module. Also, the somewhat mysterious behavior of a quantifier locally "turning into a variable" in (128) is not needed – see for example (111), and the non-trivial side conditions of (128) need no analog in the glue semantics system.

This difference has very important consequences for the ease with which one can build computational implementations of these systems. For a semanticist, it is easier to write the syntax-semantics specification in glue semantics because there is no need to worry about the order of words in a sentence. For a computational linguist, it is easier to implement

a program that finds derivations in a logical system that has just two rules (implication elimination and introduction) compared to a system with more rules (CG).

In fact, when Carpenter later wants to deal with a reciprocal expression like *each other*, the operator $\Uparrow$ and its rule (128) are not enough and need to be generalized. If a computer program that knows how to deal with $\Uparrow$ is already implemented before this need is discovered, the program might need serious revision. In contrast, as we shall see in chapter 13, reciprocals can be handled in glue semantics just as easily as other constructions. The strategy in GS is to use a simple general logic, and then, as necessary, add "bells and whistles" that may add restrictions (e.g. scoping constraints). It is easier to follow this strategy than to start with a restricted system and then try to expand it by adding more derivation rules.

Greater linguistic precision: As mentioned in Asudeh et al. (2002), CG (like LTAG) does not always clearly identify semantic modifiers. For example, both *often* and *promise* are of category $(np\backslash s)/(np\backslash s)$ because both expect a VP $(np\backslash s)$ to their right and return a new VP:

(129) (John) often leaves.

$$\frac{\lambda P\lambda x.often(P(x)) : (np\backslash s)/(np\backslash s) \quad leave : (np\backslash s)}{\lambda x.often(leave(x)) : (np\backslash s)} \; / \text{ ELIM}$$

(130) (John) promised to leave.

$$\frac{\lambda P\lambda x.promise(x, P) : (np\backslash s)/(np\backslash s) \quad leave : (np\backslash s)}{\lambda x.promise(x, P) : (np\backslash s)} \; / \text{ ELIM}$$

However, these are two quite different kinds of modifiers – the second is a control (equi) verb, the first is not. The issue here is that CG has a limited stock of atomic categories: s, np, n, etc., one for each type of grammatical category. In contrast, glue semantics has a richer stock of atomic resources – not just grammatical categories, but instances of them as they exist in a particular syntactic structure. Hence, it can distinguish different occurrences of resources, and distinguish genuine modifiers from control verbs. In "CG notation", we would say that *often* is a modifier with category $(np_1\backslash s_1)/(np_1\backslash s_1)$ whereas *promise* has category $(np_1\backslash s_1)/(np_1\backslash s_2)$. We can see that in glue notation as well:

(131) $[(\text{John})_2$ often leaves$]_1$.

$$\frac{\lambda P\lambda x.often(P(x)) : (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{2}^e \to \mathbf{1}^t \quad leave : \mathbf{2}^e \to \mathbf{1}^t}{\lambda x.often(leave(x)) : \mathbf{2}^e \to \mathbf{1}^t} \; \text{E}$$

(132) $[(\text{John})_2$ promised $[\text{to leave}]_3]_1$.

$$\frac{\lambda P \lambda x.promise(x,P) : (\mathbf{2}^e \to \mathbf{3}^t) \to \mathbf{2}^e \to \mathbf{1}^t \quad leave : \mathbf{2}^e \to \mathbf{3}^t}{\lambda x.promise(x,P) : \mathbf{2}^e \to \mathbf{1}^t}\ \text{E}$$

In conclusion, a linguist, or a computational linguist, working on the syntax-semantics interface, who likes having the nice parallelism between semantic forms and categories, does not have to use categorial grammar for that purpose. Glue Semantics has this nice parallelism while providing a greater precision than CG, as well as freeing the linguist from being restricted to surface word order.

3.5.3 Storage Methods and Quasi-Logical Forms

Cooper (1983) presents a technique for handling quantifier scope ambiguities. Each node of the parse tree is associated not with one meaning term but with a pair consisting of the "core" meaning term and a "storage" of the quantifiers that arise from nodes lower in the tree. In the pair associated with a noun-phrase, the core part is a variable, and the store part includes the quantifier itself, indexed with the variable. The store may include additional quantifiers that appear inside the NP. Schematically, we may write this as $\langle x, [(Q, N, x), \ldots] \rangle$ where N is the meaning of the N' node of the NP.

Stores of quantifiers are passed up the tree in parallel to the meaning terms. On a clausal node, quantifiers may be retrieved from the store to scope over that clause. Different orders of retrieving the quantifiers yield different scopings.

Thus, the NP node in "every man walked" is associated with $\langle x, [(every, x, man)] \rangle$. The VP is associated with $\langle \lambda y.walk(y), [\,] \rangle$. Now the first component of the VP's pair can be applied to the first component of the NP's pair, yielding $\langle walk(x), [(every, x, man)] \rangle$. At this point, the quantifier can be retrieved from the store to yield $\langle every(x, man(x), walk(x)), [\,] \rangle$. If we had two quantifiers as in "every man saw some woman", there would be two orders in which quantifiers can be retrieved, leading to two final results.

More details can be found in (Blackburn and Bos, 2005a). A similar idea appears in quasi-logical forms (QLFs) (Alshawi, 1992).

A problem with this approach is that it uses operations that manipulate syntactic expressions, but these operations are not given a justification in terms of an underlying model-theoretic semantics. In other words, we do not have an independent way to know whether the obtained results are correct. A symptom of this issue is that it was later discovered that the original formulation of the storage technique sometimes produces logical

forms that have unbounded variables in them. This unintended result was later fixed by Keller (1988), but the point is that we want to avoid getting into such problems in the first place. More on this point will be said in section 7.7.3. Another symptom of this problem is that the technique is tailored only for type $\langle 1, 1 \rangle$ quantifiers that accept two sets as arguments (roughly, the denotation of the N' and of the VP). It does not generalize to other operators such as reciprocals which are type $\langle 1, 2 \rangle$ quantifiers that accept a set and a binary relation. The reason is similar to what we will see in section 13.6.1.

Another problem is that this technique is defined in procedural rather than declarative terms: the order in which operations are done is important. This makes it harder to predict what the results will be, and to debug the implementation. Indeed, the algorithms that transform QLFs to logical forms are very complex (Hobbs and Shieber, 1987; Moran and Pereira, 1992). For more about this point, see the discussion of order-independence in section 2.4 of (Crouch, 1999).

Similar comments apply to underspecified representations, but we will have more to say about them in section 7.7.

The Glue Semantics approach does not suffer from these problems. The meaning terms in glue statements have a well-defined denotation (they denote mathematical entities), and what the framework describes is constraints on the semantic composition of these entities, not just on syntactic manipulation of formulas. Also, the framework is declarative; it simply states equations between labels of category expressions on the right-hand-side of glue statements. A separate and principled logical apparatus takes care to use these constraints in order to combine the meaning terms correctly.

Chapter 4

Implementing a Glue Specification

In this chapter, I describe a computational implementation of the glue specification we saw in the previous chapter, and extensions of it. The implementation takes as input syntactic structures in LFG, as they are produced by the XLE system, and outputs a set of glue semantics premises. This chapter only covers the translation from syntax to semantic pieces, but does not explain how to calculate all the ways in which those pieces combine to form meaning representations. A basic way of calculating this (in an unpacked way) is covered in chapter 5. Then, chapters 6 and 7 will cover how to do this calculation in an efficient packed way.

I chose LFG as the syntactic theory and the XLE implementation for several reasons.

1. LFG has two levels of syntactic representation: c-structure and f-structure, as will be explained below. It is easier to define the glue specification in relation to f-structure rather than the familiar constituent structures.

2. The XLE has a broad-coverage grammar of English (and several other languages) written in LFG which I can use off-the-shelf; only minor modifications are needed.

3. Despite this broad-coverage, the parser and the grammar are precise: They are based on a hand-written lexicon and rules that were constructed by computational linguists over many years. Because of the nature of the precise understanding applications I aim to investigate in my larger research project, I require both the high-level of linguistic accuracy provided by the grammar, and the reliability (high precision) provided by a non-statistical parser. Nonetheless, as mentioned in section 1.3.3, the

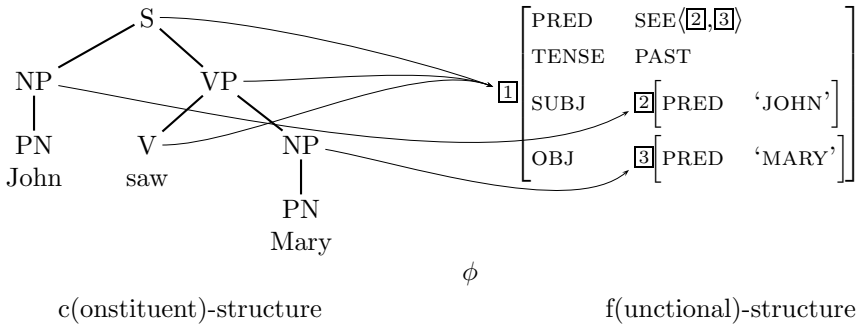


Figure 4.1: LFG c-structure and f-structure

XLE can use statistical filtering to reduce the number of possible analyses. This will not matter much in this dissertation because we are concerned here mostly with sentences that are short or have very low syntactic ambiguity.

4. The XLE has an ambiguity-management framework based on a *choice space*, which supports a uniform handling of ambiguities across components. This is quite unique to the XLE. Furthermore, the XLE comes with a packed rewriting system that is neatly integrated with this choice space framework. I will show below how to implement the glue semantics specification by using this rewriting system.

5. The generous support I received from the people at PARC: Dick Crouch, Valeria de Paiva, Tracy King, and John Maxwell.

4.1 Lexical Functional Grammar and Glue Semantics

4.1.1 A Brief Introduction to LFG

Lexical Functional Grammar (LFG) (Bresnan, 2001; Dalrymple, 2001) has several components. Here it is sufficient to discuss the two main syntactic components: c(onstituent)-structure and f(unctional)-structure. The first deals with phrase-structure trees that address sentence word order and hierarchy, whereas the second one deals with feature-structures that have a more flattened representation of the information according to syntactic roles (such as *subject* and *object*). Thus, in Figure 4.1, the subject and object are on different levels in the c-structure, but on the same level in the f-structure.

The separation of c-structure and f-structure has several advantages:

1. Some syntactic constraints (such as in binding theory) are more easily expressed as constraints on f-structure rather than c-structure.

2. Languages differ considerably on the c-structure level, but much less so on the f-structure level. Languages with a different word order than English, and even with a free word order (where syntactic roles are identified using morphological markings) still mostly have the same f-structures. Hence, syntactic constraints that are defined based on the f-structure level but are independent of the c-structure level could be ported to these other languages.

3. As we shall see below, there are also advantages for the syntax-semantics interface.

The two levels of syntax are related through a projection function ϕ, which is displayed in Figure 4.1 as the arrows. Several c-structure nodes may be related by ϕ to the same f-structure. The lexical entry of a word specifies constraints on the c-structure and f-structure. For example:

(133)

"John"	PN	$(\uparrow \textsc{pred}) = \text{`}\textsc{john}\text{'}$
"Mary"	PN	$(\uparrow \textsc{pred}) = \text{`}\textsc{mary}\text{'}$
"saw"	V	$(\uparrow \textsc{pred}) = \text{`}\textsc{see}\langle(\uparrow \textsc{subj}), (\uparrow \textsc{obj})\rangle\text{'}$
		$(\uparrow \textsc{time}) = \text{`}\textsc{past}\text{'}$

The symbol '↑' means "the f-structure of my parent node". Thus, "John" specifies that the PRED feature in the f-structure of its mother node (the NP) is equal to 'JOHN'. In Figure 4.1, the ↑ of "John" is 2, that of "Mary" is 3, and that of "saw" is 1. In addition, LFG has c-structure rules:

(134)

a. $\underset{}{\text{S}} \rightarrow \underset{(\uparrow \textsc{subj}) = \downarrow}{\text{NP}} \quad \underset{\uparrow = \downarrow}{\text{VP}}$

b. $\text{VP} \rightarrow \underset{\uparrow = \downarrow}{\text{V}} \quad \underset{(\uparrow \textsc{obj}) = \downarrow}{\text{NP}}$

c. $\text{NP} \rightarrow \underset{\uparrow = \downarrow}{\text{PN}}$

The sign '↓' means "the f-structure of this node". Thus, ↑=↓ in the second rule equates the f-structure 3 of the V node and VP node, as shown in Figure 4.1. And $(\uparrow \textsc{subj}) = \downarrow$ in the first rule specifies that the value of the SUBJ feature of the f-structure of the mother node S

is equal to the f-structure of the NP node, so in Figure 4.1 this specifies that $(\boxed{1}\ \text{SUBJ}) = \boxed{2}$. Similarly, the second rule adds the constraint $(\boxed{1}\ \text{OBJ}) = \boxed{3}$.

Of course, the lexical entries in (133) need not be specified separately for each word. We can have lexical schemata such as:

(135) proper name W with meaning n:
W PN $(\uparrow \text{PRED}) = n$

4.1.2 Glue Semantics for LFG

Using this syntactic theory, where the syntactic roles are explicitly given in the f-structure, makes it easy to provide a glue specification. We make each lexical schema include a meaning-category pair $\psi : A$ as follows:

(136) proper name n $\quad n : {\uparrow_\sigma}^e$
transitive verb m $\quad m : {(\uparrow \text{SUBJ})_\sigma}^e \to {(\uparrow \text{OBJ})_\sigma}^e \to {\uparrow_\sigma}^t$

The σ marking is the semantic projection function from an f-structure to a category. For an easier notation, we always mark an f-structure with a boxed number, e.g. $\boxed{1}$, and its corresponding semantic handle as the same number in boldface, e.g. $\mathbf{1}$. Thus, $\boxed{1}_\sigma = \mathbf{1}$. Consequently, the semantic entries contributed by the lexical schemas in this case are precisely the ones that were shown in (78), repeated here as (137): The $\uparrow$ of "John" is $\boxed{2}$, so we get $john : \mathbf{2}^e$. And the $\uparrow$ of "saw" is $\boxed{1}$, so there $(\uparrow\ \text{SUBJ}) = \boxed{2}$ and $(\uparrow\ \text{OBJ}) = \boxed{3}$, and we get $see : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t$.

(137) $john : \mathbf{2}^e \qquad mary : \mathbf{3}^e \qquad see : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t$

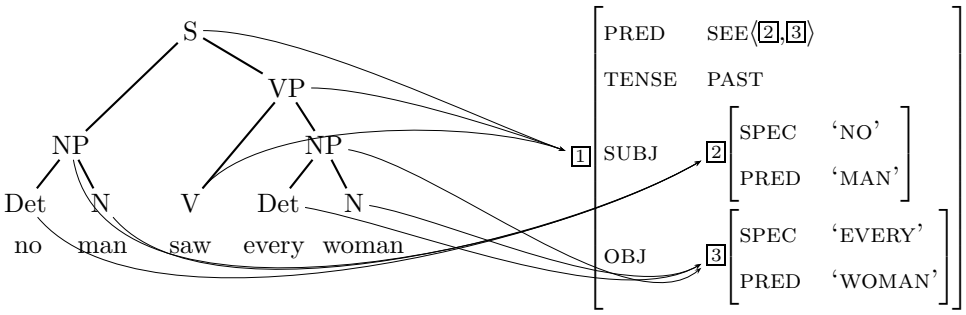
We see that the separation between c-structure and f-structure also has an advantage for the semantics. For the purpose of semantics, it is easier to think about the f-structures rather than the c-structures. For example, the subject and object are on the same level in the f-structure, as they are in the meaning expression, so one does not need to think about the hierarchy of the c-structure. Also, the advantage mentioned above about the dependence of only c-structure on a particular language's word order and the cross-linguistic similarity of the f-structure carries over to the semantics: since the semantics depends only on the f-structure, it too can be ported to other languages.

The upshot is that someone who is interested in the syntax-semantics interface, i.e. how the combination of word meanings is constrained by syntax, does not need to worry about

c-structure and may concentrate on f-structure only. The already-developed syntactic theory of LFG takes care of the c-structure level and its connection to f-structure.

I will, however, give one more example with c-structure. A quantified NP gets a c-structure and f-structure as in the following example:

(138) No man saw every woman.



The lexical entries are:

(139) "man" N (↑ PRED) = 'MAN'
"every" Det (↑ SPEC) = 'EVERY'

and the additional grammar rule is:

(140) NP → Det N
↑=↓ ↑=↓

To this we add the following, which are a restatement of (94) and (101):

(141) a noun with meaning m:
$m : {(\uparrow_\sigma)_v}^e \rightarrow {(\uparrow_\sigma)_r}^t$

(142) a quantifier with meaning Q:
$Q : (({\uparrow_\sigma)_v}^e \rightarrow {(\uparrow_\sigma)_r}^t) \rightarrow ({\uparrow_\sigma}^e \rightarrow h^t) \rightarrow h^t$
h is the label of the f-structure of my scope clause (which I am subordinated to)

When these are instantiated in (138) we get:

(143) $no : (\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t) \rightarrow (\mathbf{2}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$
$man : \mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$
$every : (\mathbf{3}_v^e \rightarrow \mathbf{3}_r^t) \rightarrow (\mathbf{3}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$
$woman : \mathbf{3}_v^e \rightarrow \mathbf{3}_r^t$

4.2 Packed Rewriting System

In order to understand how to use the XLE's packed rewriting system to transform packed F-structures to a packed set of glue premises, we first need to explain three things. First, the general idea of packing and choice-space in more detail. Second, the internal packed representation of F-structures in terms of contexted facts. And finally, how the rewriting system can take these contexted facts and rewrite them efficiently to new contexted facts, possibly with a modified choice-space.

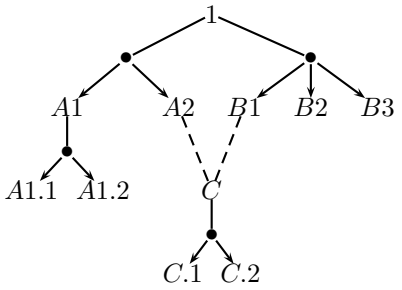
4.2.1 Choice Space

I mentioned in section 1.3.3 that the ambiguity management framework in the XLE generalizes the idea of packing in a chart parser to other data structures in other levels of analysis. This is explained in more detail here. The information here is based on (Crouch, 2005), with some added definitions.

Free Choice Space

A (*free*) *choice-space* is essentially an AND/OR tree of choices, where each choice is represented by a boolean variable. The root of the tree is called the *top choice*, *top context*, or *true context*, and is marked with 1. Here is an example of a choice space:

(144)



where $C = A2 \vee B1$

Bullets are OR nodes, whereas lines that go into bullets from above come out of AND nodes. Under the top context, one can choose exactly one of $A1$ and $A2$ (and these two choices are inconsistent with each other). There is also a choice between $B1$, $B2$, and $B3$. The latter choice is orthogonal to the former, i.e. all $2 \times 3 = 6$ combinations are possible. Context $A1$ has a further choice between the two sub-contexts $A1.1$ and $A1.2$. The context $A2$ has two sub-contexts $C.1$ and $C.2$, and so does the context $B1$. Hence the choice C, which is a disjunction of $A2$ and $B1$, is represented just once in the choice-space.

The context names in the picture are just for illustration. The important thing is that they are boolean variables that satisfy the following statements in propositional logic (where 1 stands for $true$ and 0 for $false$):

(145) $1 : \textit{one-of}(A1, A2)$
$1 : \textit{one-of}(B1, B2, B3)$
$A1 : \textit{one-of}(A1.1, A1.2)$
$A2 \vee B1 \leftrightarrow C$
$C : \textit{one-of}(C1, C2)$

where:

$$X : \textit{one-of}(Y_1, \ldots, Y_n) \quad \equiv \quad (X \leftrightarrow (Y_1 \vee \ldots \vee Y_n)) \wedge \bigwedge_{1 \leq i < j \leq n} (Y_i \wedge Y_j \leftrightarrow 0)$$

A *choice* in the choice-space is any combination of nodes in the choice space using conjunctions, disjunctions, and negation. Here are example choices in the choice-space above: $A1$, $A1 \wedge B1$, $A2 \wedge C.1$, $\neg B2$, $B1 \vee B3$ (the last two are equivalent).

Here are some definitions that will be needed later, especially in chapter 7. Two choices D_1 and D_2 are (*mutually*) *inconsistent* iff $D_1 \wedge D_2 = 0$. A choice D_1 is *more specific* than a choice D_2 if both $D_1 \rightarrow D_2$ and $D_2 \wedge \neg D_1 \neq 0$ are true. In the example above, $A1.1$ is more specific than both $A1$ and 1, but is not more specific than $B1$. A *maximally-specific choice* is a choice $D_1 \neq 0$ such that there is no other choice D_2 that is more specific than D_1 (in the example above, $B2 \wedge A1.1$ is maximally-specific, but $B2$ and $A1.1$ are not, because $B2 \wedge A1.1$ is more specific than them). We may say that a choice D_1 is *included* in a choice D_2 if $D_1 \rightarrow D_2$ holds, and we may write this as $D_1 \subseteq D_2$ (so a choice includes itself). D_1 is included in D_2 iff D_1 is equal to or more specific than D_2. If D_1 is maximally-specific and included in D_2, we may also write this as $D_1 \in D_2$.

The choice structure is defined independently from the facts that sit under the various choices. These could be facts about parts of any data structure, including morphological, syntactic, and semantic structures. Let us return to example (19) in more detail. Here is a schematic free-choice representation of the 4-way syntactically ambiguous sentence "The sheep ate the cabbage":

(146) Free choice structure:
$1 : \textit{one-of}(A1, A2)$
$1 : \textit{one-of}(B1, B2)$

Choiced clauses:

$A1$:	sg(sheep)	$A2$:	pl(sheep)
$B1$:	count(cabbage)	$B2$:	mass(cabbage)
1 :	eat(sheep,cabbage)		

In section 4.2.2, we will see how facts about pieces of packed F-structures are represented.

An important point about the free-choice structure is that the choice between $A1$ and $A2$ is completely independent from the choice between $B1$ and $B2$, so we get $2 \times 2 = 4$ analyses. However, suppose that we have further information that several (plural) sheep cannot eat a single (count) cabbage. This fact introduces an interaction between previously independent choices. In effect, we are declaring that $A2 \wedge B1$ is a *nogood*. To retain a choice-free structure, we need to re-factor the choice-space to preserve freedom of choice:

(147) New free choice structure:

1 : *one-of*$(A1, A2)$

$A1$: *one-of*$(C1, C2)$

$D \leftrightarrow A2 \vee C2$

Choiced clauses:

$A1$:	sg(sheep)	$A2$:	pl(sheep)
$C1$:	count(cabbage)	D :	mass(cabbage)
1 :	eat(sheep,cabbage)		

Here we know eat(sheep,cabbage) under the top context, and we have a free choice between $A1$ and $A2$. If we select $A2$, then the facts there are pl(sheep) and mass(cabbage). If we select $A1$ then the set of facts there includes sg(sheep), and we further have a free choice between $C1$ and $C2$. Under $C1$ we have the further fact count(cabbage), and under $C2$, mass(cabbage).

Free Choice Packing

The representation in (147) is *packed* because shared pieces appear only once. For example, even though there is a total of three choices, the fact eat(sheep,cabbage) is stored only once. This is a generalization of packing in a chart forest, as was shown in Figure 1.3.

Packing by itself is not enough. We want to not only represent each common part just once, but also to compute it just once. In other words, we do not want to merely compute all the possible complete structures and then pack them, because that would involve a lot of repeated work.

In a chart parser for a context-free grammar, context-freeness guarantees that alternative analyses of disjoint word spans are independent and do not interact. Hence, one can freely combine any analyses of disjoint spans. Chart parsers exploit this to compute all complete analyses in $O(n^3)$ time and $O(n^2)$ space. Furthermore, one can efficiently compute the number of analyses without having to enumerate all of them.

A free choice space has this last property as well. This is key for efficient stochastic disambiguation of packed structures (Riezler and Vasserman, 2004). Moreover, propositional satisfiability is trivial to check in the choice space thanks to the AND/OR tree nature of the choice structure: A conjunction of two choices is unsatisfiable iff the lowest node in the tree dominating both of them is an OR node.

However, propositional satisfiability in general is NP-complete, so these nice properties must come with a price. Re-arranging the choice space to account for additional boolean combinations of choices increases the size of the choice space. In the worst case, preserving free choice may cause the space to expand to a size proportional to the (exponential) total number of possible analyses, because the space will degrade into a disjunctive normal form enumeration of all the analyses.

Nevertheless, for typical language processing tasks, these bad cases rarely arise in practice. Choices in distant parts of a sentence or text usually do not interact (coordinations and long-distance dependencies such as wh-questions being the exceptions), so free choice spaces normally retain a manageable size. Thus, although LFG is a non-context-free grammar and analyses of disjoint word spans are not always independent, they are mostly independent and interactions are minimal. Under these conditions, one can efficiently refactor the analyses to obtain an equivalent representation that guarantees free choice alternatives. This is exploited in (Maxwell and Kaplan, 1996), which shows how a non-context-free unification-based parser can automatically take advantage of whatever context freeness does exist in the input.

4.2.2 Packed Morphosyntactic Analysis

LFG in the XLE

When the morphological and syntactic stages complete analyzing a sentence, the XLE has information about each word – its stem, prefixes and suffixes, and part of speech – and it also has c-structures and f-structures. There might be more than one analysis at any of the

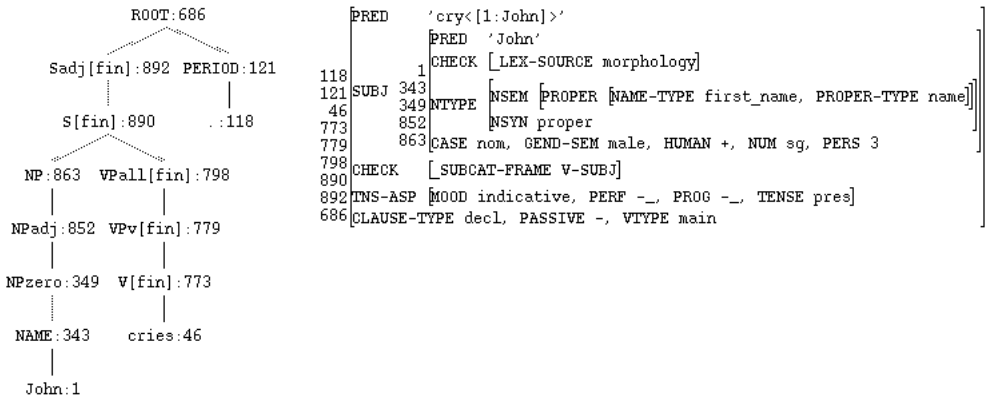


Figure 4.2: XLE's graphical presentation of c-structure and f-structure

three levels, although analyses that are possible locally are deleted if they are inconsistent with later stages. Thus, in "John will book a flight", the word "book" is analyzed by the morphological analyzer as potentially a noun or a verb, but the first option does not survive in the full analysis.

A screenshot of XLE's graphical display of the c-structure and f-structure for "John cries" is shown in Figure 4.2. The numbers denote distinct variables over f-structure nodes. Many variables may be equated through constraints like the ones we saw in (134). You can see which sub-f-structure corresponds to which c-structure node by these numbers. For example, the "John" node (numbered 1), the NP node (numbered 863), and the nodes in between them in the c-structure correspond to the sub-f-structure whose `PRED` value is `'John'`.

Notice that the value of `PRED` in the main f-structure is `'cry<[1:John]>'`. The argument `[1:John]` is simply structure-shared with the value of the `SUBJ` grammatical function. Also, the predicate is `cry` and not `cries` thanks to the morphological analysis. We can see more information about the morphology by clicking on the `cries` node in the c-structure and selecting "Show Morphemes." We then get the tree in Figure 4.3. We can see that "cries" was analyzed as a verb in the present tense and 3rd singular form. The information on tense found its way to the `TNS-ASP|TENSE pres` value in the f-structure in Figure 4.2 through the grammar equations. Similarly, the information on the verb's number and person was constrained by the grammar to be equal to that of the subject and so it ended up in John's sub-f-structure.

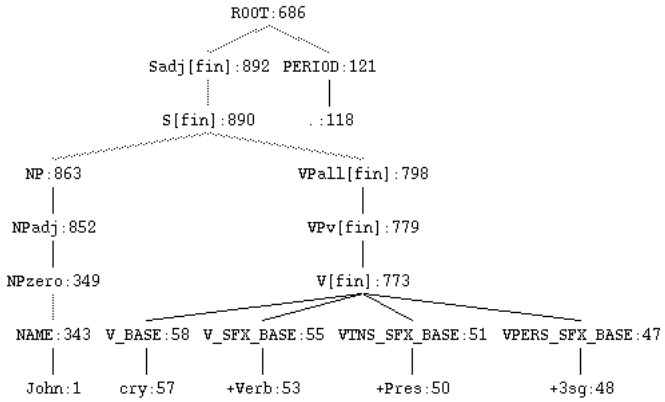


Figure 4.3: XLE's graphical presentation of c-structure with a leaf node expanded

Fact-Representation of Structures

The information in these structures is stored in the form of atomic facts. Here are some of these facts:

```
cf(1, surfaceform(46, 'cries', 6, 11))
```

This says that the word "cries" with id `46` was analyzed with morphemes `6` through `11` (not all of them were displayed in Figure 4.3).

```
cf(1, terminal(57, 'cry', [46]))
cf(1, terminal(50, '+Pres', [46]))
```

This says that the terminals `cry` and `+Pres` with ids `57` and `50` are part of the analysis of `46`.

```
cf(1, subtree(58,'V_BASE','-',57))
cf(1, subtree(51,'VTNS_SFX_BASE','-',50))
cf(1, subtree(770,'V[fin]','-',58))
cf(1, subtree(771,'V[fin]',770,55)),
cf(1, subtree(772,'V[fin]',771,51)),
cf(1, subtree(773,'V[fin]',772,47))
cf(1, subtree(779,'VPv[fin]','-',773))
cf(1, subtree(798,'VPall[fin]','-',779))
cf(1, subtree(890,'S[fin]',880,798))
cf(1, subtree(880,'S[fin]','-',863))
```

The first two facts say that `57` is the daughter node of `58`, and `50` that of `51`. The next four facts say (indirectly) that `58`, `55`, `51`, and `47` are the daughters of `773` (as was shown in Figure 4.3). Then the chain continues from `773` to `779`, `798`, and finally `890`.

Each of the c-structure nodes is mapped under the ϕ mapping to an f-structure node. Here are some of these facts:

```
cf(1, phi(57,var(0)))
cf(1, phi(773,var(0)))
cf(1, phi(798,var(0)))
cf(1, phi(890,var(0)))
cf(1, phi(343,var(1)))
cf(1, phi(863,var(1)))
```

As with the basic LFG example we saw before, the V, VP, and S nodes are mapped to the same f-structure node `var(0)` while the Name and NP nodes are mapped to the f-structure node `var(1)`.

We also have the following fact:

```
cf(1, arg(var(0),1,var(1)))
```

This says that `var(1)` is the first thematic argument of `var(0)`. In the sentence "John saw Mary", we would also have a fact like `arg(var(0),2,var(7))`. A "thematic argument" is an argument that is connected to the verb through a thematic role. An example of a non-thematic argument is the subject in "It is raining."

Packed Structures

All the above facts are surrounded by `cf(1, ...)`. This says that all the facts are in the top context of the choice space, because there were no ambiguities. In an ambiguous sentence, some of the facts would be under sub-choices. For example, the packed f-structure visualization of the sentence "The deer met the sheep" is shown in Figure 4.4.

This packed F-structure has the following facts:

```
choice([A1, A2], 1)
choice([B1, B2], 1)
```

These say that the top context is divided into A1 and A2, i.e. $1 \leftrightarrow A1$ xor $A2$, and similarly with B1 and B2. Now some of the facts are relativized to these contexts:

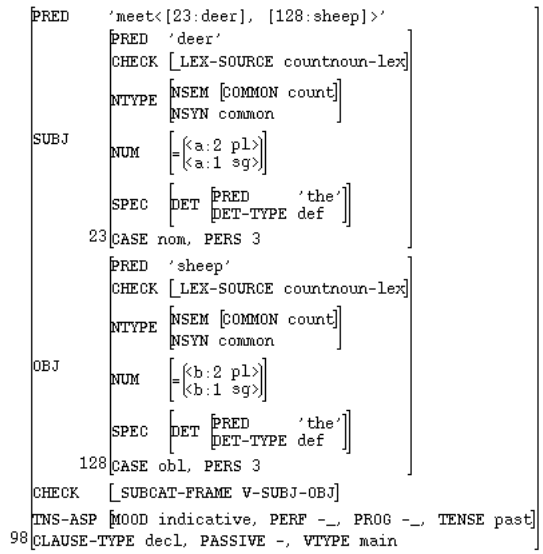


Figure 4.4: XLE's graphical presentation of a packed f-structure

```
Facts:
  cf(1, 'PRED'(var(1),deer))
  cf(1, 'NUM'(var(1),var(7)))
  cf(1, 'PRED'(var(2),sheep))
  cf(1, 'NUM'(var(2),var(13)))
Equalities:
  cf(A1, eq(var(7), 'sg'))
  cf(A2, eq(var(7), 'pl'))
  cf(B1, eq(var(13), 'sg'))
  cf(B2, eq(var(13), 'pl'))
```

These can be treated as if they were:

```
  cf(1, 'PRED'(var(1),deer))
  cf(A1, 'NUM'(var(1),'sg'))
  cf(A2, 'NUM'(var(1),'pl'))
  cf(1, 'PRED'(var(2),sheep))
  cf(B1, 'NUM'(var(2),'sg'))
  cf(B2, 'NUM'(var(2),'pl'))
```

4.2.3 Packed Rewriting

The input to the rewriting system is a packed set of clauses. Rewrite rules apply in an ordered, stepwise manner to progressively replace input clauses by output clauses. An

Rule	::=	LHS `==>` RHS.	Obligatory rewrite
		LHS `?=>` RHS.	Optional rewrite
		`|-` Clause.	Permanent, unresourced fact
LHS	::=	ClausePattern	Match & delete atomic clause
		`+`ClausePattern	Match & preserve atomic clause
		LHS`,` LHS	Boolean conjunction
		`(`LHS `|` LHS`)`	Boolean disjunction
		`-`LHS	Boolean negation
		{ ProcedureCall }	Procedural attachment (Prolog code)
RHS	::=	ClausePatterns	Set of atomic replacement clauses
		`0`	Empty set of replacement clauses
		`stop`	abandon the analysis

Figure 4.5: Format of Rewrite Rules

earlier version of the rewrite system was used in machine translation. Here is an example of a rule in such a system ('%' marks variables):

```
PRED(%X,sleep), +VTYPE(%X,main) ==> PRED(%X,dormir)
```

This rule says to rewrite the verbal predicate `sleep` to `dormir`. The `PRED` fact is *replaced* by the new `PRED` fact, whereas the `VTYPE` fact is retained since it is prefixed by '+'. Figure 4.5 (taken from (Crouch, 2005)) shows a summary of the rewriting system's notation.

To see how the rewrite system interacts with a packed input, consider first the following rule and part of a packed input:

(148) Rule: `p(%X), q(%X) ==> r(%X).`

Input: `1:p(a), m(b)` `A1:q(a)` `A2:s(a)`

where `1` ↔ `A1` xor `A2`

The rule matches the input only under choice `A1`, where both patterns on the left-hand side can simultaneously be matched. This leads to the production of a new output fact, `r(a)`, and the deletion of the input facts only under `A1`. The matched input fact `p(a)` remains present under the residual choice `A2`. Thus the packed output of the rule application is:

(149) `1:m(b)` `A1:r(a)` `A2:p(a), s(a)`

In the following example, the choices `A1`, `B2`, and `C3` are not necessarily pairwise-inconsistent:

(150) Rule: `p(%X), q(%X) ==> r(%X).`

Input: `A1:p(a)` `B2:q(a)` `C3:s(a)`

The output of this is:

(151)
A1 & ¬B2:	p(a)	B2 & ¬A1:	q(a)
A1 & B2:	r(a)	C3:	s(a)

Rewrite rules can also introduce new choices through the resolution of resource conflict. Suppose we have:

(152) Rule: `p(%X), q(%X,%Y) ==> r(%X,%Y).`
Input: `C:p(a), q(a,b), q(a,c)`

There are two ways in which the rule can apply: either on `p(a),q(a,b)` or on `p(a),q(a,c)`. Since the rule consumes the input `p(a)`, it cannot simultaneously apply on both options. The rewriting system automatically detects such competition over input resources, and resolves the conflict by introducing a new split in the choice space. Here is the output:

(153) Output: `D1:r(a,b)` `D2:r(a,c)`
New choice: `C` ↔ `D1` xor `D2`

The application of an optional rule would similarly create a split in the choice space, where the rule applies under one choice but not in the other.

Conversely, if the same fact is created under two competing contexts, the rewrite system automatically detects this and merges the copies to have just one copy in the disjoined context:

(154) Rule: `p(%X), q(%X,%Y) ==> r(%Y).`
Input: `A1:p(a), q(a,c)` `A2:p(b), q(b,c)`
where `1` ↔ `A1` xor `A2`
Initial output: `A1:r(c)` `A2:r(c)`
Final output: `1:r(c)`

Thus, as far as the rule writer is concerned, she does not need to worry or think about the choice-space or the packed input explicitly. The rewrite system operates as if the entire input is first unpacked to distinct copies of the relevant clauses under each of the possible choices, and then the rules apply on each copy separately. The only time the writer needs to think about the choice-space is when he deliberately wants to create an ambiguity. This is done by writing a rule that can consume the same resource in more than one way, as in

(152). We will use this feature in section 4.5.3 where we will create an ambiguity when a quantifier may land on more than one clause.

For more on the rewrite system, see (Crouch, 2005) and the system documentation.[1]

4.3 Glue Semantics Using the Rewriting System

Previously, the XLE had an implementation of a glue specification that converted C/F-structure facts to glue premises. Here I present a new implementation that has at least three improvements. First, the old implementation was written using the old glue notation (see section 3.2.6) which made it somewhat difficult to understand and maintain. Here I will use the new notation. Second, the treatment of verbs was not optimal, as the specification had a separate rule for each verb subcategorization frame. Here (section 4.4.4) I will present a modular solution.

Finally, the old implementation was not packed (nor did it use the rewriting system). Instead, the packed F-structure was unpacked to the individual parses, and each was converted separately to a set of glue premises for which all the possible meanings were calculated. Only then were the results put again into a packed structure. In contrast, here the translation from F-structures to glue premises is packed, by utilizing the rewriting system. We want to get a packed set of glue premises, where common premises appear just once. We will then feed this into the packed glue prover in chapter 7, which will automatically and efficiently create a packed semantic representation as in Figure 1.4.

4.3.1 Notation

I extend the notation of the transfer system to allow us to express glue statements easily. I would like to write the expression

(155) $\lambda x \lambda y.meet(x,y) : {f_\sigma}^e \rightarrow {g_\sigma}^e \rightarrow {h_\sigma}^t$

using ascii characters as either of the following:

(156) a. `x\y\meet(x,y) : %F$e -> %G$e -> %H$t`
b. `x\y\[meet,x,y] : %F$e -> %G$e -> %H$t`

[1] http://www2.parc.com/isl/groups/nltt/xle/doc/transfer-manual.html

The sort (e.g. `e` or `t`) is written after a '`$`', and we ignore the $_\sigma$ marking as it does not play an important practical role. A lambda is expressed using the '\' symbol. A predicate applying on its argument is written as in either (156)a or (156)b. The second form, but not the first, can be used when the predicate itself is a variable in the rewrite system, as in `[%F,x,y]` (not: `%F(x,y)`). Note that `x` and `y` are variables in the meaning language and are not rewrite-system variables like `%F`.[2]

4.3.2 Using F-structures

If we follow the glue semantics literature, we need to use lexical schema to define the glue specification. Ideally, this knowledge is declarative and bi-directional. Thus, given a morphosyntactic analysis, the declarative knowledge can be used to find out the glue semantic statements. Conversely, if a semantic representation for the entire sentence is given in expanded form, corresponding to a glue proof including the leaves, the declarative knowledge could be followed to figure out constraints on the morphosyntactic structure. However, generation is not a trivial reverse of interpretation, and we will not deal with it in this dissertation. For now, we will use the rewrite system to rewrite facts about the morphosyntactic analysis to glue premises.

The lexical schema for a proper name could be written:

(157)
```
+terminal(%TNode,%Stem,%%), +subtree(%CNode,NAME_BASE,-,%TNode),
+phi(%CNode,%Fnode)
==> %Stem : %Fnode$e
```

This says that if a `%Stem` of a word in a terminal node `%TNode` has syntactic category `NAME_BASE` in a C-structure node `%CNode` which is mapped to an F-structure node `%FNode`, then create the glue fact whose meaning term is `%Stem` and whose glue category is `%Fnode$e`.

The schema for an intransitive verb would be:

(158)
```
+terminal(%TNode,%Stem,%%), +subtree(%CNode,V_BASE,-,%TNode),
+phi(%CNode,%Fnode), +SUBJ(%Fnode,%FSubj),
_SUBCAT-FRAME(%Fnode,V-SUBJ)
==> %Stem : %FSubj$e -> %Fnode$t
```

[2]The XLE's rewrite system is implemented in Prolog and uses the built-in Prolog DCG parser to define the language of the rewrite rules in a notation definition file. The extensions here are implemented by modifying this file to recognize the symbols '`:`', '`->`', etc. as special operators.

The schema for a transitive verb would be:

(159)
```
+terminal(%TNode,%Stem,%%), +subtree(%CNode,V_BASE,-,%TNode),
+phi(%CNode,%Fnode), +SUBJ(%Fnode,%FSubj), +OBJ(%Fnode,%FObj),
_SUBCAT-FRAME(%Fnode,V-SUBJ-OBJ)
==> %Stem : %FSubj$e -> %FObj$e -> %Fnode$t
```

If we use these rules on the sentence "John cries" above, we will get the glue premises:

(160)
```
John : var(1)$e
cry : var(1)$e -> var(0)$t
```

The glue prover (described in later chapters) would combine these to get

(161)
```
cry(John): var(0)$t
```

The reason we need the `_SUBCAT-FRAME` facts here is that without them, both rules (158) and (159) would apply on a transitive verb. If we want to avoid mentioning these facts explicitly, we would need to (a) make sure that each rule consumes some fact (a fact which is not needed by any other rules after the subcat rules), and (b) order the subcat rules so that those which rely on more arguments appear before those that rely on fewer. For example, the modified (159) would appear before (158), and if the first of these applies, it will consume the `SUBJ` fact and prevent the second from applying. This solution is not ideal because it introduces an order dependency between the rules and makes them less declarative. We will get rid of the `_SUBCAT-FRAME` facts in a better way in section 4.4.4.

The glue specification can be defined more easily based on the f-structure level: All the relevant lexical information appears in the f-structures, and we do not need to deal with the C-structure and the projection. We can define the specification declaratively as follows:

(162)

F-structure		Glue statement
$(f\ \text{PRED}) = w \wedge (f\ \text{NTYPE NSYN}) = \textit{proper}$	$\Leftrightarrow$	$w : {f_\sigma}^e$
$(f\ \text{PRED}) = w \wedge (f\ \text{VTYPE}) \wedge (f\ \text{SUBJ}) = g$	$\Leftrightarrow$	$w : {g_\sigma}^e \multimap {f_\sigma}^t$
$(f\ \text{PRED}) = w \wedge (f\ \text{VTYPE}) \wedge (f\ \text{SUBJ}) = g \wedge (f\ \text{OBJ}) = h$	$\Leftrightarrow$	$w : {g_\sigma}^e \multimap {h_\sigma}^e \multimap {f_\sigma}^t$

This specification should be seen as implicitly quantifying over the f-structures f, g, and h. The statement $(f\ \text{VTYPE})$ says that the f-structure f has a non-empty feature VTYPE. Using the rewrite system, this specification would be written as the following rules:

(163)
```
+PRED(%FNode,%Pred), path(%F,NTYPE,NSYN,proper)
==> %Pred : %FNode$e

+PRED(%FNode,%Pred), +VTYPE(%FNode,%%), +SUBJ(%FNode,%FSubj),
_SUBCAT-FRAME(%Fnode,V-SUBJ)
==> %Pred : %FSubj$e -> %FNode$t

+PRED(%FNode,%Pred), +VTYPE(%FNode,%%),
+SUBJ(%FNode,%FSubj), +OBJ(%FNode,%FObj),
_SUBCAT-FRAME(%Fnode,V-SUBJ-OBJ)
==> %Pred : %FSubj$e -> %FObj$e -> %FNode$t
```

The `path` predicate here says that the value of the `NSYN` field of the `NTYPE` field of the f-structure `%F` is `proper`. It is defined as follows:

```
path(%node,%rel1,%val) :=
     +qp(%rel1,[%node,%val]).
path(%node,%rel1,%rel2,%val) :=
     +qp(%rel1,[%node,%node1]),
     +qp(%rel2,[%node1,%val]).
path(%node,%rel1,%rel2,%rel3,%val) :=
     +qp(%rel1,[%node,%node1]),
     +qp(%rel2,[%node1,%node2]),
     +qp(%rel3,[%node2,%val]).
```

(`+qp(%F,[%A1,...,%An])` expresses quantification over predicates `%F(%A1,...,%An)`.)

As you can see, the specification in (163) is a bit shorter than (157)-(159). It has the additional advantage that it can deal with phrasal verbs, e.g. "pick up". The terminals for the sentence "John picked it up" will contain the facts `terminal(..,'pick',..)` and `terminal(..,'up',..)`, which we would have to combine ourselves. However, this is already done for us by the XLE. The F-structure for that sentence will contain the fact `'PRED'(var(0),'pick#up')`. The third rule of (163) would then be able to produce the fact:

(164)
```
pick#up : var(1)$e -> var(2)$e -> var(0)$t
```

There is a lot of repetition in the definitions of verb schemas here – we would need one per subcategorization frame. I will improve on this in section 4.4.4 after I introduce events into glue semantics.

The rest of this chapter is dedicated to going through some important linguistic constructions and discussing their implementation in glue semantics.

4.4 Events and VP Modifiers

The treatment here is based on (Fry, 1999b,a) with some modifications: I use the new glue notation rather than the old one like he does, and I turn all the thematic roles into glue-modifiers, not just VP modifiers.

4.4.1 Event Semantics

Consider:

(165) a. $[[\text{John}]_2$ buttered $[\text{the toast}]_3]_1$
b. $[[\text{John}]_2$ $[\text{quickly}]_5$ buttered $[\text{the toast}]_3$ $[\text{with the knife}]_4]_1$

If we analyze the verb in (165)a as we did above, then once it combines with its arguments "John" and "the toast", it would produce a statement $butter(john, the(toast)) : \mathbf{1}^t$. There would be no good place where additional modifiers as in (165)b could join. We do not want to assume different versions of the verb "butter" that only differ in the number of modifiers they accept, because we want (165)a to automatically logically follow from (165)b by virtue of their meanings.

This issue is handled in the literature by using a Davidsonian (Davidson, 1967) or neo-Devidsonian representations of events (Parsons, 1990; Landman, 2000). According to this, a verb's semantics, after all its arguments are combined with it, is a property of events. For example, in (165)a, the property of events is (166)a, and in (165)b is it (166)b.

(166) a. $\lambda e.butter(e) \wedge agent(e, john) \wedge theme(e, the(toast))$
b. $\lambda e.butter(e) \wedge agent(e, john) \wedge theme(e, the(toast)) \wedge with(e, the(knife)) \wedge quickly(e)$

Now it is easy to obtain (166)b from (166)a by assigning the VP modifiers in (165)b the meanings in (167), and applying these meanings on (166)a.

(167) a. $\lambda P \lambda e.P(e) \wedge with(e, the(knife))$
b. $\lambda P \lambda e.P(e) \wedge quickly(e)$

In a declarative sentence, the event property is eventually closed off by an existential quantifier $\exists e$ at the sentential level. Notice that the sentence obtained by existentially closing (166)b *logically entails* the sentence obtained by existentially closing (166)a, as desired.[3]

4.4.2 Implementation of Verbs

In addition to the sort e of domain-individuals, we now also have a sort ev of events. The language L_0^{MR} from chapter 2 needs to be extended. The domain D_M is now divided into two parts: D_M^e, the domain of individuals, and D_M^{ev}, the domain of events. The language L^{MR} is now made a sorted language, which means that every variable and symbol has a sort attached to it (e, ev, or more complex sorts based on these), and a formula is well-formed only if combinations of symbols obey these sorts.

To implement in glue semantics the idea of VP-modifiers mentioned above, we need to investigate the type expressions of the meaning terms above, and then assign glue categories to them. The type expression that is assigned to the verb "butter" is shown in:

(168) $\lambda x \lambda y \lambda e.butter(e) \wedge agent(e,x) \wedge theme(e,y) : e \rightarrow e \rightarrow ev \rightarrow t$

When we assign glue categories, the 'e' terms correspond as usual to the places of the subject and object NPs in the syntactic structure. What about the 'ev'? There is no place in the syntactic structure that corresponds to the event variable. We had the same issue with nouns in section 3.3.2, and the solution here is the same: introduce a sub-label on the sentence's glue category $\mathbf{1}$. We will call it v for "(event) v(ariable)". To parallel the treatment of nouns, we also introduce a sub-label r for the 't' in (168). The category $\mathbf{1}_r^t$ is different from plain $\mathbf{1}^t$. The former corresponds to the result of the event predicate, while the latter to the meaning of the sentence after existential closure. The resulting statements are:

(169) [[John]$_2$ buttered [the toast]$_3$]$_1$

a. $johh : \mathbf{2}^e$

b. $the(toast) : \mathbf{3}^e$

c. $\lambda x \lambda y \lambda e.butter(e) \wedge agent(e,x) \wedge theme(e,y) : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}_v^{ev} \rightarrow \mathbf{1}_r^t$

d. $\lambda P.exists(\lambda e.P(e)) : (\mathbf{1}_v^{ev} \rightarrow \mathbf{1}_r^t) \rightarrow \mathbf{1}^t$

[3] In some cases of an embedded clause such as "John cried when Mary left", the embedded event property may not be existentially closed. See (Francez and Pratt, 1997).

When (169)c combines with (169)a,b we get the event property (170)a. When that combines with (169)d, we get (170)b.

(170) a. $\lambda e.butter(e) \wedge agent(e, john) \wedge theme(e, the(toast)) : \mathbf{1}_v^{ev} \rightarrow \mathbf{1}_r^t$
b. $exists(\lambda e.butter(e) \wedge agent(e, john) \wedge theme(e, the(toast))) : \mathbf{1}^t$

The above assignments can be obtained by revising the third rule in (163) to:

(171)
```
+PRED(%FNode,%Word), +VTYPE(%FNode,%%),
+SUBJ(%FNode,%FSubj), +OBJ(%FNode,%FObj),
_SUBCAT-FRAME(%Fnode,V-SUBJ-OBJ)
==> %Word : %FSubj$e -> %FObj$e -> VAR(%FNode)$ev -> RESTR(%FNode)$t
```

and adding:

(172)
```
+CLAUSE-TYPE(%F,decl)
==> P\exists(P) : (VAR(%F)$ev -> RESTR(%F)$t) -> %F$t
```

Note that this second rule could not be as easily defined at the lexical level since it depends on a syntactic feature.

4.4.3 Implementation of VP Modifiers

The next thing to do is to assign category expressions to the VP modifiers in (167):

(173) a. $\lambda P \lambda e.P(e) \wedge with(e, the(knife)) : (ev \rightarrow t) \rightarrow ev \rightarrow t$
b. $\lambda P \lambda e.P(e) \wedge quickly(e) : (ev \rightarrow t) \rightarrow ev \rightarrow t$

We factor this out as follows. The basic meanings of the PP and AdvP are:

(174) a. $\lambda e.with(e, the(knife)) : ev \rightarrow t$
b. $\lambda e.quickly(e) : ev \rightarrow t$

and (173) is obtained by applying on (174) the following general meaning term for intersective semantics:

(175) $\lambda Q \lambda P \lambda x.P(x) \wedge Q(x) : (a \rightarrow t) \rightarrow (a \rightarrow t) \rightarrow a \rightarrow t$

where the sort a subsumes both e and ev. Because the label names (including `VAR` and `RESTR`) are the essential parts of glue categories whereas sorts are just decorations for readability, we will simplify from now on and just use `e` instead of `ev`.

Adverb

Thus, we implement the following rule for an adverb:

```
adverb(%Pred,%%,%S,sadv)  ==> %Pred : %S$t -> %S$t.
adverb(%Pred,%F,%VP,vpadv) ==> glue_event_adverb(%Pred,%F,%VP).
```

Here, `adverb` and `glue_event_adverb` are macros that will be defined below. The first rule deals with an adverb that modifies a sentence, as in "Possibly, John left", which yields the result $possibly(left(john))$.[4] I show it here for the sake of completeness. The rule of interest in this section is the second rule for an adverb modifying a VP. The macro definitions are:

```
adverb(%Pred,%F,%Modified,%Type) :=
        +ADV-TYPE(%F,%Type),
        +PRED(%F,%Pred),
        modifier_of(%F,%Modified).

modifier_of(%Node,%Modified) :=
        +in_set(%Node,%Set),
        +ADJUNCT(%Modified,%Set).
```

`ADV-TYPE` may have two possible values: `vpadv` if the adverb modifies a VP, and `sadv` if it modifies a sentence. `modifier_of` is used because the information about the adverb appears in a F-structure which is a member of a set `%Set` which is the value of the `ADJUNCT` field:

(176) John left quickly.

$$\begin{bmatrix} \text{PRED} & \text{LEAVE}\langle \boxed{1} \rangle \\ \text{SUBJ} & \boxed{1}\begin{bmatrix} \text{PRED} & \text{'JOHN'} \end{bmatrix} \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{'QUICKLY'} \\ \text{ADV-TYPE} & \text{VPADV} \end{bmatrix} \right\} \end{bmatrix}$$

Here are the macro definitions for the RHS of the adverb rule:

```
glue_event_adverb(%Pred,%F,%Modified) :=
        [adv,%Pred] : pred_expr(%F) ,
        glue_verb_modifier(%Modified,%F).
```

[4] Here `t` is the type of propositions rather than truth values. See section 2.10.3 under "Intensionality."

We use here `[adv,%Pred]` rather than just `%Pred` as the basic meaning term merely for convenience, to mark adverbs as such in the meaning term.

```
glue_verb_modifier(%V_Node,%Mod_Node) :=
        intersective(%%) : pred_expr(%Mod_Node) -> verb_mod_expr(%V_Node).

intersective(%%) :=
        Q\P\z\[and,[P,z],[Q,z]] .

pred_expr(%Node) :=
        VAR(%Node)$e -> RESTR(%Node)$t.

verb_mod_expr(%V_Node) :=
        verb_pred(%V_Node) -> verb_pred(%V_Node).

verb_pred(%V_Node) :=
        pred_expr(%V_Node).
```

For the sake of completeness, here is what the rule definition macro-expands to:

```
+ADV-TYPE(%F,vpadv), +PRED(%F,%Pred),
+in_set(%F,%Set), +ADJUNCT(%VP,%Set)
==>
[adv,%Pred] : VAR(%F)$e -> RESTR(%F)$t ,
Q\P\z\[and,[P,z],[Q,z]] : (VAR(%F)$e -> RESTR(%F)$t) ->
        (VAR(%VP)$e -> RESTR(%VP)$t) -> VAR(%VP)$e -> RESTR(%VP)$t .
```

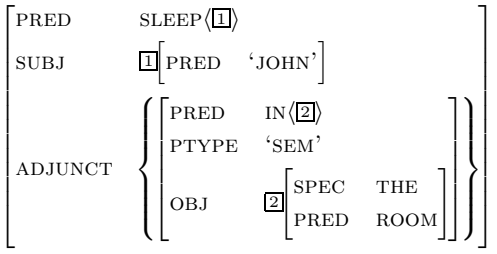
This rule yields two statements, as in (174)b and (175), which combine with each other (thanks to sharing the glue category `%F`) to form a VP-modifier which then combines with the verb. In "book notation" this rule says:

(177) VP adverb m contributes:
$m : {l_v}^e \rightarrow {l_r}^t$
$\lambda Q \lambda P \lambda x.P(x) \wedge Q(x) : ({l_v}^e \rightarrow {l_r}^t) \rightarrow ({k_v}^e \rightarrow {k_r}^t) \rightarrow {k_v}^e \rightarrow {k_r}^t$
where l is my label and k is the label of the VP that I modify

Preposition

The case of a prepositional phrase modifying a VP is very similar:

(178) John slept in the room.



The rules are simile to those for adverbs. The difference is that the base meaning requires an argument (the NP argument of the preposition). Separating the base meaning from the intersection operator is conceptually simpler and easier to define. It also allows the base meaning itself to be modified, as we will see in example (259) in section 6.4.3. (The `-OBL(%%,%Node)` line prevents this rule from firing when the PP is an oblique argument of the verb. See next section.)

```
preposition(%Node), +PRED(%Node,%Prep), +OBJ(%Node,%ObjArg),
        -OBL(%%,%Node)
        ==> glue_preposition_base(%Prep,%ObjArg,%Node).

preposition(%F), modifier_of(%F,%Verb), verb(%Verb)
        ==> glue_verb_modifier(%Verb,%F) .

glue_preposition_base(%stem,%OB,%F) :=
        Y\X\[[preposition,%stem],X,Y] : %OB$e -> pred_expr(%F) .

preposition(%Node) :=
        +PTYPE(%Node,sem).

verb(%Node) :=
        +VTYPE(%Node,%%).
```

4.4.4 More Concise Implementation of Verb Arguments

If we define a separate rule for each verb subcategorization frame as in (171), there will be a lot of repetitive work. For example, intransitive, transitive, and di-transitive verbs

in active voice all share the facts that their first argument is the agent and their result is an event predicate. A good grammar engineering practice is to have just one general treatment for verbs that takes care of all possibilities.

Moreover, to increase robustness of the system, we do not want it to fail on a verb that is used with an unexpected set of arguments for which the glue specification does not include an appropriate glue statement. We want our glue semantics statements to work with any given parse, and we do not want to duplicate all the subcategorization frames in the XLE's lexicon. It is enough to know that the parser produces a F-structure and to collect all the verb's arguments and modifiers as they appear in that structure.

We can achieve this by implementing the treatment of verb arguments as we did for VP-modifiers, i.e. with glue categories of the form $X \rightarrow X$ (which we call *glue modifiers*). Thus, instead of one rule like (171) for each subcategorization frame, we can use the following rules. For the basic meaning of the verb, we write:

```
verb(%Pred,%F) ==> [verb,%stem] : verb_pred(%F).

verb(%Pred,%F) :=
        verb(%F),  +PRED(%F,%Pred).
```

(Note that macro definitions are identified by their name and number of arguments, so this definition of the macro `verb` is different from the unary one shown previously.) For adding a glue modifier for one verb argument, here are the definitions:

```
verb_arg(%F,%A,%ThemArg), role_type(%ThemArg,%RoleType)
==> glue_verb_arg(%F,%A,%ThemArg,%RoleType).

glue_verb_arg(%F,%A,%role,%argtype) :=
        x\P\e\[and,[P,e], xle_gf(e,%role,x)] :
                %A$%argtype -> verb_mod_expr(%F) .

verb_arg(%VNode,%ArgNode,%ThemArg) :=
        verb(%VNode),
        +qp(%SynRole,[%VNode,%ArgNode]),
        is_arg(%VNode,%ArgPos,%ArgNode),
        +PASSIVE(%VNode,%Passive),
        thematic_arg(%SynRole,%ArgPos,%Passive,%ThemArg).
```

```
is_arg(%VNode,%ArgPos,%ArgNode) :=
        arg(%VNode,%ArgPos,%ArgNode).
```

These definitions use two "tables" that give information about mapping from syntactic roles to pseudo-thematic roles:

```
|- thematic_arg(SUBJ,1,-,subj).
|- thematic_arg(OBL-AG,1,+,subj).
|- thematic_arg(OBJ,2,-,obj).
|- thematic_arg(SUBJ,2,+,obj).
|- thematic_arg(OBJ-TH,%%,%%,objth).
|- thematic_arg(OBL,%%,%%,obl).
|- thematic_arg(XCOMP,%%,%%,xcomp).
|- thematic_arg(COMP,%%,%%,comp).
```

The `thematic_arg` predicate has four arguments: grammatical function, argument number, binary flag for passive (`+` or `-`), and the result role. Thus, an argument of a verb will be assigned a pseudo-thematic role of `subj` in two cases: in both, it has to be the first thematic argument of the verb (this is why we use the `is_arg` fact in the definition of `verb_arg`), but in the first case, it is under the grammatical function `SUBJ` of an active verb, while in the second case, it is under the grammatical function `OBL-AG` (marked by the preposition *by*) of a passive verb. Thus, "John" will get a `subj` role in both "John saw Mary" and "Mary was seen by John". Similarly, in both these sentences, "Mary" will get an `obj` role.

Relying on the `arg` fact also guarantees we consider only thematic arguments. For example, in:

(179) a. It bothers me that John left.

b. $\exists e.bother(e) \wedge obj(e, me) \wedge comp(e, \exists e'.leave(e') \wedge subj(e', john))$

the verb *bother* has three syntactic arguments, but the subject is not a thematic argument.

The `role_type` predicate defines which roles get a sort `e` and which sort `t`. Only `xcomp` and `comp` get a type `t`, the rest get a type `e`:

```
role_type(%Role,%RoleType) :=
     ( ( ({%Role = xcomp} | {%Role = comp}), {%RoleType = t} ) |
       ( {\+ %Role = xcomp}, {\+ %Role = comp}, {%RoleType = e} ) ).
```

('\+' is negation in Prolog notation). As mentioned in section 2.10.3 under "Intensionality", we need to regard the sort `t` here that is used for the clausal complements `comp` and `xcomp` as a type for propositions rather than simple truth values. However, for the purpose of discussing structural composition, we can ignore this point.

4.4.5 Special Cases of Verb Arguments

The verb argument rule above is implemented as a default rule by virtue of the fact that the `arg` fact does not have a `+` in front of it.[5] That means this fact is consumed once the rule fires. We can write additional rules for more specific cases and place them before the default rule. If such a more specific rule fires and consumes the `arg` fact then later rules, and the default rule in particular, will not fire.[6]

Oblique Arguments

Such more specific rules are needed for some cases. One case is the treatment of oblique arguments. For example:

(180) John gave a book to Mary.

$$\begin{bmatrix} \text{PRED} & \text{GIVE}\langle \boxed{1}, \boxed{2}, \boxed{3} \rangle \\ \text{SUBJ} & \boxed{1}\begin{bmatrix} \text{PRED} & \text{'JOHN'} \end{bmatrix} \\ \text{OBJ} & \boxed{2}\begin{bmatrix} \text{SPEC} & \text{'A'} \\ \text{PRED} & \text{BOOK} \end{bmatrix} \\ \text{OBL} & \boxed{3}\begin{bmatrix} \text{PRED} & \text{TO}\langle \boxed{4} \rangle \\ \text{OBJ} & \boxed{4}\begin{bmatrix} \text{PRED} & \text{'MARY'} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

In the semantic form, we do not want to say of the event `e` that `xle_gf(e,obl,mary)` but instead we want `xle_gf(e,obl(to),mary)`. So we have a more specific rule, which we place before the default rule:

```
verb_arg_obl(%F,%A,%Prep) ==> glue_verb_arg_obl(%F,%A,%Prep).
```

[5] In section 11.1, we will modify the definition of `is_arg` slightly.

[6] This is one of the very few places where the order of rules matters. However, the glue semantics specification is still by-and-large order independent. We limit order dependencies to very local areas of the rules set, such as to the set of rules for verb arguments. Thus, it is still easy to predict what the results will be by looking at the rules, and we do not get unlimited dependencies.

```
verb_arg_obl(%VNode,%ArgNode,%Prep) :=
        verb_arg(%VNode,%Node2,obl),
        +OBJ(%Node2,%ArgNode),
        +PRED(%Node2,%Prep).

glue_verb_arg_obl(%F,%A,%Prep) :=
        x\P\e\[and,[P,e], xle_gf(e,obl(%Prep),x)] :
             %A$e -> verb_mod_expr(%F).
```

Other special cases are intensional verbs (e.g. "Ed seeks a unicorn" – under the *de dicto* interpretation, we want to prevent the quantifier from taking scope outside the verb), and equi verbs, as I explain next.

Raising and Equi Verbs

The following two sentences seem to have the same structure at first sight:

(181) a. John seemed to leave.
b. John tried to leave.

However, there are several differences between them. One syntactic difference is:

(182) a. It seemed that John left.
b. * It tried that John left.

There are other differences. This led syntacticians to analyze the two verbs differently. Verbs like *seem* are called *raising verbs* while verbs like *try* are called *equi verbs* (or *control verbs*). In the LFG syntactic analysis of both verbs, the subject of the main clause's f-structure is structure-shared with the subject of the embedded clause's f-structure (Bresnan, 1982; Dalrymple, 2001, ch.12). The only difference between the two is that "try" selects for a thematic subject whereas "seem" selects for a non-thematic subject, in order to explain the data in (182). This difference is represented in LFG by having the thematic subject inside the angled brackets of the PRED, while the non-thematic subject is outside these brackets. Thus the f-structures for (181) are:

(183) a.
$$\boxed{1}\begin{bmatrix} \text{PRED} & \text{`SEEM}\langle\text{XCOMP}\rangle\text{SUBJ'} \\ \text{SUBJ} & \boxed{2}\begin{bmatrix}\text{PRED} & \text{`JOHN'}\end{bmatrix} \\ \text{XCOMP} & \boxed{3}\begin{bmatrix}\text{PRED} & \text{`LEAVE}\langle\text{SUBJ}\rangle\text{'} \\ \text{SUBJ} & \boxed{2}\end{bmatrix} \end{bmatrix}$$

b. $$\boxed{1}\begin{bmatrix} \text{PRED} & \text{`TRY}\langle\text{SUBJ, XCOMP}\rangle\text{'} \\ \text{SUBJ} & \boxed{2}\begin{bmatrix}\text{PRED} & \text{`JOHN'}\end{bmatrix} \\ \text{XCOMP} & \boxed{3}\begin{bmatrix}\text{PRED} & \text{`LEAVE}\langle\text{SUBJ}\rangle\text{'} \\ \text{SUBJ} & \boxed{2}\end{bmatrix}\end{bmatrix}$$

Asudeh (2002), following previous accounts in the literature regarding semantic differences between raising and equi verbs, provides a glue semantics analysis that extends this LFG syntactic analysis. His account assigns the following meanings to (181):

(184) a. $seem(leave(john))$
b. $try(john, \lambda x.leave(x))$

The meaning in (184)a is also the meaning for (182)a. The reason for using (184)b rather than

(185) $try(john, leave(john))$

is that (185) leads to false inference patterns. Consider:

(186) a. John tried to leave.
b. Andrew tried everything that John tried.
c. ⇒ Andrew tried to leave.
and not: Andrew tried for John to leave.

If (186)b is represented as $\forall x.try(john, x) \rightarrow try(andrew, x)$, then using (184)b gives the correct conclusion $try(andrew, \lambda x.leave(x))$, whereas using (185) gives the wrong conclusion $try(andrew, leave(john))$.

Following this analysis, the glue statements for (183) (ignoring event structure) are:

(187) a. $john : \mathbf{2}^e$
$\lambda S.seem(S) : \mathbf{3}^t \rightarrow \mathbf{1}^t$
$leave : \mathbf{2}^e \rightarrow \mathbf{3}^t$
$\Rightarrow\ seem(leave(john))$

b. $john : \mathbf{2}^e$
$\lambda x \lambda P.try(x, P) :\rightarrow \mathbf{2}^e \rightarrow (\mathbf{2}^e \rightarrow \mathbf{3}^t) \rightarrow \mathbf{1}^t$
$leave : \mathbf{2}^e \rightarrow \mathbf{3}^t$
$\Rightarrow\ try(john, leave)$

Note that even though *leave* may combine with *john* in both cases, this combination leads to a complete derivation only with the raising verb and not the equi verb.

For a raising verb, we do not need to change anything in the rules of the previous section. Since the subject is a non-thematic argument in (183)a, the verb argument rule

will not fire for it. It will only fire for the subject of the embedded clause, and it will use the correct glue category for the subject because of the structure-sharing of 2. In contrast, the subject of both TRY and LEAVE is thematic in (183)b, so if we do not make any changes, the verb argument rule will fire for both of them, and we will end up with a set of premises that has no solution because two premises try to consume the same subject resource. What we need is to add the following rule before the default verb argument rule:

```
verb_arg_equi(%F,%SNode,%XNode) ==> glue_verb_arg_equi(%F,%SNode,%XNode).

verb_arg_equi(%VNode,%SNode,%XNode) :=
     verb_arg(%VNode,%XNode,xcomp),
     +SUBJ(%XNode,%SNode),
     +arg(%VNode,%%,%SNode).

glue_verb_arg_equi(%F,%SNode,%XNode) :=
      R\P\e\[and,[P,e], xle_gf(e,xcomp,R)] :
        (%SNode$e -> %XNode$t) -> verb_mod_expr(%F) .
```

This rule creates a premise which adds the information `xle_gf(e,xcomp,R)` to the event predicate, where `R` will have the meaning of a predicate that is obtained from the meaning of the XCOMP f-structure after abstracting over its SUBJ. The definition of `verb_arg_equi` calls `verb_arg` that we saw before, but only if `%SNode`, the `SUBJ` of the XCOMP, is also one of the thematic arguments of `%VNode`.

As far as consumption of `arg` facts: First, the `arg` connecting the `xcomp` to `%VNode` is consumed as usual inside the call to `verb_arg`, thus preventing the default argument rule from applying on the `xcomp` argument – this is the difference compared to a raising verb, which is handled by the default rule. Second, the `+arg(%VNode,%%,%SNode)` line in the definition is prefixed by '`+`' so that it will not be consumed, and the default rule will apply on it. This argument need not be the subject of the equi verb. For example:

(188) John asked Mary to leave.
$ask(john, mary, \lambda x.leave(x))$

Here, the subject of the XCOMP is structure-shared with the object of the equi verb *ask* (John asked Mary that she would leave, not that he would leave).

Finally, it might seem that we need to add the line `arg(%XNode,%%,%SNode)` to the definition above so that this fact will be consumed, to prevent the default rule from applying

in the embedded clause, causing it to consume its subject argument. But this is a confusion between consumption of facts by the rewrite system and consumption of glue categories by glue premises. We do want the default rule to apply inside the XCOMP f-structure and to add a piece `xle_gf(e,subj,x)`. In other words, we want to get the following set of facts:

(189) $john : \mathbf{2}^e$

$try : \mathbf{1}_v^e \rightarrow \mathbf{1}_r^t$
$exists : (\mathbf{1}_v^e \rightarrow \mathbf{1}_r^t) \rightarrow \mathbf{1}^t$
$\lambda x \lambda P \lambda e.P(e) \wedge subj(e,x) : \mathbf{2}^e \rightarrow (\mathbf{1}_v^e \rightarrow \mathbf{1}_r^t) \rightarrow \mathbf{1}_v^e \rightarrow \mathbf{1}_r^t$
$\lambda R \lambda P \lambda e.P(e) \wedge xcomp(e,R) : (\mathbf{2}^e \rightarrow \mathbf{3}^t) \rightarrow (\mathbf{1}_v^e \rightarrow \mathbf{1}_r^t) \rightarrow \mathbf{1}_v^e \rightarrow \mathbf{1}_r^t$

$leave : \mathbf{3}_v^e \rightarrow \mathbf{3}_r^t$
$exists : (\mathbf{3}_v^e \rightarrow \mathbf{3}_r^t) \rightarrow \mathbf{3}^t$
$\lambda x \lambda P \lambda e.P(e) \wedge subj(e,x) : \mathbf{2}^e \rightarrow (\mathbf{3}_v^e \rightarrow \mathbf{3}_r^t) \rightarrow \mathbf{3}_v^e \rightarrow \mathbf{3}_r^t$

The second batch can combine to yield (190)a, and the third to yield (190)b. Combining these two with $john : \mathbf{2}^e$ gives the final (190)c.

(190) a. $\lambda x \lambda R.exists(\lambda e.try(e) \wedge subj(e,x) \wedge xcomp(e,R)) : \mathbf{2}^e \rightarrow (\mathbf{2}^e \rightarrow \mathbf{3}^t) \rightarrow \mathbf{1}^t$
b. $\lambda x.exists(\lambda e.leave(e) \wedge subj(e,x)) : \mathbf{2}^e \rightarrow \mathbf{3}^t$
c. $exists(\lambda e.try(e) \wedge subj(e,john) \wedge xcomp(e, \lambda x.exists(\lambda e'.leave(e') \wedge subj(e',x)))) : \mathbf{1}^t$

4.4.6 Dealing with Duplicate Derivations

If we have n verb modifiers, including arguments and VP-modifiers, we will get $n!$ possible answers because the modifiers can combine in all possible permutations. For example, "John saw Mary" would give glue premises with two modifiers of category $(\mathbf{0}_v^e \rightarrow \mathbf{0}_r^t) \rightarrow \mathbf{0}_v^e \rightarrow \mathbf{0}_r^t$. The glue prover would find two final meaning terms for the sentence:

```
exists(e\[and,[and,[see,e],xle_gf(e,subj,john)],xle_gf(e,obj,mary)])
exists(e\[and,[and,[see,e],xle_gf(e,obj,mary)],xle_gf(e,subj,john)])
```

To prevent this spurious ambiguity, we mark certain glue statements with the flag `[noscp]` for "non-scoping". For example:

```
glue_verb_arg(%F,%A,%role,%argtype) :=
        X\P\e\[and,[P,e], xle_gf(e,%role,X)] :
                %A$%argtype -> verb_mod_expr(%F) : [noscp].
```

```
preposition(%F), modifier_of(%F,%Verb), verb(%Verb)
     ==> glue_verb_modifier(%Verb,%F) : [noscp].
```

In section 6.7.1, I will show how the glue prover can take these flags into account and produce just one of the $n!$ forms.

4.4.7 Interaction with Quantifiers

This analysis interacts correctly with quantifiers. Since the second argument of a quantifier modifies the t resource of a clause, and not of the event predicate, that means the event variable in the clause's meaning must first be existentially closed before that meaning is given as argument to the quantifier. For example:

(191) Every man saw some woman. ('every' wide scope reading:)
$every(man, \lambda x.some(woman, \lambda y.exists(\lambda e.see(e) \wedge subj(e,x) \wedge obj(e,y))))$

In particular, we do not get an incorrect form such as:

(192) $exists(\lambda e.every(man, \lambda x.some(woman, \lambda y.see(e) \wedge subj(e,x) \wedge obj(e,y))))$

4.5 Determiners

As explained in section 3.3.2, I will not use variable glue categories in the premises for quantifiers, but instead a description that might be instantiated by more than one glue category.

4.5.1 Nouns

A common noun is implemented simply by:

```
common_noun(%stem,%F) ==> glue_normal_noun(%stem,%F).

common_noun(%Stem,%F) :=
        path(%F,NTYPE,NSYN,common),
        +PRED(%F,%Stem).

glue_normal_noun(%stem,%F) :=
        %stem : noun_pred(%F) .
```

```
noun_pred(%N_Node) :=
        pred_expr(%N_Node).
```

4.5.2 Dominance

We want to know all the glue categories of clauses that dominate a quantified NP. For that, we calculate domination facts. Immediate domination is defined by:

```
f_str(%F) := +PRED(%F,%%) | COORD(%F,%%).

im_dominating(%A,%B) :=
        f_str(%A),
        f_str(%B),
        ( (path(%A,%Rel,%B), -({%Rel=in_set})) |
           modifier_of(%B,%A) ).

im_dominating(%A,%B) ==> im_dominated(%B,%A).
```

The first few lines prevent facts about things that are not F-structures (e.g. C-structure facts) from participating in this calculation. The transitive closure of domination is defined by:

```
+im_dominated(%A,%B) ==> dominated(%A,%B).
-dominated(%A,%C), +dominated(%A,%B), +im_dominated(%B,%C)
       *=> dominated(%A,%C).
```

In contrast to rules written with `==>` which do not apply to their own output, a rule written with `*=>` allows limited recursion and applies to its own output. In order to prevent an infinite loop, we need to add the `-dominated(%A,%C)` condition so that the rule creates the fact only if it does not yet exist.

Given the sentence:

(193) $[[[\text{John}]_1$ and $[\text{Mary}]_2]_3$ think that $[[\text{Bill}]_5$ left$]_4]_0$

with f-structure numbers as indicated, the rules above will create the facts

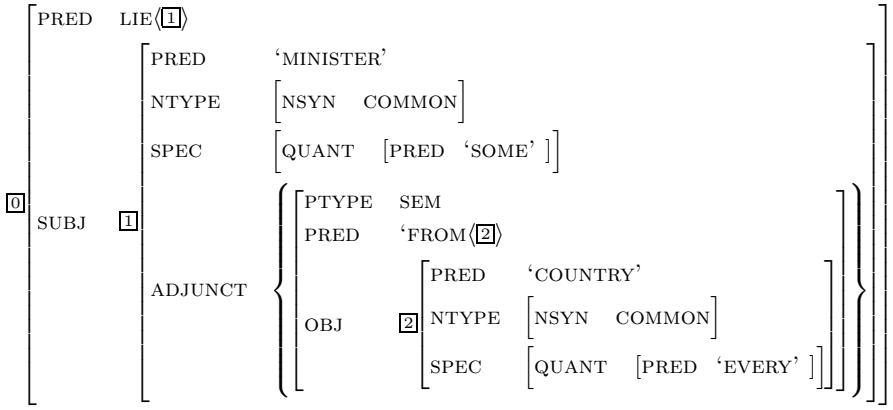
(194) `dominated(var(3),var(0))`
`dominated(var(1),var(3))`

```
dominated(var(1),var(0))
dominated(var(2),var(3))
dominated(var(2),var(0))
dominated(var(4),var(0))
dominated(var(5),var(4))
dominated(var(5),var(0))
```

4.5.3 Basic Quantifiers

Here is an example F-structure with quantifiers:

(195) Some minister from every country lied.



Here is the rule for quantifiers, and more macro definitions:

```
float_quant(%Quant,%NP,%LandingGlueCat)
==> quant_sem(%Quant,%NP,%LandingGlueCat).

quant_sem(%Quant,%Node,%LandingGlueCat) :=
        R\S\quant(%Quant,R,S) : quant_expr(%Node,%LandingGlueCat) .

quant_expr(%Node,%LandingGlueCat) :=
        noun_pred(%Node) -> (%Node$e -> %LandingGlueCat$t) -> %LandingGlueCat$t .

float_quant(%Name,%NP,%LandingGlueCat) :=
        consume_spec(%NP,%%,%SPEC,QUANT,%Name),
        -NUMBER(%Spec,%%),   "prevent numeric quantifiers"
        det_quant_float(%NP,%LandingGlueCat).
```

```
consume_spec(%NP,%SynNum,%Spec,%Rel,%Name) :=
          +NTYPE(%NP,%%),
          +SPEC(%NP,%Spec),
          +NUM(%NP,%SynNum),
          path(%Spec,%Rel,%X),
          PRED(%X,%Name).

det_quant_float(%NP,%LandingGlueCat) :=
         +dominated(%NP,%Landing),
         legal_det_landing(%Landing),
         landing_glue_cat(%Landing,%LandingGlueCat).

legal_det_landing(%F) :=
          is_clause(%F) |
          (is_np(%F), modifier_of(%A,%F), +PTYPE(%A,sem)).   "only relative nouns"

is_clause(%F) := +VTYPE(%F,%%).
is_np(%F) := +NTYPE(%F,%%).

landing_glue_cat(%F,%LGT) :=
            (is_clause(%F) , {%LGT = %F}) |
            (is_np(%F)     , {%LGT = RESTR(%F)}) .
```

The crucial point in `consume_spec` is that the `PRED` fact is consumed. If there is only one clause that dominates a NP, then just one glue premise will be created for the quantifier. But if there is more than one, then a separate glue premise will be created for each such clause. For example, in (195), two premises will be created for the quantifier *every*:

```
R\S\quant(every,R,S) : (VAR(var(2))$e -> RESTR(var(2))$t) ->
                              (var(2)$e -> var(0)$t) -> var(0)$t
R\S\quant(every,R,S) : (VAR(var(2))$e -> RESTR(var(2))$t) ->
                              (var(2)$e -> RESTR(var(1))$t) -> RESTR(var(1))$t
```

The only difference between them is their landing site, either the main clause, `var(0)`, or the noun, `RESTR(var(1))`. The quantifier rule above can apply on two different sets of premises. The sets are equal except for one difference: one of them includes the fact `dominated(var(2),var(0))` and the other `dominated(var(2),var(1))`. Furthermore,

both sets include the resource-sensitive fact `PRED(..,every)` (it comes from the last line of `consume_spec`, and it is resource-sensitive because it is not prefixed by a '`+`' there). When the same rule tries to consume a resource-sensitive fact in two different ways, it means we have an ambiguity regarding the output of the rewiring process. Therefore, the packed rewriting system *automatically* creates a new binary split in the choice space, adding two new subcontexts. Each of the two quantifier premises shown above will be put in exactly one of those subcontexts. The rest of the facts in the rewriting process will not be affected by this split and will remain just once in their respective contexts. In section 7.5.2, we will see how the glue prover will be able to take advantage of this.[7]

4.5.4 Other Determiners

In section 10.2, we will see how to define a specification for numeric quantifiers.

For the definite determiner *the*, we do not want a floating behavior, but instead a result such as:

(196) John saw the swedish man.
$exists(\lambda e.see(e) \wedge subj(e, john) \wedge obj(e, the(\lambda x.swedish(x) \wedge man(x))))$

(In the literature, iota ι is used instead of *the*.) For this, we define:

```
single_det(the,%SynNum,%NP) ==> def_det_sem(%SynNum,%NP).

single_det(%Name,%SynNum,%NP) :=
        consume_spec(%NP,%SynNum,%%,DET,%Name).

def_det_sem(%Card,%Node) :=
     R\iota(x\[and,[R,x],[cardinality,x,%Card]]) :
           noun_pred(%Node) -> %Node$e .
```

Here we actually put the syntactic number into the semantic representation since this is a useful piece of information for anaphora resolution.

[7] In `legal_det_landing`, we allow only relative nouns and not any nouns. For example, in "Some minister who read every report lied", *every* can scope over *read*, but should not be allowed to scope directly over *minister*.

4.6 Noun Modifiers

4.6.1 Cases

Prepositional Phrases

I already discussed prepositional phrases that modify a VP in section 4.4.3. The treatment of PPs that modify NPs is similar. For example:

(197) the man in the room

$$\begin{bmatrix} \text{PRED} & \text{MAN} \\ \text{SPEC} & \text{THE} \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{IN}\langle \boxed{1} \rangle \\ \text{PTYPE} & \text{`SEM'} \\ \text{OBJ} & \boxed{1}\begin{bmatrix} \text{SPEC} & \text{THE} \\ \text{PRED} & \text{ROOM} \end{bmatrix} \end{bmatrix} \right\} \end{bmatrix}$$

The rules we need to add:

```
preposition(%F), modifier_of(%F,%Noun), -verb(%Noun)
      ==> glue_noun_modifier(%Noun,%F).

glue_noun_modifier(%N_Node,%Mod_Node) :=
      intersective(%%) :
           pred_expr(%Mod_Node) -> noun_mod_expr(%N_Node).

noun_mod_expr(%N_Node) :=
        noun_expr(%N_Node) -> noun_expr(%N_Node).
```

Adjectives

Just as for VP modifiers, we separate the basic meaning of an intersective adjective from the intersection operator:

(198) [the [Swedish]$_2$ man]$_1$
$the : (\mathbf{1}_v^e \rightarrow \mathbf{1}_r^t) \rightarrow \mathbf{1}^e$
$man : \mathbf{1}_v^e \rightarrow \mathbf{1}_r^t$
$swedish : \mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$
$\lambda Q \lambda P \lambda x.P(x) \wedge Q(x) : (\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t) \rightarrow (\mathbf{1}_v^e \rightarrow \mathbf{1}_r^t) \rightarrow \mathbf{1}_v^e \rightarrow \mathbf{1}_r^t$
$\Rightarrow the(\lambda x.man(x) \wedge swedish(x)) : \mathbf{1}^e$

The rules:

```
base_adj_simple(%Node,%Pred)
      ==> glue_adj_simple(%Node,%Pred).

+ATYPE(%F,attributive), modifier_of(%F,%Noun)
      ==> glue_noun_modifier(%Noun,%F).

glue_adj_simple(%F,%Pred) :=
        %Pred : pred_expr(%F).

base_adj_simple(%Node,%Pred) :=
        +ATYPE(%Node,%%),
        +PRED(%Node,%Pred),
        -ADJUNCT(%Node,%%).
```

The `-ADJUNCT` line is to indicate this is a simple adjective that does not take arguments, in contrast to comparative adjectives, which will be handled in chapter 11.

