UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Formalizando Concorrência com Lógica Categorial

por

Aline Vieira Malanovicz

Tiarajú Asmuz Diverio

Paulo Fernando Blauth Menezes

RP-323 Junho/2002

Relatório de Pesquisa

UFRGS – II – PPGC

Caixa Postal 15064 – CEP 91509-900

Porto Alegre – RS – Brasil
Telefone: (51) 3316-6168
Fax: (51) 3316-7308
E-mail: ppgc@inf.ufrgs.br

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Malanovicz, Aline Vieira; Diverio, Tiarajú Asmuz; Menezes, Paulo Fernando Blauth.

Formalizando Concorrência com Lógica Categorial / Aline Vieira Malanovicz, Tiarajú Asmuz Diverio, Paulo Fernando Blauth Menezes.

93 p.: il.

Relatório de Pesquisa – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, 2002.

1. Lógica. 2. Lógica Categorial. 3. Lógica Intuicionista. 4. Concorrência. 5. Programação Concorrente. 6. Formalização da Concorrência. 7. Teoria das Categorias.

Aline Vieira Malanovicz

Prof. Dr. Tiarajú Asmuz Diverio

Prof. Dr. Paulo Fernando Blauth Menezes

CNPq-PIBIC/UFRGS

Sumário

SUMÁRIO 4
LISTA DE FIGURAS 6
RESUMO 7
PALAVRAS-CHAVE 7
INTRODUÇÃO 8
APRESENTAÇÃO 8
JUSTIFICATIVA 8
METODOLOGIA 8
ORGANIZAÇÃO DO TEXTO 9
1 PROGRAMAÇÃO CONCORRENTE 11
1.1 Introdução 11
1.2 Definição 12
1.3 Motivação 13
1.4 Especificação do paralelismo 16
1.5 Visão Geral e Comparação 21
1.6 Deadlock 22
1.7 Exercícios 23
2 TEORIA DAS CATEGORIAS 25
2.1 Introdução à Teoria das Categorias 25
2.2 Conceito de Categoria 27
Dualidade 27
Conjuntos parcialmente ordenados 28
Conjunto Parcialmente Ordenado Visto como uma Categoria 28
Limite e colimite 29
Produto e Coproduto como Casos Especiais de Limite e Colimite 30
3 LÓGICA 33
3.1 Apologias à Lógica 33
3.2 Introdução à Lógica 33
Satisfação 34
Sistemas de Avaliação 34
Teoria da Prova 35
Relações de Conseqüência 35
Relações de Acarretamento (entailment relations) 36
Propriedades Semânticas das Apresentações 36
3.3 Lógica Clássica Proposicional 37
Apresentação Axiomática 37

Sistema de Avaliação 37
Semântica 38
Corretude e Completude 38
3.4 Lógica Clássica de Primeira Ordem 39
Axiomas 39
Regras de Inferência 40
3.5 Relação entre Lógica e Ciência da Computação 41
3.6 Problemas Bem-Humorados Envolvendo Lógica 41
3.7 Exercícios 42
4 LÓGICAS CATEGORIAIS 44
4.1 Breve Apresentação da Lógica Categórica 44
4.2 Morfismos e Objetos Especiais 47
4.3 Isomorfismo de Categorias: Conjuntos Finitos e Lógica Booleana 48
4.4 Álgebras e Lógicas de Primeira Ordem 49
5 LÓGICA INTUICIONISTA 52
5.1 Pensamento axiomático versus pensamento intuicionista em matemática 52
5.2 Intuicionismo: teoria de tipos de Martin-Löf 58
5.3 A Interpretação Intuicionista das Constantes Lógicas 59
5.4 A Linguagem da Teoria de Tipos 62
5.5 Julgamentos e Regras de Inferência 64
5.6 A Teoria de Tipos como uma Linguagem de Programação 66
5.7 Teoria de Tipos como Categorias 68
6 PROGRAMAÇÃO CONCORRENTE FORMALIZADA POR LÓGICA CATEGORIAL 71
6.1 Relação da especificação com os comandos 71
6.2 Modelos de Máquinas Paralelas 72
6.3 Grafos Acíclicos Direcionados - (GAD) 73
6.4 Formalização de Sistemas Concorrentes 73
6.5 Aplicação da Teoria das Categorias à Especificação Formal de Sistemas Concorrentes 74
7 CONCLUSÕES 80
BIBLIOGRAFIA 82
ENDEREÇOS WEB 85
ANEXO 1 – ESQUEMA DE ESTUDO PARA LÓGICA INTUICIONISTA 86
ANEXO 2 – RESPOSTAS AOS EXERCÍCIOS PROPOSTOS 89
ANEXO 3 – ESQUEMA DE ESTUDO PARA LÓGICA CATEGORIAL 91

LISTA DE FIGURAS

Figura 1.1 Programa seqüencial fazendo acesso a arquivo e impressora 13
Figura 1.2 Linha de tempo do programa seqüencial da Figura 1.1 14
Figura 1.3 Programa concorrente fazendo acesso a arquivo e impressora 14
Figura 1.4 Linha de tempo do programa concorrente da Figura 1.3 15
Figura 1.5 Rede local incluindo um servidor de impressão dedicado 15
Figura 1.6 Servidor de impressão como programa concorrente 16
Figura 1.7 Exemplo contendo os comandos *Fork*, *Exit* e *Wait* 18
Figura 1.8 Diagrama de tempo associado com o programa da Figura 1.7 18
Figura 1.9 Diagrama de tempo associado com o programa da Figura 1.7 18
Figura 1.10 Grafo de precedência para o programa da Figura 1.7 19
Figura 1.11 Programa do exemplo da Figura 1.7 alterado 19
Figura 1.12 Exemplo contendo os comandos *Parbegin* e *Parend* 20
Figura 1.13 Gráfico representando processos e recursos 22
Figura 2.1 Conjunto parcialmente ordenado visto como uma categoria 28
Figura 2.2 Conjunto parcialmente ordenado visto como uma categoria 29
Figura 2.3 Diagrama comutativo para cone 29
Figura 2.4 Diagrama comutativo para limite 30
Figura 2.5 Diagrama comutativo para colimite 30
Figura 2.6 Objeto terminal como limite 30
Figura 2.7 Diagrama comutativo para produtos binários 31
Figura 2.8 Diagrama comutativo para coproduto 31
Figura 2.9 Produto como limite 31
Figura 5.1 Provas($P \vee Q$) = Provas(P) + Provas(Q) 69
Figura 5.2 Provas($P \wedge Q$) = Provas(P) × Provas(Q) 69
Figura 5.3 Provas($P \rightarrow Q$) = Provas(P) → Provas(Q) 69
Figura 5.4 Provas($(\exists x \in D)\, P(x)$) = $(\Sigma y \in D)$ Provas($P(y)$) 69
Figura 5.5 Provas($(\forall x \in D)\, P(x)$) = $(\Pi y \in D)$ Provas($P(y)$) 69
Figura 5.6 Exemplo de inferência 69
Figura 6.1 Produto em conjunto parcialmente ordenado: dependência causal/concorrência 75
Figura 6.2 Trecho de programa usando as primitivas fork e exit 75
Figura 6.3 Conjunto parcialmente ordenado 76
Figura 6.4 Trecho de programa usando as primitivas fork, exit e wait 76
Figura 6.5 Quadrado da independência 77
Figura 6.6 Grafos reflexivos 77
Figura 6.7 Produto de grafos reflexivos 77

RESUMO

A Teoria da Computação Concorrente, através das poderosas ferramentas que são a Teoria das Categorias e sua base, Lógica Categorial, tem alcançado importância crescente nas pesquisas científicas atualmente, devido ao seu uso em especificações de sistemas de *software* com exatidão, clareza, correção e ausência de ambigüidade. Entre os objetivos desta pesquisa, está o estudo da Lógica Categorial, visando à formalização da Teoria da Computação Concorrente. Também é um objetivo da pesquisa a produção de uma apostila sobre Lógica Categorial que tenha um texto autocontido, de fácil entendimento e enriquecido com exemplos de aplicação a sistemas concorrentes, questionamentos e exercícios com dicas. A razão para essa produção consiste em facilitar o ensino, o aprendizado e a obtenção de um embasamento teórico sobre a matéria, visando à sua aplicação à formalização da Teoria da Computação Concorrente. Até o presente momento, os resultados iniciais da pesquisa desenvolvida apontam a falta de textos didáticos sobre Lógica Categorial (tanto nacionais quanto internacionais) para o nível da graduação. Além disso, fornecem considerável quantidade de material sobre Teoria das Categorias, Lógica e aplicações de Lógica Categorial, além de uma modesta seleção de exemplos, questionamentos e exercícios a ser ampliada para uso no texto a ser produzido. Os próximos passos do trabalho envolvem a produção do material didático proposto e sua validação através de minicursos para alunos e professores de disciplinas do Departamento de Informática Teórica da UFRGS, além da conseqüente disponibilização para a comunidade acadêmica via *web* e a adoção em uma disciplina de tópicos junto ao PPGC.

Palavras-chave

Lógica, Lógica Categorial, Lógica Intuicionista, Concorrência, Programação Concorrente, Formalização da Concorrência, Teoria das Categorias.

Introdução

Apresentação

O presente trabalho consiste em uma coleção de conceitos, exemplos, construções e resultados referentes à aplicação de lógicas categóricas, especialmente a lógica intuicionista, à especificação formal de sistemas concorrentes. O objetivo é apresentar uma visão geral (introdutória) sobre Lógica Categorial para fins de ensino/relação com ensino de Teoria das Categorias/relação com especificação de sistemas concorrentes. Em outras palavras, elaborar um texto didático autocontido e de fácil entendimento para o aprendizado de Lógica Categorial, através da exposição sintética das idéias encontradas, bem como sua análise crítica e comparativa, consoante com a bibliografia consultada, para fins didáticos, com a produção do texto didático autocontido para o ensino de lógica categorial de maneira adequada ao nível de conhecimento de alunos de graduação. O presente texto resulta de extensa pesquisa bibliográfica para que seja enriquecido com o máximo de exemplos e exercícios com dicas para facilitar o aprendizado do ponto de vista do aluno de graduação.

Justificativa

A importância do assunto refere-se ao poder da ferramenta que é a Teoria das Categorias e sua base lógica para a especificação de sistemas concorrentes conforme a Teoria da Computação Concorrente, a qual vem sendo alvo de pesquisas atualmente no mundo todo. Outro aspecto que justifica a realização da presente pesquisa é a necessidade de suprir a falta de textos didáticos (tanto nacionais quanto internacionais) para o ensino de Lógica Categorial e fornecer um embasamento teórico de fácil compreensão para auxiliar no processo de ensino-aprendizado de lógica categorial para os propósitos das disciplinas dos cursos de bacharelado em Ciência da Computação que exigem essa matéria como pré-requisito. A abordagem da lógica baseada em categorias justifica-se porque a Teoria das Categorias é uma ferramenta que permite expressar as teorias da Lógica de uma maneira fácil e clara, através de sa forma diagramática de representar as inferências.

Supõe-se que, pesquisando e estudando uma considerável variedade de material sobre o assunto, possa-se atingir uma compreensão do assunto a tal ponto de poder discernir se a elaboração do texto didático é realmente necessária e reunir, adaptar e desenvolver as melhores e mais eficientes abordagens, exemplos, exercícios, dicas, questionamentos e explicações sobre o tema com vistas a enriquecer o texto que será produzido.

Metodologia

Para atingir o objetivo exposto acima, os seguintes objetivos específicos foram atingidos: pesquisa bibliográfica, estudo e compilação de trechos de obras, produção do texto didático a partir da combinação e do desenvolvimento do material resultante da pesquisa - Pesquisa bibliográfica através de sites de busca de informações eletrônicas na Internet e da biblioteca do Instituto de Informática da UFRGS. Leitura criteriosa do material selecionado, resumo de suas idéias principais e abordagens didáticas consideradas mais interessantes, assim como exemplos, questionamentos e exercícios propostos sobre o tema – Validação através de miniaulas de Lógica Categorial para

público leigo. Mais detalhadamente, as atividades desenvolvidas para a consecução deste trabalho foram: pesquisar sobre os conceitos básicos de Lógica Categorial, pesquisar sobre as aplicações da lógica categorial à Ciência da Computação, pesquisar sobre materiais didáticos produzidos sobre a Lógica Categorial, reunir material que proporcione embasamento teórico sobre o assunto, pesquisar e desenvolver exemplos, exercícios, dicas, abordagens, questionamentos e explicações que facilitem o entendimento para leigos no assunto, elaborar relatórios de pesquisa e encaminhar artigos a eventos.

Organização do Texto

O conteúdo texto do presente trabalho está distribuído entre os capítulos do mesmo como segue:

Capítulo 1: Programação Concorrente

Capítulo 2: Teoria das Categorias

Capítulo 3: Lógica

Capítulo 4: Lógicas Categoriais

Capítulo 5: Lógica Intuicionista

Capítulo 6: Programação Concorrente Formalizada por Lógica Categorial

Capítulo 7: Conclusões

Ainda são apresentados três anexos antes da bibliografia geral do trabalho, os quais contém, respectivamente: um esquema de estudo para Lógica Intuicionista (Capítulo 5); respostas aos exercícios propostos sobre Lógica (Capítulo 3); e outro esquema de estudo, desta vez sobre Lógicas Categoriais (Capítulo 4).

1 Programação Concorrente

1.1 Introdução

O estudo de sistemas distribuídos é uma área de pesquisa importante na Ciência da Computação. Um sistema distribuído consiste de um certo número de componentes autônomos que interagem entre si a fim de realizarem uma tarefa conjunta.

A teoria dos sistemas distribuídos consiste da formulação de modelos matemáticos abstratos e do estudo das propriedades desses modelos. Como motivações básicas no estudo de modelos formais, podemos citar [SOU99]:

- Desenvolver ferramentas e técnicas com as quais se possa especificar, analisar e implementar sistemas distribuídos;
- O desenvolvimento de meios formais para raciocinar sobre o comportamento de sistemas distribuídos;
- A sugestão de meios ou formas para o projetista desenvolver sistemas melhores, mais elegantes e com descrições formais mais simples.

Uma das principais características dos sistemas distribuídos é a concorrência, ou seja, o fato de que há vários processos envolvidos cujos eventos podem ocorrer simultaneamente.