Relative Clauses

A relative clause is a sentence that is missing a part of type e, and so it can function as a modifier of a noun. For example:

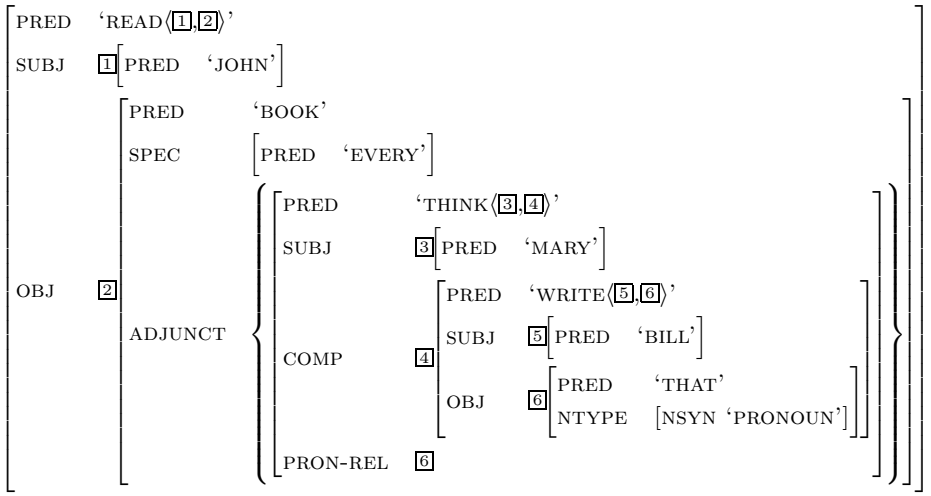
(199) John read every book that Mary wrote.

$$\begin{bmatrix}
\text{PRED} & \text{`READ}\langle\boxed{1},\boxed{2}\rangle\text{'} \\
\text{SUBJ} & \boxed{1}\begin{bmatrix}\text{PRED} & \text{`JOHN'}\end{bmatrix} \\
\text{OBJ} & \boxed{2}\begin{bmatrix}
\text{PRED} & \text{`BOOK'} \\
\text{SPEC} & \begin{bmatrix}\text{PRED} & \text{`EVERY'}\end{bmatrix} \\
\text{ADJUNCT} & \left\{\boxed{5}\begin{bmatrix}
\text{PRED} & \text{`WRITE}\langle\boxed{3},\boxed{4}\rangle\text{'} \\
\text{SUBJ} & \boxed{3}\begin{bmatrix}\text{PRED} & \text{`MARY'}\end{bmatrix} \\
\text{OBJ} & \boxed{4}\begin{bmatrix}\text{PRED} & \text{`THAT'} \\ \text{NTYPE} & [\text{NSYN `PRONOUN'}]\end{bmatrix} \\
\text{PRON-REL} & \boxed{4}
\end{bmatrix}\right\}
\end{bmatrix}
\end{bmatrix}$$

The verb *wrote* expects an object argument. This argument is not supplied in (199), and so "Mary wrote" can have a meaning of type $e \rightarrow t$. This meaning then combines with that of "book", just as the meaning of an adjective would.

The syntax of LFG knows this and supplies us with the necessary information. The missing object has a special f-structure [4] in its place, and the clause "Mary wrote" points to that f-structure through the PRON-REL feature. The dependency between the missing argument and the place where it is "discharged" is unbounded. For example:

(200) John read every book$_t$ that Mary thinks that Bill wrote $\_t$.



Thanks to this information, we can give a glue semantics specification easily:

```
relative_clause(%N_Node,%F_Missing,%F_Clause) ==>
    glue_relc_connector(%N_Node,%F_Missing,%F_Clause).

glue_relc_connector(%N_Node,%F_Missing,%F_Clause) :=
     intersective(%%) :
        (%F_Missing$e -> %F_Clause$t) -> noun_mod_expr(%N_Node).

relative_clause(%N_Node,%F_Missing,%F_Clause) :=
          +PRON-REL(%F_Clause,%F_Missing),
          path(%F_Missing,NTYPE,NSYN,pronoun),
          modifier_of(%F_Clause,%N_Node).
```

This rule says: If there is a F-structure `%F_Clause` which is a modifier of a noun node `%N_Node` and which is connected to a `%F_Missing` F-structure through `PRON-REL`, and that missing F-structure is a pronoun, then create the glue statement with the intersective meaning $\lambda Q \lambda P \lambda x.P(x) \wedge Q(x)$, where P is the meaning of the noun and Q is the meaning

of the relative clause. The type of Q is $e \rightarrow t$, where e corresponds to the missing argument `%F_Missing`, and t corresponds to the relative clause `%F_Clause`. Thus we get in (199):

(201) $book : \mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$
$mary : \mathbf{3}^e$
$write : \mathbf{3}^e \rightarrow \mathbf{4}^e \rightarrow \mathbf{5}^t$
$\lambda Q \lambda P \lambda x.P(x) \wedge Q(x) : (\mathbf{4}^e \rightarrow \mathbf{5}^t) \rightarrow (\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t) \rightarrow \mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$
$\Rightarrow \lambda x.book(x) \wedge write(mary, x) : \mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$

4.6.2 Scope Ambiguities

We need to explain how to deal with scope ambiguities for noun modifiers. We saw that with VP-modifiers, we should allow just one permutation of the modifiers by declaring that they are non-scoping. Here, in contrast, this is true only some of the time. For example, in the NP:

(202) the expensive red house on the street in Seattle

the noun "house" has four modifiers. We want to state scoping constraints on them so that if an adjective A_1 appears in the sentence before adjective A_2 then A_2 must combine with the noun before A_1 does. For post-nominal modifiers M_1 and M_2, the reverse holds, i.e. if M_1 appears before M_2 then M_1 should combine with the noun before M_2. However, there are no constraints on interleaving pre- and post-nominal modifiers.

Fortunately, the XLE gives us facts about the order of modifiers. In any `ADJUNCT` feature which has a set value with at least two elements, we get facts such as `scopes(var(2),var(5))` in the (internal representation of the) F-structure. This fact says that the modifier whose F-structure is `var(2)` appears in the sentence before the one whose F-structure is `var(5)`. The name `scopes` is a misnomer, so we can pick a better name by rewriting it:

```
scopes(%F1,%F2) ==> precedes(%F1,%F2).
```

Now we express scoping constraints by adding `outscopes` facts. To say that the modifier from F-structure `var(4)` can be used only after the modifier from F-structure `var(3)` has already been used, we write the statement: `outscopes(var(4),var(3))`. Thus, if an adjective `%F1` appears before adjective `%F2` then the `%F1` modifier should outscope `%F2`:

```
precedes(%F1,%F2), +ATYPE(%F1,%%), +ATYPE(%F2,%%) ==> outscopes(%F1,%F2).
```

In contrast, scope interactions between adjectives and post-nominal modifiers should be allowed, so we do not force the `outscopes` fact (and we delete the `precedes` fact):

```
precedes(%F1,%F2), +ATYPE(%F1,%%), -ATYPE(%F2,%%) ==> 0.
```

Finally, for everything else, the reverse order is true:

```
precedes(%F1,%F2) ==> outscopes(%F2,%F1).
```

As a bonus, this final rule holds also for verb modifiers. So together with the fact that those modifiers are non-scoping, we will actually get them to appear in the logical form in the same order that they appear in the sentence.

In section 6.7.2, I will explain how to take these `outscopes` facts into account.

Chapter 5

Basic Glue Derivation

In the previous chapter, we saw how to translate F-structures to statements in glue semantics. If there are syntactic ambiguities, we will have a non-trivial choice-space, and the packed F-structure will have some parts relativized to different contexts of the choice-space. Similarly, some of the resulting glue statements will be relativized to different parts of the choice space. This also happens when a quantifier floats to more than one location (section 4.5.3).

Now suppose we take a maximally-specific context in the choice space, and look at all the glue statements there. In this chapter, We will see a basic algorithm that takes this set of statements and calculates all the ways in which they can combine to form meaning representations. If there are more than one resulting representations, they will be output separately, not in a packed way. In the next two chapters, we will see how to revise the algorithm so that we get the results in a packed way.

5.1 Preliminaries

5.1.1 Goal

We are given as input a set Γ of glue semantics statements, where each statement has the form $\psi : A$. The term ψ is a meaning term, and A is called *category* or *type*. The language of the meaning terms is not crucial for most of the computational discussion here, and the language of the category expressions consists of basic categories and of one connector between them, written '$\rightarrow$'. We are also given an atomic *goal category* G, which stands for the semantic projection of the entire syntactic structure of a sentence.

The statements are to be combined with each other as dictated by interpreting the category expressions as formulas of the implicational fragment of linear logic (Girard, 1987), where '$\rightarrow$' is the linear implication.[1] A legal derivation from Γ uses each premise exactly once. We are guaranteed that Γ can yield at least one legal derivation of $\varphi : G$ for some φ (this is guaranteed for any input generated by a legal glue specification). By Lemma 2 below, if $\Gamma \vdash \psi : A$ then $A = G$. We want the computer to find all possible non-equivalent conclusions ψ such that $\Gamma \vdash \psi : G$ (equivalence is in the sense of the lambda calculus, as will be explained below).

5.1.2 Mathematical Foundations

Proof System

We first need to decide on a set of proof operators (rules and possibly axioms). Section 3.4 explained the difference between the "user friendly" version of Glue Semantics, which uses proof rules such as argument exchange and function composition, and the "implementation level" version, which uses only the two rules of function application and abstraction. Because the proof rules in the "user friendly" version are more powerful than in the "implementation level" in the sense that they condense several steps into one, there is usually a proof in the former that is shorter than the shortest proof in the latter. However, for the purpose of automatically searching for a proof, it is not clear a-priori which version would usually support shorter searches for a proof (it depends on issue such as branching factor and depth of search in the search space). In any case, it is probably easier to implement a search proof in the "implementation level" because it has just two simple proof rules.

Here we will call these two rules *implication elimination* (203)a and *implication introduction* (203)b because the right-hand-side of glue statements is what really drives the derivations.

(203) a. $$\frac{\psi : A \qquad \delta : A \rightarrow B}{\delta(\psi) : B}\,\mathrm{E}$$

b. $$\frac{\begin{array}{c}[x : A]_i \\ \vdots \\ \psi : B\end{array}}{\lambda x.\psi : A \rightarrow B}\,\mathrm{I}_i$$

In (203)a, $\psi : A$ is called the *minor* premise and $\delta : A \rightarrow B$ is called the *major* premise. In (203)b, the introduction step is marked with the index of the discharged assumption.

[1] In the linear logic literature, and consequently in the glue semantics literature too, '$\multimap$' is usually used instead of '$\rightarrow$'.

In contrast to classical logic, each assumption must be marked with a *unique* index. A complete derivation must use each input premise exactly once. I already discussed these rules and resource sensitivity in section 3.4.

Equivalent Proofs and Normalization

There is an isomorphism between meaning terms and proofs.[2] We can say that meaning terms faithfully encode proofs because of the following property (Howard, 1980; Gallier, 1993):

- Curry-Howard isomorphism:
  If Γ is a set of glue semantics statements, and $\mathcal{D}_1$ is a derivation of $\psi_1 : G$ that uses each member of Γ exactly once, and $\mathcal{D}_2$ is a derivation of $\psi_2 : G$ that uses each member of Γ exactly once, then $\psi_1 = \psi_2$ iff $\mathcal{D}_1 = \mathcal{D}_2$.

However, there are many meaning terms that really mean the same thing as each other, and they are equivalent in the lambda-calculus under the meaning-preserving α-renaming and β- and η-conversion rules (85)-(87). Even though they have different proofs, we would like to consider those proofs as equivalent.

One step is to stipulate that each assumption $[\nu : A]_i$ uses a fresh variable ν so that we do not need to worry about clashes between variable names and about α-renaming. We consider two proofs that are identical up to α-renaming of bound variables to be *the same* proof.

Regarding the β- and η-conversion rules in the lambda-calculus, they correspond via the Curry-Howard isomorphism to the proof normalization rules (204) and (205).

$$(204) \qquad \dfrac{\dfrac{\begin{matrix}[\nu : A]_i \\ \end{matrix} \quad \begin{matrix}\mathcal{D} \\ \psi : A \to B\end{matrix}}{\psi(\nu) : B}\,\mathrm{E}}{\lambda\nu.\psi(\nu) : A \to B}\,\mathrm{I}_i \qquad \Longrightarrow_\eta \qquad \begin{matrix}\mathcal{D} \\ \psi : A \to B\end{matrix}$$

This rule says that if there is a proof $\mathcal{D}$ that ends with $\psi : A \to B$, then the detour that introduces the assumption $\nu : A$ only to immediately discharge it can be eliminated.

[2] The following discussion draws from (Crouch and van Genabith, 2000). See that paper for more information.

(205)
$$\dfrac{\dfrac{\begin{array}{c}[\nu : A]_i \\ \vdots\,\mathcal{D}_2 \\ \psi : B\end{array}}{\lambda\nu.\psi : A \to B}\,\mathrm{I}_i \qquad \begin{array}{c}\mathcal{D}_1 \\ \varphi : A\end{array}}{(\lambda\nu.\psi)(\varphi) : B} \quad \Longrightarrow_\beta \quad \begin{array}{c}\mathcal{D}_1 \\ \varphi : A \\ \vdots\,\mathcal{D}_2[\varphi/\nu] \\ \psi[\varphi/\nu] : B\end{array}$$

This says: if there is a proof $\mathcal{D}_1$ ending with $\varphi : A$, and a proof $\mathcal{D}_2$ that proves $\psi : B$ given the assumption $\nu : A$, then instead of discharging the assumption and applying the result on φ, combine the two proofs in a way that corresponds to the β-conversion of $(\lambda\nu.\psi)(\varphi)$, i.e. using φ instead of ν in ψ (ν is not free in φ because of the convention above).

As an example, note that given a set of glue semantics statement, there is an infinite number of possible proofs. Given the set $\{f : A \to B,\ a : A\}$, we can have the proofs:

$$\dfrac{a : A \quad f : A \to B}{f(a) : B}\,\mathrm{E}$$

$$\dfrac{a : A \quad \dfrac{\dfrac{[x : A]_1 \quad f : A \to B}{f(x) : B}\,\mathrm{E}}{\lambda x.f(x) : A \to B}\,\mathrm{I}_1}{(\lambda x.f(x))(a) : B}\,\mathrm{E}$$

$$\dfrac{a : A \quad \dfrac{\dfrac{[y : A]_2 \quad \dfrac{\dfrac{[x : A]_1 \quad f : A \to B}{f(x) : B}\,\mathrm{E}}{\lambda x.f(x) : A \to B}\,\mathrm{I}_1}{(\lambda x.f(x))(y) : B}\,\mathrm{E}}{\lambda y.(\lambda x.f(x))(y) : A \to B}\,\mathrm{I}_2}{(\lambda y.(\lambda x.f(x))(y))(a) : B}\,\mathrm{E}$$

and so on. But all these proofs end in meaning terms that are equivalent to $f(a)$ in the lambda-calculus under the β- and η-conversion rules. By the isomorphism, the two right indirect proofs can be shortened using the conversion rules.

A proof on which neither conversion rules can be applied is said to be in $\beta\eta$-normal form. Under the convention of α-equivalence stipulated above, the two important properties of proof normalization using β- and η-conversion are:

- Strong Normalization:
  The application of normalizing conversions will eventually *terminate* to give a normal form proof.

- Church-Rosser property:
  Any proof has a *unique* normal form. In other words, the order in which the normalization conversions are applied to a derivation does not affect the final result.

Thus, the normalization rules divide the proofs into equivalence classes. Since we are interested only in distinct (non-equivalent) meanings, it is enough to consider just one representative from each equivalence class, and we might as well take the unique proof in

$\beta\eta$-normal form in each equivalence class. If two forms ψ and φ are in the same equivalence class, we can write $\psi \equiv_{\beta\eta} \varphi$.

The last important fact is that a finite set of glue premises always yields at most a finite number of normal proofs. The intuition is that a finite number of premises can lead to only a finite number of non-equivalent trees where the premises act as leaves. This will be shown in section 5.3.4.

The upshot is that if the computer is given a set of glue premises and is asked to find all possible unique meanings entailed from this set, it need not consider the infinitely many possible proofs from that set. It only needs to consider a finite number of normal proofs.

5.1.3 Some Lemmas

The *polarity* of a category is defined as follows. The category B in $A \rightarrow B$ has the same polarity as $A \rightarrow B$ while A has the reverse polarity. The polarity of a premise is considered to be + while that of a goal is −.

Example: $((((A^+ \rightarrow B^-)^- \rightarrow C^+)^+ \rightarrow D^-)^- \rightarrow E^+)^+$.

Lemma 1 *In linear logic, if $A_1, \ldots, A_n \vdash B$ then for every atomic category, the number of its positive occurrences is equal to the number of its negative occurrences in $A_1^+, \ldots, A_n^+, B^-$.*

Proof: by induction. See a similar claim in (van Benthem, 1986, p.133).

Example: $A,\ A \rightarrow B,\ B \rightarrow A,\ A \rightarrow C \vdash C$. In Γ, B appears once positively and once negatively, A appears twice each, and C appears just once positively, but it also appears once negatively as the goal.

Lemma 2 *In linear logic, if $\Gamma \vdash A$ and $\Gamma \vdash B$ and A is atomic then $A = B$.*

Proof: Immediately follows from Lemma 1.

5.2 Basic Algorithm

Hepple (1996) presents an algorithm for finding all resulting meaning terms (in some normal form) from a bag of expression-category pairs in implicational linear logic, and so it is suitable as a starting point for what we need here. Hepple (1998) extends this algorithm to the multiplicative fragment of linear logic (with the tensor $\otimes$) and applies it to glue

semantics in particular. That extension will not concern us here because the multiplicative fragment is needed only for one line of research that aims to treat anaphora using the resource sensitivity of glue semantics (Dalrymple et al., 1999; Crouch and van Genabith, 1999; Dalrymple, 2001, ch.11), but this idea has been argued against in (Kokkonidis, 2005) (and see also section 8.2).

I review here Hepple's algorithm, with slight modifications. No proof has been published for the correctness of Hepple's algorithm, so I will provide such a proof here.

5.2.1 First-Order Deduction

If all the premises are first-order, i.e. their right-hand-side is either atomic or of the form $A_1 \to A_2 \to \ldots \to A_n \to B$, where B and each A_i are atomic, and the goal category G is atomic, then it is very simple to compute the derivations using a dynamic programming algorithm similar to a chart parser for a context-free grammar. One difference is that we do not have a grammar with syntax rules but instead we are given rule-like formulas as assumptions.

In a chart parser, each constituent has a linear span. This is used to guarantee that the parser does not use any word more than once in a parse tree and that a tree for the entire sentence does not skip any word. In a glue derivation there is no notion of a linear span, but there is still the requirement that each premise must be used at most once in each sub-derivation and exactly once in any *overall* derivation. We can achieve this by adding a unique index to each premise, and writing a set of indices next to each derived formula, indicating all the premises that were used in its derivation.

An *element* has the form $\psi : A : S$, where $\psi : A$ is a glue semantics statement and S is a set of indices. The algorithm starts by initializing the agenda: each statement $\psi : A$ is assigned a unique index i, and the element $\psi : A : \{i\}$ is put on the agenda. We also initialize two charts: one will include elements with atomic categories, and the other with non-atomic categories. The atomic chart is a data-structure that maps a category to a list of elements with that category, while the non-atomic chart maps an atomic category A to a list of elements whose category is $A \to B$.

The algorithm iterates as follows. As long as the agenda is not empty, an element ξ is removed from it. If its category A is atomic, we try to combine ξ with each element in the non-atomic chart whose category is $A \to B$ (for some B). Conversely, if ξ's category

is a non-atomic $A \to B$, we try to combine ξ with each element in the atomic chart with category A. The combination succeeds iff the sets of indices used in the derivation of the two elements are disjoint:

$$(206) \qquad \frac{\psi : A : S_1 \qquad \delta : A \to B : S_2}{\delta(\psi) : B : S_1 \cup S_2}\,\mathrm{E} \qquad \text{provided } S_1 \cap S_2 = \emptyset$$

If a combination succeeds, we add the result to the agenda. After all such combinations are done, we finally add ξ to the appropriate chart.

The algorithm terminates when the agenda is empty. Chart elements whose set of indices includes all the indices of all the premises represent a complete derivation. They are the output of the algorithm.

As a simple example, if the premises are $m : a,\ p : a \to a,\ q : a \to b$, and we assign them the indices $1, 2, 3$ respectively, then the content of the charts when the algorithm terminates will be:

(207)

atomic chart		non-atomic chart	
$a:$	$m : a : \{1\}$	$a:$	$p : a \to a : \{2\}$
	$p(m) : a : \{1, 2\}$		$q : a \to b : \{3\}$
$b:$	$q(m) : b : \{1, 3\}$		
	$q(p(m)) : b : \{1, 2, 3\}$		

5.2.2 Higher-Order Deduction

Additional Assumptions Needed

Consider the following higher-order deduction:

$$(208) \qquad \dfrac{\dfrac{\dfrac{\dfrac{[z : Z]_1 \quad a : Z \to A}{a(z) : A}\,\mathrm{E} \qquad b : A \to B}{b(a(z)) : B}\,\mathrm{E}}{\lambda z.b(a(z)) : Z \to B}\,\mathrm{I}_1 \qquad c : (Z \to B) \to C}{c(\lambda z.b(a(z))) : C}\,\mathrm{E}$$

This is higher-order because of the non-first-order statement $(Z \to B) \to C$. If we had the statement $Z \to B$ as an assumption, we could combine the two directly. But in general, we might not have $Z \to B$ as an assumption but obtain it from a subproof. Since this subproof ends with a non-atomic category, its last step must be an implication introduction step, and therefore the subproof must have used an assumption $[Z]$. Thus, if we are given a non-atomic glue premise $B \to A$ where $B = C_1 \to \ldots \to C_n \to D$, then the proof

might need to use assumptions $[C_1], \ldots, [C_n]$, prove D from them (and other premises), and discharge these assumptions just before combining B with $B \rightarrow A$.

If any of the C_i are themselves non-atomic, the proof may need to rely on more assumptions. For example, suppose we have the two assumptions $d : ((A \rightarrow B) \rightarrow C) \rightarrow D$ and $e : (((A \rightarrow B) \rightarrow C) \rightarrow D) \rightarrow E$. The $\beta\eta$-normal proof is:

$$(209)\quad \frac{d : ((A \rightarrow B) \rightarrow C) \rightarrow D \quad e : (((A \rightarrow B) \rightarrow C) \rightarrow D) \rightarrow E}{e(d) : E}\ \mathrm{E}$$

Now consider instead this derivation:

$$(210)\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : A]_3 \quad [p : A \rightarrow B]_1}{p(x) : B}\ \mathrm{E}}{\lambda x.p(x) : A \rightarrow B}\ \mathrm{I}_3 \qquad [q : (A \rightarrow B) \rightarrow C]_2}{q(\lambda x.p(x)) : C}\ \mathrm{E}}{\lambda p.q(\lambda x.p(x)) : (A \rightarrow B) \rightarrow C}\ \mathrm{I}_1 \qquad d : ((A \rightarrow B) \rightarrow C) \rightarrow D}{d(\lambda p.q(\lambda x.p(x))) : D}\ \mathrm{E}}{\lambda q.d(\lambda p.q(\lambda x.p(x))) : ((A \rightarrow B) \rightarrow C) \rightarrow D}\ \mathrm{I}_2 \qquad e : ..}{e(\lambda q.d(\lambda p.q(\lambda x.p(x)))) : E}\ \mathrm{E}$$

This is a very roundabout way of proving E. There are η-conversions that can be done: $e(\lambda q.d(\lambda p.q(\lambda x.p(x)))) \Rightarrow_\eta e(\lambda q.d(\lambda p.q(p))) \Rightarrow_\eta e(\lambda q.d(q)) \Rightarrow_\eta e(d)$. But in general, a proof similar to (210) is required: Suppose that in addition to the two assumptions with terms d and e, we also have $m : A \rightarrow A$. Then the beginning of (210) should be modified to:

$$(211)\quad \dfrac{\dfrac{\dfrac{[x : A]_3 \quad m : A \rightarrow A}{m(x) : A}\ \mathrm{E} \qquad [p : A \rightarrow B]_1}{p(m(x)) : B}\ \mathrm{E}}{\lambda x.p(m(x)) : A \rightarrow B}\ \mathrm{I}_3$$

The final term is $e(\lambda q.d(\lambda p.q(\lambda x.p(m(x)))))$, and it cannot be η-converted further. We also need to have the three additional assumptions in (210).

The general case relies on the polarity of the categories. The above discussion shows that a proof may rely on an additional assumption $[A]_i$ (where A is not necessarily atomic) if A appears as a maximal positive subformula of another formula, i.e. in a context like $(A^+ \rightarrow B^-)^-$ but not $(B^- \rightarrow A^+)^+$.

The Compilation Procedure

We can rely on this and compile the initial premises to give a possibly larger set of first-order premises. The procedure is shown in Figure 5.1. We apply *compile-glue-formula* on

function compile-glue-formula($\psi : X$)
  $\langle Y, \Gamma \rangle$:= compile-pos(X)
  return $\{\psi : Y : \{i\}\} \cup \Gamma$ for a fresh index i

function compile-pos(X) where X atomic
  return $\langle X_{[]}, \emptyset \rangle$

function compile-pos($X \rightarrow Y$)
  $\langle \delta, \Delta \rangle$:= compile-pos(Y)
  $\langle \gamma, \Gamma \rangle$:= compile-neg(X) (γ must be atomic)
  return $\langle \gamma \rightarrow \delta, \Gamma \cup \Delta \rangle$

function compile-neg(X) where X atomic
  return $\langle X_{[]}, \emptyset \rangle$

function compile-neg($X \rightarrow Y$)
  $\langle Y'_L, \Delta \rangle$:= compile-neg(Y) (Y' must be atomic)
  $\langle X', \Gamma \rangle$:= compile-pos(X)
  return $\langle Y'_{[i|L]}, \{v_i : X' : \{i\}\} \cup \Gamma \cup \Delta \rangle$ for a fresh index i †

† $[i|L]$ means a list whose head is i and whose tail is L.

Figure 5.1: The compilation procedure

each initial premise. When this function applies on $\psi : A$, it adds a new unique index and compiles A, thereby possibly creating new formulas. The functions *compile-pos* and *compile-neg* each gets a category and returns two values: a compiled category and a set of additional elements (that are a result of breaking down some categories).

An *indexed category* is an atomic category X associated with a list of indices L, and this is written as X_L. An indexed category X_L with a non-empty index list L is the result of compiling a non-atomic category with negative polarity. The members of L are the indices of new added assumptions which must be used in the proof of X (the semantic term of those added assumptions is a variable).

The function *compile-pos* takes a category of the form $C_1 \rightarrow \ldots \rightarrow C_n \rightarrow D$ where D is atomic and essentially returns the same category, except that each C_i is compiled so that if it is non-atomic, it is converted to an atomic category with indices.

The function *compile-neg* takes a category of the form $C_1 \rightarrow \ldots \rightarrow C_n \rightarrow D$ where D is atomic and basically returns D_L, where L includes new indices for the omitted C_i's. It

input	compiled
$m : B \to A$	$m : B \to A : \{1\}$
$m : (C \to B) \to A$	$m : B_{[2]} \to A : \{1\}$ $v_2 : C : \{2\}$
$m : (D \to C \to B) \to A$	$m : B_{[2,3]} \to A : \{1\}$ $v_2 : D : \{2\}$ $v_3 : C : \{3\}$
$m : (D \to E) \to (C \to B) \to A$	$m : E_{[2]} \to B_{[3]} \to A : \{1\}$ $v_2 : D : \{2\}$ $v_3 : C : \{3\}$
$m : ((D \to C) \to B) \to A$	$m : B_{[2]} \to A : \{1\}$ $v_2 : D \to C : \{2\}$
$m : (((E \to D) \to C) \to B) \to A$	$m : B_{[2]} \to A : \{1\}$ $v_2 : D_{[3]} \to C : \{2\}$ $v_3 : E : \{3\}$

Figure 5.2: Compilation output

also compiles each C_i, and with the complied category C'_i, it creates an additional element with that category and a new variable as meaning term.

Figure 5.2 shows some examples of the output of the compilation procedure.

The Modified Chart Algorithm

Now we need to modify the rule in (206) to take the indexed categories into account:

(212) $$\frac{\psi : A : S_1 \qquad \delta : A_L \to B : S_2}{\delta(\lambda v_{i_1}, \ldots, \lambda v_{i_n}.\psi) : B : S_1 \cup S_2}\,\mathrm{E} \qquad \begin{array}{l}\text{provided } S_1 \cap S_2 = \emptyset \text{ and } L \subseteq S_1 \\ \text{and } L = [i_1, \ldots, i_n]\end{array}$$

If A_L is an indexed category with an empty L, we just have the rule as in (206) (when $n = 0$, we say that $\lambda v_{i_1}, \ldots, \lambda v_{i_n}.\psi$ is just ψ). If L is not empty, then the two elements may be combined only if all the indices in L that A_L requires were indeed used in the derivation of $\psi : A$, as indicated by their presence in S_1. If that is the case, the variables which are the semantic terms associated with these indices should first be abstracted over, according to the order of L, and only then δ should be applied.

Here is an example of how the algorithm works (we omit empty list subscripts).

(213) Input premises:

$a : Z \to A, \quad b : A \to D \to B, \quad c : D \to (D \to Z \to B) \to C, \quad d : D$

Compiled premises:

1. $a : Z \rightarrow A : \{1\}$
2. $b : A \rightarrow D \rightarrow B : \{2\}$
3. $c : D \rightarrow B_{[4,5]} \rightarrow C : \{3\}$
4. $v_4 : D : \{4\}$
5. $v_5 : Z : \{5\}$
6. $d : D : \{6\}$

Derivation:

7.	$a(v_5) : A : \{1,5\}$	[1+5]
8.	$b(a(v_5)) : D \rightarrow B : \{1,2,5\}$	[2+7]
9.	$b(a(v_5))(v_4) : B : \{1,2,4,5\}$	[8+4]
10.	$b(a(v_5))(d) : B : \{1,2,5,6\}$	[8+6]
11.	$c(v_4) : B_{[4,5]} \rightarrow C : \{3,4\}$	[3+4]
12.	$c(d) : B_{[4,5]} \rightarrow C : \{3,6\}$	[3+6]
13.	$c(d)(\lambda v_4 \lambda v_5 . b(a(v_5))(v_4)) : C : \{1,2,3,4,5,6\}$	[12+9]

When $c : D \rightarrow (D \rightarrow Z \rightarrow B) \rightarrow C$ is compiled, it is converted to assumption 3, and the additional assumptions 4 and 5 are also created. Locally, the algorithm does not know that 10 and 11 cannot be used in a global derivation. Only 9 and 12 can be combined to create the final result: their sets of indices are disjoint, and moreover, the requirement $B_{[4,5]}$ in 12 is satisfied by the presence of $4, 5$ in the index set in 9. In contrast, although the sets of indices of 10 and 11 are disjoint, the requirement $B_{[4,5]}$ in 11 is not satisfied by 10 since 4 is a member of the index set in 10. Thus, the wrong derivation that ends with $c(v_4)(\lambda v_4 \lambda v_5 . b(a(v_5))(d)) : C$ (where v_4 has an unbound occurrence) is prevented.

Non-Atomic Goal

What if the goal category is not atomic? In practice, a glue lexicon is always going to yield a set of premises that has an atomic goal category, representing the location in the syntactic structure that corresponds to the entire sentence. Nonetheless, for the sake of completeness, we can consider non-atomic goals too.

One option is to compile the goal by using *compile-neg* to get a few more compiled assumptions and a new atomic goal G that expects these assumptions $i_1, \ldots, i_n$. The chart algorithm will produce an element with goal G which used all the assumptions, except that $v_1, \ldots, v_n$ have not been abstracted yet (since that must happen at the end of the expected proof). For example:

(214) Input premises:

$a : A \rightarrow B \rightarrow C$

Goal category:

$B \rightarrow A \rightarrow C$

Compiled premises:

1. $a : A \rightarrow B \rightarrow C : \{1\}$
2. $v_2 : A : \{2\}$ (from goal)
3. $v_3 : B : \{3\}$ (from goal)
   (compiled goal: $C_{[3,2]}$)

Derivation:

4. $a(v_2) : B \rightarrow C : \{1, 2\}$ [1+2]
5. $a(v_2)(v_3) : C : \{1, 2, 3\}$ [4+3]

Final step:

abstract on $[3, 2]$ in that order (because of the compiled goal $C_{[3,2]}$)

Result: $\lambda v_3 \lambda v_2 . a(v_2)(v_3) : B \rightarrow A \rightarrow C$

This shows that $a : A \rightarrow B \rightarrow C \vdash \lambda v_3 \lambda v_2 . a(v_2)(v_3) : B \rightarrow A \rightarrow C$ (this is the EXCH rule). If we do not have the goal category in advance, the final step could calculate all $n!$ possible answers that result from permutations of the n compiled-out assumptions.

Another option is to reduce non-atomic goals to the atomic case. If the goal category is a non-atomic $A = B_1 \rightarrow \ldots B_n \rightarrow C$, create a new atomic goal G and add $goal : A \rightarrow G$ to Γ. In this way, A will be with a negative polarity and will be compiled in the same way as above. The final chart element will have a meaning term $goal(\psi)$, where $\psi = \lambda v_{i_1} \ldots \lambda v_{i_n} . \varphi$ such that $i_1, \ldots, i_n$ are the indices of $B_1 \ldots B_n$ after they are compiled out of A by the compilation algorithm. The final answer of the algorithm should then be $\psi : A$.

Comparison to (Hepple, 1996)

The algorithm above is slightly different from that of Hepple (1996). There, the compilation process converts every premise to be in η-long form. For example, line 2 of (213) would be $\lambda x_1 \lambda x_2 . b(x_1)(x_2) : A \rightarrow D \rightarrow B : \{2\}$. Also, the compilation process, and not the extended application rule (212), is what introduces the abstraction over variables. Thus, line 3 of (213) would be $\lambda x_1 \lambda x_2 . c(x_1)(\lambda v_4 \lambda v_5 . x_2) : D \rightarrow B_{[4,5]} \rightarrow C : \{3\}$. Note that the meaning term here has an abstraction over v_4 and v_5, expecting x_2 to be filled with a meaning term that has these two variables occurring free in it. Then, Hepple's application rule needs to

perform substitutions of the variables: if $\lambda v.\psi : A_L \rightarrow B$ is to be combined with $\varphi : A$, the result is $\psi[\varphi /\!\!/ v]$ using direct subsitution, i.e. where φ is substituted for v in ψ but without renaming variables. Such renaming would normally be done during β-reduction in the lambda-calculus in order to avoid accidental binding of variables, but in Hepple's version, such "accidental" binding is in fact desired, e.g. the binding of v_4 and v_5 in the meaning term that is substituted for x_2 in $\lambda x_1 \lambda x_2 . c(x_1)(\lambda v_4 \lambda v_5 . x_2)$ above. The advantage of my version is that the extra variables in the η-long forms are not needed, nor is the expensive recursive substitution procedure.

Another difference between the algorithms is that Hepple's particular formulation of the compilation procedure (his Figure 1) uses Prolog-like code, and crucially depends on a programming construct that is very specific to Prolog: the ability to pass around an unbound Prolog variable, and bind it somewhere inside a nested recursive call or sometime in a future call (notice the statement $u := \lambda v.x$ in his definition of neg). In fact, his program depends on this feature not only during the compilation procedure but also after the chart algorithm finishes running. The x_0 variable at the top of his Figure 1 is unified with the variable u during the compilation, but only after the chart algorithm terminates does u end up getting a concrete meaning term, which is the resulting meaning term of the algorithm. I personally find it easier to understand code not written in this way, and my version above is in "standard pseudo-code", as nothing hinges on the particular Prolog capability.[3]

5.2.3 Extracting a Proof

Hepple does not explain how to convert a chart derivation to the corresponding natural deduction proof, so I show here how to do this. We define a translation tr from chart elements to proofs in linear logic (I will prove below that these are indeed legal proofs). The recovered proofs will not be of much interest to us in this dissertation because we are interested only in the resulting meaning terms, but they will be useful in proving the correctness of the algorithm in section 5.3.

First, if $v_i : X' : \{i\}$ was introduced by *compile-neg* in the compilation algorithm (i.e. this is an element that was "complied out" of an original member of Γ), then we say $\sigma(i) = X'$. Let Δ be the set of such indices.

[3]This does not contradict the fact that I used Prolog to implement my version of the algorithm as well as the algorithms in the next chapters.

Next, we define an operator μ on compiled glue categories, which returns the corresponding original category.

(215) $\mu(A) = A$ for an atomic A

$\mu(A_{[i_1,\ldots,i_n]}) = \mu(i_1) \to \ldots \to \mu(i_n) \to A$

$\mu(A \to B) = \mu(A) \to \mu(B)$

where, for an index i:

$\mu(i) = \mu(\sigma(i))$

Here is an example:

(216) $\Gamma = \{d : ((A \to B) \to C) \to D),\ e : (((A \to B) \to C) \to D) \to E\}$

This compiles into:

$d : C_{[2]} \to D : \{1\}$

$v_2 : A \to B : \{2\}$

$e : D_{[4]} \to E : \{3\}$

$v_4 : B_{[5]} \to C : \{4\}$

$v_5 : A : \{5\}$

$\sigma(1)$, $\sigma(3)$ are undefined (since they are not compiled out elements)

$\sigma(2) = A \to B$

$\sigma(4) = B_{[5]} \to C$

$\sigma(5) = A$

$\mu(A) = A$, $\mu(B) = B$, ...

$\mu(B_{[5]}) = \mu(\sigma(5)) \to B = \mu(A) \to B = A \to B$

$\mu(C_{[2]}) = \mu(\sigma(2)) \to C = \mu(A \to B) \to C = (\mu(A) \to \mu(B)) \to C = (A \to B) \to C$

$\mu(D_{[4]}) = \mu(\sigma(4)) \to D = \mu(B_{[5]} \to C) \to D = (\mu(B_{[5]}) \to \mu(C)) \to D = ((A \to B) \to C) \to D$

Thus, the original category corresponding to $D_{[4]}$ is $((A \to B) \to C) \to D$.

For an atomic chart element $\gamma = \langle h,\ \psi : A : \{i\},\ \langle\rangle\rangle$, i.e. one that has no parents but is a member of the initial agenda, we define:

(217) $$tr(\gamma) = \begin{cases} \psi : \mu(A) & \text{if } i \notin \Delta \\ [\psi : \mu(A)]_i & \text{if } i \in \Delta \ \text{ (and then } \psi = v_i) \end{cases}$$

The idea is that (what corresponds to) an original premise is just put as a leaf in the proof, whereas (what corresponds to) an element that was compiled out of an original premise is put in the proof as an assumption, indexed with i.

For a non-atomic chart element $\gamma = \langle h,\ \delta(\psi) : B : S,\ \langle h_1, h_2\rangle\rangle$ with parents $\gamma_1 = \langle h_1,\ \psi : A : S_1,\ T_1\rangle$ and $\gamma_2 = \langle h_2,\ \delta : A \to B : S_2,\ T_2\rangle$, we define:

(218) $$tr(\gamma) = \dfrac{tr(\gamma_1) \quad tr(\gamma_2)}{\delta(\psi) : \mu(B)}\,\mathrm{E}$$

This is because the subproofs $tr(\gamma_1)$ and $tr(\gamma_2)$ can be combined using the elimination rule to create $\delta(\psi)$ whose category is the original category of B.

More generally, if $\gamma = \langle h,\ \delta(\lambda v_{i_1} \ldots \lambda v_{i_n}.\psi) : B : S,\ \langle h_1, h_2\rangle\rangle$, with the parents $\gamma_1 = \langle h_1,\ \psi : A : S_1,\ T_1\rangle$ and $\gamma_2 = \langle h_2,\ \delta : A_{[i_1,\ldots,i_n]} \to B : S_2,\ T_2\rangle$, we define:

(219) $$tr(\gamma) = \dfrac{\dfrac{\dfrac{tr(\gamma_1)}{\lambda v_{i_n}.\psi : \mu(i_n) \to A}\,\mathrm{I}_{i_n}}{\lambda v_{i_1} \ldots \lambda v_{i_n}.\psi : \mu(i_1) \to \ldots \to \mu(i_n) \to A}\,\mathrm{I}_{i_1} \qquad tr(\gamma_2)}{\delta(\lambda v_{i_1} \ldots \lambda v_{i_n}.\psi) : \mu(B)}\,\mathrm{E}$$

Here is an example, based on (213):

(220) The algorithm-style proof:

$$\dfrac{\dfrac{c : D \to B_{[4,5]} \to C : \{3\} \quad d : D : \{6\}}{c(d) : B_{[4,5]} \to C : \{3,6\}} \qquad \dfrac{\dfrac{b : A \to D \to B : \{2\} \qquad \dfrac{a : Z \to A : \{1\} \quad v_5 : Z : \{5\}}{a(v_5) : A : \{1,5\}}}{b(a(v_5)) : D \to B : \{1,2,5\}} \qquad v_4 : D : \{4\}}{b(a(v_5))(v_4) : B : \{1,2,4,5\}}}{c(d)(\lambda v_4 \lambda v_5.b(a(v_5))(v_4)) : C : \{1,2,3,4,5,6\}}$$

(221) The translation to a linear logic proof:

$$\dfrac{\dfrac{c : D \to (D \to Z \to B) \to C \quad d : D}{c(d) : (D \to Z \to B) \to C}\,\mathrm{E} \qquad \dfrac{\dfrac{\dfrac{\dfrac{b : A \to D \to B \qquad \dfrac{a : Z \to A \quad [v_5 : Z]_5}{a(v_5) : A}\,\mathrm{E}}{b(a(v_5)) : D \to B} \qquad [v_4 : D]_4}{b(a(v_5))(v_4) : B}\,\mathrm{E}}{\lambda v_5.b(a(v_5))(v_4) : Z \to B}\,\mathrm{I}_5}{\lambda v_4.\lambda v_5.b(a(v_5))(v_4) : D \to Z \to B}\,\mathrm{I}_4}{c(d)(\lambda v_4 \lambda v_5.b(a(v_5))(v_4)) : C}\,\mathrm{E}$$

All proofs that are obtained in this way have the property that the set of their discharged assumptions is precisely the set of assumptions that is introduced by the compilation algorithm, and furthermore, each time a category $C_1 \to \ldots \to C_n \to A$ is combined with category $(C_1 \to \ldots \to C_n \to A) \to B$, the former is obtained from a sequence of n introduction rules. Below we will call this form of proof Hepple Normal Form (HNF).

Note that HNF is not the same as η-long form. In the latter, every term with n arguments is expanded to have n trailing lambdas, but this is true in HNF only for embedded categories. Thus, (222)a is both in $\beta\eta$-normal form and in HNF whereas (222)b is in η-long form and not in HNF.

(222) a. $$\frac{a : A \quad b : A \rightarrow B}{b(a) : B}\,\mathrm{E}$$ b. $$\frac{a : A \qquad \dfrac{\dfrac{[x : A]_1 \quad b : A \rightarrow B}{b(x) : B}\,\mathrm{E}}{\lambda x.b(x) : A \rightarrow B}\,\mathrm{I}_1}{(\lambda x.b(x))(a) : B}\,\mathrm{E}$$

5.3 Analysis

5.3.1 Termination

When an element e_1 is taken from the agenda, the index set of any element e_2 that is derived from e_1 and is put on the agenda is a strict superset of the index set of e_1. Since the set of indices of any element cannot be larger than the finite set of all indices, this process must eventually terminate.

5.3.2 Correctness

Theorem 1 *(Soundness) Let Γ be a set of glue statements that are input to the algorithm. If $\psi : G$ is among the final output of the algorithm then $\Gamma \vdash_{LL} \psi : G$.*

Proof: If $\psi : G$ is among the final output of the algorithm then there is some chart element $\gamma = \langle h,\ \psi : G : S,\ T\rangle$ where S includes the indices of all premises, including all compiled premises. We show that $tr(\gamma)$ as defined in section 5.2.3 is indeed a legal LL proof, and so $\Gamma \vdash_{LL} \psi : G$.

We prove the following claims by induction on the size of S:

(223) For every chart element $\gamma = \langle h,\ \phi : A : S,\ T\rangle$:
- a. $tr(\gamma)$ is a legal LL subproof.
- b. The bottom of $tr(\gamma)$ is $\phi : \mu(A)$.
- c. The top (i.e. tree leaves) of $tr(\gamma)$ are exactly the members of $\{\mu(i) \mid i \in S\}$

Proof: When $|S| = 1$, γ is atomic, and the claims directly follow from the definition (217). The only point that needs clarification is that a legal full proof in LL may not have any undischarged assumptions, but a subproof may. So for an atomic chart element which was compiled out from an original premise, its legal LL subproof has the form $[\phi : \mu(A)]_i$.

Suppose we know the claims are true for all γ with $|S| < k$, and now we look at a non-atomic γ with $|S| = k$. For the first case of a non-atomic γ defined in (218), we know that $1 \leq |S_1| < k$ and similarly with $|S_2|$. Hence, B.I.H., we know that $tr(\gamma_1)$ and $tr(\gamma_2)$

are legal subproofs and that the bottom of $tr(\gamma_1)$ is $\psi : \mu(A)$ and the bottom of $tr(\gamma_2)$ is $\delta : \mu(A \to B)$. But $\mu(A \to B) = \mu(A) \to \mu(B) = A \to \mu(B)$ because A must be atomic and so $\mu(A) = A$ (if A were not atomic, then $A = X \to Y$ but then $(X \to Y) \to B$ would not be first order, in contradiction to the fact that all categories in chart elements are first order, as a result of the output of the compilation stage). So we get that:

$$tr(\gamma) = \quad \dfrac{\begin{array}{cc}\vdots & \vdots \\ \psi : A & \delta : A \to \mu(B)\end{array}}{\delta(\psi) : \mu(B)}\ \mathrm{E}$$

As you can see, the last step of the proof is a legal application of the elimination rule. We said that $tr(\gamma_1)$ and $tr(\gamma_2)$ are also legal subproofs. Furthermore, because $S_1 \cap S_2 = \emptyset$ (this condition must have been met for γ to be created from γ_1 and γ_2), the set of leaves of $tr(\gamma_1)$ is disjoint from that of $tr(\gamma_2)$. Hence, the whole $tr(\gamma)$ is a legal subproof. Moreover, the second and third properties of (223) also hold.

For the more general non-atomic case defined in (219), the difference in the argument would be that the bottom of $tr(\gamma_2)$ is $\delta : \mu(A_{[i_1,\ldots,i_n]} \to B)$. But $\mu(A_{[i_1,\ldots,i_n]} \to B) = \mu(A_{[i_1,\ldots,i_n]}) \to \mu(B) = (\mu(i_1) \to \ldots \to \mu(i_n) \to A) \to \mu(B)$. The final elimination step is therefore justified, and the second and third properties of (223) hold as well. It only remains to justify the introduction steps $I_{i_n}, \ldots, I_{i_1}$ in (219). For all $1 \leq j \leq n$, the introduction step I_{i_j} is legal because:

1. The assumption i_j can be discharged because it is a leaf of $tr(\gamma_1)$. We know this from property (223)c and the fact that $\{i_1, \ldots, i_n\} \subseteq S_1$ (this latter condition must have been met for γ to be created from γ_1 and γ_2).

2. The assumption i_j has not been previously discharged in $tr(\gamma_1)$. To see this, note a few facts. (i) Because the compilation procedure assigns unique indices, if $i_j \in I$ for some index set I in some category X_I then $X_I = A_{[i_1,\ldots,i_n]}$. Further, (ii) i_j can be discharged only in a translation step like (219). Hence, if i_j were previously discharged in $tr(\gamma_1)$, then it must have been by a translation of another application of $A_{[i_1,\ldots,i_n]} \to B$ on A. But that instance must share at least one index k with S_2 (because of how the compilation algorithm works). So $k \in S_1 \cap S_2$, in contradiction to $S_1 \cap S_2 = \emptyset$.

■

5.3.3 Completeness

The Theorem

Theorem 2 *(Completeness) Let* Γ *be a non-empty set of glue statements that are input to the algorithm. If* $\Gamma \vdash_{LL} \psi : G$ *then for some* ψ' *such that* $\psi' \equiv_{\beta\eta} \psi$*, the final output of the algorithm includes* $\psi' : G$*.*

We need to be careful about the claim. If there is a LL proof of $\psi : G$ from Γ then $\psi : G$ itself may not be among the final output of the algorithm. For every conclusion in LL, there is always an infinite number of proofs that end in equivalent conclusions (equivalent in the sense that the meaning terms are equivalent under the β- and η-conversion rules in the lambda-calculus). The claim is only that the algorithm will find one of these proofs.

The challenge is defining exactly which of these proofs the algorithm will find. Let us say that the proofs (under tr) which the algorithm finds are in Hepple Normal Form (HNF). We shall see that this normal form has the following properties:

1. It is in β-normal form but not necessarily in η-normal form.

2. The expansion property.

3. The compilation property.

The first property assumes that the original meaning terms in the input are atomic. A proof (under tr) that the algorithm finds is in β-normal form simply by virtue of (219) – the output of the tr translation cannot include something like:

$$\dfrac{\dfrac{b(x) : B}{\lambda x.b(x) : A \rightarrow B}\,\text{I} \qquad a : A}{(\lambda x.b(x))(a) : B}\,\text{E}$$

However, the proof is not necessarily in η-normal form. For example, the algorithm will (implicitly) find the proof (210) rather than (209) because it generalizes to the worst case, as was discussed regarding (211).

A proof in β-normal form has the *expansion property* iff any elimination step

$$\dfrac{\varphi : C_1 \rightarrow \ldots \rightarrow C_n \rightarrow A \quad \psi : (C_1 \rightarrow \ldots \rightarrow C_n \rightarrow A) \rightarrow B}{\psi(\varphi) : B}\,\text{E}$$

(where A is atomic) is such that $\varphi = \lambda v_{i_1} \ldots \lambda v_{i_n}.\varphi'$, and the left premise is obtained from a sequence of n introduction steps:

$$\dfrac{\dfrac{\dfrac{\vdots}{\varphi' : A}}{\lambda v_{i_n}.\varphi' : C_n \to A}\,\mathrm{I}_{i_n}}{\vdots}$$
$$\dfrac{\lambda v_{i_2} \ldots \lambda v_{i_n}.\varphi' : C_2 \to \ldots \to C_n \to A}{\lambda v_{i_1} \ldots \lambda v_{i_n}.\varphi' : C_1 \to C_2 \to \ldots \to C_n \to A}\,\mathrm{I}_{i_1}$$

An output of tr obviously has the expansion property (by virtue of (219)).

A proof has the *compilation property* iff the set of assumptions (i.e. leaves surrounded by brackets $[\ldots]_i$) in the proof is equal to the set of all $[\varphi : \mu(C)]_i$ such that C is compiled out of some other category by the compilation of Γ. A proof found by the algorithm obviously has this property.

Here is the main structure of the proof of the completeness theorem: If $\Gamma \vdash_{LL} \psi : G$ then there is a LL proof $\mathcal{D}_0$ of $\psi : G$ from Γ. Let $\mathcal{D}$ be its $\beta\eta$-normal form which ends in $\psi' : G$ (so also $\Gamma \vdash_{LL} \psi' : G$ and $\psi' \equiv_{\beta\eta} \psi$). By Proposition 1 below, there is a proof of $\varphi : G$ from Γ such that $\varphi \equiv_{\beta\eta} \psi' \equiv_{\beta\eta} \psi$ and the proof is in β-normal form and has the expansion property. By Proposition 2 below, it also has the compilation property. By Proposition 3 below, $\varphi : G$ is among the final output of the algorithm when it is given Γ and G.

Converting $\beta\eta$-Normal Form to HNF

Proposition 1 *If there is a LL proof of $\psi : G$ which is in $\beta\eta$-normal form then there is a LL proof of $\varphi : G$ which has the expansion property, where $\varphi \equiv_{\beta\eta} \psi$.*

Proof: I will provide a translation procedure from the $\beta\eta$-normal proof to a proof with the expansion property. First, we mark with * all potentially problematic locations in the input proof, i.e. all the minor premises of elimination steps, and add an auxiliary variable t:

(224) $$\dfrac{\varphi : A \quad \psi : A \to B}{\psi(\varphi) : B}\,\mathrm{E} \quad \rightsquigarrow \quad \dfrac{^*t : \varphi : A \quad \psi : A \to B}{\psi(t) : B}\,\mathrm{E}$$

where t is a fresh variable

Now we apply rules to convert the proof to HNF. If a starred formula has an atomic category A, we drop the star and set the variable:

(225) ${}^*t : \varphi : A \quad \rightsquigarrow \quad \varphi : A$

and set $t := \varphi$

If a starred formula with a non-atomic category was obtained by an introduction step (as it should in HNF since the star originated in a case like (224)), we move the star one line up, and place a correct value in t:

(226) $$\frac{\varphi : A}{{}^*t : \lambda v_i.\varphi : B \to A}\,\mathrm{I}_i \quad \rightsquigarrow \quad \frac{{}^*s : \varphi : A}{\lambda v_i.s : B \to A}\,\mathrm{I}_i$$

and set $t := \lambda v_i.s$

where s is a fresh variable

But if a starred formula with a non-atomic category is either a premise or was obtained from an elimination step, we do:

(227) $$\begin{array}{c}\vdots\,\mathcal{D}\\ {}^*t : \varphi : B \to A\end{array} \quad \rightsquigarrow \quad \frac{\dfrac{{}^*s : [v_i : B]_i \qquad \varphi : B \to A \;\vdots\,\mathcal{D}}{{}^*u : \varphi(s) : A}\,\mathrm{E}}{\lambda v_i.u : B \to A}\,\mathrm{I}_i$$

and set $t := \lambda v_i.u$

where s, u are fresh variables and i is a fresh index

Note: if the left has ${}^*t : [\varphi : B \to A]_j$ then the right has $[\varphi : B \to A]_j$ as well.

Example Conversion to HNF

Here is an example how this works. Consider the following set of premises:

(228) $\alpha : A \to A\ , \quad \beta : A \to B \to (C \to D) \to E\ , \quad \gamma : (A \to B \to (C \to D) \to E) \to F$

This gives one proof in $\beta\eta$-normal form:

$$\dfrac{\dfrac{\dfrac{\dfrac{[v_1 : A]_1 \quad \alpha : A \to A}{\alpha(v_1) : A}\,\mathrm{E} \qquad \beta : A \to B \to (C \to D) \to E}{\beta(\alpha(v_1)) : B \to (C \to D) \to E}\,\mathrm{E}}{\lambda v_1.\beta(\alpha(v_1)) : A \to B \to (C \to D) \to E}\,\mathrm{I}_1 \qquad \gamma : (A \to B \to (C \to D) \to E) \to F}{\gamma(\lambda v_1.\beta(\alpha(v_1))) : E}\,\mathrm{E}$$

The compilation procedure in Figure 5.1 will compile out of the second premise of (228) a new assumption with category C, and out of the third premise three new assumptions with categories A, B, and $C \to D$. We will see that in the HNF proof we get, these four will be exactly the categories of assumption leaves of the tree. (The desired HNF proof is shown below, after the words "The whole proof is now:".)

First, we mark with stars the minor premises of each elimination step:

$$\dfrac{\dfrac{\dfrac{\dfrac{{}^*t_1 : [v_1 : A]_1 \quad \alpha : A \to A}{{}^*t_2 : \alpha(t_1) : A}\,\mathrm{E} \qquad \beta : A \to B \to (C \to D) \to E}{\beta(t_2) : B \to (C \to D) \to E}\,\mathrm{E}}{{}^*t_3 : \lambda v_1.\beta(\alpha(v_1)) : A \to B \to (C \to D) \to E}\,\mathrm{I}_1 \qquad \gamma : (A \to B \to (C \to D) \to E) \to F}{\gamma(t_3) : F}\,\mathrm{E}$$

Because A is atomic, we set $t_1 = v_1$ and $t_2 = \alpha(t_1) = \alpha(v_1)$. Now we work on t_3. According to (226), we move the star up:

$$\dfrac{\dfrac{\dfrac{\dfrac{[v_1 : A]_1 \quad \alpha : A \to A}{\alpha(v_1) : A}\,\mathrm{E} \qquad \beta : A \to B \to (C \to D) \to E}{{}^*t_4 : \beta(\alpha(v_1)) : B \to (C \to D) \to E}\,\mathrm{E}}{\lambda v_1.t_4 : A \to B \to (C \to D) \to E}\,\mathrm{I}_1 \qquad \gamma : (A \to B \to (C \to D) \to E) \to F}{\gamma(t_3) : F}\,\mathrm{E}$$

and $t_3 = \lambda v_1.t_4$. Now according to (227), we change the t_4 line to:

$$\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{D}}{{}^*t_6 : [v_2 : B]_2 \qquad \beta(\alpha(v_1)) : B \to (C \to D) \to E}\,\mathrm{E}}{{}^*t_5 : \beta(\alpha(v_1))(t_6) : (C \to D) \to E}\,\mathrm{I}_2}{\lambda v_2.t_5 : B \to (C \to D) \to E}$$

where $\mathcal{D}$ is the part that was above the t_4 line, and we set $t_4 = \lambda v_2.t_5$. Since B is atomic, we get $t_6 = v_2$. Since B is atomic, we set $t_6 = v_2$. Now according to (227) again, we change the t_5 line to:

$$\dfrac{\dfrac{{}^*t_8 : [v_3 : C \to D]_3 \quad \beta(\alpha(v_1))(t_6) : (C \to D) \to E}{{}^*t_7 : \beta(\alpha(v_1))(t_6)(t_8) : E}\,\mathrm{E}}{\lambda v_3.t_7 : (C \to D) \to E}\,\mathrm{I}_3$$

and $t_5 = \lambda v_3.t_7$. Since E is atomic, $t_7 = \beta(\alpha(v_1))(t_6)(t_8)$. But since $C \to D$ is not atomic, we need to keep expanding the t_8 line to:

$$\dfrac{\dfrac{{}^{*}t_{10} : [v_4 : C]_4 \quad [v_3 : C \to D]_3}{{}^{*}t_9 : v_3(t_{10}) : D}\,\mathrm{E}}{\lambda v_4.t_9 : C \to D}\,\mathrm{I}_4$$

and $t_8 = \lambda v_4.t_9$. Since D and C are atomic, $t_9 = v_3(t_{10})$ and $t_{10} = v_4$.

So the original bottom line of the entire proof now becomes $\gamma(t_3) = \gamma(\lambda v_1.t_4) = \gamma(\lambda v_1.\lambda v_2.t_5) = \gamma(\lambda v_1.\lambda v_2.\lambda v_3.t_7) = \gamma(\lambda v_1.\lambda v_2.\lambda v_3.\beta(\alpha(v_1))(t_6)(t_8)) = \gamma(\lambda v_1.\lambda v_2.\lambda v_3.\beta(\alpha(v_1))(v_2)(\lambda v_4.t_9)) = \gamma(\lambda v_1.\lambda v_2.\lambda v_3.\beta(\alpha(v_1))(v_2)(\lambda v_4.v_3(v_4)))$.

The whole proof is now:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[v_4 : C]_4 \quad [v_3 : C \to D]_3}{v_3(v_4) : D}\,\mathrm{E}}{\lambda v_4.v_3(v_4) : C \to D}\,\mathrm{I}_4 \qquad \dfrac{[v_2 : B]_2 \qquad \dfrac{\dfrac{[v_1 : A]_1 \quad \alpha : A \to A}{\alpha(v_1) : A}\,\mathrm{E} \qquad \beta : A \to B \to (C \to D) \to E}{\beta(\alpha(v_1)) : B \to (C \to D) \to E}\,\mathrm{E}}{\beta(\alpha(v_1))(v_2) : (C \to D) \to E}\,\mathrm{E}}{\beta(\alpha(v_1))(v_2)(v_4.v_3(v_4)) : E}\,\mathrm{E}}{\lambda v_3.\beta(\alpha(v_1))(v_2)(\lambda v_4.v_3(v_4)) : (C \to D) \to E}\,\mathrm{I}_3}{\lambda v_2 \lambda v_3.\beta(\alpha(v_1))(v_2)(\lambda v_4.v_3(v_4)) : B \to (C \to D) \to E}\,\mathrm{I}_2}{\lambda v_1 \lambda v_2 \lambda v_3.\beta(\alpha(v_1))(v_2)(\lambda v_4.v_3(v_4)) : A \to B \to (C \to D) \to E}\,\mathrm{I}_1 \qquad \gamma : \ldots}{\gamma(\lambda v_1 \lambda v_2 \lambda v_3.\beta(\alpha(v_1))(v_2)(\lambda v_4.v_3(v_4))) : F}\,\mathrm{E}$$

Properties of HNF

Proposition 2 *If a proof in β-normal form has the expansion property then it has the compilation property.*

Proof: This can be seen by reviewing the expansion procedure above as well as the compilation procedure. Consider a member $H = A_1 \to \ldots \to A_n \to B$ of Γ (B atomic). The compilation procedure considers all the suffixes of this premise.

Suppose that a suffix $A_i \to \ldots \to A_n \to B$ appears in the expanded proof as a major premise of an application. If A_i is atomic then the compilation procedure will not do anything to it, and the expansion procedure will also leave it alone (this comes from a combination of (224) and (225)). If however $A_i = C_1 \to \ldots \to C_m \to D$ (D atomic), then the compilation property will compile out the categories $C_1, \ldots, C_n$. Paralleling that, the expansion procedure, using (226) and (227), will guarantee that each of the C_i already exists as an assumption in the proof, and if not, it will be added.

If $A_i \to \ldots \to A_n \to B$ does not appear in the expanded proof as a major premise of an application, then it must appear either as a minor premise of an application, or

(thanks to the expansion property) as a member of an expansion sequence (a sequence of introduction steps) that leads to an application. Let R be the major premise of that application. Therefore, there is a related A_i assumption in the expanded proof. This assumption is not compiled out of H by the compilation algorithm when it processes H, but it *is* compiled out R. Furthermore, if A_i is not atomic and $A_i = C_1 \rightarrow \ldots \rightarrow C_m \rightarrow D$ (D atomic), then the compilation algorithm will compile out each C_j. Paralleling that, the expansion procedure puts a star not only on the u line but also on the s line of (227). A relevant example is the $C \rightarrow D$ in the expansion example above.

From HNF to the Algorithm

Proposition 3 *If a LL proof $\mathcal{D}$ that ends with $\psi : G$ (G atomic) is in HNF, then the algorithm, when run on Γ, G, will create a chart element γ such that $tr(\gamma)$ is equal to $\mathcal{D}$ (under renaming of variables and indices).*

Proof: Straightforward. If a proof has the compilation property then its leaves correspond (under renaming of variables and indices) to the initial content of the agenda. The expansion property guarantees that the algorithm will be able to combine all these elements to create $\psi : G$. Simply, any subproof of $\mathcal{D}$ with the shape:

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\lambda v_{i_n}.\psi : \mu(i_n) \rightarrow A}\,\mathrm{I}_{i_n}}{\vdots}}{\dfrac{\lambda v_{i_1} \ldots \lambda v_{i_n}.\psi : \mu(i_1) \rightarrow \ldots \rightarrow \mu(i_n) \rightarrow A}{\delta(\lambda v_{i_1} \ldots \lambda v_{i_n}.\psi) : \mu(B)}\,\mathrm{E}}\,\mathrm{I}_{i_1} \qquad \mathcal{D}_2$$

corresponds to one combination step. If $\mathcal{D}_1$ and $\mathcal{D}_2$ correspond to chart elements γ_1 and γ_2 then a new element can be created from them that corresponds to this subproof.

5.3.4 Complexity

A multiset of n first-order formulas yields at most $(n-1)!$ possible ($\beta\eta$-normal) proofs. This is because each proof can be written as a unique tree when each elimination step has the minor premise on the left and the major premise on the right. Each such tree corresponds to one permutation of the premises, namely the one that can be read off the tree's leaves. However, not every permutation of the premises corresponds to a possible proof tree. The worst case of $(n-1)!$ can be obtained from a premise multiset such as:

(229) $A \quad A \rightarrow A \quad A \rightarrow A \quad \ldots$

The size of the input Γ to the algorithm is strictly speaking not the number of members of Γ but the number of distinct instances of atoms in Γ (the sum of the lengths of the members of Γ). The compilation procedure runs in time linear in the size of Γ, and produces a new set Γ' of the same size (a category A may be compiled out of a category B, leaving a shorter version of B). The compiled input has at most n (first-order) premises, and so it yields at most $(n-1)!$ possible proofs. (By Theorem 2, we can also conclude about linear logic that it has at most a finite number of proofs from a finite number of premises.)

How many steps does the algorithm do? In the case of (229), we will have one element for the original A and $(n-1)!$ additional created A elements. Each of these needs to be compared against each of the $n-1$ premises of $A \rightarrow A$. Thus we have $(1+(n-1)!)*(n-1) = O(n!)$ comparison steps. This is true also in the general case: At most $(n-1)!$ elements can be created, each being compared to at most $n-1$ other elements. Each comparison involves computation of set intersection, which takes at most $O(n)$ time using hashing. Let us assume that the chart hashing used by the algorithm takes constant time, i.e. it can find the list of all $A \rightarrow X$ given an atomic category A, or the list of A given an implication $A \rightarrow X$, in constant time. All this give us a total worst-case time of $\Theta(n!)$. Since the algorithm finds all the proofs and creates a chart element for each, this is also the space complexity.

A special case is when each atomic category appears exactly once positively in Γ. This will also be the case in the compiled Γ. In this case, there is exactly one possible proof. The algorithm will then run in $O(n^2)$ time because each agenda item has exactly one thing to match against, and the computation of each set intersection takes $O(n)$ using hashing.

5.4 Optimization

5.4.1 Local Requirements Check

Consider the analysis of equi verbs (187) in section 4.4.5, repeated here:

(230) $[[\text{John}]_b$ tried $[\text{to leave}]_c]_a$

$john : \mathbf{b}^e$

$try : \mathbf{b}^e \rightarrow (\mathbf{b}^e \rightarrow \mathbf{c}^t) \rightarrow \mathbf{a}^t$

$leave : \mathbf{b}^e \rightarrow \mathbf{c}^t$
$\Rightarrow try(john, leave)$

The compilation and derivation are:

(231)
1. $john : \mathbf{b}^e : \{1\}$
2. $try : \mathbf{b}^e \rightarrow \mathbf{c}^t_{[3]} \rightarrow \mathbf{a}^t : \{2\}$
3. $v_3 : \mathbf{b}^e : \{3\}$
4. $leave : \mathbf{b}^e \rightarrow \mathbf{c}^t : \{4\}$
5. $\mathbf{c}^t_{[3]} \rightarrow \mathbf{a}^t : \{1, 2\}$ [2+1]
6. $\mathbf{c}^t_{[3]} \rightarrow \mathbf{a}^t : \{2, 3\}$ [2+3]
7. $\mathbf{c}^t : \{1, 4\}$ [4+1]
8. $\mathbf{c}^t : \{3, 4\}$ [4+3]
9. $\mathbf{a}^t : \{1, 2, 3, 4\}$ [5+8]

Note that elements 6 and 7 cannot combine because 3 is required by $\mathbf{c}^t_{[3]}$ in element 6 but is not a member of the index set of element 7. They are both, in fact, dead ends, just as elements 10 and 11 were in example (213). Can we do something to eliminate them early?

It is easy to eliminate 6: Since ${\mathbf{c}^t}_{[3]}$ requires index 3 but this index already appears in the index set of element 6, this means element 6 can never be used. We can add a condition to the combination rule that verifies that if $X \rightarrow Y$ is applied on a X that has index set S, then S is disjoint with all index lists L that are required by categories in Y. This would prevent the creation of element 6.

In general, it is not easy to eliminate element 7 because locally, there does not seem to be a reason not to combine elements 4 and 1. However, in the specific case of equi verbs, we can modify the glue specification so that the subject role of the embedded verb would be marked differently than the subject role of the equi verb. In the example above, we would use the category ${\mathbf{b}_{equi}}^e$ rather than $\mathbf{b}^e$, in both the second argument of the equi verb, and in the contribution of the embedded verb. The glue specification for the latter will need to look whether it is an argument of an equi verb, and if so, use the category ${\mathbf{b}_{equi}}^e \rightarrow \mathbf{c}^t$. When the category that the equi verb uses, $\mathbf{b}^e \rightarrow ({\mathbf{b}_{equi}}^e \rightarrow \mathbf{c}^t) \rightarrow \mathbf{a}^t$, will be compiled, the compiled-out premise will have category ${\mathbf{b}_{equi}}^e$ rather than $\mathbf{b}^e$, and so no confusion will arise.

Chapter 6

Packed Glue Derivation I: Internal Ambiguity

So far, I have answered the first question posed in the abstract: How can meaning representations be calculated given one syntactic analysis of a sentence? In this and the next chapter, I answer the next question: How can all the meaning representations for an ambiguous sentence be calculated efficiently given a packed F-structure?

For developing our packed algorithm, it is useful to divide the ambiguities into two kinds. The first kind arises when there are morphosyntactic ambiguities in the sentence. These ambiguities manifest themselves in the packed F-structure and the resulting packed set of glue semantics statements, where some of the parts and glue statements appear in different parts of the choice-space (see section 1.3.3). Another reason why glue statements may be relativized to different choice-space contexts is that a quantifier may land on more than one clause, as discussed in section 4.5.3.

The second kind of ambiguity arises from semantic scope ambiguities where different permutations of quantifiers and other modifiers are possible. This manifests itself in glue semantics by more than one way of combining the premises. A simple example of this is a set of premises that has two modifiers of category $A \rightarrow A$, as in section 6.1. Note that I am referring here only to cases where there is more than one *normal form* derivation from the premises. (There is always an infinite number of non-normal derivations from a set of premises – see section 5.1.2 – but this is not what I mean here by "more than one way of combining the premises". The proliferation of non-normal-form derivations was already

addressed in the previous chapter by finding only one proof from each equivalence class.)

We can call the first kind of ambiguity "external ambiguity" because it exists already in the (packed) set of glue semantics statements that are the input to the glue derivation stage (the stage that takes the glue statements and combines them to form meaning representations). The second kind can be called "internal ambiguity" because it exists when there is more than one way of combining a set of glue statements (even within one maximally-specific choice in the choice-space).

In this chapter, I address internal ambiguity, and in the next chapter, external ambiguity. I cover these issues in that order because the choice-space management framework adds additional complications, so it is easier to start with internal ambiguity. In the next chapter, we will see that the two kinds of ambiguity may interact in non-trivial ways.

There are two issues to address regarding internal ambiguity:

1. How to prevent the premature combination of two elements, i.e. before all the possible modifiers already applied on one of the elements.

2. How to deal with the exponential explosion that results from all possible permutations of modifiers.

In order to address these issues, we first need to reformulate Hepple's algorithm. I do this in the first three sections. This will allow us to prevent premature combinations in section 6.4 and to address the exponential explosion in section 6.5. I then present additional optimizations in section 6.6 and additional features in section 6.7.

6.1 Meaning Terms Are Not Needed During Derivation

Recall that although a sentence may have exponentially many parse trees, a chart parser is able to record all of them in a chart forest in polynomial time. This is thanks to the fact that the top parts of two subtrees T_1 and T_2 are packed together into one element in the chart if both trees have the same span and the same root category. This is a correct packing because larger trees do not care about the internal differences between T_1 and T_2, only the span and top syntactic category matter.

Although the basic algorithm in chapter 5 uses a chart, there is no real packing of results. In other words, there is never a case where an element is created and attempted

to be added to the chart, but some similar element already exists in the chart, and so the two elements are merged into one.

The first opportunity for packing comes from noticing that meaning terms do not affect derivations at all, they simply piggy-back on top of them. Consider the following example:

(232) Given glue statements:

$a : A, \quad b : A \to A, \quad c : A \to A, \quad d : A \to B$

Computation of derivation:

1.	$a : A : \{1\}$	
2.	$b : A \to A : \{2\}$	
3.	$c : A \to A : \{3\}$	
4.	$d : A \to B : \{4\}$	
5.	$b(a) : A : \{1, 2\}$	[2+1]
6.	$c(a) : A : \{1, 3\}$	[3+1]
7.	$c(b(a)) : A : \{1, 2, 3\}$	[3+5]
8.	$b(c(a)) : A : \{1, 2, 3\}$	[2+6]
9.	$d(a) : B : \{1, 4\}$	[4+1]
10.	$d(b(a)) : B : \{1, 2, 4\}$	[4+5]
11.	$d(c(a)) : B : \{1, 3, 4\}$	[4+6]
12.	$d(c(b(a))) : B : \{1, 2, 3, 4\}$	[4+7]
13.	$d(b(c(a))) : B : \{1, 2, 3, 4\}$	[4+8]

Notice how similar 7 is to 8, and 12 to 13. Each of these pairs shares the category and set of indices, and only the meaning term is different. This duplicates work in some sense: the potential to combine, as determined by the category A and the set $\{1, 2, 3\}$ in 7 and 8, is the same, so all later combinations based on this entry will be duplicated: the duplicity in 7 and 8 causes the duplicity in 12 and 13.

We can improve on that by keeping in the chart only the category and the index set but not the meaning term, as well as remembering for each element e a set of parent-links, where each parent-link has two pointers to the parent elements of e. The first pointer in the pair points to the functor parent while the second points to the argument parent. This will create a proof forest, and the recorded information is enough to allow us to reconstruct the meaning terms. For example, instead of elements 7 and 8 we would have just one element: $A : \{1, 2, 3\}; \{\langle 3, 5\rangle, \langle 2, 6\rangle\}$, call it element 78. When we remove element 4 from the agenda, we will not need to combine it separately with the old elements 7 and 8 but only once with element 78 to produce $B : \{1, 2, 3, 4\}; \{\langle 4, 78\rangle\}$, call this element 1213.

How do we reconstruct the meaning expressions from this proof forest? We start from the goal history, and recursively construct the meaning expression by traversing parent links. The meaning expression for an atomic history which has an index set $\{i\}$ (and a null parent-link) is the meaning expression of the i premise. When we reach a history h which has a parent-link $\langle h_1, h_2 \rangle$, then $meaning(h)$ can be $apply(meaning(h_1), meaning(h_2))$. If h has more than one parent-link, we can choose one of them, depending on which result we want. After we get the final meaning expression, we can apply $\beta\eta$-reductions on it.

In the example above, $meaning(h_{1213}) = apply(meaning(h_4), meaning(h_{78}))$. We have $meaning(h_4) = d$, and for $meaning(h_{78})$ we can choose to follow $\langle 3, 5 \rangle$ or $\langle 2, 6 \rangle$. If we follow the first then $meaning(h_{78}) = apply(meaning(h_3), meaning(h_5))$. We keep going and eventually get the final meaning of $apply(d, apply(c, apply(b, a)))$ which reduces to $d(c(b(a)))$. If we followed $\langle 2, 6 \rangle$ instead, we would get $d(b(c(a)))$. In this way, we can get all possible results.

6.2 Initial Thoughts about Preventing Premature Combinations

6.2.1 The Problem

In section 6.1, only element 1213 is the final element whereas elements 9, 10, and 11 are dead-ends. If we also had the premises $B \rightarrow C$ and $C \rightarrow D$ then these two would combine with element 1213 correctly. However, elements 9, 10, and 11 would combine with these two as well and give rise to six additional spurious elements. They are spurious because they can never be part of a final derivation since they did not use all the elements that involve category A, and once category B is reached, there is no chance to get back to A again. This will be detected only once the algorithm finishes and sees that not all indices appear in the index sets of the elements. Until that time, a potentially exponential amount of unnecessary work will be done. Instead, we would like to use all the $A \rightarrow A$ modifiers before we leave category A.

This is true only in the particular example given here, but not in general in linear logic. In particular, a set of premises might contain cycles that are longer than length one, so if a derivation reaches a category $A \rightarrow B$ (A, B atomic), it is not true in general that the derivation may not return to A. Consider for example the set:

(233) $A,\ A \to A,\ A \to B,\ B \to B,\ B \to C,\ C \to C,\ C \to B,\ B \to D,\ D \to E$

A full derivation must start with A, go to B, then to C, then back to B, and finally to D. By the time we reach B, we should use all premises that involve A because there is no way to go back to it. In contrast, when we move from B to C, even though we "leave" B at that point, we must return to it later. As long as we did not reach D, it is particularly important to remember the index of $B \to B$ since we do not want to use it twice – between $A \to B$ and $B \to C$ as well as between $C \to B$ and $B \to D$ – or not use it at all.

6.2.2 Category Graph

These considerations show that we want some way to find out which categories should be used before others. We will also see in the next chapter that in the sentence "John thinks that time flies like an arrow", we would like to know that the derivation of "John thinks that" appears after the derivations of the ambiguous sentence "time flies like an arrow", so that we could have an efficient packed computation. However, we want to know this information about ordering categories without calculating all the full derivations. We also need to be aware of cycles as in (233).

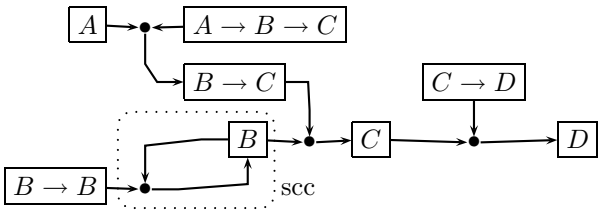
This suggests that we should compute a *category graph*. This graph contains one node for each category that could be reached in a derivation, but stripped away from information about indices. Thus, if $C_1\ {}_{L_1} \to \ldots \to C_n\ {}_{L_n} \to A$ is a (compiled) category, then each of its suffixes, stripped from $L_1, \ldots, L_n$, forms a node in the graph.[1] In addition, the graph has connector nodes. For each pair of category nodes $X \to Y$ and X, there are edges from these two to a connector node, and an edge from the connector node to the category node Y. The size of this graph is linear in the sum of the lengths of the compiled premises: Whenever you see a $X_L \to Y$, you know about X, Y, and the intermediate connector node. It can also be computed in linear time. (This is linear also in the size of the uncompiled input because the size of the complied input is linear in the size of the uncompiled input).

Since we need to be aware of cycles, we also compute the strongly-connected components (SCCs) and the reduced category graph (where each SCC is collapsed into one node), which is a directed-acyclic graph. Finally, we compute the topological sort of the reduced graph to get an ordering on the nodes. Computing the SCCs and the topological sort is also done in time linear in the size of the graph (Cormen et al., 2001, ch.22).

[1] In section 6.4.4, we will relax this a little in order to optimize the treatment of modifier categories.

Here is an example category graph:

(234) Premises: A, B, $B \to B$, $A \to B \to C$, $C \to D$



6.2.3 A First Attempt

In order to prevent the problem of premature combination, as in combining $A \to B$ with A in (232) before all $A \to A$ modifiers were used, we may try to use the category graph as a side assistance to Hepple's algorithm. In particular, we want the graph to help us answer the question: If we move from category X to category Y, what are all the indices that should already exist in the index set of X? Thus, in example (232), we would know that when we move from A to B, the index set of A should include all of 1, 2, and 3. This would prevent the creation of the unwanted elements 9, 10, and 11 there.

A first attempt at this is to collect all the indices mentioned in a SCC into a set S and convert that SCC into one node with index set S. By following the order of nodes in the topological sort of the reduced graph, we can flow the index set of each node to its successors. Formally, if we process node v that has index set S, we add all members of S to the index set of v' for all nodes v' such that there is an edge from v to v'. We mark the flowed index set of a node by σ. In example (234) we would get:

(235) $\sigma(A) = \{1\}$ $\sigma(A \to B \to C) = \{4\}$ $\sigma(B \to C) = \{1, 4\}$
The SCC containing B and $B \to B$ has $\sigma = \{2, 3\}$
$\sigma(C) = \{1, 2, 3, 4\}$ $\sigma(C \to D) = \{5\}$ $\sigma(D) = \{1, 2, 3, 4, 5\}$

It might appear that now we can modify the chart algorithm as follows: When the chart algorithm derives a category Y from categories X and $X \to Y$, then if Y is not in the same SCC as X in the graph, we make sure that the index set of Y includes all the indices of $\sigma(X)$. Thus, in example (234), since $\sigma(B) = \{2, 3\}$, we can use $B \to C$ to create C only using the element with category B and span $\{2, 3\}$, and not the element with span

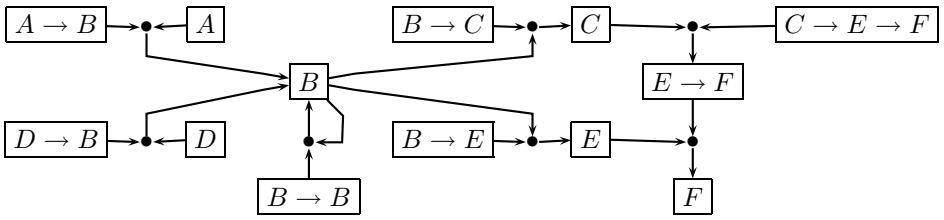
{2}. Thus we would eliminate the creation of the unwanted element with category C and span $\{1, 2, 4\}$. In example (232), we would eliminate the creation of the unwanted elements 9, 10, and 11. In other words, we "finalize" the category A there before moving to B.

There is a bug with this formulation. Consider the following example:

(236) A, $A \rightarrow B$, $B \rightarrow C$
D, $D \rightarrow B$, $B \rightarrow E$
$C \rightarrow E \rightarrow F$
$B \rightarrow B$

Here B can be created by two completely unrelated ways: $A + A \rightarrow B$ and $D + D \rightarrow B$. The category B also participates in producing two completely unrelated results: C and E which are then consumed by a larger premise $C \rightarrow E \rightarrow F$. This can be easily seen in the category graph:

(237)



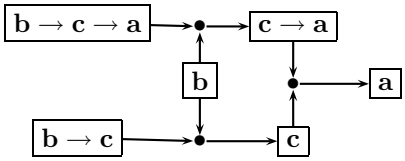
According to the formulation above, $\sigma(B)$ includes the indices of all of A, $A \rightarrow B$, D, $D \rightarrow B$, and $B \rightarrow B$. This means that when we move from B to C, the index set of B should include all of those indices. But that is not a good result. When we move from B to C what we need is to check that either all elements that involve category A have been used, or all elements that involve category D have been used, but not require both. An even more difficult problem is involved with $B \rightarrow B$. Locally, when we move out of B, we should not require this category to be used. Only when we leave F should we require that – but how can we know that leaving F requires having used $B \rightarrow B$? This is a non-local dependency. If there were three arrows exiting B and only two of them led to F, then we should not require leaving F to have used $B \rightarrow B$.

Situations such as this one do not arise in practice in a glue semantics input derived from a glue specification. This is because glue specifications yield premise sets with restricted forms. It is impossible to get a situation as in (237) because category B there, which should correspond to a particular location (or sub-location) in the syntactic structure, is created

by two *completely independent* ways, and it is also consumed by two *completely independent* consumers that compete for the same resource. Also, a situation as in (233) never arises in practice. It would require some sort of a directed cycle in the syntactic structure as a certain location B is both subordinated to another location C and subordinates it too. In a glue specification, directed cycles only arise from modifiers such as negation, modals, quantifiers, adjectives, and adverbs.

Nevertheless, it is important that our algorithm will be correct for all cases and not rely on any assumptions. The only thing that is allowed is optimizations for specific cases that arise in practice. This point is important because if we make our algorithm depend on some assumptions, and at some future time (when some complex linguistic construction is analyzed) it turns out that the assumptions are slightly wrong, then our algorithm might completely break down, and we may need to completely revise it. As an example, even though a case such as (237) seems impossible, we saw in the previous chapter that (231) *is* possible. Its category graph is:

(238)



We already have a problem with node **b** because according to what we said above, its index set would include both 1 and 3, so when leaving **b** in either direction, both 1 and 3 need to be in the index set of the element in the chart, which is impossible.

6.3 A New Algorithm

6.3.1 The Goal

The problem with using the category graph as we did above is that it is too crude of an approximation. What we really want is to explore all the ways of reaching a certain SCC before we move out of it, because once we move out of it, we can never come back to it. The problem with the chart algorithm is that the order in which elements are combined is unpredictable. If an A is combined with $A \rightarrow B$, it might be that at some later time, the algorithm will try to combine A with something else, and there is no principled way of knowing.

The solution is to impose an order on the combinations according to the topological sort of the reduced category graph. Our goal is to calculate for each node in the category graph the set of all elements that share that category. In effect, we will be doing everything that the chart algorithm did except we will do it in a specific order, as dictated by the graph.

From now on we will use the term *history* for what we used to call *element*, so as not to confuse the original chart algorithm with the new algorithm we define next. A history h consists of a category, a span (a set of indices), and a set of parent-links, where each parent-link has two pointers to the parent histories from which h was derived (if h corresponds to an initial premise, it will have a null parent-link). A history does not include meaning terms – these will be computed once the calculation of histories finishes, as was explained in section 6.1.

6.3.2 Calculating Histories

Not Inside an SCC

First consider what we should do with a graph node that is its own SCC (it is not a member of an SCC with more than one node).

Connector Node: If the node is a connector node whose two parents are the category node $B \rightarrow C$ with a set Z_1 of histories and the category node B with a set Z_2 of histories, we need to consider the cross-product of Z_1 and Z_2 to investigate all possible combinations. In other words, for each $h_1 \in Z_1$ and each $h_2 \in Z_2$, try to combine h_1 with h_2. If the combination is possible, create a new history and add it to the connector node.

When can two histories $h_1 = \langle A, S_1, pls_1 \rangle$ and $h_2 = \langle B, S_2, pls_2 \rangle$ combine? The following conditions must be met:

1. $A = B_L \rightarrow C$ for some category C and (possibly empty) index list L
2. $S_1 \cap S_2 = \emptyset$
3. $L \subseteq S_2$
4. $L' \cap S_2 = \emptyset$, where L' is the union of all index requirements on categories in C (this is the optimization from section 5.4.1)

The result of the combination is the history $\langle C, S_1 \cup S_2, \{\langle h_1, h_2 \rangle\}\rangle$.

(Condition 1 is guaranteed to hold in a connector node because the two histories come from two appropriate parent nodes in the category graph. We just need to find L.)

Category Node: In a category node C, we create one history $\langle C, \{i\}, \{null\}\rangle$ for each (compiled) glue premise with category C and index i. We also need to collect all the histories that appear on all connector nodes that lead into node C. Note that a node may require both steps. For example, if the set of glue premises includes A, $A \rightarrow B$, and B, then category node B will have both a premise history from the third premise, and an incoming connector node whose parents are A and $A \rightarrow B$.

Once we have the set of all histories in the node, we should compress all histories that share exactly the same category and span (similarly to what we did in section 6.1). For example, the history $\langle C, S, \{\langle h_{11}, h_{12} \rangle\}\rangle$ and the history $\langle C, S, \{\langle h_{21}, h_{22} \rangle\}\rangle$ should be compressed into $\langle C, S, \{\langle h_{11}, h_{12} \rangle, \langle h_{21}, h_{22} \rangle\}\rangle$, and so forth with all histories that have category C and span S. A set of parent-links with more than one member indicates there is more than one way of obtaining this category with this span.

Why must we explicitly require "share the same category" if we are already in one category node in the graph? This is because in a category node, although all histories share the same underlying category, they may differ as to the set of indices that are required by subcategories. For example, a category node $A \rightarrow A$ may have two histories with categories $A_{[5]} \rightarrow A$ and $A_{[9]} \rightarrow A$, respectively. These histories may not be compressed together because they have different potentials of combination with other histories.

Inside an SCC

When we follow the topological sort of the reduced graph and we reach a node that corresponds to a non-singleton SCC in the category graph, we do the following. First, we find all the input and output category nodes in the SCC. An input category node is a member of the SCC which either has no parents or both its parents are outside the SCC. An output category node is a member of the SCC which either feeds into a connector node outside the SCC or whose category is the global goal category.

Default Behavior: If we just want to recreate what Hepple's algorithm does, without optimizations, we do the following. We collect all the histories that already exist in category

nodes in the SCC as well as in nodes that feed into members of the SCC. We put all these histories on a new agenda, and then run the chart algorithm on this agenda (with a new and initially empty chart). This calculates all combinations as usual.

When this is done, for each output node X of the SCC, we find all elements in the chart whose underlying category is X. For each such element, we convert it into a history and put that history in the list of histories of node X. In each output node, we compress its histories as we did for a non-SCC category node above.

(This is one simple way of calculating all results in a SCC. It is conceptually simpler to state than a method that would follow the nodes inside the SCC. In such a method, we would use the procedures we used for non-SCC nodes, except now, because of the cycles, we will have at least one step where we try to recalculate the combination of two histories which we already combined in a previous step. In such a case, we should not recalculate that combination. Furthermore, because of the cycles, we need to initialize all members of the SCC as "needs visiting", and each time we add a history to a node, we need to change its status to "needs visiting", even if we already visited it. We will need to visit an already-visited node at least once. We continue with this until no node is marked as "needs visiting".)

Correctness

This reformulation of the basic algorithm clearly produces all and only the histories produced by the basic algorithm.

6.3.3 Example

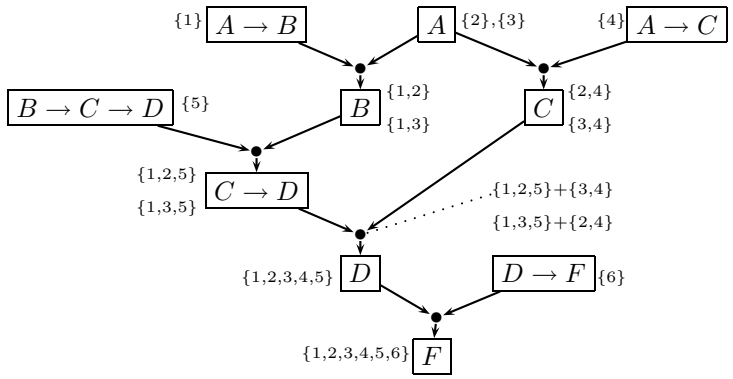
Here is an example to demonstrate how the algorithm works with the category graph (though this example does not have non-trivial SCCs). Consider the following input:

(239)

meaning	category	span
p	$A \rightarrow B$	{1}
q	A	{2}
r	A	{3}
s	$A \rightarrow C$	{4}
t	$B \rightarrow C \rightarrow D$	{5}
u	$D \rightarrow F$	{6}

This input leads to the following category graph:

(240)



The category graph itself essentially provides the skeleton of the derivation, but it is missing the information about the indices. The picture also depicts the spans in the various category nodes. After building the graph, the algorithm initializes the leaf nodes with the initial histories. Then it follows the order of a topological sort of the graph, say first processing node B, where the history from node $A \rightarrow B$ and the two histories from node A are combined, and then it keeps going. The crucial node is D and the connector node above it. There are two histories in each of C and $C \rightarrow D$, and so we need to check four combinations, but only two of them are possible (where the spans are disjoint). However, once the combinations are made, the resulting spans are identical: $\{1, 2, 3, 4, 5\}$, and so instead of keeping them separately, we compress them in node D. When the algorithm finishes, we reconstruct the meaning terms by going back from the goal node with $u(\ldots)$, and then we have a choice between $t(p(q), s(r))$ (which corresponds to $\{1, 2, 5\} + \{3, 4\}$) and $t(p(r), s(q))$ (which corresponds to $\{1, 3, 5\} + \{2, 4\}$).

The difference between the behavior here and the chart algorithm is that in the chart algorithm, when we discover one way to get D, say using $\{1, 2, 5\}+\{3, 4\}$, we put the new D element on the agenda, and immediately keep working with it to combine it with $D \rightarrow F$. Later, we will discover there is another way to get D with the same span $\{1, 2, 3, 4, 5\}$, and so we compress the new element into the existing one. In contrast, in the category graph we follow the topological sort and first find *all* the ways to get D before we continue with the derivation. This is crucial for the packed algorithm.

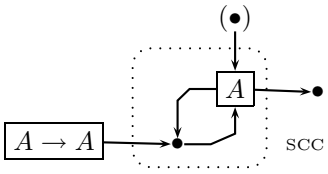
6.4 Preventing Premature Combinations

6.4.1 A Basic Optimization

The algorithm so far is guaranteed to produce exactly the same results as Hepple's original algorithm. The new algorithm just computes histories in an order that has a property that the chart algorithm lacks, namely that all ways of producing a category are explored before we move out of that category's SCC to other parts of the derivation (which cannot return to that category).

This property allows us to try the following optimization. Suppose a SCC consists of exactly one category node with atomic category A, and exactly one connector node whose parents are that A and another node $A \rightarrow A$, and whose output is that A again. Further assume that there is at most one input arrow into A from outside, and that we have just one initial history in A:

(241)



We call this a *basic SCC* or *basic cycle*. In this case, we should be able to filter, in some way, histories in the (output) node A that did not use all the material of the modifiers in $A \rightarrow A$. We can try to formulate this filtering as either of the following:

(242) Delete a history from the output node A of a basic SCC if it does not include some index i which is a member of some history in the $A \rightarrow A$ node.

(243) Delete a history from the output node A of a basic SCC if its span is S and there is another history in A whose span S' is a strict superset of S, i.e. $S' \supset S$.

Using this optimization, elements 1, 5, and 6 in section 6.1 will be filtered, and so elements 9, 10, and 11 will be prevented.

(We need to point out that without the optimization from section 5.4.1, we could obtain a defunct modifier that can combine with nothing, so optimization (242), which requires all modifiers to be used, may not work. However, with the help of that optimization, we do not need to worry about this problem.)

Is this optimization correct in the sense that it never filters a history that can participate in a legal derivation? (242) as stated above is not correct. Here is an example:

(244) $B:\{1\},\ B:\{2\},\ B \to A \to A:\{3\},\ A:\{4\},\ B \to A \to C:\{5\}$

There are two ways to get $A \to A$, using $\{1,3\}$ or $\{2,3\}$. Therefore, it is incorrect to require that A output only those histories that include all indices from all histories in $A \to A$. The correct output has two histories: one with $\{1,3,4\}$ and another with $\{2,3,4\}$.

The optimization can be revised. What we really want is to make sure the modifier itself is used. We can identify the main index of a history:

(245) Definition: *main index*

- A history with an atomic category does not have a main index.
- The main index of an initial history with index i and a non-atomic category is i.
- The main index of a non-initial history with a non-atomic category is the same as the main index of the functor parent of the history.

Thus, the main index of a history with category $X \to Y$ is i if that is the main index of the parent history with category $Z \to X \to Y$. Now we can rephrase (242):

(246) Delete a history from the output node A of a basic SCC if it does not include the main index of some history in the $A \to A$ node.

The optimization (246) is correct (under the assumption that there is just one initial history in A). Here is why. The main index i of a history in the $A \to A$ node is the index of a premise P of category $C_1 \to \ldots \to C_n \to A_L \to A$ (for some L), and so in any complete derivation, it must participate in some history that at some point combines with an A. If we allow the SCC to output a history that does not include i, then that history cannot lead to a complete derivation – there will never be another place later where i has a possibility to participate. Even if P leads to two different $A_L \to A$ modifiers, as in (244), both will have the same index i, and the optimization requires only one of them to be used.

The optimization (243) is not correct in general. For example, consider:

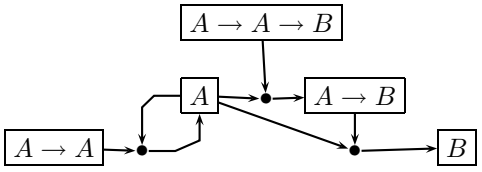
(247) $X:\{1\},\ X:\{2\},\ X \to Y:\{3\},\ Y \to X:\{4\},\ X \to A \to A:\{5\},\ A:\{6\},$
$X \to A \to B:\{7\}$

The node $A \rightarrow A$ will have four modifiers, with index sets: $\{1,5\}$, $\{2,5\}$, $\{1,3,4,5\}$, $\{2,3,4,5\}$. The basic SCC which includes A needs to output four histories that are obtained by adding 6 to each of these. Optimization (246) will work because the four modifiers have the same main index, 5. But optimization (243) will incorrectly filter the histories with the index sets $\{1,5,6\}$ and $\{2,5,6\}$. Nevertheless, there are two completely different ways to produce a basic X and to consume it, and this does not occur in practice in a glue semantics set of premises. Still, optimization (246) is preferable because it does not make assumptions about the input.

6.4.2 Extended Basic Case

In the above optimization, we required A to have just one history. What if it has more than one? Consider:

(248) $A : \{1\},\ A : \{2\},\ A \rightarrow A : \{3\},\ A \rightarrow A \rightarrow B : \{4\}$



Here we need the SCC to output all four possibilities: $\{1\}$, $\{1,3\}$, $\{2\}$, $\{2,3\}$. The first and the last will join with $\{4\}$ to form one solution, while $\{1,3\}$ and $\{2\}$ will join to form another solution. Yet, in (248) we see again that there are two completely different ways to get a basic A and to consume it, and such a thing does not occur in practice in a glue semantics set of premises.

In section 13.2.1, however, we shall see that we do get the following case:

(249) $A : \{1\},\ A \rightarrow B \rightarrow C : \{2\},\ A \rightarrow C_{[4,5]} \rightarrow C : \{3\},\ A : \{4\},\ B : \{5\}$

The last three premises are obtained from compiling $A \rightarrow (A \rightarrow B \rightarrow C) \rightarrow C$. Here we have two different A. The premise 3 can combine with each of them, but $\{3,4\}$ is immediately ruled out by the local requirements check (since $C_{[4,5]} \rightarrow C : \{3,4\}$ is defunct). The SCC of C starts with two histories: $\{1,2,5\}$ and $\{2,4,5\}$. However, the first is really a dead end since it cannot combine with the modifier $\{1,3\}$, so we get just $\{1,3\}$ combining with $\{2,4,5\}$ to give $\{1,2,3,4,5\}$. Now we should filter out the two initial histories with

sets $\{1, 2, 5\}$ and $\{2, 4, 5\}$, and this will be done by both optimizations (246) and (243). The difference between (249) and (248) is that the second A in (249) is not independent – it is compiled out of an initial premise – whereas the two A in (248) are both independent.

A similar but more complex case demonstrates that more than one modifier may have the same main index, and only one of them will end up being used. This example comes from section 13.3.3:

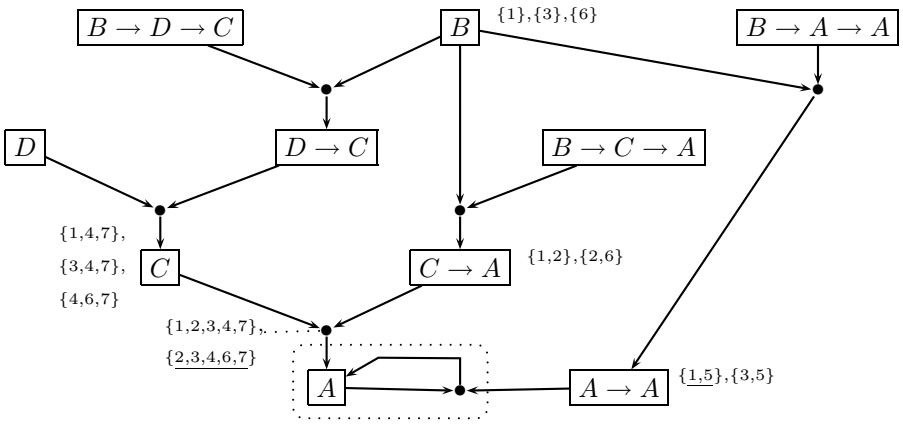
(250) Initial:

$B,\ B \to (B \to C) \to A,\ B \to D \to C,\ B \to (B \to D \to A) \to A$ (goal: A)

Compiled:

$B : \{1\},\ B \to C_{[3]} \to A : \{2\},\ B : \{3\},\ B \to D \to C : \{4\},$

$B \to A_{[6,7]} \to A : \{5\},\ B : \{6\},\ D : \{7\}$



Here there are two histories in $A \to A$ (not three, because $B \to A_{[6,7]} \to A$ requires 6 and 7 and so cannot combine with $B : \{6\}$ by the optimization from section 5.4.1), and there are two initial histories in A. But the histories with the index sets $\{1, 2, 3, 4, 7\}$ and $\{3, 5\}$ are dead ends, and there is only one possible combination of $A \to A$ with A. The main index in $A \to A$ is 5, and both optimizations (246) and (243) work correctly.

Given the situation in (248), how can we fix optimization (246) without relying on the assumption that such situations do not arise in real glue semantics inputs? One idea is to look at the category graph and try to see if there are more than one consumers of the SCC output. It is not enough to check whether there are two or more arrows exiting the SCC, because the split can occur later in the derivation. In particular, if the SCC has category X and it combines with $X \to Y$ to form Y, that Y may have two or more consumers.

This suggests looking at the path in the category graph from the SCC to the goal node. Perhaps if there is just one path, that means the SCC output is consumed just once, and then it is ok to use optimization (246)? The answer is no, as the following example shows:

(251) $X : \{1\}$, $X : \{2\}$, $X \rightarrow X : \{3\}$, $X \rightarrow Y \rightarrow Y : \{4\}$, $X \rightarrow Y \rightarrow Y : \{5\}$, $Y : \{6\}$, $Y \rightarrow Z : \{7\}$

Here, even though there is just one path from X to the goal node Z (when ignoring $X \rightarrow X$), we cannot use optimization (246) on X because the input here yields eight distinct complete derivations. There are eight because there are three binary choices: there is a choice of whether $\{3\}$ combines with $\{1\}$ or with $\{2\}$; there is a choice of whether the first X combines with the first $X \rightarrow Y \rightarrow Y$ and the second with the second or the other matching; and there is a choice of whether Y combines first with a $Y \rightarrow Y$ modifier with index 4 or first with one with index 5.

The problem here is the two initial histories of X that have disjoint spans. This observation leads us to a correct reformulation of optimization (246):

(252) In a basic SCC with node A and modifier $A \rightarrow A$, if there is an index that is shared by all initial histories of A, then apply the following optimization:
Delete a history from the output node A if that history does not include the main index of some history in the $A \rightarrow A$ node.

In examples (248) and (251), because there is no shared index in the two initial histories of nodes A and X, we do not filter any output history. In contrast, optimization (252) does apply in example (249) because the two initial histories of C share the index 2 (and 5 as well). So, if that example had another $C \rightarrow C$ modifier with main index 6, the optimization would make sure this modifier is used in the output histories of C.

Proposition 4 *Optimization (252) is correct.*

Proof: The main index i of a premise whose category ends with $X \rightarrow X$ must be part of *some* output history of node X (because $X \rightarrow X$ can only combine with X, due to the flattening achieved by the compilation stage; i.e. there is no category $(X \rightarrow X) \rightarrow Z$ after compilation).

Now, suppose to the contrary that the optimization is incorrect. So there is some input to the algorithm that yields a complete derivation $\mathcal{D}$ in which a history h_1 in node X

which does not include i should not be filtered. But because of what was said above, $\mathcal{D}$ must include another history h_2 in node X that does include i. Furthermore, because h_1 and h_2 are both part of the same $\mathcal{D}$, their spans must be disjoint. Hence, they came from two initial histories of X with disjoint spans. But this contradicts the condition of the optimization that there is some index that is shared between all initial histories of X. ■

Is optimization (252) strong enough? Note that if we relax the first condition of the optimization to say that each initial history shares an index with *some* other initial history (but not necessarily all others), then the optimization is incorrect. For example:

(253) $A,\ A \to B,\ A \to (A \to B) \to B,\ C,\ C \to B,\ C \to (C \to B) \to B,\ B \to B,$
$B \to B \to E$
compiles to:
$A : \{1\},\ A \to B : \{2\},\ A \to B_{[4]} \to B : \{3\},\ A : \{4\},$
$C : \{5\},\ C \to B : \{6\},\ C \to B_{[8]} \to B : \{7\},\ C : \{8\},$
$B \to B : \{9\},\ B \to B \to E : \{10\}$

The node B will have four initial histories with spans: $\{1, 2\}$, $\{4, 2\}$, $\{5, 6\}$, $\{8, 6\}$, each of which shares an index with another. However, it is incorrect to filter an output history in B if it does not include the index 9.

6.4.3 Basic Case with Higher-Order Modifiers

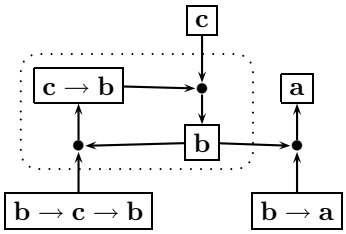
So far we have considered only a basic SCC. What about a larger SCC that is obtained from a non-atomic A and $A \to A$? For example:

(254) [the [Swedish man]$_b$]$_a$
$the : (\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{a}^e$
$swedish : (\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{c}^e \to \mathbf{b}^t$
$man : \mathbf{c}^e \to \mathbf{b}^t$
$\Rightarrow the(swedish(man)) : \mathbf{a}^e$

The compiled premises are:
$t : \mathbf{b}^t_{[2]} \to \mathbf{a}^e : \{1\}$
$v_2 : \mathbf{c}^e : \{2\}$
$s : \mathbf{b}^t_{[4]} \to \mathbf{c}^e \to \mathbf{b}^t : \{3\}$
$v_4 : \mathbf{c}^e : \{4\}$
$m : \mathbf{c}^e \to \mathbf{b}^t : \{5\}$

The category graph is:

(255)



Here the output node of the SCC is **b**, and we need to filter out histories in that node which do not include the index 3, and both optimizations (246) and (243) work correctly. Here is an example with a further level of nesting:

(256) [the [apparently Swedish man]$_b$]$_a$

$the : (\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{a}^e$

$apparently : ((\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{c}^e \to \mathbf{b}^t) \to (\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{c}^e \to \mathbf{b}^t$

$swedish : (\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{c}^e \to \mathbf{b}^t$

$man : \mathbf{c}^e \to \mathbf{b}^t$

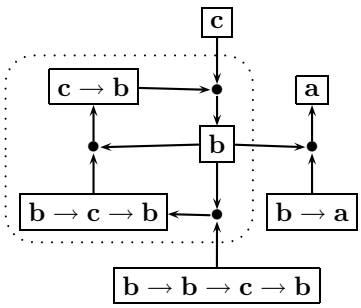
$\Rightarrow the((apparently(swedish))(man)) : \mathbf{a}^e$

The compiled premises are:

1. $t : \mathbf{b}^t_{[2]} \to \mathbf{a}^e : \{1\}$
2. $v_2 : \mathbf{c}^e : \{2\}$
3. $a : \mathbf{b}^t_{[4,5]} \to \mathbf{b}^t_{[6]} \to \mathbf{c}^e \to \mathbf{b}^t : \{3\}$
4. $v_4 : \mathbf{c}^e \to \mathbf{b}^t : \{4\}$
5. $v_5 : \mathbf{c}^e : \{5\}$
6. $v_6 : \mathbf{c}^e : \{6\}$
7. $s : \mathbf{b}^t_{[8]} \to \mathbf{c}^e \to \mathbf{b}^t : \{7\}$
8. $v_8 : \mathbf{c}^e : \{8\}$
9. $m : \mathbf{c}^e \to \mathbf{b}^t : \{9\}$

Here is the category graph, where again the output node is **b**:

(257)



Here is the entire chart derivation inside the SCC:

b	$10 : \{2,4\},\ 11 : \{5,4\},\ 12 : \{6,4\},\ 13 : \{8,4\},$
	$14 : \{2,9\},\ 15 : \{5,9\},\ 16 : \{6,9\},\ 17 : \{8,9\}$
$\mathbf{c} \rightarrow \mathbf{b}$	$22 : \{7,8,4\} : \{\langle 7,13\rangle\},\ 23 : \{7,8,9\} : \{\langle 7,17\rangle\}$
b	$24 : \{2,4,7,8\} : \{\langle 22,2\rangle\},\ 25 : \{5,4,7,8\} : \{\langle 22,5\rangle\},$
	$26 : \{4,6,7,8\} : \{\langle 22,6\rangle\},\ 27 : \{2,7,8,9\} : \{\langle 23,2\rangle\},$
	$28 : \{5,7,8,9\} : \{\langle 23,5\rangle\},\ 29 : \{6,7,8,9\} : \{\langle 23,6\rangle\}$
$\mathbf{b}_{[6]} \rightarrow \mathbf{c} \rightarrow \mathbf{b}$	$30 : \{3,4,5\} : \{\langle 3,11\rangle\},\ 31 : \{3,4,5,7,8\} : \{\langle 3,25\rangle\}$
$\mathbf{c} \rightarrow \mathbf{b}$	$32 : \{3,4,5,6,9\} : \{\langle 30,16\rangle\},$
	$33 : \{3,4,5,6,7,8,9\} : \{\langle 30,29\rangle, \langle 31,16\rangle\}$
b	$34 : \{2,3,4,5,6,9\} : \{\langle 32,2\rangle\},\ 35 : \{3,4,5,6,8,9\} : \{\langle 32,8\rangle\},$
	$36 : \{2,3,4,5,6,7,8,9\} : \{\langle 33,2\rangle\}$
a	$37 : t(\lambda v_2.v_4(v_2)) : \{1,2,4\} : \{\langle 1,10\rangle\},$
	$38 : t(\lambda v_2.m(v_2)) : \{1,2,9\} : \{\langle 1,14\rangle\},$
	$39 : t(\lambda v_2.s(v_4(v_8))(v_2)) : \{1,2,4,7,8\} : \{\langle 1,24\rangle\},$
	$40 : t(\lambda v_2.s(m(v_8))(v_2)) : \{1,2,7,8,9\} : \{\langle 1,27\rangle\},$
	$41 : t(\lambda v_2.a(\lambda v_4.\lambda v_5.v_4(v_5))(\lambda v_6.m(v_6))(v_2)) : \{1,2,3,4,5,6,9\} : \{\langle 1,34\rangle\},$
	$42 : \{1,2,3,4,5,6,7,8,9\} : \{\langle 1,36\rangle\},$
	with meaning:
	$t(\lambda v_2.a(\lambda v_4.\lambda v_5.v_4(v_5))(\lambda v_6.s(\lambda v_8.m(v_8))(v_6))(v_2))$ or:
	$t(\lambda v_2.a(\lambda v_4.\lambda v_5.s(\lambda v_8.v_4(v_8))(v_5))(\lambda v_6.m(v_6))(v_2))$

First, note that there are actually two non-equivalent results (and this is true also in linear logic). After η-reduction they are:

(258) a. $the((apparently(\lambda v_4.v_4))(sweedish(man)))$

b. $the((apparently(sweedish))(man))$

Since this is a correct result of linear logic, if we want to prevent the undesirable (258)a, we should not define adjective modifiers in this way. Instead, we should use the premises in (259), following (Dalrymple, 2001, ch.10):

(259) [the [[[apparently]$_d$ Swedish]$_c$ man]$_b$]$_a$

$the : (\mathbf{b}_v^e \rightarrow \mathbf{b}_r^t) \rightarrow \mathbf{a}^e$

$man : \mathbf{b}_v^e \rightarrow \mathbf{b}_r^t$

$swedish : \mathbf{c}_v^e \rightarrow \mathbf{c}_r^t$

$\lambda Q \lambda P \lambda x.P(x) \wedge Q(x) : (\mathbf{c}_v^e \rightarrow \mathbf{c}_r^t) \rightarrow (\mathbf{b}_v^e \rightarrow \mathbf{b}_r^t) \rightarrow \mathbf{b}_v^e \rightarrow \mathbf{b}_r^t$

$apparently : (\mathbf{c}_v^e \rightarrow \mathbf{c}_r^t) \rightarrow \mathbf{c}_v^e \rightarrow \mathbf{c}_r^t$

$\Rightarrow the(\lambda x.man(x) \wedge (apparently(swedish))(x)) : \mathbf{a}^e$

The intersective connector can combine with *swedish* to form the noun-modifier:

(260) $\lambda P \lambda x.P(x) \wedge swedish(x) : (\mathbf{b}_v^e \rightarrow \mathbf{b}_r^t) \rightarrow \mathbf{b}_v^e \rightarrow \mathbf{b}_r^t$

However, *apparently* must first apply on *swedish* to form $apparently(swedish) : \mathbf{c}_v^e \rightarrow \mathbf{c}_r^t$ before the intersective connector applies on that. So in contrast to (256), here we get only the desired result. It also makes more sense to define things as in (259), because the resulting meaning term in (259) is more clear than the higher-level *apparently* in (258)b. In general, genuine cases that require such higher-order modifiers are very rare, if they exist at all.

As a side note, we should point out that in his paper, Hepple builds into his algorithm a condition that prevents results such as (258)a. He does it by defining the side condition in the rule (212) to be $L \subset S_1$ rather than $L \subseteq S_1$. Here, this will prevent element 3 from combining with element 11, which will eliminate the path that led to (258)a. Hepple says that this move is according to the general practice in categorial grammar, where each deduction step rests on at least one real premise (not just discharged assumptions). The simplest case where this difference can be seen is in the linear logic proof of $(A \rightarrow A) \rightarrow B \vdash B$:

(261)
$$\dfrac{\dfrac{[x : A]_i}{\lambda x.x : A \rightarrow A}\,\mathrm{I}_i \qquad m : (A \rightarrow A) \rightarrow B}{m(\lambda x.x) : B}\,\mathrm{E}$$

The proof is a bit "strange" because $[x : A]_i$ is an assumption that yields itself as a conclusion. The full algorithm will find this proof: the premise is compiled out to $A_{[i]} \rightarrow B$ and $A : \{i\}$, which would then combine to form B. The modified rule, with $L \subset S_1$, will refuse to combine the two complied premises.

Now let us return to the matter of how to optimize cycles. For the graph (255) we need to filter a history if it does not use the modifier which is outside the SCC (i.e. index 3). This filters 14 elements out of the 17 elements in node **b**. Only elements 34, 35, and 36 are left. Element 35 cannot combine with the premise 1 because the latter requires the presence of index 2. But we are still left with the incomplete element 34. To filter it out by using optimization (246), we need to require that all modifiers that appear within the SCC also need to be used by final histories. This would rule out element 34 since it does not have index 7. Optimization (243) would simply filter element 34 since its index set is a subset of element 36.

6.4.4 Simplification of Higher Order Modifiers

As you can see in the chart derivation for (256), there is a massive amount of useless work (a lot of dead ends that are computed). There is a simpler solution: not to compile modifier categories of the form $X \to X$ but leave them as they are. When we recursively go through an input glue premise of category $C_1 \to (C_2 \to (\ldots \to C_n)\ldots)$, we compile the categories as we go along, except that when we reach the subcategory $C_k \to (C_{k+1} \to (\ldots \to C_n)\ldots)$, we do not compile it, nor anything inside it, if it has the form $X \to X$. For example, the fourth premise in (259) will be complied to (262)a rather than (262)b.

(262) a. $\mathbf{c}^t_{r\,[i]} \to (\mathbf{b}^e_v \to \mathbf{b}^t_r) \to \mathbf{b}^e_v \to \mathbf{b}^t_r$
$\mathbf{c}^e_v : \{i\}$

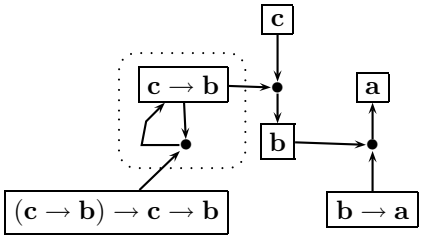
b. $\mathbf{c}^t_{r\,[i]} \to \mathbf{b}^t_{r\,[j]} \to \mathbf{b}^e_v \to \mathbf{b}^t_r$
$\mathbf{c}^e_v : \{i\}$
$\mathbf{b}^e_v : \{j\}$

By using such a compilation, (254) now becomes:

(263) $t : \mathbf{b}^t_{[2]} \to \mathbf{a}^e : \{1\}$
$v_2 : \mathbf{c}^e : \{2\}$
$s : (\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{c}^e \to \mathbf{b}^t : \{3\}$
$m : \mathbf{c}^e \to \mathbf{b}^t : \{4\}$

and the graph is:

(264)



Here the derivation will be very simple: 3 combines with 4 to get $\{3, 4\}$, which then combines with 2, and then with 1.

Even though we said that we prefer (259) to (256), let us still consider what happens with (256). Now it becomes:

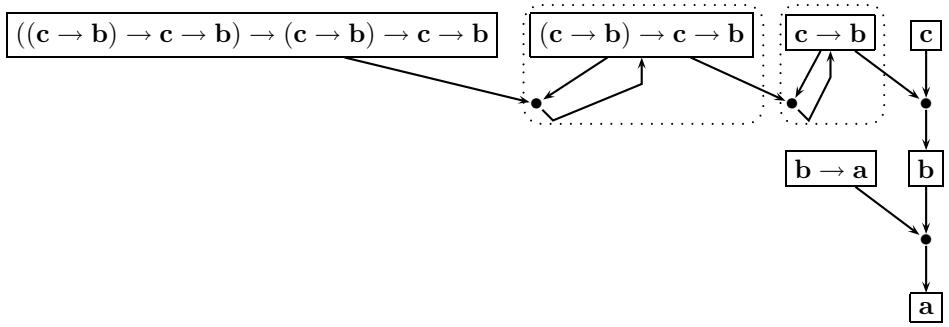
(265) [the [apparently Swedish man]$_b$]$_a$
the : $(\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{a}^e$
apparently : $((\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{c}^e \to \mathbf{b}^t) \to (\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{c}^e \to \mathbf{b}^t$
swedish : $(\mathbf{c}^e \to \mathbf{b}^t) \to \mathbf{c}^e \to \mathbf{b}^t$
man : $\mathbf{c}^e \to \mathbf{b}^t$

The compiled premises are:

1. $t : \mathbf{b}^t_{[2]} \multimap \mathbf{a}^e : \{1\}$
2. $v_2 : \mathbf{c}^e : \{2\}$
3. $a : ((\mathbf{c}^e \multimap \mathbf{b}^t) \multimap \mathbf{c}^e \multimap \mathbf{b}^t) \multimap (\mathbf{c}^e \multimap \mathbf{b}^t) \multimap \mathbf{c}^e \multimap \mathbf{b}^t : \{3\}$
7. $s : (\mathbf{c}^e \multimap \mathbf{b}^t) \multimap \mathbf{c}^e \multimap \mathbf{b}^t : \{4\}$
9. $m : \mathbf{c}^e \multimap \mathbf{b}^t : \{5\}$

Here is the category graph, where again the output node is **b**:

(266)



Again the derivation is much simpler than for (255). First 3 combines with 4 to get $\{3, 4\}$, then this combines with 5, then with 2, and finally with 1. This is a major saving and simplification.

Of course, the graphs (264) and (266) are not as general as (255) and (257). Can we lose any derivations? In arbitrary linear logic, the answer is yes. If we add to (254) a premise $k : \mathbf{c} \multimap \mathbf{c}$, then there are two histories with category **c** which can be modified, and so we get two results:

(267) a. $t(s(\lambda x.m(k(x))))$

b. $t(\lambda x.(s(m))(k(x)))$

In contrast, (263) has only one history in category **c** there that can be modified, so we only get (267)b from (263). However, such a direct modification of the variable of a noun is never used in a glue semantics specification. Similar differences occur in (266). Incidentally, with the graph (266), we do not even get (258)a.

To summarize section 6.4, the optimization of modifiers which prevents exiting an SCC without using all the required material consists of the following:

- In the compilation phase, do not compile modifier categories $X \multimap X$.
- In a basic SCC, use optimization (252) (where A need not be atomic).

6.5 Inside a Basic Cycle

The optimization above confines the exponential explosion to the SCC and prevents it from spreading to the rest of the derivation after the SCC. However, should we retain this explosion within the cycle?

We should think about this question from the perspective of a related question: How should the packed representation of the modifier ambiguity look like? There should be a one-to-one correspondence between the structure of the derivation and the structure of the packed representation. I first consider an answer that uses choice-space packing, and then an answer that uses an underspecified representation.

6.5.1 Packing Using the Choice-Space

In the example from section 6.1, there are two final complete derivations, and they yield the following two terms:

(268) a. $d(c(b(a)))$
b. $d(b(c(a)))$

These terms share the parts a, b, c, and d, and only the order of application of b and c is different. In a packed representation, we want each of the four parts to appear just once. If we use choice-space style packing, we get:

(269) $l_1 : d(\langle A1 : l_2(l_3(l_4)),\ A2 : l_3(l_2(l_4))\rangle)$
$l_2 : b$
$l_3 : c$
$l_4 : a$

The top term l_1 says that d applies on a term which is ambiguous between $b(c(a))$ and $c(b(a))$. The top context 1 in the choice-space is split into two mutually-exclusive choices $A1$ and $A2$. Although here the gain may not be apparent because it may seem the overhead is larger than an explicit representation, remember that each of the four parts may be a complex semantic term, so in general there is a gain.

With three modifiers, we have more options to represent. We may use a six-way split:

(270) $l_1 : e(\langle A1 : l_2(l_3(l_4(l_5))),\ A2 : l_2(l_4(l_3(l_5))),\ \ldots,\ A6 : l_4(l_3(l_2(l_5)))\rangle)$
$l_2 : d$

$l_3 : c$

$l_4 : b$

$l_5 : a$

or a three-way split followed by three two-way splits:

(271) $l_1 : e(\langle A1 : l_2(\langle A1.1 : l_3(l_6),\ A1.2 : l_4(l_7)\rangle),$
$A2 : l_3(\langle A2.1 : l_2(l_6),\ A2.2 : l_4(l_8)\rangle),$
$A3 : l_4(\langle A3.1 : l_2(l_7),\ A3.2 : l_3(l_8)\rangle)\ \rangle)$

$l_2 : d$ $\quad$ $l_6 : l_4(l_5)$
$l_3 : c$ $\quad$ $l_7 : l_3(l_5)$
$l_4 : b$ $\quad$ $l_8 : l_2(l_5)$
$l_5 : a$

Here the top context is split between $A1, A2, A3$, according to what is the top modifier (d, c, or b), and then $A1$ is split between $A1.1$ and $A1.2$, and so on. There are also other possible organizations or splits in the choice space.

The modifiers above (such as b) are given in η-reduced form. In general, more complex modifiers will need to be in η-expanded form. For example, a modifier may be $\lambda P.every(man, P)$.

With four modifiers we get:

(272) $l_1 : f(\langle A1 : l_2(\langle A1.1 : l_3(l_6),\ A1.2 : l_4(l_7),\ A1.3 : l_5(l_8)\rangle),$
$A2 : l_3(\langle A2.1 : l_2(l_6),\ A2.2 : l_4(l_{10}),\ A2.3 : l_5(l_{11})\rangle),$
$A3 : l_4(\langle A3.1 : l_2(l_7),\ A3.2 : l_3(l_{10}),\ A3.3 : l_5(l_{12})\rangle),$
$A4 : l_5(\langle A4.1 : l_2(l_8),\ A4.2 : l_3(l_{11}),\ A4.3 : l_4(l_{12})\rangle)\ \rangle)$

$l_2 : e$ $\quad$ $l_6 : \langle B1 : l_4(l_{13}),\ B2 : l_5(l_{14})\rangle$ $\quad$ $l_{13} : l_5(l_0)$
$l_3 : d$ $\quad$ $l_7 : \langle C1 : l_3(l_{13}),\ C2 : l_5(l_{15})\rangle$ $\quad$ $l_{14} : l_4(l_0)$
$l_4 : c$ $\quad$ $l_8 : \langle D1 : l_3(l_{14}),\ D2 : l_4(l_{15})\rangle$ $\quad$ $l_{15} : l_3(l_0)$
$l_5 : b$ $\quad$ $l_{10} : \langle E1 : l_2(l_{13}),\ E2 : l_5(l_{16})\rangle$ $\quad$ $l_{16} : l_2(l_0)$
$l_0 : a$ $\quad$ $l_{11} : \langle F1 : l_2(l_{14}),\ F2 : l_4(l_{16})\rangle$
$l_{12} : \langle G1 : l_2(l_{15}),\ G2 : l_3(l_{16})\rangle$

where:

$A1.1 \vee A2.1 \leftrightarrow B1$ xor $B2$

$A1.2 \vee A3.1 \leftrightarrow C1$ xor $C2$

...

$A3.3 \vee A4.3 \leftrightarrow G1$ xor $G2$

Here we have a new thing: The split under labels l_6 through l_{12} is not a split of a simple context but of a composite context such as $A1.1 \vee A2.1$. This means that under each of $A1.1$ and $A2.1$, there is a choice between $B1$ and $B2$. However, under any context other than $A1.1$ or $A2.1$, this choice is meaningless.

6.5.2 Calculating Packed Meanings

Obtaining these representations is done as follows. Inside each SCC, we run the basic chart algorithm in order to obtain all possible results. The output histories of the SCC are those that survive the filtering of optimization (252).

In order to calculate the packed meaning expression for the entire history forest, we need to generalize the procedure from section 6.1. Whenever a history has $n > 1$ parent-links, we want to create a n-way split in the choice space, one for each parent-link. Furthermore, certain histories h can be reached from more than one other history (h appears in more than one parent-link), and so when the recursion reaches such an h, we do not want to enter that h for two reasons. First, we will reach this h more than once, but we want to process that h only once. Second, the choice-space context of h can be determined only after all the paths to h have been followed. For example, the context of l_6 in (272), which is $A1.1 \vee A2.1$ can be determined only after l_6 was reached in the two separate ways. Therefore, the steps we need are as follows.

- Step 1: Start from the graph's goal category, and recursively traverse all the histories through the parent-links, to detect which histories actually participate in a final derivation (i.e. do not lead to a dead-end). The histories form a DAG through their parent-links. Order these histories in a topological sort L, so that if $h = \langle C, S, \{\ldots, \langle h_1, h_2\rangle, \ldots\}\rangle$ then h appears before h_1 and h_2 in the list ("top-down" order). Furthermore, detect which histories appear in more than one parent-links of other histories (they have at least two outgoing "up" arrows). Order these according to L and put in L'.

- Step 2: Now start from the goal node's histories, and start "flowing" context 1 down parent-links. Whenever you are in context C, and encounter a history with n parent-links for $n > 1$, split C into n subcontexts, one for each parent-link. Moreover, when you are in context C and reach a history h in L', just add C to it, but do not enter h at this time. Only once you finish the current run, consider the next element in L'.

That element will have at least two contexts that were put on it from above. Disjoin all these contexts to get a new current context. Now start flowing the context again staring from that element.

- Step 3: Assign a unique label l_h to each history h. Furthermore, for each elementary history $h = \langle C, \{i\}, \{null\}\rangle$, set $l_h = m$, where m is the meaning term of premise i. For each history $h = \langle C, S, \{\langle h_1, h_2\rangle\}\rangle$ with just one parent-link, set $l_h = app(l_{h_1}, l_{h_2})$ (application). And for each history $h = \langle C, S, \{C_1 : \langle p_1, q_1\rangle, \ldots, C_n : \langle p_n, q_n\rangle\}\rangle$, where $C_1, \ldots, C_n$ are the subcontexts that were created in step 2 (with $n > 1$), set $l_h = \langle C_1 : app(l_{p_1}, l_{q_1}), \ldots, C_n : app(l_{p_n}, l_{q_n})\rangle$.

- Step 4: If a label l_h appears just once in some other term, replace its occurrence with its meaning term. If a label appears more than once, replace its occurrences with its meaning term only if that meaning term is atomic (variable or constant); otherwise, do not replace the label's occurrences.

- Step 5: Calculate λ-applications (β-reductions) and η-reductions.

For example, with three modifiers we will get a result as in (271). The initial history of the SCC will have just the index of a. This history then combines with each of the histories of b, c, and d. The first of these resulting histories combines with the history of c, to give a new history h. Thanks to (normal) chart packing, that history also has another parent-link because it can be obtained also from first combining a with c and then with b. Hence, h will correspond to the form $\langle A1.1 : l_3(l_6),\ A1.2 : l_4(l_7)\rangle$. Similarly with the other histories at that level.

The contexts will be calculated correctly. When we go top down in step 2, we first encounter the final history of the SCC and since it has three parent-links, we split the top context between $A1$, $A2$, and $A3$. Then we follow the first parent-link and reach history h. We then split $A1$ into $A1.1$ and $A1.2$.

In (272), the final SCC history will have a 4-way split. Its first parent-link leads to a history with a 3-way split. The first parent-link of that history leads to a history that corresponds to label l_6. Since there is another possible path to that history, we just put on it the context $A1.1$ but do not enter the history. Later we arrive at that history again while "flowing" context $A2.1$ down. Once all these ways of arriving there are explored, we

disjoin these contexts to create a new context $A1.1 \vee A2.1$, which can then be split further to $B1$ and $B2$.

6.5.3 The Problem with Choice-Space Packing

This four-modifier case looks very complicated, but it is in fact parallel to what happens in a chart parser. For example, in the NP

(273) the expensive red house on the street in Seattle

the noun "house" has four modifiers (let us ignore here the possibility that "in Seattle" may modify "street"). For the substring "red house on the street", there will be two possible ways to form a N': (red+house)+PP or red+(house+PP). This N' parallels the l_6 through l_{12} labels in (272) in the following sense: The N' can be obtained in two ways, but larger constituents that use this N' (i.e. expensive+N' or N'+in_Seattle) do not care about the internal structure of this N'. Similarly, l_6 has two internal options, but the larger constituents $l_3(l_6)$ and $l_2(l_6)$ in options $A1.1$ and $A2.1$ do not care about the internal structure of l_6.

For the substrings of (273) with three modifiers, there are three possibilities in the chart parser, and for the entire N', there are six. For n modifiers we do not get $n!$ possibilities in the chart parser but at most n^3 because of the constraints imposed by the linear order of the modifiers (e.g. we cannot combine "expensive" before using "red" first). In contrast, we do potentially get $n!$ possibilities when a node in the category graph has n modifiers.[2]

The problem with using the choice-space mechanism for this kind of ambiguity is that with n modifiers we need at least $n!$ splits in the choice-space. The reason that this exponential explosion in the choice-space happens here but not with most syntactic ambiguities is that the different options here are not independent – they are highly interdependent. If a modifier is used in one place, it cannot be used in another. The choice space idea works well only for independent ambiguities, such as those in section 4.2.1. The worst case is exponential, and that is what happens here.

There are two strategies to deal with this. In strategy 1, we accept this situation and note that although there is an explosion, it is local: The explosion affects only the point

[2]To get only the correct results in the case of nominal modifiers in glue semantics, we use scoping constraints that are based on the linear position of the modifiers, as we saw in section 4.6.2.

in a representation that has modifiers, but not places where there is only one option for pieces to combine. For example, in an input like:

(274) $a : A$, $b : A \rightarrow B$, $c : B \rightarrow C$, $d : C \rightarrow D$, $d_1 : D \rightarrow D$, $d_2 : D \rightarrow D$, $\ldots$, $d_n : D \rightarrow D$, $e : D \rightarrow E$, $f : E \rightarrow F$, $g : F \rightarrow G$

the parts $d(c(b(a)))$ and $g(f(e(\ldots)))$ appear just once, and their computation is also done just once. Other things that can help will be discussed below.

6.5.4 Packing Using Underspecified Forms

In strategy 2, we create an underspecified representation (UR) rather than a packed representation. That is, instead of (272), we create a representation as in (275)a, and for (274), the UR is (275)b.

(275) a. $f(mod\_str(a, [b, c, d, e]))$
b. $g(f(e(\ mod\_str(d(c(b(a))), [d_1, d_2, \ldots, d_n])\ )))$

When all modifiers are freely permutable as above, it is difficult to see any advantage to the choice-space strategy. That strategy goes exponential whereas it seems the underspecified strategy does not require any calculation – just put all modifiers in a list.

However, modifiers are not always freely permutable. Consider again example (98) from section 3.3.2, and suppose we instantiate both H^t and G^t to $\mathbf{1}^t$. So we have (renaming the categories to simplify the discussion here, and merging "a city"):

(276) Input premises:

$i : \mathbf{b} \rightarrow \mathbf{a}$, $e : (\mathbf{d} \rightarrow \mathbf{e}) \rightarrow (\mathbf{b} \rightarrow \mathbf{a}) \rightarrow \mathbf{a}$, $m : \mathbf{d} \rightarrow \mathbf{c} \rightarrow \mathbf{e}$, $c : (\mathbf{c} \rightarrow \mathbf{a}) \rightarrow \mathbf{a}$

Compiled premises:

1. $i : \mathbf{b} \rightarrow \mathbf{a} : \{1\}$
2. $e : \mathbf{e}_{[3]} \rightarrow \mathbf{a}_{[4]} \rightarrow \mathbf{a} : \{2\}$
3. $v_3 : \mathbf{d} : \{3\}$
4. $v_4 : \mathbf{b} : \{4\}$
5. $m : \mathbf{d} \rightarrow \mathbf{c} \rightarrow \mathbf{e} : \{5\}$
6. $c : \mathbf{a}_{[7]} \rightarrow \mathbf{a} : \{6\}$
7. $v_7 : \mathbf{c} : \{7\}$

Derivation:

8.	$i(v_4) : \mathbf{a} : \{1,4\}$	[1+4]
9.	$m(v_3) : \mathbf{c} \rightarrow \mathbf{e} : \{3,5\}$	[5+3]
10.	$m(v_3, v_7) : \mathbf{e} : \{3,5,7\}$	[9+7]
11.	$e(\lambda v_3.m(v_3, v_7)) : \mathbf{a}_{[4]} \rightarrow \mathbf{a} : \{2,3,5,7\}$	[2+10]

At this point, the three relevant elements are:

(277)
8.	$i(v_4) : \mathbf{a} : \{1,4\}$
6.	$c : \mathbf{a}_{[7]} \rightarrow \mathbf{a} : \{6\}$
11.	$e(\lambda v_3.m(v_3, v_7)) : \mathbf{a}_{[4]} \rightarrow \mathbf{a} : \{2,3,5,7\}$

The two modifiers can be written as:

(278) $\lambda P.c(\lambda v_7.P)$
$\lambda P.e(\lambda v_3.m(v_3, v_7))(\lambda v_4.P)$

However, it would be incorrect to simply put these modifiers in a *mod_str* as follows:

(279) $mod\_str(\ i(v_4),\ [\lambda P.c(\lambda v_7.P),\ \lambda P.e(\lambda v_3.m(v_3, v_7))(\lambda v_4.P)]\ )$

This structure as it is written here can be expanded into the two options:

(280) a. $c(\lambda v_7.e(\lambda v_3.m(v_3, v_7))(\lambda v_4.i(v_4)))$
b. $e(\lambda v_3.m(v_3, v_7))(\lambda v_4.c(\lambda v_7.i(v_4)))$

Notice that (280)b is an undesirable result because v_7 appears free (and $\lambda v_7 \ldots$ binds nothing). The chart algorithm in fact prevents this result because element 6 above cannot directly combine with element 8, since the $\mathbf{a}_{[7]}$ in element 6 requires index 7 which does not appear in the index set of element 8. Therefore, following the choice-space strategy above will not lead to a problem. There will simply be just one way to combine the modifiers in the SCC, and so each history in the SCC will have just one parent-link.

Can we repair the underspecified representation? In (Gupta and Lamping, 1998), any derivation that ends in $X \rightarrow X$ is not allowed to combine with the main skeleton derivation (which is computed in linear time), thus obtaining an underspecified derivation. The derived modifiers can then be inserted into the skeleton derivation in different orders. Although Gupta and Lambing mention the issue of incorrect derivations as shown above, they do not flesh out the details of how to deal with them or how to represent the constraints.

MRS (Copestake et al., 2005) creates underspecified representations that allow free permutations of modifiers. To address the problem of dependencies between modifiers, their strategy is to first freely generate all possibilities and then filter any form that has unbound variables. However, this is an unprincipled solution that lacks justification (more on this point in section 7.7.3). For example, suppose that in some version of the system, some bug is introduced in the algorithm or the syntax-semantics specification, which accidentally generates a form with unbound variables. This will be hard to detect.

In contrast, we can resolve this problem by writing the dependencies in the representation. For example, instead of (279) we will have

(281) $mod\_str(\ i(v_4),\ [\lambda P.c(\lambda v_7.P),\ \lambda P.e(\lambda v_3.m(v_3,v_7))(\lambda v_4.P)],\ [1>2])$

The constraint $1 > 2$ indicates that the first modifier must outscope the second, i.e. must apply after the second does. With this constraint, there is only one possibility of applying the modifiers, and so (281) describes only (280)a and not (280)b. In "Every mayor of a city was indicted for accepting three bribes", we would have the UR:

(282) $mod\_str(\ i(v_4,v_9),\ [\lambda P.c(\lambda v_7.P),\ \lambda P.e(\lambda v_3.m(v_3,v_7))(\lambda v_4.P),\ \lambda P.b(\lambda v_9.P)],\ [1>2])$

With the constraint taken into account, this can be resolved not to $3! = 6$ possibilities, but only 3:

(283) $b(\lambda v_9.c(\lambda v_7.e(\lambda v_3.m(v_3,v_7))(\lambda v_4.i(v_4,v_9))))$
$c(\lambda v_7.b(\lambda v_9.e(\lambda v_3.m(v_3,v_7))(\lambda v_4.i(v_4,v_9))))$
$c(\lambda v_7.e(\lambda v_3.m(v_3,v_7))(\lambda v_4.b(\lambda v_9.i(v_4,v_9))))$

(In the choice space strategy, we will simply get a three-way split in the final history of the SCC, and each of these will lead to a chain with no splits).

6.5.5 Calculating Underspecified Meanings

How do we actually compute the underspecified representations in strategy 2?

We can calculate ordering constraints between modifiers as follows. A history $h_1 = \langle A_{L_1} \to A, S_1, T_1\rangle$ must be used after a history $h_2 = \langle A_{L_2} \to A, S_2, T_2\rangle$ (if they can both be used together) if $L_1 \cap S_2 \neq \emptyset$. This is because L_1 requires the presence of some index (or indices) in A, and if at least one of them appears in S_2 then h_2 must be combined with

A before h_1 can. Hence, in such a case, we would add a constraint $j > k$ to the *mod_str*, where j and k are the places of the modifiers h_1 and h_2 in the *mod_str* list.

Can we spare the local exponential computation? An initial attempt is simply to collect all the modifiers into the list of the *mod_str*, and add the ordering constraints. The base of the *mod_str* would be determined by the initial history in A. However, there may be more than one initial histories, although some of them may not end up contributing to a final history, as we saw in examples (249) and (250). Some of the modifier histories too may not contribute to a final history in the SCC, as we saw in example (250).

One option is then to allow the local exponential explosion of the computation inside the SCC (since it does not spread outside the SCC, and since in practice the number of modifiers is small). We then look at the (filtered) final histories. Each of them tells us what the base history was and which of the modifiers actually combined with it, and so we can put these into a *mod_str* together with any ordering constraints between the modifiers.

Is there a way to improve on this? In particular, if there is just one base history, and the n modifiers do not have ordering constraints, we want an $O(n)$ computation rather than $O(n!)$.

A truly packed computation that avoids calculating all possibilities builds an "ordered filtering lattice". This lattice has one bottom node that corresponds to A. Then, the ordering between modifiers, as determined above, is a partial order between the modifiers, which determines their place in the lattice. If all the modifiers are freely permutable vis-a-vis these ordering constraints, then the bottom node will be connected to each of the modifiers directly. But if one modifier has to be used after another modifier, the former will appear above the latter in the lattice. We also add a "mux edge" connecting two modifiers if their index sets have a non-empty intersection, indicating that both modifiers cannot be used together.

Now, for each initial history, we try to advance it through the lattice. For an initial history to survive, it must pass through at least one path. For a modifier-path to survive, at least one initial history should be able to pass all the way through the path. For example, in (250), the initial history $\{1, 2, 3, 4, 7\}$ in the SCC of A does not survive because it cannot be combined with any of the modifiers. Also, the modifier $\{3, 5\}$ does not survive because none of the two initial histories can be passed through it.

We then collect the surviving histories and the surviving modifiers. We now need to take into account the mux edges, and to filter initial histories that cannot be combined

with enough of the modifiers. At least in cases that have no or few dependencies, the computation time will be close to linear in the number of modifiers. However, in the worst case, we may not be able to avoid doing as much work as the chart algorithm of checking all possible combinations.

6.5.6 Summary

How should we choose between the two strategies? The second may be more compact in some cases, but it has several disadvantages. First, it introduces a new mechanism of ambiguity management (the *mod_str*) that is different from the choice-space mechanism. Therefore, any treatment of ambiguity resolution will need to always take care of both mechanisms. Their interaction may be complex. Also, trying to avoid the local explosion requires a complicated procedure, and may not always be possible. The complication will be even greater when we take into account external ambiguity in the next chapter.

Second, with a packed representation, any particular result can be obtained in linear time by simply making choices in the choice-space (a few additional lambda-applications of the modifiers may still be needed, but they are easy to compute). In contrast, the information in a UR is more implicit. In order to get a particular representation from the UR, we need to solve the dominance constraints in the UR. This additional step of indirection makes it more difficult to see how we could extend the idea of packing to do packed inference with the packed representations (of the kind discussed at the end of section 1.3.3).

Third, if we follow the UR strategy, we lose the advantage of using the choice-space mechanism that was mentioned in section 4.2.1, namely that one can efficiently compute the number of analyses without having to enumerate each one, which is important for efficient stochastic disambiguation of packed structures. The reason is that by looking at the underspecified representation, it is not immediately apparent how many solutions the set of dominance constraints has.

Finally, remember what I set out to do: to calculate all analyses in a fully fleshed-out yet packed way, using the choice space, such that common parts are calculated and represented just once. This is in fact what strategy 1 here does (together with the algorithm in the next chapter). Remember that the explosion within a basic cycle does not spread out of the cycle. The fact that there are exponentially many results within a basic cycle

should not by itself force us to change the pure algorithm. In other words, it is not the responsibility of the ambiguity management scheme to deal with this problem.

In practice, in a certain context of application, we do not really get exponentially many distinct *relevant* readings for a sentence. This is due to a combination of reasons that have to do with language use in humans and limits on the possible complexity they can handle. Theoretically, there is no limit to the number of permuting modifiers in a node, including: the number of adjectives and post-nominal modifiers for a noun; and the number of quantifiers in arguments and modifiers to one verb. But in practice, a large number of permuting modifiers for one node is very rare. Therefore, the exponential explosion will be small (and local to the cycle). Even in cases such as:

(284) Three men thought that four women saw that five boys concluded that ...

where theoretically, all the quantifier may float to the top level and interact with each other, it is a good bet in practice to limit to a very small number how many nested clauses an embedded quantifier may float above.

What we should do, then, is identify methods by which humans rule out theoretically possible scopings and interleave these methods into the SCC algorithm as an additional component (rather than evading the question by using a UR instead of the choice-space strategy). Sections 6.6.1 and 6.6.2 below will discuss such methods for restricting the range of possibilities by detecting logical equivalences between options or by imposing heuristical and statistical filtering.

Another Idea

There is another idea for packing, suggested by Dick Crouch, that uses the choice-space more efficiently. Suppose we have four modifiers that can freely permute. We can visualize them as boxes that at first are arranged next to each other along a line in positions 1 to 4. We can stack these boxes on top of each other, or on top of position 0. We start from the highest-numbered box. Box4 has four options: we can stack it on any of positions 0 to 3. Then, Box3 has three options: we can stack it on any of positions 0 to 2. If we first decided to stack Box4 on top of Box3, then when we move Box3, we also move Box4 that is stacked on top of it. Then for Box2 and the stack above it there are 2 options, and finally, for Box1 and its stack there is just one option, to be put on top of whatever stack there is in position 0.

For example, to get the order where the final stack in position 0 has, from bottom to top, Box2, Box4, Box1, and Box3, we follow the steps: stack(4,2), stack(3,1), stack(2,0), stack(1,0). The third step takes the stack consisting of Box4 on Box2 and puts it in position 0. The fourth step takes the stack consisting of Box3 on Box1 and puts it above the Box2+Box4 stack, in position 0.

The number of choices is $4 \times 3 \times 2 \times 1$, which gives us all 4! possible permutations of the final stack. However, we can represent these choices in the choice space with only $O(n^2)$ choices rather than $O(n!)$. We split the choice space to choices $1 : \textit{one-of}(D1, D2, D3, D4)$ for the four choices of Box4. We make a further, independent split, of $1 : \textit{one-of}(C1, C2, C3)$ for the Box3 choices. Finally, another split $1 : \textit{one-of}(B1, B2)$ for the Box2 choices. Each of these choices is independent of the others, and so we get all permutations.

There are a few open questions about this idea. What happens when we have dependencies between the choices, as we saw in section 6.5.4? What is the algorithm that translates these dependencies to nogoods in the choice space? How big does the choice space become as a result of re-factoring using these nogoods? Does the algorithm need to calculate all possibilities as we did in the chart algorithm and had to do in the worst case with the UR approach? How does this interact with the packing in the next chapter (section 7.4.5)? Finally, how does the packed meaning representation itself look like? Just as with the underspecified approach above, and unlike the choice-space approach above, even if one maximally-specific choice in the choice-space is chosen, we do not automatically get a normal (unpacked) meaning representation. We first need to compute an additional step of stacking the modifiers. What are the consequences of this for the next stage of packed inference with packed meaning representations?

6.6 Additional Optimizations

6.6.1 Detecting Logical Equivalences

When a sentence has a scope ambiguity between quantifiers or other operators, different scopings may still yield meanings that are logically equivalent. For example, in the sentence "A man saw a woman", one scoping would correspond to $\exists x.man(x) \wedge \exists y.woman(y) \wedge see(x, y)$, and the other reading to the reverse order of the quantifiers. With more than two quantifiers, the number of equivalent readings may be much larger.

An inefficient way of identifying these equivalences is to calculate all the readings and then ask automated reasoners whether two readings are logically equivalent. With m readings, this requires $O(m^2)$ queries (and with n modifiers, m may be as much as $n!$ to begin with!).

A better method is shown in (Chaves, 2003). Although it does not detect all equivalent readings, in practice it does detect many cases. For example, in the sentence "A man with a hat saw a house of a professor", the optimization is significant: only one reading is produced instead of 14 (not all the $4! = 24$ orderings of the four quantifiers are possible, because of the hierarchical structure of the noun phrases).

The method was shown by Chaves for Hole Semantics (Blackburn and Bos, 2005a), but we can adapt it to our algorithm. The main idea is that if two scopings differ only with respect to the order of *adjacent* operators that have the same logical force then we know that the scopings must yield logically equivalent formulas, and so we can filter one of them. For example, the sentence "A man gave every woman a present" has 3!=6 scopings, which can be written as:

(285) a. $\exists_1 > \forall > \exists_2$
b. $\exists_2 > \forall > \exists_1$
c. $\forall > \exists_1 > \exists_2$
d. $\forall > \exists_2 > \exists_1$
e. $\exists_1 > \exists_2 > \forall$
f. $\exists_2 > \exists_1 > \forall$

There are only four non-equivalent readings, because (285)c,d are logically equivalent to each other, and so are (285)e,f. We can detect this because in (285)c,d, we have two adjacent quantifiers with the same logical force ($\exists_1$ and $\exists_2$), and the same holds in (285)e,f. To filter, we index all the operators, and if two adjacent operators have the same logical force then we allow the scoping only if the higher-indexed operator outscopes the lower-indexed one. Thus, we filter (285)c and (285)e.

We need to define exactly what *logical force* means and what *adjacent* means. For the first, we notice that the two scopings of:

(286) Every sculpture must be exhibited.
$must(every(sculpture, exhibited))$
$every(sculpture, \lambda x.must(exhibited(x)))$

are logically equivalent. Similarly if we replace *every*, *must* with *some*, *may*. We therefore divide operators into those with universal force, those with existential force, and all others. We can define the logical force of an operator in the glue specification of chapter 4.

For adjacency, it we follow the strategy of computing all results in an SCC, then when we combine one modifier with A and then another, we check that if they have the same logical force, the main index of the first is lower than that of the second. If we decide to follow the underspecified representation strategy, we can write additional constraints in the *mod_str*. For modifiers i and j, such a constraint may say that j may not *immediately* dominate i (this still allows j to dominate i non-immediately).

Furthermore, we notice that in

(287) Some relative of some student arrived.

a. $some(\lambda x.some(student, \lambda y.relative\text{-}of(x, y)), arrive)$

b. $some(student, \lambda y.some(\lambda x.relative\text{-}of(x, y), arrive))$

the two readings are logically equivalent. However, in

(288) Every relative of every student arrived.

a. $every(\lambda x.every(student, \lambda y.relative\text{-}of(x, y)), arrive)$

b. $every(student, \lambda y.every(\lambda x.relative\text{-}of(x, y), arrive))$

the two readings are not logically equivalent. The first, which is the less likely reading, claims something about every person who happens to be a relative of every student. The second reading entails the first, but not vice versa. Therefore, we can add an optimization that says: when we combine an existential quantifier with its restrictor, we filter the combination if the restrictor's body is headed by another existential operator. This will filter (287)a and also (289)a,b.

(289) Some relative of some student saw every play.

a. $some(\lambda x.some(student, \lambda y.relative\text{-}of(x, y)), \lambda x.every(play, \lambda z.see(x, z)))$

b. $every(play, \lambda z.some(\lambda x.some(student, \lambda y.relative\text{-}of(x, y)), \lambda x.see(x, z)))$

c. $every(play, \lambda z.some(student, \lambda y.some(\lambda x.relative\text{-}of(x, y), \lambda x.see(x, z))))$

d. $some(student, \lambda y.every(play, \lambda z.some(\lambda x.relative\text{-}of(x, y), \lambda x.see(x, z))))$

e. $some(student, \lambda y.some(\lambda x.relative\text{-}of(x, y), \lambda x.every(play, \lambda z.see(x, z))))$

There is an important difference between the application of the idea here and in (Chaves, 2003). There, detecting logical equivalences is done only by enumerating the

UR into all its possible solutions. This destroys the whole point of having a UR in the first place. This problem was solved in (Koller and Thater, 2006), where the filtering is done on the UR itself without enumerating all its solutions (the filtering reduces the number of dominance relations in the UR). However, that work still first computes the entire UR with all the redundancies before proceeding to eliminate them. Here, in contrast, we filter the redundant possibilities immediately when they are detected during the packed derivation, and they are not put in the packed semantic representation at all.

6.6.2 Applying Filtering Heuristics and Statistics

Moran and Pereira (1992) suggest several heuristics that allow to prefer one scoping over another. Higgins and Sadock (2003) and Andrew and MacCartney (2004) induce such rules based on a statistical analysis of a corpus of sentences annotated with preferred scopings. In keeping with the tradition of ambiguity management in the XLE, we can use such heuristics and statistics to filter out highly unlikely options while still keeping the top N-best options. This can be done by declaring some choices in the choice-space as a *nogood*. In the *mod_str* strategy, this is done by adding constraints to the representation. We can further use the heuristics and statistics when we extract solutions from the packed or underspecified representations, so that the most likely reading is extracted first, then the next best, and so on.

6.7 Additional Features

6.7.1 Non-Scoping Modifiers

In section 4.4.6, we saw that for certain modifiers (such as verb modifiers), we want to declare that the order in which they apply should not matter, by writing the flag [*noscp*] in the glue specification. Here we extend the algorithm to take this into account. First, when the algorithm compiles the input premises, if a premise with main index i and the flag [*noscp*] is encountered, the index i is added to the list of non-scoping modifiers.

When calculating histories inside a basic SCC, we first use all and only the modifiers from $A \rightarrow A$ whose main index is in the list of non-scoping modifiers.

A basic procedure is to run the basic chart algorithm on the initial histories and non-scoping modifiers (ignoring the scoping modifiers for now), but with a twist. In the original

algorithm, if a history h is created and it has the same span as another history h' existing in the chart, the parent-link of h is added to the parent-link list of h', and h is dropped (thus, h and h' are compressed to a new version of h'). Here, since the order of non-scoping modifiers is not important, h is simply dropped (not put on the agenda) without modifying the parent-link list of h'.

After we finish calculating all non-equivalent possibilities in this way, we delete the non-scoping modifiers from the chart so they could not be used anymore, but we keep the rest of the chart content (all histories there have category A). This content provides a new set of initial histories in the SCC. We proceed with this set and the set of scoping modifiers from the $A \to A$ node.

Because the modifiers are non-scoping, ideally we do not want to calculate all possibilities but rather find *some* order of the non-scoping modifiers to combine with each initial history. However, some modifiers may not be compatible with others (if this happens, it must be because one of the modifiers eventually turns out to be incapable of participating in a complete derivation – see for example the $A \to A$ modifier $\{3, 5\}$ in the figure in (250)).

We can improve on the basic procedure by computing the set of pairs of modifiers such that the two modifiers in the pair cannot be combined with each other because their spans overlap. For k modifiers, there are $\frac{1}{2}k(k-1)$ comparisons. Then, we can deal with the two most common cases:

- Collect all the non-scoping modifiers that are compatible with all non-scoping as well as scoping modifiers, and simply combine them in some arbitrary order with each initial history in the SCC. This gives us a new set S of histories.

- For each non-scoping modifier m_1 that is compatible with all scoping modifiers and incompatible with exactly one other non-scoping modifier m_2, we can duplicate S, add m_1 to all histories in one copy to get S_1, add m_2 to all histories in the second copy to get S_2, and then remove m_1 and m_2 from consideration. We set $S := S_1 \cup S_2$ and repeat.

There may be more complex cases, but they are very rare. The general computational question is this: Given a set S of elements, and a set T of pairs of elements from T, find the maximal subsets of S such that for each pair $\langle e_1, e_2 \rangle \in T$, e_1 and e_2 are not both in S. The set S represents the modifiers, and the set T represents the incompatibility constraints between them. In the worst case, there can be exponentially many such maximal subsets.

However, cases that go beyond the two discussed above are rare, and for those few cases we can just use the basic procedure above since not much would be gained in practice from a more optimized solution.

6.7.2 Scoping Constraints

In section 4.6.2, we allowed the user to specify scoping constraints between modifiers. Extending the algorithm to implement them is straightforward. After we compile the premises, we translate each `outscopes`(A,B) fact to the set of all facts $j > i$, where j is an index of a premise with a category that includes A as a subcategory, and i is an index of a premise with a category that includes B as a subcategory.

Then, whenever a modifier with main index i is considered for combination with a history h inside a basic SCC, allow the combination only if there is no scoping constraint $j > i$ where j is a member of the index set of h. Or, if we use the *mod_str* option, then a scoping constraint $j > i$ between the main indices of two modifiers would cause us to add a constraint $k_j > k_i$ to the list of constraints in the *mod_str*, where k_j is the position of the j modifier in the list of modifiers in the *mod_str*, and k_i is the position of the i modifier.

6.8 Complexity

The complexity of the algorithm here is not worse than the basic chart algorithm of the previous chapter, and it is usually better. If there are no cycles in the category graph, and each atomic category appears exactly once positively in the input, then there is exactly one possible proof, and the running time is $O(n^2)$: There are $O(n)$ nodes in the category graph, each will have exactly one history in it, and each connector node will have a cost of $O(n)$ for calculating set intersection between the spans of the two parent histories.

A more general input (but still ignoring cycles for the moment) can have more than one history in some nodes, leading to more set intersection computations. The worst case may be exponential. Such situations can happen only when there is a "confusion" in the input. This happens when there are two different ways of creating a certain category, in a similar way to the graphs (237), (248), and similar examples.

However, in practice, in an input that comes from a valid glue semantics specification applied on a valid F-structure, these confusions can arise only from cases such as in (238) and (250), where a *local* confusion exists because a category has two instances, one directly

in the input, and one that is compiled out of another statement. The confusion is local because at a certain point in the derivation, this confusion is found out, and only one result survives (unlike the situation in (237) etc. where there are genuinely two separate solutions). Therefore, the proliferation of options is restricted to a small part of the overall derivation. Furthermore, in this work I have only found two cases in the glue specification that create such a local confusion, namely equi verbs as in (238) (but that confusion can be circumvented for them as explained in section 5.4.1), and reciprocals as in (250) (see chapter 13). Hence, in practice, the number of histories in the category graph (ignoring cycles) for a glue semantics input arising from an F-structure will be almost linear in the size of the graph. This is similar to the claim about the syntax of natural language made in (Maxwell and Kaplan, 1996): Although a unification-based grammar for NL is not purely context-free and has some long-distance dependencies, it is almost context-free, and this property can be exploited to get a running time that is usually polynomial in practice.

As for cycles, a glue semantics input arising from an F-structure should have only basic cycles, corresponding to modifiers such as quantifiers and adjectives. Thus, although we get an explosion of combinations within such cycles, we can apply the optimizations shown in this chapter, which prevent this explosion from spreading to the rest of the derivation. The explosion is also usually small in practice, and can be further reduced by interleaving various techniques as explained above, and so it does not usually cause a problem.

Chapter 7

Packed Glue Derivation II: External Ambiguity

In the previous chapter we assumed that we are given a simple set of glue premises, and they may combine in more than one way to form a result. In this chapter I explain how to handle a set of glue premises which are themselves relativized to different choices in the choice space. The goal is to automatically produce packed semantic representations as in Figure 1.4, without the user having to worry about any of this when writing the glue specification. Furthermore, the computation of these packed representations should itself be packed. In other words, we do *not* want to simply unpack the packed input, run the glue prover on each choice, and then pack the results again. That would defeat the purpose of having packed representations.

7.1 Packed Input and Output

Consider again the sentence:

(290) Bill saw the girl with the telescope.

Because of the syntactic ambiguity, this sentence gets the packed F-structure shown in (291). Under choice $A1$, "with the telescope" modifies the verb, while under choice $A2$, it modifies "girl".

(291)
$$\boxed{0}\begin{bmatrix}
\text{PRED} & \text{SEE}\langle\boxed{1},\boxed{2}\rangle \\
\text{SUBJ} & \boxed{1}\begin{bmatrix}\text{PRED} & \text{BILL}\\ \text{NTYPE} & \text{NAME}\end{bmatrix} \\
\text{OBJ} & \boxed{2}\begin{bmatrix}\text{SPEC} & \text{THE}\\ \text{PRED} & \text{GIRL}\\ \text{ADJUNCT} & \left\{\langle \text{A2:}\ \boxed{4}\rangle\right\}\end{bmatrix} \\
\text{ADJUNCT} & \left\{\langle \text{A1:}\ \boxed{4}\begin{bmatrix}\text{PRED} & \text{WITH}\langle\boxed{3}\rangle\\ \text{OBJ} & \boxed{3}\begin{bmatrix}\text{SPEC} & \text{THE}\\ \text{PRED} & \text{TELESCOPE}\end{bmatrix}\end{bmatrix}\rangle\right\}
\end{bmatrix}$$

Here are the premises as they are obtained by rewriting this F-structure using the glue specification from chapter 4:

(292)

context	premises
1	$bill : \mathbf{1}^e$
	$the : (\mathbf{2}_v^e \to \mathbf{2}_r^t) \to \mathbf{2}^e$
	$girl : \mathbf{2}_v^e \to \mathbf{2}_r^t$
	$the : (\mathbf{3}_v^e \to \mathbf{3}_r^t) \to \mathbf{3}^e$
	$telescope : \mathbf{3}_v^e \to \mathbf{3}_r^t$
	$see : \mathbf{0}_v^e \to \mathbf{0}_r^t$
	$\lambda x\lambda P\lambda e.P(e) \wedge subj(e,x) : \mathbf{1}^e \to (\mathbf{0}_v^e \to \mathbf{0}_r^t) \to \mathbf{0}_v^e \to \mathbf{0}_r^t$ $[noscp]$
	$\lambda x\lambda P\lambda e.P(e) \wedge obj(e,x) : \mathbf{2}^e \to (\mathbf{0}_v^e \to \mathbf{0}_r^t) \to \mathbf{0}_v^e \to \mathbf{0}_r^t$ $[noscp]$
	$\lambda P.exists(P) : (\mathbf{0}_v^e \to \mathbf{0}_r^t) \to \mathbf{0}^t$
	$\lambda y\lambda x.with(x,y) : \mathbf{3}^e \to \mathbf{4}_v^e \to \mathbf{4}_r^t$
$A1$	$\lambda Q\lambda P\lambda x.P(x) \wedge Q(x) : (\mathbf{4}_v^e \to \mathbf{4}_r^t) \to (\mathbf{0}_v^e \to \mathbf{0}_r^t) \to \mathbf{0}_v^e \to \mathbf{0}_r^t$ $[noscp]$
$A2$	$\lambda Q\lambda P\lambda x.P(x) \wedge Q(x) : (\mathbf{4}_v^e \to \mathbf{4}_r^t) \to (\mathbf{2}_v^e \to \mathbf{2}_r^t) \to \mathbf{2}_v^e \to \mathbf{2}_r^t$

All the premises except the last two are in the top context. The premise $\lambda y\lambda x.with(x,y) : \mathbf{3}^e \to (\mathbf{4}_v^e \to \mathbf{4}_r^t)$ was created separately under each of the two contexts $A1$ and $A2$ by the rules for prepositional phrases in sections 4.4.3 and 4.6.1. The packed rewriting system automatically detected the two identical copies of this premise and moved them to the disjunction of $A1$ and $A2$, namely to the top context.

Here is the packed representation we want:

(293) $l_1 : exists(\lambda v_1.\langle A1 : l_2 \wedge l_4,\ A2 : l_2\rangle)$

$l_2 : see(v_1) \wedge subj(v_1, bill) \wedge obj(v_1, the(\lambda v_2.\langle A1 : l_3,\ A2 : l_3 \wedge l_4\rangle)$

$l_3 : girl(v_2)$

$l_4 : with(\langle A1 : v_1,\ A2 : v_2\rangle,\ the(telescope))$

7.2 First Attempt

7.2.1 Direct Extension

For the moment, let us ignore modifier ambiguities (there are none in this example because of the [*noscp*] marks), and let us see whether we can take Hepple's basic chart algorithm from chapter 5 and extend it to handle such packed input.

A first attempt to do this is to simply add a choice variable to each element, and to modify the rule (212) as follows:

$$(294)\qquad \frac{C_1 : \psi : A : S_1 \qquad C_2 : \delta : A_L \rightarrow B : S_2}{C_1 \wedge C_2 : \delta(\lambda v_{i_1}, \ldots, \lambda v_{i_n}.\psi) : B : S_1 \cup S_2}\ \mathrm{E} \qquad \begin{array}{l}\text{provided } S_1 \cap S_2 = \emptyset \text{ and } L \subseteq S_1\\ \text{and } L = [i_1, \ldots, i_n]\\ \text{and } C_1 \wedge C_2 \neq 0\end{array}$$

This rule is the same as (212) except that now we allow a combination of elements only if their choices, C_1 and C_2, are mutually consistent. If so, the combined element resides in the intersection of these contexts, $C_1 \wedge C_2$ (not in each separately). For most combinations, $C_1 = C_2 = 1$ and $C_1 \wedge C_2 = 1$, but for some, there will be a more restricted context.

Furthermore, we should get at some point in the derivation an element E_1 in context $A1$ which uses all the premises except for those compiled from

$$(295)\quad \lambda P.exists(P) : (\mathbf{0}_v^e \rightarrow \mathbf{0}_r^t) \rightarrow \mathbf{0}^t$$

and from the original premise in $A2$. We should also get another element E_2 which is in context $A2$ and which uses all the premises except for those compiled from (295) and from the original premise in $A1$. Because these two elements are in competing contexts, but they use the same premises (up to the competing ones from $A1$ and $A2$), the two should be compressed into one element in context $A1 \vee A2 = 1$, which uses the union of the premises of the two elements.

7.2.2 Problems

The Order of Exploring Combinations

There are three problems here. One problem is that we have no control over the order in which elements are combined in the chart algorithm. Suppose the sentence were "John thinks that Bill saw the girl with the telescope." The chart algorithm may combine the premises from $A1$ with premises in the top context to obtain new elements in $A1$, until it

finds an element E_1 that covers the inner sentence. The algorithm may keep going within $A1$ until it finds an element E_3 that covers the entire sentence. It is possible that only after doing so, it may discover that there is in fact another way of getting an element E_2 that covers the inner sentence under choice $A2$.

At that point, E_2 should be compressed with the parallel $A1$ element, E_1. But what about all the work that has already been done in $A1$ after E_1 to reach E_3? We do not want the computer to repeat this work – this is precisely what a packed computation should not do. But if this work is not repeated, then all the elements that were created between E_1 and E_3 now need to be updated because we discovered that they are not only under $A1$ but under $A2$ as well, i.e. they are under the top context. This updating is complicated to implement, and it is inefficient because previous work may need to be revisited over and over again.

What we want is for the algorithm to first explore all possible ways of obtaining the category that stands for the inner sentence, and only then to move forward with the rest of the derivation. We had a similar issue in the previous chapter, and we can use the same solution: the category graph. Using this graph, we are guaranteed that both elements that cover the inner sentence will be produced at the same time, in the node that corresponds to that inner sentence. At that point, we can compress them into a new element at the top context, and the rest of the derivation of "John thinks that" will be done just once from then on.

Assigning Inidces

There is still another problem. How should we assign indices to the premises in (291) after they are compiled? When we have only premises in the top context, we simply assign a unique index to each premise. But it may seem that we need to assign *the same* index to the two premises under $A1$ and $A2$ because they are mutually inconsistent. It seems that a span of a complete derivation would have to include exactly one or the other.

However, it is not clear how we can know to assign indices in such a way. We may try to take inspiration from the way that the parser knows that two elements are inconsistent – they come from overlapping places in the sentence. Perhaps we should assign to each premise an index that is based on the place in the sentence from which the premise came?

It is not immediately clear whether we can track each premise to a particular place in the sentence because rules in the glue specifications look at pieces of F-structure rather than

directly at words. Worse, there may be two premises in context $A1$ and three in context $A2$. The two premises in $A1$ are mutually consistent, but they are each inconsistent with each of the premises under $A2$. This gets even more complex if $A1$ or $A2$ has subcontexts.

Assigning Contexts

Another problem is that the assignment of contexts by rule (294) is not correct. If we combine "with the telescope" with "girl" under $A2$, we get an element in $A2$ that covers "girl with the telescope". This then combines with

(296) $the : (\mathbf{3}_v^e \rightarrow \mathbf{3}_r^t) \rightarrow \mathbf{3}^e$

to form an element in $A2$ that covers "the girl with the telescope". In addition, the derivation will also combine "girl" with (296) to form an element which covers "the girl", of category $\mathbf{3}^e$. The premises that lead to this element are all in context 1, and according to the rule (294), this element will also be in context 1. However, this element should really be in context $A1$ because only in that context, and not in $A2$, should (296) combine with "girl" directly, without first "girl" combining with "with the telescope". It is as if we need to notice that if "with the telescope" under $A2$ combines with "girl", then we need to create another element of "girl" under $A1$ that does not combine with "with the telescope", and delete the "girl" element under context 1. It is complicated to define such a computation directly in the chart algorithm.

7.3 Packing the Derivations

7.3.1 Investigating Unpacked Derivations

We are getting stuck by these problems here because we are tying to directly impose the choice space on the combination rule. Instead, the general correct approach for designing a packed representation or algorithm is to first investigate how they work in an unpacked way in each of the choices of the choice space, and only then to notice the commonalities and factor them out.

To demonstrate this, let us first consider a slightly easier example.

(297) $[[\text{John}]_1$ thinks that $[[\text{time}]$ flies $[\text{like } [\text{an arrow}]_3]]_2]_0$
$[[\text{John}]_1$ thinks that $[[\text{time flies}]$ like $[\text{an arrow}]_3]_2]_0$

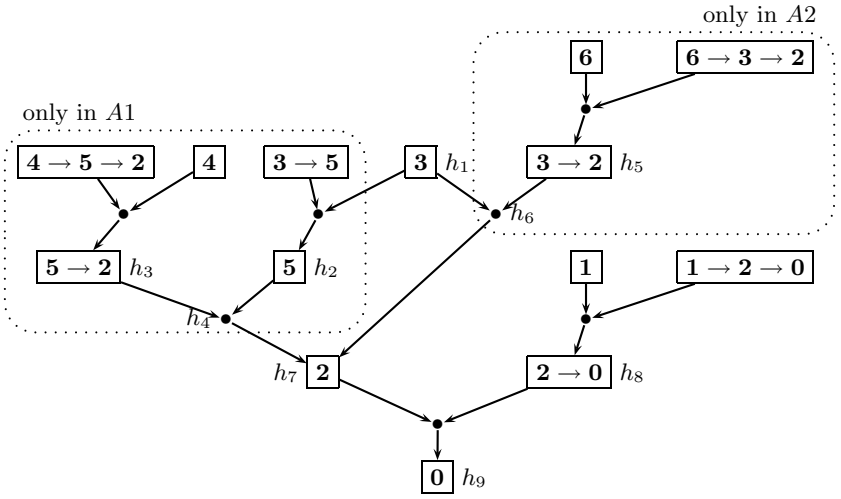


Figure 7.1: A Packed Category Graph

Suppose that the first analysis is obtained under choice $A1$, and the second under $A2$. For the purpose of demonstrating the point in this discussion, let us assume a simplified set of premises:

choice	premises
1	$john : \mathbf{1} : \{1\}$
	$think : \mathbf{1} \rightarrow \mathbf{2} \rightarrow \mathbf{0} : \{2\}$
	$anarrow : \mathbf{3} : \{3\}$
$A1$	$time : \mathbf{4} : \{4\}$
	$fly : \mathbf{4} \rightarrow \mathbf{5} \rightarrow \mathbf{2} : \{5\}$
	$like : \mathbf{3} \rightarrow \mathbf{5} : \{6\}$
$A2$	$timeflies : \mathbf{6} : \{7\}$
	$like : \mathbf{6} \rightarrow \mathbf{3} \rightarrow \mathbf{2} : \{8\}$

For now, we assigned a unique index to each premise. Let us see what would happen under each of the two choices in the choice-space, if we were to work with unpacked input.

Under $A1$, we would get the category graph in Figure 7.1 minus the nodes $\mathbf{6}$, $\mathbf{6} \rightarrow \mathbf{3} \rightarrow \mathbf{2}$, and $\mathbf{3} \rightarrow \mathbf{2}$. Under $A2$, the graph would be as in the figure, minus the nodes $\mathbf{4}$, $\mathbf{4} \rightarrow \mathbf{5} \rightarrow \mathbf{2}$, $\mathbf{5} \rightarrow \mathbf{2}$, and $\mathbf{5}$. The spans in the histories shown in the graph would be those in the columns "under $A1$" and "under $A2$" below:

history	under $A1$	under $A2$	packed
h_1	$\{3\}$	$\{3\}$	$\langle 1 : \{3\}\rangle$
h_2	$\{3,6\}$		$\langle A1 : \{3,6\}\rangle$
h_3	$\{4,5\}$		$\langle A1 : \{4,5\}\rangle$
h_4	$\{3,4,5,6\}$		$\langle A1 : \{3,4,5,6\}\rangle$
h_5		$\{7,8\}$	$\langle A2 : \{7,8\}\rangle$
h_6		$\{3,7,8\}$	$\langle A2 : \{3,7,8\}\rangle$
h_7	$\{3,4,5,6\}$	$\{3,7,8\}$	$\langle 1 : \{3\},\ A1 : \{4,5,6\},\ A2 : \{7,8\}\rangle$
h_8	$\{1,2\}$	$\{1,2\}$	$\langle 1 : \{1,2\}\rangle$
h_9	$\{1,2,3,4,5,6\}$	$\{1,2,3,7,8\}$	$\langle 1 : \{1,2,3\},\ A1 : \{4,5,6\},\ A2 : \{7,8\}\rangle$

7.3.2 Packed Histories and Spans

The last column shows how the two columns can be packed. In histories h_1 and h_8, because the facts $\{3\}$ and $\{1,2\}$ appear under each of the contexts $A1$ and $A2$, they can be put under the top context 1. In histories h_2 through h_6, only one of the contexts has a non-empty span. The interesting lines are h_7 and h_9, where the indices that appear in the spans under both choices $A1$ and $A2$, namely 1, 2, and 3, are shifted to the top context.