A Lógica Categorial é uma linguagem formal de especificação que serve bem para a descrição e análise de aspectos comportamentais de sistemas concorrentes. Ela fornece uma maneira simples e natural, porém precisa, de falar sobre o ordenamento da ocorrência das interações [Manna e Pnuelli *apud* SOU99].

Uma *metodologia formal* tipicamente consiste de vários elementos. Um elemento é a linguagem de especificação na qual os requerimentos antecipados de um programa podem ser formalmente especificados. Um outro é um repertório de métodos de prova pelos quais a corretude de um programa proposto, com relação à especificação, pode ser formalmente verificada. As vantagens de uma metodologia formal são óbvias. A especificação formal força os projetistas de um programa a tomar antes decisões precisas sobre as principais funcionalidades do programa e a remover ambigüidades da descrição de seu comportamento esperado. A verificação formal de uma propriedade garante que a propriedade vale sobre todas as possíveis execuções do programa.

Como linguagem de especificação, a Lógica Categorial é apropriada e conveniente para especificar o comportamento dinâmico de programas concorrentes e descrever suas propriedades. A principal vantagem da Lógica Categorial é que ela fornece, através do uso de um conjunto especial de operadores, expressão sucinta e natural de propriedades de programas que ocorrem freqüentemente. [SOU99]

ÁREAS DE APLICAÇÃO DE PROCESSAMENTO PARALELO (Simulação numérica de fenômenos físicos)

- Química quântica, mecânica estatística e física relativística;
- Cosmologia e astrofísica
- Dinâmica de fluidos computacional e turbulência;

- Projeto de materiais e supercondutividade;
- Biologia, farmacologia, seqüenciamento de genoma, engenharia genética, modelagem celular;
- Medicina e modelagem de órgãos e ossos;
- Modelagem ambiental e climática global e lunar;
- Processamento sísmico.

1.2 Definição

Programação concorrente tem sido usada freqüentemente na construção de sistemas operacionais e em aplicações nas áreas de comunicação de dados e controle industrial. Esse tipo de programação torna-se ainda mais importante com o advento dos sistemas distribuídos e das máquinas com arquitetura paralela. Neste capítulo, serão discutidos os conceitos básicos e alguns mecanismos clássicos da programação Concorrente. Maiores detalhes podem ser encontrados no livro [OLI2000].

Um programa que é executado por apenas um processo é chamado de **programa seqüencial**. A grande maioria dos programas escritos são programas seqüenciais. Nesse caso, existe somente um fluxo de controle durante a execução. Isso permite, por exemplo, que o programador realize uma "execução imaginária" de seu programa, apontando com o dedo, a cada instante, a linha do programa que está sendo executada no momento.

Um **programa concorrente** é executado simultaneamente por diversos processos que cooperam entre si, isto é, trocam informações. Para o programador realizar agora uma "execução imaginária", ele vai necessitar de vários dados, um para cada processo que faz parte do programa. Nesse contexto, trocar informações significa trocar dados ou realizar algum tipo de sincronização. É necessária a existência de interação entre processos para que o programa seja considerado concorrente. Embora a interação entre processos possa ocorrer através do acesso a arquivos comuns, esse tipo de concorrência é tratado na disciplina de Banco de Dados. A programação concorrente tratada neste capítulo desta apostila é a utilizada nas disciplinas de Sistemas Operacionais, que utiliza mecanismos rápidos para interação entre processos: variáveis compartilhadas e troca de mensagens. [OLI2000]

O termo "programação concorrente" vem do inglês *concurrent programming*, no qual *concurrent* significa "acontecendo ao mesmo tempo". Uma tradução mais exata seria programação concomitante. Entretanto, o termo programação concorrente já está solidamente estabelecido no Brasil. Algumas vezes, é usado o termo **programação paralela** com o mesmo sentido.

O verbo "concorrer" admite, em português, vários sentidos. Pode ser usado no sentido de cooperar, como em "tudo concorria para o bom êxito da operação". Também pode ser usado com o significado de disputa ou competição, como em "ele concorreu a uma vaga na universidade". Em uma forma menos comum, ele significa também existir simultaneamente. De certa forma, todos os sentidos são aplicáveis aqui na programação concorrente. Em geral, processos concorrem (disputam) pelos mesmos recursos do hardware e do sistema operacional. Por exemplo, processador, memória, periféricos,

estruturas de dados, etc. Ao mesmo tempo, pela própria definição de programa concorrente, eles concorrem (cooperam) para o êxito do programa como um todo. Certamente, vários processos concorrem (existem simultaneamente) em um programa concorrente. [OLI2000] Logo, programação concorrente é um bom nome para o que vamos tratar neste capítulo.

É comum, em sistemas multiusuário, que um mesmo programa seja executado simultaneamente por vários usuários. Por exemplo, um editor de texto. Entretanto, executar simultaneamente dez instâncias do editor de texto não faz dele um programa concorrente. Apenas o código é possivelmente compartilhado pelos dez processos. Cada processo executa sobre sua própria área de dados e ignora a existência de outras execuções do programa. Esses processos não cooperam entre si, isto é, não trocam informações. Nesse exemplo, temos apenas a execução de dez instâncias do mesmo programa seqüencial, e não um programa concorrente.

1.3 Motivação

Notadamente, a programação concorrente é mais complexa que a programação seqüencial. Um programa concorrente pode apresentar todos os tipos de erro que normalmente aparecem em programas seqüenciais. Além disso, existem os erros associados com as interações entre os processos. Muitos erros dependem da velocidade relativa dos processos. Ou ainda, do exato instante de tempo em que o escalonador do sistema operacional realizou um chaveamento de contexto. Isso toma muitos erros difíceis de reproduzir e de identificar.

Mesmo com todas as suas complexidades inerentes, existem muitas áreas nas quais a programação concorrente é útil. Em sistemas nos quais existem vários processadores (máquinas paralelas ou sistemas distribuídos), é possível aproveitar esse paralelismo explicitamente e acelerar a execução do programa. Mesmo em sistemas com um único processador, existem razões para o seu uso em determinados tipos de aplicações.

Considere um programa que deve ler registros de um arquivo, colocá-los em um formato apropriado e então enviar para uma impressora física (em oposição a uma impressora lógica ou virtual, implementada com arquivos). Podemos fazer isso com um programa seqüencial que, dentro de um laço, faz as três operações. A Figura 1.1 ilustra tal programa, e a Figura 1.2 mostra a respectiva linha de tempo.

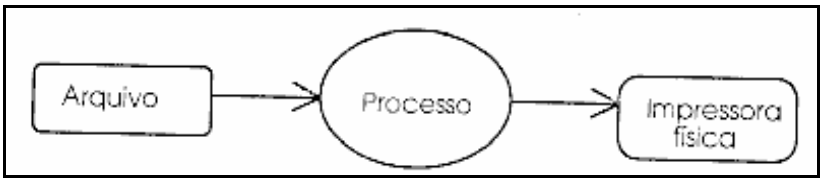


Figura 1.1 Programa seqüencial fazendo acesso a arquivo e impressora

Inicialmente, o processo envia um comando para a leitura do arquivo e fica bloqueado. O disco então é acionado para realizar a operação de leitura. Uma vez concluída a leitura, o processo realiza a formatação e inicia a transferência dos dados para a impressora. Como se trata de uma impressora física, o processo executa um laço no qual os dados são enviados para a porta serial ou paralela apropriada. Como o *buffer* da impressora é relativamente pequeno, o processo fica preso até o final da impressão. Observe no diagrama da Figura 1.2 que o disco e a impressora nunca trabalham simultaneamente, embora não exista nenhuma limitação de natureza eletrônica. O programa seqüencial é que não consegue ocupar ambos.

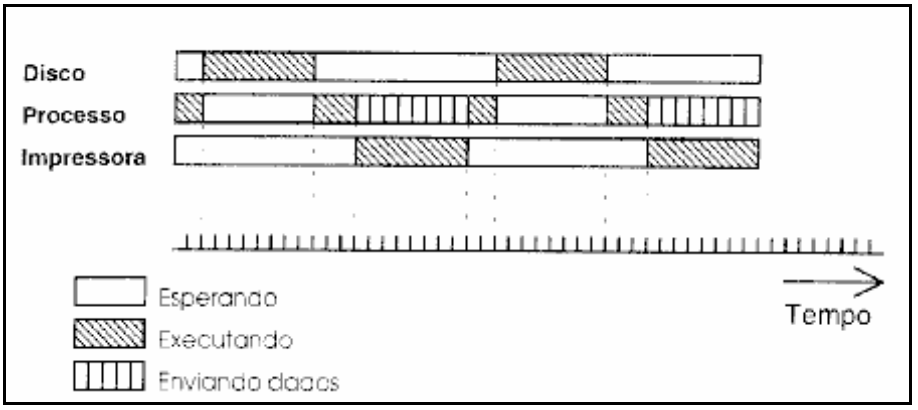


Figura 1.2 Linha de tempo do programa seqüencial da Figura 1.1

Vamos agora empregar uni programa concorrente como o mostrado na Figura 1.3 para realizar a impressão do arquivo. Dois processos dividem o trabalho. O processo leitor é responsável por ler registros do arquivo, formatar e colocar em um *buffer* na memória. O processo impressor retira os dados do *buffer* e envia para a impressora. É suposto aqui que os dois processos possuem acesso à memória onde está o *buffer*. A Figura 1.4 mostra a linha de tempo resultante.

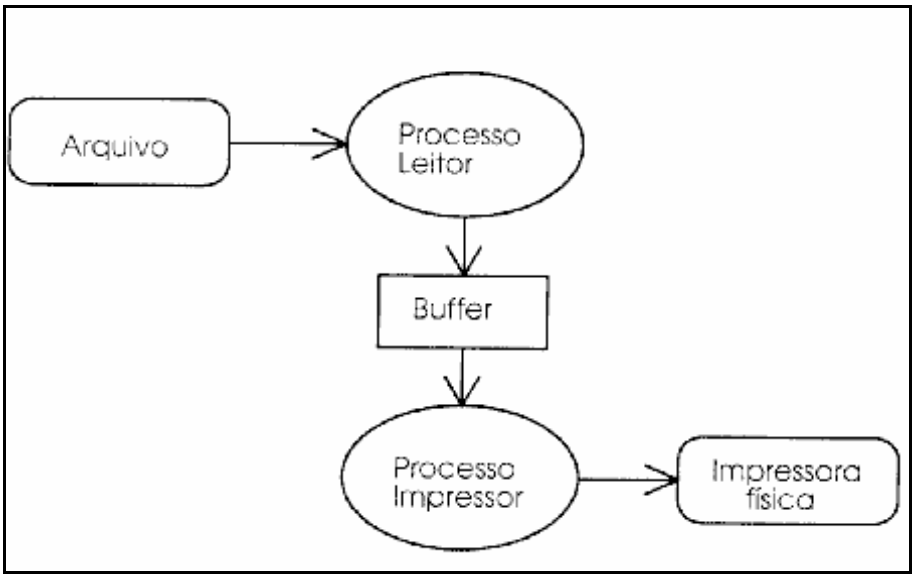


Figura 1.3 Programa concorrente fazendo acesso a arquivo e impressora

O programa concorrente é mais eficiente, pois consegue manter o disco e a impressora trabalhando simultaneamente. O tempo total para realizar a impressão do arquivo é menor quando a solução concorrente é empregada. É claro que a solução possui limitações. Se o processo leitor for sempre mais rápido, o *buffer* ficará cheio, e então o processo leitor terá que esperar até que o processo impressor retire algo do *buffer*. Por outro lado, se o processo impressor for sempre mais rápido, eventualmente o *buffer* ficará vazio, e ele terá que esperar pelo processo leitor. É isso que acontece no diagrama da Figura 1.4. De qualquer forma, a solução concorrente para esse problema nunca será mais lenta que a solução seqüencial.

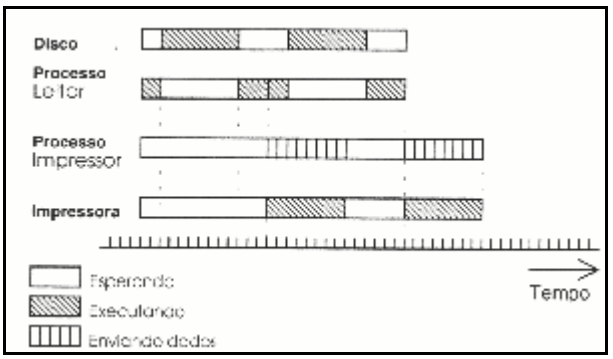


Figura 1.4 Linha de tempo do programa concorrente da Figura 1.3

Na verdade, a maior motivação para a programação concorrente é a engenharia de software. Aplicações inerentemente paralelas (aplicações que possuem paralelismo intrínseco) são mais facilmente construídas se a programação concorrente é utilizada. Estão enquadrados nesse grupo aplicações envolvendo protocolos de comunicação, aplicações de supervisão e controle industrial e, como era de se esperar, a própria construção de um sistema operacional.

Vamos ilustrar esse aspecto através de uma aplicação hipotética, um servidor de impressão para uma rede local. A Figura 1.5 ilustra uma rede local na qual existem diversos computadores pessoais (PC) utilizados pelos usuários e existe um computador dedicado ao papel de servidor de impressão da rede. Quando algum usuário deseja fazer uma impressão, ele envia o arquivo para o servidor de impressão. Como todos os usuários da rede compartilham a mesma impressora, ela possivelmente estará ocupada no momento que um arquivo for enviado. Por isso, o servidor de impressão usa um disco magnético para manter os arquivos que estão na fila de impressão, ou seja, esperando para serem impressos.

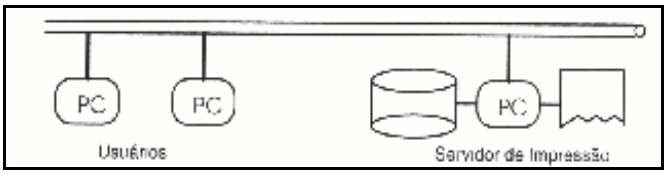


Figura 1.5 Rede local incluindo um servidor de impressão dedicado

E importante observar que o programa "servidor de impressão" possui um certo grau de paralelismo intrínseco. Ele deve receber mensagens pela rede; escrever em disco os pedaços de arquivos recebidos; enviar mensagens pela rede contendo, por exemplo, respostas às consultas sobre o seu estado; ler arquivos previamente recebidos; enviar dados para a impressora. Todas essas atividades devem ser realizadas simultaneamente. Uma forma clara de programarmos o servidor de impressão é usar vários processos de tal forma que cada processo fique responsável por uma atividade em particular. Obviamente, esses processos vão precisar trocar informações para realizar o seu trabalho. Logo, teremos o servidor de impressão implementado na forma de um programa concorrente.