In order to obtain such histories, we need to expand the notion of a history in two ways. First, each history has a context from the choice-space. The history exists only in that context (and its sub-contexts) but not in any other context that is inconsistent with it. Thus, histories h_2 to h_4 exist only in context $A1$, while histories h_5 and h_6 only in context $A2$. Second, instead of a span, a history now has a *packed span*, as was shown above. The context of a packed span must be equal or more general than each of the sub-contexts mentioned in the span, and furthermore it must be the most specific context with that property. Hence, the context of a span is equal to the disjunction of all the sub-contexts mentioned in the span. The context of a history should be equal to the context of the history's packed span.

A compiled premise with category A under context C which has the unique index i gives rise to the history $\langle C, A, \langle C : \{i\}\rangle, \{null\}\rangle$. The crucial node in Figure 7.1 is **2**. The histories

(298) $h_4 = \langle A1, \mathbf{2}, \langle A1 : \{3,4,5,6\}\rangle, \{\langle h_3, h_2\rangle\}\rangle$ and
$h_6 = \langle A2, \mathbf{2}, \langle A2 : \{3,7,8\}\rangle, \{\langle h_5, h_1\rangle\}\rangle$

feed into this node so we want to compress them. In section 6.3.2 we said that two histories

may be compressed if they share exactly the same category and exactly the same span. Here, however, we need not require that the spans be identical, only that they be *consistent* in some sense that we will explore below. If they are consistent, then we compute their packed union, where each index is placed in the disjunction of the contexts in which it appears in the packed spans. Thus, 4 would be placed only under $A1$ whereas 3 would be placed under context $A1 \vee A2 = 1$. Furthermore, when compressing two histories, the context of the resulting history is the disjunction of the histories' contexts. Thus, we get the following compressed history:

(299) $h_7 = \langle 1, \mathbf{2}, \langle 1 : \{3\}, A1 : \{4,5,6\}, A2 : \{7,8\}\rangle, \langle A1 : \{\langle h_4\rangle\}, A2 : \{\langle h_6\rangle\}\rangle\rangle$.

Here, the set of parent-links is also relativized to contexts, and a parent-link $\langle h\rangle$ with just one item simply says that the current history was obtained from history h.

We achieved out goal: Thanks to the category graph, we found out all the ways of computing the category **2** of the inner sentence "time flies like an arrow" before moving to the derivation of "John thinks that". The computation was done just once for the shared piece "an arrow" (if **3** was obtained from a sub-derivation in context 1, it would be done just once rather than once in $A1$ and once in $A2$) and it was done just once for the shared piece "John thinks that", thanks to the packing in h_7.

There are a few details here that need to be fleshed out. We do this now by reviewing the algorithm from the previous chapter and extending it to its packed version. We first need a few definitions.

7.3.3 Definitions

An ordinary span in the previous chapter was simply a set of indices. A *packed span* is a set of pairs $D : i$, where D is a context in the choice-space, and i is an index. Thus, a packed span is a set of indices relativized to contexts in the choice-space. For example, in (300)a, the indices 4 and 5 appear under the top context, 6 and 7 under the context $A1$, and 8 under $A2$. For convenience, we write a packed span organized by the contexts rather than as a set of pairs. Thus, we write (300)b instead of (300)a.

(300) a. $\{1 : 4,\ 1 : 5,\ A1 : 6,\ A1 : 7,\ A2 : 8\}$

b. $\langle 1 : \{4,5\},\ A1 : \{6,7\},\ A2 : \{8\}\rangle$

We say that $i \in PS$ iff there is a context $D \neq 0$ such that $D : i \in PS$. If $i \notin PS$ then we say that $0 : i \in PS$.

The context of a packed span PS is the disjunction of all the contexts mentioned in the span:

$$cxt(PS) := \bigvee_{D:i \in PS} D$$

If C is a choice in the choice space then the *restriction* of a packed span PS to C is written $PS|_C$ and consists of restricting all the pairs in PS to C. Formally:

$$PS|_C = \{(D \wedge C) : i \mid D : i \in PS \text{ and } D \wedge C \neq 0\}$$

Thus, the restriction of (300) to $A2$ gives us $\langle A2 : \{4, 5, 8\}\rangle$. If C is a maximally-specific choice in the choice space then restricting PS to C will necessarily give a packed span of the form $\langle C : S\rangle$, where S is a (simple) set of indices. If we restrict a packed span PS to a context that is inconsistent with $cxt(PS)$, we get the empty span $\langle\rangle$.

What does a packed span PS mean? We need to think about what it says for each maximally-specific choice C of the choice space. For each such C which is consistent with $cxt(PS)$ we get that PS specifies a normal span, $PS|_C$, under C. For each C that is inconsistent with $cxt(PS)$, PS claims nothing (since $PS|_C$ is empty).

The *packed union* (or *factored union*) of two packed spans PS_1 and PS_2 is obtained by factoring out common parts in the union of the two spans. It is defined:

$$union(PS_1, PS_2) = \{(D_1 \vee D_2) : i \mid D_1 : i \in PS_1 \text{ and } D_2 : i \in PS_2\}$$

(where $i \notin PS$ is considered as $0 : i \in PS$). For example:

(301) $union(\langle 1 : \{4\},\ A1 : \{5, 6\}\rangle,\ \langle 1 : \{7\},\ A2 : \{5, 8\}\rangle) = \langle 1 : \{4, 5, 7\},\ A1 : \{6\},\ A2 : \{8\}\rangle$

It is easy to see that *union* (like set union and boolean disjunction) is idempotent, commutative, and associative. So instead of $union(\ldots union(PS_1, PS_2)\ldots, PS_n)$, we can write $union\{PS_1, \ldots, PS_n\}$. Here are three useful lemmas about spans and unions:

Lemma 3 $PS = union\{PS|_C \mid C \text{ maximally-specific}\}$

Proof: Suppose $D : i \in PS$. Let $D = C_1 \vee \ldots \vee C_n$ be the breakup of D into the maximally-specific contexts it includes. Then by definition of a restricted span we have that for all $1 \leq j \leq n$, $C_j : i \in PS|_{C_j}$, and also $i \notin PS|_C$ for any other maximally-specific C. By

definition of *union* we have that $(C_1 \vee \ldots \vee C_n) : i \in union\{PS|_C \mid C \text{ maximally-specific}\}$. But that means $D : i$ is included there.

Conversely, suppose $D : i \in union\{PS|_C \mid C \text{ maximally-specific}\}$. Then by definition of *union*, $D = C_1 \vee \ldots \vee C_n$ where for all $1 \leq j \leq n$, C_j is maximally-specific and $C_j : i \in PS|_{C_j}$. So it must mean that for some B, $B : i \in PS$, and for all j, $C_j \in B$. Hence $D \subseteq B$. But also $D = B$ because if maximally-specific $C \in B$ then $i \in PS|_C$ and so $C = C_j$ for some j (we put in D *all* the maximally specific C such that $i \in PS|_C$). Hence $D : i \in PS$. ■

Lemma 4 $union(PS_1, PS_2)|_C = union(PS_1|_C, PS_2|_C)$

Proof: $union(PS_1, PS_2)|_C =$
$\{((D_1 \vee D_2) \wedge C) : i \mid D_1 : i \in PS_1 \text{ and } D_2 : i \in PS_2\} =$
$\{((D_1 \wedge C) \vee (D_2 \wedge C)) : i \mid D_1 : i \in PS_1 \text{ and } D_2 : i \in PS_2\} =$
$union(PS_1|_C, PS_2|_C)$ ■

Lemma 5 $(PS|_{C_1})|_{C_2} = PS|_{C_1 \wedge C_2}$

Proof: Easily follows from the definitions. ■

7.4 Packed Algorithm

7.4.1 Compilation

The only change needed in the compilation procedure is to add the context of a premise to the initial history created from the premise. If a premise has context C then so do all the new premises compiled out of it.

7.4.2 Complete Packed Histories

When the packed algorithm terminates, a history $\langle C, G, PS, pls \rangle$ in the goal node (with category G) is *complete* iff PS includes all the indices that are consistent with context C. (An index i is consistent with context C iff the context D of the premise indexed with i is consistent with C.)

7.4.3 Combination

We need to revise the definition of history combination. We are given two histories $h_1 = \langle C_1, A, PS_1, pls_1 \rangle$ and $h_2 = \langle C_2, B, PS_2, pls_2 \rangle$, where C_1 and C_2 are their contexts, PS_1 and PS_2 are packed spans, and pls_1 and pls_2 are sets of parent-links. For the histories to combine, we must have $A = B_L \rightarrow F$ for some category F and (possibly empty) index list L. Let L' be the union of all index-requirements on categories in F (for example, $L' = [1,2,3]$ if $F = X_{[1,2]} \rightarrow Y_{[3]} \rightarrow Z$). The unpacked definition required the two spans of h_1 and h_2 to be disjoint. In the packed version, however, we need to relax and generalize this definition.

Inefficient Combination

When can h_1 and h_2 combine, and what is the result of the combination? I first show a definition that guarantees correctness but is inefficient. I later show a more efficient definition and prove that it is equivalent to the first definition.

The basic definition is that we need to consider each maximally-specific context C in the choice space, and then consider what happens in it. First, if C is inconsistent with either C_1 or C_2 then h_1 and h_2 cannot be combined under C. So the first requirement for combination is that C is consistent with both C_1 and C_2. We further need the usual conditions from section 6.3.2 to hold under C:

1. $PS_1|_C \cap PS_2|_C = \emptyset$
2. $L \subseteq PS_2|_C$
3. $L' \cap PS_2|_C = \emptyset$

If all these conditions are met then we say that h_1 and h_2 are combinable under C. The result of that combination is the history $\langle C, F, PS_1|_C \cup PS_2|_C, \langle C : \{\langle h_1, h_2 \rangle\}\rangle\rangle$.

Theoretically, we could go over all the maximally-specific contexts C in the choice-space and calculate the combination for each one. We can then pack together all the results. The packed result exists under the context *comb-cxt*(h_1, h_2) defined as:

$$\textit{comb-cxt}(h_1, h_2) = \bigvee \{C \mid C \text{ is maximally-specific, and } h_1, h_2 \text{ are combinable under } C\}$$

and its packed span is the packed union of all the spans calculated for the different contexts under *comb-cxt*(h_1, h_2). We could write this as:

$$comb(PS_1, PS_2) = union\{comb(PS_1|_C, PS_2|_C) \mid C \text{ maximally-specific}\}$$

(if h_1 and h_2 are not combinable under C then $comb(PS_1|_C, PS_2|_C)$ is the empty span.)

For example, the two packed spans in (302)a are both under the top context. Suppose the choice-space is defined simply by $1 : \textit{one-of}(A1, A2)$.[1] Then under the maximally-specific context $A1$, we get the spans and the result in (302)b, and under $A2$, we get those in (302)c. If we pack those results using *union* then we get (302)d.

(302) a. $\langle 1 : \{3\},\ A1 : \{4,5\}\rangle$, $\langle 1 : \{6\},\ A2 : \{4,7\}\rangle$

b. $A1 : \quad \{3,4,5\}$, $\{6\} \ \Rightarrow\ \{3,4,5,6\}$

c. $A2 : \quad \{3\}$, $\{4,6,7\} \ \Rightarrow\ \{3,4,6,7\}$

d. $\langle 1 : \{3,4,6\},\ A1 : \{5\},\ A2 : \{7\}\rangle$

Now consider what happens if we have overlapping contexts. Suppose the choice space is defined by $1 : \textit{one-of}(A1, A2)$ and $1 : \textit{one-of}(B1, B2)$. Then given the two packed spans under (303)a, we consider the four maximally-specific contexts and get the results under (303)b-e. Only under the conjunction of $A1$ and $B1$ do we get a result in (303)b, while in the other contexts, the two histories cannot be combined. The final result is therefore (303)f.

(303) a. $\langle A1 : \{3\}\rangle$, $\langle B1 : \{4\}\rangle$

b. $A1 \wedge B1 : \quad \{3\}$, $\{4\} \ \Rightarrow\ \{3,4\}$

c. $A1 \wedge B2 : \quad \{3\}$, $\{\} \ \Rightarrow\ \{\}$

d. $A2 \wedge B1 : \quad \{\}$, $\{4\} \ \Rightarrow\ \{\}$

e. $A2 \wedge B2 : \quad \{\}$, $\{\} \ \Rightarrow\ \{\}$

f. $\langle A1 \wedge B1 : \{3,4\}\rangle$

Finally, suppose the choice-space is defined by $1 : \textit{one-of}(A1, A2)$ and consider what happens when we get the two histories in (304)a. Although the two spans can be combined under both contexts, as shown in (304)b,c, we also have the requirement in h_1 that the span of h_2 should include 5. This happens only under $A1$, and so the combination in (304)c fails, and the final history in (304)d is only under $A1$.

[1] See section 4.2.1 for the definition of *one-of*.

(304) a. $h_1 = \langle 1,\ X_{[5]} \rightarrow Y,\ \langle 1 : \{3\}\rangle\rangle\ ,\ h_2 = \langle 1,\ X,\ \langle 1 : \{4\},\ A1 : \{5\}\rangle\rangle$

b. $A1 :\ \{3\}\ ,\ \{4,5\}\ \Rightarrow\ \{3,4,5\}$

c. $A2 :\ \{3\}\ ,\ \{4\}\ \Rightarrow\ \{3,4\}$ but need 5 so fail

d. $\langle A1,\ Y,\ \langle A1 : \{3,4,5\}\rangle,\ \{\langle h_1, h_2\rangle\}\rangle$

Efficient Combination

Computing results in the way shown above defeats our purpose because it is an *unpacked combination*. Essentially, we unpack the packed histories, calculate the results separately for each choice in the choice space, and then re-pack the results. I now show how to define the efficient *packed combination*. First, we need a few definitions.

For each index i, let D^i_j be the context under which i appears in PS_j $(j = 1, 2)$. If $i \notin PS_j$ we say that $D^i_j = 0$.

The Intersection-Context (TC) of PS_1 and PS_2 is defined:

$$TC(PS_1, PS_2) = \bigwedge_{i \in PS_1 \text{ or } i \in PS_2} \neg(D^i_1 \wedge D^i_2)$$

The Inclusion-Context (IC) of PS_2 w.r.t. L is defined as 1 if $L = [\,]$, and otherwise:

$$IC(PS_2, L) = \bigwedge_{i \in L} D^i_2$$

The Exclusion-Context (EC) of PS_2 w.r.t. L' is defined as 1 if $L' = [\,]$, and otherwise as:

$$EC(PS_2, L') = \bigwedge_{i \in L'} \neg D^i_2$$

If we get two histories $h_1 = \langle C_1, A, PS_1, pls_1\rangle$ and $h_2 = \langle C_2, B, PS_2, pls_2\rangle$, such that $A = B_L \rightarrow F$ for some (possibly empty) index list L, and L' is the union of all index-requirements on categories in F, then the Combination-Context (CC) of h_1 and h_2 is defined:

$$CC(h_1, h_2) = C_1 \wedge C_2 \wedge TC(PS_1, PS_2) \wedge IC(PS_2, L) \wedge EC(PS_2, L')$$

Then, h_1 and h_2 are combinable iff $CC(h_1, h_2) \neq 0$. If that is the case then let $C = CC(h_1, h_2)$, and the result of the combination is the history:

$\langle C,\ F,\ union(PS_1, PS_2)|_C,\ \{C : \{\langle h_1, h_2\rangle\}\}\rangle$. [2]

I will prove below that this definition gives the same results as the definition above. Here is the intuition behind this definition. The simplest case is when an index i appears in just one of the spans, say PS_1. Then $D_2^i = 0$ and so the conjunct $\neg(D_1^i \wedge D_2^i) = \neg 0 = 1$ does not affect TC.

In the unpacked algorithm, if any index is present in both spans then the two histories cannot be combined. However, in the packed algorithm we may have a situation where the index appears in both spans but that does not completely prevent combination. If the two contexts under which the index appears are mutually-inconsistent, then the two occurrences of the index to not interfere with each other. An example was shown in (302). Even though the index 4 appears in both spans, its contexts there, $A1$ and $A2$, are mutually-inconsistent, and so it is possible to combine the two spans. According to the definition of TC, we will have $D_1^4 = A1$, $D_2^4 = A2$, and we will add the conjunct $\neg(D_1^4 \wedge D_2^4) = \neg(A1 \wedge A2) = \neg 0 = 1$ to $TC(PS_1, PS_2)$.

The other extreme is when an index i appears under the same context D in both spans. Then $D_1^i = D_2^i = D$, and so the combination of h_1 and h_2 can occur only under $\neg D$.

A case in between is when an index appears under two partially-overlapping contexts. For example, suppose the choice space is defined by $1 : \textit{one-of}(A1, A2)$ and $1 : \textit{one-of}(B1, B2)$. The two spans in (305)a yield the unpacked combinations under (305)b-e. The final result is therefore (305)f.

(305) a. $\langle 1 : \{3\}, A1 : \{4\}\rangle$, $\langle 1 : \{5\}, B1 : \{4\}\rangle$

b. $A1 \wedge B1$: $\{3,4\}$, $\{4,5\}$ $\Rightarrow$ fail

c. $A1 \wedge B2$: $\{3,4\}$, $\{5\}$ $\Rightarrow$ $\{3,4,5\}$

d. $A2 \wedge B1$: $\{3\}$, $\{4,5\}$ $\Rightarrow$ $\{3,4,5\}$

e. $A2 \wedge B2$: $\{3\}$, $\{5\}$ $\Rightarrow$ $\{3,5\}$

f. final context: $A2 \vee B2$. final span: $\langle A2 \vee B2 : \{3,5\},\ Z : \{4\}\rangle$

were $Z = (A1 \text{ xor } B1) = (A2 \text{ xor } B2) = (A1 \wedge B2) \vee (A2 \wedge B1)$

This result is also obtained by using the definition of efficient packed combination because $D_1^4 = A1$, $D_2^4 = B1$, and so TC will have the conjunct $\neg(A1 \wedge B1) = A2 \vee B2$.

[2] We mark C on the parent-link $\langle h_1, h_2\rangle$ in order to help later with reconstructing the meaning terms in section 7.4.6.

Proofs of Correctness

Proposition 5 *The two definitions of packed combination are equivalent.*

Proof: We need to show that the packed combination of two packed histories h_1, h_2 is equal to the packing of the unpacked combination of h_1, h_2. The proof has two parts. The first part proves that we get the same resulting contexts, i.e. *comb-cxt*$(h_1, h_2) = CC(h_1, h_2)$. The second part proves that the resulting packed spans are the same.

Part I: We start by ignoring the complication involved with the required sets of indices L and L', and later bringing this back in. If $L = [\ ]$ and $L' = [\ ]$ then $IC(PS_2, L) = EC(PS_2, L') = 1$ and so we want to show that *comb-cxt*$(h_1, h_2) = C_1 \wedge C_2 \wedge TC(h_1, h_2)$.

We first show that *comb-cxt*$(h_1, h_2) \subseteq C_1 \wedge C_2 \wedge TC(h_1, h_2)$ holds. We need to show that for any maximally-specific context C, if it is included on the left side then it is included on the right side. By definition, if C is included on the left side then it is consistent with both C_1 and C_2. It remains to show that for every index i that is in PS_1 or PS_2, C is consistent with $\neg(D_1^i \wedge D_2^i)$. Suppose to the contrary it is not. Since C is a maximally-specific context (and $C \neq 0$) then C must by consistent with $D_1^i \wedge D_2^i$, i.e. $C \wedge D_1^i \wedge D_2^i \neq 0$. So $D_1^i \neq 0$ and $D_2^i \neq 0$. But that means $i \in PS_1|_C \cap PS_2|_C \neq \emptyset$, which contradicts $C \in$ *comb-cxt*(h_1, h_2).

Now we show that $C_1 \wedge C_2 \wedge TC(PS_1, PS_2) \subseteq$ *comb-cxt*(h_1, h_2) holds. If C is a maximally-specific context that is included on the left then it is consistent with both C_1 and C_2. It remains to show that if it is included in $TC(PS_1, PS_2)$ then $PS_1|_C \cap PS_2|_C = \emptyset$. Suppose to the contrary this is false. Then there is an index $i \in PS_1|_C \cap PS_2|_C \neq \emptyset$. So $D_1^i \neq 0$ and $D_2^i \neq 0$, and C is consistent with both of them. Since C is a maximally-specific context, $D_1^i \wedge D_2^i \neq 0$, and $C \wedge \neg(D_1^i \wedge D_2^i) = 0$. But the last step means that C is not included in $TC(PS_1, PS_2)$, which is a contradiction.

Now we need to bring the index requirements back. If C is maximally-specific and $L \subseteq PS_2|_C$ then for all $i \in L$, $i \in PS_2|_C$, and so C is consistent with D_2^i for all such i. Therefore, C is included in $IC(PS_2, L)$. The reverse of this chain also holds. A similar argument applies for the condition $L' \cap PS_2|_C = \emptyset$ vis-a-vis $EC(PS_2, L')$.

Part II: Let $\mathbf{C} =$ *comp-cxt*$(h_1, h_2) = CC(h_1, h_2)$. We need to prove:

$$\mathit{union}\{PS_1|_C \cup PS_2|_C \ \mid \ C \in \mathbf{C}\} = \mathit{union}(PS_1, PS_2)|_{\mathbf{C}}$$

(remember that $PS_1|_C$ and $PS_2|_C$ are disjoint for all $C \in \mathbf{C}$). This can be shown by the

following sequence of steps, each of which is straightforward to prove from the definitions and from Lemmas 3 and 4:

$$
\begin{aligned}
&\mathit{union}\{PS_1|_C \cup PS_2|_C \mid C \in \mathbf{C}\} = \\
&\mathit{union}(\mathit{union}\{PS_1|_C \mid C \in \mathbf{C}\},\ \mathit{union}\{PS_1|_C \mid C \in \mathbf{C}\}) = \\
&\mathit{union}(PS_1|_{\mathbf{C}}, PS_2|_{\mathbf{C}}) = \\
&\mathit{union}(PS_1, PS_2)|_{\mathbf{C}}
\end{aligned}
$$

■

Lemma 6 *For every categories X and Y (in the category graph) that combine to give another category, and every packed histories h_1, h_2 in X, Y respectively, and every maximally specific context C, $\mathit{comb}(h_1|_C, h_2|_C) = \mathit{comb}(h_1, h_2)|_C$.*

Proof: By definition above, $\mathit{comb}(PS_1, PS_2) =$
$\mathit{union}\{\mathit{comb}(PS_1|_C, PS_2|_C) \mid C$ maximally-specific$\}$. If we restrict both sides to C then on the left we get $\mathit{comb}(PS_1, PS_2)|_C$, and it is easy to prove that on the right we get only $\mathit{comb}(PS_1|_C, PS_2|_C)$. ■

We need a few definitions. A *history forest* is a directed-acyclic graph whose nodes are histories and which has an edge from history h to history h' iff h' is a parent history of h. A *complete history forest*, which corresponds to one or more complete derivations in the category graph, has one goal history which serves as the parent of no other history, and one leaf history for each premise that was in the output of the compilation phase. In the packed algorithm (but not the unpacked algorithm), one or more premises with the same category could share the same history if they were compressed together into one (leaf) history.

We can restrict a history forest $\mathcal{F}$ to a context C by restricting each of the histories in $\mathcal{F}$ to C and removing any empty histories (with an empty packed span) and their associated links.

Theorem 3 *Packed combination leads to correct results.*

Proof: We will use the definition for inefficient combination, but the result holds for efficient combination as well by Proposition 5.

We need to show: If Γ is the packed input to the packed algorithm, then for every maximally specific context C we have the following: $\mathcal{D}$ is a complete derivation produced

by the unpacked algorithm from $\Gamma|_C$ iff $\mathcal{D}$ is a (complete derivation) subgraph of the restricted forest $\mathcal{F}|_C$ where $\mathcal{F}$ is obtained by the packed algorithm from Γ. The 'if' part shows soundness of the packed algorithm, i.e. if an unpacked derivation is included in the packed result then it is in fact a correct derivation, and the 'only if' part shows completeness, i.e. all derivations found by the unpacked algorithm are included in the result of the packed algorithm.

Let us examine what happens under C in each of the algorithms. The unpacked algorithm starts with $\Gamma|_C$ and creates leaf histories from it, while the packed algorithm starts with Γ and creates leaf packed histories. But if h is one of these packed histories then $h|_C$ is one of the unpacked histories under C (this follows from Lemma 3). Conversely, if h is an unpacked history then there will be a packed history h' such that $h'|_C = h$ (again by Lemma 3).

Now if we consider the combinations done by the algorithms, we first notice that by Lemma 3, if h is a history in the unpacked algorithm under C then $h = h'|_C$ for some packed history h'. Furthermore, we see that the theorem follows from Lemma 6 because at each step of the way, any combination done by the unpacked algorithm will be included in the restricted forest of the packed algorithm, and any combination included in the restricted forest of the packed algorithm will be done by the unpacked algorithm. ■

7.4.4 Compression

For a non-SCC category node, we need to revise the definition of history compression (we will deal with SCCs in section 7.4.5). Compression of two histories in the unpacked algorithm required the two spans to be identical. In the packed algorithm, however, we need to relax and generalize this definition.

Compression of Two Histories

We are given two histories $h_1 = \langle C_1, A, PS_1, pls_1 \rangle$ and $h_2 = \langle C_2, B, PS_2, pls_2 \rangle$. In order to compress them, we require that $A = B$ (as in section 6.3.2). Since both histories are in the same node, A will be equal to B if we ignore their index requirements, but we should not compress h_1 and h_2 if their index requirements are different (e.g. $C_{[1]} \rightarrow D \neq C_{[2]} \rightarrow D$) because they have different potentials of combination with other histories.

When can h_1 and h_2 be compressed, and what is the result of the compression? We

first consider the case where h_1 and h_2 are the only histories in the category node. To figure out how to define consistency of spans, we need to remind ourselves what a packed span means: it is a packed way of storing a list of context-span pairs, where here 'span' is a simple set of indices. We need to consider what happens in each maximally-specific context C in the choice space. If C is inconsistent with both C_1 and C_2 then h_1 and h_2 do not exist under C, and so the result should not exist under C either. If C is consistent with C_1 but not with C_2 then h_2 does not exist under C and the compression under C is simply $PS_1|_C$. Similarly if C is consistent with C_2 only. Finally, if C is consistent with both C_1 and C_2 then we must require that $PS_1|_C = PS_2|_C$.

The condition is therefore:

(306) If a category node includes exactly two histories h_1 and h_2 then they can be compressed if their categories are identical (including required index sets), and for every maximally-consistent context $C \in C_1 \wedge C_2$ we have $PS_1|_C = PS_2|_C$.

If this condition holds, we say that h_1 and h_2 are compressible. How should h_1 and h_2 be compressed in that case? We should simply use (307)a. It is easy to verify that (307)a=(307)b.

(307) a. $\mathit{union}(\{PS_1|_C \mid C \in C_1\} \cup \{PS_2|_C \mid C \in C_2\})$
b. $\mathit{union}(PS_1, PS_2)$

If h_1 and h_2 in node A are compressible then we use $\mathit{pack}(h_1, h_2)$ to designate the compressed span: $\langle C_1 \vee C_2,\ A,\ \mathit{union}(PS_1, PS_2),\ \{\langle h_1, h_2\rangle\}\rangle$.

Let's check this definition on a few examples. If the two spans are identical, they are obviously consistent. Also, if their contexts are mutually inconsistent, then the spans are compressible, simply because each one talks about a different region of the choice-space and there is no clash. A third case is theoretically possible. Suppose that the top context is divided into three subcontexts, i.e. $1 : \mathit{one\text{-}of}(A1, A2, A3)$. Now consider the two spans:

(308) $[A1 \vee A2;\ \langle A1 : \{1\}, A2 : \{2\}\rangle]$
$[A2 \vee A3;\ \langle A2 : \{2\}, A3 : \{3\}\rangle]$

They are not identical, but their contexts are not mutually inconsistent, since $(A1 \vee A2) \wedge (A2 \vee A3) = A2$ (because $A1 \wedge A3 = 0$), and $A2 \neq 0$. Because the spans are equal under the shared context $A2$, they are compressible, with the result:

(309) $[1;\ \langle A1:\{1\}, A2:\{2\}, A3:\{3\}\rangle]$

The following two may at first seem compressible, because what they say about context 1 is the same, and also $A1$ and $A2$ are mutually inconsistent:

(310) $\langle 1:\{5\}, A1:\{6\}\rangle \qquad \langle 1:\{5\}, A2:\{7\}\rangle$

However, when we consider what happens under each context of the choice-space, we see that these spans are equivalent to:

(311) $\langle A1:\{5,6\}, A2:\{5\}\rangle \qquad \langle A1:\{5\}, A2:\{5,7\}\rangle$

Here, the first span has $\{5,6\}$ under $A1$ while the second has $\{5\}$ there; and they also disagree under $A2$. Therefore, the two spans cannot be compressed. To make this point clearer, even $\langle 1:\{5\}\rangle$ is not compressible with $\langle 1:\{5\}, A2:\{7\}\rangle$ because the first claims that the span under $A2$ is $\{1\}$ while the second claims it is $\{5,7\}$.

The definition in (306) is inefficient because we need to check each maximally-consistent context $C \in C_1 \wedge C_2$. However, the following efficient definition is equivalent to (306):

(312) If a category node includes exactly two histories h_1 and h_2 then they can be compressed if their categories are identical (including required index sets), and: $PS_1|_{C_1 \wedge C_2} = PS_2|_{C_1 \wedge C_2}$.

Compression of More Than Two Histories

If a category node has more than two histories then we follow these steps:

- First, every pair of histories that have identical categories (including required indices) and identical packed spans can be simply compressed (this is as in the unpacked algorithm).
- If $n > 1$ histories remain then for each pair of histories, calculate whether they are compressible. Any disjoint clique of histories under the compressibility relation can be compressed (by taking the *union* of the histories' spans). A disjoint clique under the compressibility relation is a maximal set of histories that are pairwise compressible, where each of the histories is not compressible with any other history outside the set.

- Do not compress anything else. In particular, if h_1 is compressible with each of h_2 and h_3, but h_2 and h_3 are not compressible, then ($\{h_1, h_2, h_3\}$ does not form a clique of histories, and) do not compress h_1 with either h_2 or h_3 (we cannot decide which one to compress h_1 with).

For the last point, note that compressibility is not an equivalence relation, i.e. if h_1 is compressible with h_2, and h_2 is with h_3, it does not follow that h_1 is compressible with h_3. For example, take $PS_1 = \langle A1 : \{1\}\rangle$, $PS_2 = \langle A2 : \{3\}\rangle$, $PS_3 = \langle A1 : \{2\}\rangle$. An example where such a situation can happen in practice is when a VP in the top context has one VP-modifier under context $A2$ and two modifiers under context $A1$.

Proofs of Correctness

Proposition 6 *The two definitions of compression, (306) and (312), are equivalent.*

Proof: Suppose that (306) holds. From Lemma 3 it easily follows that $PS_1|_{C_1 \wedge C_2} = union\{PS_1|_C \mid C \in C_1 \wedge C_2\}$. By (306), this is equal to $union\{PS_2|_C \mid C \in C_1 \wedge C_2\}$, which is equal to $PS_2|_{C_1 \wedge C_2}$. Hence we showed (312).

Now suppose that (312) holds, so $PS_1|_{C_1 \wedge C_2} = PS_2|_{C_1 \wedge C_2}$. Then for all $C \in C_1 \wedge C_2$ we have $(PS_1|_{C_1 \wedge C_2})|_C = (PS_2|_{C_1 \wedge C_2})|_C$. By Lemma 5, $PS_1|_{C_1 \wedge C_2 \wedge C} = PS_2|_{C_1 \wedge C_2 \wedge C}$. But $C \in C_1 \wedge C_2$ so $C_1 \wedge C_2 \wedge C = C$, and so $PS_1|_C = PS_2|_C$. ■

Now, for the proof that compression is correct. We first need a few lemmas.

Lemma 7 *If two histories h_1 and h_2 with respective contexts C_1 and C_2 are compressible then $pack(h_1, h_2)|_{C_1} = h_1$, and similarly with h_2 and C_2.*

Proof: By definition, $PS_2 = PS_2|_{C_2}$, and so $PS_2|_{C_1} = (PS_2|_{C_2})|_{C_1}$, which is equal to $PS_2|_{C_1 \wedge C_2}$ by Lemma 5. Because h_1 and h_2 are compressible, this is equal to $PS_1|_{C_1 \wedge C_2}$. So:

(a) $PS_2|_{C_1} = PS_1|_{C_1 \wedge C_2}$

By Lemma 4, $union(PS_1, PS_2)|_{C_1} = union(PS_1|_{C_1}, PS_2|_{C_1})$. This is equal, because of (a), to $union(PS_1|_{C_1}, PS_1|_{C_1 \wedge C_2})$. From the definitions it follows that this is equal to: $union(union(PS_1|_{C_1 \wedge C_2}, PS_1|_{C_1 \wedge \neg C_2}), PS_1|_{C_1 \wedge C_2})$ which is equal to:
$union\{PS_1|_{C_1 \wedge C_2}, PS_1|_{C_1 \wedge \neg C_2}, PS_1|_{C_1 \wedge C_2}\} = PS_1|_{C_1} = PS_1$.
Hence, $union(PS_1, PS_2)|_{C_1} = PS_1$. ■

Lemma 8
$comb\text{-}cxt(h_1|_C, h_2) = comb\text{-}cxt(h_1, h_2)|_C = comb\text{-}cxt(h_1, h_2|_C)$
$comb(h_1|_C, h_2) = comb(h_1, h_2)|_C = comb(h_1, h_2|_C)$

Proof: This easily follows from the definitions of *comb-cxt* and *comb*. ■

Theorem 4 *Compression in the packed algorithm leads to correct results.*

Proof: We need to show that every time we do a compression, we do not change the resulting proofs under each of the contexts of the compressed histories. Specifically, suppose a category node A includes exactly two histories h_1 and h_2 (with contexts C_1 and C_2) that are compressible, and suppose that A combines with another category node B. Then for every history h_3 in B, we need to show:

(a1) $comb(h_1, h_3) = comb(pack(h_1, h_2), h_3)|_{C_1}$ if A is the major category, or

(a2) $comb(h_3, h_1) = comb(h_3, pack(h_1, h_2))|_{C_1}$ if B is the major category

and similarly with h_2 and C_2. For brevity, we'll simply stick with (a1) now, but the argument is the same for (a2) as well. The next step is: if $comb(h_1, h_3)$ combines with another history h_4, we need to show:

(b) $comb(comb(h_1, h_3), h_4) = comb(comb(pack(h_1, h_2), h_3), h_4)|_{C_1}$

And so on with any additional history that combines in the rest of the derivation.

For (a1), we have $comb(h_1, h_3) = comb(pack(h_1, h_2)|_{C_1}, h_3)$ by Lemma 7. By Lemma 8, this is equal to $comb(pack(h_1, h_2), h_3)|_{C_1}$. The second step (b) is obtained using (a1) and Lemma 8 again: $comb(comb(h_1, h_3), h_4) = comb(comb(pack(h_1, h_2), h_3)|_{C_1}, h_4) = comb(comb(pack(h_1, h_2), h_3), h_4)|_{C_1}$. We can keep going like this for any further combinations.

The proof that compression is correct when there are more than two histories in a node is similar. In particular, generalizing Lemma 7, if $h_1, \ldots, h_n$ are pairwise compressible then $pack(h_1, \ldots, h_n)|_{C_j} = h_j$ for $1 \leq j \leq n$ (if h_1, h_2, h_3 are pairwise-compressible then h_1 is compressible with $pack(h_2, h_3)$). ■

7.4.5 Inside a Basic Cycle

Default Behavior

The default behavior is simply to run a chart algorithm on all the input histories, calculating all packed combinations using the definition from section 7.4.3. Compression is allowed

only between histories that have identical categories and identical spans.

Preventing Premature Departure from a Basic SCC

We want to extend optimization (252) from section 6.4 to the packed case. Consider what happens with the premises:

(313)

context	premise
1	$C:\{1\},\ C \to C:\{2\}$
$A1$	$C \to C:\{3\}$
$A2.1$	$C \to C:\{4\}$

where the choice-space is defined using 1 : *one-of*$(A1, A2)$ and $A2$: *one-of*$(A2.1, A2.2)$. Under context $A1$, the derivation will have two permutations of the modifiers: first adding 2 and then 3 or vice-versa. Furthermore, using the optimizations from section 6.4, only one final history will remain, with index-set $\{1, 2, 3\}$ and two parent-links. Under context $A2.1$, there will also be two permutations and one final history with index-set $\{1, 2, 4\}$ and two parent-links. We also need to remember context $A2.2$, under which there will be one final history with index-set $\{1, 2\}$ (even though this is a subset of $\{1, 2, 3\}$ and $\{1, 2, 4\}$).

The default behavior of the chart algorithm (with packed combination) in this example will yield the following histories:

history	context	category	span	parent-links
h_1	1	C	$\langle 1:\{1\}\rangle$	
h_2	1	$C \to C$	$\langle 1:\{2\}\rangle$	
h_3	$A1$	$C \to C$	$\langle A1:\{3\}\rangle$	
h_4	$A2.1$	$C \to C$	$\langle A2.1:\{4\}\rangle$	
h_5	1	C	$\langle 1:\{1,2\}\rangle$	$\langle 1:\{\langle h_2, h_1\rangle\}\rangle$
h_6	$A1$	C	$\langle A1:\{1,3\}\rangle$	$\langle A1:\{\langle h_3, h_1\rangle\}\rangle$
h_7	$A2.1$	C	$\langle A2.1:\{1,4\}\rangle$	$\langle A2.1:\{\langle h_4, h_1\rangle\}\rangle$
h_8	$A1$	C	$\langle A1:\{1,2,3\}\rangle$	$\langle A1:\{\langle h_3, h_5\rangle, \langle h_2, h_6\rangle\}\rangle$
h_9	$A2.1$	C	$\langle A2.1:\{1,2,4\}\rangle$	$\langle A2.1:\{\langle h_4, h_5\rangle, \langle h_2, h_7\rangle\}\rangle$

The compression in h_8 is thanks to the fact that the two histories that led to it share the same category C and span $\{1, 2, 3\}$; and similarly with h_9. There are no histories with spans $\{1, 3, 4\}$ or $\{1, 2, 3, 4\}$ because the modifiers 3 and 4 are in mutually-inconsistent contexts.

As we did when we extended the definition of combination and compression above, the correct behavior here can be obtained if we consider what happens in each maximally-specific context in the choice-space. Extending (252), we get:

(314) In a basic SCC with node A and modifier $A \to A$:

For each maximally-specific context C, let H be the set of all initial histories of A whose context is consistent with C, and let H' be the restriction of the histories in H to C. If there is an index that is shared by all histories in H' then apply the following optimization:

Let K be the set of all histories in A whose context is consistent with C, let M be the set of all histories in $A \to A$ whose context is consistent with C, and let M' be the restriction of the histories in M to C.
Then, if a history h in K, restricted to C, does not include the main index of some history in M', then replace h with $h|_{\neg C}$.

This is the inefficient definition as it considers each maximally-specific C separately. The efficient definition is the following:

(315) In a basic SCC with node A and modifier $A \to A$, let H be the set of initial histories in A. For an index i and a context G, define:
$sc(i, G) = G \wedge \bigwedge\{D \mid \exists h.\ D : i \in h,\ h \in H,\ cxt(h) \wedge G \neq 0\}$. This is the sub-context of G in which i is shared in all initial histories that are consistent with G.

For each main index i of a history in $A \to A$ such that i appears there with span D, do:
For each history h in A with context C, where $B : i \in span(h)$ ($B = 0$ if $i \notin span(h)$), do:
Let $E = D \wedge C \wedge \neg B$, and let $F = \bigvee_i sc(i, E)$.

Now, if $F = 0$ then do nothing. If $F \neq 0$ then filter h from the output of node A, and if $G = (C \wedge \neg F) \neq 0$ then add a new history $\langle G,\ A,\ span(h)|_G,\ \langle G : \{h\}\rangle\rangle$ to A.

The explanation for this definition is: If we have a main index i in a history in $A \to A$, and it appears there with span D, then we need to investigate histories in which i does not appear under context D. For a history h, we look at E, which is the sub-context of

$cxt(h)$ in which i does not appear but potentially should appear. We need to filter h only in those parts of E where in fact we have a shared index between all initial histories.

Although the definition looks complicated, often many context terms are either 0 or 1, and the computation is easy. For example, if a modifier has been used to get to h then $B = C$, and so we can quickly see that $E = 0$, $F = 0$, $G = C$, and h need not be changed. If a modifier is inconsistent with h then $D \wedge C = 0$, and again h need not be changed.

Furthermore, in practice, with a glue semantics input, the initial histories in a basic SCC are typically all under one context K, and share an index under that context. This then causes $sc(i, E)$ to become $E \wedge K$. We can check for such cases first in order to speed up the computation of the general definition. Or we can simply memoize the computation of sc.

In example (313), h_2 causes h_1 to be filtered: $i = 2$, $D = 1$, $C = 1$, $B = 0$, $E = 1$, $F = 1$, $G = 0$, and $h_1|_G$ is then completely filtered. Similarly, h_2 causes the filtering of h_6 and h_7. The interesting step is that h_5 is not completely filtered out. First, h_3 causes h_5 to be filtered to $h_5|_{\neg A1}$ (because $i = 3$, $D = A1$, $C = 1$, $B = 0$, $E = A1$, $F = A1$, $G = A2$). Then, h_4 causes this to be filtered further to $(h_5|_{\neg A1})|_{\neg A2.1}$. Therefore, h_5 survives under $A2.2$, as desired. Thus, the three histories that are to be output from the SCC are:

history	context	category	span	parent-links
h_8	$A1$	C	$\langle A1 : \{1,2,3\}\rangle$	$\langle A1 : \{\langle h_3, h_5\rangle, \langle h_2, h_6\rangle\}\rangle$
h_9	$A2.1$	C	$\langle A2.1 : \{1,2,4\}\rangle$	$\langle A2.1 : \{\langle h_4, h_5\rangle, \langle h_2, h_7\rangle\}\rangle$
$h_5{}'$	$A2.2$	C	$\langle A2.2 : \{1,2\}\rangle$	$\langle A2.2 : \{\langle h_5\rangle\}\rangle$

Before these final histories are actually output from the SCC, they can be compressed as in section 7.4.4. This gives:

(316) $\langle 1,\ C,\ \langle 1 : \{1,2\}, A1 : \{3\}, A2.1 : \{4\}\rangle,\ \langle A1 : \{\langle h_8\rangle\}, A2.1 : \{\langle h_9\rangle\}, A2.2 : \{\langle h_5{}'\rangle\}\rangle\rangle$

Theorem 5 *The optimization in a basic SCC in the packed algorithm leads to correct results.*

Proof: Optimization (314) leads to correct results by virtue of Proposition 4 from section 6.4. To show that optimization (315) is equivalent to optimization (314), we need to show that for every maximally-specific context J, a history h is restricted to $\neg J$ by (314) iff $J \in C \wedge F$ in (315). This easily follows from the definitions. ■

7.4.6 Calculating Packed Meanings

Choice-Space Strategy

The revision needed for the algorithm in section 6.5.2 is in step 2. When you encounter a history with $n > 1$ parent-links, there can be two cases. In the first case, all the parent-links are under one context. This is the case that was discussed in section 6.5.2, and it results from an internal (modifier) ambiguity. In that case, the context needs to be split into n sub-contexts.

In the second case, the parent-links are under at least two different contexts, and all the contexts are pairwise inconsistent. The set of parent-links under each of these contexts may have more than one member. Each such set with $n > 1$ elements indicates an internal modifier ambiguity within that context, and that context needs to be split into n sub-contexts.

If the parent-links are under at least two different contexts, but these contexts are not all pairwise inconsistent, the situation requires re-arranging the compressed histories. For example, suppose the choice-space is defined by $1 : \textit{one-of}(A1, A2, A3)$, and a history h_1 has a set of parent-links: $\langle A1 \vee A2 : \{\langle h_2 \rangle\}, A2 \vee A3 : \{\langle h_3 \rangle\}\rangle$, and h_2 has a parent-link $\langle A1 : \{\langle h_4, h_5 \rangle\}, A2 : \{\langle h_6, h_7 \rangle\}\rangle$, and h_3 has a parent-link $\langle A2 : \{\langle h_8, h_9 \rangle\}, A3 : \{\langle h_{10}, h_{11} \rangle\}\rangle$. Then we must have $span(h_2)|_{A2} = span(h_3)|_{A2}$ (otherwise, h_2 and h_3 could not compress into h_1). We see there are two different ways to get that span under $A2$. So we would want to create a split in the choice space under $A2$, i.e. $A2 : \textit{one-of}(A2.1, A2.2)$, and a meaning term: $l_{new} = \langle A2.1 : app(l_6, l_7), A2.2 : app(l_8, l_9)\rangle$, and another meaning term:
$l_1 = \langle A1 : app(l_4, l_5), A2 : l_{new}, A3 : app(l_{10}, l_{11})\rangle$. However, this situation does not occur with glue semantics input because it would correspond to having two completely different ways of getting to a category but not because of either an internal modifier ambiguity or an external ambiguity in the packed input.

The other change needed in step 2 is that when we flow a context C down through a parent-link with context C', we need to flow $C \wedge C'$ under that parent-link.

As an example, consider Figure 7.1 again. Step 1 detects that h_1 participates in two histories (that lead to the final history h_9), so $L' = \{h_1\}$. Step 2 flows context 1 starting in h_9. The traversal then reaches h_7. That history, which is a compression of h_4 and h_6, has the packed parent-links component: $\langle A1 : \{\langle h_4 \rangle\}, A2 : \{\langle h_6 \rangle\}\rangle$. Therefore, we flow the context $A1$ to h_4 and its predecessor histories, except that when we reach h_1 we do not

continue to its parent-links (if it had any) because $h_1 \in L'$, and instead we just indicate $A1$ on it. Similarly, we flow $A2$ to h_6 and its predecessor histories, except we just mark $A2$ on h_1. When this round is done, we start a new round at h_1, with the context which is the disjunction of all the contexts that appear in it, i.e. $A1 \vee A2 = 1$, which is indeed the correct context of h_1.

At the end of step 3, we will have the following chunks of meaning:

(317)
$l_1 : anarrow$	$l_{14} : like$ (verb)
$l_{10} : like$ (preposition)	$l_5 : app(l_{14}, l_{13})$
$l_2 : app(l_{10}, l_1)$	$l_6 : app(l_5, l_1)$
$l_{11} : time$	$l_7 : \langle A1 : l_4, A2 : l_6 \rangle$
$l_{12} : fly$	$l_{15} : john$
$l_3 : app(l_{12}, l_{11})$	$l_{16} : think$
$l_4 : app(l_3, l_2)$	$l_8 : app(l_{16}, l_{15})$
$l_{13} : timeflies$	$l_9 : app(l_8, l_7)$

At the end of step 4, we will have:

(318) $l_1 : anarrow$

$l_9 : app(app(think, john),$

$\langle A1 : app(app(fly, time), app(like, l_1)), A2 : app(app(like, timeflies), l_1) \rangle)$

Since l_1 is used twice, we do not substitute its meaning into the form. At the end of step 5, we will have:

(319) $l_1 : anarrow$

$l_9 : think(john, \langle A1 : fly(time, like(l_1)), A2 : like(timeflies, l_1) \rangle)$

This is the desired result. (Remember this was a simplified example, so it is not a realistic meaning representation). More examples will be given in section 7.5.

Underspecified Representations

Continuing example (313) above, assume meaning m_i for premise i. In each of the three final contexts, we get:

(320)
context	underspecified meaning form
$A1$	$mod\_str(m_1, [m_2, m_3])$
$A2.1$	$mod\_str(m_1, [m_2, m_4])$
$A2.2$	$mod\_str(m_1, [m_2]) \quad = m_2(m_1)$

A simple way to represent this with few repetitions is:

(321) $l_5 : \langle A1 : mod\_str(l_1, [l_2, l_3]), A2.1 : mod\_str(l_1, [l_2, l_4]), A2.2 : mod\_str(l_1, [l_2])\rangle$
$l_1 : m_1 \quad l_2 : m_2 \quad l_3 : m_3 \quad l_4 : m_4$

There are still some repetitions here. For example, imagine we have 5 elements like each of l_2, l_3, and l_4. We may want to write under $A1$: $mod\_str(l_1, append(l_6, l_7))$ where $l_6 = [l_{2a}, l_{2b}, l_{2c}, l_{2d}, l_{2e}]$ is shared between the three options. Although there is some saving here, it is not that easy to calculate. Since we do not have many modifiers per node, it is not worth the trouble.

So we simply follow the strategy of calculating all histories within a basic SCC, and then for each final history, we create a *mod_str* for it in its proper context.

7.4.7 Unpacking Packed Representations

Given a packed representation and a context C in the choice space, we want to get a less ambiguous representation that includes only those parts consistent with C. In the extreme case, the resulting representation is one normal unpacked representation.

We simply start flowing C recursively from the top of the representation. Whenever we reach an ambiguity junction $\langle D_1 : \varphi_1, \ \ldots, \ D_n : \varphi_n\rangle$, we intersect C with each D_i. Only the non-false intersections where $C \wedge D_i \neq 0$ survive. This gives us a new ambiguity junction $\langle C \wedge D_{i_1} : \varphi_{i_1}, \ \ldots, \ C \wedge D_{i_k} : \varphi_{i_k}\rangle$, where $i_1, \ldots, i_k$ are the indices of the non-false intersections. We keep flowing each $C \wedge D_{i_j}$ into its respective φ_{i_j}. If we have $k = 1$ then we just replace the ambiguity junction with φ_{i_1} and flow $C \wedge D_{i_1}$ into it. Note that as long as the original C is not a nogood in the choice-space, we will never reach a case where the context that is flowed into an ambiguity junction gives a false intersection with all the contexts there.

7.5 Interesting Cases

7.5.1 PP Attachment Ambiguity

Basic Results

Consider example (292) again. It is instructive to go through all the steps in detail to see what the algorithm does. At the end of step 3 from section 6.5.2, we get:

```
meaning(l:1,(v:12)\exists((v:13)\[v:12,v:13]))
meaning(l:2,(v:18)\(v:19)\(v:20)\[and,[v:19,v:20],xle_gf(v:20,subj,v:18)])
meaning(l:3,Bill)
meaning(l:4,app(l:2,l:3))
meaning(l:5,(v:21)\(v:22)\(v:23)\[and,[v:22,v:23],xle_gf(v:23,obj,v:21)])
meaning(l:6,(v:4)\iota(v:4))
meaning(l:7,girl)
meaning(l:8,(v:8)\(v:9)\(v:10)\[and,[v:9,v:10],[v:8,v:10]])
meaning(l:9,(v:1)\(v:2)\[[preposition,with],v:2,v:1])
meaning(l:10,(v:6)\iota(v:6))
meaning(l:11,telescope)
meaning(l:12,v:5)
meaning(l:13,app(l:11,l:12))
meaning(l:14,(v:5)\(l:13))
meaning(l:15,app(l:10,l:14))
meaning(l:16,app(l:9,l:15))
meaning(l:17,v:14)
meaning(l:18,v:7)
meaning(l:19,ambig([A1-(l:17),A2-(l:18)]))
meaning(l:20,app(l:16,l:19))
meaning(l:21,(v:7)\(l:20))
meaning(l:22,app(l:8,l:21))
meaning(l:23,app(l:22,l:7))
meaning(l:24,ambig([A1-(l:7),A2-(l:23)]))
meaning(l:25,v:3)
meaning(l:26,app(l:24,l:25))
meaning(l:27,(v:3)\(l:26))
meaning(l:28,app(l:6,l:27))
meaning(l:29,app(l:5,l:28))
meaning(l:30,[verb,see])
meaning(l:31,app(l:29,l:30))
meaning(l:32,app(l:4,l:31))
meaning(l:33,(v:15)\(v:16)\(v:17)\[and,[v:16,v:17],[v:15,v:17]])
meaning(l:34,(v:14)\(l:20))
meaning(l:35,app(l:33,l:34))
meaning(l:36,app(l:35,l:32))
meaning(l:37,ambig([A2-(l:32),A1-(l:36)]))
meaning(l:38,v:11)
meaning(l:39,app(l:37,l:38))
meaning(l:40,(v:11)\(l:39))
meaning(l:41,app(l:1,l:40))
```

At the end of step 4 we get:

```
meaning(l:42,app(app((v:1)\(v:2)\[[preposition,with],v:2,v:1],
                     app((v:6)\iota(v:6),(v:5)\app(telescope,v:5))),
                 ambig([A1-(v:14),A2-(v:7)])))
meaning(l:43,app(app((v:18)\(v:19)\(v:20)\[and,[v:19,v:20],
                                           xle_gf(v:20,subj,v:18)],Bill),
                 app(app((v:21)\(v:22)\(v:23)\[and,[v:22,v:23],xle_gf(v:23,obj,v:21)],
                         app((v:4)\iota(v:4),
                             (v:3)\app(ambig([A1-girl,A2-T]),v:3))),
```

```
                    [verb,see])))
T = app(app((v:8)\(v:9)\(v:10)\[and,[v:9,v:10],[v:8,v:10]],
            (v:7)\(l:42)),
        girl)
meaning(l:44,app((v:12)\exists((v:13)\[v:12,v:13]),
                 (v:11)\app(ambig([A2-(l:43),
                                   A1-app(app((v:15)\(v:16)\(v:17)\[and,[v:16,v:17],
                                                                    [v:15,v:17]],
                                              (v:14)\(l:42)),
                                          l:43)]),
                            v:11)))
```

(The "T = ..." part is not part of the representation, I use it here just because there is not enough space on the page to fit the entire representation.) Notice that girl appears twice because it is an atom, and so it is not worth it to keep it under a separate label. In contrast, in the ambig under l:44, we have l:43 twice because its meaning is complex.

In step 5 we do app reduction:

$$(322)\ \texttt{app}(\varphi,\psi) \quad \rightsquigarrow \quad \begin{cases} \text{subst } \psi \text{ for } (\texttt{v}{:}i) \text{ in } \delta & \text{if } \varphi = (\texttt{v}{:}i)\backslash\delta \\ [\varphi,\psi] & \text{if } \varphi \text{ is basic} \\ \texttt{app}(\varphi,\psi) & \text{otherwise} \end{cases}$$

Thus we get:

```
meaning(l:45,[[preposition,with],
              ambig([A1-(v:14),A2-(v:7)]),
              iota((v:5)\[telescope,v:5])])
meaning(l:46,
  (v:20)\[and,
          [(v:23)\[and,
                   [[verb,see],v:23],
                   xle_gf(v:23,obj,
                          iota((v:3)\app(ambig([A1-girl,
                                                A2-(v:10)\[and,
                                                           [girl,v:10],
                                                           [(v:7)\(l:45),v:10]]]),
                                         v:3)))
                   ],
           v:20],
          xle_gf(v:20,subj,Bill)])
meaning(l:47,exists((v:13)\[(v:11)\app(ambig([A2-(l:46),
                                              A1-(v:17)\[and,
                                                         [l:46,v:17],
                                                         [(v:14)\(l:45),v:17]]]),
                                       v:11),
                            v:13]))
```

Here is an explanation of these results. The initial history in the SCC of $girl : \mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$ has the meaning `girl` (this is `l:7` above) in context `1`. The SCC has one modifier under context `A2` ("with the telescope"). The meaning of the basic unit of this modifier is

(323) `(v:2)\[[preposition,with],v:2,iota((v:5)\[telescope,v:5])]` : $\mathbf{4}_v^e \rightarrow \mathbf{4}_r^t$

which corresponds to `l:16` above. Depending on which context we are in, `A1` or `A2`, this meaning is applied on either the variable `v:14` or the variable `v:7`. Both are of category $\mathbf{4}_v^e$, so they are compressed in `l:19`, and the application of `l:16` on that yields `l:20`, which is later renamed to `l:45`.

Now, under `A2`, we abstract over the correct variable `v:7` (in `l:21`), and then use the intersective modifier `l:8` on it to get the following modifier of "girl" (this meaning corresponds to `l:22`):

(324) `(v:9)\(v:10)\[and,[v:9,v:10],[(v:7)\(l:45),v:10]])` : $(\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t) \rightarrow \mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$

Under `A2`, we apply this modifier on

(325) `girl` : $\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$

to get:

(326) `(v:10)\[and,[girl,v:10],[(v:7)\(l:45),v:10]])` : $\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$

but under `A1`, (325) remains unmodified. This is why we get

```
app(ambig([A1-girl,
           A2-(v:10)\[and,
                      [girl,v:10],
                      [(v:7)\(l:45),v:10]]]),
    v:3)
```

as part of the meaning of `l:46` (this part corresponds to `l:26`).

Further Reductions

The representations under `l:45`, `l:46`, `l:47` above are not exactly as in (293). Should we keep refining the representations here until they match those exactly? At this point, we need to stop and ask ourselves again what is our goal.

The algorithm presented here so far does achieve the goal of computing a correct packed meaning representation in an efficient manner. Efficient computation of this representation was one aim. A second aim is efficient computation *with* this representation, i.e. the next step in the pipeline of processing. This aim has been partly achieved at PARC by skolemizing and flattening the representations, i.e. performing further reductions on the representations, as was mentioned in section 1.3.3 under "Packed Reasoning".

A third aim might be creating a packed representation that is easily readable by a human. One thing that can help to reduce the clutter above is performing η-reductions, thus getting rid of the variables `v:5` and `v:13`. Another thing we could do is add a clause to (322) which reduces `app` when the functor is ambiguous:

(327) `app(ambig(`C_1`-`φ_1`,...,`C_1`-`φ_n`),`ψ`)`
$\rightsquigarrow$ `ambig(`C_1`-app(`φ_1`,l:`new`),...,`C_n`-app(`φ_n`,l:`new`))`
`meaning(l:`new`,`ψ`)`

This simply pushes the `app` into the `ambig` form. We want `l:`new in case ψ is complex (so that we will not have multiple copies of it), but if ψ is simple, such as a variable, we can just put it in place of `l:`new. In `l:46` above, this will give us:

```
ambig([A1-app(girl,v:3),
       A2-app((v:10)\[and,
                      [girl,v:10],
                      [(v:7)\(l:45),v:10]],v:3)])
```

By applying the `app` reduction again, we get:

```
ambig([A1-[girl,v:3],
       A2-[and,
           [girl,v:3],
           [(v:7)\(l:45),v:3]]])
```

In this way, we got rid of all the `app`s. But we still have further applications within square brackets. A suggestion is to use another beta-reduction:

(328) `[(v:`i`)\`φ`,`ψ`]` $\rightsquigarrow$ subst ψ for `(v:`i`)` in φ

However, we need to be careful here. It is not fine to simply substitute `v:3` instead of `v:7`. This is because `v:7` occurs inside `l:45`. The reduction (328) should apply only under the

condition that (a) either (v:i) does not appear in any meaning term that can be reached from inside φ by following labels, or (b) if it does appear in this way, then substituting ψ for it there is consistent with all other uses of the embedded meaning term. Here, for example, v:7 is mentioned only once under l:46 and once under the embedded meaning l:45. So we can simplify the part of l:46 to

```
ambig([A1-[girl,v:3],
       A2-[and,
           [girl,v:3],
           l:45]])
```

provided we change v:7 to v:3 inside l:45. A similar thing can be said about v:14 inside l:47. We can also try to detect the special case where [girl,v:3] appears twice and factor it out. All these changes will result in a representation as in (293).

7.5.2 Quantifier Scope Ambiguity

In section 4.5.3, I showed how the glue category of a quantifier can be made ambiguous depending on which clause the quantifier lands on. If there is just one possibility, we get just one answer, as in (329) (this assumes eta-reduction applied).

(329) Every man left.

```
quant(every,
      man,
      (v:1)\exists((v:5)\[and,[[verb,leave],v:5],xle_gf(v:5,subj,v:1)]))
```

Simple Ambiguity

If there are two possibilities, we get a packed result. For example, assume that *every man* can float to the top clause:

(330) [John thinks that [[every man]$_2$ left]$_1$]$_0$.

```
meaning(l:51,exists((v:9)\[and,[[verb,leave],v:9],
                                xle_gf(v:9,subj,ambig([A2-(v:5),A1-(v:1)]))]))
meaning(l:52,[man,ambig([A2-(v:6),A1-(v:2)])])
meaning(l:53,exists((v:12)\[and,
                            [and,[[verb,think],v:12],
                                 xle_gf(v:12,comp,ambig([A2-(l:51),
```

```
                                                    A1-quant(every,
                                                             (v:2)\(l:52),
                                                             (v:1)\(l:51))])),
                                  xle_gf(v:12,subj,John)]))
meaning(l:54,ambig([A1-(l:53),A2-quant(every,(v:6)\(l:52),(v:5)\(l:53))]))
```

The reason for the ambiguity under `l:48` here is that we get the following premises:

(331)

context	premises
A1	every : $(\mathbf{2}_v^e \to \mathbf{2}_r^t) \to (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{1}^t$
A2	every : $(\mathbf{2}_v^e \to \mathbf{2}_r^t) \to (\mathbf{2}^e \to \mathbf{0}^t) \to \mathbf{0}^t$

They are different only at the end of the category. Therefore, when these are compiled, we will get two premises of type $\mathbf{2}_v^e$, one with the meaning `v:6` and one with the meaning `v:12`.

One way to fix this is to enhance the algorithm so that such commonalities between premises will be detected automatically. The two premises will then be converted to one premise which is itself internally packed:

(332) every : $(\mathbf{2}_v^e \to \mathbf{2}_r^t) \to (\mathbf{2}^e \to \langle A1 : \mathbf{1}^t, A2 : \mathbf{0}^t\rangle) \to \langle A1 : \mathbf{1}^t, A2 : \mathbf{0}^t\rangle$

Then, the entire algorithm will have to be revised in order to work with packed glue categories. In particular, when applying a packed category $\langle A1 : \mathbf{a},\ A2 : \mathbf{b}\rangle$ on $C : \mathbf{a}$, the resulting history exists only under $A1 \wedge C$ and only if this context is consistent.

Another solution is not to modify the main algorithm, but to detect such cases on the resulting packed form. In (330) we can notice that the ambiguity under `l:48` is spurious in the sense that each of the variables is used only once elsewhere, and hence they can both be replaced (in all places) by one variable, say `v:50`. Further investigation will then reveal that `(v:50)\(l:48)` appears twice in two different places, and so we could factor it out (just as we could have done with `[girl,v:3]` in the previous section). The same applies to `v:5` and `v:1`. Eliminating these two ambiguities will leave only the two real ambiguities under `l:50` and `l:49`.

However, the extra ambiguities do not add much overhead, so except for the purpose of presenting forms to humans, it is probably fine to leave them as they are.

Another example similar to (330) is:

(333) [[The mother of [every student]$_2$]$_1$ arrived]$_0$

Here, *every student* may land on $\mathbf{0}^t$ (this is actually the preferred reading here) or on $\mathbf{1}_r^t$.

```
meaning(l:48,[and,[mother,v:11],
                  [[preposition,of],v:11,ambig([A1-(v:7),A2-(v:3)])]])
meaning(l:49,[student,ambig([A1-(v:8),A2-(v:4)])])
meaning(l:50,exists((v:17)\[and,[[verb,arrive],v:17],
                            xle_gf(v:17,subj,
                                   iota((v:11)\ambig([A2-(l:48),
                                                      A1-quant(every,
                                                               (v:8)\(l:49),
                                                               (v:7)\(l:48))]))))])
meaning(l:51,ambig([A1-(l:50),A2-quant(every,(v:4)\(l:49),(v:3)\(l:50))]))
```

Same-Level Quantifiers

For the sentence

(334) Every man saw some woman.

we get the following result:

```
meaning(l:40,(v:8)\quant(every,man,v:8))
meaning(l:41,(v:4)\quant(some,woman,v:4))
meaning(l:42,exists((v:9)\[and,
                           [and,[[verb,see],v:9],xle_gf(v:9,subj,v:5)],
                           xle_gf(v:9,obj,v:1)]))
meaning(l:43,ambig([A1-[l:40,(v:5)\[l:41,(v:1)\(l:42)]],
                    A2-[l:41,(v:1)\[l:40,(v:5)\(l:42)]]]))
```

This example really belongs in the previous chapter because the premises are unambiguous. The ambiguity is an internal ambiguity between the order of the quantifiers. It is automatically introduced inside the SCC of $\mathbf{0}^t$.

For the sentence:

(335) Every man gave each woman some present.

we get the following result:

```
meaning(l:73,(v:8)\quant(every,man,v:8))
meaning(l:74,(v:12)\quant(each,woman,v:12))
meaning(l:75,(v:4)\quant(some,present,v:4))
```

```
meaning(l:76,exists((v:13)\[and,[and,[and,[[verb,give],v:13],
                                          xle_gf(v:13,subj,v:5)],
                                    xle_gf(v:13,objth,v:1)],
                              xle_gf(v:13,obj,v:9)]))
meaning(l:77,[l:75,(v:1)\(l:76)])
meaning(l:78,[l:74,(v:9)\(l:76)])
meaning(l:79,[l:73,(v:5)\(l:76)])
meaning(l:80,ambig([A1-[l:73,(v:5)\ambig([A1.1-[l:74,(v:9)\(l:77)],
                                          A1.2-[l:75,(v:1)\(l:78)]])],
                   A2-[l:74,(v:9)\ambig([A2.1-[l:73,(v:5)\(l:77)],
                                          A2.2-[l:75,(v:1)\(l:79)]])],
                   A3-[l:75,(v:1)\ambig([A3.1-[l:74,(v:9)\(l:79)],
                                          A3.2-[l:73,(v:5)\(l:78)]])]]))
```

A Nested Quantifier

(336) [[Some representative of [every department]$_2$]$_1$ saw [most samples]$_3$]$_0$

(337)

context	premises
1	$see : \mathbf{0}_v^e \rightarrow \mathbf{0}_r^t$
	$\lambda P.exists(P) : (\mathbf{0}_v^e \rightarrow \mathbf{0}_r^t) \rightarrow \mathbf{0}^t$
	$\lambda x \lambda P \lambda e.P(e) \wedge subj(e,x) : \mathbf{1}^e \rightarrow (\mathbf{0}_v^e \rightarrow \mathbf{0}_r^t) \rightarrow \mathbf{0}_v^e \rightarrow \mathbf{0}_r^t$ [noscp]
	$\lambda x \lambda P \lambda e.P(e) \wedge obj(e,x) : \mathbf{3}^e \rightarrow (\mathbf{0}_v^e \rightarrow \mathbf{0}_r^t) \rightarrow \mathbf{0}_v^e \rightarrow \mathbf{0}_r^t$ [noscp]
	$some : (\mathbf{1}_v^e \rightarrow \mathbf{1}_r^t) \rightarrow (\mathbf{1}^e \rightarrow \mathbf{0}^t) \rightarrow \mathbf{0}^t$
	$rep : \mathbf{1}_v^e \rightarrow \mathbf{1}_r^t$
	$department : \mathbf{2}_v^e \rightarrow \mathbf{2}_r^t$
	$\lambda y \lambda x.of(x,y) : \mathbf{2}^e \rightarrow \mathbf{3}_v^e \rightarrow \mathbf{3}_r^t$
	$\lambda Q \lambda P \lambda x.P(x) \wedge Q(x) : (\mathbf{3}_v^e \rightarrow \mathbf{3}_r^t) \rightarrow (\mathbf{1}_v^e \rightarrow \mathbf{1}_r^t) \rightarrow \mathbf{1}_v^e \rightarrow \mathbf{1}_r^t$
$A1$	$every : (\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t) \rightarrow (\mathbf{2}^e \rightarrow \mathbf{1}_r^t) \rightarrow \mathbf{1}_r^t$
$A2$	$every : (\mathbf{2}_v^e \rightarrow \mathbf{2}_r^t) \rightarrow (\mathbf{2}^e \rightarrow \mathbf{0}^t) \rightarrow \mathbf{0}^t$

The unique thing about this example is that we have here both kinds of ambiguities, and an interaction between them. We have an external ambiguity in the input: whether *every* lands on the noun or on the top sentence. In the first option, `A1`, we get only two modifiers at the top level: $\lambda P.most(sample, P)$ and $\lambda P.some(\lambda x.every(dep, \lambda y.rep(y) \wedge of(y,x)), P)$, and so two permutations. In the second option, `A2`, we have three modifiers at the top level, but there is a constraint that *every* must outscope *some* (this is similar to what we saw in the discussion of (276)), so there are three possible permutations.

Here is the meaning representation. The top label is `1:87`, and there are eight other labels:

```
meaning(l:80,[and,[representative,v:4],
                  [[preposition,of],v:4,ambig([A1-(v:15),A2-(v:11)])]])
meaning(l:81,[department,ambig([A1-(v:16),A2-(v:12)])])
meaning(l:82,(v:6)\quant(some,
                         (v:4)\ambig([A2-(l:80),
                                      A1-quant(every,
                                               (v:16)\(l:81),
                                               (v:15)\(l:80))]),
                         v:6))
meaning(l:83,exists((v:23)\[and,[and,[[verb,see],v:23],
                                    xle_gf(v:23,obj,v:7)],
                                 xle_gf(v:23,subj,v:3)]))
meaning(l:85,(v:14)\quant(every,(v:12)\(l:81),v:14))
meaning(l:79,(v:10)\quant(most,sample,v:10))

meaning(l:84,[l:82,(v:3)\(l:83)])
meaning(l:86,ambig([C.1-[l:82,(v:3)\[l:79,(v:7)\(l:83)]],
                    C.2-[l:79,(v:7)\(l:84)]]))
meaning(l:87,ambig([A1-(l:86),
                    A2-ambig([A2.1-[l:79,(v:7)\[l:85,(v:11)\(l:84)]],
                              A2.2-[l:85,(v:11)\(l:86)]])]))
```

where:

(338) $\texttt{1} \leftrightarrow \texttt{A1} \vee \texttt{A2}$

$\texttt{A2} \leftrightarrow \texttt{A2.1} \vee \texttt{A2.2}$

$\texttt{C} \leftrightarrow \texttt{A1} \vee \texttt{A2.2}$

$\texttt{C} \leftrightarrow \texttt{C.1}$ xor $\texttt{C.2}$

Figure 7.2 shows a sketch of the main parts of the packed meaning representation. This is not the actual packed meaning representation (that was shown above), but a simplified diagram to help with visualizing the result.

How do we get this? In the SCC of category $\mathbf{0}^t$, we have two modifiers under the top context: *most* and *some representative* (+ *of every department* under `A1`); and a third modifier under context `A2`: *every*. According to the optimization in section 7.4.5, two histories will remain after the filtering, and they will be compressed. The first one will be

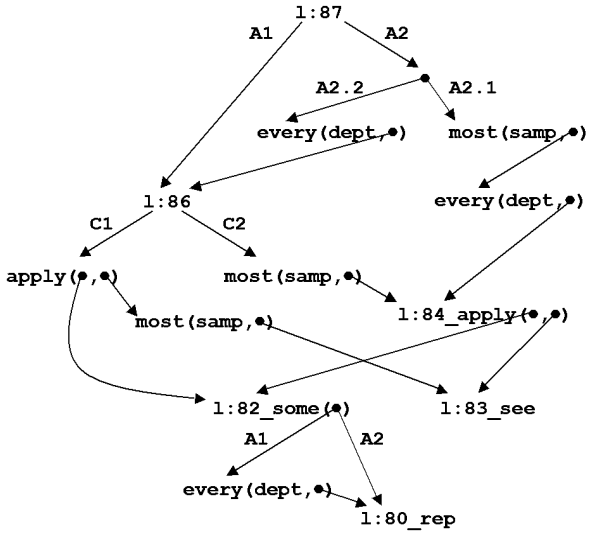


Figure 7.2: Sketch of a packed meaning representation

under `A2` and will use all three modifiers. The second one will be under `A1` and will use just the first two. This compression corresponds to the first `ambig` under `l:87`.

The final history under `A2` in $\mathbf{0}^t$ has two parent-links, and so `A2` is split into `A2.1` and `A2.2`. Under `A2.1`, *most* is the most recently applied quantifier. This forces an order between the other two quantifiers since *every* must be above *some*. Under option `A2.2`, *every* is the top quantifier. Then we have two possible orders between *most* and *some* under `l:86`. The final history under `A1` in $\mathbf{0}^t$ goes directly to `l:86` because under `A1`, only *most* and *some* exist at the top level.

When we calculate contexts in step 2 of section 7.4.6, we start from the final history in the goal node of the category graph and flow context 1 through parent-links. We encounter the $\mathbf{0}^t$ SCC, where the final compressed history already has the split between `A1` and `A2`. When we follow the `A1` parent-link, we get to the history that gives rise to `l:86`. Since that history is used twice, we do not go into it but add `A1` to its list of contexts. Later we arrive at this history again from option `A2.2`, and so we add it there. This is why `C`, the context of that history, is a disjunction of `A1` and `A2.2`. The context `C` is split again because there are two ways to use *most* and *some* under that history.

Thus we get the five analyses. There is a completely free choice between `A1` or `A2`, corresponding to whether *every* lands on the noun or at the sentence level. Under `A1`, there is a free choice between `C.1` and `C.2` corresponding to the order of *most* and *some* at

the sentence level. Under `A2`, there is a free choice between `A2.1` and `A2.2`, corresponding to whether *most* or *every* is the top quantifier. Only under `A2.2`, there is again the choice between `C.1` and `C.2`.

7.6 Complexity

The complexity of the packed algorithm is about the same as that of the algorithm in the previous chapter (see section 6.8), except for two factors. First, the local "confusion" may become a bit larger because of interaction between glue statements from different (but mutually-consistent) contexts in the choice space. Second, each set intersection computation now needs to invoke tests of consistency in the choice space. How much these cost depends on the complexity of the choice space and on the location of the particular contexts in question in the choice space.

However, we can again say that although in the worst case these factors may become exponential, in practice this rarely happens. In particular, if the choice space has a choice between $A1$ and $A2$, and an orthogonal choice between $B1$ and $B2$, then usually the two ambiguities to not interact in the glue semantics phase. For example, in the sentence "Fall leaves fall and spring leaves spring", each half is 2-way ambiguous, but the ambiguity is local to that half. Both in the F-structure and in the glue-semantics derivation and resulting packed meaning representation, the two parts will not interact. In particular, in this example we will not get a history in context $A1$ being tested for potential combination with a history in context $B1$.

7.7 Comparing to Other Underspecified Representations

In this section I compare the present work, especially chapters 4, 6, and 7, to Hole Semantics (Bos, 1996; Blackburn and Bos, 2005a). Similar comments apply to related UR formalisms: MRS (Copestake et al., 2005) and CLLS (Egg et al., 2001) (as shown in (Koller et al., 2003; Niehren and Thater, 2003; Fuchss et al., 2004), HS, MRS, and CLLS are very similar to each other).

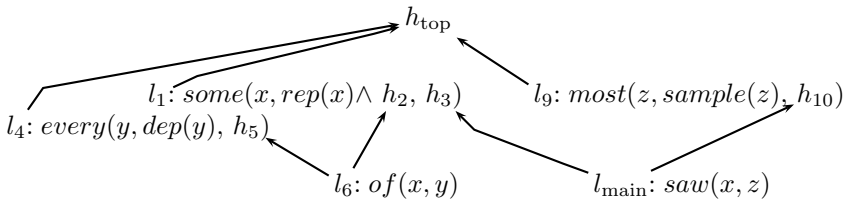


Figure 7.3: ULF for "Some representative of every department saw most samples."

7.7.1 Hole Semantics Representations

Hole Semantics (HS) is a framework for specifying syntactic dominance constraints between pieces of logical formulas. Given a logical language L, it is extended to L^* by adding meta-variables called *holes* and *labels* which can serve as (sub-)formulas of L^*. Holes are marked by $h_1, h_2, ..$ and labels by $l_1, l_2, ...$ For example, if L is first-order logic then L^* includes $\forall x.\ man(x) \rightarrow h_1$. An Unplugged Logical Form (ULF) is a tuple $\langle F, C \rangle$ consisting of a set F of labeled forms from L^* and a set C of dominance constraints between labels and holes.[3]

Figure 7.3 is a graphical representation of the ULF for sentence (336). An arrow from l to h represents the constraint $l \leq h$ meaning that l has to be subordinated to h in the final logical formula viewed as a tree. Thus $saw(x, z)$ must appear inside the scope of both quantifiers *some* and *most*, but these quantifiers' relative scoping is not determined (it is underspecified).

A legal solution ("plugging") to a ULF is a bijection from labels to holes which satisfies all the dominance constraints.[4] The formulas described by a ULF are those that can be obtained from a legal solution by actually plugging each labeled form into its corresponding hole. One legal plugging for the ULF above is:

(339) $[h_{\text{top}} = l_9;\ h_{10} = l_1;\ h_2 = l_4;\ h_5 = l_6;\ h_3 = l_{\text{main}}]$

which produces the formula:

(340) $most(z, sample(z), some(x, rep(x) \wedge every(y, dep(y), of(x, y)),$
$gave(x, John, z))).$

[3] In (Blackburn and Bos, 2005a), each ULF also consists of a set of all the holes and labels it uses. However, this set can be recovered from C.

[4] Formally, given a bijection P from labels to holes, let $G_P = \langle V, E \rangle$ be a graph, where the vertices V are all the holes and labels mentioned in the ULF, and E includes all edges (h, l) such that $P(l) = h$, and all edges (l, h) such that h is mentioned in the form labeled by l. Then P is legal iff G_P is a tree rooted at h_{top} such that for every dominance constraint $l \leq h$ in the ULF, h is an ancestor of l in G_P.

In other pluggings where h_5 dominates l_1, *every* would outscope *some*.

Blackburn and Bos (2005a) provide an algorithm for enumerating explicitly all the structures that are described by a ULF. The algorithm starts with h_{top} and non-deterministically selects a label to plug there. It then recursively traverses the labeled form and tries to plug holes there with remaining labels. A plugging fails if it violates one of the dominance constraints. The algorithm goes over all possible selections.

7.7.2 Specification of the Syntax-Semantics Interface

How are Hole Semantics ULFs defined and computed? Following (Blackburn and Bos, 2005a), this is done by using simple lambda-application, which follows the syntax via a rule-by-rule translation (as we saw in section 3.1.3), with two differences:

1. Instead of meaning terms we have partial ULFs, which sometimes include a ULF merge operation $\oplus$.

2. Each of these is preceded by the usual lambda variables, which are then usually followed by two additional lambda variables called the *local hole* and *local label*.

The merge operation is defined: $\langle F_1, C_1\rangle \oplus \langle F_2, C_2\rangle = \langle F_1 \cup F_2, C_1 \cup C_2\rangle$. Examples of entries for a proper name and an intransitive verb are:

(341) John $\lambda P\lambda h\lambda l.\ P(john, h, l)$
left $\lambda y\lambda h\lambda l.\ \langle\{l : left(y)\}, \{l \leq h\}\rangle$

We already have here the type-raised "John", for the reasons explained in section 3.1.3. When these two combine, we get

(342) $(\lambda P\lambda h\lambda l.\ P(john, h, l))(\lambda y\lambda h'\lambda l'.\ \langle\{l' : left(y)\}, \{l' \leq h'\}\rangle)\ =_\beta$
$(\lambda h\lambda l.\ (\lambda y\lambda h'\lambda l'.\ \langle\{l' : left(y)\}, \{l' \leq h'\}\rangle)(john, h, l)\ )\ =_\beta$
$\lambda h\lambda l.\ \langle\{l : left(john)\}, \{l \leq h\}\rangle$

Finally, when all the pieces of the sentence combine to form a partial ULF $\lambda h\lambda l.\psi$, we apply this on the "top hole" and the "main label" in the sentence.[5] Thus we get:

(343) $(\lambda h\lambda l.\ \langle\{l : left(john)\}, \{l \leq h\}\rangle)(h_{top}, l_{main})\ =_\beta$
$\langle\{l_{main} : left(john)\}, \{l_{main} \leq h_{top}\}\rangle$

[5] This is done by the operator **C** in (Bos, 2004).

This is the ULF for the sentence. It has only one possible plugging, with $h_{top} = l_{main}$, and so we finally get:

(344) $l_{main} : left(john)$

There have already been four levels of indirection in this simple example. The hole and label did not yet play an interesting role, but they must be defined for all entries. In the definition of quantifiers, they play a more relevant role:

(345) man $\lambda x \lambda h \lambda l.\ \langle \{l : man(x)\}, \{l \leq h\} \rangle$
every $\lambda P \lambda Q \lambda h \lambda l.\ \langle \{l_1 : every(x, l_2, h_1)\}, \{l_1 \leq h,\ l \leq h_1\} \rangle \oplus P(x, h, l_2) \oplus Q(x, h, l)$

In the definition of the quantifier, we have the local hole h and label l which are given as arguments to this partial ULF functor. The form $l_1 : every(\ldots)$ is constrained to be under the local hole h by $l_1 \leq h$. The local label l is supposed to represent the "main" part above which the l_1 form floats, and therefore l is given as argument to Q (the last argument of the quantifier) while l is restricted by $l \leq h_1$ to be under the scope part h_1 of the *every* form. The main label l_2 of the quantifier's restrictor, however, does not participate in scope flexibility. This label is just given to P and is used directly in the *every* form (i.e. we do not have there a hole h_2 and a constraint $l_2 \leq h_2$). Notice that x is not explicitly bound by lambda-abstraction, but it is implicitly bound by *every*.

After beta-reductions and merges, we get:

(346) every man
$\lambda Q \lambda h \lambda l.\ \langle \{l_1 : every(x, l_2, h_1),\ l_2 : man(x)\}, \{l_1 \leq h,\ l \leq h_1,\ l_2 \leq h\} \rangle \oplus Q(x, h, l)$

which can be visualized as:

$\lambda Q \lambda h \lambda l.$ [h ↑ $l_1 : every(x, l_2, h_1)$; $l_2 : man(x)$; l] $\oplus\ Q(x, h, l)$

The solid line represents simple equality, while the arrows represent dominance constraints. The constraint $l_2 \leq h$ is not depicted because it is implied by the fact that l_2 is a sub-expression of l_1, and $l_1 \leq h$. The constraint $l \leq h$ does not yet exist here explicitly (it will

be added inside Q), but it is implicitly included because of $l \leq h_1$, h_1 sub-expression of l_1, and $l_1 \leq h$.

When this combines with the verb *left*, we get:

$$\lambda h \lambda l. \left[\begin{array}{l} h \\ l_1 : every(x, l_2, h_1) \\ \qquad l_2 : man(x) \qquad l : left(x) \end{array} \right]$$

and this can only resolve to $every(x, man(x), left(x))$.

For a transitive verb, we have:

(347) saw $\lambda K \lambda x. K(\lambda y \lambda h \lambda l.\ \langle \{l : saw(x, y)\}, \{l \leq h\} \rangle)$

This is like the type raising of the object NP we saw in (69)a in section 3.1.3 except that every term now also accepts a hole and label, and so K is such a functor. So in fact, not all lexical entries have a leading sequence of lambda variables that ends with a local hole and label.

7.7.3 Comparison

In a syntactically unambiguous sentence, a Hole Semantics ULF and the packed semantic representation of this dissertation are comparable. Both mention common pieces just once. However, there are several important differences.

Complex Specification: First, the specification of the syntax-semantics interface in Hole Semantics is much more complicated than the glue semantics specification we saw in chapters 3-4. Not only does it suffer from the complexities of the traditional approaches that were discussed in sections 3.1.3 and 3.5.1, but it adds further complexities because the grammar writer needs to manipulate pieces of ULF directly. In particular, each semantic piece needs to specify what dominance constraints it has internally as well as w.r.t. other pieces, so one needs to resort to using a local hole and label and merge operations. Moreover, it is hard for a human to comprehend the big and complex partial ULFs and to visualize in one's mind graphically what an expression like $\langle \{l_1 : every(x, l_2, h_2), ...\}, \{l \leq h_2, ...\} \rangle$

means, and so grammar writers who want to work with partial ULFs and debug them often need to draw their constraint graphs on paper.[6] In contrast, the specification in glue semantics is straightforward. One only needs to specify the basic semantic term, its type, and simple equations relating the type's pieces to locations in the syntactic structure.

Unprincipled: Second, the underspecified representations in HS and similar frameworks are unprincipled. They were developed in an ad-hoc way rather than based on a theory of composition, like the one that glue semantics has (based on linear logic). In particular, there is no guarantee that the local hole and label game can generalize to all cases. Here is what Asudeh and Crouch (2001) say when they compare glue semantics to MRS. The point applies to HS and similar UR frameworks as well:

> There are differences between glue and MRS. Perhaps the most significant is that MRS is explicitly set up to describe logical forms. There is a long tradition in semantics, dating back to Montague's eliminable level of intensional logic, that regards this as a suspect activity. The basis of this tradition is that logical forms have no relevant, non-trivial identity criteria other than those given by their model theoretic semantics. The logical forms themselves are just arbitrary bits of syntactic notation. It is hard to tell when constraints on logical forms [such as ULF dominance constraints – I.L.] capture constraints on the underlying semantic objects they represent, and when they merely constrain arbitrary features of the chosen logical representation.
>
> Glue semantics does not run into this problem. To the extent that glue premises and scope constraints describe anything, it is the *composition* of a semantic object, and not the form of syntax that happens to be used to represent it. The semantic composition is represented as a linear logic derivation. There is a strict separation between linear logic formulas and the meaning terms attached to them, and this guarantees that the logical derivation / semantic composition is independent of syntactic idiosyncracies of the meaning representation language. Moreover, results in proof theory (e.g., proof normalization, Curry-Howard Isomorphism) establish that glue derivations have precisely the kind

[6] The same point holds for feature-structure representations of partial ULFs as well, like (Chaves, 2002; Richter and Sailer, 2003), which can be viewed simply as notational variants of partial ULFs, as well as for the syntax-semantics interface in MRS and CLLS.

> of non-trivial identity criteria required of genuine objects ("no entity without identity").

Part of the issue is that the left-hand-side of glue semantic statements is always a semantic term with a well-defined denotation, i.e. it has no free variables, unlike partial ULF. In particular, it seems that in HS most meaning terms of type τ are "type-shifted" to $\tau \rightarrow \mathbf{h} \rightarrow \mathbf{l}$, where $\mathbf{h}$ represents holes and $\mathbf{l}$ represents label. However, holes and labels are *syntactic* devices, and they do not have a denotation in the MTS of the semantic representation language. The definitions in HS, therefore, confuse two completely different things in the formulas. This is not a semantically well-justified thing to do. It is a manipulation of syntactic pieces.

A related point is the choice to use forms such as $every(x, man(x), left(x))$ in HS and MRS rather than $every(\lambda x.man(x), \lambda x.left(x))$. The choice to use the former in the standard HS grammar is symptomatic of the issue here, and in fact leads into difficulties that cannot be resolved when one tries to extend the grammar's coverage to higher-order constructions such as reciprocals (see section 13.6.1). Another related point was already mentioned in section 6.5.4 regarding constraints that are expressed directly on the presence or absence of free variables in MRS.

Procrastination: Here is another relevant quote from (Crouch, 2005):

> Packing differs from underspecification as a technique for ambiguity management. Underspecification delivers a set of fragmentary analyses plus a set of constraints recording interactions between fragments and limiting the ways they can be put together. Packing delivers a chart-like structure that records all completely assembled analyses, and does not require further constraint satisfaction checks to read out or count individual analyses. Underspecification is a form of procrastination: work is not done on evaluating constraints until it is necessary, in the expectation that for many of the constraints it will never become necessary. Packing computes everything, whether it needs to or not, on the basis that it is often easier to do everything at once than it is to carefully distinguish what needs to be computed now from what can be left until later. While underspecification in semantics has attracted a lot of attention, packing has been relatively and unjustly neglected.

One point that should be repeated about the difference is that it is trivial to count how many analyses one has in a packed representation whereas this is not the case with an underspecified representation because one needs to calculate how many solutions a set of constraints has. As mentioned at the end of section 4.2.1, the ability to efficiently count the number of solutions is important for efficient stochastic disambiguation of packed structures.

Syntactic Ambiguity: All previous frameworks for underspecified representations dealt only with semantic ambiguities, and in fact mostly or only with scope ambiguity of quantifiers. Although it is not hard to imagine how a ULF can capture syntactic ambiguities (in a similar way to Figure 1.4), it is not immediately clear how these can be calculated efficiently given the packed forest output of a chart parser. To the best of my knowledge, the work in this dissertation is the first to show how to carry over syntactic ambiguities into the (packed) semantic representation, and how to do this efficiently.

7.8 Summary

By now, I have answered the first two questions of the abstract. I showed how we can use the flexible framework of glue semantics to specify how syntactic structures can be mapped to exact meaning representations, and how this is simpler and more principled than traditional approaches as well as frameworks for underspecified representations. I also showed how to compute all the meaning representations for a sentence, and unlike existing research on underspecified representations, I took into account not only semantic ambiguities but also syntactic ambiguities. The computation is done efficiently in the sense that it takes as input a packed syntactic analysis, and pushes the packing through the semantics stage to obtain a packed meaning representation.

Did we achieve the maximal possible packing? Is no further compression possible? By and large, the answer is yes. If a piece of meaning representation is shared by more than one analysis, then the algorithm calculates it and represents it just once.[7] If that piece is obtained by combining several other pieces, then the computation of that combination is also done just once.

The only issue remaining is that theoretically, we may get an explosion when calculating

[7]Up to the issues discussed under "further reductions" in section 7.5.1.

the permutations of modifiers inside a basic cycle. A question remains: is there a better solution *that would work in all cases* than calculating all possibilities? Nevertheless, I explained in section 6.5.6 that in practice, the explosion is usually small and can be further reduced by interleaving various techniques, and so it does not usually cause a problem.

Part III

Extending the Linguistic Coverage

Introduction to Part III

In Part II, I have answered the first two questions of the abstract, and have seen how meaning representations can be computed compactly and efficiently from NL sentences. The coverage of semantic phenomena shown so far is non-trivial but still quite limited. In Part III, I turn to the third question mentioned in the abstract, and investigate how the coverage of semantic analysis can be extended to more complex linguistic constructions.

Extending the coverage is not a trivial exercise. There is no one source of difficulties; each semantic phenomenon poses its own challenges. Perhaps we can say, though, that a lot of the difficulties arise from a mismatch between the required meaning representations and the material that is explicit in the text, even if we consider only the text's literal meaning. There are aspects of the meaning that go beyond what we could reasonably expect the semantic composition module to do.

This led some researchers in NLP to abandon exact meaning representations altogether. However, anaphora, comparatives, and reciprocals are ubiquitous in exact NLU texts, and representing their truth conditions correctly is crucial for these applications, so it is very important to tackle the obstacles.

In this part of the dissertation I therefore show how to extend the semantic coverage to these complex semantic constructions. One thing that will help us with this task, in comparison to previous work, is the power we get from combining XLE's syntactic module (which uses a broad-coverage yet linguistically-accurate grammar) with Glue Semantics (the flexible framework for specifying the syntax-semantics interface, together with the algorithms developed in previous chapters). Thus, we will see in chapter 13 that we can account for reciprocals in a natural way within the glue semantics framework.

In addition, we need to carefully investigate what should be done by the semantic composition module and what should be left to other modules. We start in chapter 8 with anaphora. Although many people have investigated this topic in the past, I need to deal with it nonetheless because it is ubiquitous in NL, and later chapters will depend on it. Also, there have been some work in the glue semantics literature that suggests doing anaphora resolution within glue derivations. I discuss why this approach is trying to do too much in the semantic composition module, and show an alternative solution for combining anaphora resolution with glue semantics.

In chapters 10, 11, and 14, I tackle numerical and gradable comparative as well as

same and *different* and cases of covert reciprocals. Comparatives are ubiquitous in exact NLU texts, but there are very few existing computational implementations of comparatives. There are no implementations that I know of that cover as many cases of all three kinds as I do here in a systematic and uniform way. One of the reasons is that comparative constructions like to interact with ellipsis, so it is hard to get the complete meaning representations from what is explicitly given in the sentence. Moreover, it is not always clear what should be done by the independently-motivated ellipsis module and what should be done by ordinary semantic composition. I show how to carefully delineate the borderline between the two modules. This allows the semantic composition to be simpler and not to try to copy abilities that are also needed in the ellipsis module. The conclusions I reach are made possible by examining a broad range of cases rather than just a few examples as in some previous work. As I explained at the end of section 1.3.4, examining only a few cases may lead us to develop an analysis that does not generalize to unseen cases.

Chapter 8

Anaphora

Anaphoric expressions are ubiquitous in any NL texts, and so it is important to mention them as part of the semantic coverage. Here is an example from a logic puzzle text that talks about farmers planting various kinds of crops:

(348) If a farmer plants kale one year, *the farmer* does not plant *it* the *next year*.

Here, *the farmer* refers to the same farmer introduced by *a farmer*; *it* refers to *kale*, and although *next year* does not directly refer to the year introduced by *one year*, it makes use of that entity to refer to the year after it.

This chapter explains how anaphora representation and resolution can interact with glue semantics. I first review proposals for doing anaphora resolution within glue derivations, and show that this idea is trying to do too much in the semantic composition module. I then show another proposed solution where the meaning language is taken to consist of DRSs. This seems to require us to revise all our previous work on the glue specification so that it would use these DRSs. I improve this proposal by showing how our previous specification can be retained with only minor modifications and additions. This is achieved by treating the resulting meaning forms as a shorthand for the more elaborate DRSs.

8.1 Overview of Anaphora

I review here background material that is relevant to this dissertation. Anaphora occurs when one expression (the *anaphoric* expression) refers to another entity (the *antecedent*) that was mentioned previously in the text. I will only treat here explicit anaphora, where

the antecedent is explicitly mentioned in the text rather than inferrable from it based on world knowledge, as in:

(349) John went to a restaurant. *The waiter* asked him what he wants to eat.

I will also not treat here *cataphora* where the antecedent appears after the anaphoric expression, as in:

(350) Seeing *his*$_i$ face in the mirror, John$_i$ was shocked.

8.1.1 Constraints on Anaphora

There are several kinds of constraints on the possible antecedents of an anaphoric expression:

Linear Order: In this dissertation, I ignore caratphora, so all anaphoric expressions must appear after their antecedents in the text.

Gender and Number: The grammatical features of gender and number usually must agree between a pronoun and its antecedent:

(351) John$_i$ told Mary$_j$ that he$_{i/*j}$ / she$_{*i/j}$ is smart.

(352) a. John$_i$ likes himself$_i$ / *themselves$_i$.
b. [John and Bill]$_i$ think that Mary likes *him$_i$ / them$_i$.

There are some exceptions for a plural pronoun. A singular quantified NP, such as *every person* or *no person*, is grammatically singular, as (353)a shows, and so a pronoun anaphoric to it is supposed to be singular, as in (353)b. In casual language, however, one sometimes uses a plural pronoun in order not to commit to the gender of the quantified element, as in (354).

(353) a. Every person likes/*like eating cake.
b. Every person$_i$ thinks that he$_i$ is smart.

(354) a. I also know that nearly every$_i$ person running thinks that they$_i$ will win.[1]
b. Every$_i$ smart marketer thinks that they$_i$ know how to improve conversion rate.[2]

[1] www.mydd.com/story/2005/8/3/143252/6381 10-apr-2006
[2] www.offermatica.com/press-1.1.3.html 10-apr-2006

If all persons or marketers were known to be male (or female) then "he" (respectively, "she") could be used instead of "they". (In contrast, in "[John and Bill]$_i$ think they$_i$ like each other", we cannot use "he" instead of "they" because of syntactic agreement requirements.)

To summarize the constraints:

(355) Grammatical Constraints:

A pronoun must agree in number and (if specified) gender with its antecedent.

(Except, if we wish to allow it, that a plural pronoun may be anaphoric to a singular quantified NP.)

Binding Theory: This is a branch of linguistic syntax that investigates constraints on the possible antecedents of pronouns within a sentence. It accounts for differences such as:

(356) a. Bill$_i$ saw himself$_i$.

b. * Bill$_i$ said that John saw himself$_i$.

(357) a. * Bill$_i$ saw him$_i$.

b. Bill$_i$ said that John saw him$_i$.

We can roughly summarize these constraints as:

(358) Binding Theory Constraints:

1. A reflexive pronoun must be anaphoric to an NP in the pronoun's minimal clause.
2. A nonreflexive pronoun may not co-refer with any NP in the pronoun's minimal clause.

Note that the second constraint does not merely say that a nonreflexive pronoun may not be anaphoric to any NP in the pronoun's minimal clause – that is a weaker constraint. To see the difference, consider:

(359) Mary$_i$ looked in the mirror. John said that Mary saw her$_i$ in the mirror.

The weaker constraint would prevent *her* from being anaphoric to *Mary* in the second sentence, but not in the first. Yet the anaphoric link in (359) is very unnatural and should not be allowed.

Binding theory is more complex than (358), but this will do for now.

DRT Accessibility Constraints: We will see these in (388) below.

Centering Theory: There are other parameters such as recency and the grammatical function of NPs that influence the likelihood of NPs being the antecedent of anaphoric expressions. See for example Centering Theory (Grosz et al., 1995). In this dissertation, since I will hardly ever have to deal with inter-sentential anaphora, I will ignore such factors.

8.2 Anaphora Resolution Using Resource Management

8.2.1 The Basic Issue

Consider the following examples:

(360) a. John$_i$ likes himself$_i$.
$like(john, john)$
b. Bill$_i$ said that John saw him$_i$.
$said(bill, saw(john, bill))$

The Glue Semantics analysis of such bound anaphora as presented in (Dalrymple et al., 1999) assumes that the GS statement of a bound pronoun such as *himself* consumes the resource of its antecedent and supplies it as the meaning of the pronoun. I present the analysis in two stages. First, the basic glue statement for a pronoun is:

(361) pronoun: $\lambda x.x : a^e \rightarrow l^e$ where l is the label of my NP and a is the label of my antecedent

Thus, we have:

(362) [[John]$_2$ likes [himself]$_3$]$_1$
$john : \mathbf{2}^e$
$like : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$
$\lambda x.x : \mathbf{a}^e \rightarrow \mathbf{3}^e$ where $\mathbf{a}^e = \mathbf{2}^e$

If we apply EXCH on *like* and then compose that with the statement for "himself", we get:

(363) $\lambda y \lambda x.like(x, y) : \mathbf{2}^e \rightarrow \mathbf{2}^e \rightarrow \mathbf{1}^t$

Essentially, the statement in (363) wants to consume "John" twice: once normally as the subject, and once as the object through the pronoun. But as we have seen in section 3.4.2, we require each statement to be used exactly once.

8.2.2 Several Proposals

One idea to deal with this problem is to add the rule:

(364) from $\psi : A \to A \to B$
infer $\lambda x.\psi(x, x) : A \to B$

As we saw in section 3.4.2, this is a theorem of classical logic but not of linear logic. If we add this rule, it will considerably complicate the algorithm that searches for a proof. In particular, the indices of certain premises would be allowed to be repeated in the index sets of histories. Instead of complicating the algorithm, there are ways of resolving the resource conflict by writing appropriate glue semantic statements.

One such solution, presented in (Dalrymple et al., 1999), is to modify the contribution of the pronoun as follows:

(365) pronoun: $\lambda x.\langle x, x\rangle : a^e \to (a^e \otimes l^e)$ where l is the label of my NP and a is the label of my antecedent

The expression $\langle x, x\rangle$ is a tuple of two elements. The connective $\otimes$ is linear logic's resource-sensitive conjunction. The idea behind this is that since the pronoun knows it is consuming the resource a^e that someone else also wants, it is making a copy of this resource so that there will not be a resource deficit. The derivation would then be:

(366)

a.	$john : \mathbf{2}^e$	
b.	$like : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t$	
c.	$\lambda x.\langle x, x\rangle : \mathbf{2}^e \to (\mathbf{2}^e \otimes \mathbf{3}^e)$	
d.	$\langle john, john\rangle : \mathbf{2}^e \otimes \mathbf{3}^e$	(from a+c)
e.	$\lambda\langle x, y\rangle.like(x, y) : (\mathbf{2}^e \otimes \mathbf{3}^e) \to \mathbf{1}^t$	(equivalent to b in linear logic)
f.	$like(john, john) : \mathbf{1}^t$	(from e+d)

However, using tuples in the semantics and linear logic conjunction complicates the framework and its computational implementation. At least in the cases reviewed here, we can use a different solution:

(367) pronoun:

$\lambda x.x : a^e \rightarrow l^e$

$\lambda P \lambda x.P(x,x) : (a^e \rightarrow a^e \rightarrow H^t) \rightarrow (a^e \rightarrow H^t)$

where l is the label of my NP and a is the label of my antecedent

In this solution, a pronoun contributes two glue statements. The first says that the meaning of a pronoun simply takes the meaning of its antecedent and returns it. But we know that there must be some other predicate that also wants to consume that resource. So this means there will be a resource deficit, i.e. we know that eventually there will be some predicate, of category $a^e \rightarrow a^e \rightarrow H^t$, that wants to consume this a^e resource twice.[3] This is where the second statement kicks in. It allows, just one time, to consume this resource twice. Essentially, the second sentence is a one-time-use of the rule (364). (It is also an example of a manager resource, or resource management – see (Asudeh et al., 2002; Asudeh, 2004, 2005)). Now the derivation looks like this:

(368)

a.	$john : \mathbf{2}^e$	
b.	$like : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$	
c.	$\lambda z.z : \mathbf{2}^e \rightarrow \mathbf{3}^e$	
d.	$\lambda P \lambda x.P(x,x) : (\mathbf{2}^e \rightarrow \mathbf{2}^e \rightarrow H^t) \rightarrow \mathbf{2}^e \rightarrow H^t$	
e.	$\lambda y \lambda x.like(x,y) : \mathbf{3}^e \rightarrow \mathbf{2}^e \rightarrow \mathbf{1}^t$	EXCH on b
f.	$\lambda z \lambda x.like(x,z) : \mathbf{2}^e \rightarrow \mathbf{2}^e \rightarrow \mathbf{1}^t$	COMP on e+c
g.	$\lambda x.like(x,x) : \mathbf{2}^e \rightarrow \mathbf{1}^t$	APP on d+f (with $H^t = \mathbf{1}^t$)
h.	$like(john, john) : \mathbf{1}^t$	APP on g+a

Thus, the second line of (367) acts as a relation reducer, turning (368)f into (368)g.[4]

This also works when a quantifier is involved:

[3] H^t is a variable category, just as was used in section 3.3.2. Below I will modify this.

[4] This derivation actually fails in the implementation, but for a small technical reason that is easily fixed. The culprit is the definition of optimization of higher-order modifiers in section 6.4.4. This definition causes premise d in (368) to be compiled to

(369) $(\mathbf{2}^e \rightarrow H^t)_{[j]} \rightarrow \mathbf{2}^e \rightarrow H^t$

where j is a new index for category $\mathbf{2}^e$. The optimization does not compile the inner $\mathbf{2}^e \rightarrow H^t$ further because (369) has the shape $X \rightarrow X$. A solution is to use the optimization on $X \rightarrow X$ only when the first X was not obtained by simplifying a larger category. Another solution is to define the order of arguments in (367) as $\lambda x \lambda P$ rather than $\lambda P \lambda x$. We want this anyway for the solution of Problem 3 below.

(370) [[Every man]$_2$ likes [himself]$_3$]$_1$

a. $every : (\mathbf{2}_v^e \to \mathbf{2}_r^t) \to (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{1}^t$

b. $man : \mathbf{2}_v^e \to \mathbf{2}_r^t$

c. $like : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t$

d. $\lambda x.x : \mathbf{2}^e \to \mathbf{3}^e$

e. $\lambda P \lambda x.P(x,x) : (\mathbf{2}^e \to \mathbf{2}^e \to H^t) \to \mathbf{2}^e \to H^t$

f. $every(man) : (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{1}^t$ a+b

g. $\lambda x.like(x,x) : \mathbf{2}^e \to \mathbf{1}^t$ like above

h. $every(man)(\lambda x.like(x,x)) : \mathbf{1}^t$ f+g

8.2.3 Problem With Duplicate Derivations

There are several problems with this proposal. The first three can be rectified, but not the latter three.

Problem 1: Disconnected Contribution

One problem with this solution is that it allows the result to be derived in two non-equivalent ways, i.e. there are two nonequivalent $\beta\eta$-normal derivations. Essentially, if we let $connector = \lambda P \lambda x.P(x,x)$, then the two derivations correspond to: $connector(\lambda x \lambda y.like(x,y))(john)$ and $connector(\lambda y \lambda x.like(x,y))(john)$ (the latter can be paraphrased as "John is liked by himself"). Both proofs share the following sub-proof:

(371)
$$\dfrac{\dfrac{y : [\mathbf{3}^e]_2 \quad \dfrac{x : [\mathbf{2}^e]_1 \quad like : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t}{like(x) : \mathbf{3}^e \to \mathbf{1}^t}\,\text{APP}}{like(x,y) : \mathbf{1}^t}\,\text{APP}}{\dfrac{\lambda y.like(x,y) : \mathbf{3}^e \to \mathbf{1}^t \qquad \dfrac{w : [\mathbf{2}^e]_3 \quad \lambda z.z : \mathbf{2}^e \to \mathbf{3}^e}{w : \mathbf{3}^e}\,\text{APP}}{like(x,w) : \mathbf{1}^t}\,\text{APP}}$$

Now one may abstract on either of the $\mathbf{2}^e$ assumptions (i.e. either assumption 1 or assumption 3).[5]

[5] The implementation as described so far actually produces only one of the proofs in this particular case if we use the event-based represented for $like(x,y)$, namely $\exists e.like(e) \wedge subj(e,x) \wedge obj(e,y)$. The reason is that the *subj* and the *obj* are added to *like* by an event modifier that is marked as nonscoping in section 4.4.6. Each of these two modifiers will arrive in the SCC of $\mathbf{0}_v^e \to \mathbf{0}_r^t$ with three versions, depending on which version of $\mathbf{2}^e$ was used to produce the modifier (i.e. the $\mathbf{2}^e$ from premise *a* in (368) or one of the two $\mathbf{2}^e$ that was complied out of premise *d* in (368)). Each of the three *subj* modifiers may combine with two of the *obj* modifiers (namely those that did not use the same $\mathbf{2}^e$ instance). The order of those variables is important, and so we should get 6 final results. However, according to the implementation in section 6.7.1, we will get only three, because the basis for dropping a history in the SCC is that it has an identical span with another history.

One possible solution to this problem is to modify the definition of a proof to require that if two assumptions have the same category (e.g. $\mathbf{2}^e$ here), then they must be abstracted away in lexicographic order. But this looks like a hack and it may not work for some complicated cases. A more elegant solution is to rethink the contribution of a pronoun as follows:

(372) pronoun:
$\lambda P \lambda x.P(x,x) : (a^e \to l^e \to H^t) \to a^e \to H^t$
where l is the label of my NP and a is the label of my antecedent

The intuition behind this statement is that the pronoun knows that if you keep on the side both the pronoun l^e and its antecedent a^e, then some predicate P will eventually be formed that will want to consume both of them. In that case, this predicate can be converted to a predicate $\lambda x.P(x,x)$ that applies on the antecedent and "duplicates" it into the right place. So now we get the derivation:

(373)
a. $john : \mathbf{2}^e$
b. $like : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t$
c. $\lambda P \lambda x.P(x,x) : (\mathbf{2}^e \to \mathbf{3}^e \to H^t) \to (\mathbf{2}^e \to H^t)$
d. $\lambda x.like(x,x) : \mathbf{2}^e \to \mathbf{1}^t$ APP on c+b (with $H^t = \mathbf{1}^t$)
e. $like(john, john) : \mathbf{1}^t$ APP on d+a

Now there is just one normal proof.

Problem 2: Floating Too High or Too Low

There is another cause for redundant derivations.[6] In:

(374) [Mary knows that [[John]$_3$ thinks [[he]$_5$ snores]$_4$]$_2$]$_1$

To get the 6 results, we would have to refine the implementation to compare histories (in the non-scoping modifiers phase of a SCC processing) not based on spans but based on whether they used the same set of non-scoping modifiers. Usually the two notions are the same, but here not. We could number modifiers in a SCC and then drop a history if another history used the same modifiers (not just the same span). However, we do not need to do this in practice because we wouldn't want to get two derivations here anyway, and because we eventually abandon in this chapter the idea of using glue semantics for resolving bound anaphora.

Even though the implementation does not produce two final derivations, having two non-equivalent normal forms is still a problem because it produces a lot of redundant work for the algorithm.

[6] This was pointed out to me by Mary Dalrymple.

the H^t in the contribution of "he" can be instantiated to either $\mathbf{2}^t$ or $\mathbf{1}^t$, but both options would produce the same final result. In the latter case, we would get a sub-derivation of (375)a, which would then combine with (375)b-c to form (375)d.

(375) a. $\lambda x \lambda y.knows(mary, thinks(x, snore(y))) : \mathbf{3}^e \rightarrow \mathbf{5}^e \rightarrow \mathbf{1}^t$

b. $\lambda P \lambda x.P(x, x) : (\mathbf{3}^e \rightarrow \mathbf{5}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$

c. $john : \mathbf{3}^e$

d. $\Rightarrow knows(mary, thinks(john, snore(john))) : \mathbf{1}^t$

(In fact, H^t could also be instantiated as $\mathbf{4}^t$, but this cannot lead to a complete derivation. To see this, note that $\mathbf{4}^t$ appears only in four places:

(376) a. $\lambda x.snore(x) : \mathbf{5}^e \rightarrow \mathbf{4}^t$

b. $\lambda P \lambda x.P(x, x) : (\mathbf{3}^e \rightarrow \mathbf{5}^e \rightarrow \mathbf{4}^t) \rightarrow \mathbf{3}^e \rightarrow \mathbf{4}^t$

c. $\lambda x \lambda y.think(x, y) : \mathbf{3}^e \rightarrow \mathbf{4}^t \rightarrow \mathbf{2}^t$

The category $\mathbf{4}^t$ appears with positive polarity twice and with negative polarity twice. However, the positive occurrence at the end of (376)b cannot connect with the negative occurrence in the middle of (376)b (that would be circular) and must connect with the $\mathbf{4}^t$ in (376)c. But then there is no way to get $(\mathbf{3}^e \rightarrow \mathbf{5}^e \rightarrow \mathbf{4}^t)$ in (376)b from (376)a.)

This is related to the problem that was noted in section 3.3.2 regarding the use of variable categories in the analysis of quantifiers, and we can resolve it in the same way by replacing the variable category with a constrained category:

(377) pronoun:

$\lambda P \lambda x.P(x, x) : (a^e \rightarrow l^e \rightarrow k^t) \rightarrow a^e \rightarrow k^t$

where l is the label of my NP, a is the label of my antecedent,

and k is the label of the *minimal clause* containing a

The label of the scope is treated as a normal category (just as the label a is), and the constraint on it uniquely identifies one value. The derivation (375)a is now prohibited because the category of the minimal clause that contains the antecedent $\mathbf{3}^e$ is $\mathbf{2}^t$ and not $\mathbf{1}^t$.

Problem 3: Irrelevant Scope Ambiguity

This revision still leaves us with a redundant derivation in the following case:[7]

(378) [[John]$_2$ probably fooled [himself]$_3$]$_1$.

a. $john : \mathbf{2}^e$

b. $probably : \mathbf{1}^t \rightarrow \mathbf{1}^t$

c. $fool : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$

d. $\lambda P \lambda x.P(x,x) : (\mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{2}^e \rightarrow \mathbf{1}^t$

The problem here is that we get two derivations: there is an irrelevant scope ambiguity regarding what modifies the verb first: *probably* or d+a.

This can be solved by marking the contribution of a pronoun in (377) non-scoping, as we did for verb modifiers in section 4.4.6. However, for that to work, the statement needs to be a modifier of k^t. So we need to change (377) to:

(379) pronoun:

$\lambda x \lambda P.P(x,x) : a^e \rightarrow (a^e \rightarrow l^e \rightarrow k^t) \rightarrow k^t : [noscp]$

where l is the label of my NP, a is the label of my antecedent,

and k is the label of the *minimal clause* containing a

Problem 4: Two Equal Pronouns

Despite all these fixes, we still cannot get rid of all redundant derivations. First, we still get two normal proofs for:

(380) John thinks that [he snores and he drinks].

where both occurrences of "he" are anaphoric to "John". This is because the normal proof first forms the term $think(x_1, and(snore(x_2), drink(x_3))) : \mathbf{1}^t$, and then there are two options: either to abstract on x_1 and x_2 and apply the first "he" on this, then abstract on x_3 and apply the second "he" on the result; or first abstract on x_1 and x_3, apply the second "he", then abstract on x_2 and apply the first "he".[8]

[7] This was pointed out to me by Avery Andrews.

[8] If we use (379) then we get only one derivation in the implementation, for a similar reason to the one described in footnote 5. Still, having two different normal derivations creates redundant work for the algorithm.

The "variable-free semantics" of Jacobson (1999) also has this problem. She would assign two variable free terms to "he snores and he drinks": either $\lambda x_2 \lambda x_3.and(snore(x_2), drink(x_3))$ or with the reverse order of variables. They both get the category of a sentence that is missing two NPs, but with two possible orders for the NPs: one in which the first NP should be bound first by some larger context, and one in which the second NP should be bound first. Thus she will get two different derivations that produce the same answer.

Problem 5: Equational Redundancy

Sentence (380) exhibits another kind of ambiguity that is inherent with anaphoric expressions: whether the second *he* is anaphoric to *John* or to *he*. It is the job of anaphora resolution computation to realize that the two options yield the same result. But it seems it would be better to leave this equational reasoning to a separate module rather than cause a multiplicity in glue derivations.

Problem 6: Inter-sentential and Unbound Anaphora

Even if we accept redundant derivations, the method here only treats intra-sentential anaphora, where one derivation knows about all possible antecedents of pronouns. It is unclear how glue derivations can address cases where an antecedent appears in a previous sentence. The same problem occurs in the "variable-free semantics" of Jacobson (1999). Her proposed solution to this is to say that if "John" appears in one sentence and "he" appears in the next sentence and is anaphoric to "John", then the meaning of the first sentence should really be obtained by abstracting "John" away, i.e. it should be $\lambda x.\varphi(x)$ where x stands for "John". It is only the pronoun in the second sentence, acting as a relation reducer, which is responsible for combining this predicate with "John". This solution seems unintuitive, and it is hard to see how it could work in practice with arbitrary distances between anaphor and antecedent. Furthermore, it is hard to see how this could handle the kind of anaphora appearing in "donkey sentences" (see section 8.3.4 below). We do not want to assume separate mechanisms for intra-sentential and bound anaphora on the one hand and inter-sentential and donkey anaphora on the other hand, if we do not have to.

To sum up, this discussion leads us to the conclusion that if we follow the kind of analysis

shown here, pronouns simply do not know enough about their context to be able to provide a meaning term and combinatorial constraints that would produce a *unique* derivation for each anaphoric linking. Furthermore, it is unclear how this method can account for unbound anaphora. There is also a modularity argument: We should not try to do too much with one mechanism. Anaphoric expressions are really contextual parameters, and their resolution is a different process from semantic composition. Therefore, glue semantics should only handle semantic composition while anaphora resolution should be done by a separate module.

The work of this section is not a total loss, however. First, it was important to try this direction even if it turned out we do not want to pursue it. It was suggested in the literature so we had to give it a chance. Moreover, in chapter 13 we will see that the analysis of reciprocals resembles the analysis of bound pronouns here, and we will be able to use there a couple of the insights from this section.

In the rest of this chapter, I will discuss how glue semantics can be combined with structures like DRSs and so all types of anaphora can be accounted for in a uniform way.

8.3 DRT and Pronouns

The simplest case is when a pronoun refers to a simple entity mentioned previously as in:

(381) a. John$_i$ likes *himself*$_i$.
b. John$_i$ arrived. *He*$_i$ was smiling.

We would eventually like to get the following semantic representations:

(382) a. $\exists e.like(e) \wedge subj(e, john) \wedge obj(e, john)$
b. $\exists e_1.arrive(e_1) \wedge subj(e_1, john) \wedge \exists e_2.smile(e_2) \wedge subj(e_2, john)$

Generally speaking, each sentence yields a semantic representation (or more than one if there is an ambiguity) that expresses the truth conditions of the sentence. But it also yields a list of entities that were mentioned – these are usually called *discourse entities*. In this sense, some expressions, such as proper names, have two "tracks" in which they contribute – the truth conditions track, and the discourse entities track.

8.3.1 DRSs

A common framework for calculating these is Discourse Representation Theory (DRT) (Kamp, 1981; Heim, 1982; Kamp and Reyle, 1993; van Eijck and Kamp, 1997), and its compositional λ-DRT versions (Bos et al., 1994). I will give here a very brief review; for more details, see these references.

Clauses are represented by Discourse Representation Structures (DRSs). Each DRS consists of two tracks: a set of discourse variables and a set of conditions. Thus, the initial representations for (381) are:

(383) a. $[x, y, e \mid name(x) = \text{`John'},\ male(y),\ y =?,\ like(e),\ subj(e, x),\ obj(e, y)]$

b. $[x, e_1 \mid name(x) = \text{`John'},\ arrive(e_1),\ subj(e_1, x)],$
$[y, e_2 \mid male(y),\ y =?,\ smile(e_2),\ subj(e_2, y)]$

The equations $y =?$ indicate that this discourse variable needs to be resolved, i.e. equated with some previous discourse variable.[9] In (383)a, this is done by $y = x$ since *himself* is anaphoric to *John*. In (383)b, the two DRSs first need to be unioned according to the following rule:

(384) Union of DRSs:
$[V_1|C_1] \oplus [V_2|C_2] = [V_1 \cup V_2|C_1 \cup C_2]$

to get

(385) $[x, e_1, y, e_2 \mid name(x) = \text{`John'},\ arrive(e_1),\ subj(e_1, x),$
$male(y),\ y =?,\ smile(e_2),\ subj(e_2, y)]$

and then $y = x$ can be added.

8.3.2 Quantifiers in DRSs

DRSs also capture the behavior of basic quantifiers as in:

(386) a. A man^i arrived. He$_i$ smiled.

b. * Every man^i arrived. He$_i$ smiled.

c. Every man^i likes himself$_i$.

[9] A discourse variable arising from a plural pronoun may be equated with a collection of previous discourse variables.

d. Every man^i thinks that he$_i$ is smart.

e. Every man^i thinks that he$_i^j$ likes himself$_j$.

Ignoring event variables for the moment, these are represented by:

(387) a. $[l \mid man(l),\ arrive(l)] \oplus [l_2\ male(l_2),\ l_2 =?,\ smile(l_2)]$

b. $[\ \mid [l \mid man(l)] \Rightarrow [\ \mid arrive(l)]]\ \oplus\ [l_2\ male(l_2),\ l_2 =?,\ smile(l_2)]$

c. $[\ \mid [l \mid man(l)] \Rightarrow [l_2 \mid male(l_2),\ l_2 =?,\ likes(l, l_2)]]$

d. $[\ \mid [l \mid man(l)] \Rightarrow [\ \mid think(l, [l_2 \mid male(l_2),\ l_2 =?,\ smart(l_2)])]]$

e. $[\ \mid [l \mid man(l)] \Rightarrow [\ \mid think(l, [l_2, l_3 \mid male(l_2),\ l_2 =?,\ male(l_3),\ l_3 =?,\ likes(l_2, l_3)])]]$

An existential quantifier simply introduces a discourse variable with constraints on it, as in the representation (387)a for (386)a. Once the two DRSs from the two sentences are merged, l_2 can be equated with l. In contrast, a universal quantifier introduces two embedded DRSs. According to the accessibility rules in DRSs, a discourse variable l_1 can be equated with another variable l_2 only if the DRS where l_2 is introduced is accessible from the DRS where l_1 is introduced. Accessibility is defined by:

(388) A DRS_2 is accessible from a DRS_1 iff

(a) DRS_2 contains DRS_1 (any DRS contains itself), or

(b) The condition $DRS_2 \Rightarrow DRS_3$ appears in some DRS, and DRS_3 contains DRS_1

Thus, l_2 may be equated with l in (387)c,d but not in (387)b. Furthermore, in (387)e, l_3 can be equated with l_2, which is equated with l.

Similar cases are:

(389) a. * No man^i arrived. He$_i$ smiled.

b. No man^i likes himself$_i$.

with the representations:

(390) a. $[\ \mid \neg[l \mid man(l),\ arrive(l)]] \oplus [l_2\ male(l_2),\ l_2 =?,\ smile(l_2)]$

b. $[\ \mid \neg([l \mid man(l)] \oplus [l_2 \mid male(l_2),\ l_2 =?,\ likes(l, l_2)])]$

8.3.3 Meaning of DRSs

What do DRSs mean? There are two views. One view treats a DRS as a convenient shorthand for a FOL formula. A DRS can be expanded to the FOL formula it represents using the following definition:

(391) Translation of DRSs to FOL:

$tr([l_1 \ldots l_n|C] \Rightarrow D) \quad = \quad \forall l_1, \ldots, \forall l_n.(tr(C) \rightarrow tr(D))$

$tr([l_1 \ldots l_n|C]) \quad = \quad \exists l_1, \ldots, \exists l_n.tr(C_1)$

$tr(\{\varphi_1, \ldots, \varphi_n\}) \quad = \quad tr(\varphi_1) \wedge \ldots \wedge tr(\varphi_n)$

$tr(R(t_1, \ldots, t_n)) \quad = \quad R(t_1, \ldots, t_n)$

$tr(t_1 = t_2) \quad = \quad t_1 = t_2$

According to this, we can translate (383)a and (385) to:

(392) a. $\exists x \exists y \exists e.name(x) = \text{'John'} \wedge male(y) \wedge y = x \wedge like(e) \wedge subj(e, x) \wedge obj(e, y)$

b. $\exists x \exists e_1 \exists y \exists e_2.name(x) = \text{'John'} \wedge arrive(e_1) \wedge subj(e_1, x) \wedge male(y) \wedge y = x \wedge smile(e_2) \wedge subj(e_2, y)$

We can simplify (392) by using the following logical equivalence in FOL with equation:

(393) $\exists y.(y = t \wedge \varphi) \quad \Leftrightarrow \quad \varphi[t/y]$ provided y not free in t

We can also introduce a new constant for each individual:

(394) $name(x) = Y \leftrightarrow x = c_Y$ where c_Y is a constant whose name is Y

With these two simplifications, (392) ends up being the same as (382). Also, the translation of (387)a,c (after merging and resolving $l_2 = l$) is:

(395) a. $\exists l \exists l_2.man(l) \wedge arrive(l) \wedge male(l_2) \wedge l_2 = l \wedge smile(l_2) \quad \Leftrightarrow$
$\exists l.man(l) \wedge arrive(l) \wedge male(l) \wedge smile(l)$

b. $\forall l.man(l) \rightarrow \exists l_2.male(l_2) \wedge l_2 = l \wedge like(l, l_2) \quad \Leftrightarrow$
$\forall l.man(l) \rightarrow male(l) \wedge like(l, l)$

If (according to this view) DRSs are just a notational shorthand for FOL and add no expressive power, why are they needed? The reason is that they provide explicit representations of discourse variables that can be equated with one another, where the *shape* of the representations adds constraints on possible equations, according to the accessibility relation (388). The shape also makes the representations more amenable than FOL to compositional computation from the surface syntax of NL because they do not specify the range of quantifiers (only after the entire DRS is constructed, the range of the quantifiers is determined by the translation to FOL). The importance of the shape of semantic representations was already discussed in sections 2.6 and 2.8.1.

In this view, certain manipulations of formulas are defined, and the accessibility relation is defined based on the shape of formulas as well. However, in section 7.7.3 under the point "unprincipled," a lot of emphasis has been placed on the unprincipledness of manipulating formulas without this manipulation being justified by an underlying model-theoretic semantics. This issue has led several researchers to the second view of DRSs, where they *are* given a direct model-theoretic semantics (Barwise, 1987; Rooth, 1987; Groenendijk and Stokhof, 1990, 1991; Chierchia, 1995; Muskens, 1995, 1996; Kohlhase et al., 1996; van Eijck and Kamp, 1997). These approaches usually go by the name *dynamic semantics* because the truth conditions of a sentence are not just static per each sentence, but also change some (discourse) state after each sentence is processed. The state contains information about discourse variables such as restrictions on their values and whether they are bound. The approaches vary regarding whether the discourse variables and the state are part of the logical apparatus (at the level of "model" and "assignment" in FOL) or are part of the ontology (the domain of models).

We could extend the definition of our MRL from chapter 2 and incorporate such dynamic semantics into it. Then, we could try to devise a direct proof system for it (see section 2.8.1 again). For some work on proof systems for dynamic semantics, see (Veltman, 2000). Or, we could try to find a translation from the dynamic MRL to an implementational language L^{IMP}. That L^{IMP} would have to talk about and reason about discourse states and discourse variables. Perhaps this is required for the most general solution. However, for many purposes, especially when the anaphora is quite local, it suffices to use the first view of DRT (and rely on the second view as a more principled justification for the first view). In other words, we view the DRS-based representations as "pre-expressions" or underspecified expressions of the MRL, which are later resolved and converted to proper expressions of the MRL.

8.3.4 Donkey Anaphora

Before explaining how DRSs are calculated from the sentence, I need to point out that DRT can deal with a certain set of more tricky cases with so-called *donkey anaphora*, exemplified in the following classical cases:

(396) a. Every man who owns a donkey$_j$ beats it$_j$.

b. If a man$_i$ owns a donkey$_j$, he$_i$ beats it$_j$.

c. $\forall x \forall y.[man(x) \wedge donkey(y) \wedge own(x,y)] \rightarrow beat(x,y)$

One challenge is that what seems to be a quantifier '*a*' seems to bind *it* and *he* even though those pronouns appear outside the quantifier's range. In fact, this was the case in (386)a as well. A further challenge is that in (386)a, "a man" is translated to an existential quantifier, whereas in (396), the indefinites are translated to universal quantifiers.

These cases are treated in DRT as a result of treating indefinites as introducing a discourse variables (with restrictions), treating conditionals and universal quantifiers by using the operator $\Rightarrow$, and using the definition in (391). Thus, (396)b gets the representation (simplifying event variables here):

(397) $[\ |\ [x,y \mid man(x),\ donkey(y),\ own(x,y)] \Rightarrow$
$[z,w \mid male(z),\ non\text{-}human(w),\ z =?,\ w =?,\ beat(z,w)]]$

and then z is resolved to x, and w to y, so together with the definition (391) we get the truth conditions (396)c. Also (396)a gets a similar representation, except x is used directly instead of z.[10]

8.4 Calculating the Representations

8.4.1 Standard Ways

How are DRSs calculated from the syntactic analysis of a sentence? Bos et al. (1994) present λ-DRS, an extension of the syntax-to-semantics mappings to handle DRSs. Here is a simple λ-DRS lexicon:

(398)

John	$\lambda P.[l \mid name(l) = \text{`}John\text{'}] \oplus P(l)$
likes	$\lambda y \lambda x.[e \mid like(e),\ subj(e,x),\ obj(e,y)]$
book	$\lambda x.[\ |\ book(x)]$
a	$\lambda P \lambda Q.[l \mid\ ] \oplus P(l) \oplus Q(l)$
every	$\lambda P \lambda Q.[\ |\ ([l \mid\ ] \oplus P(l)) \Rightarrow Q(l)]$
that	$\lambda Q \lambda P \lambda x.P(x) \oplus Q(x)$ (relative clause)

Semantic pieces are combined by using an extended functional composition:

(399) $\phi \odot \psi\ :=\ \lambda \overrightarrow{\sigma}.\phi(\lambda v.(\psi(v)(\overrightarrow{\sigma})))$

[10] It is necessary to mention that DRT does not generalize well some cases of donkey anaphora involving numeric and proportional quantifiers (like those discussed in sections 10.2 and 10.9). See section 12.3.2.

This operator extends standard function composition $\lambda v.\phi(\psi(v))$ by accepting a n-place predicate ψ and binding only its first argument, while abstracting over the remaining $n-1$ arguments in $\overrightarrow{\sigma}$.

Example derivation:

(400) John read a book.

- a $\odot$ book =
  $(\lambda P\lambda Q.([l \mid] \oplus P(l) \oplus Q(l)))(\lambda x.[\mid book(x)]) =$
  $\lambda Q.([l \mid] \oplus (\lambda x.[\mid book(x)])(l) \oplus Q(l)) =$
  $\lambda Q.([l \mid] \oplus [\mid book(l)] \oplus Q(l)) =$
  $\lambda Q.([l \mid book(l)] \oplus Q(l))$
- (a $\odot$ book) $\odot$ read =
  $(\lambda Q.([l \mid book(l)] \oplus Q(l)))(\lambda y\lambda x.[e \mid read(e),\ subj(e,x),\ obj(e,y)]) =$
  $\lambda x.(\lambda Q.([l \mid book(l)] \oplus Q(l)))(\lambda y.[e \mid read(e),\ subj(e,x),\ obj(e,y)]) =$
  $\lambda x.([l \mid book(l)] \oplus (\lambda y.[e \mid read(e),\ subj(e,x),\ obj(e,y)])(l)) =$
  $\lambda x.([l \mid book(l)] \oplus [e \mid read(e),\ subj(e,x),\ obj(e,l)]) =$
  $\lambda x.([l,e \mid book(l),\ read(e),\ subj(e,x),\ obj(e,l)])$
- John $\odot$ ((a $\odot$ book) $\odot$ read) =
  $(\lambda P.[l_2 \mid name(l_2) = \text{`John'} \oplus P(l_2))(\lambda x.([l,e \mid book(l),\ read(e),\ subj(e,x),\ obj(e,l)])) =$
  $[l_2 \mid name(l_2) = \text{`John'}] \oplus (\lambda x.([l,e \mid book(l),\ read(e),\ subj(e,x),\ obj(e,l)]))(l_2) =$
  $[l_2 \mid name(l_2) = \text{`John'}] \oplus ([l,e \mid book(l),\ read(e),\ subj(e,l_2),\ obj(e,l)]) =$
  $[l,l_2,e \mid name(l_2) = \text{`John'},\ book(l),\ read(e),\ subj(e,l_2),\ obj(e,l)])$

This idea was later revised using hole-semantics (Reyle, 1993; Blackburn and Bos, 2005b) in order to deal with scope ambiguities. The entries are then similar to what we saw in section 7.7.2, except that λ-DRSs are used instead of the simpler meaning terms there.

8.4.2 Existing Suggestions for Glue-DRT

I have already discussed in sections 3.1.3, 3.5.1, and 7.7 the shortcomings of this kind of solution. Instead, we can use glue semantics for defining the composition because GS is compatible with different meaning representation languages, including DRSs (see section 3.2.4). This idea was first proposed in (van Genabith and Crouch, 1999; Kokkonidis, 2005), and here I present a variation of it.

In the λ-DRT lexicon, instead of relying on standard lambda-composition, we can add the usual glue specifications:

(401)

John	$\lambda P.[\hat{l} \mid name(\hat{l}) = \text{`John'}] \oplus P(\hat{l})$: $(l^e \rightarrow k^t) \rightarrow k^t$
	where l is the label of my NP and k is the label of the clause I scope at
likes	$\lambda x \lambda y.[e \mid like(e),\ subj(e,x),\ obj(e,y)]$: $a^e \rightarrow b^e \rightarrow l^h$
	where a is the label of my subject, b is the label of my object,
	and l is the label of my clause.
book	$\lambda x.[\ \mid book(x)]$: ${l_v}^e \rightarrow {l_r}^t$
	where l is the label of my NP
a	$\lambda P \lambda Q.[\hat{l} \mid\ ] \oplus P(\hat{l}) \oplus Q(\hat{l})$: $({l_v}^e \rightarrow {l_r}^t) \rightarrow (l^e \rightarrow k^t) \rightarrow k^t$
	where l is the label of my NP and k is the label of the clause I scope at
every	$\lambda P \lambda Q.[([\hat{l} \mid\ ] \oplus P(\hat{l})) \Rightarrow Q(\hat{l})]$: *ditto*
that	$\lambda Q \lambda P \lambda x.P(x) \oplus Q(x)$: $(a^e \rightarrow b^t) \rightarrow ({l_v}^e \rightarrow {l_r}^t) \rightarrow {l_v}^e \rightarrow {l_r}^t$ (relative clause)
	where l is the label of my NP, b is the label of my clause complement,
	and a is the level of the NP it is missing

The hat $\hat{}$ is a function that takes a basic glue label and returns a discourse variable. It allows us to link the two domains. Constraints on possible anaphoric links which are expressed in terms of glue categories can be used to constrain equations between discourse variables. Thus, if we have the fact $before(f, g)$ where f and g are glue categories of NPs such that the first NP appears before the second in the sentence, then a pronoun in the g NP could be anaphoric to the f NP, but not vice versa. And if we have the fact *not-antecedent*(f, g) that arises from binding theory on the F-structure, it induces the constraint that an equation $\hat{g} =?$ cannot be resolved with $\hat{g} = \hat{f}$.

8.4.3 My Version

While this solution works, it is very tedious to require writing DRS structures and $\oplus$ operators for every entry – many entries have nothing to do with anaphora (see e.g. *book* and *that*). This is like using the L^{IMP} of FOL directly as the MRL instead of first translating to an independent L^{MR} (see section 2.6 again). Also notice the treatment of proper names as scoping quantifiers (recall the discussion about this from section 3.1.3). It would also be nice if we did not have to revise all the glue entries that we have already developed in previous chapters. What we really want is to keep what we have done, and change only the parts that have something to do with anaphora. For example, we do not

want to change our treatment of nouns and relative clauses. Even for verbs, if we decide for now not to handle expressions that are anaphoric to event variables, we can keep our old specifications for verbs.

My aim is that at the end of the glue composition phrase, we will end up with a representation that looks something like this:

(402) Every man that saw Mary liked her.

$$every(\lambda x.man(x) \wedge \exists e_1.see(e_1) \wedge subj(e_1, x) \wedge obj(e_1, entity(l_1, \{name(l_1) = \text{`Mary'}, female(l_1), individual(l_1)\})), \lambda x.\exists e_2.like(e_2) \wedge subj(e_2, x) \wedge obj(e_2, dref(l_2, \{individual(l_2), female(l_2)\})))$$

Notice that overall the representation looks like what we had until this chapter, and only certain pieces mention discourse variables. The operators *entity* and $dref$ could be seen as shorthands as follows:

(403) $entity(v, S) \equiv [\langle v, S\rangle \mid v]$

$dref(v, S) \equiv [\langle v, (\{v =?\} \cup S)\rangle \mid v]$

In these definitions, I use a slightly different version of DRSs. Instead of a pair consisting of a set of discourse variables and a set of conditions, I have a pair whose second element is just one meaning expression and whose first element is a set of *conditioned* discourse variable, where a conditioned discourse variables is a pair consisting of a discourse variable and a set of associated conditions on it. In (402), the discourse variable l_1 has three conditions associated with it, including $name(l_1) =$'*Mary*'. I use this form to make it easier to locate the constraints that are relevant for each discourse variable.

The *entity* and $dref$ expressions act locally as a v in the expression where they appear. The discourse variable introduced by an *entity* expression should be accessible everywhere, but the *entity* expression is written *in situ* rather than in a global place to make it easier to write the glue semantics specification (the information will be moved up by the anaphora resolution algorithm below). The information in a $dref$ expression should be used by the anaphora resolution module when the pronoun is resolved and the $dref$ expression is replaced by another variable.

In light of this, we change our glue rule for proper names as follows:

(404) proper name n of gender g:

$entity(\hat{l}, \{name(\hat{l}) = n,\ gender(\hat{l}) = g\})\ :\ l^e$

where l is the label of my NP

We also add for pronouns:

(405) pronoun with gender g and number n (sg or pl):
$dref(\hat{l}, \{gender(\hat{l}) = g,\ number(\hat{l}) = n\})\ :\ l^e$
where l is the label of my NP

And that's it. There is no need to make any more changes to the glue specification. All the rest of the work will be done by the anaphora resolution module, as described below. This includes treating quantifier expressions as a shorthand for the appropriate DRS structures. Thus, we get modularity: The specification of composition is simple, supplying just enough information needed to handle anaphora. Anaphora resolution using DRSs is done in a separate module, and the writer of the glue specification does not need to worry about using DRSs explicitly. The justification for this approach is that it produces the same results that would be obtained if we followed the more explicit glue-DRT approach above.

8.4.4 Calculating Constraints

The above definitions can be easily incorporated into the glue specification. In addition to calculating the representations, we can also use the rewriting system to calculate relevant constraints on anaphora resolution, expressed using glue categories.

We first calculate some auxiliary facts. Facts about the linear order of NPs are calculated as follows. Here, `np_place(%F,%Place)` means that `%Place` is the number of the morpheme that starts the position of the NP of `%F`. (The predicates `subtree`, `terminal`, etc. where explained in chapter 4.)

```
+subtree(%N,NP,%%,%%), +phi(%N,%F),
+terminal(%N2,%%,[%Place]), +phi(%N2,%F)
     ==> np_place(%F,%Place).

+np_place(%F1,%P1), +np_place(%F2,%P2), { %P1 < %P2 }
     ==> linear_order(%F1,%F2).
```

Facts about the immediately dominating clause are calculated using:

```
+im_dominated(%F,%DominC), is_clause(%DominC)
     ==> dominating_clause(%F,%DominC).
```

```
+im_dominated(%F,%P), is_pp(%P),
+im_dominated(%P,%DominC), is_clause(%DominC)
     ==> dominating_clause(%F,%DominC).
```

(The predicate `im_dominated` was defined in section 4.5.2.) Finally, we mark all NPs as possible antecedents:

```
is_np(%F) ==> possib_ant(%F).
```

Now, to prevent cataphora (pronoun anaphoric to something after it), we add:

```
+possib_ant(%A), base_pronoun(%F,%%,%%), +linear_order(%F,%A)
        ==> not_ant(%A,%F).
```

We can implement the two constraints from binding theory in (358) as follows. A reflexive pronoun cannot be anaphoric to NP not in its clause:

```
+possib_ant(%A), +dominating_clause(%A,%S),
base_pronoun(%F,sg,refl), -dominating_clause(%F,%S)
     ==> not_ant(%A,%F).
```

A nonreflexive pronoun cannot co-refer with any NP in its clause:

```
base_pronoun(%F,sg,pers), dominating_clause(%F,%S),
+possib_ant(%A), +dominating_clause(%A,%S), -({%F=%A})
     ==> not_eq(%F,%A).

+not_eq(%F,%A) ==> not_ant(%A,%F).
```

8.5 Resolving the Representations

Since the glue specification was left mostly unchanged, I need to define here how to expand some of the expressions, such as the quantifiers, and how to deal with *entity* and *dref* expressions.

8.5.1 Preparing

Entity Raising

The conditions inside an *entity* expression should be moved to the topmost DRS because there should be no DRS-accessibility restrictions on proper names serving as antecedents of anaphoric expressions.[11] Therefore, every $entity(l, S)$ in φ is replaced with l to obtain φ'. Additionally, for every such entity, the pair $\langle l, S\rangle$ is added to a collection C. Then, a top DRS is created:

(406) $[C \mid \varphi']$

For example,

(407) John saw Mary.

$$\exists e.see(e) \wedge subj(e, entity(l_1, \{name(l_1) = \text{'John'}, gender(l_1) = male, num(l_1) = sg\})) \wedge obj(e, entity(l_2, \{name(l_2) = \text{'Mary'}, gender(l_2) = female, num(l_2) = sg\}))$$

$$\Rightarrow$$

$$[\langle l_1, \{name(l_1) = \text{'John'}, gender(l_1) = male, num(l_1) = sg\}\rangle, \langle l_2, \{name(l_2) = \text{'Mary'}, gender(l_2) = female, num(l_2) = sg\}\rangle \mid \exists e.see(e) \wedge subj(e, l_1) \wedge obj(e, l_2)]$$

Quantifier Expansion

We treat quantifier expressions as shorthands, and expand them as follows:

(408) $tr(a(P, Q)) = [\langle l, \{tr(P(l))\}\rangle \mid tr(Q(l))]$

$tr(every(P, Q)) = [\ |\ (\ [\langle l, \{tr(P(l))\}\rangle \mid]\ \Rightarrow [\ |\ tr(Q(l))]\ )\ ]$

$tr(no(P, Q)) = [\ |\ \neg tr(a(P, Q))]$

Merge Embedded DRS

If the body of a DRS is itself a DRS, the two can be merged:

(409) $tr([V_1 \mid [V_2 \mid B]]) = tr([V_1 \cup V_2 \mid B])$

[11] This is not the most general treatment because in some sentences in NL, proper names may not be so accessible. For example: "The John from our street met the John that was elected as Mayor." However, this treatment will do for now.

8.5.2 Resolution of *dref*

After all these transformations are done, we are ready to resolve anaphora. We do this by recursively traversing top-down the DRS for the entire text. At each step, the possible antecedents of a pronoun are those that are possible according to the DRS accessibility relations. These are remembered in a second argument to the *resolve* predicate as follows:

(410) $\mathit{start\text{-}resolve}(\varphi) \;=\; \mathit{resolve}(\varphi, \{\})$
$\mathit{resolve}(\mathit{drs}(V, D), S) \;=\; \mathit{resolve}(D, V \cup S)$
$\mathit{resolve}((([V|B] \Rightarrow D), S) \;=\; \mathit{resolve}([V|B], S) \Rightarrow \mathit{resolve}(D, V \cup S)$

Notice that when we enter the right-hand-side of a DRS implication, we add the discourse variables of the implication's antecedent to the list of possible antecedents. This is in accordance with the definition of DRS accessibility in (388), and it allows us to resolve donkey anaphora correctly.

Furthermore, the algorithm takes into account the gender constraints, as they are written in the conditions associated with the discourse variables. We also take into account the `not_ant` and `not_eq` constraints from section 8.4.4. For example, $\mathit{not\text{-}ant}(g, h)$ prevents the algorithm from resolving a $\mathit{dref}(\hat{h}, \ldots)$ to $\hat{g}$.

There may be more than one possible resolution to a *dref* expression. In that case, we can find and return all possibilities. If a *dref* cannot be resolved to any discourse variable, then the particular interpretation that is processed is incorrect. For example, consider:

(411) a. Every man likes a woman. She is happy.
b. $\mathit{every}(\mathit{man}, \lambda x.\mathit{a}(\mathit{woman}, \lambda y.\mathit{like}(x, y))) \oplus \mathit{happy}(\mathit{dref}(l, \{\mathit{female}(l)\}))$
c. $\mathit{a}(\mathit{woman}, \lambda y.\mathit{every}(\mathit{man}, \lambda x.\mathit{like}(x, y))) \oplus \mathit{happy}(\mathit{dref}(l, \{\mathit{female}(l)\}))$

The first sentence in (411)a has a scope ambiguity, so the two possible resulting representations are (411)b,c. These are converted to the DRS representations:

(412) a. $[\ |\ ([\langle l_1, \{\mathit{man}(l_1)\}\rangle\ |\ ] \Rightarrow [\langle l_2, \{\mathit{woman}(l_2)\}\rangle\ |\ \mathit{like}(l_1, l_2)]),\ \mathit{happy}(\mathit{dref}(l, \{\mathit{female}(l)\}))]$
b. $[\langle l_2, \{\mathit{woman}(l_2)\}\rangle\ |\ ([\langle l_1, \{\mathit{man}(l_1)\}\rangle\ |\ ] \Rightarrow [\ |\ \mathit{like}(l_1, l_2)]),\ \mathit{happy}(\mathit{dref}(l, \{\mathit{female}(l)\}))]$

In the first representation, l_2 is not accessible to l, and as there is no other possibility, *dref* cannot be resolved. We therefore mark as "nogood" the choice-space context that gave rise to this interpretation. In the second representation, l_2 appears in the top DRS and so is accessible to l, so that representation can be resolved to:

(413) $[\langle l_2, \{woman(l_2), female(l)\}\rangle \mid ([\langle l_1, \{man(l_1)\}\rangle \mid] \Rightarrow [\mid like(l_1, l_2)]),\ happy(l_2)]$

The resolution of a definite expression $the(P)$ (and of possessive expressions) is similar to the resolution of $dref$ but more complex because testing whether a prior NP is a possible antecedent depends on more factors. We may need to do (automated) reasoning to check whether that NP satisfies P, and the reasoning may depend on additional knowledge. For example, if the NP is "a man" and the definite expression is "the human", the computer needs to know that every man is a human. Also, in some cases, the antecedent is not mentioned explicitly in the text and needs to be introduced via "accommodation." See (van der Sandt, 1992; Chierchia, 1995; Beaver, 1997; Blackburn and Bos, 2005b) among others.

8.5.3 Packed Version

The preparation and resolution algorithms above assume they get an unpacked meaning representation. They should be adaptable to an input consisting of a packed meaning representation (the output of the packed glue prover). Here is a sketch of how this could be done.

All calculations need to be be relativized to choice-space contexts. For example, the anaphora resolution procedure passes around not a simple list of possible antecedents but a contexted list, i.e. each member of it is relativized to a context. It also passes around the "current context". The *start-resolve* function starts with context 1, but if an ambiguity junction such as $\langle A1 : \varphi,\ A2 : \psi\rangle$ is encountered while the current context is C, then each of the options is followed separately, under $A1 \wedge C$ and $A2 \wedge C$. Recall that when we calculated packed meanings in section 6.5.2, the algorithm went top-down but stopped at each label (chunk), accumulated a context on it, and did not enter the label at that time (only later). Similarly, the resolution algorithm here needs to treat "diamonds" (labels that are accessible from above in more than one way) in a similar way, so as not to repeat the work on the material under the label. Once all those contexts for a label of a chunk are collected, they are disjoined to know in which context we are, and resolution continues downward. Also, if a *dref* may be resolved in more than one way, this creates a split in the choice space.

8.6 Summary

Now we have a way of calculating meaning representations that involve anaphoric expressions. I showed why the resolution of these expressions should not be done within the semantic composition module but by a separate module. I also showed how the specification can be written so that all our work thus far does not need to be revised to use DRSs throughout. Additional constructions in later chapters will also not need to worry about DRSs for the same reason (unless these constructions have some non-trivial interaction with anaphora). Further work is needed to extend the specification to additional anaphoric expressions, and extending the anaphora resolution algorithm. In chapter 12, I will address some issues involved with plural anaphora.

Chapter 9

Ellipsis

As mentioned in section 2.10.3, part of a sentence may not be expressed explicitly when it is very similar to another part of the sentence or discourse. Developing ways of parsing sentences that may be missing parts and then reconstructing the missing semantic material is a very complex issue. Therefore, in this dissertation I choose not to pursue an investigation of this topic, even though handling ellipsis is an important part of any precise understanding system, in particular solving logic puzzles. Whenever we encounter a sentence that includes ellipsis, I will just assume that some as-yet-undeveloped enhancement of existing system components takes care to supply us with the linguistic material as if it existed in the text explicitly.

Nevertheless, in chapters 10, 11, and 14, I will discuss various comparative constructions. In such constructions, it is easy to get confused about what can be calculated neatly in a compositional way (using glue semantics or other compositional frameworks) and what cannot be so calculated and must rely on additional ellipsis-reconstruction mechanisms. Conclusions one way or the other depend on knowing about independently-motivated cases of ellipsis, and so I review some of them here.

9.1 VP Ellipsis

The content of a verb-phrase may be dropped if it is similar enough to the content of another VP in the discourse, but the auxiliary *do* may be needed to indicate this drop. Here are some examples from logic puzzle texts:

(414) a. If Valdez goes to the soccer game then Olson does too.
b. If Valdez goes to the soccer game then Olson does so too.
c. If Valdez goes to the soccer game then so does Olson.
d. If Greta left any message, Fleure and Pasquale did also.

The missing VP content in the first sentence is "go to the soccer game". Words like *too*, *so*, *also* provide clues. They are needed in some cases, as in (414), and *either* is needed when both VPs are negative (415), but these words cannot be used when the polarity of the content is reversed (416).

(415) If Valdez does not go to the soccer game then Olson does not either.

(416) a. If Valdez goes to the soccer game then Olson does not (*too/*either).
b. If owls are not in the forest then sparrows are (*too/*either).

A special case of VP ellipsis is antecedent-contained ellipsis (ACE):

(417) a. John reads every book that Mary does.
b. Greyhounds are featured on every day that Himalayans are.
c. Rottweilers are featured on every day that Himalayans are not.

What is special about these is that the elided VP resides inside an argument of the antecedent VP. In a normal VP ellipsis as (414)-(416), the entire content of the antecedent VP is "copied", but in an ACE, the copied content should not just include a simple copy of the argument, or else we will get an infinite recursion. Thus, the "reconstruction" of (417)a is (418)a, with the meaning (419), and not (418)b or (418)c.

(418) a. John reads every book$_t$ that Mary reads $\_t$.
b. * John reads every book that Mary reads every book.
c. * John reads [every book that Mary reads [every book that Mary reads [every book that
...

(419) $every(\lambda x.boox(x) \wedge read(mary, x), \lambda y.read(john, y))$

The sentence (418)a does not have an ellipsis. It has a simple relative clause with the unbounded dependency $\_t$, just as we saw in section 4.6.1.

9.2 Overriding Parts

In the default case such as (420)a, the entire content of the antecedent VP is "copied", so Bill introduced Mary to Sue yesterday. However, some parts of the antecedent VP can be overridden by the new VP (adjuncts more easily than verb arguments), as in (420)b.

(420) a. John introduced Mary to Sue yesterday, and Bill did (so) too.
b. John introduced Mary to Sue yesterday, and Bill did so today.

In such cases, the "did so" part can, and sometimes must be omitted:

(421) a. John owns a Mercedes, and Mary, a Porsche. (gapping)
b. John introduced Mary to Sue yesterday, and Zelda to Sara today. (ACC)
c. John introduced Mary to Sue yesterday, and to Sara today. (ACC)
d. John gave Mary a book yesterday, and today too / and also today.
e. John ate a cake today, and Bill, too. (stripping)
f. John ate a cake today, but not Bill. (stripping)

(ACC = argument cluster coordination.) We may get ambiguities:

(422) John introduced Mary to Sue yesterday, and Sarah, today.

This is three-way ambiguous between:

(423) and Sarah introduced Mary to Sue today.
and John introduced Sarah to Sue today.
and John introduced Mary to Sarah today.

They have different likelihoods in different contexts.

The elided part can involve an embedded VP:

(424) a. John wanted to play the piano, and Mary, [wanted to play] the violin.
b. John could have been playing the guitar, and Mary, [could have been playing] the violin.

The literature on ellipsis reconstruction is too large to enumerate here. For some references, see (Dalrymple et al., 1991; Lappin and Benmamoun, 1999; Hardt, 1999; Ginzburg and Sag, 2001; Beavers and Sag, 2004).[1] For work on ellipsis in glue semantics, see (Crouch, 1999; Asudeh and Crouch, 2002).

[1] See also http://www.everything2.com/index.pl?node_id=1534953.

Chapter 10

Numerical Comparatives

Many logic puzzle texts include numeric constraints. One example is condition (4) in Figure 1.1. The second part of that condition says that for each room, the size of the set of sculptures exhibited in that room is at most 3. In the following examples, the sizes of two sets are compared:

(425) a. The planting committee has exactly two more members than the trails committee.
b. Cage Z contains exactly twice as many lizards as cage Y.

In this chapter I therefore investigate constructions which compare the number of elements satisfying a predicate either with a fixed number or with the number of elements which satisfy another predicate. The reason I chose to focus on such comparative constructions (as well as those in chapters 11 and 14) is that specifying their syntax-semantics interface is particularly interesting and challenging. These constructions have various cases, and the analysis of some of them is not trivial because of interactions with ellipsis. Sometimes it is not clear whether or not ellipsis plays a role. This is why it is particularly important to show how to handle such constructions. At the end of the chapter we will see that we need to introduce an additional level of complexity into the logical apparatus so that it could reason about sets of elements and their sizes.

10.1 Overview

In the following sentence, the number of boxes Ella lifted seems to be compared directly to Victor. But Victor is not a number. How can the expression denoting the real intended

number, which is the number of boxes Victor lifted, be calculated from this impoverished input?

(426) Ella lifted more boxes than Victor.
$|\lambda x.box(x) \wedge lift(ella, x)| > |\lambda x.box(x) \wedge lift(victor, x)|$

Notice also that the following two sentences seem very similar:

(427) a. John has more cats than Bill.
b. John has more cats than dogs.

but they have very different truth conditions:

(428) a. $|\lambda x.cat(x) \wedge has(john, x)| > |\lambda x.cat(x) \wedge has(bill, x)|$
b. $|\lambda x.cat(x) \wedge has(john, x)| > |\lambda x.dog(x) \wedge has(john, x)|$

In particular, note that in the first, *cat* is duplicated while in the second, $john$ is duplicated. Here is another example:

(429) a. More students attended the party than teachers.
b. More students attended the party than the reception.

(430) a. $|\lambda x.student(x) \wedge attend(x, the(party))| > |\lambda x.teacher(x) \wedge attend(x, the(party))|$
b. $|\lambda x.student(x) \wedge attend(x, the(party))| > |\lambda x.student(x) \wedge attend(x, the(reception))|$

How can these differences be explained and how can different truth conditions be calculated for seemingly identical syntactic structures (all the *than*-complements above are NPs)?

The principle that will be demonstrated in this chapter (as well as the next and 14) is that just looking at a few examples such as the ones above can be misleading. One might be tempted to develop a solution that treats these sentences at face value. However, such a solution will fail when more variations are taken into consideration. Therefore, what needs to be done is a systematic linguistic (manual) exploration of all possibilities, in order to achieve a deep and comprehensive analysis.

As we shall see, the examples above interact with ellipsis. For example, (427)a can be continued with *does*, and (429)a with *did*. Therefore, the first step is to simplify the matter and investigate cases of numerical comparison that do not involve ellipsis. A systematic investigation reveals several cases, with the following representative examples:

(431) a. More than five students arrived.

b. More students than teachers arrived.

c. More students than Bill has ever met arrived.

d. More students than I thought attended the party.

e. More students attended the party than teachers attended the reception.

Their truth conditions can be written as:

(432) a. $|student \cap arrive| > 5$

b. $|student \cap arrive| > |teacher \cap arrive|$

c. $|student \cap arrive| > |student \cap \lambda x.meet(bill, x)|$ [1]

d. $|student \cap \lambda x.attend(x, the(party))| >$
$\iota m.think(I, \langle\langle |student \cap \lambda x.attend(x, the(party))| = m \rangle\rangle)$ [2]

e. $|student \cap \lambda x.attend(x, the(party))| > |teacher \cap \lambda x.attend(x, the(reception))|$

The first one has a comparison to a fixed number. The rest have various possibilities that are different from each other with respect to what the *than*-complement is, and how the three sets are used in intersections (the three sets come from the noun's predicate *student*, the main predicate, and the *than*-complement's predicate). Therefore, they constitute different cases (one cannot be derived from any of the others). We will see below how the truth conditions can be calculated.

In (431)c-e, the *than*-complement is a clause, and so it is amenable to ellipsis. This occurs often in comparative constructions, and we will explore this interaction in section 10.8.

The analysis of comparatives here is similar to the one in (Pulman, 1991). The main differences are: 1) the analysis here is expressed using glue semantics on F-structures (in the framework that supports compact ambiguity management) rather than a simple c-structure grammar with Montague-like semantics; 2) the analysis here covers more cases (and will be extended to gradable comparatives in the next chapter and to *same* and *different* in chapter 14); and 3) the analysis here is connected to a computational component that can calculate inferences based on the representations (section 10.10).

[1] It is a common practice to identify a set with its characteristic function or predicate expression, namely identify $\{x \mid \varphi\}$ with $\lambda x.\varphi(x)$. This allows us to write things like $student \cap \lambda x.meet(bill, x)$.

[2] This is an informal notation. The $\langle\langle \ldots \rangle\rangle$ is intended to represent the proposition that the number of students who attended the party is m. The whole sentence claims that the number of students who actually attended the party is larger than the m such that I thought m students attended the party.

10.2 Comparison to an Explicit Number

In this section, we will handle sentences such as:

(433) More than five students sneezed.

10.2.1 MRL

Quantifiers such as *more than n*, *at most n*, and similar others are called type $\langle 1,1\rangle$ quantifiers because they expect two sets: a restrictor and a body. They make a statement about the size of the intersection of the two sets. We can extend our L^{MR} to include an additional sort (beyond e and ev), namely n for natural numbers. For a numeric comparison operator Q of type $n \times n \rightarrow t$ (one of: $\geq$, $\leq$, $>$, $<$, $=$), a number m, and two et predicates A and B, we add the form *num-compar*(Q, m, A, B) with the meaning:

(434) $[\![\textit{num-compar}(Q, m, A, B)]\!]^g_M = \textit{true}$ iff $Q(|[\![A]\!]^g_M \cap [\![B]\!]^g_M|, m)$

For example, we represent the meaning of (433) using:

(435) *num-compar*$(\geq, 5, \textit{student}, \textit{sneeze})$

and this gives the truth conditions:

(436) $|\textit{student} \cap \textit{sneeze}| \geq 5$

10.2.2 Glue Semantics Specification

The F-structure for (433) in the XLE is:

(437)
$$\begin{bmatrix} \text{PRED} & \text{`SNEEZE}\langle \boxed{1} \rangle\text{'} \\ \text{SUBJ} & \boxed{1}\begin{bmatrix} \text{PRED} & \text{`STUDENT'} \\ \text{NUM} & \text{`PL'} \\ \text{SPEC} & \begin{bmatrix} \text{NUMBER} & \begin{bmatrix} \text{PRED} & \text{`FIVE'} \\ \text{ADJUNCT} & \left\{\begin{bmatrix} \text{PRED} & \text{`MORE THAN'} \end{bmatrix}\right\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The same F-structure is obtained if we replace *more than* with any of: *less than*, *exactly*, *at least*, *at most*, *as many as*, *as few as*, and *fewer than*. For the sentence:

(438) No more than five students sneezed.

we also have the F-structure [PRED 'NO'] under SPEC|QUANT (as was the case for *every* in section 4.5). In the sentence:

(439) Five students sneezed.

we simply omit the adjunct list.

The glue semantics specification is very similar to that of the basic quantifiers in section 4.5.3, including the possibility of the quantifiers to float. We need to look for the appropriate material under SPEC|NUMBER, and not only SPEC|QUANT. Also, we need to rely on a table that translates number words to their numerical values, and on a table `quant_num_rel` that maps quantifier names to the appropriate numerical relation.

```
float_num_compar(%Rel,%n,%NP,%LandingGlueCat)
        ==> num_compar_sem(%Rel,%n,%NP,%LandingGlueCat).

num_compar_sem(%Rel,%n,%Node,%LandingGlueCat) :=
        A\B\num_compar(%Rel,%n,A,B) : quant_expr(%Node,%LandingGlueCat).

float_num_compar(%Rel,%Number,%NP,%LandingGlueCat) :=
        -path(%NP,NTYPE,NSEM,COMMON,measure),
           "this is to prevent degrees like [6 feet] in numerical comparatives"
        consume_spec(%NP,%%,%Spec,NUMBER,%NumberWord),
        number_word_number(%NumberWord,%Number),
        calc_quant_num_rel(%Spec,%Rel),
        det_quant_float(%NP,%LandingGlueCat).

calc_quant_num_rel(%Spec,%Rel) :=
        NUMBER(%Spec,%Number),
        ( (-ADJUNCT(%Number,%%),
           quant_num_rel(na,%Rel)) |
          (ADJUNCT(%Number,%Set),
           in_set(%A,%Set),
           +PRED(%A,%Name),
           ( ( -QUANT(%Spec,%%),
               quant_num_rel(%Name,%Rel) ) |
             ( path(%Spec,QUANT,PRED,no),
               quant_num_rel(no,%Name,%Rel) ) )) ).
```

```
|- number_word_number(one,1).
|- number_word_number(two,2).
|- number_word_number(three,3).
...

|- quant_num_rel(na,'=').
|- quant_num_rel(exactly,'=').
|- quant_num_rel('at least','>=').
|- quant_num_rel('at most','<=').
|- quant_num_rel('more than','>').
|- quant_num_rel('less than','<').
|- quant_num_rel('as many as','=').
...

|- quant_num_rel(no,'more than','<').
|- quant_num_rel(no,'less than','>').
```

For a sentence such as (439), there is no ADJUNCT, and we call the quantifier name NA (not-available). If in addition to the fact `quant_num_rel(na,'=')` we also add the fact `quant_num_rel(na,'>=')` to indicate the ambiguity of (439), then we will get a split in the choice space and two results, thanks to the fact that `consume_spec` consumes the name of the number. This is another example where an ambiguity is easily introduced by using the power of the rewriting system (as we did for quantifier landing in section 4.5.3).

10.3 Comparison to a NP

10.3.1 MRL

In the sentence:

(440) More students than teachers attended the party.

we have a type $\langle 1,1,1 \rangle$ quantifier, which expects not only a restrictor and a body, but also another set to compare with. We compare the number of students who attended the party to a number which is not given explicitly as a numerical expression but implicitly through the predicate *teachers*. The meaning can be expressed as:

(441) $|student \cap \lambda x.attend(x, the(party))| > |teacher \cap \lambda x.attend(x, the(party))|$

We can extend our MRL as follows. For a numeric comparison operator Q, and three *et* predicates A, B and C, we add to L^{MR} the form *num-compar2*(Q, A, B, C) with the meaning:

(442) $[\![\textit{num-compar2}(Q, A, B, C)]\!]^g_M = \textit{true}$ iff $Q(|[\![A]\!]^g_M \cap [\![B]\!]^g_M|, |[\![C]\!]^g_M \cap [\![B]\!]^g_M|)$

Thus, the meaning of (440) is represented as:

(443) $\textit{num-compar2}(>, \textit{student}, \lambda x.\textit{attend}(x, \textit{the}(\textit{party})), \textit{teacher})$

10.3.2 Why Introduce More Operators?

Before we proceed, I need to explain why I do not simply add the numerical comparison operators to L^{MR} together with the function symbol $|\cdot|$ for getting the size of sets, and instead use *num-compar*, *num-compar2*, and further operators below. I do this in order to adhere to the goal of having the *shape* of the semantic representations as similar as possible to the English/syntax surface form on the one hand, while still having a precise MTS on the other hand. This is important in order to facilitate the computation of the semantic representations from the syntax (see section 2.6 again). In particular, all the cases analyzed here will use MRL representations that look very similar to each other even though the *num-compar* operators expand to very different truth conditions. Writing those conditions directly would obfuscate the similarities.

Notice that "attended the party" appears once in (440) and twice in the truth conditions (441). Although we could write an entry in the glue specification which takes this as one argument and uses it twice (i.e.: $\lambda A \lambda B \lambda C.Q(|A \cap \underline{B}|, |C \cap \underline{B}|)$), this duplication would have negative consequences to readability of the specification, and more importantly, to cases of ambiguity and their compact representation by the algorithms of chapter 7. A simplified demonstration is the following.

Suppose the VP meaning needs to be substituted instead of B in φ, and suppose that the VP is ambiguous between m_1 and m_2, in contexts $A1$ and $A2$, respectively. If B appears just once in φ, then we can simply substitute $\langle A1 : m_1, A2 : m_2\rangle$ instead of B in φ. But if B appears twice in φ then we no longer want to do that because $\langle A1 : m_1, A2 : m_2\rangle$ itself would appear twice. Instead, we would introduce an additional label, say l_{17}, whose value is $\langle A1 : m_1, A2 : m_2\rangle$, and now use l_{17} instead of the occurrences of B in φ. This additional level of indirection can be prevented if we use the representation in (443).

Just as in software engineering, it is better to keep separate levels: one for the semantic representation, and one for its implementation. The translation from the first to the second can be done very efficiently, but we want to delay its application to a later stage and not be forced to integrate it in the (glue) composition stage (we saw a similar point in section 8.4.3).

10.3.3 Glue Semantics Specification

The F-structure given by the XLE to (440) is:

(444)
$$\begin{bmatrix} \textsc{pred} & \text{`}\textsc{attend}\langle \boxed{1},\boxed{2}\rangle\text{'} \\ \textsc{subj} & \boxed{1}\begin{bmatrix} \textsc{pred} & \text{`}\textsc{student}\text{'} \\ \textsc{num} & \text{`}\textsc{pl}\text{'} \\ \textsc{spec} & \begin{bmatrix} \textsc{aquant} & \begin{bmatrix} \textsc{pred} & \text{`}\textsc{many}\text{'} \\ \textsc{adjunct} & \left\{ \begin{bmatrix} \textsc{pred} & \text{`}\textsc{more}\langle\boxed{3}\rangle\text{'} \\ \textsc{obl-c} & \boxed{3}\begin{bmatrix} \textsc{pred} & \textsc{than}\langle\boxed{4}\rangle \\ \textsc{obj} & \boxed{4}\begin{bmatrix} \textsc{pred} & \text{`}\textsc{teacher}\text{'} \\ \textsc{num} & \text{`}\textsc{pl}\text{'} \end{bmatrix} \end{bmatrix} \end{bmatrix} \right\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsc{obj} & \boxed{2}\begin{bmatrix} \textsc{pred} & \text{`}\textsc{party}\text{'} \\ \textsc{spec} & \begin{bmatrix} \textsc{det} & \begin{bmatrix} \textsc{pred} & \text{`}\textsc{the}\text{'} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

(Here OBL-C is a shorthand for OBL-COMPAR). Instead of *more ... than* in (440) we could have: *as many ... as*. This is the reason for the 'MANY' in (444). With *as many as*, we get 'AS⟨OBL-COMPAR⟩' instead of 'MORE⟨OBL-COMPAR⟩', and 'AS⟨OBJ⟩' instead of 'THAN⟨OBJ⟩'. Also, instead of *more* above we could have *fewer*, in which case we have 'FEW' instead of 'MANY' in the F-structure.

We extend the glue semantics specification as follows:

```
float_num_compar2(%Rel,%Restr,%Compared,%LandingGlueCat)
          ==> num_compar2_sem(%Rel,%Restr,%Compared,%LandingGlueCat).

num_compar2_sem(%Rel,%Restr,%Compared,%LandingGlueCat) :=
      C\A\B\num_compar2(%Rel,A,B,C) :
          noun_expr(%Compared) ->
            quant_expr(%Restr,%LandingGlueCat).
```

```
float_num_compar2(%Rel,%NP,%Compared,%LandingGlueCat) :=
          consume_spec(%NP,%%,%Spec,AQUANT,%Base),
          path(%Spec,AQUANT,ADJUNCT,%Set),
          in_set(%A,%Set),
          +PRED(%A,%Name),
          quant_num_compar(%Base,%Name,%Rel),
          path(%A,OBL-COMPAR,OBJ,%Compared),
          det_quant_float(%NP,%LandingGlueCat).

 |- quant_num_compar(many,more,'>').
 |- quant_num_compar(many,as,'=')
 |- quant_num_compar(few,more,'<').
```

These definitions could be further refined to account for: *no more / no fewer students than teachers* and *at least / at most as many students as teachers*.

This solution works just as well when the *than* complement is a more complex NP. For example, in the sentence:

(445) More students than French teachers attended the party.

the modification of *teachers* will work as usual based on the specification for adjectives in section 4.6.1.

10.4 Comparison to a Clause Missing an NP

10.4.1 Analysis

In the following sentences:

(446) a. More people than you have ever met in your life are waiting now in the lobby to greet you.

b. More students than Bill (has ever) met attended the party.

we again have a type $\langle 1,1,1\rangle$ quantifier, except that now the extra set argument needs to be intersected with the restrictor set of the quantifier *more* rather than with its body set. The truth conditions of (446)b can be written as:

(447) $|student \cap \lambda x.attend(x, the(party))| > |student \cap \lambda x.meet(bill, x)|$

How do we get these truth conditions? "than Bill met" has the same meaning as "that Bill met" has in a relative clause that modifies a noun. This is a clause of type t which is missing an element of type e, thus giving a predicate of type $e \rightarrow t$. This case is called "comparative deletion" in the literature.

10.4.2 MRL

We extend our MRL as follows. For a numeric comparison operator Q and three *et* predicates A, B, and C, we add the form *num-compar3*(Q, A, B, C) with the meaning:

(448) $[\![\textit{num-compar3}(Q, A, B, C)]\!]^g_M = \textit{true}$ iff $Q(|[\![A]\!]^g_M \cap [\![B]\!]^g_M|,\ |[\![A]\!]^g_M \cap [\![C]\!]^g_M|)$

Notice the difference between this and *num-compar2*. (It happens to be the case that *num-compar3*$(Q, A, B, C) \equiv$ *num-compar2*(Q, B, A, C).) The meaning of (446)b is now represented as:

(449) *num-compar3*$(>, \textit{student}, \lambda x.\textit{attend}(x, \textit{the}(\textit{party})), \lambda x.\textit{meet}(\textit{bill}, x))$

10.4.3 Syntax

The F-structure for (446)b should be:

(450)
$$\begin{bmatrix}
\text{PRED} & \text{`ATTEND}\langle \boxed{1}, \boxed{2} \rangle\text{'} \\
\text{SUBJ} & \boxed{1}\begin{bmatrix}
\text{PRED} & \text{`STUDENT'} \\
\text{NUM} & \text{`PL'} \\
\text{SPEC} & \begin{bmatrix} \text{AQUANT} & \begin{bmatrix} \text{PRED} & \text{`MANY'} \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{`MORE}\langle \boxed{3} \rangle\text{'} \\ \text{OBL-COMPAR} & \boxed{3}\begin{bmatrix} \text{PRED} & \text{THAN}\langle f \rangle \\ \text{OBJ} & f \end{bmatrix} \end{bmatrix} \right\} \end{bmatrix} \end{bmatrix}
\end{bmatrix} \\
\text{OBJ} & \boxed{2}\begin{bmatrix} \text{PRED} & \text{`PARTY'} \\ \text{SPEC} & [\text{DET } [\text{PRED `THE'}]] \end{bmatrix}
\end{bmatrix}$$

where:

$$f = \begin{bmatrix}
\text{PRED} & \text{MEET}\langle \boxed{5}, \boxed{6} \rangle \\
\text{SUBJ} & \boxed{5}\begin{bmatrix} \text{PRED} & \text{BILL} \end{bmatrix} \\
\text{OBJ} & \boxed{6}\begin{bmatrix} \text{PRED} & \text{PRO} \end{bmatrix} \\
\text{REL-PRO} & \boxed{6}
\end{bmatrix}$$

The F-structure of the clause "than Bill met" which is missing a direct object is identical to the F-structure of the relative clause "that Bill met" (see section 4.6.1).[3]

10.4.4 Glue Semantics Specification

The following allows us to connect the meaning of the complement to the comparator. It is similar to the definition for `num_compar2`, except that the way it expects its complement is similar to the definition for relative clauses in section 4.6.1.

```
float_num_compar_clause(%Rel,%Restr,%Compared,%Missing,e,%LandingGlueCat)
        ==> num_compar3_sem(%Rel,%Restr,%Compared,%Missing,%LandingGlueCat).

num_compar3_sem(%Rel,%Restr,%Compared,%Missing,%LandingGlueCat) :=
        C\A\B\num_compar3(%Rel,A,B,C) :
          (%Missing$e -> %Compared$t) -> quant_expr(%Restr,%LandingGlueCat).

float_num_compar_clause(%Rel,%NP,%Compared,%Missing,%MissType,%LandingGlueCat) :=
          consume_spec(%NP,%%,%Spec,AQUANT,%Base),
          path(%Spec,AQUANT,ADJUNCT,%Set),
          in_set(%A,%Set),
          +PRED(%A,%Name),
          quant_num_compar(%Base,%Name,%Rel),
          path(%A,OBL-COMPAR,OBJ,%Compared),
          +REL-PRO(%Compared,%Missing),
          (+qp(%Feat,[%%,%Missing]), -{%Feat=REL-PRO}),
          compar_missing_type(%Feat,%MissType),
          det_quant_float(%NP,%LandingGlueCat).

 |- compar_missing_type(OBJ,e).
 |- compar_missing_type(COMP,t).
 |- compar_missing_type(SPEC,q).
```

The macro `float_num_compar_clause` will serve us also in the two other cases of clausal *than*-complement below. The line which starts with `+qp` checks under what grammatical function the missing element resides, and based on it, determines its type. Here the missing element is of type e. Below, t and q will also be possible.