A Figura 1.6 mostra uma das possíveis soluções para a organização interna do programa concorrente "servidor de impressão". Cada círculo representa um processo. Cada flecha representa a passagem de dados de um processo para o outro. Essa passagem de dados pode ser feita, por exemplo, através de variáveis que são compartilhadas pelos processos envolvidos na comunicação. Vamos agora descrever rapidamente a função de cada processo. O processo “Receptor” é responsável por receber mensagens da rede local. Ele faz isso através de chamadas de sistema apropriadas e descarta as mensagens com

erro. As mensagens corretas são então passadas para o processo "Protocolo". Ele analisa o conteúdo das mensagens recebidas à luz do protocolo de comunicação suportado pelo servidor de impressão. É possível que sejam necessários a geração e o envio de mensagens de resposta. O processo “Protocolo” gera as mensagens a serem enviadas e passa-as para o processo “Transmissor” que as envia através de chamadas de sistema apropriadas.

Algumas mensagens contêm pedaços de arquivos a serem impressos. É suposto aqui que mensagens são da ordem de alguns Kbytes. Dessa forma, um arquivo deve ser dividido em várias mensagens para transmissão através da rede. Quando o processo “Protocolo” identifica uma mensagem que contém um pedaço de arquivo, ele passa esse pedaço de arquivo para o processo “Escritor”. Passa também a identificação do arquivo ao qual o pedaço em questão pertence. Cabe ao processo “Escritor” usar as chamadas de sistema apropriadas para escrever no disco. Quando o pedaço de arquivo em questão é o ultimo de seu arquivo, o processo “Escritor” passa para o processo “Leitor” o nome do arquivo que está pronto para ser impresso.

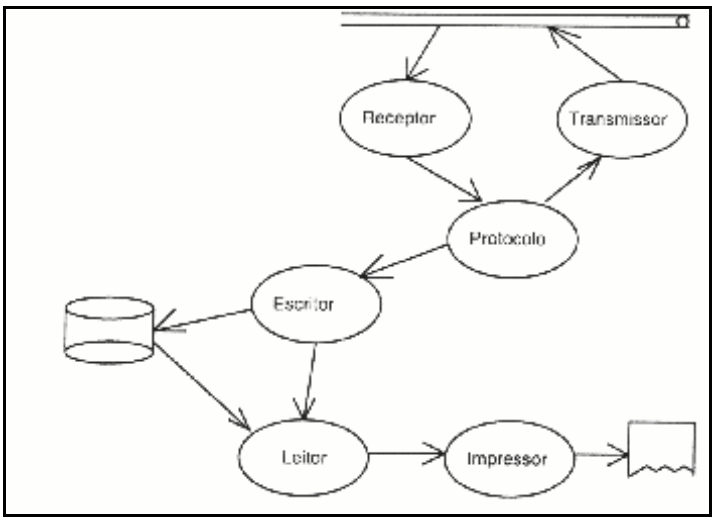


Figura 1.6 Servidor de impressão como programa concorrente

O processo “Leitor” executa um laço externo no qual ele pega um nome de arquivo, envia o conteúdo para o processo “Impressor” e então remove o arquivo lido. O envio do conteúdo para o processo “Impressor” é feito através de um laço interno composto pela leitura de uma parte do arquivo e pelo envio dessa parte. Finalmente, o processo “Impressor” é encarregado de enviar os pedaços de arquivo que ele recebe para a impressora. O relacionamento entre os processos “Leitor” e “Escritor” foi descrito antes, no início desta seção.

Embora o servidor de impressão descrito aqui tenha sido simplificado em vários aspectos, ele ilustra o emprego da programação concorrente na construção de aplicações com paralelismo intrínseco. O resultado é uma organização interna clara e simples para o programa. Além disso, um programa seqüencial equivalente seria provavelmente menos eficiente.

1.4 Especificação do Paralelismo

Para construir um programa concorrente, é necessário, antes de mais nada, ter a capacidade de **especificar o paralelismo** dentro do programa. Em algum ponto, é necessário especificar quantos processos farão parte do programa e exatamente quais rotinas cada um executará. Existem, na prática, diversas maneiras para realizar essa

tarefa. Nesta seção, serão descritas duas das mais usuais.

É possível especificar paralelismo através do conjunto de comandos: *Fork*, *Exit* e *Wait*. **O** comando *Fork* permite a criação de um segundo fluxo de execução, paralelo àquele que executou o comando. Em geral, é necessário fornecer uma indicação de onde o novo fluxo de execução deverá iniciar. Por exemplo, o nome de uma sub-rotina do programa.

Os comandos *Exit* e *Wait* são auxiliares ao *Fork*. Quando o comando *Exit* é executado, o fluxo de controle que o executa é imediatamente terminado. Um comando imediatamente após o comando *Exit* jamais será executado. O comando *Wait* permite que um fluxo de execução espere outro fluxo terminar. Em geral, é necessário fornecer uma indicação do fluxo de execução cujo término está sendo esperado. Por exemplo, um número inteiro retornado pelo comando *Fork*.

Vamos usar a sintaxe da linguagem C para exemplificar o uso dos comandos *Fork*, *Exit* e *Wait.* Considere o trecho de código mostrado na Figura 1.7. Nesse programa, o processo inicial (pai) executa duas vezes o comando *Fork*, criando dois processos adicionais (filhos 1 e 2). Ele então espera que cada um dos filhos termine, escreve uma mensagem para cada um e termina também. Os dois processos criados executam a mesma função “código-do-filho”. Eles apenas colocam uma mensagem na tela e terminam. Uma possível saída para esse programa é:

```
Alo do pai
Alo do filho
Alo do filho
Filho 1 morreu
Filho 2 morreu
```

A Figura 1.8 mostra um diagrama de tempo capaz de gerar a saída mostrada acima. É importante observar que processos paralelos podem executar em qualquer ordem. Na saída mostrada antes, o segundo filho foi executado antes que o pai percebesse a morte do primeiro filho. Caso o segundo filho tivesse executado após a morte do primeiro filho, a saída seria:

```
Alo do pai
Alo do filho
Filho 1 morreu
Alo do filho
Filho 2 morreu
```

```
/* Programa principal */
        main()
{
        int f1;/* Identifica processo filho 1 */
        int f2;/* Identifica processo filho 2 */

        printf(“Alo do pai\n”);

        f1 = fork( codigo-do-filho );        /* Cria filho 1 */
        f2 = fork( codigo-do-filho );        /* Cria filho 2 */

        wait(fl);
        printf("Filho 1 morreu\n”>;

        wait(f2);
        printf("Filho 2 morreu\n");
```

```
        exit ();
}

/* Funcao executada pelos dois processos filhos */
código-do-filho()
{
        printf("Alo do filho\n");
        exit ();
}
```

Figura 1.7 Exemplo contendo os comandos *Fork*, *Exit* e *Wait*

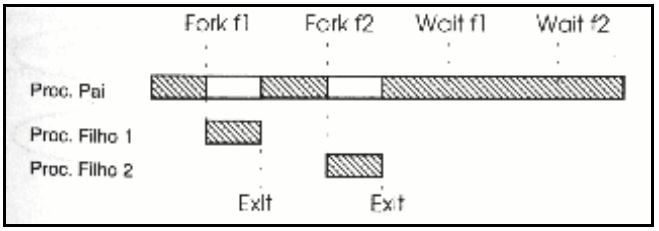


Figura 1.8 Diagrama de tempo associado com o programa da Figura 1.7

A Figura 1.9 mostra o diagrama de tempo associado com essa segunda possibilidade. Essa é uma característica importante dos programas concorrentes. Duas execuções consecutivas do mesmo programa, com os mesmos dados de entrada, podem gerar resultados diferentes. Isso não é necessariamente um erro. Cabe ao programador fazer com que todos os resultados possíveis sejam igualmente corretos. Mais adiante serão apresentados mecanismos capazes de controlar explicitamente a ordem de execução dos processos, se assim for desejado.

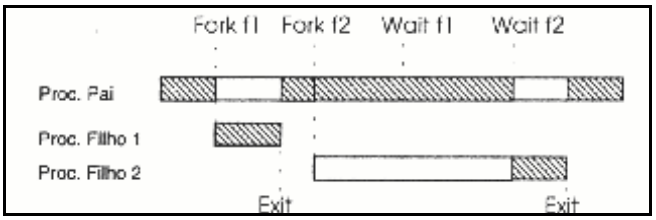


Figura 1.9 Diagrama de tempo associado com o programa da Figura 1.7

Muitas vezes, são usados grafos de precedência para representar o paralelismo existente em um programa. No grafo de precedência, os nodos representam trechos de código, e os arcos representam relações de precedência. Um arco do nodo X para o nodo Y representa que o código associado com Y somente poderá ser executado após o término do código associado com X. Por exemplo, o programa da Figura 1.7 pode ser representado pelo grafo que aparece na Figura 1.10. É importante notar que, nas Figuras anteriores, os arcos representavam o fluxo dos dados, enquanto na Figura 1.10 (um grafo de precedência), os arcos não representam fluxo de dados, mas sim o fluxo de controle.

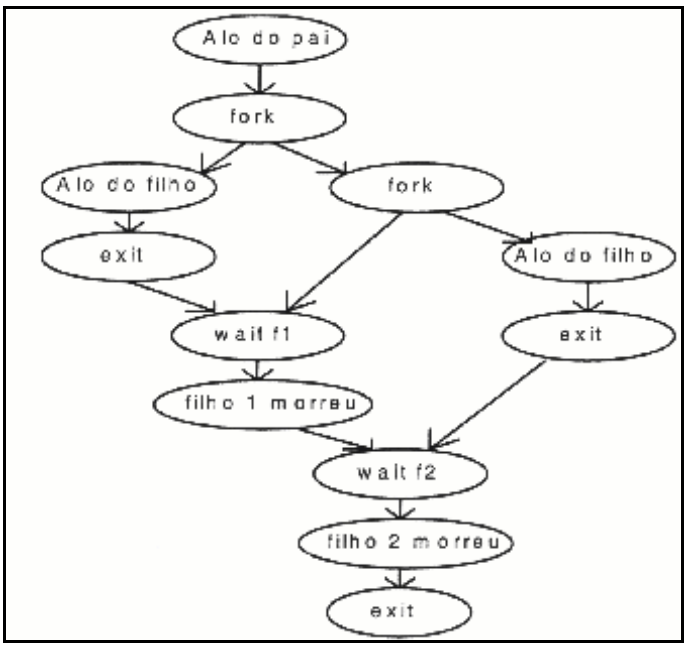


Figura 1.10 Grafo de precedência para o programa da Figura 1.7

Vamos supor agora que o programador deseje que, necessariamente, o processo do filho 1 termine para então o processo do filho 2 iniciar. A Figura 1.11 mostra como ficaria o código nesse caso.

```
/* Programa principal */
main()
{
        int f1;/* Identifica filho 1 */
        int f2;/* Identifica filho 2 */

        printf("Alo do pai\n");

        f1 = fork( codigo_do_filho );        /* cria filho 1 */
        wait( f1);
        printf("Filho 1 morreu\n");

        f2 = fork( codigo_do_filho );        /* cria filho 2 */
        wait( f2);
        printf("Filho 2 morreu\n");

        exit();
}

/* Função executada pelos dois processos filhos */
codigo_do_filho()
{
        print("Alo do filho\n");
        exit();
}
```

Figura 1.11 Programa do exemplo da Figura 1.7 alterado

Uma forma mais estruturada de especificar paralelismo em programas concorrentes é conseguida com o uso dos comandos *Parbegin* e *Parend* (*"parallel begin"* e *"parallel end"*). Enquanto o par *Begin-End* delimita um conjunto de comandos que serão executados seqüencialmente, o par *Parbegin-parend* delimita um conjunto de comandos que serão executados em paralelo. O comando que segue ao *Parend* somente será executado quando todos os fluxos de controle criados na execução do *Parbegin-Parend* tiverem terminado. A Figura 1.12 mostra como esses comandos podem ser utilizados para implementar um programa equivalente àquele apresentado na Figura 1.7.

Novamente, a linguagem C é utilizada.

```
/* Programa principal */
main();
{
        printf("Alo do pai\n")

        Parbegin       /* Define conjunto de execuções paralelas */
                codigo_do_filho();     /+ Cria um filho */
                codigo_do_filho();     /* Cria outro filho */
        Parend;

        printf("Filho 1 morreu\n");
        printf("Filho 2 morreu\n");
}

/* Função executada pelos dois processos filhos */
codigo_do_filho()
{
        printf("Alo do filho\n")
{
```

Figura 1.12 Exemplo contendo os comandos *Parbegin* e *Parend*

Os comandos *Fork*, *Wait* e *Exit* são facilmente incorporados a uma linguagem de programação procedimental. Nos exemplos mostrados antes, eles foram incorporados à linguagem C na forma de funções de uma biblioteca especial. Já os comandos *parbegin-parend* definem uma nova estrutura de controle para a linguagem. Nos exemplos, a sintaxe da linguagem C foi estendida com a inclusão desse novo comando. Também é possível usar uma versão menos elegante para o comando *Parbegin* que, entretanto, mantém sua funcionalidade. Dessa vez, *Parbegin* aparece como uma função de biblioteca cujos parâmetros são os nomes das funções a serem disparadas em paralelo. O exemplo da Figura 1.12 seria adaptado para conter:

```
Parbegin( codigo_do_fi1ho, codigo_do_fi1ho);
```

que substitui a estrutura de controle *Parbegin-Parend*. De qualquer forma, fica evidente que não é difícil adaptar as linguagens de programação para, de alguma forma, permitir a especificação de paralelismo dentro dos programas.

Vamos agora considerar como seria a implementação dos comandos apresentados antes. O comando *Fork* representa a criação de um novo processo. A gerência do processador inicializa as estruturas de dados necessárias e insere o novo processo na fila do processador. O espaço de endereçamento do novo processo deve ser igual ao espaço de endereçamento do processo que executou o comando *Fork*. O valor inicial para o registrador PC do novo processo deve ser o endereço fornecido como parâmetro para o comando *Fork*. O valor retornado pelo *Fork* pode ser o identificador do novo processo.

O comando *Wait k* resulta no bloqueio do processo chamador caso o processo identificado por *k* ainda exista. Nesse, caso é necessário criar uma fila de processos bloqueados à espera de que o processo *k* termine. Essa fila será implementada como uma lista encadeada, associada ao próprio processo *k*. Dessa forma, quando ele terminar, parte do processo de limpeza interna do sistema operacional será liberar todos os processos que esperavam pela sua morte. Quando o processo *k* termina, é necessário armazenar essa informação. Se, depois disso, algum processo executar o comando *Wait k*, ele não ficará bloqueado e continuará sua execução.

O comando *Exit* é implementado por uma chamada de sistema que resulta na destruição do processo chamador. É necessário verificar se não existe nenhum processo bloqueado por *Wait* à espera da morte desse processo. Nesse caso, é necessário retirar da situação de bloqueio o processo que estava esperando e colocá-lo novamente na disputa pelo

processador.

Os comandos *Parbegin* e *Parend* podem ser implementados de forma semelhante. Uma possibilidade é o sistema operacional oferecer chamadas de sistema *Fork*, *Wait* e *Exit*, e o compilador da linguagem de programação traduzir *Parbegin-Parend* para uma seqüência dessas chamadas. Nesse caso, o trecho em linguagem de programação:

```
Inicio ()
Parbegin
Comando_l();
Comando_2();
Parend;
Fim();
```

seria traduzido pelo compilador para:

```
Inicio()
f1 = Fork( Comando_l);
f2 = Fork( Comando_2);
Wait( f1);
Wait( f2);
Fim();
```

A especificação de paralelismo é um aspecto importante da programação concorrente, mas não é o único. Nas próximas seções, serão discutidos e comparados aspectos ligados à troca de informação entre processos e será feita uma descrição do problema do *deadlock*.

Em princípio, devemos supor que um sistema distribuído deve parecer a seus usuários exatamente como se fosse um sistema tradicional, com um único processador usado em regime de tempo compartilhado. O que acontece se um programador descobrir que seu sistema distribuído possui 1000 processadores, e quiser usar uma parte substancial de tais processadores na análise paralela de uma jogada de xadrez? A resposta teórica a essa questão é que o compilador, o sistema operacional e o ambiente de execução devem ser capazes de, em conjunto, mostrar como se pode tirar vantagem desse paralelismo potencial sem que o programador nem mesmo tome conhecimento de sua existência. Os programadores que realmente desejarem usar vários processadores na solução de um único problema devem manifestar esse desejo de forma explícita em seu programa. A transparência ao paralelismo deve ser encarada como o supremo objetivo de todos os projetistas de sistemas distribuídos. Quando isso for obtido, o trabalho estará terminado, e poderemos partir para novos desafios em outros campos.

1.5 Visão Geral e Comparação

Esta seção contém comentários sobre algumas técnicas para a comunicação entre processos. Como visto no início deste capítulo, um programa concorrente é formado por processos que necessitam trocar informações. Essa troca de informações pode ocorrer através de variáveis compartilhadas ou de mensagens.

Nas soluções baseadas em variáveis compartilhadas, o sistema operacional oferece algum mecanismo de sincronização auxiliar, como semáforos. Também é necessário que a gerência de memória permita o acesso de dois processos a uma mesma memória física. Quando mensagens são empregadas, o sistema operacional deve implementar algum mecanismo de troca de mensagens para ser usado pelos processos.

Essas duas soluções (memória compartilhada e mensagens) são equivalentes no sentido de que qualquer programa concorrente pode ser implementado com uma ou com outra forma. Em geral, memória compartilhada é mais eficiente que troca de mensagens. Com memória compartilhada, não existe a necessidade de copiar os dados da memória do remetente para uma área do sistema operacional e depois para a memória do destinatário. Os processos alteram diretamente as variáveis compartilhadas. Entretanto, em sistemas distribuídos, mensagens são a forma natural de comunicação.

Muitos sistemas operacionais oferecem os dois mecanismos. Nesses sistemas, processos na mesma máquina podem compartilhar memória. A forma como a memória compartilhada é implementada depende da gerência de memória empregada, sendo relativamente simples em sistemas que contam com paginação ou segmentação. Processos também podem usar mensagens para comunicação local ou remota. A implementação de um sistema de mensagens sem buferização pode ser feita através da manipulação dos descritores de processos. Um sistema de mensagens com buferização implica a reserva de memória para armazenar mensagens e algoritmos para gerenciar essa memória. Além disso, é necessário liberar automaticamente áreas de memória ocupadas por mensagens cujos processos destinatários já foram destruídos. Finalmente, a implementação de mensagens entre máquinas diferentes exige a existência de um subsistema de comunicação que implemente os protocolos de comunicação necessários.

1.6 Deadlock

Embora ***deadlocks*** possam ocorrer em diversos pontos de um sistema operacional, eles são um dos principais problemas dos programas concorrentes e, por isso, serão discutidos nesta seção. Também é comum a ocorrência de *deadlocks* envolvendo arquivos e periféricos.

Por definição, um conjunto de *N* processos está em *deadlock* quando cada um dos *N* processos está bloqueado à espera de um evento que somente pode ser causado por um dos *N* processos do conjunto. Obviamente, essa situação somente pode ser alterada por alguma iniciativa que parta de um processo fora do conjunto dos *N* processos.

A Figura 1.13 ilustra graficamente uma situação de *deadlock*. Círculos representam processos, e quadrados representam recursos. Uma flecha do processo para o recurso significa que o processo está bloqueado à espera daquele recurso. Uma flecha do recurso para o processo significa que aquele recurso foi alocado ao processo.

No exemplo da Figura 1.13, o processo P5 está bloqueado à espera do recurso R3. Entretanto, ele não está em *deadlock*, pois o processo P2 não está bloqueado. Ele pode executar até o fim, liberar o recurso R3, e então o processo P5 poderá executar. O mesmo não ocorre com os processos P1, P3 e P4. O processo P4 está bloqueado à espera do recurso R2, ocupado pelo processo P1. Os processos P1 e P3 estão bloqueados à espera do recurso R1, ocupado pelo processo P4. Logo, P1, P3 e P4 estão em *deadlock*.

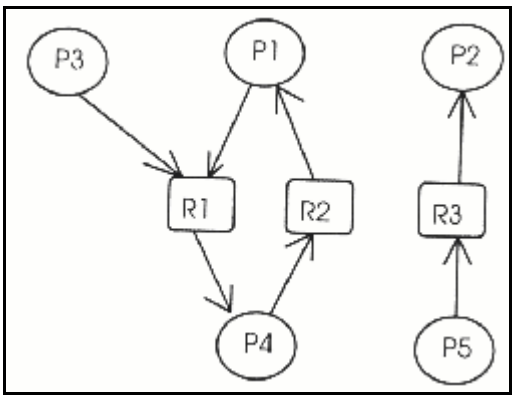


Figura 1.13 Gráfico representando processos e recursos

Existem quatro condições necessárias para ser possível a ocorrência de um *deadlock* em determinado sistema. São elas:

- Existência de recursos que precisam ser acessados de forma exclusiva;
- Possibilidade de processos manterem recursos alocados enquanto esperam por recursos adicionais;
- Necessidade de os recursos serem liberados pelos próprios processos que os estão utilizando;
- Possibilidade da formação de uma espera circular do tipo: o processo P_1 espera pelo recurso R_1 que está com o processo P_2 que espera pelo recurso R_2 que está com o processo P3, etc, e P_N espera pelo recurso R_N que está com o processo P_1.

Existem várias formas de tratar o problema do *deadlock*. Por exemplo, é impossível a ocorrência de um *deadlock* no sistema se, para cada recurso, for eliminada pelo menos uma das quatro condições necessárias. Infelizmente, isso não é facilmente realizável. Também é possível evitar a ocorrência de um *deadlock* se, para cada pedido de alocação de recursos, o sistema analisar as implicações de atendê-lo. Essa técnica implica subutilização dos recursos do sistema e *overhead* de execução. A teoria sobre *deadlocks* é extensa e foge ao escopo deste livro.

Uma forma usual é deixar acontecer o *deadlock*, detectá-lo e então eliminá-lo. A detecção pode ser feita de forma automática ou manual. O *deadlock* é eliminado através da destruição dos processos envolvidos e da liberação dos respectivos recursos. A quase totalidade dos sistemas operacionais utiliza essa abordagem, o que na prática significa que nenhuma técnica especial com respeito aos *deadlocks* é utilizada. O mesmo não acontece nos sistemas de banco de dados, nos quais cuidados especiais são tomados a esse respeito.

1.7 Exercícios

⇨ Desenhe o grafo de precedência relativo ao código das Figuras 1.11 e 1.12.

⇨ Pesquise a literatura a respeito dos mecanismos *mutex* e *variáveis condição* do POSIX. Implemente as operações *P* e *V* em um semáforo usando aquelas duas construções básicas do POSIX. As operações *P* e *V* deverão ser substituídas por um código C com semântica similar. Lembre-se de que essas operações devem ser atômicas.

⇨ Localize, na literatura sobre programação concorrente, a descrição do *problema dos filósofos jantadores* e implemente uma solução usando semáforos. Faça o mesmo com o *problema dos leitores e escritores*.

⇨ Imagine formas de tornar impossível alguma das quatro condições necessárias para a ocorrência de um *deadlock* em determinado sistema. Essa forma deve ser tal que, uma vez aplicada, *deadlocks* não acontecerão. Discuta a viabilidade prática de sua solução.

2 Teoria das categorias

2.1 Introdução à Teoria das Categorias

Aparentemente, *Teoria das Categorias* e *Ciência da Computação* constituem dois campos científicos completamente diferentes. Na realidade, não só possuem muito em comum, como são enriquecidos mutuamente a partir de visões e abordagens de um campo sobre o outro.

Em um primeiro momento, a Teoria das Categorias pode ser vista como uma generalização da álgebra de funções. Nesse contexto, claramente, a principal operação sobre funções é a de composição. Na realidade, uma *Categoria* é uma estrutura abstrata, constituída de *objetos* e *setas* entre os objetos, com uma propriedade fundamental que é a *composicionalidade das setas*. Por exemplo, objetos e setas podem ser conjuntos e funções. Alguns exemplos de categorias são ilustrados na Tabela 2.1.

Tabela 2.1 Exemplos de Categorias

Categoria	Objetos	Setas	Composição
Conjuntos e Funções	conjuntos	funções (totais)	composição de funções
Figuras	Figuras	transformações de Figuras	construtor de transformações "complexas"
Um Programa Funcional	tipos de dados	operações	construtor de operações não-primitivas
Espaços Vetoriais	espaços vetoriais	transformações lineares	composição de transformações lineares
Grafos	grafos	homomorfismo de grafos	composição de homomorfismo de grafos
Lógica	proposições	provas	transitividade das provas
Uma Máquina de Estados	estados	transições	construtor de computações
Conjuntos Parcialmente Ordenados	conjuntos parcialmente ordenados	funções monotônicas	composição de funções monotônicas
Um Conjunto Parcialmente Ordenado	elementos do conjunto	pares da relação de ordem parcial	transitividade da relação de ordem parcial

Muitas áreas da matemática podem ser vistas sob o ponto de vista das categorias. Na Ciência da Computação, existe interesse pela *Lógica Categorial*, que pode ser vista a partir do seguinte exemplo:

Pode-se chamar os objetos em uma determinada categoria de FÓRMULAS e os morfismos de PROVAS. Um morfismo f : A-->B é visto como uma prova da implicação lógica A=>B. Em particular, o morfismo identidade é uma instância do axioma da reflexividade, e a composição de morfismos abaixo é uma regra de inferência que afirma a transitividade da implicação lógica.

$$\frac{f : A \rightarrow B \qquad g : B \rightarrow C}{gof : A \rightarrow C}$$

Em uma categoria, qualquer modificação sobre os objetos, as setas ou a composição resulta em uma nova categoria. Por exemplo, na primeira categoria da Tabela 2.1, a substituição das funções (totais) por funções parciais resulta em uma nova categoria (conjuntos e funções parciais), com diferentes propriedades. Como ilustração, nesse caso, um simples produto cartesiano (generalizado categorialmente) de dois conjuntos é diferente nas duas categorias.

Assim, as três componentes básicas de uma categoria (objetos, setas e composição) são fundamentais. Em especial, a composição freqüentemente possui importantes interpretações como, por exemplo, nas seguintes categorias da Tabela 2.1.

⇨ um programa funcional visto como uma categoria: a operação de composição é vista como um construtor de operações a partir de operações mais elementares;

⇨ uma máquina de estados vista como urna categoria: a operação de composição define a noção de computação (ou transação) da máquina, refletindo o funcionamento da mesma (ou seja, fornecendo uma semântica).

É importante observar que as noções de objeto, seta e composição não necessariamente possuem estruturas que lembrem as usadas na Teoria dos Conjuntos. Por exemplo, no caso de um conjunto parcialmente ordenado visto como uma categoria, os objetos não possuem qualquer estrutura (um objeto é um elemento de um conjunto), as setas são pares de elementos, e a composição é dada pela transitividade da relação.

Adicionalmente, uma mesma estrutura pode constituir uma categoria por si só ou ser objeto de uma categoria como nos dois casos referentes aos conjuntos parcialmente ordenados ilustrados na Tabela 2.1. Assim a categoria dos conjuntos parcialmente ordenados pode ser vista como uma categoria de categorias (ou seja, uma categoria cujos objetos são conjuntos parcialmente ordenados vistos como categorias).

Teoria das Categorias é uma teoria recente, tendo sido criada por S. Eilenberg e S. Mac Lane em 1945 [MEN2001], como uma decorrência de seus trabalhos em topologia algébrica. Desde então, tem influenciado muitas áreas, como uma forma revolucionária de entendimento e de abordagem. Atualmente, Teoria das Categorias e Ciência da Computação constituem uma área de pesquisa extremamente ativa internacionalmente.