[3] Just for convenience, we renamed PRON-REL to REL-PRO here, and we use here PRED PRO instead of NTYPE|NSYN PRONOUN as in section 4.6.1.

This specification works because "Bill" and "met" get the usual entries, and can combine to form the glue statement $\lambda x.meet(bill, x) : \boxed{6}^e \rightarrow f^e$ (for $\boxed{6}$ and f from (450)). The macro `num_compar3_sem` contributes a glue statement that expects exactly this $\boxed{6}^e \rightarrow f^e$ statement as its argument.

10.5 Comparison to a Clause Missing a Clause

The analysis of:

(451) More students than Bill thought attended the party.

is very similar, except that now the *than*-complement is missing the complement of *thought*, which is itself a clause. For a numeric comparison operator Q, two *et* predicates A and B, and a $t \rightarrow t$ predicate S,[4] we add the form $num\text{-}compar4(Q, A, B, S)$ with the meaning:

(452) $[\![num\text{-}compar4(Q, A, B, S)]\!]^g_M = true$ iff
$Q(|[\![A]\!]^g_M \cap [\![B]\!]^g_M|,\ \iota m.[\![S]\!]^g_M(\langle\!\langle |[\![A]\!]^g_M \cap [\![B]\!]^g_M| = m \rangle\!\rangle))$

The meaning of (451) is now represented as:

(453) $num\text{-}compar4(>, student, \lambda x.attend(x, the(party)), \lambda x.think(bill, x))$

which gives the truth conditions:

(454) $|student \cap \lambda x.attend(x, the(party))| >$
$\iota m.think(bill, \langle\!\langle |student \cap \lambda x.attend(x, the(party))| = m \rangle\!\rangle)$

The F-structure for (451) is like the one in (450) except with a different f:

(455) $f = \begin{bmatrix} \text{PRED} & \text{THINK}\langle \boxed{5}, \boxed{6} \rangle \\ \text{SUBJ} & \boxed{5}\begin{bmatrix} \text{PRED} & \text{BILL} \end{bmatrix} \\ \underline{\text{COMP}} & \boxed{6}\begin{bmatrix} \text{PRED} & \text{PRO} \end{bmatrix} \\ \text{REL-PRO} & \boxed{6} \end{bmatrix}$

We use again the macro `float_num_compar_clause` from above, and add to the specification:

[4] Here t represents the type of propositions rather than truth values.

```
float_num_compar_clause(%Rel,%Restr,%Compared,%Missing,t,%LandingGlueCat)
        ==> num_compar4_sem(%Rel,%Restr,%Compared,%Missing,%LandingGlueCat).

num_compar4_sem(%Rel,%Restr,%Compared,%Missing,%LandingGlueCat) :=
        C\A\B\num_compar4(%Rel,A,B,C) :
          (%Missing$t -> %Compared$t) ->
            quant_expr(%Restr,%LandingGlueCat).
```

The fact `compar_missing_type(COMP,t)` from above makes this work.

10.6 Comparison to a Clause Missing a Quantifier

10.6.1 Analysis

In the following sentence:

(456) More students attended the party than teachers attended the reception.

we have yet again a type $\langle 1,1,1\rangle$ quantifier, except that now the size of the extra set argument provides the number to be compared against, without intersecting the set with the restrictor or body set. The truth conditions can be written as:

(457) $|student \cap \lambda x.attend(x, the(party))| > |teacher \cap \lambda x.attend(x, the(reception))|$

The question is how do we get these truth conditions? What is the meaning of the complement "teachers attended the reception" here? Generally speaking, this clause can stand as a separate sentence, where the quantifier in the NP "teachers" is implicit. In a separate sentence, this quantifier could be the existential quantifier (some teachers) or the generic quantifier. However, in (456), the denotation of the complement cannot simply be something that gives a truth value. It has to be missing something in order to provide, in some way, a number to compare against.

What exactly is missing? One option is to assume that a quantifier is missing. So the meaning of "teachers attended the reception" in (456) is

(458) $\lambda Q.Q(teacher, \lambda x.attend(x, the(reception)))$

Then the operator *more ... than* could apply this meaning on $\lambda x \lambda y.|x \cap y|$ to obtain the number of teachers who attended the reception, so that this number could be compared to the number of students who attended the party.

Another option is to assume that only the number is missing, and there is an implicit quantifier in place. If that quantifier is *exactly*, then the meaning of the clause is $\lambda n.\textit{num-compar}(=, n, \textit{teacher}, \lambda x.\textit{attend}(x, \textit{the}(\textit{reception})))$. Then the operator *more ... than* could apply $\lambda P.\iota(P)$ on that to obtain the unique number which satisfies this predicate. If the quantifier is *at-least*, then the meaning of the clause is $\lambda n.\textit{num-compar}(\geq, n, \textit{teacher}, \lambda x.\textit{attend}(x, \textit{the}(\textit{reception})))$, and the operator *more ... than* could apply $\lambda P.\max(P)$ to obtain that number. However, both these ideas are more complicated than assuming that the whole determiner (of *teachers*) is missing. The only incentive to use these more complex options would be if we had evidence that the quantifier could be said explicitly where only the number is missing. But this is not the case:

(459) * More students attended the party than exactly/at-least/some $_{\neg t}$ teachers attended the reception.

We conclude that the quantifier is missing, and the meaning of the complement is (458). Because a quantifier rather than a more standard constituent is missing in the complement, this case is called "comparative subdeletion" in the literature.

10.6.2 Solution

We add another operator to our MRL. For a numeric comparison operator Q, two et predicates A and B, and an expression S which is missing a quantifier,[5] we add the form $\textit{num-compar5}(Q, A, B, S)$ with the meaning:

(460) $[\![\textit{num-compar5}(Q, A, B, S)]\!]^g_M = \textit{true}$ iff $Q(|[\![A]\!]^g_M \cap [\![B]\!]^g_M|,\ \varphi)$

What is φ? It is tempting to take $\varphi = [\![S]\!]^g_M(\lambda u \lambda v.|u \cap v|)$, and thus obtain in the example above $|\textit{teacher} \cap \lambda y.\textit{attend}(y, \textit{the}(\textit{reception}))|$. Although this works in practice, strictly speaking it is not correct because we are giving $[\![S]\!]^g_M$ an argument of the wrong type. Since $[\![S]\!]^g_M$ is a clause missing a quantifier, its argument is of type $(et \to et \to t)$, and it returns a t. However, $\lambda u \lambda v.|u \cap v|$ is of type $et \to et \to n$. Instead, we could use either of the following:

(461) $\varphi = \iota m.[\![S]\!]^g_M(\lambda u \lambda v.|u \cap v| = m)$
$\varphi = |\lambda x.[\![S]\!]^g_M(\lambda u \lambda v.\ x \in u \cap v)|$

[5] It is therefore of type $((e \to t) \to (e \to t) \to t) \to t$.

Thus, the meaning of (456) is represented as:

(462) $num\text{-}compar5(>, student, \lambda x.attend(x, the(party)),$
$\lambda Q.Q(teacher, \lambda y.attend(y, the(reception))))$

We can expand the definition and see that this is equivalent to:

(463) $|student \cap \lambda x.attend(x, the(party))| >$
$\iota m.(\lambda Q.Q(teacher, \lambda y.attend(y, the(reception))))(\lambda u \lambda v.|u \cap v| = m) \equiv$
$|student \cap \lambda x.attend(x, the(party))| >$
$\iota m.(\lambda u \lambda v.|u \cap v| = m)(teacher, \lambda y.attend(y, the(reception))) \equiv$
$|student \cap \lambda x.attend(x, the(party))| >$
$\iota m.|teacher \cap \lambda y.attend(y, the(reception))| = m \equiv$
$|student \cap \lambda x.attend(x, the(party))| > |teacher \cap \lambda y.attend(y, the(reception))|$

The F-structure for (456) should be like the one in 450, except for the f:

(464) $f =$
$\begin{bmatrix} \text{PRED} & \text{ATTEND}\langle \boxed{5}, \boxed{6} \rangle \\ \text{SUBJ} & \boxed{5} \begin{bmatrix} \text{PRED} & \text{'TEACHER'} \\ \text{NUM} & \text{'PL'} \\ \text{SPEC} & \boxed{7}[\text{PRED} \quad \text{PRO}] \end{bmatrix} \\ \text{OBJ} & \boxed{6} \begin{bmatrix} \text{PRED} & \text{'RECEPTION'} \\ \text{SPEC} & [\text{DET} \ [\text{PRED} \ \text{'THE'}]] \end{bmatrix} \\ \text{REL-PRO} & \boxed{7} \end{bmatrix}$

As in the previous two cases, we need to add a specification of this case. However, just doing what we did above will not be enough because the missing quantifier is not a "natural" missing constituent like a NP or a sentence above were. Instead of `%Missing$q` below, we could have written an expression of type $et \rightarrow et \rightarrow t$, but that would just add a complication without any benefits. The purpose of `missing_quant_sem` is to add a glue statement which takes the materials contributed by "teachers" and "attended the reception", and creates $\lambda Q.Q(teacher, \lambda x.attend(x, the(reception)))$ from them.

```
float_num_compar_clause(%Rel,%Restr,%Compared,%Missing,q,%LandingGlueCat)
        ==> num_compar5_sem(%Rel,%Restr,%Compared,%Missing,%LandingGlueCat),
            missing_quant_sem(%Missing,%Compared).
```

```
num_compar5_sem(%Rel,%Restr,%Compared,%Missing,%LandingGlueCat) :=
        C\A\B\num_compar5(%Rel,A,B,C) :
          (%Missing$q -> %Compared$t) ->
            quant_expr(%Restr,%LandingGlueCat).

missing_quant_sem(%Missing,%Compared) :=
        Q\R\S\mquant(Q,R,S) :
           %Missing$q ->
           noun_expr(%Node) ->
             (%Node$e -> %Compared$t) -> %Compared$t .
```

This will create two glue statements:

(465) a. $\lambda Q \lambda R \lambda S.mquant(Q,R,S) : \boxed{7}^q \rightarrow (\boxed{5}^e_v \rightarrow \boxed{5}^t_r) \rightarrow (\boxed{5}^e \rightarrow f^t) \rightarrow f^t$

b. $\lambda C \lambda A \lambda B.num\text{-}compar5(>,A,B,C) : (\boxed{7}^q \rightarrow f^t) \rightarrow \ldots$

The first statement will combine with $teacher : \boxed{5}^e_v \rightarrow \boxed{5}^t_r$ and with $\lambda x.attend(the(reception)) : \boxed{5}^e \rightarrow f^t$ to create:

(466) $\lambda Q.mquant(Q, teacher, \lambda x.attend(the(reception))) : \boxed{7}^q \rightarrow f^t$

This will then serve as the C argument of (465)b, as desired. The only thing left to do is to define in the MRL:

(467) $mquant(Q,R,S) \equiv Q(R,S)$

10.7 Specified Difference

A variation of the above cases specifies exactly by how much one number is bigger than the other:

(468) a. * At least three more than five students arrived.

b. At least three more students than teachers attended the party.

c. At least three more students than Bill has met attended the party.

d. At least three more students than I thought attended the party.

e. At least three more students attended the party than teachers attended the reception.

In place of *at least three*, we could also use *at most three*, *exactly three*, *as many as three*, and nothing (i.e. just "Three more students ..."), although probably not *as few as three*, *more/less than three*, *fewer than three*. It seems that we must use *more* just before the noun: "At least/most three fewer students" is dubious.

We can treat these cases by adding a *diff* version for each of *num-compar2* to *num-compar5*. For example:

(469) $[\![\textit{num-diff2}(Q, d, A, B, C)]\!]^g_M = \textit{true}$ iff $Q(|[\![A]\!]^g_M \cap [\![B]\!]^g_M| - |[\![C]\!]^g_M \cap [\![B]\!]^g_M|,\ d)$
$[\![\textit{num-diff3}(Q, d, A, B, C)]\!]^g_M = \textit{true}$ iff $Q(|[\![A]\!]^g_M \cap [\![B]\!]^g_M| - |[\![C]\!]^g_M \cap [\![A]\!]^g_M|,\ d)$

Thus, (468)b gets the representation:

(470) $\textit{num-diff2}(\geq, 3, \textit{student}, \lambda x.\textit{attend}(x, \textit{the}(\textit{party})), \textit{teacher})$

Extending the glue specification to account for such cases is done in a very similar way to the previous cases.

The difference may also be expressed in terms of how many times one number is bigger than another:

(471) a. * At least three times as many as five students arrived.
b. At least three times as many students as teachers attended the party.
c. At least three times as many students as Bill has met attended the party.
d. At least three times as many students as I thought attended the party.
e. At least three times as many students attended the party as teachers attended the reception.

(Instead of "two times", the word "twice" is used.) For these, we can define operators *num-tdiff2* to *num-tdiff5* along similar lines.

10.8 Interaction with Ellipsis

10.8.1 Extraposition

Before we get to ellipsis, a few notes about extraposition. As we saw in the first four cases above, the *than* clause can appear inside the main NP. This is often very strange with the last (subdeletion) case – (472)a, but not always – (472)b.

(472) a. ?? More students [than teachers attended the reception] attended the party.
b. More humans [than there are $\__t$ grains of sand on the beach] will inherit the Earth.

The *than* complement can be extraposed to be outside the NP, except when the complement has an explicit number:

(473) $*$ More students than five arrived.
$*$ More students arrived than five.

(474) a. More students arrived than teachers.
b. More students arrived than Bill has ever met in his life.
c. More students attended the party than I thought.
d. More students attended the party than teachers attended the reception.

As usual with extraposition, it is strongly preferred when the extraposed expression is long. Thus (474)b is preferred to:

(475) More students than [Bill has ever met in his life] arrived.

The XLE grammar already recognizes the possibility of extraposition, and assigns the same F-structure to a sentence with extraposition as to one without. For example, (474)a gets exactly the same F-structure (444) as (440) does.

The ability to extrapose supports a wide range of ellipsis possibilities as we shall see next.

10.8.2 Interaction with Subdeletion

There are many potential omissions in the subdeletion case. Many of them sound even better than the non-elided version. Compare these cases to the ellipsis cases mentioned in chapter 9.

(476) More students attended the party than teachers attended the reception. (full)
More students attended the party than teachers did. (VPE)
More students attended the party than teachers the reception. (gapping)
More students attended the party yesterday than the reception today. (ACC)
More students attended the party yesterday than teachers the reception today.
More students attended the party than the reception. (ACC)
More students attended the party than attended the reception.

Here are more examples:

(477) a. John gave more books to Mary than Bill gave magazines to Sue. (full)
b. John gave more books to Mary than Bill did. (VPE)
c. John gave more books to Mary than Bill did/gave magazines. (VPE+override)
d. John gave more books to Mary than Bill to Sue. (gapping)
e. John gave more books to Mary than magazines to Sue. (ACC)
f. John gave more books to Mary than to Sue. (ACC)

Just as in (422), we get ambiguities:

(478) John gave more books to Mary than Bill.
a. (stripping) ⇒ John gave more books to Mary than Bill did.
b. (ACC) ⇒ John gave more books to Mary than to Bill.

In all these cases, it is clear that ellipsis plays a role because the cases are the same as those we saw in chapter 9, and the full range of ellipsis variations is possible. Therefore, all these cases should be automatically handled by an interaction of the treatment of numerical comparatives we saw in sections 10.2-10.6 with an independently-motivated treatment of ellipsis (which would be needed also for the cases in chapter 9). The sentences here should not be handled by additional special cases. In particular, *to Mary than to Sue* in (477)f is not a syntactic constituent. An even more striking example of this is:

(479) More kids like to eat hamburger with ketchup than without.
= More kids like to eat hamburger with ketchup than kids like to eat hamburger without ketchup.

Here is a demonstration that this kind of ellipsis is independently-motivated, i.e. it appears even when there is no comparative construction:

(480) John did his homework with the help of the TA whereas Bill did (so) without.

This conclusion contrasts with previous work such as (Rayner and Banks, 1990). That paper provides a separate special case for each of the sentences in a subset of (476) and (477). However, that solution is quite complicated because it conflates what should be two separate mechanisms, namely ellipsis-free comparatives and independently-motivated ellipsis, and tries to do again what the ellipsis mechanism should do anyway. Moreover,

their solution does not generalize to additional cases. In contrast, according to the analysis provided above, once the basic ellipsis-free cases of comparatives are analyzed, all other cases follow from the interaction with ellipsis. (This has not yet been implemented in this work, but one could take computational solutions provided in the literature for ellipsis and, if they are general enough, they should work without any addition with the comparatives here.)

10.8.3 Clarifying More Interactions

Let us examine some more cases. First, (481)a is a case of an (extraposed) *than*-complement which has a clause missing a clause, and it is *not* obtained from the subdeletion case (481)b by ellipsis.

(481) a. More students attended the party than I thought.

b. More students attended the party than I thought students attended the party.

This is because clausal arguments of verbs are not amenable to this kind of ellipsis. For example, the sentence:

(482) # John arrived late. But I didn't think.

cannot mean: "John arrived late. But I didn't think so / But I didn't think that John arrived late."

Second, the sentence (483)a belongs to the comparative deletion case, and it is not obtained by ellipsis from the subdeletion case (483)b.

(483) a. John read more books$_t$ than Bill read $\_{}_t$.

b. John read more books than Bill read _ books.

This is because it is unlikely there are independently-motivated cases of ellipsis where a direct object can be elided:

(484) # John read a magazine. And Bill read, too.

⇒ ? John read a magazine. And Bill read a magazine, too.

Even if this were possible, it would not support ellipsis in (483)a. We can show this by adapting the argument that Kennedy (1997) gave for gradable comparatives, and which will be reviewed in section 11.2.5. I will not repeat the whole argument and just show

one part of it. If (483)a were an elided form of (483)b, then the elided NP should be "more books" rather than "books". It cannot be argued that the ellipsis mechanism can ignore this difference because such leniency does not appear in other cases of ellipsis. For example, in (485), the missing material can only be "more articles" and not just "articles".

(485) John wrote more articles this year, and Bill wrote $\triangle$, too.

This argument also shows that (486)a belongs to the deletion and not the subdeletion case. It obviously involves stripping, i.e. it must be derived from (486)b. And (486)b must be derived from (486)c and not (486)d, for a similar reason to the argument above.

(486) a. John read more books than Bill.
b. John read more books than Bill did.
c. John read more books$_t$ than Bill read _$_t$.
d. John read more books than Bill read books.

In contrast, there are some cases of derivational ambiguity, where a sentence may be obtained by extraposition with no ellipsis, or by ellipsis. Thus, because ellipsis clearly plays a role in (477)e, it could also play a role in (487)a, i.e. (487)a would be derived from (487)b by ellipsis. But another option is that (487)a would be derived from (487)c by extraposition with no ellipsis. Perhaps both options are possible.

(487) a. John gave more books to Mary than magazines.
b. John gave more books to Mary than John gave magazines to Mary.
c. John gave more books than magazines to Mary.

A final interesting thing to note is that the ability of extraposition points out that antecedent contained ellipsis (see the end of section 9.1) should be defined with respect to F-structure rather than C-structure. In (488)b we have a case of comparative deletion, and (488)a is its elided form. That sentence has antecedent contained ellipsis – it is very similar to (417)a.

(488) a. John read more books than Bill did.
b. = John read more books$_t$ than Bill read _$_t$.

The sentences in (489)a-b parallel those in (488). Note that the *than*-complement has an ellipsis, but the *than*-complement does not reside inside an argument (the NP "more books") of the antecedent VP, in the surface structure. The "antecedent containment" can be seen only when the F-structure is considered. It is as if the sentence was (489)c, with the reconstruction (489)d.

(489) a. John gave more books to Mary than Bill did.
b. = John gave more books$_t$ to Mary than Bill gave $\__t$ to Mary.
c. John gave more books [than Bill did] to Mary.
d. = John read more books$_t$ [than Bill gave $\__t$ to Mary] to Mary.

10.9 Proportional Quantifiers

Proportional quantifiers provide an additional way that the sizes of sets may be compared. First, the quantifier *most* as in:

(490) Most students arrived.

can be treated in a similar way to *every* in sections 2.7.2 and 4.5:

(491) $I_M(most) = \{\langle A, B\rangle \in \wp(D^e_M) \times \wp(D^e_M) \mid |A \cap B| > |A - B|\}$.

Other proportional quantifiers are obtained by using any of the comparators mentioned in section 10.2, followed by a fraction instead of a whole number, and using the partitive form:

(492) At least / at most / exactly / (no) more than / (no) less than / ...
a. a/one half/third/fourth/... of the N P
b. two/three/four/dots halves/thirds/fourths/... of the N P

We can add to L^{MR}:

(493) $[\![\textit{frac-compar}(Q, p, A, B)]\!]^g_M = Q(|A \cap B|, p \cdot |A|)$

For example:

(494) At least three fifths of the students arrived.
$\textit{quant-frac}(\geq, 3/5, \textit{student}, \textit{arrive})$
$\equiv |\textit{student} \cap \textit{arrive}| \geq \frac{3}{5}|\textit{student}|$

"Most" could be also be interpreted as "at least half of the" in finite domains.

10.10 Logic and Computability

10.10.1 Direct Proof System

Can we devise a sound and complete proof system for L^{MR} and its semantics? (i.e. the logic L_0^{MR} that we defined in chapter 2, with the extensions from this chapter). The logic of L^{MR} is complex because L^{MR} is complex – it has many different kinds of expressions and interactions between them. It will become even more complex the more we progress with the fragments in subsequent chapters.

Notice that none of the MRL representations obtained for the sentences in our NL fragment *directly* involve sizes of sets. Instead, they involve the *num-compar* and *num-diff* operators, whose meanings depend on sizes of sets. Why is this point important? The reason is that there is a line of research (sometimes called "Natural Logic") that attempts to devise a logic which directly mirrors the structure of natural language without further assumptions. For example, (Moss, 2006) investigates a fragment of English that includes only syllogisms based on the quantifiers *every*, *some*, and *no*, and then extends it to a larger fragment with *most* and *there are more X than Y*. Moss investigates how to develop a sound and hopefully complete proof system for these operators directly. A similar work is (Pratt-Hartmann, 2006) (and see other works by that author), with an emphasis on studying the complexity classes of NL fragments.

These studies are interesting and important because they tackle what is needed for reasoning with NL directly. In particular, given a certain NL query, a computer can efficiently check whether it falls within any of the studied fragments, and if it does, the specialized proof system (and tools) developed for it could be used to answer the query more efficiently than when relying on more general methods (which might not even guarantee decidability).

However, until the time that the studied fragments are large enough to cover phenomena that are of interest here, we need other methods. In the rest of this section, I will show a proof-of-concept reasoning system that relies on existing FOL reasoners. This is not the best or most efficient way to do the reasoning. In particular, a specialized component for reasoning about sets and their sizes will be more efficient than a general-purpose FOL TP/MB reasoner that works with axioms that encode properties of sets. What we really need below is a combination of boolean arithmetic on sets and Presburger arithmetic (natural numbers with addition but without unbounded multiplication). This theory is

decidable, with a decision algorithm that relies on quantifier elimination (Kuncak et al., 2006). In fact, the kind of reasoning we are interested in involves only finite arithmetic, where we have an upper bound on the range of numbers we need to consider. Open questions remain regarding how to combine efficient reasoners for such theories with the work described here.

10.10.2 Naive Translation to FOL

The numeric quantifiers from section 10.2 are definable in FOL: Their truth conditions can be expressed in pure FOL (with equality) without assuming anything more. The translation μ_0 from section 2.8.2 from the MRL to FOL can be straightforwardly extended as follows:

(495) $\mu_0(\texttt{quant-num(>=,}n\texttt{,}\varphi\texttt{,}\psi\texttt{)}) =$

$$\exists \mathtt{x}_1 \ldots \exists \mathtt{x}_n \,.\, [\mathtt{x}_1 \neq \mathtt{x}_2 \;\&\; \mathtt{x}_1 \neq \mathtt{x}_3 \;\&\; \ldots \;\&\; \mathtt{x}_{n-1} \neq \mathtt{x}_n \;\&$$
$$\mu_0(\varphi)(\mathtt{x}_1) \;\&\; \ldots \;\&\; \mu_0(\varphi)(\mathtt{x}_n) \;\&\; \mu_0(\psi)(\mathtt{x}_1) \;\&\; \ldots \;\&\; \mu_0(\psi)(\mathtt{x}_n)]$$

The translation of *more than n*, *exactly n*, and the other quantifiers can proceed in a similar way (e.g., $\textit{quant-num}(\leq, n, A, B) \equiv \neg \textit{quant-num}(\geq, n+1, A, B)$).

This kind of translation can indeed support some inferences, such as:[6]

(496) At least four boys sneezed.
⇒ At least three boys sneezed.

(497) At most two sculptures are in room 1.
Sculpture C is in room 1.
Sculpture D is in room 1.
⇒ Sculpture E is not in room 1.

However, this direct translation has severe drawbacks:

Inefficiency: Except for small values of n, the approach is inefficient. In particular, humans can just as easily understand and reason about the expression "at least 100" as they can with "at least 50", but the representation for the former has twice as many quantifiers and approximately 2,500 times as many clauses.

[6] The inference in (497) relies on the Unique Names Assumption – see section 15.2.4.

Numerical Reasoning: The approach does not support full numerical reasoning. For example:

(498) At least three boys arrived.
More girls than boys arrived.
⇒ At least four girls arrived.

The truth conditions of the second sentence say that if the size of the group of boys who arrived is n then the size of the group of girls who arrived is at least $n + 1$. But it is not possible to express this by using the method of (495), because there is no way in FOL itself to talk about the difference between the number of elements that satisfy one predicate compared to those that satisfy another.

A precise formalization of this claim has been made and proved for several cases. For example, it has been shown that *more than half of the* (and hence *most*) is not definable in FOL (Barwise and Cooper, 1981). See also (Peters and Westerståhl, 2006) for a review of this and similar results as well as the logical tools that can be used to prove such claims.

The solution is to use a different translation to FOL. We need to reify the kind of entities we want to talk about directly, namely sets and their sizes. These will now be members of the domains in models, and we will use a collection of axioms to guarantee that the symbols in our FO vocabulary have the expected properties.

10.10.3 General Translation to FOL

The Language L^{IMP}

For L^{IMP}, I will use a four-sorted FOL (with equality) with the sorts e for individuals and ev for events, as well as s for sets of individuals and n for natural numbers. We have four corresponding unary predicate symbols: At, $Event$, Set, and Num. For convenience in the definition below, I also use t to designate the boolean truth values $true$ and $false$, but these two are not elements of the domain. In principle, every symbol is decorated with its sort and only appropriate combinations of symbols produce wffs; however, I will omit the sort decorations whenever they are clear. I am using sorted FOL for notational convenience and to reduce clutter. Also, there are some automated reasoners that can work with a sorted language and make the reasoning more efficient by taking advantage of the sort information. However, it is well known that a sorted FO language can be faithfully

embedded in the unsorted language of FOL (see the Box below), so what follows could be adapted also for plain FOL reasoners.

Embedding a Sorted FO Language in Unsorted FOL

A sorted FO language with sorts $s_1, \ldots, s_n$ can be embedded in the unsorted language of FOL as follows. We introduce new predicate symbols $Sort_1, \ldots, Sort_n$. We translate formulas in the sorted language to FOL as follows:

$\tau(\exists x^{s_i}.\varphi) = \exists x.[Sort_i(x) \wedge \tau(\varphi)]$
$\tau(\forall x^{s_i}.\varphi) = \forall x.[Sort_i(x) \rightarrow \tau(\varphi)]$

We also add a finite set of axioms AX. Axioms that specify sorts are:

- For every object constant c^{s_i} of sort s_i we add the axiom: $Sort_i(c)$
- For every function constant f of sort $i_1 \times \ldots \times i_n \rightarrow j$ we add the axiom: $\forall x_1 \ldots \forall x_n \forall y. f(x_1, \ldots, x_n) = y \rightarrow Sort_{i_1}(x_1) \wedge \ldots \wedge Sort_{i_n}(x_n) \wedge Sort_j(y)$
- For every relation constant R of sort $i_1 \times \ldots \times i_n \rightarrow t$ we add the axiom: $\forall x_1 \ldots \forall x_n. R(x_1, \ldots, x_n) \rightarrow Sort_{i_1}(x_1) \wedge \ldots \wedge Sort_{i_n}(x_n)$

Axioms that categorize each element to exactly one sort:

- For every $1 \leq i \leq n$, the axiom: $\exists x. Sort_i(x)$
- The axiom: $\forall x.[Sort_1(x) \vee \ldots \vee Sort_n(x)]$
- For every $1 \leq i < j \leq n$, the axiom: $\forall x. \neg[Sort_i(x) \wedge Sort_j(x)]$

Now we have: $\Gamma \models^{sFOL} \varphi$ iff $AX \cup \tau(\Gamma) \models^{FOL} \tau(\varphi)$.

The language has the following relation symbols:

symbol	sort	intended interpretation
$\leq$ $<$	$n \times n \rightarrow t$	numeric less than (or equal)
$\in$	$e \times s \rightarrow t$	set membership
$\subseteq$ $\subset$	$s \times s \rightarrow t$	set inclusion

The language also has the following object symbols and function symbols:

symbol	sort	intended interpretation
$\emptyset$	s	the empty set
$\{\cdot\}$	$e \rightarrow s$	singleton set construction
$\cap \cup -$	$s \times s \rightarrow s$	set intersection, union, and difference
$\lvert \cdot \rvert$	$s \rightarrow n$	set size
0 1 2 ...	n	natural numbers
$+$	$n \times n \rightarrow n$	number addition

Axioms

Equality: If the given FOL reasoner cannot reason directly about equality, we can add appropriate axioms:

(Ax.eq.reflx)	$\forall x.\ x = x$
(Ax.eq.symm)	$\forall x \forall y.\ x = y \rightarrow y = x$
(Ax.eq.trans)	$\forall x \forall y \forall z.\ x = y \wedge y = z \rightarrow x = z$
(Ax.eq.func)	For every n-ary function symbol f and for all $1 \leq i \leq n$
	$\forall x_1 \ldots \forall x_n \forall y.\ x_i = y \rightarrow f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, y, \ldots, x_n)$
(Ax.eq.pred)	For every n-ary predicate symbol P and for all $1 \leq i \leq n$
	$\forall x_1 \ldots \forall x_n \forall y.\ x_i = y \rightarrow [P(x_1, \ldots, x_i, \ldots, x_n) \leftrightarrow P(x_1, \ldots, y, \ldots, x_n)]$

The first three should have a version for each sort. The last two axiom schemata need to be relativized to the appropriate sorts of the function and predicate symbols.

Set and Size Axioms: We need axioms about sets of individuals but not higher order sets:

(Ax.ext)	$\forall x^s \forall y^s.[x = y \leftrightarrow \forall z^e.(z \in x \leftrightarrow z \in y)]$	(set extensionality)
(Ax.empty)	$\forall x^e.\neg[x \in \emptyset]$	
(Ax.sing)	$\forall x^e \forall y^e.[y \in \{x\} \leftrightarrow y = x]$	
(Ax.union)	$\forall x^s \forall y^s \forall z^e.[(z \in x \cup y) \leftrightarrow (z \in x \vee z \in y)]$	
(Ax.intersect)	$\forall x^s \forall y^s \forall z^e.[(z \in x \cap y) \leftrightarrow (z \in x \wedge z \in y)]$	
(Ax.diff)	$\forall x^s \forall y^s \forall z^e.[(z \in x - y) \leftrightarrow (z \in x \wedge \neg(z \in y))]$	
(Ax.inclusion)	$\forall x^s \forall y^s.[x \subseteq y \leftrightarrow \forall z^e.(z \in x \rightarrow z \in y)]$	
(Ax.inclusion2)	$\forall x^s \forall y^s.[x \subset y \leftrightarrow (x \subseteq y \wedge \neg(x = y))]$	

These guarantee that we have the empty set, that we have a singleton set for each individual element, and that the union of any two sets exists. The axiom of extensionality guarantees that any two sets are considered the same iff they have the same individual members.

We also need axioms about set sizes:[7]

[7]The first axiom can be proved from the other two and the rest of the axioms.

(Ax.empty.size) $|\emptyset| = 0$
(Ax.sing.size) $\forall x^e.|\{x\}| = 1$
(Ax.set.size) $\forall x^s \forall y^s.[x \cap y = \emptyset \rightarrow |x \cup y| = |x| + |y|]$

Number Axioms: We need axioms about numbers.

(Ax.zero) $\neg\exists x^n.\ x + 1 = 0$
(Ax.add.zero) $\forall x^n.\ x + 0 = x$
(Ax.add.comm) $\forall x^n \forall y^n.\ x + y = y + x$
(Ax.add.assoc) $\forall x^n \forall y^n \forall z^n.\ (x + y) + z = x + (y + z)$
(Ax.ord.add) $\forall x^n \forall y^n.\ x < y \leftrightarrow [\exists z^n.\ x + z = y \wedge \neg(z = 0)]$
(Ax.ord.leq) $\forall x^n \forall y^n.\ x \leq y \leftrightarrow [(x < y) \vee (x = y)]$

In addition, we may add an infinite supply of number constants $2, 3, 4, \ldots$ together with all facts about them, such as $1 + 1 = 2$, $1 < 2$, etc. Or instead, we could represent all numbers using the successor function s, where 1 is $s(0)$, 2 is $s(s(0))$, etc., and add axioms:

(Ax.succ) $\forall x^n.\ s(x) = x + 1$
(Ax.add1) $\forall x^n \forall y^n.s(x) = s(y) \rightarrow x = y$

Translation from L^{MR} to L^{IMP}

The first step of the translation is to extend L^{MR} with numerical comparators and the set size function $|\cdot|$, with their intended interpretations, and then expand the special forms we introduced in L^{MR} such as *every*, *most*, and *num-compar* to equivalent and more basic expressions:

(499) $tr(every(A, B)) = \forall x.A(x) \rightarrow B(x)$
$tr(most(A, B)) = |tr(A) \cap tr(B)| > |tr(A) - tr(B)|$
$tr(num\text{-}compar(Q, n, A, B)) = Q(|tr(A) \cap tr(B)|, n)$
$tr(num\text{-}compar2(Q, A, B, C)) = Q(|tr(A) \cap tr(B)|, |tr(A) \cap tr(C)|)$
etc.

The remaining obstacle is lambda expressions. We can get rid of some of them directly by applying them on an individual term and performing β-reduction, as in section 2.8.2. However, we cannot do this for all of them directly because they are also used to name sets

which participate in larger expressions, either forming larger set expressions using $\cap, \cup, -$, or yielding the size of a set using $|\cdot|$.

We can treat each lambda expression as a special kind of term of sort s, which names a set. To remind us of this fact, we extend L^{IMP} with an infinite supply of constant symbols and function symbols as follows:

- If P is a unary predicate symbol of L^{IMP} of type et then $\lfloor P \rfloor$ is an object symbol of type s.
- If φ is a formula (of L^{IMP}) with exactly one free variable x of type e, then $\lfloor_x \lambda x.\varphi \rfloor$ is an object symbol of type s.
- If φ is a formula with the free variables $x, y_1, \ldots, y_m$, each of type e, then $\lfloor_x \lambda x.\varphi_{\ y_1,\ldots,y_n} \rfloor$ is a function symbol of type $\underbrace{e \times \ldots \times e}_{m} \to s$.

For this to be meaningful, we need to be able to read into the name of the new symbols. We can do this using axiom schemata:

- If P is a unary predicate symbol of L^{IMP} of type et, then the following is an axiom:
  $\forall y^e.\ y \in \lfloor P \rfloor \leftrightarrow P(y)$
- If φ is a formula with exactly one free variable x of type e, then the following is an axiom:
  $\forall y^e.\ y \in \lfloor_x \lambda x.\varphi \rfloor \leftrightarrow \varphi[y/x]$
- If φ is a formula with the free variables $x, y_1, \ldots, y_m$, each of type e, then the following is an axiom:
  $\forall y_1^e \ldots \forall y_n^e.[\forall y^e.\ y \in \lfloor_x \lambda x.\varphi_{\ y_1,\ldots,y_n} \rfloor(y_1, \ldots, y_n) \leftrightarrow \varphi[y/x]]$

For readability, I will use shorter constant and function names instead of the long ones.

For any particular query, we will only need a finite number of instances of these axiom schemata.

10.10.4 Examples of Inferences

Example 1

(500) At least nine French students arrived.
Most students sneezed.
⇒ At least five students sneezed.

This is translated using the glue specification to:[8]

(501) a. $\textit{num-compar}(\geq, 9, \lambda x.\textit{student}(x) \wedge \textit{french}(x), \textit{arrive})$
b. $\textit{quant}(\textit{most}, \textit{student}, \textit{sneeze})$
c. $\textit{num-compar}(\geq, 5, \textit{student}, \textit{sneeze})$

We first expand these to more basic expressions:

(502) a. $|\lambda x.\textit{student}(x) \wedge \textit{french}(x) \cap \textit{arrive}| \geq 9$
b. $|\textit{student} \cap \textit{sneeze}| > |\textit{student} - \textit{sneeze}|$
c. $|\textit{student} \cap \textit{arrive}| \geq 5$

There are four set expressions: $\lambda x.\textit{student}(x) \wedge \textit{french}(x)$, *student*, *arrive*, and *sneeze*. We need to translate them to the object constants of type s, namely: $\lfloor_x \textit{student}(x) \wedge \textit{french}(x)\rfloor$, $\lfloor\textit{student}\rfloor$, $\lfloor\textit{arrive}\rfloor$ and $\lfloor\textit{sneeze}\rfloor$. For convenience, let us shorten these to *fst*, *st*, *sn*, and *ar*. So we finally get these representations in L^{IMP}:

(503) a. $|\textit{fst} \cap \textit{ar}| \geq 9$
b. $|\textit{st} \cap \textit{sn}| > |\textit{st} - \textit{sn}|$
c. $|\textit{st} \cap \textit{ar}| \geq 5$

In addition, we instantiate the axiom schemata for the four set constants:

(504) a. $\forall y^e.\ y \in \textit{st} \leftrightarrow \textit{student}(y)$
b. $\forall y^e.\ y \in \textit{ar} \leftrightarrow \textit{arrive}(y)$
c. $\forall y^e.\ y \in \textit{sn} \leftrightarrow \textit{sneeze}(y)$
d. $\forall y^e.\ y \in \textit{fst} \leftrightarrow \textit{student}(y) \wedge \textit{french}(y)$

Now we are ready to show the proof of (503)c from (503)a-b. We negate the goal and derive a contradiction.

[8] I simplify events here, but a similar formalization and proof is possible with events.

(505)

1.	$\|fst \cap ar\| \geq 9$	(503)a
2.	$\|st \cap sn\| > \|st - sn\|$	(503)b
3.	$\|st \cap sn\| < 5$	negated (503)c
4.	$\|st \cap sn\| \leq 4$	from 3 and lemma
5.	$\|st - sn\| \leq 3$	from 2,4 and lemma (transitivity of $<$)
6.	$\|st \cap sn\| + \|st - sn\| \leq 7$	from 4,5 and lemma
7.	$st = (st \cap sn) + (st - sn)$	lemma
8.	$\|st\| \leq 7$	from 6,7 and lemma
9.	$\|st\| \geq \|st \cap ar\|$	by lemma
10.	$fst \cap ar \subseteq fst$	by lemma
11.	$\|fst\| \geq 9$	from 1,10 and lemma
12.	$fst \subseteq st$	from (504)a,d and lemma
13.	$\|st\| \geq 9$	from 11,12 and lemma
14.	$9 \leq 7$	from 8,13 and lemma
15.	contradiction	from 14 and number aximos

All the lemmas are provable from the axioms and premises. For example, the proof of the lemma in line 12:

(506)

1.	$\forall y^e.\ y \in st \leftrightarrow student(y)$	(504)a
2.	$\forall y^e.\ y \in fst \leftrightarrow student(y) \wedge french(y)$	(504)d
3.	$\forall y^e.\ student(y) \wedge french(y) \rightarrow student(y)$	tautology
4.	$\forall y^e.\ y \in fst \rightarrow y \in st$	from 1,2,3, and a few basic steps
5.	$fst \subseteq st$	from 4 and (Ax.inclusion)

This shows why we need to be able to read into the set constants – so that facts like line 3 in (506) can be used to determine relationships between the set constants.

This is quite a long proof and it also rests on lemmas. However, notice that if we changed the numbers in (500) to 90 and 50, the proof would still be of exactly the same length, and would look exactly the same except for the numbers.[9]

Relying on a general-purpose FOL reasoner to find such proofs based on the axioms of sets and numbers may be impractical as the search space may be too large. We may want to rely on special-purpose reasoners that can reason more efficiently with sets and their sizes. But at least I have shown here a proof-of-concept reasoning solution.

If we changed the conclusion line in (500) to "At least six students sneezed", it would no longer follow from the assumptions. A model builder that knows how to deal with

[9]This assumes that it costs us the same to calculate the facts $5 < 9$ and $50 < 90$.

numbers (i.e. it does not try to instantiate infinitely-many natural numbers) would find a counter model. An example of such a model M is:

(507) $[\![st]\!]^M = [\![fst]\!]^M = [\![st]\!]^M = \{e1, e2, e3, e4, e5, e6, e7, e8, e9\}$
$[\![sn]\!]^M = \{e1, e2, e3, e4, e5\}$

It is very expensive for a model builder to reason like that because of the size of models and of the search space. This also becomes less and less efficient as the numbers we use are larger, even though humans can reason with 50 and 90 just as easily as 5 and 9. A better solution would be to represent set sizes as integer variables, to use knowledge about sets to derive inequalities about these variables, and reason about these inequalities.

Example 2

(508) a. At most two sculptures are exhibited in room 1.
b. Sculpture C is exhibited in room 1.
c. Sculpture D is exhibited in room 1.
d. Hence, sculpture E is not exhibited in room 1.

Formalization in L^{MR}: [10]

(509) a. $\mathit{quant\text{-}num}(\leq, 2, \mathit{sculpture}, \lambda x.\mathit{exhibited\text{-}in}(x, R1))$
b. $\mathit{exhibited\text{-}in}(C, R1)$
c. $\mathit{exhibited\text{-}in}(D, R1)$
d. $\neg\mathit{exhibited\text{-}in}(E, R1)$

(509)a is expanded to:

(510) $|\mathit{sculpture} \cap \lambda x.\mathit{exhibited\text{-}in}(x, R1)| \leq 2$

We use sc as a shorthand for the set constant $\lfloor \mathit{sculpture} \rfloor$, and $er1$ as a shorthand for the set constant $\lfloor_x \lambda x.\mathit{exhibited\text{-}in}(x, R1)\rfloor$. So we have:

(511) a. $|sc \cap er1| \leq 2$
b. $\forall y^e.\ y \in sc \leftrightarrow \mathit{sculpture}(y)$
c. $\forall y^e.\ y \in er1 \leftrightarrow \mathit{exhibited\text{-}in}(y, R1)$

[10] $\mathit{exhibited\text{-}in}(x, y)$ is a shorthand for $\exists e.\mathit{exhibit}(e) \wedge \mathit{obj}(e, x) \wedge \mathit{in}(e, y)$.

We write $\{C, D, E\}$ as a shorthand for $\{C\} \cup \{D\} \cup \{E\}$.

(512) Proof by contradiction:

1.	*exhibited-in*$(E, R1)$	negated goal
2.	$\forall x^e.\ x \in \{C\} \cup \{D\} \cup \{E\} \rightarrow x = C \vee x = D \vee x = E$	lemma
3.	$\forall x^e.\ x = C \vee x = D \vee x = E \rightarrow$ *exhibited-in*$(E, R1)$	from 1, (509)b,c
4.	$\forall x^e.\ x \in \{C, D, E\} \rightarrow$ *exhibited-in*$(x, R1)$	from 2,3
5.	$\forall x^e.\ x \in \{C, D, E\} \rightarrow x \in er1$	from 4, (511)c
6.	$\{C, D, E\} \subseteq er1$	from 5, (Ax.inclusion)
7.	$sculpture(C) \wedge sculpture(D) \wedge sculpture(E)$	assumption [11]
8.	$\{C, D, E\} \subseteq sc$	from 7, similar method
9.	$\{C, D, E\} \subseteq sc \cap er1$	from 6,8, lemma
10.	$\lvert\{C, D, E\}\rvert \leq 2$	from 9, (511)a, lemma
11.	$C \neq D \wedge C \neq E \wedge D \neq E$	unique names assumption
12.	$\lvert\{C, D, E\}\rvert = 3$	from 11, lemma
13.	$3 \leq 2$	from 10,12, equality
14.	contradiction	from 13, number axioms

[11] This material is not part of the truth conditions of (508) but is a presupposition there.

Chapter 11

Gradable Comparatives

In the previous chapter I discussed numerical comparatives, and in this chapter I will discuss gradable comparatives, as in the following examples from logic puzzle texts:

(513) a. Alfonso is six feet tall.

b. The Physics textbook is heavier than the Math textbook.

c. J is more popular than Q.

d. Fazio's flight is earlier than Simon's flight.

e. Flight 105 must depart earlier than Flight 103.

f. Wanda is assigned to a lower-numbered bench than Joan is.

These sentences compare two degrees. The first is the degree to which one object satisfies a predicate. The second degree is either given explicitly or it is the degree to which another object satisfies a predicate.

There are various cases of such constructions, and there are parallels between them and the numerical comparatives from the previous chapter, especially regarding interaction with ellipsis. I am covering gradable comparatives because they have some additional issues and interesting variations. I first dealt with numerical comparatives because they are simpler. Their semantics is more clear than that of gradable comparatives. The latter have three families: predicative adjectives, attributive adjectives, and adverbs, rather than just one. And because of the more limited structure of the copula construction, there are fewer opportunities for predicative adjectives to interact with ellipsis, so for them, it is more tricky to delineate the borderline between semantic composition and ellipsis than for numerical comparatives.

11.1 Copula

Before we proceed, we need to extend the glue specification to handle predicative adjectives, and more generally, copula constructions, i.e. verbs that take an adjectival phrase or a prepositional phrase as argument. Such verbs include: *be*+AP, *seem*+AP, *become*+AP, *consider*+NP+AP. For example:

(514) The box is empty.

$$\begin{bmatrix} \text{PRED} & \text{BE}\langle \boxed{2} \rangle \boxed{1} \\ \text{SUBJ} & \boxed{1}\begin{bmatrix} \text{PRED} & \text{BOX} \\ \text{SPEC} & \begin{bmatrix} \text{DET} & \begin{bmatrix} \text{PRED} & \text{`THE'} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{XCOMP-PRED} & \boxed{2}\begin{bmatrix} \text{PRED} & \text{EMPTY}\langle \boxed{1} \rangle \\ \text{SUBJ} & \boxed{1} \\ \text{ATYPE} & \text{PREDICATIVE} \end{bmatrix} \end{bmatrix}$$

(515) John is in London.

$$\begin{bmatrix} \text{PRED} & \text{BE}\langle \boxed{2} \rangle \boxed{1} \\ \text{SUBJ} & \boxed{1}\begin{bmatrix} \text{PRED} & \text{JOHN} \end{bmatrix} \\ \text{XCOMP-PRED} & \boxed{2}\begin{bmatrix} \text{PRED} & \text{IN}\langle \boxed{1}, \boxed{3} \rangle \\ \text{SUBJ} & \boxed{1} \\ \text{OBJ} & \boxed{3}\begin{bmatrix} \text{PRED} & \text{LONDON} \end{bmatrix} \\ \text{PTYPE} & \text{SEM} \end{bmatrix} \end{bmatrix}$$

It is tempting to produce the following representations:

(516) a. $empty(the(box))$
b. $in(john, london)$

However, this is not general enough. Instead, I would like the representations:

(517) a. $\exists e.be(e) \wedge subj(e, the(box)) \wedge pred(e, \lambda x.empty(x))$
b. $\exists e.be(e) \wedge subj(e, john) \wedge pred(e, \lambda x.in(x, london))$

because I want to have the event variable so that it could be modified by time and other modifiers (as in: "John *was* in London *yesterday*").

The basic specifications for prepositional phrases and adjectives in sections 4.4.3 and 4.6.1 remain the same. There, we had rules that added an intersection that combined the meaning of the modifier with the meaning of the noun or verb-predicate. Here, we use a different binder:

```
+ATYPE(%F,predicative), +XCOMP-PRED(%Verb,%F)
     ==> glue_verb_arg_predicative(%Verb,%F).

preposition(%F), +XCOMP-PRED(%Verb,%F)
     ==> glue_verb_arg_predicative(%Verb,%F).

glue_verb_arg_predicative(%Verb,%A) :=
      X\P\V\[and,[P,V], xle_gf(V,pred,X)] :
        pred_expr(%A) -> verb_mod_expr(%Verb) : [noscp].
```

In addition, notice that in (514) and (515), the SUBJ is not a thematic argument of 'be', so it will not be handled by the default `verb_arg` as defined in section 4.4.4. The same is true for the OBJ argument 'Mary' in "John considers Mary happy." The XLE produces the F-structures above because as far as thematic roles are concerned, these elements are the SUBJ of the XCOMP-PRED complement and not the main verb (e.g. *be* is a state verb that does not have an agent role). However, since I want the representations in (517), I need to revise the definition of `is_arg` as follows:

```
is_arg(%VNode,%ArgPos,%ArgNode) :=
   arg(%VNode,%ArgPos,%ArgNode) |
   (-arg(%VNode,%%,%ArgNode),
    +XCOMP-PRED(%VNode,%XP),
    +SUBJ(%XP,%ArgNode)) .
```

The first case is what we had before. The second option is a non-thematic argument which is the `subj` of a predicative argument of the verb. Using this, we will get the missing argument as a normal argument, and this will work for passive voice as well.

11.2 Predicative Gradable Adjectives

11.2.1 Direct Degree

In order to handle:

(518) John is six feet tall.

we need to extend our L^{MR} as follows. We add a new sort d for degrees. A member of D^d_M is a tuple with two components $\langle v, u \rangle$, where v is a value and u is a measurement unit

name. For example, the meaning of "six feet" is $\langle 6, foot\rangle$. Degrees on the same scale can be compared, even if their units are different (e.g., $\langle 1, foot\rangle =_{deg} \langle 12, inch\rangle$).

We also extend the language by adding expressions that name attributes (such as *tall*). There is no need to allow quantification on them. We will just assume that our models come with a function map_M which takes as arguments an object of type e and an attribute name, and returns an element of type d, which is the degree to which the object has the attribute. For example, if John is 6 feet tall in model M then $map_M(I_M(john), tall) = \langle 6, foot\rangle$.

In addition, we will add several operators to the language. The first one is: *deg-compar*(Q, P, x, a), where Q is one of $=, \geq, >, \leq, <$, P is the name of an attribute, x is of type e, and a is of type d. The meaning is given by:

(519) $[\![\textit{deg-compar}(Q, P, x, a)]\!]^g_M = true$ iff $Q(map_M([\![x]\!]^g_M, P), [\![a]\!]^g_M)$

Thus, (518) can be represented using:

(520) $\exists e.be(e) \wedge subj(john, e) \wedge pred(e, \lambda x.\textit{deg-compar}(=, tall, x, \langle 6, foot\rangle))$

Below, we will use a shorthand for this for brevity:

(521) $be(john, \lambda x.\textit{deg-compar}(=, tall, x, \langle 6, foot\rangle)$

The F-structure for (518) is like the one in (514) except that XCOMP-PRED has the value:

(522)
$$\begin{bmatrix} \text{PRED} & \text{TALL}\langle \boxed{1} \rangle \\ \text{SUBJ} & \boxed{1} \\ \text{ATYPE} & \text{PREDICATIVE} \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{FOOT} \\ \text{NTYPE} & [\text{NSEM}\ [\text{COMMON}\ \text{`MEASURE'}]] \\ \text{SPEC} & [\text{NUMBER}\ [\text{PRED}\ \text{`SIX'}]] \end{bmatrix} \right\} \end{bmatrix}$$

The F-structure for:

(523) John is at least six feet tall.

is like (522) but under SPEC|NUMBER there is:

(524)
$$\begin{bmatrix} \text{PRED} & \text{SIX} \\ \text{ADJUNCT} & \left\{ \left[\text{PRED}\ \text{`AT LEAST'} \right] \right\} \end{bmatrix}$$

Then the representation is:

(525) $be(john, \lambda x.deg\text{-}compar(\geq, tall, x, \langle 6, foot \rangle))$

Similarly if we replace *at least* with *exactly*, *more than*, etc.

The glue specification is the following (some macros were defined in chapter 10):

```
adj_with_degree(%F,%Rel,%Pred,%Number,%Measure)
        ==> deg_compar_sem(%F,%Rel,%Pred,%Number,%Measure).

deg_compar_sem(%F,%Rel,%Pred,%Number,%Measure) :=
        X\deg_compar(%Rel,%Pred,X,[%Number,%Measure]) :
           pred_expr(%F).

adj_with_degree(%F,%Rel,%Pred,%Number,%Measure) :=
        base_adj_modified(%F,%Pred,%M),
        path(%M,NTYPE,NSEM,COMMON,measure),
        +PRED(%M,%Measure),
        +SPEC(%M,%Spec),
        path(%Spec,NUMBER,PRED,%NW),
        number_word_number(%NW,%Number),
        calc_quant_num_rel(%Spec,%Rel).

base_adj_modified(%Node,%Pred,%M) :=
        +ATYPE(%Node,%%),
        +PRED(%Node,%Pred),
        modifier_of(%M,%Node).
```

11.2.2 Comparison to NP

Consider:

(526) John is taller than Mary.

Like (518), the F-structure under XCOMP-PRED has a PRED of TALL, and an ADJUNCT. This time, the ADJUNCT contains a structure similar to the comparative expressions we saw in the previous chapter. So, instead of [2] in (514), we have:

(527)
$$\boxed{2}\begin{bmatrix}\text{PRED} & \text{TALL}\langle\boxed{1}\rangle \\ \text{SUBJ} & \boxed{1} \\ \text{ATYPE} & \text{PREDICATIVE} \\ \text{ADJUNCT} & \left\{\begin{bmatrix}\text{PRED} & \text{`MORE}\langle\boxed{3}\rangle\text{'} \\ \text{OBL-COMPAR} & \boxed{3}\begin{bmatrix}\text{PRED} & \text{THAN}\langle\boxed{4}\rangle \\ \text{OBJ} & \boxed{4}\begin{bmatrix}\text{PRED} & \text{`MARY'}\end{bmatrix}\end{bmatrix}\end{bmatrix}\right\}\end{bmatrix}$$

Notice how the XLE's morphology and syntax take care of analyzing *taller* as *more tall*. Instead of *more ... than*, we could have any of the comparators *less ... than*, *as ... as*, *no more ... than*, *at least as ... as*, etc.

The desired representation is:

(528) $be(john, \lambda x.deg\text{-}compar2(>, tall, x, mary))$

The new form *deg-compar2* can be defined in terms of *deg-compar*:

(529) $deg\text{-}compar2(Q, P, x, y) \equiv deg\text{-}compar(Q, P, x, \iota z.deg\text{-}compar(Q, P, y, z))$

(However, in queries were we only need to compare the relative heights of people without knowing their actual heights, using only *deg-compar2* can allow the reasoning to be more efficient than using *deg-compar*.)

The glue specification is as follows:

```
adj_compar_np(%F,%Rel,%Pred,%Obj)
    ==> deg_compar2_sem(%F,%Rel,%Pred,%Obj).

deg_compar2_sem(%F,%Rel,%Pred,%Obj) :=
        Y\X\deg_compar2(%Rel,%Pred,X,Y) :
          %Obj$e -> pred_expr(%F).

adj_compar_np(%F,%Rel,%Pred,%Obj) :=
    base_adj_modified(%F,%Pred,%M),
    +PRED(%M,%Compar),
    rel_deg_compar(%Compar,%Rel),
    path(%M,OBL-COMPAR,OBJ,%Obj),
    +NTYPE(%Obj,%%).

 |- rel_deg_compar(more,'>').
 |- rel_deg_compar(less,'<').
 |- rel_deg_compar(as,'=').
```

11.2.3 Clausal Comparatives

There are three cases, which are somewhat similar to the cases 3-5 in the previous chapter:

(530) a. John is taller than Mary was.

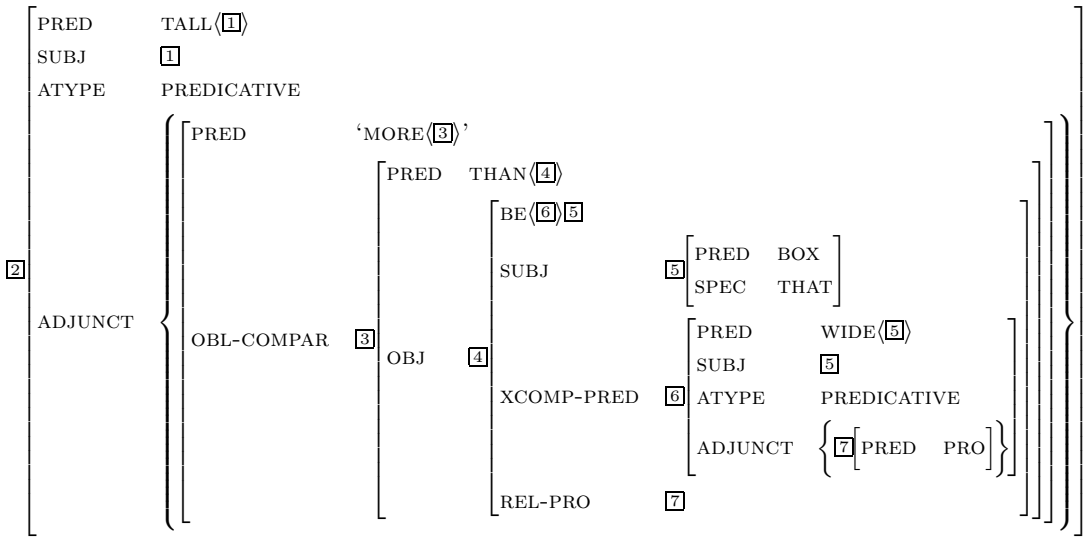
b. John is taller than Bill thought.

c. This box is taller than that box is wide.

Clause Missing a Degree

The situation in (530)c is similar to the one we saw in section 10.6. There, the clausal complement was missing a quantifier. Here, it is missing a degree:

(531) taller than that Box is _ wide



Similarly to what we saw previously, the meaning of 4 ends up being:

(532) $\lambda y.be(that(box), \lambda x.deg\text{-}compar(=, wide, x, y)$

i.e., the property of being the degree to which that box is wide. In this case, (530)c gets the representation:

(533) $be(this(box), \lambda x.deg\text{-}compar5(>, tall, x, \varphi))$

where φ = (532)

The operator *deg-compar5* can be defined in terms of *deg-compar*:

(534) *deg-compar5*$(Q, P, x, S) \equiv$ *deg-compar*$(Q, P, x, \iota z.S(z))$

i.e. we take the property S and find the single degree that satisfies it.

The details of the glue specification are very similar to those at the beginning of this chapter, modified along the lines of the specifications for the clausal cases in sections 10.4-10.6. For example, the case of a clause missing a degree has the following specification. First, the specification in section 11.2.1 will not work for the inner adjective that is missing its modifying degree (e.g. 6 in (531)), and so we need:

```
adj_with_degree_mis(%F,%Pred,%DegNode)
        ==> deg_compar_sem_mis(%F,%Pred,%DegNode).

deg_compar_sem_mis(%F,%Pred,%NumNode) :=
        Y\X\deg_compar(=,%Pred,X,Y) :
           %NumNode$d -> pred_expr(%F).

adj_with_degree_mis(%F,%Pred,%M) :=
        base_adj_modified(%F,%Pred,%M),
        +PRED(%M,PRO).
```

Then we need the specification for the comparative adjective (e.g. 2 in (531)):

```
adj_compar_clause(%F,%Rel,%Pred,%Clause,%Missing,n)
        ==> adj_compar5_sem(%F,%Rel,%Pred,%Clause,%Missing).

adj_compar5_sem(%F,%Rel,%Pred,%Clause,%Missing) :=
        P\X\deg_compar5(%Rel,%Pred,X,P) :
          (%Missing$n -> %Clause$t) -> pred_expr(%F).

adj_compar_clause(%F,%Rel,%Pred,%Clause,%Missing,%MissingType) :=
        base_adj_modified(%F,%Pred,%M),
        +PRED(%M,%Compar),
        rel_deg_compar(%Compar,%Rel),
        path(%M,OBL-COMPAR,OBJ,%Compared),
        +REL-PRO(%Compared,%Missing),
        (( in_set(%Missing,%%), {%MissingType=d} ) |
         ( +qp(%Feat,[%%,%Missing]), -{%Feat=REL-PRO}),
           compar_missing_type(%Feat,%MissingType) )).
```

The macro `compar_missing_type` was defined in section 10.4. Here we need additionally (for the case of a clause missing an AP that will be below):

```
|- compar_missing_type(XCOMP-PRED,a).
```

Clause Missing a Clause

For (530)b, the F-structure is the same as in (531), except that now [4] is:

(535)
$$\begin{bmatrix} \text{PRED} & \text{THINK}\langle [5], [7] \rangle \\ \text{SUBJ} & [5]\begin{bmatrix}\text{PRED} & \text{'BILL'}\end{bmatrix} \\ \text{COMP} & [7]\begin{bmatrix}\text{PRED} & \text{PRO}\end{bmatrix} \\ \text{REL-PRO} & [7] \end{bmatrix}$$

The meaning of "than Bill thought" is (ignoring tense):

(536) $\lambda z.think(bill, z)$

Now, (530)b can be represented using:

(537) $be(this(box), \lambda x.deg\text{-}compar4(>, tall, x, \varphi))$
where $\varphi =$ (536)

The operator *deg-compar4* can be defined in terms of *deg-compar*:

(538) $deg\text{-}compar4(Q, P, x, S) \equiv deg\text{-}compar(Q, P, x, \iota y.S(\langle\langle deg\text{-}compar(=, P, x, y)\rangle\rangle))$

We apply (536) on the missing proposition $\langle\langle deg\text{-}compar(=, tall, x, y)\rangle\rangle$, i.e. the proposition that x is y-tall. Thus, (537) says that the degree to which this box is tall is larger than the degree y such that Bill things that x (this box) is tall to degree y.

Clause Missing an AP

For (530)a, the first question to ask is whether it is obtained by some form of ellipsis from "John is taller than Mary was tall" or whether it is a separate case. The answer is the second, as the next section will discuss. The F-structure is the same as in (531), except that [5] is [PRED 'MARY'], and instead of [6] we have just [7][PRED PRO] directly. The meaning of "than Mary was" is (ignoring tense):

(539) $\lambda R.be(mary, R)$

This is the meaning of the sentence "Mary was R", where R is abstracted over. Now, (530)a can be represented using:

(540) $be(john, \lambda x.deg\text{-}compar3(>, tall, x, \varphi))$
where φ = (539)

The operator *deg-compar3* can be defined in terms of *deg-compar*:

(541) $deg\text{-}compar3(Q, P, x, R) \equiv deg\text{-}compar(Q, P, x, \iota y.R(\lambda z.deg\text{-}compar(=, P, z, y)))$

We apply (539) on the missing property $\lambda z.deg\text{-}compar(=, tall, z, y)$, i.e. the property of being y-tall. Thus, (540) says that the degree to which John is tall is larger than the degree y such that Mary was tall to degree y.

Note that long distance dependency in such sentences works correctly. The missing adjective phrase could be deeply embedded, as shown in (542)a. The subdeletion case (542)b is also theoretically possible even if it is awkward in practice.

(542) a. John is taller than I thought Bill was.
b. John is taller than I thought Bill was wide.

11.2.4 Variations

As with numeric comparatives in section 10.7, we can get:

(543) John is at most six inches taller than Mary.

and similarly with the other cases above. The F-structure for (543) is the same as for (526), except that now the ADJUNCT of (527) has an additional member:

(544)
$$\begin{bmatrix} \text{PRED} & \text{INCH} \\ \text{NTYPE} & \left[\text{NSEM} \quad \left[\text{COMMON} \quad \text{MEASURE}\right]\right] \\ \text{SPEC} & \left[\text{NUMBER} \quad \left[\begin{matrix}\text{PRED} & \text{SIX} \\ \text{ADJUNCT} & \left\{\left[\text{PRED} \quad \text{AT LEAST}\right]\right\}\end{matrix}\right]\right] \end{bmatrix}$$

We can represent this with:

(545) $be(john, \lambda x.deg\text{-}diff2(\leq, \langle 2, foot\rangle, tall, x, mary))$

where:

(546) $\textit{deg-diff2}(Q, a, P, x, y) \equiv Q((\iota z_1.\textit{deg-compar}(=, P, x, z_1) - \iota z_2.\textit{deg-compar}(=, P, y, z_2)), a)$

This gives us:

(547) $be(john, \lambda x.[(\iota z_1.\textit{deg-compar}(=, tall, x, z_1) - \iota z_2.\textit{deg-compar}(=, tall, mary, z_2)) \leq \langle 2, foot\rangle])$

Similar remarks apply to:

(548) John is at least three times taller than Mary.

and similar sentences.

11.2.5 Why So Many Cases?

Do we really have so many different cases? This question is important in continuation to section 10.8. We want to know whether certain cases fall out naturally from the interaction of non-elided comparatives and independently-motivated ellipsis. Cases that do should not be treated by additional ad-hoc special cases.

One evidence that we do have the different cases above is that they parallel the cases we saw for numeric comparatives. Further arguments about the the following four will be presented below.

(549) a. John is taller than 6 feet.
b. John is taller than Bill.
c. John is taller than Bill is.
d. John is taller than Bill is tall.

Degree Comparative vs. Phrasal Comparative

The first question is whether (549)b is a metonymy of (549)a. In other words, is (549)b a way to say "John is taller than [some relevant property of] Bill", where the relevant property is "the tallness of"? For some evidence against that view, notice the following differences:

(550) a. John is (3 inches) taller than Bill.
b. ? John is (3 inches) taller than 6 feet.

(551) a. Who is John taller than?
b. ? How many feet is John taller than?
c. It is Bill that John is taller than.
d. ? It is 6 feet that John is taller than.

Phrasal Comparative vs. Clausal Comparative

Is the phrasal case (549)b an elided form of the clausal case (549)c? First, there is evidence that this cannot be the *only* way that (549)b is created. This is because "taller than Bill" is an independent constituent in some sentences:

(552) a. Jonn seems taller than Bill.
b. John became taller than Bill.
c. John considered Mary taller than Bill.

The verbs *seem*, *become*, and *consider* take an adjective phrase as argument. Moreover, these sentences mean the same as:

(553) a. Jonn seems taller than Bill is.
b. John became taller than Bill was (tall).
c. John considered Mary taller than Bill was (tall).

The *is/was* in these sentences cannot be elided because the verb *to be* does not appear anywhere before. Therefore, since "taller than Bill" must be an independent constituent in (552), it is likely that it could be so in (549)b as well.

Also note that the sentences in (552) are in fact ambiguous between the meanings given in (553) and an elided form where *did* is elided. For example, (552)b, in addition to the analysis above, also has an analysis where *did* is elided from (554)a using stripping (see (421)e-f). However, (554)a means (554)b and not (553)b.

(554) a. John became taller than Bill did.
b. John became taller than Bill became.

Kennedy (1997) presents arguments that (549)b *cannot* be an elided form of (549)c, but I do not think his arguments are very strong. In fact, since (552) has an ellipsis derivation where *did* is elided, it is reasonable that (549)b has an ellipsis derivation too, where *is/was* is elided.

To summarize, (549)b has two meanings. One meaning is expressed in (554), and this meaning arises when (549)b is obtained by stripping from (554)a. The second meaning of (549)b is expressed in (549)c, and there are two ways in which this meaning arises. One way is by stripping from (549)c to get (549)b, and the other way is directly from a separate case of comparative complement (*deg-compar2*), which is also needed for cases like (552).

Comparative Deletion vs. Comparative Subdeletion

Kennedy (1997) shows evidence that (549)c is not an elliptical form of (549)d, i.e. that "than Bill is" does not have an ellipsis of the AP "tall".

First, assume to the contrary that (549)c is an elided form of (549)d, shown again here:

(555) John is [$_{\text{AP}}$ taller than [$_{\text{S}}$ Bill is [$_{\text{AP}}$ tall]]].

The problem is that if (549)c has an ellipsis, the elided AP should be *taller*, like the antecedent, rather than *tall*. It cannot be argued that the ellipsis mechanism can ignore this difference because such leniency does not appear in other cases of ellipsis. For example, in (556), the missing material can only be "more useful" and not just "useful".

(556) The space telescope was more useful this year, and the gamma ray satellite was $\triangle$, too.

Similarly, if the comparative is modified by a degree, we get the same problem:

(557) a. Smith wants the novel to be 100 pages longer than her editors do.
b. Smith wants the novel to be 100 pages longer than her editors want it to be.
c. Smith wants the novel to be 100 pages longer than her editors want it to be long.

The sentence (557)a clearly has an antecedent-contained VP-ellipsis, and is derived from (557)b. However, if we assume that (557)b is an elided form of (557)c then the problem is that the elided AP should ignore the degree modifier "100 pages". But leaving out such degrees cannot be done in general: the missing material in (558) can only be "300 pages long" and not just "long".

(558) Smith's novel will be 300 pages long, and Jones' will be $\triangle$, too.

The second argument points out that VP ellipsis can in principle take its antecedent from anywhere in the prior discourse. For example, Kennedy claims that (559)a has two possible interpretations: (559)b and (559)c:

(559) a. Marcus read every book I did and I bought every book Charles did.
b. Marcus read every book I read and I bought every book Charles bought.
c. Marcus read every book I read and I bought every book Charles read.

In contrast, (560)a can only have the interpretation (560)b but not (560)c.

(560) a. The table is wider than this rug is, but this rug is longer than the desk is.
b. The table is wider than this rug is wide, but this rug is longer than the desk is long.
c. The table is wider than this rug is wide, but this rug is longer than the desk is wide.

Kennedy claims that the reason for this difference is that "wider than this rug is" in (560)a is not obtained by ellipsis of an AP "wide", and so such an AP is not available as an antecedent for an ellipsis in the second conjunct.

This also explains why the following sentence does have both interpretations (560)b and (560)c:

(561) The table is wider than the rug is wide, but this rug is longer than the desk is.

In this sentence, the second conjunct is structurally ambiguous. One structure is the usual comparative deletion, giving the interpretation (560)b. But now since the first conjunct in (561) has comparative subdeletion and the AP "wide" is explicitly available there, the second conjunct of (561) can be obtained from the comparative subdeletion case (560)c through VP-ellipsis that elides the AP "wide" in the second conjunct, just as in:

(562) John is happy, and Bill is, too.

Ellipsis

The comparatives here interact with ellipsis in similar ways to what we saw in section 10.8, though because of the more limited structure of the copula construction, there are fewer opportunities for ellipsis. An example is:

(563) John was more hansom yesterday than [he was] intelligent a year ago.

A similar example is:

(564) John is more hansom than intelligent.
= John is more hansom than John is intelligent.

However, an argument similar to the one we discussed above regarding (552) may show that we actually have another comparative case here:

(565) John considers Mary more beautiful than intelligent.

We may want to add a case, *deg-compar6*, which expects an adjective complement:

(566) $\textit{deg-compar6}(Q, P, x, P_2) \equiv \textit{deg-compar}(Q, P, x, \iota z.\textit{deg-compar}(=, P_2, x, z))$

Just like the case in (487), it is unclear whether (567)a is obtained by stripping from (567)b, or whether extraposition from (567)c is also possible.

(567) a. John is taller today than Bill.
b. John is taller today than Bill is.
c. John is taller than Bill today.

Speculation

Why do we have all these separate cases of comparative constructions in this chapter and the previous one? Is there a commonality to all of them? In the previous chapter, the *than*-complement in all the cases is a description of a number, and in this chapter, a description of a degree. The description can be either direct, as in (433) and (518), or indirect. An indirect description can be supplied by various means, some of them are dropping some element in a clause. Given an indirect description, one wants to get a number or a degree out of it. We can speculate that if an object of type τ is expected by some operator but an object of another type is given, some general (cognitive) process can convert the latter object to an object of type τ. However, I do not know yet how to model this general process, and so for now, separate cases are needed in the glue specification.

11.3 Attributive Gradable Adjectives

In section 4.6.1, we saw the specification for the attributive case of simple adjectives, and in section 11.1 we saw the predicative case. Comparative adjectives also have these two cases, and the difference between them in the glue specification is the same as the difference between the two versions of simple adjectives.

11.3.1 Direct Degree

The sentence:

(568) A man who is 6 feet tall arrived.

has a relative clause that uses the predicative version of *tall*. It can be paraphrased using the attributive version:

(569) A 6 foot tall man arrived.

This has the F-structure:

(570)
$$\begin{bmatrix} \textsc{pred} & \textsc{man} \\ \textsc{spec} & [\textsc{det} \ [\textsc{pred} \ \text{`a'}]] \\ \textsc{adjunct} & \left\{ \begin{bmatrix} \textsc{pred} & \textsc{tall} \\ \textsc{atype} & \textsc{attributive} \\ \textsc{adjunct} & \left\{ \begin{bmatrix} \textsc{pred} & \textsc{foot} \\ \textsc{ntype} & [\textsc{nsem} \ [\textsc{common} \ \text{`measure'}]] \\ \textsc{spec} & [\textsc{number} \ [\textsc{pred} \ \text{`six'}]] \end{bmatrix} \right\} \end{bmatrix} \right\} \end{bmatrix}$$

The ADJUNCT of "man" is the attributive version of (522). The desired meaning representation is:

(571) $quant(a,\ \lambda x.man(x) \wedge \textit{deg-compar}(=, tall, x, \langle 6, foot \rangle),\ arrive)$

Nothing more needs to be added to the glue specification. This is because the specification in section 11.2.1 already takes care to add the basic contribution of the direct degree case, and the specification in section 4.6.1 already takes care to combine this contribution with the meaning of the NP.

11.3.2 Clause Missing a Degree

(572) A box that is [taller than this one is wide] cannot be inserted into the hole.
A taller box than this one is wide cannot be inserted into the hole.

This sounds fine, and is covered by the interaction of the above specification for *deg-compar5* with the specification for intersective adjectives in section 4.6.1.

A similar case is (573)a. This sentence must be obtained by normal ellipsis from (573)b, which has the same structure as (573)a, even if (573)b does not sound so good itself.

(573) a. John read a longer book than a magazine.
b. John read a longer book than he read a magazine.
c. John read a longer book than Bill read a magazine.

Although the *predicative* version of a clause missing a degree cannot apply when the degree is missing from inside a NP rather than an AP (see (574)), the cases in (573) are still covered by the specification for *deg-compar5* because all that matters is that the degree is missing.

(574) * This book is longer than Bill read a magazine.

11.3.3 Clause Missing an NP

Before we proceed with the rest of the cases from section 11.2, consider the following sentences, where the *than*-clause complement of the comparator is missing an NP:

(575) a. A man who is [taller than the/a/any man that Mary (has ever) met] jumped on the table.
b. A taller man than Mary (has ever) met jumped on the table.

(576) a. John read a book that was [more interesting than a/the/any book Mary read.]
b. John read a more interesting book than Mary read.

There is no parallel predicative version:

(577) * John is taller than Bill has ever met.

The F-structure for (576)b includes:

(578)
$$\begin{bmatrix}
\text{PRED} & \text{BOOK} \\
\text{SPEC} & [\text{DET} \ [\text{PRED} \ \text{`A'}]] \\
\text{ADJUNCT} & \left\{ \begin{bmatrix}
\text{PRED} & \text{INTERESTING} \\
\text{ATYPE} & \text{ATTRIBUTIVE} \\
\text{ADJUNCT} & \left\{ \begin{bmatrix}
\text{PRED} & \text{`MORE}\langle \boxed{3} \rangle\text{'} \\
\text{OBL-COMPAR} & \boxed{3} \begin{bmatrix}
\text{PRED} & \text{THAN}\langle \boxed{4} \rangle \\
\text{OBJ} & \boxed{4} \begin{bmatrix}
\text{PRED} & \text{READ}\langle \boxed{6}, \boxed{5} \rangle \\
\text{SUBJ} & \boxed{6}[\text{PRED} \quad \text{MARY}] \\
\text{OBJ} & \boxed{5}[\text{PRED} \quad \text{PRO}] \\
\text{REL-PRO} & \boxed{5}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix} \right\}
\end{bmatrix} \right\}
\end{bmatrix}$$

For this case, we need to define a new operator:

(579) $\textit{deg-compar7}(Q, P, x, S) \equiv \textit{deg-compar}(Q, P, x, \iota a.\textit{deg-compar}(=, P, \iota z.S(z), a))$

or perhaps:

(580) $\textit{deg-compar7}(Q, P, x, S) \equiv \textit{deg-compar}(Q, P, x, \max a.[\exists z.S(z) \wedge \textit{deg-compar}(=, P, z, a)])$

It would have been easy if the comparator simply took its clause complement as argument and produced an $e \rightarrow t$ predicate, which would then be combined with the noun's meaning using the usual intersective adjective connector from section 4.6.1. However, things are not so simple. In (576)b, we cannot simply say that the thing which John read is the intersection of $\lambda x.\textit{book}(x)$ and $\lambda x.\textit{deg-compar7}(>, \textit{interesting}, x, \lambda z.\textit{read}(\textit{mary}, z))$, because that would give us the truth conditions that John read a book that was more interesting than *something* that Mary read (not necessarily a book). The fourth argument to *deg-compar7* should be $\lambda z.\textit{book}(z) \wedge \textit{read}(\textit{mary}, z)$ and not just $\lambda z.\textit{read}(\textit{mary}, z)$. This is similar to the appearance of A twice in the definition of *num-compar3* in (448).

How can we accommodate this? Technically, there are two ways to do this in the glue specification, and they have different consequences. The first option is to have the glue specification of the comparative adjective access the PRED of the NP that the adjective modifies, and then use it. So *instead* of simply having this specification:

```
adj_compar_clause(%F,%Rel,%Pred,%Clause,%Missing,e)
        ==> adj_compar7_sem(%F,%Rel,%Pred,%Clause,%Missing).

adj_compar7_sem(%F,%Rel,%Pred,%Clause,%Missing) :=
        P\X\deg_compar7(%Rel,%Pred,X,P) :
          (%Missing$e -> %Clause$t) ->
            pred_expr(%F).
```

we have this one:

```
adj_compar_clause(%F,%Rel,%Pred,%Clause,%Missing,e),
 modifier_of(%F,%NP),
 +PRED(%NP,%Noun)
        ==> adj_compar7_sem_ex(%F,%Rel,%Pred,%Clause,%Missing,%Noun).

adj_compar7_sem_ex(%F,%Rel,%Pred,%Clause,%Missing,%Noun) :=
        P\X\deg_compar7(%Rel,%Pred,X,Z\[and,[%Noun,Z],[P,Z]]) :
```

```
          (%Missing$e -> %Clause$t) ->
            pred_expr(%F).
```

But this specification is not general enough, because in:

(581) John read a more interesting French book than Mary did.

we need to compare the French book John read to the French book that Mary read, not just to any book she read. This suggests that the comparator needs to take the `VAR(%NP)$e->RESTR(%NP)$t` meaning itself as an input:

```
adj_compar_clause(%F,%Rel,%Pred,%Clause,%Missing,e),
 modifier_of(%F,%NP)
          ==> adj_compar7_sem_ex(%F,%Rel,%Pred,%Clause,%Missing,%NP).


adj_compar7_sem_ex(%F,%Rel,%Pred,%Clause,%Missing,%NP) :=
      P\N\X\[and,[N,X],deg_compar7(%Rel,%Pred,X,Z\[and,[N,Z],[P,Z]])] :
          (%Missing$e -> %Clause$t) ->
             noun_mod_expr(%NP).
```

We also need this rule to override the default rule for intersective adjectives from section 4.6.1 (because the rule here does the work that the default rule would). We can do this by either having the rule here consume some fact and placing the rule before the default rule, or better: modify the rule in section 4.6.1 to be:

```
+ATYPE(%F,attributive), modifier_of(%F,%Noun),
 -(modifier_of(%A,%F), +OBL-COMPAR(%A,%%))
     ==> glue_noun_modifier(%Noun,%F).
```

The second line of conditions here guarantees that this rule will not operate when the attributive adjective is a comparative with a clausal complement.

11.3.4 Comparison to NP, and to a Clause Missing an AP

The sentence:

(582) A man who is taller than John jumped on the table.

has a relative clause that uses the predicative version of *tall*. It can be paraphrased using the attributive version:

(583) A taller man than John jumped on the table.

This has the F-structure:

(584)
$$\begin{bmatrix} \text{PRED} & \text{MAN} \\ \text{SPEC} & [\text{DET}\ [\text{PRED}\ \text{`A'}]] \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{TALL} \\ \text{ATYPE} & \text{ATTRIBUTIVE} \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{`MORE}\langle \boxed{3} \rangle\text{'} \\ \text{OBL-COMPAR} & \boxed{3}\begin{bmatrix} \text{PRED} & \text{THAN}\langle \boxed{4} \rangle \\ \text{OBJ} & \boxed{4}[\text{PRED}\ \ \text{`JOHN'}] \end{bmatrix} \end{bmatrix} \right\} \end{bmatrix} \right\} \end{bmatrix}$$

At first it seems that this is similar to the predicative case from section 11.2.2, and we can use *deg-compar2* without adding anything to the glue specification.

However, according to (Doherty and Schwartz, 1967; Bresnan, 1973), the sentence (585)b (when it means the same as (585)a rather than "I know a taller man than Mary does") carries the presupposition that Mary is a man while (585)a does not, and this explains why (585)a is fine while (585)b sounds odd.

(585) a. I know a man (who is) taller than Mary.

b. I know a taller man than Mary.

We can see this also in the case of a clause missing an AP:

(586) a. A man who is [more arrogant than Mary used to be] arrived.

b. A more arrogant man than Mary used to be arrived.

We can accommodate this fact in a similar way to what we did for the case of *deg-compar7* above. The difference is that we can choose to put the fact about Mary being a man not as part of the truth conditions but as a presupposition (the meaning of the noun applies to the element that the comparative adjective compares against).

Is a more explanatory analysis possible? Could "Mary" in (585)b in fact be a clause that everything has been elided from except the subject? For that to work, (585)b needs to be obtained from (587)a, which in turn would be obtained from (587)b.

(587) a. I know a taller man than Mary is.

b. I know a taller man than Mary is a man.

In order for this explanation to be valid, this kind of ellipsis should be independently-motivated, i.e. possible in non-comparative sentences. The closest thing we have is:

(588) a. John is a teacher, and Bill is a teacher too.
b. John is a teacher, and Bill is, too.
c. John is a teacher, and Bill, too.

However, going from (587)b to (587)a to (585)b is strictly speaking not a simple application of this independently-motivated ellipsis because there is no *explicit* antecedent "is a man" for the ellipsis to latch on. Perhaps we can say: when "a taller man than" is encountered, it is treated as "a thing which is a man and which is taller than" in order to get the explicit antecedent, and then ellipsis reconstruction is activated. It seems to me that this solution is not simpler than the direct solution proposed above.

11.3.5 Comparison to a Clause Missing a Clause

Now consider:

(589) a. A man who was [taller than Mary thought] arrived.
b. A taller man than Mary thought arrived.

This is covered by *deg-compar4*. However, (589)b could also mean:

(590) A man arrived who was taller than the man$_t$ that Mary thought $\__t$ arrived.

This is reminiscent of the case discussed in section 10.5. There, the second argument of the comparative quantifier (B) was also used in combination with the first argument. The problem here, however, is that *taller* is not the quantifier in the NP in which it appears, and so it does not have access to *arrive*.

One possible explanation is that (589)b is obtained from (591)a by "intraposition" (the reverse of extraposition), and (591)a is in turn obtained from (591)b by some rare case of ellipsis. The case in (591)b is a simple case of a clause missing a NP (which we handled above).

(591) a. A taller man arrived than Mary thought.
b. A taller man arrived than Mary thought $\_$ arrived.

Perhaps a simpler explanation is that the comparator here is a quantifier after all. Here is some evidence. It seems that only the determiner 'a' and the null plural determiner are possible with the attributive adjectival comparative cases here. All the sentences in (592) as well as (593)b are ungrammatical:

(592) ∗ John read every more interesting book than Mary did.
∗ John read most/few more interesting books than Mary did.
∗ John read no [more interesting books than Mary did].
∗ Every taller man than Mary thought arrived.

(593) a. John read a more interesting French book than Mary did.
b. ∗ John read a French, more interesting book than Mary did.

If this is true then the 'a' determiner is just for decoration here. If we choose to pursue this analysis, we would need to change the specification of the adjectival comparator to be like a quantifier, and to modify the specification of a quantifier in section 4.5 to not apply when the noun is modified by a comparative adjective with a clausal argument.

11.3.6 Comparison to AP

The sentence:

(594) A man who is taller than wide jumped on the table.

has a relative clause that uses the predicative version of *tall*. We can try the attributive version:

(595) A taller man than wide jumped on the table.

if this is at all possible, then it is covered by *deg-compar6* at the end of section 11.2.5 as it interacts with the specification in section 4.6.