Entre as diversas características da Teoria das Categorias que motivam o seu uso em Ciência da Computação, destacam-se:

1 Independência de Implementação;

2 Dualidade;

3 Herança de Resultados;

4 Comparação de Expressividade de Formalismos;

5 Notação Gráfica;

6 Expressividade de suas Construções.

A expressividade das construções categoriais tem sido uma das principais motivações (senão a principal) para o uso da Teoria das Categorias na Ciência da Computação. Considerando-se a complexidade dos sistemas computacionais atuais, verifica-se que, de certa forma, o desenvolvimento de soluções para os problemas propostos está limitado à capacidade do ser humano de expressar os problemas e suas soluções. Assim, quanto mais expressivo for o formalismo usado, mais avanços podem ser esperados. Inclusive, formalismos mais expressivos auxiliam não só nas especificações e provas, mas principalmente, em um melhor entendimento dos problemas, bem como em uma

maior simplicidade e clareza nas soluções.

Como exemplos da grande influência dessa Teoria em muitos campos da Ciência da Computação, pode-se citar modelos semânticos de linguagens de programação, modelos de concorrência, linguagens de especificação, lógica construtiva, teoria dos autômatos, processamento paralelo e distribuído e várias outras áreas.

Este capítulo apresenta conceitos básicos da Teoria das Categorias tais como Categoria, Limite e Colimite em geral, Produto e Coproduto em particular. O objetivo é apresentar essa teoria como um formalismo capaz de expressar concorrência e auxiliar a especificação formal de sistemas concorrentes. Para tanto, são apresentadas e utilizadas as estruturas de conjunto parcialmente ordenado e de grafos reflexivos vistos como categorias para formalizar as definições de tais aplicações.

2.2 Conceito de Categoria

Categorias são estruturas matemáticas abstratas que servem para representar classes de objetos que possuem algumas propriedades básicas em comum, generalizando-as e unificando-as em uma propriedade universal que serve para todos os objetos de uma mesma categoria. Por exemplo, objetos e setas e composição de setas podem ser conjuntos, funções e composição de funções, respectivamente. Essa abstração possui vantagens como o fato de permitir que propriedades comuns sejam facilmente identificáveis.

Portanto, Teoria das Categorias pode ser considerada como uma formalização adequada para tratar propriedades abstratas independentes de estruturas. Essa é mais uma característica relevante relativamente à Ciência da Computação: permite tratar propriedades "independentemente de implementação", o que é essencial para a análise de algoritmos quanto a suas dependências de controle e de dados.

Existem dois conceitos básicos em categorias: *objeto* e *morfismo*. *Objetos não necessariamente* são coleções de elementos, enquanto *morfismos* (ou *setas*) *não necessariamente* são funções entre conjuntos. Dessa forma, morfismos podem ser compostos com morfismos. Também é importante afirmar que, na Teoria das Categorias, são os morfismos, e não os objetos, que desempenham o papel principal.

Definição 1 – *Categoria* - Uma categoria C é uma seis-upla: $C = \langle Ob_C, Mor_C, \partial_0, \partial_1, \iota, o?$ onde:

a) Ob_C é uma coleção de Objetos também denotada por C_0;

b) Mor_C é uma coleção de morfismos ou setas também denotada por C_1;

c) $\partial_0,\partial_1: Mor_C \rightarrow Ob_C$ são operações denominadas Domínio ou Origem e Codomínio ou Destino, respectivamente. Um morfismo f tal que $\partial_0(f) = A$ e $\partial_1(f) = B$ é usualmente denotado por: $f: A \rightarrow B$;

d) $?: (Mor_C)^2 \rightarrow Mor_C$ é uma operação parcial denominada Composição tal que cada par de morfismos: $\langle f: A \rightarrow B, g: B \rightarrow C \rangle$ é associado a um morfismo: $g \circ f: A \rightarrow C$;

A operação de composição deve satisfazer à propriedade associativa, segundo a qual, para quaisquer morfismos $f: A \rightarrow B$, $g: B \rightarrow C$ e $h: C \rightarrow D$ em Mor_C, tem-se que: $(h \circ g) \circ f = h \circ (g \circ f)$;

e) $\iota: Ob_C \rightarrow Mor_C$ é uma operação denominada *Identidade* tal que cada objeto A é associado a um morfismo: $\iota_A: A \rightarrow A$.

A operação identidade deve satisfazer à propriedade da identidade, segundo a qual, para qualquer morfismo $f: A \rightarrow B$ em Mor_C, tem-se que: $f \circ \iota_A = \iota_B \circ f = f$.

Dualidade

Relativamente à Teoria das Categorias, é comum encontrar a seguinte afirmação: "*a*

noção de dualidade divide o trabalho pela metade". O dual de uma categoria é, basicamente, a inversão do sentido das suas setas.

Definição 2 – Categoria Dual - Seja C = ⟨Ob_C, Mor_C, ∂_0, ∂_1, ι, o? uma categoria. A correspondente Categoria Dual C^{op} é: C^{op} = ⟨Ob_C, Mor_C, ∂_1, ∂_0, ι, o^{op}? tal que:

a) As coleções de objetos e morfismos e a operação identidade são as mesmas em C e em C^{op};

b) Se ∂_0 e ∂_1 são as operações de origem e destino em C, respectivamente, então são destino e origem em C^{op}, respectivamente (atenção para a ordem invertida);

c) Se, em C, tem-se que h = g o f, então, em C^{op}, tem-se que h = f o^{op} g.

Resumidamente, a categoria dual consiste basicamente na troca da operação de origem pela operação destino e vice-versa. Verifica-se, pela definição acima, que o dual de uma categoria também é uma categoria.

Conjuntos Parcialmente Ordenados

Um conjunto parcialmente ordenado (A, ≤) é tal que A é um conjunto e ≤ é uma relação de ordem parcial (reflexiva, anti-simétrica e transitiva). Conjuntos parcialmente ordenados ou estruturas baseadas neles são usados com freqüência em Ciência da Computação. São importantes na semântica para *sistemas concorrentes*. De fato, fornecem uma clara e simples visão de concorrência, no sentido verdadeiro da palavra, usualmente denominada *concorrência verdadeira*. É importante lembrar que a definição adotada para *concorrência*, neste trabalho, é a seguinte, resumidamente: duas ações *a* e *b* são ditas concorrentes se a ordem de ocorrência delas entre si é irrelevante. Ou, seja, existem caminhos alternativos que refletem as possíveis serializações de *a* e *b* (*a* seguida de *b* ou *b* seguida de *a*).

Conjunto Parcialmente Ordenado Visto como uma Categoria

Exemplo 1 - *Conjunto Parcialmente Ordenado visto como uma Categoria*

a) Considere o conjunto parcialmente ordenado ⟨{0, 1, 2}, ≤? . De fato, ⟨{0, 1, 2}, ≤? pode ser visto como uma categoria na qual os objetos são {0, 1, 2}, e os morfismos são os pares da relação como ilustrado na Figura 2.1.

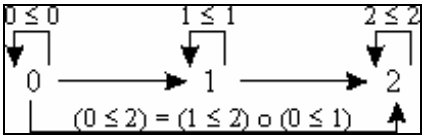


Figura 2.1 Conjunto parcialmente ordenado visto como uma categoria

b) Considere ⟨{c1, c2, c3, p1, p2}, ≤? onde ≤ é uma relação de ordem tal que c1 ≤ c2 ≤ c3, p1 ≤ p2 e c2 ≤ p2. Analogamente, tal conjunto parcialmente ordenado pode ser visto como uma categoria na qual os objetos são {c1, c2, c3, p1, p2}, e os morfismos são os pares da relação como ilustrado na Figura 2.2.

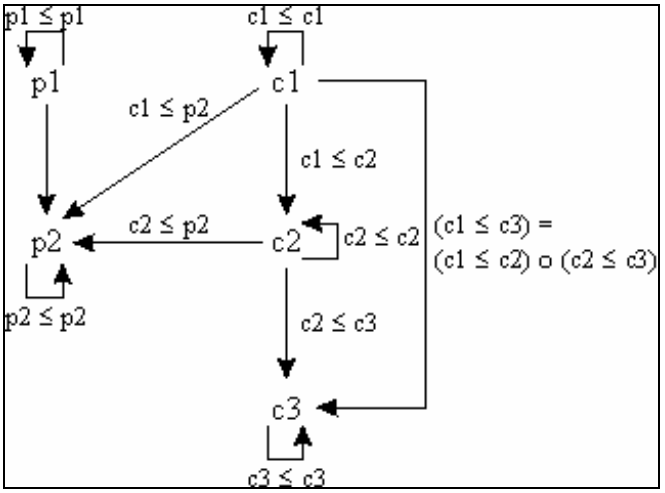


Figura 2.2 Conjunto parcialmente ordenado visto como uma categoria

Neste artigo, serão apresentados exemplos de limites e colimites e de seus casos especiais. Como exemplos de casos especiais de cones, podemos citar limites, produtos e objetos iniciais, e de colimites, temos cocones, coprodutos e objetos terminais.

Limite e Colimite

Os conceitos de limites e colimites são definidos sobre *diagramas*, *cones* e *cocones* .

Intuitivamente, para uma categoria C, um *diagrama* em C é uma multicoleção (coleção na qual podem existir elementos repetidos) de objetos e de morfismos de C tal que, para cada morfismo do diagrama, os seus correspondentes objetos origem e destino são elementos do diagrama.

Um diagrama é *comutativo* em C se, para quaisquer dois objetos, os diversos caminhos alternativos resultantes da composição das setas componentes são iguais.

Definição 3 – *Cone* - Sejam C uma categoria e D um diagrama em C. Um *cone* do diagrama D é um objeto C, juntamente com uma coleção de morfismos c_i: C→Ai, para todo objeto Aj de D, tais que, para todos os objetos A_u e A_v e todos os morfismos d : $A_u \rightarrow A_v$ do diagrama D, tem-se que o diagrama ilustrado na Figura 2.3 comuta.

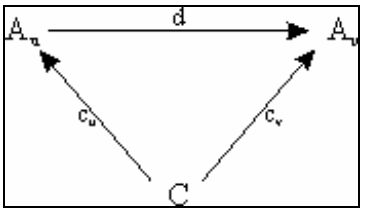


Figura 2.3 Diagrama comutativo para cone

Um limite de um diagrama é um cone que satisfaz à *Propriedade Universal do Limite*. De outra forma, pode-se dizer também que um cone é um tipo especial de limite.

Definição 4 – *Limite*- Sejam C uma categoria e D um diagrama em C. Um Limite do diagrama D é um cone $\langle P, \{p_i: P \rightarrow A_i \mid i \in I\}\rangle$ onde, para qualquer outro cone $\langle C, \{c_i : C \rightarrow A_i \mid i \in I\}\rangle$ de D, existe um único morfismo h : C→P tal que o diagrama ilustrado na Figura 2.4 comuta.

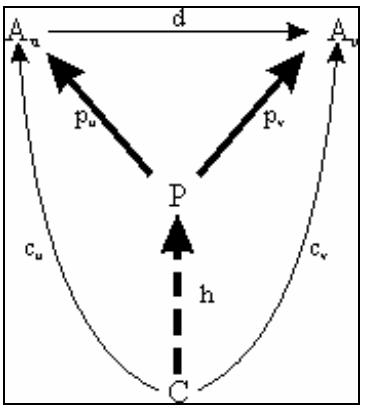


Figura 2.4 Diagrama comutativo para limite

Colimite é o conceito dual de limite e pode ser definido da seguinte forma: é um *cocone* P, $\langle \{p_i : A_i \to P \mid i \in I\} \rangle$ onde, para qualquer outro cocone $\langle C, \{c_i : A_i \to C \mid i \in I\} \rangle$ de D, existe um único morfismo $h : P \to C$ tal que o diagrama ilustrado na Figura 2.5 comuta.

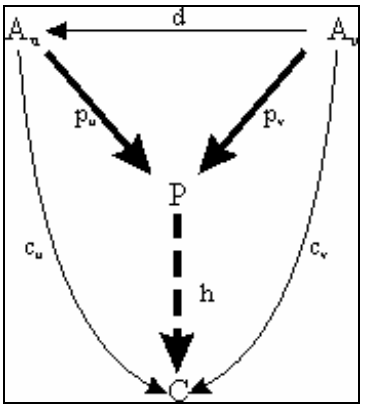


Figura 2.5 Diagrama comutativo para colimite

Exemplo 2 – *Objeto Terminal como Limite* - Sejam C uma categoria e D um diagrama vazio em C, ou seja, sem objetos nem morfismos. Então,

um *cone* de D é qualquer par ordenado $\langle C, \varnothing \rangle$, ou seja, um objeto C sem qualquer morfismo associado;

um limite $\langle P, \varnothing \rangle$ do diagrama D é um *objeto terminal*. De fato, para qualquer cone $\langle C, \varnothing \rangle$, existe um único morfismo $h: C \to P$ tal que o diagrama ilustrado na Figura 2.6 comuta, o que coincide com a definição de produto, definida a seguir.

Figura 2.6 Objeto terminal como limite

Produto e Coproduto como Casos Especiais de Limite e Colimite

Definição 5 – *Produto* - Sejam C uma categoria e $A, B \in Ob_C$. Um *Produto* de A e B é composto por um C-objeto $A \times B$ e dois C-morfismos $\pi_1: A \times B \to A$ e $\pi_2: A \times B \to B$ tais que para todo C-objeto C e para todos os C-morfismos $f_1: C \to A$ e $f_2: C \to B$, existe um único morfismo $h: C \to A \times B$ tal que o diagrama ilustrado na Figura 2.7 comuta.

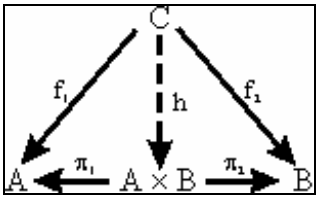


Figura 2.7 Diagrama comutativo para produtos binários

Portanto, um produto categorial não é um objeto, mas sim uma tripla constituída por um objeto e dois morfismos. Se for desejado referir-se exclusivamente ao objeto da tripla, é usual denominá-lo de *objeto resultante do produto*. Na definição acima, o seguinte é usual:

- os morfismos π_1 e π_2 são denominados de *projeções*;
- um produto constituído pelo objeto $A \times B$ juntamente com os morfismos $\pi_1: A \times B \rightarrow A$ e $\pi_2: A \times B \rightarrow B$ também é denotado por uma tripla ordenada na forma: $\langle A \times B, \pi_1, \pi_2 \rangle$
- qualquer objeto C juntamente com quaisquer morfismos $f_1: C \rightarrow A$ e $f_2: C \rightarrow B$ é usualmente denominado pré-produto dos objetos A e B. Logo, o produto $\langle A \times B, \pi_1, \pi_2 \rangle$ também é um pré-produto.

A definição de *coproduto* é a correspondente dual de *produto*, como ilustrado na Figura 2.8.

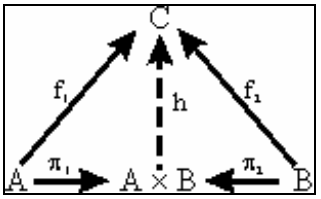


Figura 2.8 Diagrama comutativo para coproduto

Exemplo 3 – *Produto como Limite* - Sejam C uma categoria e D um diagrama em C constituído somente pelos objetos A e B e sem morfismos. Então:

qualquer *cone* $\langle C, \{c_i: C \rightarrow A_i \mid i \in I\} \rangle$ é um pré-produto de A e B;

um *limite* $\langle P, p_1: P \rightarrow A, p_2: P \rightarrow B \rangle$ de A e B é um produto de A e B. De fato, para qualquer cone $\langle C, c_1: C \rightarrow A, c_2: C \rightarrow B \rangle$ existe um único morfismo $h: C \rightarrow P$ tal que o diagrama ilustrado na Figura 2.9 comuta.

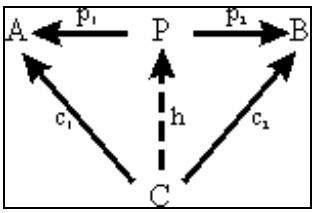


Figura 2.9 Produto como limite

Cocone e *Colimite* são os conceitos duais de *cone* e *limite*, respectivamente. Por dualidade, tem-se a construção de *coproduto* como colimite.

3 Lógica

O objetivo deste capítulo é apresentar algumas noções básicas de Lógica que, por certo, contribuirão para melhor assimilação dos assuntos tratados nos demais capítulos A Lógica fundamenta os raciocínios e as ações; o pensamento lógico geralmente é criativo e inovador.

"As regras podem mandar no mundo da tecnologia, mas não mandam na imaginação". A cabeça humana é uma máquina notável que não pode, nem deve ser robotizada. O raciocínio lógico lubrifica e torna mais produtivo o pensar em direção ao porvir. É dos hábitos da reflexão que brota o aprender [SER98].

Daremos aqui apenas os primeiros passos da Lógica, sem a pretensão de um curso completo, pois escaparia ao escopo do trabalho.

3.1 Apologias à Lógica

A Lógica é a ciência do raciocínio. **Malba Tahan**

A Lógica é a ciência das leis ideais do pensamento e a arte de aplicá-las corretamente na demonstração da verdade.**R. Solivete**

A Lógica é o instrumento da demonstração; a intuição é o instrumento da invenção. **Henri Poincaré**

A Lógica é o primeiro bater de asas para o infinito. É o grande esforço, à cata de novas certezas, de novas claridades, de mundos novos. A Lógica é a mãe da ciência; é a ordem nas idéias, é a ordem nas ações; a ordem nas palavras; a ordem no pensamento e a ordem no raciocínio: a ordem do mundo: a ordem no homem e a ordem no universo. O ilogismo é o caos, dentro e de rés. **Gomes Ribeiro**

A Matemática é a Lógica em ação. **Dominique François Arago** (1786-1853)

Fronteiras entre a Matemática e a Lógica nunca foram demarcadas. Não sabemos onde acaba a matemática e começa a Lógica, e reciprocamente. **KarI Poppev**

Matemática é Lógica. Lógica é Matemática. **Bertrand Russell** [SER98]

3.2 Introdução à Lógica

Uma lógica é um sistema formal, um sistema de manipulação de símbolos contendo uma linguagem formalmente descrita e um conjunto de regras para manipulá-los previamente estabelecidos. Uma lógica simbólica é um conjunto de estudos tendentes a expressar em linguagem matemática as estruturas e operações do pensamento, deduzindo-as a partir de um número reduzido de axiomas, com a intenção de criar uma linguagem rigorosa, adequada ao pensamento científico.

Na lógica, a linguagem formal que se manipula é formada de entidades básicas chamadas sentenças, as quais são usualmente dadas por definições indutivas e constituem uma linguagem formal. As regras da lógica devem permitir estabelecer relacionamentos, chamados de inferências, entre as sentenças. Inferência é um tipo de argumento que tem premissas e conclusão. As regras da lógica definem quais são as inferências válidas, ou seja, aquelas que, se aceitas as premissas, obrigam a que se aceitem as conclusões.

Considera-se aqui a validade formal, ou seja, um argumento (inferência) será formalmente válido se for válido em função da forma (estrutura sintática) das premissas e da conclusão. As regras de manipulação (no caso acima, o estabelecimento de relações entre premissas e conclusão) das sentenças definem o que se chama de relação de conseqüência de uma lógica.

Para tanto, há a *abordagem baseada em satisfação*, que focaliza a manipulação de

sistemas de avaliação – por exemplo, o raciocínio sobre tabelas-verdade –, interessando-se em definir qual o valor-verdade que uma *interpretação* decide para as sentenças da linguagem da lógica; e a *abordagem baseada em provas*, que focaliza a definição de meios para a manipulação de expressões sintáticas com o objetivo de formular e construir provas de argumentos, segundo a *teoria da prova,* pois as premissas e a conclusão são ligadas por uma *prova,* construída com base na forma sintática das premissas e da conclusão.

Satisfação

A definição de uma linguagem exige a especificação de uma *sintaxe* e de uma *interpretação* que seja capaz de avaliar *todas* as sentenças da linguagem com um dos valores-verdade pré-definidos na interpretação dada. Essa interpretação (dos operadores da linguagem) também pode ser equivalentemente apresentada na forma de tabelas-verdade.

A notação de símbolos A, B, etc. para sentenças e Γ, Δ, etc. para conjuntos de sentenças é muito usada. Se A é verdadeiro na interpretação *m*, escreve-se *m* ||— A, e diz-se que *m satisfaz A.* Também se diz que *A vale em m*, e que *m é um modelo para A.* Chama-se ||— uma *relação de satisfação.* Dadas uma coleção de sentenças e uma coleção de interpretações, a abordagem baseada em satisfação da lógica preocupa-se em identificar essa relação.

Sistemas de Avaliação

Na definição das linguagens (gramáticas), podem existir categorias gramaticais maiores (fórmulas, por exemplo) definidas em termos das mais simples (fórmulas atômicas e operadores, por exemplo), por motivos que serão esclarecidos nesta seção.

Uma *linguagem proposicional* L é composta de um conjunto contável de sentenças atômicas P e um conjunto de operadores (ou conectivos) O. Os operadores possuem uma aridade, que é o número de sentenças que eles tomam para formar outra sentença. O conjunto de sentenças de uma linguagem proposicional é o menor conjunto que contém o conjunto contável de sentenças atômicas e que é fechado sob os operadores do conjunto de operadores. Na seção seguinte, a qual trata da Lógica Clássica Proposicional, ela será apresentada segundo esta definição.

Um *sistema de avaliação* para uma linguagem proposicional *m* é composto por um conjunto de valores-verdade (com pelo menos dois elementos) M, um subconjunto próprio não-vazio dos valores-verdade (chamado conjunto dos valores-verdade designados) D, e um conjunto de funções F, cada uma correspondendo a um dos operadores do conjunto de operadores, tal que f_o: M^{n0} → M (onde n_0 é a aridade do operador o). Diz-se que f_o interpreta o.

Um sistema de avaliação fornece uma descrição composicional dos operadores da linguagem da lógica considerada, mas não diz como avaliar sentenças atômicas. Para que seja possível saber o valor-verdade de uma sentença, é necessário que se atribuam valores-verdade às sentenças atômicas que a constituem.

Uma *atribuição* ou *valoração* (a) relativa a um sistema de avaliação para uma linguagem é uma função que parte do conjunto contável de sentenças atômicas da linguagem (P) e chega ao conjunto de valores-verdade (M). Ou seja, a : P → M. Dada uma atribuição, é possível calcular o valor-verdade de qualquer sentença da linguagem.

Cada atribuição (a) relativa a um sistema de avaliação (M) induz uma *interpretação* ou

avaliação v_a dada por:

- $v_a(p) = a(p)$, para $p \in P$ (conjunto contável das de sentenças atômicas da linguagem)
- $v_a(o(A_1,...,A_n)) = f_o(v_a(A_1),...,v_a(A_n))$, onde *n* é a aridade do operador o, f_o interpreta o em M, e $A_1,...,A_n$ são sentenças da linguagem.

Uma *interpretação* é um sistema de avaliação mais uma atribuição. A parte do sistema de avaliação garante a avaliação dos operadores, enquanto a parte da atribuição especifica os valores-verdade das sentenças atômicas. A divisão da interpretação em dois componentes permite que os valores-verdade das sentenças atômicas (atribuição) variem, enquanto o tratamento dos operadores permanece fixo. Dessa forma, o sistema de avaliação *m* é uma coleção de interpretações, cada uma das quais tratando os operadores da linguagem de uma mesma maneira.

Uma sentença A é uma *tautologia* em um sistema de avaliação *m* se, para toda atribuição (a) relacionada a *m*, $v_a(A) \in D$.

Teoria da Prova

Uma descrição de uma lógica através da abordagem baseada em prova é chamada de *apresentação*. Uma apresentação especifica uma relação de conseqüência $\vdash$ como segue: $\Gamma \vdash A$ (lê-se *"A é derivável, ou dedutível, a partir de* Γ") se existe uma *prova* de A a partir de Γ, ou seja, uma prova com *conclusão* A e *premissas* Γ. Existem na literatura três estilos de apresentações de uso bastante difundido: axiomáticas; em dedução natural; e em seqüentes (a expressão $\Gamma \vdash A$ é um seqüente, representando o argumento com *premissas* Γ e *conclusão* A). Na seção seguinte, a qual trata da Lógica Clássica Proposicional, ela será apresentada utilizando-se o estilo axiomático.

No estilo axiomático, a relação de conseqüência é caracterizada por um conjunto dado de axiomas e um conjunto finito de regras de inferência. Os axiomas fazem o papel de "verdades básicas", e a idéia é gerar verdades adicionais pela aplicação das regras de inferência. Uma prova de A, a partir de premissas Γ, é uma seqüência finita de fórmulas, terminando com A, tal que cada fórmula da seqüência atende a pelo menos uma das seguintes condições: é um dos axiomas; ou é um membro de Γ; ou é derivável das fórmulas anteriores na seqüência por meio de uma regra de inferência. Se existe uma prova de A a partir de premissas Γ, escreve-se $\Gamma \vdash A$. Uma sentença A é um teorema da apresentação se existe uma prova de A a partir de um conjunto vazio de premissas (ou seja, $\varnothing \vdash A$).

Relações de Conseqüência

Uma relação de conseqüência sobre um conjunto de sentenças (de uma linguagem L) é a relação, escrita $\vdash$, entre P(L) (o conjunto potência de L) e L, a qual satisfaz pelo menos as seguintes propriedades tidas como essenciais em um sistema lógico (há outras propriedades, como compacidade e propriedade da dedução, por exemplo). Na notação, as vírgulas significam união de conjuntos; daí Δ,Γ significa $\Delta \cup \Gamma$ e Δ,C significa $\Delta \cup \{C\}$:

- inclusão: se $A \in \Gamma$ então $\Gamma \vdash A$
  - se A está explicitamente entre as premissas de um argumento, então A é uma conseqüência válida;

- monotonicidade: se Γ |— A, então Γ, Δ |— A
  - (regra de enfraquecimento): se a sentença A segue de Γ, então ela segue de qualquer conjunto em que Γ esteja incluído (ou seja, o conjunto de conseqüências de Γ não decresce quando Γ cresce);
- corte: se Γ |— C, Δ e C |— A, então Δ, Γ |— A
  - se é possível derivar A de algumas premissas Δ e de uma sentença C, então A pode ser derivada daquelas premissas (Δ) e de mais algumas outras premissas a partir das quais se tem C (essa propriedade é chamada de *corte*, pois podemos tirar/cortar a fórmula C e ir diretamente para a fórmula A).

Relações de Acarretamento (entailment relations)

Relações de acarretamento são um caso especial de relações de conseqüência e são definidas em termos de sistemas de avaliação. São escritas |=, possivelmente subscritas pelo sistema de avaliação em questão.

Define-se que um conjunto de sentenças Γ de uma linguagem *acarreta* uma sentença A dessa linguagem em um sistema de avaliação *m* para essa linguagem (com seus valores-verdade em M e seus valores designados em D), e escreve-se $\Gamma \models_m A$, se, para toda atribuição **a** tal que $v_a(B) \in D$ para cada $B \in \Gamma$, então $v_a(A) \in D$ também. Ou seja, se as premissas são todas designadas (isto é, avaliadas para valores-verdade designados), o mesmo acontece com a conclusão. $\Gamma \models_m A$ significa que, de acordo com o sistema de avaliação *m*, o argumento com premissas Γ e conclusão A é válido. A prova de que a relação de acarretamento é uma relação de conseqüência.

Propriedades Semânticas das Apresentações

Duas propriedades são de extrema importância quando se comparam ⊢ (relação de conseqüência definida por teoria da prova – uma apresentação) e |= (relação de acarretamento): corretude (*soundness*) e completude (*completeness*).

Corretude (*soundness*): ⊢ é correta com relação a |= se

$$\Gamma \vdash A \text{ implica } \Gamma \models A.$$

Provar que *G |¾ A implica G |= A* mostra que o sistema de prova é *correto* (*sound*) com relação ao sistema de avaliação no sentido de que ele *não é capaz de provar algo que não seja válido*. Para provar a *corretude*, é suficiente que se mostre que as regras de construção de provas preservam *acarretamento.*

Completude (*completeness*): ⊢ é completa com relação a |= se

$$\Gamma \models A \text{ implica } \Gamma \vdash A.$$

Provar que Γ |= A implica Γ ⊢ A mostra que o sistema de prova é suficientemente forte para provar tudo o que é válido (com relação ao sistema de avaliação). A prova da completude de um sistema de prova é mais difícil que a prova da corretude do mesmo. Uma maneira de proceder é mostrar que o sistema de prova pode ser estendido pela adição de qualquer sentença para a qual nem ela própria nem a sua negação sejam teorema como um axioma, de tal forma que exista uma atribuição para o sistema de avaliação a qual faça todos os sistemas verdadeiros.