11.3.7 Interaction with Ellipsis

The interactions of the comparative cases here with ellipsis are very similar to those we saw in chapter 10. For example, (576)b, repeated here as (596)a, can undergo VP-ellipsis to get (596)b, and that sentence in turn can undergo stripping to get (596)c (stripping is a kind of ellipsis which is independently-motivated and appears also in sentences without comparatives – see for example (421)e-f.)

(596) a. John read a more interesting book than Mary read.
b. John read a more interesting book than Mary did.
c. John read a more interesting book than Mary.

We mentioned in section 10.8.2 that some people try to explain all uses of numerical comparatives without resorting to ellipsis, but this ends up complicating the "compositional" analysis, because it essentially needs to reduplicate the mechanisms of ellipsis reconstruction. In contrast, my analysis, which relies on independently-motivated ellipsis mechanisms in the appropriate places, and distinguishes between these mechanisms and the need for separate cases of comparative complements, is simpler and covers more data.

The same conclusion is true for gradable comparatives here. Some people have tried to give a "direct" analysis. For example, Heim (1985) considers the following sentence:

(597) I have introduced better drummers to you than Karl.

According to her analysis, "the two compared items . . . somehow form a pair that is assigned scope as a pair":

(598) $\langle I, Karl\rangle\ \lambda x.[x$ has introduced better drummers to you]

The comparator then takes scope at the same level of that pair, leaving an open variable:

(599) *-er* $\langle I, Karl\rangle\ \lambda x.\iota y.[x$ has introduced y-good drummers to you]

The comparator is defined to take a pair of compared elements and a degree function f by which they are compared:

(600) *-er* $\langle a, b\rangle\ f$ is true iff $f(a) > f(b)$

The sentence (597) has the same structure as (596)c. Obviously, Heim's analysis cannot be extended to clausal complement cases such as (596)a-b, because the clausal predicate may be different from the main predicate:

(601) I have introduced better drummers to you than Karl has ever met.

We can say that Heim did not consider enough cases, and so her analysis is not generalizable to explicit clausal complements.

Her analysis is also very different from the analysis of all the other cases shown in this chapter, as she assumes a quite complex and idiosyncratic syntax-semantics interface:

some item from the main sentence is paired with the item from the comparative phrase by some unknown mechanism which she does not elaborate on; and the *-er* is a scope taking operator (independently of the NP in which it is located). Such an analysis is therefore inconsistent with the rest of the usual syntax-semantics interface.

Further evidence that (597) is obtained by ellipsis is that this sentence is in fact three-way ambiguous, as predicted by what the ellipsis mechanism can do. The first interpretation, which I discussed above, is obtained by stripping from (602)a. The second interpretation is obtained from (602)b1 (compare to (422)-(423)), which is an ACC case obtained from (602)b2. The sentence (602)b2 involves extraposition and antecedent-contained ellipsis as defined in the F-structure and not the C-structure – as was discussed for (489). The third interpretation of (597) is also obtained by stripping, but this time from (602)c. That sentence means: the drummers I have introduced to you are better than Karl (in his role as a drummer).

(602) a. I have introduced better drummers to you than Karl did.

b1. I have introduced better drummers to you than to Karl.

b2. I have introduced better drummers to you than I did to Karl.

c. I have introduced better drummers to you than Karl is.

My analysis, which relies on independently-motivated ellipsis mechanisms, predicts these interpretations without needing to rely on several different "direct" and complex analyses of (597).

11.4 Adverbial Comparatives

Gradable adverbial comparatives are similar to the adjectival comparatives above. Their meaning depends on degrees. Here is a sketch of the analysis.

(603) Mary ran 5mph.

$\exists e.run(e) \wedge subj(e, mary) \wedge$ *deg-compar-ev*$(=, ?, e, \langle 5, mph \rangle)$

Compare this to (520). We want to say that the speed of e is 5, measured by the mph units, just as we say the tallness of John is 6, measured by the foot units. The difference is that here we do not have the name of the scale explicitly (we cannot say "Mary ran 5mph fast"), and so we leave a placeholder '?' for it. We need to infer it from the unit names.

We also have:

(604) Mary ran faster than 5mph.

$\exists e.run(e) \wedge subj(e, mary) \wedge deg\text{-}compar\text{-}ev(>, fast, e, \langle 5, mph \rangle)$

(605) Mary ran faster than John ran.

$\exists e.run(e) \wedge subj(e, mary) \wedge deg\text{-}compar\text{-}ev(>, fast, e, \iota y.S(R(y)))$

where:

$S = \lambda P.\exists e.run(e) \wedge subj(e, john) \wedge P(e)$

$R = \lambda y \lambda e.deg\text{-}compar\text{-}ev(=, fast, e, y)$

Here, S is the meaning of the clause "John ran _" which is missing an adverb phrase. What adverb phrase? This is filled by $R(y)$, which says that the fastness of the event it modifies is y. The glue specification will therefore be similar to that of a predicative gradable adjective whose clausal complement is missing an AP (as in: "Mary is taller than John is _").

In (606)a, we do not want to compare the fastness of the event directly to John but to how fast John ran, as (606)b indicates. Hence, (606)b is obtained by VP-ellipsis from (605), and (606)a is obtained by stripping from (606)b.

(606) a. Mary ran faster than John.

b. Mary ran faster than John did.

It should be mentioned that there is more to be said about the semantics of adverbial comparatives. The sentence (607)a says that the time of the event of John eating dinner was before the time of the event of Mary arriving. In contrast, (607)b does not say the parallel thing about the two events. It says that the second event did not actually occur.

(607) a. John ate dinner before Mary arrived.

b. John defused the bomb before it exploded.

Also, *before* and *after* are not converses of each other at the sentence level, i.e. it is not always the case that "S_1 before S_2" is true iff "S_2 after S_1". For example, if Cleo has been in America since 1995 and David since 1997, then (608)a is true while (608)b is false.

(608) a. Cleo was in America after David was in America.

b. David was in America before Cleo was in America.

A more subtle analysis is needed to show that the two words are lexical converses (Beaver and Condoravdi, 2003). However, this point is largely orthogonal to the issue of specifying the syntax-semantics interface here.

Chapter 12

Plurals

Most puzzle texts start with a claim that some groups of elements exist and are related to each other. This brings into play a potential ambiguity between distributive, collective, cumulative, and other readings of predicates that have one or more plural arguments (see (Lønning, 1997) for a survey). For example, what does a sentence such as the first sentence of Figure 1.1 actually say in the context of the puzzle? It certainly does not claim that each sculpture is to be exhibited in each room. Perhaps it says that each sculpture is to be exhibited in one of the rooms? It is doubtful because such constructions are sometimes used in puzzles where a valid solution may use some but not all of the objects mentioned.

There are several additional interesting issues about plurality in the puzzles, including appositions that name the members of the group or their types (609)a,b, *respectively* constructions (609)c, and adjectives that denote collective properties (609)d,e.

(609) a. Six sculptures – C, D, E, F, G, and H – are to be exhibited in rooms 1, 2, and 3 of an art gallery.

b. At a small press, six textbooks, three introductory – F, G, and H – and three advanced – X, Y, and Z – will each be evaluated ...

c. Flights 103 and 104 are scheduled to depart at 11:00 a.m. and 12 noon, respectively.

d. Books G and X may not be evaluated during any two *consecutive* weeks.

e. Sculptures T, S, and P must be on stands that are *immediately adjacent* to one another.

Notice that all that (609)e says is: there are some stands that are immediately adjacent to one another, and sculptures T, S, and P must be on those stands. In the real world, there might be just two stands that support the three sculptures, or more than three stands,

some of which need not even touch any sculpture. It is only the addition of a constraint that there is a one-to-one mapping between sculptures and stands that tells us there are exactly three stands. This constraint results from a combination of world knowledge and constraints expressed explicitly in the puzzle.

Going into the details of all these issues is beyond the scope of this dissertation. In this chapter, I will cover only some issues of plurality that are needed in other chapters.

12.1 Extending the MRL

12.1.1 Pluralities

Certain predicates are really true of individuals only, even if the sentence seems to predicate them of a group:

(610) John and Mary sneezed.
⇒ John sneezed and Mary sneezed.

Other predicates are irreducibly collective:

(611) a. Ten thousand soldiers surrounded the city.
⇏ Each of the ten thousand soldiers surrounded the city.
b. John and Mary weigh 230 lbs together.
⇏ John weighs 230 lbs (together).
c. John and Mary like each other.
⇏ John likes each other. Or even: John likes himself.

One approach to deal with this uses a higher-order logic, where the domain of models includes only individuals but the logical language may talk about sets of individuals (e.g.: (Scha, 1981; van der Does, 1993)). Another approach is to enrich the domain with entities called *pluralities* (Link, 1998), and to arrange the pluralities in a lattice structure. I will follow the second approach because I think that we want our meaning representation language to talk about pluralities directly. (See (Lønning, 1997) for further discussion.)

We extend the definition of our L^{MR} as follows:

- In the models, we still have the sort e. However, D^e_M has a lattice structure, and it is further subdivided into the set of atoms D^a_M and non-atoms D^{na}_M.

- The language now has a new binary function symbol: $\oplus$, which can apply on a pair of expressions of type e and yields an expression of type e. It represents the lattice join operator. For example, "John and Mary" will be represented as $john \oplus mary$.

We need to impose certain properties. Here, we can impose the properties by inserting them into the definition of the logic itself. Let $\hat{\oplus} = [\![\oplus]\!]$, i.e. $\hat{\oplus}$ is the interpretation of the language's binary symbol. For any $a, b, c \in D^e_M$, this interpretation satisfies the properties:

- Idempotent: $a \hat{\oplus} a = a$
- Commutative: $a \hat{\oplus} b = b \hat{\oplus} a$
- Associative: $a \hat{\oplus} (b \hat{\oplus} c) = (a \hat{\oplus} b) \hat{\oplus} c$

We add to the language a notational convenience: We may write $\oplus\{a_1, \ldots, a_n\}$ instead of $a_1 \oplus \ldots \oplus a_n$. We also define the operations:

- For $a \in D^e_M$, $at(a)$ is the set of all atomic elements in a ($at(a) \subseteq D^a_M$).
- For $a \in D^e_M$, $|a|$ is the number of atomic elements comprising a, i.e.: $|a| = |at(a)|$.
- If φ is an expression of type $e \to t$ then $[\![plur(\varphi)]\!]_M = \{a \in D^{na}_M \mid at(a) \subseteq [\![\varphi]\!]\}$.

As a possible first step towards the implementation language L^{IMP} and axioms that guarantee the properties above, see the first-order axioms in (Link, 1998, ch.6).

12.1.2 Underspecified Forms

An important thing to note is that now predicates of type $e \to t$ may be collective in the sense that they include in their denotation tuples of elements, where one or more of these elements are non-atomic pluralities. The correct representation in L^{MR} of the truth conditions of (610)a is $sneeze(john) \wedge sneeze(mary)$ and not $sneeze(john \oplus mary)$ because only individuals can sneeze. We are only interested in models of L^{MR} that interpret predicates correctly. In particular, we should have $[\![sneeze(\gamma)]\!]^M = false$ when $[\![\gamma]\!]^M \in D^{na}_M$, and also $[\![surround(\gamma, x)]\!]^M = false$ when $[\![\gamma]\!]^M \in D^a_M$. As with the properties of $\oplus$ above, we insert these stipulations into the definition of the MTS of L^{MR}.

Nevertheless, it would be nice if both distributive and collective predicates would get the same *initial representation* because their surface structures are so similar. Therefore,

I introduce a new kind of underspecification regarding plurality ambiguities: Predicates applied on one or more plural arguments could mean different things. The readings cannot always be classified simply as a distributive reading and a collective reading. They cannot even be classified as distributive or collective per argument. There are sometimes additional readings:

(612) John and Mary met Bill and Sue.

Initial underspecified representation (ignoring entity raising from section 8.5.1):

$meet^{\Box}(john \oplus mary, bill \oplus sue)$

This could be resolved to any of:

$meet(john \oplus mary, bill \oplus sue)$	col-col
$\forall x \in john \oplus mary.\ meet(x, bill \oplus sue)$	dist-col
$\forall x \in bill \oplus sue.\ meet(john \oplus mary, x)$	col-dist
$\forall x \in john \oplus mary. \forall y \in bill \oplus sue.\ meet(x, y)$	dist-dist
$\forall \langle x, y \rangle \in (john \oplus mary) \times (bill \oplus sue).\ meet(x, y)$	respectively
$\forall x \in john \oplus mary. \exists y \in bill \oplus sue.\ meet(x, y)$	$\forall\exists$
etc.[1]	

The $\forall\exists$ reading is the right one for the predicate *exhibited in* in the following sentence from Figure 1.1:

(613) Six sculptures – C, D, E, F, G, and H – are to be exhibited in rooms 1, 2, and 3 of an art gallery.

A natural reading of:

(614) Ten companies own forty computers.

is the cumulative reading, where there are ten companies and there are forty computers. Each company owns at least one of the computers, and each computer is owned by at least one of the companies.

Which of the readings is the correct one depends on the context. As long as the context is not determined, we can use the form that is underspecified for plurality, which is marked with a $\Box$. This box can be seen as a pragmatic parameter that determines a specific reading once it is set.

[1] Our L^{MRL} should be extended to include the symbols $\in$, $\langle\cdot\rangle$, $\times$. The $\Box$ however is not in L^{MRL}. It is a symbol of underspecified expressions of L^{MRL}. The 'respectively' reading requires refining pluralities to be ordered lists of elements.

Note that $P^{\Box}(a, b)$ is not the same as $P(a, b)$. The latter is a formula of L^{MR} whereas the former is not. The former is rather an underspecified representation of formulas in L^{MR}, and it may be disambiguated to one or more of those, depending on the context and application. An expression using $\Box$ is underspecified very much like the $dref$ expressions we saw in chapter 8 which were underspecified w.r.t. the resolution of anaphoric expressions.

12.2 Some Issue with the Underspecification Operator

12.2.1 Conjoined Names

The F-structure of a conjunction of names is, for example:

(615) John, Bill, and Mary sneezed.

$$\begin{bmatrix} \text{PRED} & \text{`SNEEZE}\langle\text{SUBJ}\rangle\text{'} \\ \text{SUBJ} & \begin{bmatrix} \text{COORD} & \text{AND} \\ \text{COORD-LEVEL} & \text{NP} \\ \left\{ \begin{bmatrix} \text{PRED} & \text{`JOHN'} \end{bmatrix}, \begin{bmatrix} \text{PRED} & \text{`BILL'} \end{bmatrix}, \begin{bmatrix} \text{PRED} & \text{`MARY'} \end{bmatrix} \right\} \end{bmatrix} \end{bmatrix}$$

Following the discussion above, the initial semantic representation (ignoring *entity* from chapter 8), should be:

(616) $sneeze^{\Box}(john \oplus bill \oplus mary)$

For now, we can assume that each predicate is initially represented with a $\Box$ which can then be resolved in different ways. In the next sub-section, we will have to revise this assumption. For now, we just need to add to the glue specification rules that state how to construct the plurality. We can get this by:

(617) A coordination NP whose label is l

where (l COORD) = AND and (l COORD-LEVEL) = NP:

$\lambda x.\oplus x \ : \ conj(l)^e \rightarrow l^e$

For the rightmost conjunct, whose label is k:

$\lambda x.\{x\} \ : \ k^e \rightarrow conj(l)^e$

For any other conjunct, whose label is k:

$\lambda x \lambda y.(\{x\} \cup y) \ : \ k^e \rightarrow conj(l)^e \rightarrow conj(l)^e$

This specification has the effect that the rightmost conjunct x adds $\{x\} : conj(l)^e$, then each further conjunct adds itself to this set, and finally, the first line of the specification closes this set off with a '$\oplus$'. Note that $\{x\} \cup y$ is intended to be *evaluated* during the construction of the expression for the NP in the glue derivation.[2] Thus, at the end of the glue derivation, we will get (618)b rather than (618)a.

(618) "John, Bill, and Mary"

a. $(\{john\} \cup (\{bill\} \cup \{mary\}))$

b. $\oplus\{john, bill, mary\}$

In order to enforce the order of combination and prevent a spurious ambiguity (i.e. whether *john* or *bill* is added first to *mary*), we add a scoping constraint:

```
precedes(%F1,%F2), conjunct(%F1), conjunct(%F2) ==> outscopes(%F1,%F2).

conjunct(%Conjunct) :=
       +COORD-LEVEL(%Main,%%),
       +in_set(%Conjunct,%Main).
```

We add this before the last (default) scoping constraint of section 4.6.2.

Bringing anaphora and *entity*'s back into the picture, we just need to change the third line of (617) to:

(619) $\lambda x.entity(\hat{l}, \{\hat{l} = \oplus(x)\}) \ : \ conj(l)^e \rightarrow l^e$

The result is then:

(620) $entity(\hat{l}_1, \{\hat{l}_1 = \oplus(\{entity(\hat{l}_2, \{name(\hat{l}_2) = \text{`John'}\}), entity(\hat{l}_3, \{name(\hat{l}_3) = \text{`Mary'}\}),$
$entity(\hat{l}_4, \{name(\hat{l}_4) = \text{`Bill'}\})\})\})$

When this is raised by the entity raising rule from section 8.5.1, we get a DRS:

(621) $[\langle \hat{l}_1, \hat{l}_1 = \oplus(\{\hat{l}_2, \hat{l}_3, \hat{l}_4\})\rangle, \langle \hat{l}_2, name(\hat{l}_2) = \text{`John'}\rangle, \langle \hat{l}_3, \ldots\rangle, \langle \hat{l}_4, \ldots\rangle \ \ldots \ | \ \ldots]$

[2] This is implemented by using the expression `x\y\[x|y]` in the Prolog notation file on the LHS of the glue rule. When the glue prover applies this expression by replacing the variables `x` and `y` with their values (an expression and a list of expressions), we automatically get a new list of the values.

12.2.2 Scope Ambiguity

Consider:

(622) John and Mary saw a film.

If we assume that the $^{\Box}$ just comes automatically with the predicate, we get only (623)a, which can be resolved in (623)b to say that there is one film that both John and Mary saw. But we also would like to get the representation (623)c which can be resolved in (623)d to say that each of them saw a film, but not necessarily the same one.

(623) a. $a(film, \lambda x.saw^{\Box}(john \oplus mary, x))$
b. $a(film, \lambda x.\forall y \in john \oplus mary.\ saw(y, x))$
c. $(\lambda y.a(film, \lambda x.saw(y, x)))^{\Box}(john \oplus mary)$
d. $\forall y \in john \oplus mary.\ a(film, \lambda x.saw(y, x))$

To get both forms, we need to revise (619) again to:

(624) Coordination whose label is l:
$\lambda x \lambda P.\ P^{\Box}(entity(\hat{l}, \{\hat{l} = \oplus(x)\}))\ :\ conj(l)^e \rightarrow (l^e \rightarrow H^t) \rightarrow H^t$

This essentially behaves like a quantifier, because we potentially want to distribute the predicate over the members of the NP. If we have two or more of these on a predicate, we get more than one box. For example:

(625) John and Mary like Bill and Sue.
$(\lambda x.(\lambda y.like(x, y))^{\Box}(bill \oplus sue))^{\Box}(john \oplus mary)$

To handle this correctly, we simply consider a form like (626)a as a shorthand for (626)b.

(626) a. $(\lambda x_1 \ldots (\lambda x_n.\varphi)^{\Box}(a_n) \ldots)^{\Box}(a_n)$
b. $(\lambda x_1 \ldots \lambda x_n.\varphi)^{\Box}(a_1, \ldots, a_n)$

We also need to specify (following the discussion in section 6.6.1) that if two $^{\Box}$ operators are consecutive, they do not have a scope ambiguity with each other. A $^{\Box}$ may still have an ambiguity with other quantifiers and operators unless they have universal force.

12.2.3 Composite Predicates

Consider:

(627) John and Mary like themselves.

One reading is the double distributive reading where each of John and Mary likes each of the two. Another one is the 'respectively' reading, where John likes himself and Mary likes herself. It is tempting to say that the source of this ambiguity is the box on top of *like* in the representation for (627):

(628) $like^{\Box}(john \oplus mary, john \oplus mary)$

One evidence against this is the oddness of:

(629) John and Mary like themselves, respectively.

A stronger evidence is that this solution will not work for composite predicates:

(630) [John and Mary are running against each other in the election].
a. John and Mary think they will win.
b. $think^{\Box}(j \oplus m, win^{\Box}(j \oplus m))$
c. $(\lambda x.think^{\Box}(x, win^{\Box}(x)))^{\Box}(j \oplus m)$

Notice that in the second reading (630)c, we first need to create a composite predicate "x thinks x will win", and then mark a $^{\Box}$ on it, and apply that on $j \oplus m$. When that $^{\Box}$ is interpreted distributively, the two inner ones must too (as they apply on an individual), and we get:

(631) $\forall x \in j \oplus m.\ think(x, win(x))$

This reading cannot be obtained from a "respectively" reading of (630)b because "respectively" only operates on arguments of a predicate at the same level:

(632) * John and Mary think they will win, respectively.

We will get the correct result using the definition in (624). Now we get:

(633) think they will win =
$\lambda x.think(x, win(dref(l_4, num(l_4) = pl)))$

John and Mary =
$\lambda P.\ P^{\Box}(entity(l_1, l_1 = \oplus(\{entity(l_2, l_2 = john), entity(l_3, l_3 = mary)\})))$
John and Mary think they will win. =
$(\lambda x.think(x, win(dref(l_4, num(l_4) = pl))))^{\Box}(entity(l_1, l_1 = \oplus(\{entity(l_2, l_2 = john),$
$entity(l_3, l_3 = mary)\})))$

With entity raising, we get:

(634) $[\langle l_1, \{l_1 = \oplus(l_2, l_3)\}\rangle, \langle l_2, \{l_2 = john\}\rangle, \langle l_3, \{l_3 = mary\}\rangle \mid$
$(\lambda x.think(x, win(dref(l_4, num(l_4) = pl))))^{\Box}(l_1)]$

12.2.4 Resolution of *dref* to an abstracted variable

Now you can see that we must allow l_4 in (634) to be resolved to x, to get:

(635) $[\langle l_1, \{l_1 = \oplus(l_2, l_3)\}\rangle, \langle l_2, \{l_2 = john\}\rangle, \langle l_3, \{l_3 = mary\}\rangle \mid$
$(\lambda x.think(x, win(x)))^{\Box}(l_1)]$

And once we resolve $^{\Box}$ to the distributive reading, we get:

(636) $[\langle l_1, \{l_1 = \oplus(l_2, l_3)\}\rangle, \langle l_2, \{l_2 = john\}\rangle, \langle l_3, \{l_3 = mary\}\rangle \mid$
$\forall y \in l_1.think(y, win(y))]$

So we need to revise the algorithm in section 8.5.2 to allow a *dref* to be resolved to a lambda variable that appears somewhere above it.

Some people would object to this sort of move because we are equating here two very different kinds of things. In the interpretation of DRSs using a model-theoretic semantics (along the lines of (Muskens, 1995, 1996; Kohlhase et al., 1996; van Eijck and Kamp, 1997)), discourse variables denote entities in the domain. In contrast, a lambda-abstracted variable like x above does not denote anything. It is just a notational convenience. There is even a way to get rid of variables by using only combinatory functors (see e.g. (Carpenter, 1998, section 2.5)).[3] No such omission is possible with discourse variables.

Nonetheless, this is what we need. A possible way to make the equating of the two variables "kosher" is to make both of them parameters in the sense of Situation Semantics

[3] A very simple illustration of this point is that a composition of two functions f and g need not be written as $\lambda x.f(g(x))$ as it can be written using the functor $f \circ g$ without resorting to an "intermediate" variable x.

(Barwise and Perry, 1983; Devlin, 1991, 2006; Gawron and Peters, 1990). I present here a simplified version of Situation Semantics (only what is needed for the discussion here), and the reader is invited to consult those sources for further details. In this simplified version, the meaning of a sentence is taken to consist of a *structured object* called *infon*. We can then further check whether a particular situation in a particular context *supports* this infon (the claim that the sentence is making is true in the situation) or not (the claim is false in the situation) or neither (the situation is silent on that matter). Furthermore, the structured object may contain certain objects called *parameters*, in which case it is a parametric object. Practitioners of Situation Semantics go to great pains to emphasize that a parameter is not a variable in the formal language but an element of the domain. Therefore, a whole new mathematical notation for talking about the structured objects is used.

An important difference between the lambda calculus and Situation Semantics is this: In the former, (637)a and (637)b are two different expressions in the formal language which nonetheless denote the same mathematical object. In contrast, (637)c and (637)d denote two different mathematical objects (in this case, infons) in Situation Semantics.

(637) assume t can be freely substituted for x in φ

a. $(\lambda x.\varphi)(r)$ (e.g.: $(\lambda x.sleep(x))(r)$)

b. $\varphi[r/x]$ (e.g.: $sleep(r)$)

c. $\langle\langle [p \mid \varphi],\ r \rangle\rangle$ (e.g.: $\langle\langle [p \mid \langle\langle sleep, p \rangle\rangle],\ r \rangle\rangle$)

d. $\varphi[r/p]$ (e.g.: $\langle\langle sleep, r \rangle\rangle$)

If we wish, we could say that (under normal conditions), if (637)c and (637)d are infons, then one is supported by a situation iff the other is. However, we are not required to make this move.

Using the notation of Situation Semantics,[4] we could say that the meaning of the sentence (630)a should be represented as (638) rather than (633).

(638) $\langle\langle\ [p \mid \langle\langle think,\ p,\ \langle\langle win, q_{they} \rangle\rangle\rangle]\ ,\ r_\sigma\ \rangle\rangle$

where σ is $\langle\langle =,\ r,\ s_{John} \oplus t_{Mary} \rangle\rangle$

[4] I am using here a simplified version of the notation in (Gawron and Peters, 1990).

The terms p, q, r, s, and t are all parameters, not variables.[5] A parameter may have restrictions on the possible values it could refer to. Thus, s_{John} has the restriction that the parameter may refer only to a human entity whose name is "John." So now the denotation of the semantic representations we assign to sentences are not functions and predicates but rather structured mathematical objects from the Situation Semantics ontology.

How does all this help us? First, we do not necessarily need to use the $\Box$ mechanism. This is possible if we do not say that (637)c is supported by a situation iff (637)d is, and instead say that this holds only in some contexts (this gives the "collective" reading). In other contexts, (637)c is supported in a situation iff the following infon is supported there (giving the "distributive" reading):

(639) $\langle\langle forall,\ u,\ \langle\langle imply,\ \langle\langle \in, u, r\rangle\rangle,\ \varphi[u/p]\rangle\rangle\rangle\rangle$

Second, p and q in (638) are both parameters of exactly the same status (and therefore can be legitimately equated), whereas x and $dref$ had a different status in (633). Here is how it works. Suppose that in a particular context C_1, we are given further information that the parameter q has a restriction $\langle\langle =,\ q,\ a\rangle\rangle$ for some group object a that was mentioned previously in the context. Then in context C_1, the sentence means that both John and Mary think that the group a will win. In another context C_2, we are given the information that the parameter q has a restriction $\langle\langle =,\ q,\ p\rangle\rangle$. So in C_2, the proposition's meaning is:

(640) $\langle\!\langle\ [p \mid \langle\langle think,\ p,\ \langle\langle win, q_{\langle\langle =,q,p\rangle\rangle}\rangle\rangle\rangle\rangle]\ ,\ r_\sigma\ \rangle\!\rangle$
where σ is $\langle\langle =,\ r,\ s_{John} \oplus t_{Mary}\rangle\rangle$

Because p is abstracted over, (640) denotes the same object as (641) under the rules of Situation Semantics.

(641) $\langle\!\langle\ [p \mid \langle\langle think,\ p,\ \langle\langle win, p\rangle\rangle\rangle\rangle]\ ,\ r_\sigma\ \rangle\!\rangle$
where σ is $\langle\langle =,\ r,\ s_{John} \oplus t_{Mary}\rangle\rangle$

This is what we wanted to get. If we further assume that C_2 is one of those contexts where (637)c is supported in a situation iff (639) is supported, then we get that in C_2, the following is supported:

[5] If we considered (638) to be an expression in something like a sorted first-order language, we would say that each of p, q, r, s, and t is an object constant, whose denotation is an element (a member of the domain) of type "parameter".

(642) $\langle\langle forall,\ u,\ \langle\langle imply,\ \langle\langle \in, u, r_\sigma\rangle\rangle,\ \langle\langle think,\ u,\ \langle\langle win, u\rangle\rangle\rangle\rangle\rangle\rangle\rangle\rangle$
where σ is $\langle\langle =,\ r,\ s_{John} \oplus t_{Mary}\rangle\rangle$

Let us recap. We started with a representation (633). We made some transformations to obtain (634). These transformations were justified by the intended MTS. However, the transformation from that to (635) was not justified by the MTS because we equated a language variable with a discourse variable. The solution was to provide a more complex MTS that uses Situation Semantics objects. In this MTS, equating p and q was justified since they were both of the same kind (parameters).

Is this move illuminating? Some would say yes. Furthermore, Situation Semantics has several other independent motivations, and is needed when analyzing more complex intensional and context-dependent linguistic constructions, so if one is using that framework, one might as well use the solution proposed here.

When moving to Situation Semantics, we essentially decide to rely on some distinctions between representations which previously we considered to be equivalent. If one takes this process to its extreme, we eventually decide that no two representations are inherently equivalent. This kind of move might in fact be unavoidable given the nature of natural language. Two distinct expressions that might at first seem equivalent in all respects may later turn out to have some subtle differences in meaning, which surface in different contexts, and so the two expressions should be given similar but distinguishable meanings (and representations).

However, for the simpler sentences investigated here, the re-interpretation using Situation Semantics has few if any practical implications as far as the computational implementation is concerned.

12.2.5 Resolution of Singular $dref$, Revisited

Now that we know that a plural $dref$ must be allowed to refer to an abstracted variable, should we allow this for a singular $dref$ as well? In (643), this would give us (643)c rather than (643)b.

(643) a. John thinks that he is smart.

b. $think(entity(\hat{l}_1, \{name(\hat{l}_1) = \text{`John'}\}), smart(dref(\hat{l}_2, sg, male))))$

c. $(\lambda x.think(x, smart(dref(\hat{l}_2, sg, male))))^{app}(entity(\hat{l}_1, \{name(\hat{l}_1) = \text{`John'}\}))$

After entity raising, $\hat{l}_2$ in (643)b could be equated only to $\hat{l}_1$, but in (643)c, $\hat{l}_2$ could be equated to either $\hat{l}_1$ or x. All three options would eventually reduce to the same final result:

(644) $[\langle \hat{l}_1, \{name(\hat{l}_1) = \text{`John'}\} \rangle \mid think(\hat{l}_1, smart(\hat{l}_1))]$

To obtain (643)c, we need to revise the entries of NPs. For example, we need (645)b instead of (645)a.

(645) proper name n

a. $entity(\hat{l}, \{name(\hat{l}) = n\})$: l^e where l is my NP's label

b. $\lambda P.P(entity(\hat{l}, \{name(\hat{l}) = n\}))$: $(l^e \rightarrow h^t) \rightarrow h^t$
where l is my NP's label and h^t is the topmost clause

According to some theories of ellipsis, e.g. (Gawron and Peters, 1990), both the bound reading (643)c and the direct reading (643)b are in fact necessary, as they provide the "sloppy" and "strict" readings of sentences such as:

(646) John thinks he is smart. Bill does too.

If (643)b is used, then the predicate $\lambda s.think(x, smart(john))$ is applied on Bill in the second sentence to get the strict reading. If (643)c is used, then $\lambda s.think(x, smart(x))$ is first created in the first sentence and so it is available in the context to be applied on Bill in the second sentence, giving the sloppy reading.

12.2.6 How Far Can the Operator Float?

Can H^t in (624) be the category of any dominating clause, or only the immediately dominating one? Consider:

(647) a. Bill thinks that John and Mary left.

b. $think(bill, left(john \oplus mary))$

c. $think(bill, \forall x \in john \oplus mary.\ left(x))$

d. $\forall x \in john \oplus mary.\ think(bill, left(x))$

In (647)b we see the collective reading, which is unlikely here. In (647)c we see the distributive reading, where H^t is the inner clause. It is hard to get the distributive reading in (647)d where H^t is the outer clause. It is possible to get it with our entry (624), but for practical purposes we can choose to limit H^t there to be the immediately dominating clause.

12.2.7 The Location of Ambiguity

Some people explained why plurality ambiguity cannot originate in the NP but must be in the VP (see e.g. (Link, 1998, pp.180-183)). For example, in the sentence:

(648) John, Bill, and Mary met in a bar and had a bear.

if the ambiguity was in the NP, then how can the same NP support on the one hand a collective reading for "met in a bar" and a distributive one for "had a bear"?

How is that compatible with the entry we gave above which locates the $^{\square}$ in the NP? Well, I think that the ambiguity is in *both* the NP and the VP. What needs to be said is that there might be a dependence between them: If the VP's $^{\square}$ is resolved to a collective reading, then the $^{\square}$ of the NP cannot be resolved distributively. This can be enforced by allowing an expression of the form $\forall x \in y.\varphi$ only if y is a non-atomic plurality.

12.2.8 Event Predicates

So far we ignored event representation by using forms like (649)b. When we take events into consideration, the correct representation is not (649)c or even (649)d but (649)e, so that we could correctly resolve $^{\square}$ to all the possible readings.

(649) a. John and Mary like Bill and Sue.

b. $like^{\square}(john \oplus mary, bill \oplus sue)$

c. $\exists e.like(e) \wedge subj(e, john \oplus mary) \wedge obj(e, bill \oplus sue)$

d. $\exists e.like(e) \wedge subj^{\square}(e, john \oplus mary) \wedge obj^{\square}(e, bill \oplus sue)$

e. $(\lambda x \lambda y.\exists e.like(e) \wedge subj(e, x) \wedge obj(e, y))^{\square}(john \oplus mary, bill \oplus sue)$

The desired result will fall out automatically from the move we made in (624) and (626) because a conjoined NP now behaves like a quantifier, so it requires a $e \rightarrow t$ predicate to work on. The right behavior will be obtained for the same reason that was discussed in section 4.4.7.

12.3 Interaction with Anaphora

I mention here some points that are needed in other sections of this dissertation, but I do not provide solutions for them.

12.3.1 Basic

Plural pronouns may "collect" antecedents, and convert them to a plurality:

(650) John$_i$ likes Mary$_j$. They$_{\{i,j\}}$ met in school.
$like(entity(l_1, \{name(l_1) = \text{'John'}\}), entity(l_2, \{name(l_2) = \text{'Mary'}\}))$
$meet(dref(l_3, \{num(l_3) = pl\}))$

The algorithm in section 8.5 needs to be extended to allow a plural $dref$ to collect several discourse variables and connect them with '$\oplus$', to get the result:

(651) $[\langle l_1, \{name(l_1) = \text{'John'}\}\rangle, \langle l_2, \{name(l_2) = \text{'Mary'}\}\rangle, \langle l_3, \{l_3 = l_1 \oplus l_2\}\rangle \mid$
$like(l_1, l_2) \wedge meet(l_3)]$

12.3.2 E-Type Anaphora and Donkey Anaphora

A plural pronoun may also be anaphoric to a quantifier where the pronoun is not in the scope of the quantifier:

(652) [Every student]$_i$ in the class wrote [a poem]$_j$. They$_i$ collected them$_j$ in a book.

Here, *they* is anaphoric to *every student*, and *them* to *a poem*, but the pronouns are not within the scope of the quantifiers they are anaphoric to. This kind of anaphora is called E-type anaphora (Evans, 1980). On the one hand, *every* is quantifying over students, but on the other hand, it seems to introduce a plural entity, namely the plurality of all students in the class, which is then referred to by *they*. Things are not so simple, however. It is unlikely that the quantifier itself introduces the entity: *them* refers to the plurality of the poems that the students wrote. Why would the singular *a poem* here introduce such a plurality?

Clearly, an E-type anaphoric expression uses the material in the NP it is anaphoric to, but it needs to do something new with it. The technical question is this: If we say that expressions in the sentence denote mathematical entities (as discussed in chapter 2), then the denotation of the first sentence in (652) is a truth value, or a proposition, but in any case, the denotation of the quantifiers has already been "used up" by the semantic composition process and are not available again to be used by the next sentence.

A DRT-like solution would be to introduce more entities into the DRSs. Just as we add a singular discourse variable for *John*, we would add a plurality discourse variable for

a quantifier. This was suggested in (Kamp and Reyle, 1993). However, this idea is too simple as the discussion above explained.

Another instance where the problem with DRT's solution can be seen is in donkey anaphora (see section 8.3.4). It is in fact a special kind of E-type anaphora where the clauses of the antecedent and of the anaphoric expression are connected by a quantifier or a conditional. While DRT works for simple cases as we saw in section 8.3.4, it does not generalize well when other quantifiers are involved. In particular, there is the "proportion problem" (Kadmon, 1987). For example, in the sentence "Most farmers who own a donkey beat it", the truth conditions in DRT come out as: most farmer-donkey pairs in which the farmer owns the donkey are such that the farmer beats the donkey, whereas the correct truth conditions should be: most farmers who own any donkey beat all the donkeys they own. Thus, *it* here constructs, for each farmer, the set of all donkeys owns by that farmer.

I think the solution would probably be closer to what can be done with Situation Semantics (see section 12.2.4). Because more structural information is available in the meaning of expressions and sentences, the anaphoric expressions can use this information to construct new semantic entities as needed. It is the plural pronoun rather than its antecedent(s) that constructs these entities, just as in section 12.3.1 above.

For relevant references about this topic, see (Gawron et al., 1991; Lappin and Francez, 1994; Kanazawa, 1994; Peters and Westerståhl, 2002). For relevant work in glue semantics, see (Crouch and van Genabith, 1999).

What are the consequences for our glue semantics specification? Because we made sure in section 8.4.3 to separate that specification from the anaphora resolution level, we do not need to change our glue specification. Instead, we can now easily translate quantifier and anaphora constructs to representations of Situation Semantics rather than DRSs. If instead we encoded DRSs directly into the syntax-semantics mapping (as in section 8.4.2), we would need to redo all our work yet again.

Chapter 13

Overt Reciprocals

13.1 Introduction

An interesting construction that appears in exact NLU texts, including logic puzzles, is reciprocation expressed using "each other" and "one another." It is interesting because it can take on a variety of meanings, depending on the particular lexical items used and the context. For example, in (653)a, the binary relation *must-refer-to-indirectly* holds on each pair of distinct elements from the set *members-of-parliament*. But this is not the case in (653)b, where *sit-alongside* forms a chain of elements from *five-Boston-pitchers*. What (653)c says is that each member of *the pirates* is related to one other member of the group through *stare-at-in-surprise*, but again not every pair of members are related in this way. There are several other options (see (Dalrymple et al., 1998) for a systematic analysis).

(653) a. Members of Parliament must refer to each other indirectly.

b. Five Boston pitchers sat alongside each other on a bench.

c. The pirates stared at each other in surprise.

Here are some examples from logic puzzle texts:

(654) a. V and S must be separated from each other by exactly one song.

b. Each bridge directly connects exactly two islands with one another.

c. The four paperback books must be next to each other.

d. Sculptures T, S, and P must be on stands that are immediately adjacent to one another.

e. The stores on the north side are located directly across the street from those on the south side, facing each other in pairs, as follows: 1 and 2; 3 and 4; 5 and 6; 7 and 8; 9 and 10.

The semantic composition aspect of reciprocation is also interesting because of this construct's unique properties. Although the simplest meaning as in (653)a can be reduced to two universal quantifiers, if we want to capture all of the other possible meanings, we need to analyze the reciprocal using a higher-order operator RECIP(A, R), where A is a set, and R is a binary relation. The operator RECIP itself can mean different things depending on the particular A, R, and the context, and so as far as the syntax-semantics interface, it is an underspecified form. This operator is interesting because it is a quantifier that is not the usual type-$\langle 1,1\rangle$ quantifier we saw until now but a type-$\langle 1,2\rangle$ (polyadic) quantifier where its second argument R is a binary relation rather than a monadic set.[1] Furthermore, the reciprocal also has an anaphoric component – its first argument A comes from an NP that the reciprocal is anaphoric to. I will show below that we can account for this unique operator very naturally in glue semantics thanks to the generality of that framework, whereas it is very difficult to do so in other frameworks.

Another reason why reciprocation is interesting is that sometimes the reciprocal expression can be covert (not expressed) without changing the meaning of the sentence. I will investigate such cases in the next chapter:

(655) a. Oscar and Pedro do not attend the same class (as each other).
b. Bill 2 and Bill 6 are paid on different days (than each other).

13.2 Basic Syntax-Semantics Interface

13.2.1 Basic Case

Statements

Recall the contribution of words in the simple sentence:

(656) [[John]$_2$ saw [Mary]$_3$]$_1$
$john : \mathbf{2}^e$
$mary : \mathbf{3}^e$
$saw : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$
Result: $saw(john, mary) : \mathbf{1}^t$

For a sentence with a reciprocal we have:

[1] See (Keenan and Westerståhl, 1997; Peters and Westerståhl, 2006).

(657) [[John and Mary]$_2$ saw [each other]$_3$]$_1$

The contribution of the subject is $john \oplus mary : \mathbf{2}^e$ as we saw in the previous chapter (this is the simplest version, and below we will move to the more complex versions). Below, I will use the shorthand jm for this. The contribution of the reciprocal operator should be $\lambda z \lambda R.\text{RECIP}(z, R)$ of type $e \rightarrow (e \rightarrow e \rightarrow t) \rightarrow t$, where RECIP is a type $\langle 1, 2 \rangle$ polyadic quantifier. Since this operator needs to take jm and saw as arguments, we have the following statements:

(658) $jm : \mathbf{2}^e$
$saw : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$
$\lambda z \lambda R.\text{RECIP}(z, R) : \mathbf{2}^e \rightarrow (\mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$

These combine to yield:

(659) $\text{RECIP}(jm, saw) : \mathbf{1}^t$

Note that saw could also apply directly on jm (as is required in a sentence such as "John and Mary saw the cat"). However, this route would lead to a dead-end derivation because "each other" would then not be able to combine with the result $saw(john \oplus mary) : \mathbf{3}^e \rightarrow \mathbf{1}^t$. This does not pose a problem, and it is similar to the situation with equi verbs that was discussed in section 5.4.1.

In the previous chapter, we saw a more general meaning expression for the subject. With it, we get:

(660) $\lambda P.P^{\square}(john \oplus mary) : (\mathbf{2}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$
$saw : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t$
$\lambda z \lambda R.\text{RECIP}(z, R) : \mathbf{2}^e \rightarrow (\mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$
$\Rightarrow (\lambda z.\text{RECIP}(z, saw))^{\square}(john \oplus mary) : \mathbf{1}^t$

Now $^{\square}$ can be resolved only collectively because the first argument to RECIP cannot be an individual. After resolving $^{\square}$, we get (659). Therefore, below I will usually use the simpler $jm : \mathbf{2}^e$ to show derivations, unless the analysis needs the more complex entry (however, all derivations can work correctly with the more complex entry).

Glue Specification

It remains to show the glue specification for "each other" and "one another":[2]

(661) $\lambda z \lambda R.\textsc{recip}(z, R) : a^e \rightarrow (a^e \rightarrow l^e \rightarrow k^t) \rightarrow k^t$
where l is my label, k is the label of the clause that I scope over,
and a is my (ultimate) antecedent's label;
where k = the label of the minimal clause containing a,
and [some more constraints on a]

In example (658), l is instantiated to **3** and a is instantiated to **2** since the NP "John and Mary" is the antecedent of "each other". The "some constraints" part above should be filled with conditions on possible antecedents for a reciprocal. I assume that just as for reflexives (in chapter 8), these constraints are the responsibility of the syntax. They are more complicated than those for a reflexive, as will be discussed in section 13.3. In example (658), we can write the fact about the antecedent as $\mathbf{3}_{ant} = \mathbf{2}$. The constraint on k is the same as the one we saw for pronouns and reflexives in section 8.2. I will return to discuss this in section 13.3.

As you can see in the specification, a reciprocal operator behaves like a combination of a quantifier (sections 3.3 and 4.5) and a bound anaphoric expression (section 8.2). As explained in the solution to problem 3 in section 8.2.3, the order of the variables is important if we want the packed algorithm to recognize the reciprocal as a sentential modifier and compute its scope interactions with quantifiers and other sentential modifiers more efficiently.

In chapter 8, we concluded that the bound-anaphora analysis of pronouns in glue semantics is less preferred than the DRT-based analysis. Does this conclusion apply to reciprocals? The answer is *no* because reciprocals have an aspect that other pronouns don't, namely the quantificational force. Moreover, reciprocals can only be bound (they cannot be anaphoric to a previous sentence), and in fact they are usually anaphoric to a previous NP in the same clause, or at most the next clause up (see section 13.3).

Here are the details of the glue semantics specification. Note that `PRON-TYPE` is without `+`. This fact needs to be consumed in order to create a split in the choice-space if there is more than one possible antecedent, or more than one possible landing sites.

[2]This will be revised in section 13.4.

```
PRON-TYPE(%F,recip),
plural_np(%A),
-not_ant(%A,%F), {\+ %A=%F},
+dominating_clause(%A,%Land)
   ==> glue_recip(%F,%A,%Land).

glue_recip(%F,%A,%Land) :=
   X\R\recip(X,R) : %A$e -> (%A$e -> %F$e -> %Land$t) -> %Land$t .

plural_np(%A) :=
   (+COORD-LEVEL(%A,NP), +COORD-FORM(%A,and)) |
   (+NTYPE(%A,%%), +NUM(%A,pl)).
```

13.2.2 "Each Other" Is Not Always Anaphoric to the Subject

"Each other" is not always anaphoric to the subject. We get this result without any further effort:

(662) $[[\text{Bill}]_2$ introduced [John and Mary]$_3$ to [each other]$_4]_1$

$bill : \mathbf{2}^e$

$introduce : \mathbf{2}^e \rightarrow \mathbf{3}^e \rightarrow \mathbf{4}^e \rightarrow \mathbf{1}^t$

$jm : \mathbf{3}^e$

$\lambda z \lambda R.\text{RECIP}(z, R) : {\mathbf{4}_{ant}}^e \rightarrow ({\mathbf{4}_{ant}}^e \rightarrow \mathbf{4}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$

where $\mathbf{4}_{ant} = \mathbf{3}$

$bill + introduced \Rightarrow \lambda y \lambda z.introduce(bill, y, z) : \mathbf{3}^e \rightarrow \mathbf{4}^e \rightarrow \mathbf{1}^t$

$+\text{RECIP} \Rightarrow \lambda z.\text{RECIP}(z, \lambda y \lambda z.introduce(bill, y, z)) : \mathbf{3}^e \rightarrow \mathbf{1}^t$

$+jm \Rightarrow \text{RECIP}(jm, \lambda y \lambda z.introduce(bill, y, z)) : \mathbf{1}^t$

It is also predicted correctly that the following sentence is ambiguous:

(663) John and Mary introduced Sue and Tom to each other.

$\text{RECIP}(st, \lambda u \lambda v.introduce(jm, u, v))$ ("each other" is anaphoric to st)

$\text{RECIP}(jm, \lambda u \lambda v.introduce(u, st, v))$ ("each other" is anaphoric to jm)

13.2.3 Scope Ambiguity at One Level

"Each other" may have scope interaction with other scope-bearing operators (see (Dalrymple et al., 1998, pp.183–184) for a discussion):

(664) John and Mary gave each other some present.

$\textsc{recip}(jm, \lambda x\lambda y.some(present, \lambda z.gave(x, y, z)))$

$some(present, \lambda z.\textsc{recip}(jm, \lambda x\lambda y.gave(x, y, z)))$

Here is the analysis:

(665) [[John and Mary]$_2$ gave [each other]$_3$ [some present]$_4$]$_1$

$gave : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{4}^e \to \mathbf{1}^t$

$jm : \mathbf{2}^e$

$\lambda z\lambda R.\textsc{recip}(z, R) : {\mathbf{3}_{ant}}^e \to ({\mathbf{3}_{ant}}^e \to \mathbf{3}^e \to k^t) \to k^t$

where $\mathbf{3}_{ant} = \mathbf{2}$

$some(present) : (\mathbf{4}^e \to k^t) \to k^t$

Instantiate $k^t = \mathbf{1}^t$ in both "each other" and "some present":

$\textsc{recip} : \mathbf{2}^e \to (\mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t) \to \mathbf{1}^t$

$some(present) : (\mathbf{4}^e \to \mathbf{1}^t) \to \mathbf{1}^t$

Now, if we first combine "some present" with "gave" (using function composition) we get:

(666) $\lambda x\lambda y.some(present, \lambda z.gave(x, y, z)) : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t$

Then "each other" can apply on "John and Mary" and on (666) to obtain the first reading of (664). If instead we re-arrange the arguments of "gave" using EXCH:

(667) $\lambda z\lambda x\lambda y.gave(x, y, z) : \mathbf{4}^e \to \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t$

and then combine "each other" with "John and Mary" using APP, and then with (667) using COMP, we get:

(668) $\lambda z.\textsc{recip}(jm, \lambda x\lambda y.gave(x, y, z)) : \mathbf{4}^e \to \mathbf{1}^t$

This can combine with "some present" to get the second reading of (664).

The two derivations are shown in Figure 13.1 using the "user friendly" style of derivation. The linear logic normal form proofs are given in Figure 13.2. The resulting packed meaning representation is:

(669) $l_1 : \langle A1 : l_2(\lambda x\lambda y.l_3(\lambda z.l_4)),\ A2 : l_3(\lambda z.l_2(\lambda x\lambda y.l_4))\rangle$

$l_2 : \lambda R.\textsc{recip}(jm, R)$

$l_3 : \lambda P.some(present, P)$

$l_4 : gave(x, y, z)$

$$\dfrac{\dfrac{g : \mathbf{2} \to \mathbf{3} \to \mathbf{4} \to \mathbf{1} \qquad s(p) : (\mathbf{4} \to \mathbf{1}) \to \mathbf{1}}{\lambda x, y.s(p, \lambda z.g(x, y, z)) : \mathbf{2} \to \mathbf{3} \to \mathbf{1}}\,\text{COMP} \qquad \dfrac{jm : \mathbf{2} \qquad \text{RECIP} : \mathbf{2} \to (\mathbf{2} \to \mathbf{3} \to \mathbf{1}) \to \mathbf{1}}{\text{RECIP}(jm) : (\mathbf{2} \to \mathbf{3} \to \mathbf{1}) \to \mathbf{1}}\,\text{APP}}{\text{RECIP}(jm, \lambda x \lambda y.some(present, \lambda z.gave(x, y, z))) : \mathbf{1}}\,\text{APP}$$

$$\dfrac{\dfrac{g : \mathbf{2} \to \mathbf{3} \to \mathbf{4} \to \mathbf{1}}{\lambda z, x, y.g(x, y, z) : \mathbf{4} \to \mathbf{2} \to \mathbf{3} \to \mathbf{1}}\,\text{EXCH} \qquad \dfrac{jm : \mathbf{2} \qquad \text{RECIP} : \mathbf{2} \to (\mathbf{2} \to \mathbf{3} \to \mathbf{1}) \to \mathbf{1}}{\text{RECIP}(jm) : (\mathbf{2} \to \mathbf{3} \to \mathbf{1}) \to \mathbf{1}}\,\text{APP}}{\lambda z.\text{RECIP}(jm, \lambda x, y.g(x, y, z)) : \mathbf{4} \to \mathbf{1} \qquad s(p) : (\mathbf{4} \to \mathbf{1}) \to \mathbf{1}}\,\text{COMP}$$
$$\dfrac{}{some(present, \lambda z.\ \text{RECIP}(jm, \lambda x \lambda y.gave(x, y, z))) : \mathbf{1}}\,\text{APP}$$

Figure 13.1: Two derivations in "user friendly" style

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{2} \to \mathbf{3} \to \mathbf{4} \to \mathbf{1} \quad [\mathbf{2}]_2}{\mathbf{3} \to \mathbf{4} \to \mathbf{1}} \quad [\mathbf{3}]_3}{\mathbf{4} \to \mathbf{1}} \quad (\mathbf{4} \to \mathbf{1}) \to \mathbf{1}}{\mathbf{1}}}{\mathbf{3} \to \mathbf{1}}\,^3}{\mathbf{2} \to \mathbf{3} \to \mathbf{1}}\,^2 \qquad \dfrac{\mathbf{2} \to (\mathbf{2} \to \mathbf{3} \to \mathbf{1}) \to \mathbf{1} \quad \mathbf{2}}{(\mathbf{2} \to \mathbf{3} \to \mathbf{1}) \to \mathbf{1}}}{\mathbf{1}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{2} \to \mathbf{3} \to \mathbf{4} \to \mathbf{1} \quad [\mathbf{2}]_2}{\mathbf{3} \to \mathbf{4} \to \mathbf{1}} \quad [\mathbf{3}]_3}{\mathbf{4} \to \mathbf{1}} \quad [\mathbf{4}]_4}{\mathbf{1}}}{\mathbf{3} \to \mathbf{1}}\,^3}{\mathbf{2} \to \mathbf{3} \to \mathbf{1}}\,^2 \qquad \dfrac{\mathbf{2} \to (\mathbf{2} \to \mathbf{3} \to \mathbf{1}) \to \mathbf{1} \quad \mathbf{2}}{(\mathbf{2} \to \mathbf{3} \to \mathbf{1}) \to \mathbf{1}}}{\mathbf{1}} \qquad \dfrac{}{(\mathbf{4} \to \mathbf{1}) \to \mathbf{1}}$$
$$\dfrac{}{\mathbf{1}}$$

Figure 13.2: Two linear logic derivations (meaning terms omitted)

Scope ambiguities with negation and modals work in the same way. For example:

(670) John and Mary don't like each other.

$\neg\text{RECIP}(jm, \lambda x \lambda y.like(x, y))$

$\text{RECIP}(jm, \lambda x \lambda y.\neg like(x, y))$

The packed meaning representation for (670) is:

(671) $l_1 : \langle A1 : l_2(\lambda x \lambda y.not(l_3)),\ A2 : not(l_2(\lambda x \lambda y.l_3)) \rangle$

$l_2 : \lambda R.\text{RECIP}(jm, R)$

$l_3 : like(x, y)$

Here *not* could be factored out as well instead of appearing twice, but because it is an atom, we wouldn't get much out of doing that (see step 4 in section 6.5.2).

In the last few examples, I used $john \oplus mary$ directly instead of using the $^{\square}$ version (as in (660)). If we use the $^{\square}$ version, we get more scope ambiguities due to $^{\square}$. For example, instead of (671), we get the following packed meaning representation:

(672)
$$\begin{aligned} l_1 : \langle & A1 : l_2(\lambda u.\ \langle A1.1 : l_3(\lambda x \lambda y.not(l_5)), \\ & \qquad\qquad A1.2 : not(l_4)\rangle\ ), \\ & A2 : not(l_2(\lambda u.l_4))\rangle \\ l_2 : & \lambda P.P^{\square}(jm) \\ l_3 : & \lambda R.\textsc{recip}(u, R) \\ l_4 : & l_3(\lambda x \lambda y.l_5) \\ l_5 : & like(x, y) \end{aligned}$$

This describes three results:

(673)
$$\begin{aligned} A1.1 &: (\lambda u.\textsc{recip}(u, \lambda x \lambda y.not(like(x, y))))^{\square}(jm) \\ A1.2 &: (\lambda u.not(\textsc{recip}(u, \lambda x \lambda y.like(x, y))))^{\square}(jm) \\ A2 &: not((\lambda u.\textsc{recip}(u, \lambda x \lambda y.like(x, y)))^{\square}(jm)) \end{aligned}$$

Options $A1.2$ and $A2$ are logically equivalent due to the fact that $^{\square}$ can only be resolved collectively, since u must be a collective argument of RECIP. We could detect such cases and require that in $(\lambda u.\varphi)^{\square}(\psi)$, if u must be used collectively, then the $^{\square}$ must either be the topmost operator or outscoped only by other $^{\square}$s. This will filter option $A2$ in (673). Then we can also resolve the $^{\square}$ collectively with β-reduction. This will give us (671) back again.

13.3 Scope Flexibility

13.3.1 Overview

Can the k^t in the lexical entry (661) for "each other" be instantiated with the label of a clause that does not immediately contain the reciprocal? The following example has been suggested as evidence that it can:[3]

[3] See (Higginbotham, 1980).

(674) a. [[John and Mary]$_2$ think [[they]$_4$ like [each other]$_5$]$_3$]$_1$

b. 1. $\forall u \in jm.\ think(u, \text{RECIP}(jm, \lambda x \lambda y.like(x, y)))$
2. $\text{RECIP}(jm, \lambda x \lambda y.think(x, like(x, y)))$

The first reading is obtained when k^t is instantiated to $\mathbf{3}^t$, and the second reading can only be obtained if k^t is instantiated to $\mathbf{1}^t$, as we shall see below.

The possibility of a wide-scope RECIP reading comes out more clearly in the sentence "John and Mary think they will defeat each other",[4] where the wide-scope reading in which each thinks s/he will defeat the other makes more sense than the narrow-scope reading in which each thinks an inconsistent thought. For this to work, "they" should somehow co-vary with a singular entity of type e (even though it seems to refer to the plural "John and Mary") so that it can combine with "think" and "like" to form the $e \rightarrow e \rightarrow t$ relation $\lambda x \lambda y.think(x, like(x, y))$. We have already seen a similar situation in the sentence "John and Mary think they will win" in section 12.2.4, and the solution below will be the same.

13.3.2 Analysis

The glue statements we get are:

(675)
$$\begin{array}{ll}
\lambda P.P^{\Box}(entity(l_1, l_1 = \oplus(\{entity(l_2, l_2 = john), & \\
\qquad\qquad entity(l_3, l_3 = mary)\}))) & : (\mathbf{2}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t \\
think & : \mathbf{2}^e \rightarrow \mathbf{3}^t \rightarrow \mathbf{1}^t \\
dref(\hat{\mathbf{4}}^e, \{num(\hat{\mathbf{4}}^e) = pl\}) & : \mathbf{4}^e \\
like & : \mathbf{4}^e \rightarrow \mathbf{5}^e \rightarrow \mathbf{3}^t \\
\lambda z \lambda R.\text{RECIP}(z, R) & : {\mathbf{5}_{ant}}^e \rightarrow ({\mathbf{5}_{ant}}^e \rightarrow \mathbf{5}^e \rightarrow l^t) \rightarrow l^t
\end{array}$$

The first reading of (674) in which RECIP takes narrow scope can only be obtained if $l = \mathbf{3}$ and $\mathbf{5}_{ant} = \mathbf{4}$. We then get:

(676)
$$\begin{array}{l}
\lambda z \lambda R.\text{RECIP}(z, R) : \mathbf{4}^e \rightarrow (\mathbf{4}^e \rightarrow \mathbf{5}^e \rightarrow \mathbf{3}^t) \rightarrow \mathbf{3}^t \\
+like \Rightarrow \lambda z.\text{RECIP}(z, like) : \mathbf{4}^e \rightarrow \mathbf{3}^t \\
+they \Rightarrow \text{RECIP}(dref(\hat{\mathbf{4}}^e, ..), like) : \mathbf{3}^t \\
+think \Rightarrow \lambda u.think(u, \text{RECIP}(dref(\hat{\mathbf{4}}^e, ..), like)) : \mathbf{2}^e \rightarrow \mathbf{1}^t \\
+jm \Rightarrow (\lambda u.think(u, \text{RECIP}(dref(\hat{\mathbf{4}}^e, ..), like)))^{\Box}(entity(l_1, ..)) : \mathbf{1}^t
\end{array}$$

With entity raising, we get:

[4] From (Roberts, 1991), repeated in (Carpenter, 1998, p.361).

(677) $[\langle l_1, \{l_1 = \oplus(l_2, l_3)\}\rangle, \langle l_2, \{l_2 = john\}\rangle, \langle l_3, \{l_3 = mary\}\rangle \mid$
$(\lambda u.think(u, \text{RECIP}(dref(\hat{\mathbf{4}}^e, ..), like)))^{\square}(l_1)]$

Now, $^{\square}$ should be resolved distributively (because of world knowledge, we know that *think* can hold only on individuals and not on collectives, unless we are in a science fiction situation). Also, because $dref(\hat{\mathbf{4}}^e, ..)$ is the first argument of RECIP, it must not be resolved to an individual, so its antecedent cannot be u and must be l_1. This gives us:

(678) $[\langle l_1, \{l_1 = \oplus(l_2, l_3)\}\rangle, \langle l_2, \{l_2 = john\}\rangle, \langle l_3, \{l_3 = mary\}\rangle \mid$
$\forall y \in l_1.think(y, \text{RECIP}(l_1, like))]$
= John thinks 'John and Mary like each other', and Mary thinks the same.

The second reading of (674) in which RECIP takes wide scope can only be obtained if $l = \mathbf{1}$ and $\mathbf{5}_{ant} = \mathbf{2}$.[5]

(679) $\lambda R\lambda z.\text{RECIP}(z, R) : (\mathbf{2}^e \to \mathbf{5}^e \to \mathbf{1}^t) \to \mathbf{2}^e \to \mathbf{1}^t$
$they + like \Rightarrow \lambda y.like(dref(\hat{\mathbf{4}}^e, ..), y) : \mathbf{5}^e \to \mathbf{3}^t$
$+think \Rightarrow \lambda x\lambda y.think(x, like(dref(\hat{\mathbf{4}}^e, ..), y)) : \mathbf{2}^e \to \mathbf{5}^e \to \mathbf{1}^t$
$+recip \Rightarrow \lambda z.\text{RECIP}(z, \lambda x\lambda y.think(x, like(dref(\hat{\mathbf{4}}^e, ..), y))) : \mathbf{2}^e \to \mathbf{1}^t$
$+jm \Rightarrow (\lambda z.\text{RECIP}(z, \lambda x\lambda y.think(x, like(dref(\hat{\mathbf{4}}^e, ..), y))))^{\square}(entity(l_1, ..)) : \mathbf{1}^t$

With entity raising, we get:

(680) $[\langle l_1, \{l_1 = \oplus(l_2, l_3)\}\rangle, \langle l_2, \{l_2 = john\}\rangle, \langle l_3, \{l_3 = mary\}\rangle \mid$
$(\lambda z.\text{RECIP}(z, \lambda x\lambda y.think(x, like(dref(\hat{\mathbf{4}}^e, ..), y))))^{\square}(l_1)]$

Now, $^{\square}$ must be resolved collectively because z is the first argument of RECIP. If $dref(\hat{\mathbf{4}}^e, ..)$ is resolved to l_1 then we get the (less likely) reading where John thinks that John and Mary like Mary and Mary thinks that John and Mary like John. This time, however, $dref(\hat{\mathbf{4}}^e, ..)$ can also be resolved to x, just like $dref(l_4)$ was resolved to x in (634)-(635) of section 12.2.4. Since x is 'closer' than l_1 to $dref(\hat{\mathbf{4}}^e)$, this resolution is preferred, giving us:

(681) $[\langle l_1, \{l_1 = \oplus(l_2, l_3)\}\rangle, \langle l_2, \{l_2 = john\}\rangle, \langle l_3, \{l_3 = mary\}\rangle \mid$
$\text{RECIP}(l_1, \lambda x\lambda y.think(x, like(x, y)))]$
= John thinks that he likes Mary and Mary thinks she likes John.

[5] See (686) below for a justification of this option.

What prevents $dref(\hat{\mathbf{4}}^e)$ in (680) from being resolved to y? In the sentence, $dref$ originates from the NP *they*, which linearly precedes the NP *each other* from which y originates. Therefore we have a constraint that *they* cannot be anaphoric to the later NP. This constraint in terms of syntactic positions induces a constrain in terms of glue categories. All we then need is to remember the glue category from which each variable came. When the algorithm compiles the RECIP premise in (674) and creates $y : \mathbf{5}^e$, it can mark the glue category on the variable, i.e. $y^{[\mathbf{5}]} : \mathbf{5}^e$, and similarly with $x^{[\mathbf{2}]} : \mathbf{2}^e$ that is compiled out of the first premise there. Then, $y^{[\mathbf{5}]}$ cannot serve as the antecedent of $dref(\hat{\mathbf{4}}^e)$ because of the constraint $not\text{-}ant(\mathbf{5}, \mathbf{4})$ (which comes from the fact that the NP *each other* appears linearly after the NP *they* in the sentence). In contrast, $x^{[\mathbf{2}]}$ can serve as the antecedent of $dref(\hat{\mathbf{4}}^e)$.

What about the two remaining options for the values of l and $\mathbf{5}_{ant}$? If $l = \mathbf{3}$ and $\mathbf{5}_{ant} = \mathbf{2}$, then the premises cannot combine in any glue derivation.[6] However, if $l = \mathbf{1}$ and $\mathbf{5}_{ant} = \mathbf{4}$, we actually get a third reading:

(682) $\lambda z \lambda R.\text{RECIP}(z, R) : \mathbf{4}^e \to (\mathbf{4}^e \to \mathbf{5}^e \to \mathbf{1}^t) \to \mathbf{1}^t$
$think + like \Rightarrow \lambda z \lambda x \lambda y.think(z, like(x, y)) : \mathbf{2}^e \to \mathbf{4}^e \to \mathbf{5}^e \to \mathbf{1}^t$
$+recip \Rightarrow \lambda z \lambda u.\text{RECIP}(u, \lambda x \lambda y.think(z, like(x, y))) : \mathbf{2}^e \to \mathbf{4}^e \to \mathbf{1}^t$
$+they \Rightarrow \lambda z.\text{RECIP}(dref(\hat{\mathbf{4}}^e, ..), \lambda x \lambda y.think(z, like(x, y))) : \mathbf{2}^e \to \mathbf{1}^t$
$+jm \Rightarrow (\lambda z.\text{RECIP}(dref(\hat{\mathbf{4}}^e, ..), \lambda x \lambda y.think(z, like(x, y))))^{\Box}(entity(l_1, ..)) : \mathbf{1}^t$

With entity raising, we get:

(683) $[\langle l_1, \{l_1 = \oplus(l_2, l_3)\}\rangle, \langle l_2, \{l_2 = john\}\rangle, \langle l_3, \{l_3 = mary\}\rangle \mid$
$(\lambda z.\text{RECIP}(dref(\hat{\mathbf{4}}^e, ..), \lambda x \lambda y.think(z, like(x, y))))^{\Box}(l_1)]$

Since z is the first argument of *think*, it should be individual, and so $^{\Box}$ should be resolved distributively. In contrast, $dref(\hat{\mathbf{4}}^e)$ is the first argument of RECIP and should refer to a collection, so it cannot be resolved to z but only to l_1. Thus we get:

(684) $[\langle l_1, \{l_1 = \oplus(l_2, l_3)\}\rangle, \langle l_2, \{l_2 = john\}\rangle, \langle l_3, \{l_3 = mary\}\rangle \mid$
$\forall z \in l_1.\text{RECIP}(l_1, \lambda x \lambda y.think(z, like(x, y)))]$
= John thinks that John likes Mary and John thinks that Mary likes John and
Mary thinks that John likes Mary and Mary thinks that Mary likes John.

[6] To see this, note that $\mathbf{4}^e$ would have to combine with $\mathbf{4}^e \to \mathbf{5}^e \to \mathbf{3}^e$ to give $\mathbf{5}^e \to \mathbf{3}^e$. But this cannot fit as the second argument of RECIP : $\mathbf{2}^e \to (\mathbf{2}^e \to \mathbf{5}^e \to \mathbf{3}^t) \to \mathbf{3}^t$. It will not help to try to first combine $\mathbf{5}^e \to \mathbf{3}^e$ with $\mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t$.

This result is obviously different from (681), but it is also not the same as (678) because although the propositions $\textsc{recip}(a \oplus b, R)$ and $R(a,b) \wedge R(b,a)$ are logically equivalent, they are not the same proposition and not the same thought (a person thinking a proposition may not also think all its logically equivalent propositions). Is this third reading available for the sentence? It is hard to say. It seems we would want to block it. This is why we required in (661) that the clause over which the reciprocal scopes is the minimal clause containing the reciprocal's antecedent. This is a natural requirement, as we saw in the discussion of Problem 3 in section 8.2.3. This requirement does not allow us to instantiate $l = \mathbf{1}$ if $\mathbf{5}_{ant} = \mathbf{4}$.

One issue that remains to be mentioned: How can we justify the equation $\mathbf{5}_{ant} = \mathbf{2}$ in the first solution? It seems at first that anaphoric constraints on reciprocals are similar to those for reflexives. For example:

(685) a. John likes himself. But not: Himself likes John.

b. John and Mary like each other. But not: Each other like John and Mary.

c. John thinks that Mary likes him/*himself.

d. John and Mary think that Bill likes them/*each other.

Therefore, it seems at first that "each other" cannot be directly anaphoric to "John and Mary". However, "each other", unlike a reflexive pronoun, is also a quantifier that can take scope. To allow for that while still taking account of (685)d, we can specify the constraints in (661) as follows:[7]

(686) The *immediate* antecedent label α_1 of a reciprocal must satisfy the same constraints as the antecedent label of a reflexive (including: α_1 is in the minimal clause containing the reciprocal). This label starts a chain $\alpha_1, \alpha_2, \ldots, \alpha_m$ of labels of anaphoric expressions where for all $1 \leq j < m$, α_j labels an expression which is anaphoric to the expression that α_{j+1} labels. Then the *ultimate* antecedent label of the reciprocal (marked a in (661)) can be any label α_j along the chain. (In practice, m is rarely greater than 2.)

13.3.3 Interaction with Equi and Raising Verbs

Does a reciprocal exhibit a similar kind of scope flexibility when it interacts with raising and equi (control) verbs, as in (687)? A-priori it seems like it might since there are two

[7] Cf. (Higginbotham, 1980).

clauses: the main clause and the embedded clause which is the argument of the main verb. So we now turn to investigating this interaction.

(687) a. John and Mary tried to hit each other.

b. John and Mary seem to like each other.

We discussed the glue analysis of equi and raising verbs in section 4.4.5. The f-structures for (687) are:

(688) a.
$$\begin{bmatrix} \text{PRED} & \text{`SEEM}\langle\text{XCOMP}\rangle\text{SUBJ'} \\ \text{SUBJ} & \boxed{2}[\text{“John and Mary”}] \\ \boxed{1} & \\ \text{XCOMP} & \boxed{3}\begin{bmatrix} \text{PRED} & \text{`LIKE}\langle\text{SUBJ, OBJ}\rangle\text{'} \\ \text{SUBJ} & \boxed{2} \\ \text{OBJ} & \boxed{4}[\text{“each other”}] \end{bmatrix} \end{bmatrix}$$

b.
$$\begin{bmatrix} \text{PRED} & \text{`TRY}\langle\text{SUBJ, XCOMP}\rangle\text{'} \\ \text{SUBJ} & \boxed{2}[\text{“John and Mary”}] \\ \boxed{1} & \\ \text{XCOMP} & \boxed{3}\begin{bmatrix} \text{PRED} & \text{`LIKE}\langle\text{SUBJ, OBJ}\rangle\text{'} \\ \text{SUBJ} & \boxed{2} \\ \text{OBJ} & \boxed{4}[\text{“each other”}] \end{bmatrix} \end{bmatrix}$$

The glue statements for (688)a should be:

(689) $jm : \mathbf{2}^e$

$seem : \mathbf{3}^t \rightarrow \mathbf{1}^t$

$like : \mathbf{2}^e \rightarrow \mathbf{4}^e \rightarrow \mathbf{3}^t$

$\lambda z \lambda R.recip(z, R) : {\mathbf{4}_{ant}}^e \rightarrow ({\mathbf{4}_{ant}}^e \rightarrow \mathbf{4}^e \rightarrow k^t) \rightarrow k^t$

The antecedent of “each other” is “John and Mary” (syntactically, we might need to say that the antecedent of “each other” is the SUBJ of the embedded clause 3, but 3 is structure-shared with the SUBJ of the main clause 1). Hence $\mathbf{4}_{ant} = \mathbf{2}$. According to (661), k should be the label of the minimal clause containing a. But does that mean 1 only, or is 3 also possible?

(690) If $k = \mathbf{1}$ then:

$\lambda z \lambda R.recip(z, R) : \mathbf{2}^e \rightarrow (\mathbf{2}^e \rightarrow \mathbf{4}^e \rightarrow \mathbf{1}^t) \rightarrow \mathbf{1}^t$

$seem + like \Rightarrow \lambda x \lambda y.seem(like(x, y)) : \mathbf{2}^e \rightarrow \mathbf{4}^e \rightarrow \mathbf{1}^t$

$+recip \Rightarrow \lambda z.recip(z, \lambda x \lambda y.seem(like(x, y))) : \mathbf{2}^e \rightarrow \mathbf{1}^t$

$+jm \Rightarrow recip(jm, \lambda x \lambda y.seem(like(x, y))) : \mathbf{1}^t$

= “John seems to like Mary and Mary seems to like John”

(691) If $k = \mathbf{3}$ then:

$\lambda z \lambda R.recip(z, R) : \mathbf{2}^e \to (\mathbf{2}^e \to \mathbf{4}^e \to \mathbf{3}^t) \to \mathbf{3}^t$

$+like \Rightarrow \lambda z.recip(z, like) : \mathbf{2}^e \to \mathbf{3}^t$

$+jm \Rightarrow recip(jm, like) : \mathbf{3}^t$

$+seem \Rightarrow seem(recip(jm, like)) : \mathbf{1}^t$

= "It seems that John and Mary like each other"

It is not easy to tell whether only one of the options gives the right truth conditions. What about the equi verb?

(692) $jm : \mathbf{2}^e$

$\lambda x \lambda P.try(x, P) : \mathbf{2}^e \to (\mathbf{2}^e \to \mathbf{3}^t) \to \mathbf{1}^t$

$hit : \mathbf{2}^e \to \mathbf{4}^e \to \mathbf{3}^t$

$\lambda z \lambda R.recip(z, R) : \mathbf{2}^e \to (\mathbf{2}^e \to \mathbf{4}^e \to k^t) \to k^t$

(693) If $k = \mathbf{1}$ then:

$\lambda z \lambda R.recip(z, R) : \mathbf{2}^e \to (\mathbf{2}^e \to \mathbf{4}^e \to \mathbf{1}^t) \to \mathbf{1}^t$

(assumption) $[v : \mathbf{4}]$

$+hit \Rightarrow \lambda y.hit(y, v) : \mathbf{2}^e \to \mathbf{3}^t$

$+try \Rightarrow \lambda x.try(x, \lambda y.hit(y, v)) : \mathbf{2}^e \to \mathbf{1}^t$

(discharge assumption + exchange) $\Rightarrow \lambda x \lambda v.try(x, \lambda y.hit(y, v)) : \mathbf{2}^e \to \mathbf{4}^e \to \mathbf{1}^t$

$+recip \Rightarrow \lambda z.recip(z, \lambda x \lambda v.try(x, \lambda y.hit(y, v))) : \mathbf{2}^e \to \mathbf{1}^t$

$+jm \Rightarrow recip(jm, \lambda x \lambda v.try(x, \lambda y.hit(y, v))) : \mathbf{1}^t$

= "John tried to hit Mary and Mary tried to hit John"

(694) If $k = \mathbf{3}$ then:

$\lambda z \lambda R.recip(z, R) : \mathbf{2}^e \to (\mathbf{2}^e \to \mathbf{4}^e \to \mathbf{3}^t) \to \mathbf{3}^t$

$+hit \Rightarrow \lambda z.recip(z, hit) : \mathbf{2}^e \to \mathbf{3}^t$

$+try \Rightarrow \lambda x.try(x, \lambda z.recip(z, hit)) : \mathbf{2}^e \to \mathbf{1}^t$

$+jm \Rightarrow try(jm, \lambda z.recip(z, hit)) : \mathbf{1}^t$

= "John and Mary collectively tried to bring it about that John and Mary would hit each other" (or distributively, "John tried to bring it about that John and Mary would hit each other, and so did Mary")

The second option seems much less likely (and is unavailable according to Higginbotham (1980)). So we can reach the conclusion that although the SUBJ of the main clause is

structure-shared with that of the embedded clause, its more prominent place is in the main clause 1 while its place in the embedded clause is dependent. Therefore, when the lexical schema for the reciprocal refers to "the minimal clause containing my antecedent", it means only the main clause.

13.4 Type Flexibility

Higginbotham (1980) points out that a reciprocal may also scope around an NP, not just a clause:

(695) a. Bill likes [John and Mary]$_i$'s pictures of [each other]$_i$.
$like(bill, \text{RECIP}(jm, \lambda u\lambda v.poss(u, \lambda z.picture\text{-}of(z, v))))$
b. [[John and Mary]$_i$'s pictures of [each other]$_i$] collided.
$collide(\text{RECIP}(jm, \lambda u\lambda v.poss(u, \lambda z.picture\text{-}of(z, v))))$

Here, *picture-of* is of type $e \to e \to t$ and *poss* is of type $e \to (e \to t) \to e$, i.e. $poss(a, P)$ returns the element x such that $P(x)$ and x belongs to a (or is related to a in some other way).

The RECIP here is used to form a plurality of type e. If R is of type $e \to e \to e$ then $\text{RECIP}(a \oplus b, R)$ means $R(a, b) \oplus R(b, a)$. So (695)a can be expanded to (696)a. Although in this case, this is equivalent to (696)b, it is only thanks to the particular distributive perdicate *like*. The collective predicate *collide* in (695)b shows that in general, the sentence cannot be reduced to a reciprocal that returns a proposition.

(696) a. $like(bill,\ poss(john, \lambda z.picture\text{-}of(z, mary)) \oplus poss(mary, \lambda z.picture\text{-}of(z, john)))$
b. $like(bill, poss(john, \lambda z.picture\text{-}of(z, mary))) \wedge like(bill, poss(mary, \lambda z.picture\text{-}of(z, john)))$

The conclusion is that RECIP is polymorphic, just like *and* and *or* are. Therefore, (661) should be revised:

(697) $\lambda z\lambda R.\text{RECIP}(z, R) : a^e \to (a^e \to l^e \to k^\tau) \to k^\tau$
where l is my label, k is the label of the expression α that I scope over,
and a is my (ultimate) antecedent's label;
where if $\tau = t$ then α is a clause, and if $\tau = e$ then α is a NP;
and where $k =$ the label of the minimal expression containing a,
and [some more constraints on a]

Interestingly, when RECIP shows scope flexibility as in section (13.3), it shows type flexibility at the same time:

(698) John and Mary like their pictures of each other.

a. $like(jm, \text{RECIP}(jm, \lambda u \lambda v.poss(u, \lambda z.picture\text{-}of(z, v))))$

b. $\text{RECIP}(jm, \lambda u \lambda v.like(u, poss(u, \lambda z.picture\text{-}of(z, v))))$

I assume that *their* can be analyzed as *they's*, so the discussion about the anaphoric chain *each other* → *they* → *John and Mary* from (686) carries over. In (698)a, RECIP takes scope over the NP and therefore returns a plurality of type e. But in (698)b, it takes scope over the sentence, so it returns type t. I assume that in:

(699) John and Mary think their pictures of each other collided.

a. $think(jm, collide(\text{RECIP}(jm, \lambda u \lambda v.poss(u, \lambda z.picture\text{-}of(z, v)))))$

b. $\text{RECIP}(jm, \lambda u \lambda v.think(u, collide(poss(u, \lambda z.picture\text{-}of(z, v)))))$

the second reading is unavailable not because the syntax-semantics interface blocks it but because of the meaning of *collide*, just as "John collided" is blocked not by the syntax-semantics interface but by the meaning of *collide*.

13.5 Interaction with a Quantified Antecedent

For completion's sake, I should mention the question: what happens when the reciprocal's antecedent NP is quantified as in:

(700) At least five students stared at each other.

At most five students stared at each other.

The issue is that *stare at each other* is a property of pluralities, but *at least/most five students* as we saw in chapter 10 expects a property of individuals as its argument.

This question is an instance of the more general question: What happens when we apply a type $\langle 1, 1 \rangle$ quantifier Q on a noun denoting a set A and on a collective predicate P, i.e. what "Q A P" means. According to (van der Does, 1993; Dalrymple et al., 1998) and (Peters and Westerståhl, 2006, section 10.4.4), if Q is monotonically increasing in its right argument then the meaning of the combination is (701)a whereas if Q is monotonically decreasing, the meaning is (701)b.

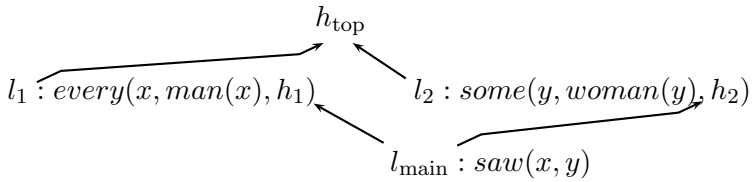


Figure 13.3: Hole semantics ULF for "Every man saw some woman."

(701) a. $\text{C}^{\uparrow}(Q, A, P) \;:=\; \exists X \subseteq A.[P(X) \wedge Q(A, X)]$

b. $\text{C}^{\downarrow}(Q, A, P) \;:=\; \forall X \subseteq A.[P(X) \rightarrow Q(A, X)]$

These are dual in the sense that:

(702) $\text{C}^{\downarrow}(Q, A, P) \equiv \neg\text{C}^{\uparrow}(\neg Q, A, P)$

For example, (703)a means there is a set of students who collaborated and the size of this set is at least 4, while (703)b means that any set of students who collaborated has at most four members.

(703) a. At least four students collaborated.

$\exists X \subseteq \textit{student}.\ [\textit{collab}(X) \wedge \textit{at-least}[4](\textit{student}, X)]$

b. No more than four students collaborated.

$\forall X \subseteq \textit{student}.\ [\textit{collab}(X) \rightarrow \textit{at-most}[4](\textit{student}, X)]$

For more on this issue, see the references cited above.

13.6 Comparison to Other Frameworks

13.6.1 Comparison to Hole Semantics

We can now return to the point of "unprincipled" from section 7.7.3 and demonstrate it more clearly. As we saw in section 7.7.3, the sentence "Every man saw some woman" would get the Hole Semantics ULF in Figure 13.3.

Now suppose we want to try analyzing "John and Mary saw each other" in Hole Semantics (HS). The final logical form is: $\textsc{recip}(\textit{john} \oplus \textit{mary}, \lambda x \lambda y.\textit{saw}(x, y))$. In HS, we need to break this syntactic object (the formula) into smaller *syntactic* pieces, i.e. just as they appear in the formula. We can try to break this formula into the following pieces and label them:

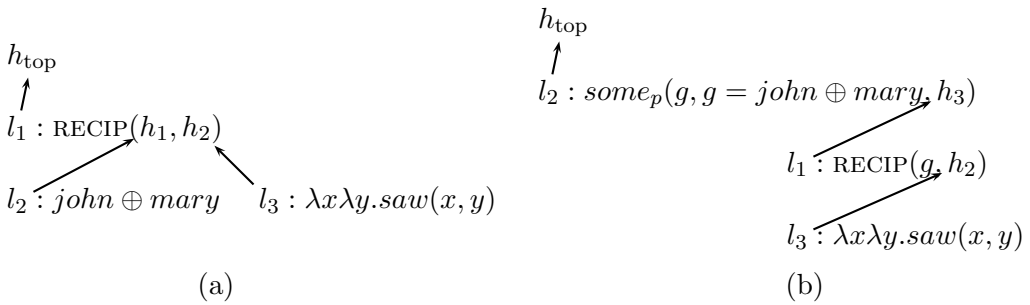


Figure 13.4: Possible ULFs for "John and Mary saw each other."

(704) a. $l_1 : \text{RECIP}(h_1, h_2)$
b. $l_2 : john \oplus mary$
c. $l_3 : \lambda x \lambda y.saw(x, y)$

A ULF from these pieces might look like Figure 13.4(a). Notice that the transitive verb somehow has to contribute the piece $l_3 : \lambda x \lambda y.saw(x, y)$, whereas in Figure 13.3 there is no such piece, only $saw(x, y)$. Another difference is that all pieces that appear in ULFs in the literature on HS represent propositions, but in Figure 13.4(a) the pieces l_2 and l_3 are not propositions: the first is a set and the second is a binary relation. We might be able to fix this issue for $john \oplus mary$ by assuming it is introduced by a quantifier $some_p$, a plural-version of *some*, as shown in Figure 13.4(b), but we cannot do a similar thing with $\lambda x \lambda y.saw(x, y)$ because we genuinely require access to the binary relation itself.

Consequently, when we consider how the ULFs are constructed by the HS grammar, we discover that the semantic grammar rules required to obtain Figure 13.3 are very different from those required for Figure 13.4. The standard HS grammar (section 7.7.2) is constructed in such a way that the main transitive verb expects to combine with a generalized quantifier of type $(e \to t) \to t$ for the direct object, so it does not expect a polyadic quantifier such as RECIP in the syntactic place of the direct object, as in "John and Mary saw each other". The grammar entries are written to produce the labelled piece $saw(x, y)$ and not what we need: $\lambda x \lambda y.saw(x, y)$. Even if we try to put $\lambda x \lambda y.h_2$ as the second argument of $l_1 : \text{RECIP}$ in Figure 13.4(b), we cannot get this result by the usual play of the local hole and local label in the standard HS grammar.

Glue semantics can handle reciprocals very naturally because it is a framework for specifying the combination of *semantic* objects (which are represented by expressions with a well-defined denotation). The difficulties in HS can be traced to the fact that it operates

at the level of *syntactic* manipulations of formulas, without a principled theory that justifies these manipulations. They happen to work for some cases but not for others, especially when higher-order operators such as RECIP are involved.