Há também a questão da *decidibilidade* (grosso modo: mesmo sabendo que o sistema

dedutivo é capaz de provar todas as fórmulas válidas, existe um procedimento que garanta que se pode sempre obter tal prova?; ou, no caso de uma fórmula não válida, mostrar que tal prova não existe?). Essa questão não será descrita formalmente aqui.

3.3 Lógica Clássica Proposicional

Os símbolos da Lógica Clássica Proposicional são

- um conjunto enumerável φ de símbolos proposicionais;
- pontuação: "(" e ")";
- *conectivos* ou *operadores*: "¬", "∨", "∧" e "→"

As *fórmulas* ou *sentenças* da Lógica Clássica Proposicional são:

- todo símbolo proposicional é uma fórmula chamada *fórmula atômica*;
- se A é uma fórmula, então (¬A) também é uma fórmula;
- se A e B são fórmulas, então (A ∨ B), (A ∧ B) e (A → B) também são fórmulas;
- nada é fórmula, a menos que seja forçada a ser por um dos itens acima.

Apresentação Axiomática

O *esquema de axiomas* da Lógica Clássica Proposicional é definido considerando que, se A, B e C são fórmulas (da Lógica Clássica Proposicional), então as seguintes fórmulas são axiomas:

AXM1 A → (B → A)

AXM2 (C → (A → B)) → ((C → A) → (C → B))

AXM3 (¬A → ¬B) → ((¬A → B) → A)

Utiliza-se a *regra de inferência* chamada *modus ponens* para a formação do sistema dedutivo da Lógica Clássica Proposicional. Ela é definida (sendo A e B fórmulas e Γ um conjunto de fórmulas) como:

se Γ ⊢ A **e** Γ ⊢ (A → B) **então** Γ ⊢ B

É importante ressaltar que o conjunto de axiomas acima foi apresentado sob a forma de *esquema de axiomas*, ou seja, os esquemas apresentados não são axiomas – nem ao menos são fórmulas – são fôrmas em que cada letra (A, B e C) é receptáculo para as fórmulas que as substituem (instanciam), obtendo-se assim os axiomas. Na definição da regra de inferência, a existência de Γ nos dois "lados" da regra mostra que a conclusão depende das mesmas hipóteses de que dependem as premissas.

Sistema de Avaliação

O *sistema* clássico *de avaliação* para a Lógica Clássica Proposicional é definido sobre os seguintes conjuntos: conjunto O dos operadores da linguagem = {∨, ∧, →, ? }; conjunto M dos valores-verdade = {V, F}; conjunto D dos valores-verdade designados = {V}; e conjunto F de funções sobre o conjunto o dos operadores dado pelas seguintes tabelas:

$f_{Ú}$	**V**	**F**
V	V	V

$f_{Ù}$	**V**	**F**
V	V	F

$f_{\rightarrow}$	**V**	**F**
V	V	F

$f_{?}$	
V	F

F	V	F

F	F	F

F	V	V

F	V

Semântica

Uma semântica para a Lógica Clássica Proposicional tem aqui sua definição apresentada pela composição de diversos conceitos relacionados.

Uma função de valoração V tem a seguinte assinatura: V : φ → {V, F}. Uma extensão da função de valoração V, chamada de V^o, tem a assinatura V^o : *F* → {V, F} (onde *F* é o conjunto de todas as fórmulas, *p* é um símbolo proposicional, V é uma função de valoração, e A e B ∈ *F*) e é definida da seguinte forma:

(*S* p) $V^o(p) = V(p)$;

(*S* ¬) $V^o(\neg A) = V$, se $V^o(A) = F$

$V^o(\neg A) = F$, caso contrário;

(*S* ∧) $V^o(A \wedge B) = V$, se $V^o(A) = V$ e $V^o(B) = V$;

$V^o(A \wedge B) = F$, caso contrário;

(*S* ∨) $V^o(A \vee B) = V$, se $V^o(A) = V$ ou $V^o(B) = V$;

$V^o(A \vee B) = F$, caso contrário;

(*S* →) $V^o(A \rightarrow B) = V$, se $V^o(A) = F$ ou $V^o(B) = V$;

$V^o(A \rightarrow B) = F$, caso contrário;

Uma fórmula A é *satisfatível* se existir uma função de valoração $V^o(A) = V$. Nesse caso, diz-se que a função de valoração V satisfaz A. Uma fórmula A é *válida* se ela for satisfeita por toda função de valoração. Uma fórmula A é uma *contradição* se ela não é satisfeita por nenhuma função de valoração. Uma fórmula A é uma *conseqüência tautológica* de um conjunto de fórmulas Γ – escreve-se Γ |= A – se toda função de valoração que satisfizer todo membro de Γ também satisfaz A.

A expressão ∅ |= A pode ser interpretada da seguinte forma: como não se pode apresentar nenhuma função de valoração que não satisfaça algum membro de ∅, pois esse não possui membro algum, podemos dizer, por *vacuidade*, que qualquer função de valoração satisfaz todos os membros de ∅. Assim, se ∅ |= A, então A é satisfeita por todas as funções de valoração, ou seja, é válida. Nesse caso, escreve-se |= A e diz-se que A é uma *tautologia*.

Corretude e Completude

Como visto em uma seção anterior, o problema de provar a corretude e a completude de uma apresentação refere-se a mostrar, respectivamente, que:

$$\Gamma \vdash A \text{ implica } \Gamma \models A$$

e que

$$\Gamma \models A \text{ implica } \Gamma \vdash A$$

para A sendo uma fórmula e Γ, um conjunto de fórmulas.

Essas demonstrações são parte das demonstrações correspondentes para a Lógica Modal (capítulo 3), portanto não serão provadas neste ponto, mas no capítulo 3. As provas de tais propriedades e de outros resultados para a lógica clássica podem ser encontrados na obra de Goldblatt [GOL87]. Sendo assim, apenas se afirma que, para a semântica e para

a apresentação dadas neste capítulo para a Lógica Clássica Proposicional, temos que o conjunto de fórmulas válidas (*tautologias*) é igual ao conjunto dos teoremas, ou seja:

$\Gamma \vdash A$ se e somente se $\Gamma \models A$.

3.4 Lógica Clássica de Primeira Ordem

Esta seção descreverá brevemente a lógica clássica de primeira ordem. Alguns detalhes de formalização não serão apresentados; apenas as idéias mais gerais. Uma referência em português é a obra de Costa [COS92], pois aborda tanto os conceitos como os formalismos com uma linguagem de fácil entendimento, apresentando inúmeros demonstrações e exemplos e ainda propondo exercícios com respostas ao final da obra.

Uma lógica de primeira ordem (ou cálculo de predicados) é um sistema formal aplicado à definição de teorias do universo de discurso da matemática. Uma teoria é um conjunto de assertivas tradicionalmente chamadas de proposições, lemas, teoremas, etc., acerca de um universo de discurso. Com o uso de Lógica Clássica de Primeira Ordem, é possível representar afirmações sobre os indivíduos e suas propriedades ou relações.

A Lógica Clássica de Primeira Ordem utiliza, além dos conectivos lógicos da Lógica Clássica Proposicional, os símbolos lógicos: ∀ (para todo) quantificador universal; ∃ (existe) quantificador existencial; variáveis (para denotar indivíduos arbitrários do universo do discurso) e, opcionalmente, o símbolo de igualdade. Também são utilizados os símbolos não-lógicos constantes e símbolos funcionais (para denotar elementos específicos do universo de discurso) e símbolos predicativos para denotar relacionamentos entre indivíduos e que recebem os valores-verdade falso ou verdadeiro.

A sintaxe da lógica de primeira ordem baseia-se em um conjunto de variáveis individuais x_0, ..., x_n, ... junto com um conjunto R_n (para cada n>0) de símbolos de relações de *n*-lugares e um conjunto f_n de símbolos de funções *n*-lugares (para cada n=0). Mais tarde usaremos *Var* para o conjunto de variáveis e *Fun* para o conjunto de símbolos de funções. Adicionalmente, os símbolos básicos da linguagem *L* incluem os conetivos lógicos ∧, ∨, ? , →, os quantificadores ∀ e ∃, e a igualdade. [TUR84]

Fórmulas bem formadas (wff) são construídas a partir de certas wff atômicas através dos conetivos e quantificadores lógicos. As wff atômicas são da forma $C(t_0, ..., t_{n-1})$ onde *C* é um símbolo de relação de *n*-lugares (a igualdade (=) é de dois-lugares) e cada t_i é um termo onde *t* é um termo se é uma variável individual ou se é da forma $f(t'_0, ..., t'_{m-1})$ onde os t'_i são termos e *f* é um símbolo de função *m*-lugares (em particular, os símbolos de função de zero-lugares são termos). Wff mais complexas são formadas por conjunção (∧), disjunção (∨), negação (?) e quantificação da seguinte forma: se *A* e *B* já são wff, então *AÙB, AÚB, ? A, A→B, "xA, $xA*. Onde não há perigo de confusão, não usaremos os subscritos e os superescritos nas constantes e nas variáveis.

Lógica clássica de primeira ordem é definida, especificando-se um conjunto finito de esquemas de axiomas e regras de inferência. A maneira mais comum de apresentá-lo é a seguinte [TUR84]:

Axiomas

Para quaisquer wff *A, B, C* de *L*

1. $A \rightarrow (B \rightarrow A)$
2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
3. $(? B \rightarrow ?A) \rightarrow ((? B \rightarrow A) \rightarrow B)$

4. *"xA(x) → A(t)* onde *t* é um termo livre de *x* em *A(x)*
5. *("x)(A→B) → (A → "xB)* onde *A* não contém ocorrências livres de *x*.

Regras de Inferência

MP A, A→B + B

GEN A + ∀xA

Uma *prova* é qualquer seqüência da forma A_1, ..., A_n onde cada A_i é ou uma instância de um esquema de axiomas como segue de membros anteriores da seqüência por uma aplicação de MP ou GEN. Um *teorema* é qualquer wff que resulta de uma prova, isto é, o último membro de uma seqüência como essa. A semântica para o cálculo de predicados é fornecida pela seguinte noção.

Um *frame M de primeira ordem* consiste de um domínio *D* não-vazio, junto com uma função *F* que associa a cada símbolo de função *n*-lugares *f* uma função *f'*de D^n→*D* e a cada constante de relação de *n*-lugares, *C*, um elemento *C'* de 2^{Dn}.

Para fornecer a semântica para *L*, com respeito a tal frame, empregaremos uma função de associação g que associa a cada variável individual um elemento de *D.* Empregaremos a notação

M $|=_g$ A

para indicar que a função de associação *g satisfaz* a wff *A* no frame M.

1. $M |=_g C(t_0, ..., t_{n-1})$ se e somente se $(Val(t_0, g), \ldots, Val(t_{n-1}, g)) \hat{I} C'$.

   onde $Val(t,g)=g(t)$ se *t* é uma variável individual e $f'(Val(t'_0,g), ..., Val(t'_{m-1},g))$ se *t* é da forma $f(t'_0, ..., t'_{m-1})$.
2. $M |=_g ?A$ se e somente se $M |=/=_g A$
3. $M |=_g AÙB$ se e somente se $M|=_g A$ e $M |=_g B$
4. $M |=_g "xA$ se e somente se $M|=_g (d|x)A$ para cada *d* em *D*

onde g(d|x) é aquela função de associação idêntica a g exceto na variável x; aqui ela associa o valor d.

As condições-verdade para os outros conetivos podem ser deduzidas a partir das equivalências

A∨B ? ? (? A∧? B)

A→B ? ? A∨B

A? B ? (A→B)∧(B→A)

∃xA ? ? ∀x ? A

Uma wff *A* é *universalmente válida* se e somente se, para cada frame *M* e cada função de associação *g*, $M|=_g A$.

Essas duas noções (isto é, sendo um teorema e sendo universalmente válida) são conectadas com o teorema da completude.

Teorema da Completude: Uma wff do cálculo de predicados é um teorema se e somente se ela é universalmente válida. (não será demonstrado).

3.5 Relação entre Lógica e Ciência da Computação

Dois aspectos podem ser distinguidos na conexão entre Lógica e Ciência da Computação:

Um aspecto "Fundacional", no qual a Lógica é usada para dar um modelo do fenômeno de computação. Por exemplo: o isomorfismo Curry-Howard é uma descrição do paralelo entre computação e transformação de provas na lógica intuicionista.

Um aspecto de "Sistema de Raciocínio", no qual a Lógica é usada para a descrição e implementação de sistemas que raciocinam sobre um domínio particular. Por exemplo: a lógica temporal é usada para raciocinar sobre domínios em que o tempo desempenha uni papel chave; a lógica deôntica para domínios que envolvem comportamento normativo; etc. [SOU99]

No presente trabalho, o segundo aspecto é o que será desenvolvido.

3.6 Problemas Bem-Humorados Envolvendo Lógica

CALOUROS E VETERANOS [RAM]

Num determinado colégio, os calouros sempre mentiam, e os veteranos só diziam a verdade. Um desconhecido abordou um grupo de três alunos. O grupo era composto apenas de calouros e veteranos. O desconhecido perguntou ao primeiro aluno se ele era calouro. Este respondeu à pergunta, mas o desconhecido não escutou a resposta.

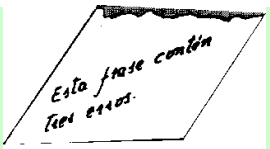
O segundo aluno então disse que o primeiro aluno negara ser calouro. Então o terceiro aluno disse que o primeiro era de fato calouro.

Por essas informações, você poderá decidir quantos eram calouros?

Resposta no ANEXO 2

OS ERROS [RAM]

Marcelo entrega esta folha de papel à professora e pergunta: “Diga-me: quais são esses erros?”



Resposta no ANEXO 2

TAMPAS TROCADAS [RAM]

Em cada uma das três caixas iguais há duas camisas. Numa estão duas brancas; em outra, duas pretas; e na terceira, uma preta e uma branca. As tampas das caixas, de acordo com o conteúdo, têm as inscrições: BB, PP e PB, mas as tampas foram trocadas e nenhuma está na caixa certa. De que caixa tem que se tirar uma camisa, sem olhar para a segunda, para se saber quais são as camisas que estão em cada uma das caixas?