13.6.2 Comparison to Categorial Grammar

I already mentioned in section 3.5.2 under "elegant treatment of non-contiguous components" that Categorial Grammar requires adding a special mechanism to the system in order to deal with reciprocals, because the limitation of the direction-sensitive logic of CG needs to be circumvented (the required special mechanism is slightly more complicated than the one shown in (128) – see (Carpenter, 1998, ch.9) for the details). In contrast, as we have seen above, we can account for reciprocals in glue semantics very naturally and no special additions are needed.

Chapter 14

Covert Reciprocals

14.1 Overview

Sometimes a reciprocal may be expressed covertly. However, not all cases that seem like they have a covert reciprocal are indeed so. In particular, I will discuss the questions for each of the pairs in (705): What is the connection between the two sentences? Do they mean the same? If so, is one of them basic and one derived? Are all these pairs instances of the same kind of alteration?

(705) a1. John and Mary collided.
a2. John and Mary collided with each other.
b1. John and Mary kissed.
b2. John and Mary kissed each other.
c1. John and Mary are similar.
c2. John and Mary are similar to each other.
d1. John and Mary read the same book.
d2. John and Mary read the same book as each other.

Below, I will need the following definition:

(706) The *reciprocal completion* of $R(a, b)$ is $\textsc{recip}(a \oplus b, R)$.

For example, (707)b is the reciprocal completion of (707)a when $R = saw$.

(707) a. John saw Mary.
b. John and Mary saw each other.

14.2 Two Kinds of Predicates

14.2.1 Group-Monadic Predicates

According to Ginzburg (1990), there are two unrelated kinds of symmetric alterations: those based on an inherently group-monadic predicate such as *collide*, and those based on an inherently singular-dyadic predicate. I discuss the latter in section 14.2.2.

A *comitative construction* is a variation on a collective predicate, where the plurality argument to the predicate is split. For example, (708)b is the comitative version of (708)a.

(708) a. John and Mary lifted the table (together).
b. John lifted the table (together) with Mary.

Ginzburg (1990) points out that group-monadic predicates usually have a comitative construction version and shows this for inherently collective predicates expressing joint activity such as *meet*, *fight*, *correspond*, *compete*, *collaborate*, *copluate*, *coexist*, *play chess*, *agree*, *have an affair*, as in (709)a,b.

(709) a. John and Mary collided.
b. John collided with Mary.

According to Ginzburg, for this kind of predicates, the group-monadic predicate is basic and the comitative construction is derivative. In a comitative construction, the main predicate is collective, but the elements of the group are expressed separately because one element of the group is 'focussed' above the other(s). Ginzburg suggests (in the framework of Situation Semantics) that pairs of sentences such as (709)a,b express different propositions but the two have the same truth conditions, and this is enforced by a postulate on the meaning of the relation *WITH* (which takes two arguments: the NP of the PP, and the VP).

The comitative version of a group-monadic predicate is itself a dyadic predicate, and so it can undergo reciprocal completion. The reciprocal completion of (709)b is:

(710) John and Mary collided with each other.

The conclusion is that (709)a is not derived by ellipsis from (710).

There is another kind of variation, which Ginzburg does not discuss:

(711) a. John and Mary kissed.

b. ? John kissed with Mary.

c. ? John and Mary kissed with each other.

d. John kissed Mary.

e. John and Mary kissed each other.

For some reason, group-monadic *kiss* does not like to use the comitative construction. There is a transitive version of *kiss* whose meaning is that the agent kissed the patient, not necessarily on the lips, and the patient did not necessarily kiss back, as in (711)d. This version can undergo reciprocal completion, as in (711)e.

Dimitriadis (2004) notes (following Schwarzschild and others) that while (711)e could mean two kissing events, (711)a is "irreducibly symmetric" (this is Dimitriadis' term) and can mean only one event of simultaneous kissing on the lips. This difference also comes up when we examine scope flexibility: (712)a is unambiguous and has only one reading because *kiss* is just group-monadic, while (712)b is ambiguous as we saw in section 13.3.[1]

(712) a. John and Mary said they kissed.

$say(jm, kiss_{iv}(jm))$

b. John and Mary said they kissed each other.

$say(jm, \text{RECIP}(jm, kiss_{tv}))$

$\text{RECIP}(jm, \lambda x \lambda y.say(x, kiss_{tv}(x, y)))$

To sum up, there is nothing special that needs to be said about the reciprocal beyond what needs to be said about the comitative constructions and lexical alterations between intransitive and transitive versions of verbs like *kiss*. In particular, there is no covert reciprocal operator at work in (705)a1 or (705)b1, i.e. these sentences are not derived by ellipsis from (705)a2 or (705)b2.

14.2.2 Singular-Dyadic Predicates

In contrast to group-monadic predicates, Ginzburg claims that in symmetric singular-dyadic predicates, the singular form is basic, as in (713)a, whereas the collective predicate, as in (713)b, is derivative. This is explained below.

[1] This fact is noted by Siloni (2005).

(713) a. John is similar to Mary.
b. John and Mary are similar.

Examples of singular-dyadic predicates are: *similar*, *synonymous*, *different*, *parallel*, *perpendicular*, *adjacent*, *equal*, *equivalent*, *identical*, *isomorphic*, *distinct*, *disjoint*, *separate*, *related*, *unrelated*, *opposite*, *near*.

First, how does one tell whether a predicate is group-monadic or singular-dyadic? Ginzburg points to several differences, and I repeat here only some of them:

1. The dyadic form of group-monadic predicates always uses the preposition *with* since this is in fact the comitative construction. In contrast, singular-dyadic predicates use various prepositions:

   (714) a. This is similar / equal / isomorphic *to* that.
   b. This is different / separate / disjoint *from* that.
   c. This is the same *as* that.

2. Predicates whose basic form is group-monadic are odd with a reflexive object whereas predicates whose basic form is singular-dyadic are not. Compare:

   (715) a. ? John collaborated with himself.
   b. This line is parallel to itself.

3. Predicates whose basic form is group-monadic do not take null complements as singular-dyadic ones do. (716)a can only have the reciprocal reading where John and Mary fight each other, and it cannot have the null complement reading where they fight Bill (that is why this discourse is odd). In contrast, (716)b could have the null complement reading where John and Mary are different from Bill (they dislike ice-cream), and (716)c could have the reciprocal reading as usual.

   (716) a. # Bill is very belligerent. John and Mary fight all the time.
   b. Bill likes ice-cream, but John and Mary are different.
   c. John and Mary are different (from each other).

There seems to be another difference, which Ginzburg did not point out: group-monadic predicates are verbs while singular-dyadic ones are adjectives or adverbs. This seems to be

a very robust test, with only the following exception. For a few adjectives, it is not clear whether they are singular-dyadic and have a PP argument with preposition *with* or they are group-monadic, and can participate in a comitative construction:

(717) a. Regulations 4 and 9 are compatible (with each other).

b. Regulation 4 is compatible with regulation 9.

Other such adjectives are: *(mutually) incompatible*, *co-extensive*, and *inter-twined*. Second, two adjectives – *inter-dependent* and *pairwise disjoint* – can neither take an additional argument nor participate in a comitative construction. But all these seem to have a part that expresses reciprocation: *co-*, *inter-*, *pairwise*, *mutually*, so that might explain why they are not singular-dyadic.

Now, how do we get the group-monadic form of singular-dyadic predicates, e.g. (713)b from (713)a? First, if the reciprocal is overt as in (718) then there is no issue. (718) is simply the reciprocal completion of (713)a, and the usual treatment we saw in the previous chapter works.

(718) John and Mary are similar to each other.

As for (713)b, Ginzburg suggests that the group-monadic form of a singular-dyadic predicate is obtained through a *reciprocalization* operator that applies on that predicate. To generalize his proposal (which uses the strongers reading of RECIP, namely Full Reciprocity), this operator, call it *rcp*, is defined:

(719) For a binary relation R, $R^{rcp} := \lambda z.\textsc{recip}(z, R)$

For example, (713)a should be analyzed as $similar^{rcp}(john \oplus mary)$.

There is a point, however, that needs to be addressed and which is missing from Ginzburg's account. If there is a reciprocalization operator, it can sometimes apply on non-atomic predicates. The first case where we might be able to show this is:

(720) a. John and Mary think they are similar to each other.

$think(jm, \textsc{recip}(jm, similar))$

$\textsc{recip}(jm, \lambda x \lambda y.think(x, similar(x, y)))$.

b. John and Mary think they are similar.

That (720)a has the two readings shown follows from what was said in section 13.3. The two readings are very similar in the present case because it is hard to imagine how a

person could think $similar(a, b)$ without also thinking $similar(b, a)$ given that *similar* is a symmetric relation. So it is hard to tell whether the two readings are available for (720)b. But if we use a subject that has three members, such as "John, Bill, and Mary", the two readings become more clearly distinct: if John, Bill and Mary think they are similar to each other, then John can think that he is similar to Mary and that he is similar to Bill, while not necessarily thinking at all about the similarity of Mary and Bill. It is not clear whether this reading is also available when the reciprocal is covert. It think that with the right context, it can be available.

A clearer example of the technical problem with the reciprocalization operator has to do with *same* and *different* in the following examples:

(721) a. John and Mary read the same book (as each other).

b. Men and women have a different sense of humour (than each other).[2]

These examples show that we cannot assume the reciprocalization operator applies only on lexical predicates: here it needs to apply on the complex predicates:

(722) $\lambda x \lambda y.$[x read the same book as y (did)]

$\lambda x \lambda y.$[x have a different sense of humour than y (do)]

These issues lead me to propose that there is no reciprocalization operator, but instead an omission. I propose the following claim:

(723)

Symmetric adjectives and comparative adjectives may drop their complement "p X" (where p is a preposition or *than*) if X is an expression that is anaphoric to an entity salient enough in the context. This includes the case where X is a reciprocal.

This theory gives a unified explanation for the possibility to drop all the complements in the following sentences (given the right context):

(724) a. John is 6 feet tall. But Bill is taller (than him/that).

b. Bill likes spicy foods. John is similar (to Bill).

c. Bill read "War and Peace". John read the same book (as that) / a different book (than that).

b. These two are similar (to each other) but this one is distinct (from them).

[2] http://newmediasphere.blogs.com/nms/2005/02/men_and_women_h.html 1-mar-2005

The upshot is that during interpretation, if a symmetric or comparative adjective is encountered which does not have its expected complement, that complement should be reconstructed as "*p* that/him/her/..." or "*p* each other". In particular, in contrast to what we said regarding (705)a1-b2, we conclude that (705)c1 and (705)d1 have a covert reciprocal, and these sentences are obtained from (705)c2 and (705)d2 by omitting the reciprocal.

In the next section, I give more details about what's involved with *same* and *different* and their interaction with overt and covert reciprocals, and show further support for this theory.

14.3 Details of the Dyadic Adjectives

We saw in chapter 11 that an adjective can appear in a predicative version, and can be modified:

(725) a. John is tall.
b. John is 6 foot tall.
c. John is very tall.

Similarly, the singular-dyadic adjectives mentioned above can too:

(726) a. This book is similar to that book.
b. This set is disjoint from that set.

They cannot be modified by a degree because there is no gradable scale associated with them, but they can be modified by general intensifiers:

(727) a. * This book is ten centimeters different from that book.
b. This book is very different from that book.

We also saw that an adjective can appear in a relative clause and in an attributive version:

(728) a. A man who was tall arrived.
b. A man who was 6 foot tall arrived.
c. A man who was very tall arrived.

(729) a. A tall man arrived.
b. A 6 foot tall man arrived.
c. A very tall man arrived.

Similarly with singular-dyadic adjectives:

(730) a. John drew a line that was perpendicular to the line Mary drew.
b. John drew a line that was (very) different from the line Mary drew.
c. John drew a perpendicular line to the line Mary drew.
d. John drew a (very) different line from the line Mary drew.

The fact that the singular-dyadic adjectives take a complement, where the adjective appears pre-nominally and the complement post-nominally, is similar to other adjectives such as *easy*:

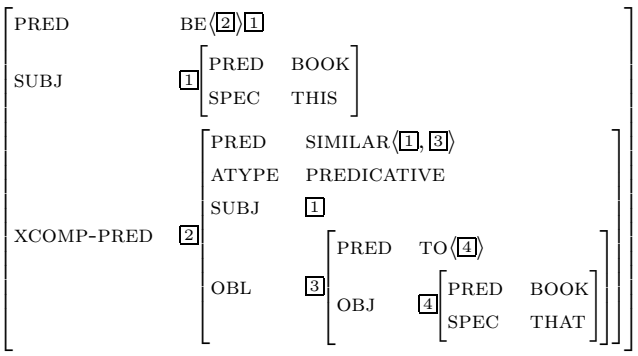
(731) a. John is easy to please.
b. An easy man to please arrived.

(Similar adjectives are: *difficult to understand*, *costly to replace*.) If such an adjective appears pre-nominally and its complement post-nominally, they do not like to appear with other quantifiers (probably because of processing load and the existence of simpler paraphrases):

(732) ? Every easy man to please arrived.
? John read every similar book to "War and Peace".
(ok) Every man who is easy to please arrived.
(ok) Every easy-to-please man arrived.

The F-structure of the predicative version is similar to (514), except that the adjective also has an OBL argument:

(733) This book is similar to that book.

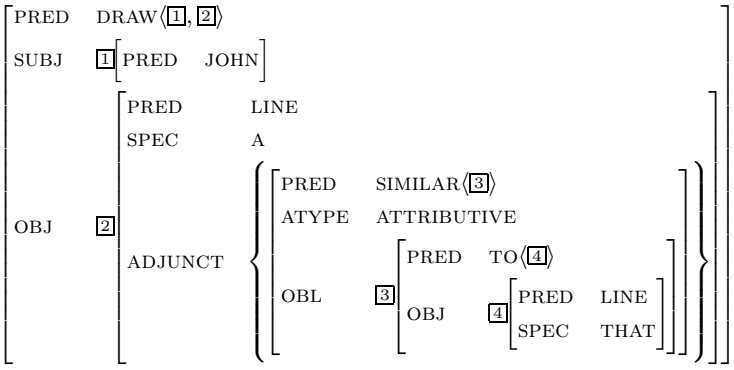


The meaning representation:

(734) $be(this(book), \lambda x.similar(x, that(book)))$

The F-structure of the attributive version is also as expected:

(735) John drew a similar line to that line.



The meaning representation:

(736) $a(\lambda x.line(x) \wedge similar(x, that(line)), \lambda y.draw(john, y))$

At first it seems that the glue specification for these cases will be similar to those in sections 11.1 and 4.6.1, except that the adjective will also have an additional argument. However, just as with the issue discussed in section 11.3.4, the attributive version adds a presupposition:

(737) a. John read a similar book to "War and Peace".

b. ? John read a similar book to Mary.

In these sentences, the argument of *similar* is predicated to be a book as well. So the glue specification for the attributive version needs to follow the solution in section 11.3.4. If (737)b is at all possible, the explanation for it is probably metonymy, i.e. using one object to stand for another. In this case, "Mary" stands for the book that Mary read.

The predicative version of these adjectives can interact with a reciprocal, and (738)b is the reciprocal completion of (738)a.

(738) a. John is similar to Mary.
b. John and Mary are similar to each other.

The meaning representation for (738)b is:

(739) $\textsc{recip}(john \oplus mary, \lambda x \lambda y.be(x, \lambda z.similar(z, y)))$

This is obtained as usual by the glue specification for the reciprocal. Using (723), we get this representation also for:

(740) John and Mary are similar.

14.4 *Same* and *Different*

14.4.1 Basics

The meaning of *same* and *different* typically relies on some salient dimension f in the context of utterance, as indicated in (741)-(742). This contextually salient feature may be the item's identity, its type, and so on (Nunberg, 1984). I will mostly ignore this feature in the discussion below because I am only interested here in the syntax-semantics specification.

(741) $same[f](x, y) \equiv f(x) = f(y)$
$different[f](x, y) \equiv f(x) \neq f(y)$
for some contextually salient feature f

(742) a. John is different from Bill.
b. $different[f](john, bill)$
c. E.g. $f = \lambda x.height\text{-}of(x)$

Clausal Complement

Everything that was said in section 14.3 is true also about *different*. The adjective *same* behaves similarly:

(743) a. John is the same as Mary (in that/some respect).
b. John read a book that was the same as the book Mary read.
c. John read the same book as the book Mary read.
d. John and Mary are the same.
e. John and Mary are the same as each other.

This adjective must appear with the determiner *the*. The reason is probably that at least in some sense, an element x has exactly one y such that $same(x, y)$.

The unique thing about *different* and *same*, is that they also accept clausal complements, in contrast to the other symmetric adjectives above:

(744) a. * This line is perpendicular to/than that line is.
b. John is different than Mary used to be.
c. John is the same as Mary used to be.

(745) a. * John drew a perpendicular line to/than Mary did/drew.
b. John read a different book than Mary read.
c. John read the same book as Mary read.

The sentences (745)b,c mean:

(746) a. John read a book that was different from [*the book / some book / all the books*] that Mary read.
b. John read a/the book that was the same as *a/the book* that Mary read.

The representations are:

(747) $a(\lambda x.book(x) \wedge different2(x, \lambda y.book(y) \wedge read(mary, y)),\ \lambda z.read(john, x))$
$the(\lambda x.book(x) \wedge same2(x, \lambda y.book(y) \wedge read(mary, y)),\ \lambda z.read(john, x))$

The meaning of the operators is:

(748) $different2(x, P) \equiv different(x, \iota z.P(z))$
$same2(x, P) \equiv same(x, \iota z.P(z))$

Notice that the meaning of the noun has to be "duplicated" to be used in the second argument of these adjectives. This is exactly what happened with the comparative adjectives in section 11.3.3, and the solution here would be similar to the glue specification for *deg-compar7*. Here is part of the F-structure for (745)b.

(749)
$$\begin{bmatrix} \text{PRED} & \text{BOOK} \\ \text{SPEC} & [\text{DET} \ [\text{PRED} \ \text{`A'}]] \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{DIFFERENT}\langle \boxed{1} \rangle \\ \text{ATYPE} & \text{ATTRIBUTIVE} \\ \text{OBL-COMPAR} & \boxed{1} \begin{bmatrix} \text{PRED} & \text{THAN}\langle \boxed{2} \rangle \\ \text{OBJ} & \boxed{2} \begin{bmatrix} \text{PRED} & \text{READ}\langle \boxed{6}, \boxed{5} \rangle \\ \text{SUBJ} & \boxed{6}[\text{PRED} \ \ \text{MARY}] \\ \text{OBJ} & \boxed{5}[\text{PRED} \ \ \text{PRO}] \\ \text{REL-PRO} & \boxed{5} \end{bmatrix} \end{bmatrix} \end{bmatrix} \right\} \end{bmatrix}$$

Interaction with Ellipsis

Just as in (596)c-d, the sentences (745)b-c can also interact with independently-motivated ellipsis to get:

(750) a. John read a different book than Mary did.
b. John read the same book as Mary did.

(751) a. John read a different book than Mary.
b. John read the same book as Mary.

The challenge with (751)b is that this sentence also has another structural analysis, where the book is directly compared to Mary, as was discussed for (737)b. This analysis is wrong, since Mary is not a book (unless we consider metonymy).

The two analyses for *same* explain the confusion between the uses of *than* and *from* with *different*. People disagree whether *than* or *from* should be used in sentences such as the following:

(752) a. This book is different from that book.
b. This book is different than that book.

My analysis here shows that both are in fact possible. The first sentence either directly compares the two books, or through metonymy (as in (737)b), while the second sentence is an elided form of:

(753) This book is different than that book is.

which is rarely used. However, the close similarity in surface forms confuses people and makes some of them accept only one of (752)a,b and reject the other, but still intend just the simple direct comparison. Therefore, we need to inform the computer that in some texts, *than* can be used synonymously with *from* after *different*, namely (752)b is used when in fact (752)a is intended, and so the more complex ellipsis analysis is unnecessary / not intended.

In sections 10.8.2 and 11.3.7, I discussed the problems with proposals that attempt to analyze numerical and gradable comparatives directly without relying on ellipsis at all. Attempts were also made to analyze the adjective *different* without resorting to ellipsis, but these suffer from the same problems. For example, Beck (2000) analyzes sentence (754)a as (754)b.

(754) a. Luise owns a different car than Otto.

b. $\mathit{anders}(\mathit{Luise}, \mathit{Otto}, \lambda z \lambda v.[\mathit{car}(v) \wedge \mathit{own}(z, v)])$
where $\mathit{anders}(x, y, R) \equiv \exists u.[Rxu \wedge \mathit{diff}(u, \iota v.Ryv)]$

The sentence (754)a has the same structure as (751)a. The analysis (754)b is not compositional because it derives the relation $\lambda z \lambda v.[\mathit{car}(v) \wedge \mathit{owns}(z, v)]$ by conjoining the meanings of the noun property $\lambda v.\mathit{car}(v)$ and the main predicate $\lambda z \lambda v.\mathit{own}(z, v)$ directly. This move is problematic because it introduces a conjunction that cannot be traced back to an expression in the sentence, and does not yield an appropriate NP denotation. Moreover, (754)b suffers from the same problems that I pointed out regarding Heim's analysis in section 11.3.7. It cannot be uniformly extended to sentences where *than* takes a clausal complement, because the predicate in the *than* clause may be different from that of the main clause, as in (755).

(755) Luise owns a different car than Otto sold.

$a(\lambda x.\mathit{car}(x) \wedge \mathit{different2}(x, \lambda y.\mathit{car}(y) \wedge \mathit{sold}(\mathit{otto}, y)),\ \lambda z.\mathit{own}(\mathit{luise}, x))$

14.4.2 Overt Reciprocal

Now back to reciprocals. (756)a,b are the reciprocal completions of (751)b,a:

(756) a. John and Mary read the same book as each other.

b. John and Mary read a different book than each other.

Are these sentences acceptable? (756)a is very natural in a context where one does not want to confuse the book that John and Mary read with some contextually salient book. If (757)b appears immediately after (757)a then the most likely reading is that John and Mary read "War and Peace" too. If one simply wants to say they read the same book as each other, but not necessarily "War and Peace", one can use (756)a.

(757) a. Bill and Sue read "War and Peace".

b. John and Mary read the same book.

What about (756)b? Shouldn't one use the following instead?

(758) John and Mary read different books than/from each other?

I will discuss plural *different* in section 14.4.4. There are however natural examples of sentences with the same structure as (756)b. One was shown in (721)b, and here are a few more:

(759) a. ... as long as the road is slick enough for the front wheels to turn at a different rate than each other around turns.[3]

b. Black folks and white folks ... are raised in a different culture than each other, even within one country.[4]

c. If you look at the ends of the cable, you'll see different colored wires inside. If they are in a different order than each other, you're probably looking at a crossover cable.[5]

d. Where the child's parents have a different surname from each other, great care should be taken when deciding the surname by which their child will be known.[6]

The reason people use such forms seems to be to emphasize that each element of the subject is related to just one element of the NP which has *different* in it. For example, using "different rates" in (759)a might confuse the reader regarding how many rates each of the wheels turns at.

Now, consider the following sentences which are a variation on (750):

(760) a. John read the same book as Mary wrote.

b. John read a different book than Mary wrote.

[3] http://www.ford-trucks.com/forums/archive/topic/203644.html
[4] http://www.dccycles.com/arch/01/03/mar00325
[5] http://forums.megatokyo.com/index.php?s=763b9b8a37730564550bfbf71e17b0bb&showtopic=1684330
[6] http://www.nottinghamshire.gov.uk/home/youandyourcommunity/registrars/registrars-births.htm

It is semantically perfectly reasonable to apply reciprocal completion on these sentences. For example, the reciprocal completion of (760)b would say that John read a different book than Mary wrote and Mary read a different book than John wrote. However, applying reciprocal completion on (760) yields ungrammatical sentences because *each other* stands as the subject of a clause:

(761) a. * John and Mary read the same book as each other wrote.
b. * John and Mary read a different book than each other wrote.

The tricky issue here is that if "as/than Mary" in (751) is a clausal and not a phrasal complement, and it is obtained by stripping from (750), then (756) should be also be obtained by stripping from (761)a,b, even though the latter are ungrammatical.

The same issue comes up with reflexives:

(762) a. John cannot possibly own a different computer than David does.
b. * John$_i$ cannot possibly own a different computer than himself$_i$ does.
c. John$_i$ cannot possibly own a different computer than himself$_i$.

There are two possible explanations. One explanation is that (756) is possible because it involves metonymy, i.e. it is the reciprocal completion of (751)a,b when those sentences are analyzed using the metonymy explanation rather than the stripping explanation. A second explanation is that even though there is an underlying clausal complement for *different*, after stripping we get a NP fragment which is considered part of the larger clause it appears in. Hence, this NP could be a reflexive or a reciprocal that is anaphoric to another NP in that clause. Historically, this exception might have developed because of the similarity of "as/than Mary" in (751) to a phrasal complement. In either explanation, some process is needed beyond the straightforward rules of compositional semantics.

According to the stripping explanation, the way to analyze (756) is to assume that the stripping-recovery mechanism takes care to turn (756) into:

(763) a. John and Mary read the same book as each other did.
b. John and Mary read a different book than each other did.

and then the VP-ellipsis-recovery mechanism turns that into:

(764) a. John and Mary read the same book as each other read.
b. John and Mary read a different book than each other read.

Although both are ungrammatical, they are acceptable semantically. The derivation for these proceeds in the normal way, using the usual specification for the reciprocal from the previous chapter, and the specification for *same2* and *different2* which accept a clausal complement. The result is:

(765) $\textsc{recip}(jm,\ \lambda u\lambda v.a(\lambda x.book(x) \wedge \mathit{different2}(x, \lambda z.book(z) \wedge read(v,z)),\ \lambda x.read(u,x)))$

The meaning is a bit redundant since if John read a book which was different from (some/all of) the book(s) that Mary read, then Mary read a book which was different from (some/all of) the book(s) that John read. This is because *different* is symmetric. If we replace the subject with "John, Bill, and Mary", then this becomes non-redundant, since one needs to assert pairwise distinctness.

The advantage of the ellipsis explanation over the metonymy explanation is that it gives more specific details about how the semantic representation should be obtained from the sentences.

14.4.3 Covert Reciprocal

My Proposal

Consider the following:

(766) a. John and Mary read the same book.
b. John and Mary read a different book.

Each of these has two readings. In one reading, "the same book" and "a different book" are anaphoric to some salient book(s) in the discourse, as in:

(767) a. Bill read "I, Robot". John and Mary read a different book than that.
b. Bill read "I, Robot". John and Mary read the same book as that.

Here *that* is anaphoric to (the book) "I, robot", and both *same* and *different* behave very simply. According to (723), this first reading is obtained by dropping the complement "as/than that" since it is clear from the context.

The second reading of (766) is reciprocal, meaning the same as (756). As with (756)b, the reciprocal reading of (766)b is more often expressed using plural *different*. However, people do occasionally use this construction, as (768) shows, for the same reason: to

emphasize that each element of the subject is related to just one element of the NP. For example, using "different books" in (766)b and (768)b might confuse the hearer regarding how many books each of the two members of the subject actually read.

(768) a. Men and women have a different sense of humour! (= (721)b)

b. [After a disagreement about what is written in some book by a certain title:] Clearly, you and I read a different book by that title.[7]

How should this reading of (766) be obtained? According to my theory (723), it is obtained by dropping the complement "as/than each other" from (756), just as the anaphoric reading is obtained by dropping the complement "as/than that" from (767).

Obstacle 1: Ungrammatical Reconstruction

In this and the next sub-sections I look at potential obstacles to my theory, and answer them.

Sentence (769)b is modelled after (756)a. The story in (769)a provides a plausible context, explaining why "as each other" is used in (769)b, to prevent the hearer from assuming that the two buyers were the same.

(769) a. We were surprised to find out that pictures 1 and 2 were bought by the same person, but it became really bizarre when we discovered that ...

b. [pictures 3 and 4]$_i$ were also bought by the same person as [each other]$_i$.

But now consider (770)a, which is the covert reciprocal version of (769)b. If we change it to active voice in (770)b we get a fine sentence that means the same as (770)a.

(770) a. Pictures 3 and 4 were bought by the same person.

b. The same person bought pictures 3 and 4.

c. * The same person as [each other]$_i$ bought [pictures 3 and 4]$_i$.

According to my account, (770)a is obtained from (769)b by omitting "as each other", and so (770)b should also be obtainable from (770)c in the same way. But the latter is ungrammatical. Is this a problem for my account?

[7] http://www.spinnoff.com/bb/viewtopic.php?p=31228& 1-mar-2005

I do not think this is a problem. Sentence (770)c makes sense semantically, just as (770)b does, and it is just a fact about Syntax that does not allow such sentences. The adjective *same* can still take an *as* argument:

(771) The same person as *that one* bought pictures 3 and 4.

where *that one* refers to some salient person. My conclusion is that the reconstruction mechanism here may create structures that are coherent semantically even if not syntactically. We have already seen a similar situation with (756) and (764). Although (764) is ungrammatical due to the reciprocal being in subject position of a clause, (756) can still be derived from it. What makes this possible is the similarity of (756) to the phrasal case. Similarly here: (770)c is ungrammatical, but (770)b is still derived from it and is possible thanks to its similarity to (770)a.

Obstacle 2: "Pure" Truth Conditions vs. Conventional Use

Consider the following:

(772) a. John, Bill, and Mary read the same book as each other.

b. John, Bill, and Mary read the same book.

The likely reading of (772)b seems to be that there is just one book that John, Bill, and Mary read. But according to my account, the meaning of (772)a is: for each pair of people from the group, the two people read the same book, though there need not be one book that all three people read. Therefore, is it not wrong to say that (772)b is derived from (772)a by omitting "as each other"?

I do not think so. First, note that if the subject has just two members, or if the situation is such that each person reads just one book, then the truth conditions of the two accounts come out the same. It is not unreasonable that the situation is implicitly restricted in this way. Natural language is very flexible in terms of zoning in on one situation that is a subset of the larger ongoing discourse, and doing so implicitly (as mentioned in section 1.2.3). Figuring out this situation is not part of the "pure" truth conditions but is a contextual/pragmatic issue.

Further evidence is given by considering the following sentences:

(773) a. John, Bill, and Mary have *all* read the same book as each other.

b. John, Bill, and Mary have *all* read the same book.

These sentences seem to emphasize that there is indeed just one book that all three people read. This suggests that (772) does not force one book, or else there would be no difference in meaning between (773) and (772). If this is true, it does come out that (772)b means the same as (772)a and is derived from it.

If we still want the computer to calculate truth conditions that force just one book in (772)b, we would need to do so by a mechanism beyond pure compositional semantics. The justification is that this is an implementation of a case of (pragmatic / contextual) "language use", while keeping the machinery of compositional semantics simple. Other instances of such "language use" mechanisms that go beyond pure compositional semantics are hyperbole and metonymy (see section 1.2.3).

The Price of a Compositional Analysis

To strengthen this point, let us see what would be the price we would need to pay in order to develop a purely compositional account of (766), which assumes that nothing has been deleted, and the words in the sentence must contribute all the semantic material.

The goal is to get (765), repeated here as (774)a, but with having neither the reciprocal nor the second occurrence of *read* overtly. So *different* should be a complex operator that takes much of the other parts of the sentence as arguments and creates the desired result from them. We get this operator by taking (774)a and abstracting all the parameters:

(774) a. $\textsc{recip}(jm,\ \lambda u\lambda v.a(\lambda x.book(x) \wedge \textit{different2}(x, \lambda y.book(y) \wedge read(v,y)),\ \lambda x.read(u,x)))$

b. $\lambda Q\lambda P\lambda z\lambda R.\textsc{recip}(z,\ \lambda u\lambda v.Q(\lambda x.P(x) \wedge \textit{different2}(x, \lambda y.P(y) \wedge R(v,y)),\ \lambda x.R(u,x))) :$
$((l_v{}^e \to l_r{}^e) \to (l^e \to H^t) \to H^t) \to (l_v{}^e \to l_r{}^e) \to a^e \to (a^e \to l^e \to H^t) \to H^t$
where l is the label of my NP, a is the label of my antecedent, and H is the label of the clause I scope over.

Why does a strictly compositional account require such a complex operator? This is because *different* needs to behave both as a local comparator and as a reciprocal. This operator needs to take the noun's meaning P as argument, as discussed in the paragraph just above (749), because that meaning is used twice. For the same reason, it needs to take the main binary relation R as argument because that meaning is used twice (the first occurrence is what would be reconstructed by the VP-ellipsis mechanisms under my account). Another way of seeing this is that *different* needs to contribute the reciprocation (because we are assuming it is not reconstructed by (723)), and so as a reciprocal, it needs

to take both a^e and $a^e \to l^e \to H^t$ as arguments. Finally, because the quantifier 'a' appears inside (774)a, the normal contribution of 'a' cannot be left untouched, and it must also be taken as argument by the operator.

The reciprocation is needed here for generality: In sentences such as (768), if the subject has more than two elements, then each pair of them is asserted to be related to a different object. However, in a strictly compositional account for *same*, we can use a slightly simpler operator, because of the simpler truth conditions:

(775) a. John, Bill, and Mary read the same book.

b. $a(\lambda x.book(x), \lambda x.read(j \oplus b \oplus m, x))$

c. $\lambda Q \lambda P \lambda S.a(P, S)$:

$((l_v{}^e \to l_r{}^e) \to l^e) \to (l_v{}^e \to l_r{}^e) \to (l^e \to H^t) \to H^t$

where l is the label of my NP, and I must scope over my antecedent.

The difference here is that "the same" is treated simply as 'a' instead of contributing a RECIP . Notice that Q is a vacuous argument: it is intended to absorb the contribution of *the* and do nothing with it.[8] The stated scoping restriction guarantees that "the same book" will scope over the subject. This contrasts with:

(776) John, Bill, and Mary read a book.

where the 'a'-narrow-scope reading says that each of the three people read a book, but not necessarily the same one.

What is the price of this direct analysis? First, the operator in (774) is very complex. Second, both operators are not cleanly compositional because they need to take their NP's quantifier as an argument, and in the case of *same*, ignore it. Third, the two operators are quite different from each other even though *same* and *different* are very similar, and these operators also do not generalize to cases with an overt reciprocal as in (756). I think that this solution is not simpler than my theory (723) which simply reconstructs "as/than each other" when *same/different* does not have a complement.

[8]Barker (2004b) also proposes a strictly compositional analysis of *same* for the reciprocal reading. His proposal is much more complicated and relies on higher-order types and choice functions because he does not want *same* to take *the* as a vacuous argument. There is no advantage to his analysis in terms of truth conditions, and there is no advantage to keeping *the* as it is because the "direct" specification for *same* is so ad-hoc anyway, as discussed below.

E-Type Anaphora

Consider the following sentence from Figure 1.1:

(777) If sculptures E and F are exhibited in the same room, no other sculpture may be exhibited in that room.

The expression *that room* is anaphoric to *the same room*. But according to my theory, the antecedent of the conditional really has a covert reciprocal quantification there. Strictly speaking, there is no one room in the antecedent, so how can *that room* refer to the one room in which sculptures E and F are exhibited?

One solution is to use the tool of "accommodation" in DRT (van der Sandt, 1992; Blackburn and Bos, 2005b). According to this, the expression *the same room* can be resolved to a discourse variable introduced previously in the discourse. For example, in (778), it is resolved to the discourse variable introduced by *room 1*.

(778) Sculpture E is exhibited in room 1. Sculpture F is exhibited in the same room.

Sometimes, a definite expression does not have an explicit antecedent in the discourse. For example, *her husband* in (779)a refers to Jody's husband and assumes she has one, even though this may not have been mentioned explicitly in the discourse. According to the theory of accommodation, the definite expression needs to be accommodated in such a case, and what it refers to needs to be explicitly introduced. A new discourse variable representing Jody's husband is added so that the expression *her husband* can be resolved by being equated with that variable.

(779) a. Jody loves her husband.
b. (means roughly:) Jody loves the husband of Jody.

If we apply this idea to (777), we might say that *the same room* does not have an explicit antecedent in the text, and so the expression should be accommodated to introduce such an entity. In particular, a discourse variable is introduced. Then, it is an easy matter to have the anaphoric expression *that room* in (777) be resolved to that discourse variable.

However, this solution does not generalize to the case when *each other* is said explicitly as in (780)a because now we have a reciprocal quantification. It also does not generalize to (780)b for the same reason.

(780) a. If sculptures E and F are exhibited in the same room as each other, no other sculpture may be exhibited in that room.

b. If sculptures E and F are exhibited in different rooms (than each other), no other sculptures may be exhibited in those rooms.

We really do have a quantification here. According to my theory, *that room* in (777) and (780)a, and *those rooms* in (780)b, are cases of E-type anaphora (see section 12.3.2). These anaphoric expressions need to take the material in their antecedents and construct a reference to the semantic entity they need, but the solution is not simply picking up on an existing discourse variable introduced by the antecedent.

Nevertheless, if someone is interested in "rounding corners" as a practical matter to simplify things just for (777) but not for the other cases, one can use the solution in (775). The quantifier 'a' there would introduce a discourse variable which could be picked up by *that room* in (777).

14.4.4 Plural *Different*

I have not investigated above cases where *different* modifies a plural noun, as in:

(781) a. John and Mary read different books from each other.

b. John and Mary read different books than each other. (cf. (756)b)

c. John and Mary read different books. (cf. (766)b)

d. Different people bought pictures 3 and 4. (cf. (770)b)

These cases have an additional complexity in comparison to the cases using *same* and *a different*, namely that here *each other* has two possible antecedents: *books* and *people*. What does (781)a mean when *each other* is anaphoric to *books*? It means the same as *various*, i.e. each book is different from the other books with respect to some contextually salient feature. This is also possible when "from/than each other" is not mentioned, as in (781)c,d.

I deliberately started this chapter by considering singular rather than plural *different*. If I had started with plural *different*, we might have gotten confused about what the overt or covert reciprocal is anaphoric to and say the following: Since the internal-reciprocal reading is available, where *each other* is anaphoric to the plural noun preceding it, perhaps this is the *only* reading we have, and the contextually salient feature takes care of the

rest. For example, in (781)b,c, my theory from previous sections predicts that *each other* could be anaphoric to "John and Mary", but the contextual theory could propose instead that only the internal-reciprocal reading is available, and the relevant contextual feature is "according to who read them". In other words, the contextual theory says that (781)b,c mean: John and Mary read books, and those books were different from each other with respect to who read them.

This is in fact what Beck (2000) proposes. However, she considers only cases where *different* modifies a plural noun, and so her analysis does not cover cases like (768), and it could not be extended to cover such cases because the singular noun does not denote a non-singular plurality that could be divided into contextually-salient sets. She also does not consider cases where *each other* appears overtly, as in (759), and so her analysis does not explain the proper interaction between *different* and *each other*.

Since *each other* must be anaphoric to the subject in (782)a, it is reasonable to conclude that this is also possible when *different* is followed by a plural noun, as in (782)b.

(782) a. The four wheels turn at a different speed (from/than each other). (cf. (759)a)
b. The four wheels turn at different speeds (from/than each other).

In fact, even for plural *different*, there are cases where the overt reciprocal cannot possibly be anaphoric to the noun and must be anaphoric to the subject. This happens in languages with a richer gender morphology than English.

For example, in Hebrew, if the feminine-gender noun *speed* appears in singular, as in (783), the reciprocal cannot be anaphoric to it, and so only the masculine and not the feminine version of the reciprocal is possible.

(783)

arba'at	ha-galgalim$_i$		mistovevim		bi-mhirut		shona	
four.*def*	wheels.*def.masc*		turn.*pl*		in.speed.*sg.fem*		different.*sg.fem*	
[ze	mi-	ze]$_i$	/	*[zo	mi-	zo]$_i$		
this.*masc*	from	this.*masc*		this.*fem*	from	this.*fem*		

"The four wheels$_i$ turn at a different speed than each other$_i$."

In contrast, if *speed* is plural as in (784), then both forms of the reciprocal are possible, but each version can be anaphoric to either the subject or the plural noun. The masculine reciprocal must be anaphoric to the subject, and the meaning is: each wheel turns at a different speed than each of the other wheels. The feminine reciprocal must be anaphoric

to the plural noun, and the meaning is: the wheels turn at speeds which are different from each other (with respect to some salient property).

(784) arba'at ha-galgalim$_i$ mistovevim bi-mhiruyot$_j$ shonot
four.*def* wheels.*def.masc* turn.*pl* in.speed.*pl.fem* different.*pl.fem*

[ze mi- ze]$_{i/*j}$ / [zo mi- zo]$_{*i/j}$
this.*masc* from this.*masc* this.*fem* from this.*fem*

"The four wheels$_i$ turn at different speeds$_j$ than each other$_{i/j}$."

A similar situation can be found in Spanish, where the reciprocal must bear the same agreement features as its antecedent. Observe that the masculine reciprocal in (785) must be anaphoric to the masculine subject. Thus (785) means that each living being performs actions that are different from those performed by other living beings. In contrast, sentence (786) means that living beings perform various actions.

(785) [Los seres vivos]$_i$ generamos acciones$_j$ diferentes
the.*pl.masc.* beings living.*pl.masc.* perform actions.*pl.fem.* different

[los unos de los otros]$_{i/*j}$.
the.*pl.masc.* ones.*pl.masc.* from the.*pl.masc.* others.*pl.masc.*

"Living beings$_i$ perform different actions than each other$_i$."

(786) [Los seres vivos]$_i$ generamos acciones$_j$ diferentes
the.*pl.masc.* beings living.*pl.masc.* perform actions.*pl.fem.* different

[las unas de las otras]$_{*i/j}$. [9]
the.*pl.fem.* ones.*pl.fem.* from the.*pl.fem.* others.*pl.fem.*

"Living beings perform different actions$_j$ from each other$_i$."

My conclusion is that when *different* modifies a plural noun, whether or not there is an overt reciprocal, then *both* the internal-reciprocal and the external-reciprocal readings are available.

To sum up, Beck's theory does not account for singular *different* and for overt *each other*, and even for plural *different* with covert *each other*, her explanation does not provide a systematic syntax-semantics specification that explains how the subject NP produces a salient cover which supports the NP-dependent reading. Here theory also does not generalize to some other languages with gender markings. In contrast, my theory does

[9] http://www.arnoldoaguila.com/finalidad.html.

not rely on pragmatic cover sets, and relies only on a compositional calculation of the semantics, after an appropriate reconstruction of the convert reciprocal. It also accounts correctly for all the cases, including gender agreement in other languages.

Part IV

Application and Conclusion

Chapter 15

Application

In section 1.3.1 I mentioned four parts of an exact NLU application. So far in the dissertation, I addressed the task-independent linguistic analysis and related ambiguity management. In this chapter, I consider the other parts involved in an application. I mention here only briefly some of the main issues that need to be addressed. Let us focus, as an example, on an application for solving logic puzzles given their textual descriptions (see section 1.1.2). Motivation for working on this application is given in (Lev, 2006), and a description of preliminary work on an implementation can be found in (Lev et al., 2004).

15.1 The Back-End: Puzzle Solver

Let us first consider a simple system where the puzzle is already given in a formalized form. The formalization consists of a finite set Γ of FOL sentences that encode the main constraints of the puzzle, and a set of queries. Each query Q consists of an additional assumption ψ, a set of five answer choices $\phi_1, \ldots, \phi_5$, and an instruction. There are four possible instructions:

Instruction	Find the unique $i \in [1..5]$ such that:
must-be-true	$\Gamma, \psi \models \phi_i$
must-be-false	$\Gamma, \psi \models \neg\phi_i$
may-be-true	$\Gamma \cup \{\psi, \phi_i\}$ is consistent
may-be-false	$\Gamma \cup \{\psi, \neg\phi_i\}$ is consistent

For example, in the formalization of Figure 1.1, Γ will include, among other statements,

a FOL encoding of the third constraint there, shown in (2) and repeated here:

(787) $\forall x.[(room(x) \wedge \textit{exhibited-in}(E,x) \wedge \textit{exhibited-in}(F,x)) \rightarrow$
$\neg\exists y.sculpture(y) \wedge y \neq E \wedge y \neq F \wedge \textit{exhibited-in}(y,x)]$

In question 1 of Figure 1.1, we have:

(788) $\psi =$ $\textit{exhibited-in}(D, room3) \wedge \textit{exhibited-in}(E, room1) \wedge \textit{exhibited-in}(F, room1)$
$\phi_1 =$ $\textit{exhibited-in}(C, room1)$
instruction = may-be-true

In question 2, we have:

(789) $\psi =$ $\textit{exhibited-in}(G, room1)$
$\phi_2 =$ $\forall x.(x = E \vee x = F) \leftrightarrow (sculpture(x) \wedge \textit{exhibited-in}(x, room2))$
instruction = must-be-false (= may not be true)

Because the puzzles always talk about a finite domain, calculating the answer to a puzzle is decidable. One way a computer could find out whether or not $\Gamma, \psi \vDash \phi_i$ is to feed this query to a theorem prover (TP), and in parallel feed the query $\Gamma \cup \{\psi, \neg\phi_i\}$ to a model builder (MB) to check whether this set is consistent. If the instruction is must-be-true and the correct answer-choice is i, the TP will say *Yes* (= follows) for this i. Furthermore, the MB will say *Yes* (= consistent) for the other four i's. This is because the puzzles talk about a finite domain and have a closed world assumption, and therefore, if φ does not follow from ψ then there is always a finite countermodel. For $j \neq i$, the TP may not discover that $\Gamma, \psi \nvDash \phi_j$ i.e. it may not terminate because FOL is only semi-decidable. However, because we know the upper limit on the size of relevant models in the domain (e.g. in Figure 1.1 the size is 9 if you count just sculptures and rooms, or 10 if you count the art gallery as well), once the MB exhausted all possible models up to that size, it can terminate and say *No* (= the set is inconsistent).

Another way a computer could answer the queries is to first translate them from FOL to finite Constraint Satisfaction Problems (CSPs), where a finite number of variables are each assigned a value from a finite number of possible values, subject to certain constraints. For example, the puzzle from Figure 1.1 can be recast as a CSP, where the variables represent the sculptures, and the values represent the room number that each sculpture is exhibited in. Solving such CSPs using specialized algorithms may be more efficient than answering

the FOL queries using general-purpose TP/MBs. However, figuring out how to translate a FOL query from a logic puzzle to an equivalent finite CSP in a general way may not be easy. In particular, some of the constraints can be numeric, such as: the number of variables that are assigned the value '1' is at most 3 (this comes from the second half of constraint (4) in Figure 1.1).

An input is considered erroneous if it does not yield exactly one correct answer-choice for each question. This can occur when the puzzle has an error, but more often it occurs when a human tries to write a FOL formalization of the puzzle and forgets to state some of the information that is implicit in the text (e.g. the closed world assumption, or that a sculpture may not be exhibited in more than one room). The assumption about multiple-choice questions is important as it can be utilized to help disambiguate the NL input.

15.2 Additional Stages

Now that we have seen the back-end that we need to work towards, let us start from the beginning and see what else is needed. A general linguistic analysis of the puzzle text is insufficient, by itself, for generating appropriate input to the back-end puzzle solver. Several additional computations are needed, and most of them depend in some ways on the specific context of the puzzle solving application.

15.2.1 Preprocessing

A system for solving logic puzzles from their textual descriptions does not need to submit the input text to a complete linguistic analysis.

We would run into unnecessary complications if we simply fed the text to a parser – it would get confused by elements idiosyncratic to puzzle texts such as numbering of constraints, questions, and answer choices. Also, the preface often ends with a sentence such as "The ... must conform to the following conditions", which adds no information to the puzzle formalization and can be ignored.

It is simpler to have a task-specific preprocessor that identifies the different regions of the text – the preamble which consists of a preface and a list of constraints, and then the list of queries (question + answer choices) – and simplifies the input, so that pure sentences or noun-phrases could then be fed to the task-independent linguistic component.

In the pipeline of system stages, preprocessing is the first stage. Then comes the task-independent, general linguistic analysis of the text, which was the focus of this dissertation. Subsequent stages are discussed below.

15.2.2 Adding Lexical Knowledge

One kind of lexical knowledge needed for the constructions we covered is stating the connection between opposite comparative words. For example, the computer needs to know that if x is earlier than y then y is later than x. Other such pairs are *taller*/*shorter*, *heavier*/*lighter*, *before*/*after*, etc.

Another kind of knowledge pertains to the truth conditions that words contribute to the puzzle. Consider question 2 in Figure 1.1. A general linguistic analysis might produce a representation such as (790)b for the sentence (790)a.

(790) a. 'Sculptures E and H' is a complete list of the sculptures exhibited in room 2.

b. $is(\{E, H\}, \lambda x.complete(x) \wedge list(x) \wedge$
$of(x, the(\lambda y.sculpture(y) \wedge exhibited\text{-}in(y, room2))))$

For the purpose of reasoning, it is insufficient to simply say about x that it is *complete* and a *list*. To obtain the final FOL representation in (789), the computer needs the following definition of "complete list":

(791) $is(g, \lambda x.complete(x) \wedge list(x) \wedge of(x, the(P))) \rightarrow \forall y.[y \in g \leftrightarrow P(y)]$

This axiom, together with additional axioms about $\in$, allow an automated reasoner to prove formula ϕ_2 in (789) from (790)b.

Although there are quite a few such expressions in logic puzzle texts, they recur across puzzle instances. For example, many puzzles talk about sets and membership using the expressions *group*, *committee*, *member of*, *in*, *contains*, etc., or about relations such as adjacency. Hence it might be possible to develop a relatively compact library of axioms which explicate the relations between the meanings of these words.

15.2.3 Filling Knowledge Gaps

The puzzle in Figure 1.1 does not state explicitly that no sculpture may be exhibited in more than one room. Without this piece of knowledge, the explicit conditions in the text are insufficient to yield exactly one answer for each multiple choice question (for question

1, answers B, D and E would all be correct). The text does not specify the condition explicitly because human readers know this piece of world knowledge, but it has to be given to the puzzle solver in explicit form.

In general, representing and acquiring such world knowledge is a very difficult AI problem. However, in the case of GRE/LSAT logic puzzles, this kind of knowledge can often be recast as mathematical properties of the relations mentioned in the puzzle. For instance, the unique location constraint on sculptures is equivalent to constraining the mapping from sculptures to rooms to be injective (one-to-one); other possible properties are surjective and total. The computer could systematically search for such implicit constraints that, in combination with the explicitly stated constraints, yield exactly one answer per question.

15.2.4 Task-Specific Assumptions

Further assumptions about the logic puzzle task need to be encoded and given explicitly to the puzzle solver.

One such assumption is the Unique Names Assumption. Each name mentioned in the text refers to a different entity (e.g. C through H in Figure 1.1). This should be formalized using inequalities between object constants in the logical formulas. For example, we need to state $E \neq F$ in addition to the formula in (787). Without stating this explicitly, the reasoners may interpret the two constants as referring to the same entity, and this could lead to completely wrong results.

Another assumption is the Domain Closure Assumption. The objects that need to be considered for solving a puzzle are only those mentioned explicitly in the text. For example, suppose a puzzle starts with "Five films – A, B, C, D, E – are shown to a group of students" and this is followed by some constraints. The question "Can any film be shown after film C?" should be interpreted to mean "Can any film of the films A, B, C, D, and E be shown after film C?", otherwise the answer may be trivially yes.

I already mentioned above the assumption that each multiple-choice question has exactly one correct answer choice, and the other answer choices are incorrect.

15.3 Ambiguity Management

As mentioned at the end of section 1.3.3, ideally we would extend the idea of packing to the rest of the stages of the pipeline, including the back-end solver. However, at this time

it is unknown how to do this kind of extension.

Currently, the computer would have to unpack the representations in order to do automated reasoning with them. Before it does so, it may be able to use domain-specific heuristics to filter out some of the analyses that the general-purpose linguistic analyzer has produced, and to rank the others in the order in which it would be useful to explore them.

There are a few other things that can help. In line with the maxims of conversation (Grice, 1975; Blackburn and Bos, 2005b, ch.5), we can verify that the interpretation we select for each sentence is both consistent with the interpretations of its predecessors and informative in relation to them (i.e. it is not entailed by them). If this is not the case, this gives a very strong clue that the interpretation is incorrect and another should be selected. If all the interpretations of a sentence a ruled out in this way, this is a strong clue that one or more interpretations that we selected for previous sentences are wrong.

Also, questions usually have the form: "If X, then which of the following ..." We need to verify that the question's antecedent X is consistent with the preamble and is not entailed by it. In effect, we are temporarily adding another piece of information to the puzzle formulation. If these tests fail then either the question or part of the preamble was not interpreted correctly.

Another maxim dictates saying as much as required, and hence the sentence "No more than three sculptures may be exhibited in any room" from Figure 1.1 carries the implication that in some possible solution, three sculptures are indeed exhibited in the same room. Although these "scalar implicatures" are defeasible by further explicit information to the contrary, they may provide a useful "sanity check" that can help us reduce ambiguities.

Overall, the process of zoning in on a formalization of the puzzle that can be fed to the back-end puzzle solver needs to be iterative. An interpretation is selected for each sentence, and then the resulting representations are given to the solver together with background assumptions and domain knowledge. If the solver finds that each multiple-choice question has exactly one correct answer, it is highly likely we fed it the correct formalization, and we can accept the answer. Otherwise, the system needs to backtrack to try another formalization.

Chapter 16

Conclusion

Exact meaning representations are paramount in real-world, exact NLU applications but are also important for other NLP applications. However, there were several obstacles that led some researchers away from them. These include: the non-trivial mapping from syntactic to semantic structures due to the mismatch between them; the open problem of how to handle both syntactic and semantic ambiguities in a uniform way and how to overcome the problem that a locally less-likely analysis may be the globally correct one; and the lack of coverage of many semantic phenomena in existing systems, which results from the fact that some aspects of even literal meaning go beyond the explicit material given in a sentence. This dissertation made important steps towards overcoming these obstacles.

In chapter 2, I explained what exact meaning representations are and how to do basic inference with them. Because a sentence's meaning can be expressed by many logically equivalent formulas, the important point when designing a meaning representation language is to choose those logical forms whose shape is amenable to a compositional computation from the surface structure of sentences. The meaning representation language may thus not be the most suitable for performing inference, and so these representations should be translated to another implementation language tailored to efficient reasoning in a particular application. In later chapters, I extended the initial meaning representation language, and in some places I discussed the relevant implementation language as well.

In chapter 3, I explained the problems that traditional approaches to the syntax-semantics interface encountered. I showed how Glue Semantics is a more flexible framework

that does not suffer from these problems, and I provided a new introduction to that framework, which unlike previous introductions, did not rely on prior knowledge of LFG, nor heavily relied on linear logic.

Having these two frameworks in place, I proceeded to show in chapters 4-7 the details of how a computer can actually compute all meaning representations for a sentence, even when the sentence is syntactically ambiguous, and do so in an efficient packed manner. I showed the details of how to map F-structures to the pieces of meaning (Glue Semantics statements) and how to compute all their possible combinations. I then proceeded to solve the open problem of how to combine Glue Semantics with the framework of choice-space packing by providing a novel algorithm that packs multiple results arising from the syntactic and semantic ambiguities such that each shared piece is computed and represented just once. I proved the correctness of Hepple's basic (unpacked) algorithm, as well as of my packed algorithm. In a certain way, we got maximal packing (see section 7.8). I also extended the algorithm with some additional features that are needed in practice, including non-scoping modifiers and scoping constraints, and I discussed how to detect and eliminate logical equivalences. In sections 3.5, 7.7, and 13.6, I showed how my work provides a better solution than other approaches to our goals regarding the syntax-semantics interface.

In the rest of the dissertation, I showed how the combination of Glue Semantics with XLE's broad-coverage LFG grammar is flexible and powerful, by extending the semantic coverage to more challenging constructions. Thus, I showed in chapter 13 how the GS framework allows us to handle reciprocals naturally. For the other constructions, a main question was what should be the role of the compositional semantics component, and what should be handled by additional components.

In chapter 8, the conclusion was that anaphora resolution should be handled by a separate component rather than by "variable-free semantics"-style resolution. This means using something like DRSs. Then a new problem arose – using the DRS language directly would require revising all our previous work on glue specification. I solved this problem by showing how the previous specification can be maintained with very minor revisions, and the new anaphoric elements are added to it. This was achieved by providing another level of translation to the DRS level. This separation turned out to be useful once again when we considered E-type anaphora and the possibility of using Situation Semantics entities instead of DRSs (section 12.3.2). Thus, the theme of "separation of powers" was seen here once again: separation of anaphora resolution from the composition process; separation

of explicit representation of DRSs from the composition process; and separation of the meaning representation language from the inference/implementation language.

In chapters 10, 11, and 14, the conclusion was that certain cases of comparatives should be handled by various rules in the compositional specification, but that these rules should not do all the work. I provided details of a careful analysis of where the borderline lies between the compositional component and the independently-motivated component of ellipsis. By separating the computation into these two components, a more elegant solution emerged: the composition is not overly complicated and does not duplicate what the ellipsis component can do better. I also discussed cases of covert reciprocals in chapter 14 and showed that we can get a more elegant and consistent overall analysis by acknowledging the covert reciprocals than by trying to develop a strictly compositional solution.

Finally, in chapter 15, I discussed a few issues that need to be addressed when constructing a system for solving logic puzzles and that go beyond the general-purpose linguistic analysis discussed in the rest of the dissertation.

What are the consequences and gains of this work? A computational linguist who wants to build exact NLU applications can now use the work of this dissertation as a good starting point. He can find here a good coverage of many constructions, including basic ones such as quantifiers, VP-modifiers, and noun-modifiers, as well as more advanced ones such as anaphora, comparatives, overt and covert reciprocals, and *same/different*. Thanks to this existing work – having overcome some important hurdles in this enterprise and having learnt from these cases – and thanks to the powerful and flexible framework of glue semantics, it would now be more feasible than before for researchers to extend this coverage to additional complex constructions that appear in texts of exact NLU tasks. The next step would be to extend the coverage to additional anaphoric expressions including E-type anaphora, to coordination and lists, generic NPs, various other functional words such as *except* and *only*, and combining this work with resolution of anaphora and ellipsis.

This direction is very promising because the work here shows that one Ph.D. student can do quite a lot in a few years in terms of a systematic development of structural semantics knowledge; and because the size of that knowledge is relatively small, we can say that if several people continued with this kind of effort in a concentrated manner, we could get a reasonably good coverage of most of structural semantics. The obtained knowledge and tools could then be usable more-or-less directly in exact NLU applications, and would form a basis for further work in other NLP applications. This is reminiscent of

the efforts in syntax (in the XLE and the Lingo projects) to create hand-written, precise, broad-coverage grammars of natural language, which produce comparable results to purely statistical parsers in terms of coverage, but also provide higher accuracy (Kaplan et al., 2004a). This undermines the claim that it is too hard and complicated to make the computer calculate exact meaning representations efficiently for a broad enough part of NL. There are knowledge acquisition problems, to be sure, but they have been successfully tackled in syntax. Progress has also been made in this dissertation, and so the problem for structural semantics does not look as intractable as it used to.

Moreover, unlike existing frameworks for underspecified representations, the computational semanticist does not need to take any special steps to ensure that the syntax-semantics interface knows about syntactic and semantic ambiguities and computes compactly all the analyses. All this has been taken care of by the algorithms presented in this dissertation.

To take advantage of the packed computation presented here, two things can be done as the next step. First, the packed computation should be extended to the next stages of linguistic processing, taking care of anaphora and plurality ambiguities, as well as packed reasoning at the end stage (continuing such work that was started at PARC (Crouch, 2005)). Second, filtering out the least-likely analyses based on statistical methods while keeping the N-best options, which was done at the syntax stage (Riezler and Vasserman, 2004), should be extended to the semantic and later stages.

In terms of the goal of exact NLU itself, it would be interesting to continue working on a system for solving logic puzzles (Lev, 2006). The system described in (Lev et al., 2004) was an initial attempt based on techniques that existed at the time. The next step would be to rebuild the system using the work developed in this dissertation, replacing the statistical parser with the XLE parser, and replacing hole semantics with glue semantics, including the algorithms and knowledge developed here. Further work that is needed, beyond extending the semantic coverage, is developing the lexical and world knowledge that is needed in such puzzles, as explained in chapter 15. It might also be possible to make use of existing lexical semantics resources such as FrameNet, VerbNet, and PropBank.

Bibliography

Allen, James. 1995. *Natural language understanding*. Benjamin Cummings.

Alshawi, Hiyan, ed. 1992. *The core language engine*. MIT Press.

Andrew, Galen, and Bill MacCartney. 2004. Statistical resolution of scope ambiguity in natural language. Unpublished MS, Stanford University. http://nlp.stanford.edu/projects/nlkr/scoper.pdf.

Androutsopoulos, I., G.D. Ritchie, and P. Thanisch. 1995. Natural language interfaces to databases–an introduction. *Journal of Language Engineering* 1:29–81.

Androutsopoulos, Ion. 2002. *Exploring time, tense, and aspect in natural language database interfaces*. John Benjamins Publishing Company.

Asudeh, Ash. 2002. A resource-sensitive semantics for equi and raising. In *The construction of meaning*, ed. D. Beaver et al. CSLI Publications.

Asudeh, Ash. 2004. Resumption as resource management. Doctoral Dissertation, Stanford University.

Asudeh, Ash. 2005. Relational nouns, pronouns, and resumption. *Linguistics and Philosophy* 28:375–446.

Asudeh, Ash, and Richard Crouch. 2001. Glue semantics for HPSG. In *Proc. of the 8th intl. HPSG conference*, ed. F. van Eynde, L. Hellan, and D. Beermann.

Asudeh, Ash, and Richard Crouch. 2002. Derivational parallelism and ellipsis parallelism. In *Proc, of WCCFL 21*, ed. L. Mikkelsen and C. Potts.

Asudeh, Ash, Richard Crouch, and Mary Dalrymple. 2002. The syntax-semantics interface: Theory and implementation. NASSLLI'02 lecture handouts.

Austin, J. A. 1962. *How to do things with words*. Harvard University Press.

Barker, Chris. 2004a. Continuations in natural language. In *Proc. of the 4th ACM SIGPLAN Continuations Workshop (CW'04)*, ed. Hayo Thielecke.

Barker, Chris. 2004b. Parasitic scope. Submitted. Based on a 2004 SALT paper. http://ling.ucsd.edu/∼barker/Research/barker-same.pdf.

Barwise, Jon. 1987. Noun phrases, generalized quantifiers, and anaphora. In *Generalized quantifiers*, ed. Peter Gärdenfors. D. Reidel Publishing.

Barwise, Jon, and Robin Cooper. 1981. Generalized quantifiers and natural language. *Linguistics and Philosophy* 4:159–219.

Barwise, Jon, and John Perry. 1983. *Situations and attitudes*. Bradford Books – MIT Press.

Beaver, David. 1997. Presupposition. In *Handbook of logic and language*, ed. J. van Benthem and A. ter Meulen, 939–1008. MIT Press.

Beaver, David, and Cleo Condoravdi. 2003. A uniform analysis of *Before* and *After*. In *Proc. of SALT XIII*, ed. R. Young and Y. Zhou, 37–54.

Beavers, John, and Ivan A. Sag. 2004. Coordinate ellipsis and apparent non-constituent coordination. In *Proc. of the HPSG'04 conference*, ed. Stefan Müller.

Beck, Sigrid. 2000. The semantics of *Different*: Comparison operator and relational adjective. *Linguistics and Philosophy* 23:101–139.

van Benthem, Johan. 1986. *Essays in logical semantics*. Reidel.

van Benthem, Johan. 2003. The categorial fine-structure of natural language. Tech. Report PP-2003-20, ILLC Amsterdam. http://www.illc.uva.nl/Publications/ResearchReports/PP-2003-20.text.pdf.

Blackburn, Patrick, and Johan Bos. 2005a. *Representation and inference for natural language: A first course in computational semantics*. CSLI Publications.

Blackburn, Patrick, and Johan Bos. 2005b. *Working with discourse representation theory: An advanced course in computational semantics*. http://www.blackburnbos.org/.

Bobrow, Danny, Cleo Condoravdi, Richard Crouch, Ronald Kaplan, Lauri Kartunnen, Tracy Holloway King, Valeria de Paiva, and Annie Zaenen. 2005. A basic logic for textual inference. In *Proc. of the AAAI Workshop on Inference for Textual Question Answering*.

Bos, Johan. 1996. Predicate logic unplugged. In *Proc. of the 10th Amsterdam Colloquium*, 133–142.

Bos, Johan. 2004. Computational semantics in discourse: Underspecification, resolution, and inference. *Journal of Logic, Language and Information* 13:139–157.

Bos, Johan, E. Mastenbroek, S. McGlashan, S. Millies, and M. Pinkal. 1994. A compositional DRS-based formalism for NLP applications. In *Proc. of the International Workshop on Computational Semantics*.

Bresnan, Joan. 1973. Syntax of the comparative clause construction in English. *Linguistic Inquity* IV:275–343.

Bresnan, Joan. 1982. Control and complementation. In *The mental representation of grammatical relations*, ed. Joan Bresnan, 282–390. MIT Press.

Bresnan, Joan. 2001. *Lexical Functional Syntax*. Blackwell Publishers.

Carpenter, Bob. 1998. *Type-logical semantics*. MIT Press.

Chaves, Rui P. 2002. Principle-based DRTU for HPSG: a case study. In *1st Intl. Workshop on Scalable Natural Language Understanding - SCANALU'02*.

Chaves, Rui P. 2003. Non-redundant scope disambiguation in underspecified semantics. In *Proc. of the 8th ESSLLI Student Session*, ed. Balder ten Cate, 47–58.

Chierchia, Gennaro. 1995. *Dynamics of meaning: Anaphora, presupposition, and the theory of grammar*. University of Chicago Press.

Clark, Peter, et al. 2005. Reading to learn. Boeing Phantom Works. Presentation, October 17th.

Cooper, Robin. 1983. *Quantification and syntactic theory*. Dordrecht: Reidel.

Copestake, Ann, Dan Flickinger, Carl Pollard, and Ivan Sag. 2005. Minimal Recursion Semantics: an introduction. *Research on Language and Computation* 3:281–332.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to algorithms*. The MIT Press, 2nd edition.

Crouch, Richard. 1999. Ellipsis and glue languages. In (Lappin and Benmamoun, 1999), 32–67.

Crouch, Richard. 2005. Packed rewriting for mapping semantics to KR. In *Proc. of the 6th IWCS*.

Crouch, Richard, and Josef van Genabith. 1999. Context change, underspecification, and the structure of glue language derivations. In (Dalrymple, 1999).

Crouch, Richard, and Josef van Genabith. 2000. Linear logic for linguists. Introductory Course, ESSLLI-00. http://www2.parc.com/istl/members/crouch/esslli00_notes.pdf.

Dagan, Ido, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Proc. of the PASCAL Challenge Workshop on Recognizing Textual Entailment*.

Dalrymple, Mary, ed. 1999. *Semantics and syntax in lexical functional grammar: The resource logic approach*. MIT Press.

Dalrymple, Mary. 2001. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics Series*. Academic Press.

Dalrymple, Mary, Makoto Kanazawa, Yookyung Kim, Sam Mchombo, and Stanley Peters. 1998. Reciprocal expressions and the concept of reciprocity. *Linguistics and Philosophy* 21:159–210.

Dalrymple, Mary, John Lamping, Fernando Pereira, and Vijay Saraswat. 1999. Quantification, anaphora, and intensionality. In (Dalrymple, 1999).

Dalrymple, Mary, Stuart Shieber, and Fernando Pereira. 1991. Ellipsis and higher-order unification. *Linguistics and Philosophy* 14:399–452.

Davidson, Donald. 1967. The logical form of action sentences. In *The logic of decision and action*, ed. Nicholas Rescher. University of Pittsburgh Press.

Devlin, Keith. 1991. *Logic and information*. Cambridge University Press.

Devlin, Keith. 2006. Situation theory and situation semantics. In *Handbook of the history of logic*, ed. Dov M. Gabbay and John Woods, volume 7, 601–664.

Dimitriadis, Alexis. 2004. Discontinuous reciprocals. Unpublished Ms, Utrecht institute of Linguistics OTS, http://www.let.uu.nl/∼alexis.dimitriadis/personal/papers/discon-long-ms04.pdf.

van der Does, Jaap. 1993. Sums and quantifiers. *Linguistics and Philosophy* 16:509–550.

Doherty, Paul C., and Arthur Schwartz. 1967. The syntax of the compared adjective in English. *Language* 43:903–936.

Dowty, David, Robert E. Wall, and Stanley Peters. 1980. *Introduction to Montague semantics*. Springer.

Egg, Markus, Alexander Koller, and Joachim Niehren. 2001. The constraint language for lambda structures. *Journal of Logic, Language, and Information* 10:457–485.

van Eijck, Jan, and Hans Kamp. 1997. Representing discourse in context. In *Handbook of logic and language*, ed. J. van Benthem and A. ter Meulen. MIT Press.

Emms, Martin. 1992. Logical ambiguity. Doctoral Dissertation, University of Edinburgh.

Evans, Gareth. 1980. Pronouns. *Linguistic Inquiry* 11:337–362.

Francez, Nissim, and Ian Pratt. 1997. Derivation of temporal preposition meanings in LFG's glue-language approach. In *Proc. of LFG'97*.

Frank, Anette, and Josef van Genabith. 2001. LL-based semantics construction for LTAG – and what it teaches us about the relation between LFG and LTAG. In *Proc. of the LFG 2001 conference*, ed. M. Butt and T. H. King. CSLI Publications.

Fry, John. 1999a. Resource-logical event semantics for LFG. Unpublished draft. http://www-csli.stanford.edu/∼jfry/events.html.

Fry, John. 1999b. Resource-logical event semantics for LFG. Talk given at the LFG'99 conference. http://www-csli.stanford.edu/∼jfry/events.html.

Fuchs, Norbert E. 2005. Attempto Controlled English. Talk slides, Stanford University, December 2005. http://attempto.ifi.unizh.ch/site/talks/files/Talk.Stanford.05.pdf.

Fuchs, Norbert E., Kaarel Kaljurand, and Gerold Schneider. 2006. Attempto Controlled English meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. In *Proc. of FLAIRS'2006*.

Fuchss, Ruth, Alexander Koller, Joachim Niehren, and Stefan Thater. 2004. Minimal recursion semantics as dominance constraints: Translation, evaluation, and analysis. In *Proc. of the 42nd ACL*.

Gabbay, D.M., C.J. Hogger, and J.A. Robinson, ed. 1994. *Handbook of logic in artificial intelligence and logic programming*, volume 3: Nonmonotonic Reasoning and Uncertain Reasoning.

Gallier, Jean. 1993. Constructive logics. Part I: A tutorial on proof systems and typed lambda-calculi. *Theoretical Computer Science* 110(2):249–239.

Gamut, L.T.F. 1991a. *Logic, language, and meaning*, volume I: Introduction to Logic. University of Chicago Press.

Gamut, L.T.F. 1991b. *Logic, language, and meaning*, volume II: Intensional Logic and Logical Grammar. University of Chicago Press.

Gawron, Jean Mark, John Nerbonne, and Stanley Peters. 1991. The absorption principle and E-type anaphora. In *Situaion theory and its applications*, ed. J. Barwise et al., volume 2 of *CSLI Lecture Notes No. 26*. CSLI Publications.

Gawron, Jean Mark, and Stanley Peters. 1990. *Anaphora and quantification in situation semantics*. CSLI Publications.

van Genabith, Josef, and Richard Crouch. 1999. How to glue a donkey to an F-structure, or porting a dynamic meaning representation language into LFG's linear logic based glue language semantics. In *Computing meaning, volume 1*, ed. Harry Bunt and Reinhard Muskens, volume 73 of *Studies in Linguistics and Philosophy*, 129–148. Kluwer Academic Press.

Ginzburg, Jonathan. 1990. On the non-unity of symmetric predicates: Monadic comitatives and dyadic equivalence relations. In *Proc. of NELS 20, GLSA, UMass, Amherst*, ed. J. Carter et al.

Ginzburg, Jonathan, and Ivan A. Sag. 2001. *Interrogative investigations: The form, meaning, and use of English interrogatives*. CSLI Publications.

Girard, Jean-Yves. 1987. Linear logic. *Theoretical Computer Science* 50:1–102.

Grice, Paul H. 1975. Logic and conversation. In *Syntax and semantics*, ed. P. Cole and J. Morgan, volume 3, 41–58. New York: Academic Press.

Groenendijk, Jeroen, and Martin Stokhof. 1990. Dynamic Montague grammar. In *Papers from the second symposium on logic and language*, ed. L. Kalman and L. Polos, 3–48. Akademiai Kiadoo, Budapest.

Groenendijk, Jeroen, and Martin Stokhof. 1991. Dynamic predicate logic. *Linguistics and Philosophy* 14:39–100.

Grosz, Barbara J., Aravind K. Joshi, and Scott Weinstein. 1995. Centering: A framework for modelling the local coherence of discourse. *Computational Linguistics* 21:203–226.

Gupta, Vineet, and John Lamping. 1998. Efficient linear logic meaning assembly. In *Proc. of COLING/ACL'98*.

Hardt, Daniel. 1999. Dynamic interpretation of verb phrase ellipsis. *Linguistics and Philosophy* 22:185–219.

Heim, Irene. 1982. The semantics of definite and indefinite noun phrases in English. Doctoral Dissertation, University of Massachusetts, Amherts.

Heim, Irene. 1985. Note on comparatives and related matters. Ms., University of Texas, Austin. http://semanticsarchive.net/Archive/zc0ZjY0M/Comparatives%2085.pdf.

Hendriks, Herman. 1993. Studied flexibility. Doctoral Dissertation, University of Amsterdam.

Hepple, Mark. 1996. A compilation-chart method for linear categorial deduction. In *Proc. of COLING'96*, 537–542.

Hepple, Mark. 1998. Memoisation for glue language deduction and categorial parsing. In *Proc. of COLING-ACL'98*, 538–544.

Higginbotham, James. 1980. Reciprocal interpretation. *Journal of Linguistic Research* 1:97–117.

Higgins, Derrick, and Jerrold M. Sadock. 2003. A machine learning approach to modeling scope preferences. *Computational Linguistics* 29:73–96.

Hobbs, Jerry R., and Stuart M. Shieber. 1987. An algorithm for generating quantifier scopings. *Computational Linguistics* 13:47–63.

Howard, W. A. 1980. The 'formulae as types' notion of construction. In *To H. B. Curry: Essays on combinatory logic, lambda calculus and formalism*, ed. J. P. Seldin and J. R. Hindley, 479–490. Academic Press. Reprint of manuscript first published in 1969.

Jacobson, Pauline. 1999. Towards a variable-free semantics. *Linguistics and Philosophy* 22:117–184.

Kadmon, Nirit. 1987. On unique and non-unique reference and asymmetric quantification. Doctoral Dissertation, University of Massachusettes, Amherst.

Kamp, Hans. 1981. A theory of truth and semantic representation. In *Formal methods in the study of language*, ed. Jeroen Groenendijk, T.M.V. Janssen, and Martin Stokhof, 277–322. Mathematical Centre Tract 135, Amsterdam.

Kamp, Hans, and Uwe Reyle. 1993. *From discourse to logic*. Dordrecht: Kluwer.

Kanazawa, Makoro. 1994. Weak vs. strong readings of donkey sentences and monotonicity inference in a dynamic setting. *Linguistics and Philosophy* 17:109–158.

Kaplan, R., S. Riezler, T. H. King, J. T. Maxwell III, A. Vasserman, and R. Crouch. 2004a. Speed and accuracy in shallow and deep stochastic parsing. In *Proc. of HLT-NAACL'04*.

Kaplan, Ronald, John T. Maxwell III, Tracy Holloway King, and Richard Crouch. 2004b. Integrating finite-state technology with deep LFG grammars. In *Proc. of the ESSLLI 2004 Workshop on Combining Shallow and Deep Processing for NLP*.

Keenan, Edward L., and Dag Westerståhl. 1997. Generalized quantifiers in linguistics and logic. In *Handbook of logic and language*, ed. J. van Benthem and A. ter Meulen, 837–893. MIT Press.

Keller, W. R. 1988. Nested cooper storage: The proper treatment of quantification in ordinary noun phrases. In *Natural language parsing and linguistic theories*, ed. U. Reyle and C. Rohrer. Dordrecht: Reidel.

Kennedy, Christopher. 1997. Projecting the adjective: The syntax and semantics of gradability and comparison. Doctoral Dissertation, University of California, Santa Cruz.

Kohlhase, Michael, Susanna Kuschert, and Manfred Pinkal. 1996. A type-theoretic semantics for lambda-drt. In *Proc. of the 10th Amsterdam Colloquium*, ed. P. Dekker and M. Stokhof, 479–498. de Gruyter.

Kokkonidis, Miltiadis. 2005. Why glue your donkey to an F-structure when you can constrain and bind it instead? In *Proceedings of LFG05 Conference*, ed. Miriam Butt and Tracy Holloway King. CSLI Publications.

Kokkonidis, Miltiadis. 2006. First-order glue. *Journal of Logic, Language and Information* To appear.

Koller, Alexander, Joachim Niehren, and Stefan Thater. 2003. Bridging the gap between underspecification formalisms: Hole semantics as dominance constraints. In *Proc. of the 11th EACL*.

Koller, Alexander, and Stefan Thater. 2006. An improved redundancy elimination algorithm for underspecified descriptions. In *Proc. of COLING/ACL-2006*.

Kuncak, Viktor, Huu Hai Nguyen, and Martin Rinard. 2006. Deciding boolean algebra with Presburger arithmetic. *Journal of Automated Reasoning* 36:213–239.

Lakoff, George, and Mark Johnson. 1980. *Metaphors we live by*. University Of Chicago Press, 2nd edition.

Landman, Fred. 2000. *Events and plurality: The Jerusalem lectures*. Studies in Linguistics and Philosophy. Kluwer Academic Publishers.

Lappin, Shalom, and Elabbas Benmamoun. 1999. *Fragments: Studies in ellipsis and gapping*. Oxford University Press.

Lappin, Shalom, and Nissim Francez. 1994. E-type pronouns, I-sums, and donkey anaphora. *Linguistics and Philosophy* 17:391–428.

Lev, Iddo. 2005a. Decoupling scope resolution from semantic composition. In *Proc. of the 6th International Workshop on Computational Semantics (IWCS-6)*, 139–150.

Lev, Iddo. 2005b. Gradable comparatives: Syntax and syntax-semantics interface. Unpublished MS., Stanford University. http://www-csli.stanford.edu/∼iddolev/papers/iddolev_ling221b_paper.pdf.

Lev, Iddo. 2006. Logic puzzles: A new test-suite for compositional semantics and reasoning. http://www-csli.stanford.edu/∼iddolev/papers/lev_logic_puzzles_submitted.pdf.

Lev, Iddo, Bill MacCartney, Christopher D. Manning, and Roger Levy. 2004. Solving logic puzzles: From robust processing to precise semantics. In *Proc. of the 2nd workshop on text meaning and interpretation, ACL'04*.

Levin, Beth, and Malka Rappaport Hovav. 2005. *Argument realization*. Research Surveys in Linguistics Series. Cambridge University Press.

Link, Godehard. 1998. *Algebraic semantics in language and philosophy*. Number 74 in CSLI Lecture Notes. CSLI Publications.

Lønning, Jan Tore. 1997. Plurals and collectivity. In *Handbook of logic and language*, ed. J. van Benthem and A. ter Meulen, 1009–1053. MIT Press.

Maxwell, John T., III, and Ronald M. Kaplan. 1991. A method for disjunctive constraint satisfaction. In *Current issues in parsing technology*, ed. Masaru Tomita, 173–190. Kluwer Academic Publishers. Reprinted in Dalrymple et al. (eds), Formal Issues in Lexical-Functional Grammar, CSLI Publications, 1995.

Maxwell, John T., III, and Ronald M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics* 19:571–590.

Maxwell, John T., III, and Ronald M. Kaplan. 1996. Unification-based parsers that automatically take advantage of context freeness. In *Proc. of LFG'96 Conference*. http://www.parc.xerox.com/research/publications/details.php?id=3115.

McCarthy, John. 1981. Formalization of two puzzles involving knowledge. http://www-formal.stanford.edu/jmc/puzzles.pdf.

McDermott, Drew. 1978. Tarskian Semantics, or No Notation Without Denotation! *Cognitive Science* 2.

Montague, R. 1974. The proper treatment of quantification in ordinary English. In *Formal philosophy*, ed. R. Thomason. Yale University Press.

Moortgat, Michael. 1990. *The quantification calculus: questions of axiomatization*. In Deliverable R.1.2.A of DYANA: Dynamic Interpretation of Natural Language. ESPRIT Basic Research Action BR3175. Centre for Cognitive Science, Edinburgh.

Moran, Douglas B., and Fernando C. N. Pereira. 1992. Quantifier scoping. In Alshawi (1992), chapter 8.

Morrill, Glyn V. 1994. *Type logical grammar: Categorial logic of signs*. Springer.

Moss, Lawrence S. 2006. Completeness theorems for syllogistic fragments. http://www.indiana.edu/∼iulg/moss/uwe.pdf.

Muskens, Reinhard. 1995. *Meaning and partiality*. CSLI Publications.

Muskens, Reinhard. 1996. Combining Montague semantics and discourse representation. *Linguistics and Philosophy* 19:143–186.

Nairn, Rowan, Cleo Condoravdi, and Lauri Karttunen. 2006. Computing relative polarity for textual inference. In *Proc. of ICoS-5 (Inference in Computational Semantics)*.

Nelken, Rani, and Nissim Francez. 1996. Translating natural language system specifications into temporal logic via DRT. Technical report LCL 9602, Laboratory for Computational Linguistics, Technion, Israel.

Nelken, Rani, and Nissim Francez. 2000. Querying temporal databases using controlled natural language. In *Proc. of COLING'2000*.

Niehren, Joachim, and Stefan Thater. 2003. Bridging the gap between underspecification formalisms: Minimal recursion semantics as dominance constraints. In *Proc. of the 41st ACL*, 367–374.

Nunberg, Geoffrey. 1984. Individuation in context. In *Proceedings of WCCFL 3*, 203–217.

Parsons, Terence. 1990. *Events in the semantics of English*. The MIT Press.

Pasca, Marius, and Sanda M. Harabagiu. 2001. High performance question/answering. In *Proc. of SIGIR*, 366–374.

Peters, Stanley, and Dag Westerståhl. 2002. Does English really have resumptive quantification? In *The construction of meaning*, ed. D. Beaver et al. CSLI Publications.

Peters, Stanley, and Dag Westerståhl. 2006. *Quantifiers in language and logic*. Oxford University Press.

Popescu, Ana-Marie, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proc. of the conference on Intelligent User Interfaces*.

Pratt, Ian, and Nissim Francez. 2001. Temporal prepositions and temporal generalized quantifiers. *Linguistics and Philosophy* 24:187–222.

Pratt-Hartmann, Ian. 2003. A two-variable fragment of English. *Journal of Logic, Language and Information* 12:13–45.

Pratt-Hartmann, Ian. 2006. On the complexity of the numerically definite syllogistic and related fragments. http://arxiv.org/PS_cache/cs/pdf/0701/0701039v1.pdf.

Pulman, Stephen. 1991. Comparatives and ellipsis. In *Proc. of the fifth conference of the EACL*. http://acl.ldc.upenn.edu/E/E91/E91-1002.pdf.

Rayner, Manny, and Amelie Banks. 1990. An implementable semantics for comparative constructions. *Computational Linguistics* 16:86–112.

Reyle, Uwe. 1993. Dealing with ambiguities by underspecification: Construction, representation and deduction. *Journal of Semantics* 10:123–179.

Richter, Frank, and Manfred Sailer. 2003. Basic concepts of lexical resource semantics. In *Series of the Kurt Gödel society, Vienna*. Also appeared as ESSLLI'03 course notes.

Riezler, Stefan, and Alexander Vasserman. 2004. Incremental feature selection and L1 regularization for relaxed maximum-entropy modeling. In *Proc. of EMNLP'04*.

Roberts, Craige. 1991. Distributivity and reciprocal distributivity. In *Proc. of the first SALT conference*.

Rooth, Mats. 1987. Noun phrase interpretation in Montague grammar, file change semantics, and situation semantics. In *Generalized quantifiers*, ed. Peter Gärdenfors. D. Reidel Publishing.

Russel, Stuart, and Peter Norvig. 2003. *Artificial intelligence: A modern approach*. Prentice Hall, 2nd edition.

van der Sandt, Rob A. 1992. Presupposition projection as anaphora resolution. *Journal of Semantics* 9:333–377.

Scha, Remko. 1981. Distributive, collective and cumulative quantification. In *Formal methods in the study of language*, ed. J.A.G. Groenendijk, T.M.V. Janssen, and M.B.J. Stokhof, 483–512. Amsterdam: Mathematisch Centrum.

Searle, John R. 1969. *Speech acts*. Cambridge University Press.

Siloni, Tal. 2005. The syntax of reciprocal verbs: An overview. In *Reciprocals and reflexives: Cross-linguistic and theoretical explorations*, ed. Ekkehard König and Volker Gast, Trends in Linguistics. Mouton de Gruyter. (To appear). http://www.tau.ac.il/∼siloni/ReciprocalVerbs2005.pdf.

Smullyan, Raymond M. 1978. *What is the name of this book?*. Prentice-Hall.

Steedman, Mark. 2000. *The syntactic process*. MIT Press.

de Swart, Henriette. 1998. *Introduction to natural language semantics*. CSLI Publications.

Veltman, Frank. 2000. Proof systems for dynamic predicate logic. Ms., Department of Philosophy, University of Amsterdam. http://staff.science.uva.nl/∼veltman/papers/DPL2000.pdf.

Weber, Karl. 1999. *The unofficial guide to the GRE test*. ARCO Publishing, 2000 edition.

Winter, Yoad. 2005. On inference with scopally ambiguous sentences. Talk at the LSA workshop: Proof Theory at the Syntax/Semantics Interface. http://www.cs.technion.ac.il/∼winter/papers/ms-talk-LSA.pdf.