Resposta no ANEXO 2

3.7 Exercícios

1 Das sentenças seguintes, assinale com o valor lógico correspondente, V ou F, as que são proposições (ou sentenças declarativas).

a) A Argentina e o Brasil.

b) O Brasil é um país da América do Sul.

c) Existem políticos que são honestos.

d) Será que meu médico é competente?

e) Jõ Soares é um artista consagrado.

O O quilômetro tem 100 metros.

2 Das expressões seguintes, diga quais são sentenças abertas e quais são proposições declarativas:

a) $x + 2 = 7$ b) $5 + 4 = 8$ c) $2 - x < 7$ d) $3 > 6$ e $3 + 2 = 4$

3 Sejam as proposições:
p: O empregado foi demitido.
q: O patrão indenizou o empregado.
Forme sentenças, na linguagem natural, que correspondam às proposições seguintes:
a) $\neg p$ b) $\neg q$ c) $p \wedge q$ d)$p \vee q$ e)$\neg p \wedge q$ f)$p \vee \neg q$

4 Sejam as proposições:
p: Jô Soares é gordo.
q: Jô é artista.
Escreva, na forma simbólica, cada uma das proposições seguintes:
a) Jô Soares não é gordo.
b) Jô Soares não é artista.
c) Não é verdade que Jô Soares não é gordo.
d) Jô Soares é gordo ou artista.
e) Jô Soares não é gordo e é artista.

5 Determinar o valor lógico (V ou F) de cada uma das seguintes proposições:
a) Não é verdade que 1998 é um número ímpar.
b) É falso que $3 + 4 = 7$ e $2 + 2 = 5$.
c) $\neg(3 + 3 = 7$ e $4 + 4 = 9)$

6 Duas grandezas x e y são tais que "se $x = 2$, então, $y = 6$". Pode-se concluir que:
a) Se $x \neq 2$, então $y \neq 6$ b) Se $y = 6$, então $x = 2$ c) Se $y \neq 6$, então $x \neq 2$
d) Se $x \neq 4$, então $y = 4$ e) Nenhuma das conclusões anteriores é válida.

7 Determine o valor lógico da proposição "se $9 + 7 = 17$, então eu sou a rainha da Inglaterra".

8 Considere a proposição: "Se ela é uma boa cozinheira, então, ela é pobre".
Agora, determine:
a) a proposição recíproca;
b) a proposição inversa;
c) a proposição contrapositiva.

4 Lógicas Categoriais

Neste capítulo, temos o objetivo de apresentar as idéias elementares que envolvem a interação entre a lógica matemática e a teoria das categorias. Especialmente, procuraremos dar uma atenção especial à semântica categórica da teoria dos tipos algébrica, abordando interpretações categóricas para variáveis, substituição e modelos. Brevemente, uma teoria na teoria de tipos algébrica envolve uma coleção de tipos, termos e axiomas, na qual as equações são os axiomas da teoria. Após uma revisão da abordagem tradicional para a definição da sintaxe (linguagem) e semântica de uma teoria, descreveremos a semântica categorial, que envolve basicamente a interpretação de tipos como objetos e termos como morfismos. Especialmente, veremos que a categoria adequada para a interpretação desse tipo de teoria é essencialmente uma categoria com produtos finitos. Após abordarmos resultados esperados como coerência e completeza, partimos para o ponto crucial desta interação. Para cada teoria TH, construímos uma categoria Cl(TH) (chamada categoria classificadora de TH) diretamente da sintaxe de TH, e mostramos que existe um modelo da teoria nessa categoria. Além disso, toda categoria com produtos finitos C define uma teoria TH(C). Essas construções são inversas uma à outra (no sentido em que C isso Cl(TH(C)) e TH(Cl(TH)) isso TH). E é por essa razão que podemos considerar teorias algébricas e categorias com produtos finitos como sendo essencialmente a mesma coisa.

"Por que abordar lógica baseando-se em categorias?"

Porque essa é uma ferramenta que permite expressar essa teoria de uma maneira fácil e clara – forma diagramática de representar as inferências.

O uso de categorias resultou em uma forma mais intuitiva de trabalhar com essa teoria. Sua aplicação parece ser de valor, pois permite uma melhor visualização das inferências feitas. [CAM2001]

4.1 Breve Apresentação da Lógica Categorial

Baseado nos conjuntos parcialmente ordenados vistos como categorias, seguem alguns exemplos de *lógicas categóricas*.

EXEMPLO 3.28 – Categoria TH

Seja TH a categoria das teorias da lógica de primeira ordem como segue:

- objetos: todas as teorias da lógica de primeira ordem clássica;
- morfismos: todas as relações de inclusão, isto é, lógicas de primeira ordem clássica: existe um morfismo de T_1 para T_2 se e somente se $T_1 \subseteq T_2$

Como seria a definição da identidade e da composição?

Existem casos interessantes de relações de ordem que são somente reflexivas e transitivas, denominadas de re1ações *de pré-ordem*. Esse é o caso da relação de dedutibilidade entre fórmulas, dado um sistema dedutivo e uma lógica. Assim, tem-se:

$\alpha \vdash \alpha$

se $\alpha \vdash \beta$ e $\beta \vdash \gamma$, então $\alpha \vdash \gamma$

Entretanto, a anti-simetria não é válida, pois se pode ter $\alpha \vdash \beta$ e $\beta \vdash \gamma$ sem que $\beta = \alpha$. Dada uma lógica, a relação de pré-ordem de dedutibilidade entre fórmulas claramente define uma categoria, como exemplificado a seguir.

EXEMPLO 3.29 - Categoria Pre_L

Dada uma lógica L, a relação de pré-ordem de dedutibilidade define uma categoria denotada por **Pre**$_L$**,** como segue:

- objetos: todas as proposições de L;
- morfismos: todos os pares da relação de pré-ordem de dedutibilidade, isto é, existe um (único) morfismo de A para B Se e somente se A |-- B.

Nesse caso, as proposições A e B diferentes, porém equivalentes, estariam na categoria como objetos distintos com dois morfismos, um de A para B e outro de B para A.

Em geral, dada uma relação de pré-ordem, é possível construir uma relação de ordem parcial induzida. Uma solução é definir uma relação de equivalência sobre os elementos de uma relação de pré-ordem de tal forma que:

se a e b são tais que $a \leq b$ e $b \leq a$, então a e b são equivalentes ($a \equiv b$)

o que determina uma classe de equivalência denotada por [a] (em que a é um elemento representativo da classe). Assim, tomando classes de equivalência de elementos em vez de elementos, tem-se que:

se $[a] \leq [b]$ e $[b] \leq [a]$, então $a \equiv b$ (e portanto $[a] = [b]$)

Dessa forma, obtém-se um conjunto parcialmente ordenado, tendo por elementos classes de equivalência e por relação de ordem $[a] \leq [b]$ se e somente se $a \leq b$. O leitor deve convencer-se de que a escolha do representante da classe na definição da ordem parcial não modifica a mesma.

Assim, a construção do conjunto parcialmente ordenado sobre o conjunto de proposições de uma lógica em que a relação de pré-ordem é a dedutibilidade ou conseqüência lógica resulta em um conjunto que pode ser usado como suporte de uma álgebra, ou seja, o conjunto que contém os elementos da mesma (a definição formal de álgebra é apresentada ao longo do livro). Essa álgebra recebe o nome de *Álgebra de Lindenbaum* (o primeiro a usar esse tipo de construção). Álgebras de Lindenbaum são bastante usadas como semântica (algébrica) de lógicas, fazendo corresponder a cada conetivo uma operação da álgebra. Por exemplo, na lógica clássica, tem-se que:

$[A] \cap [B] = [A \wedge B]$

$[A] \cup [B] = [A \vee B]$

$[\neg A]$ é o complemento de [A], etc.

A álgebra de Lindenbaum da lógica clássica proposicional é um exemplo típico de álgebra de Boole.

É possível representar o conjunto parcialmente ordenado associado a uma álgebra de Lindenbaum de uma lógica L como sendo uma categoria, denotada por **Lind**$_L$, o que é sugerido como exercício.

Por que, em vez de representar a relação de dedutibilidade como morfismos, como na categoria **Pre**$_L$, não usar os morfismos para representar provas? Essa possibilidade funciona, como ilustrado no exemplo a seguir.

EXEMPLO 3.30 – Categoria ***Prov***$_L$

Dada uma lógica qualquer L, a categoria das proposições e das provas de L, denotada por **Prov**$_L$, é como segue:

⇨ objetos: todas as proposições de L;

- ⇨ morfismos: Π_1 é um morfismo de β para α se e somente se Π_1 é uma prova de α a partir de β;
- ⇨ identidade: para cada proposição α, a prova de α a partir da própria proposição é o morfismo identidade;
- ⇨ composição: se Π_1 é uma prova de α a partir de β, e Π_2 é uma prova de β a partir de γ, então pode-se "colar" as provas de forma que toda ocorrência da hipótese β seja provada por Π_2, resultando em uma prova de α a partir de γ. Assim, a composição de morfismos é como segue:

$$\begin{array}{ccccc} & & & & \gamma \\ \underline{\gamma} & & \underline{\beta} & & \underline{\Pi_2} \\ \underline{\Pi_2} & \circ & \underline{\Pi_1} & = & \beta \\ \beta & & \alpha & & \underline{\Pi_1} \\ & & & & \alpha \end{array}$$

Por razões que ficarão claras mais tarde, os morfismos considerados são somente as deduções normais. Assim, uma vez que é feita a composição, o morfismo resultante é a dedução normal associada. O leitor pode verificar que essa operação é associativa, visto que nada mais é que uma substituição simples. No decorrer dos capítulos subseqüentes, será estudada a estrutura da categoria $\mathbf{Prov}_{Int}$, baseada na lógica intuicionista. Em lógica categórica, estudam-se as propriedades (categóricas) de $\mathbf{Prov}_L$ para lógicas específicas e também se estuda o relacionamento entre essas diversas categorias. Como exemplo, em lógica categórica, será vista uma categoria baseada na semântica da lógica de primeira ordem clássica.

*EXEMPLO 3.31 – Categoria **Est***

Considere a seguinte formalização do conceito de estrutura para uma linguagem L:

$$\langle U, Preds, Funcs, Consts \rangle$$

em que U é o conjunto de indivíduos da estrutura, e Prods, Funcs e Consts são a lista de relações, de funções e de constantes associadas a cada elemento da linguagem L, respectivamente. Dadas as seguintes estruturas:

$$\langle U_1, Preds_1, Funcs_1, Consts_1 \rangle \quad \text{e} \quad \langle U_2, Preds_2, Funcs_2, Consts_2 \rangle$$

para as linguagens L_1 e L_2, respectivamente, um homomorfismo entre essas estruturas é uma função $h : U_1 \rightarrow U_2$, juntamente com um mapeamento que associa:

- ⇨ a cada relação R_1 de aridade n em $Preds_1$ uma relação R_2 de aridade n em $Preds_2$;
- ⇨ a cada função f_1 de aridade k em $Funcs_1$ uma função f_2 de aridade k em $Funcs_2$;
- ⇨ a cada constante c_1 de $Const_1$ uma constante c_2 em $Const_2$.

Esse mapeamento deve obedecer às seguintes condições:

- ⇨ para todo $c_1 \in Const_1$, $h(c_1) = c_2$;
- ⇨ para todo $f_1 \in Funcs_1$, $h(f_1(u_1,...,u_k)) = f_2(h(u_1),...,h(u_k))$;
- ⇨ para todo $R_1 \in Preds_1$, se $\langle u_1,...,u_n \rangle \in R_1$ então $\langle h(u_1),...,h(u_n) \rangle \in R_2$.

Dada uma estrutura, a identidade do conjunto de indivíduos da estrutura e a identidade nos mapeamentos para os outros componentes (relações, funções e constantes) determina o homomorfismo identidade da estrutura. A composição de homomorfismos é dada pela composição dos mapeamentos componentes, a qual é claramente associativa. A categoria de todas as estruturas sobre qualquer linguagem (como objetos) e de todos os homomorfismos entre essas estruturas é denotada por **Est**. Se todas as estruturas são relativas a uma determinada linguagem L, então a categoria é denotada por $\mathbf{Est}_L$, a qual é uma subcategoria (plena/larga?) de **Est**.

No que segue, são discutidos alguns casos de morfismos e objetos especiais aplicados à Lógica Categórica.

4.2 Morfismos e Objetos Especiais

EXEMPLO 4.43 - Proposições Isomorfas

Lembre-se de que uma relação de pré-ordem induz uma relação de equivalência de tal forma que, se a e b são tais que $a \leq b$ e $b \leq a$, então a e b são equivalentes. Suponha que L denota uma lógica. Então (procure justificar cada uma das afirmações que seguem):

Na categoria $\mathbf{Pre}_L$, na qual os objetos são proposições, e os morfismos são pares da relação de pré-ordem de dedutibilidade entre proposições de L, existe um (único) morfismo de A para B se e somente se A |-- B. Portanto, duas proposições equivalentes são sempre isomorfas;

Na categoria $\mathbf{Prov}_L$, a categoria das proposições e provas de L, proposições equivalentes *não* necessariamente são isomorfas;

Na categoria $\mathbf{Lind}_L$, do conjunto parcialmente ordenado associado a uma álgebra de Lindenbaum de L como sendo uma categoria, proposições equivalentes não só são isomorfas, como também estão associadas a objetos idênticos na categoria.

EXEMPLO 4.44 - Objetos Inicial e Final

Na categoria $\mathbf{Pre}_{Int}$, o $\bot$ e o $T = A \vee \neg A$ (para toda proposição A) são objetos inicial e final, respectivamente.

Na categoria $\mathbf{Prov}_L$, quais podem ser objetos inicial e final, respectivamente? Se $\bot$ for um objeto inicial, então só pode haver uma prova de $\bot$ para qualquer proposição A. Abaixo seguem duas provas normais de $A \wedge B$ a partir de $\bot$ diferentes:

$$\dfrac{\bot}{A \wedge B} \qquad\qquad \dfrac{\bot \quad \bot}{A \quad B}\,\dfrac{}{A \wedge B}$$

Quanto ao objeto terminal T, observe que, dada qualquer proposição A, existe uma única prova normal de A para T, que é uma aplicação de $\vee$-Introdução à primeira;

Na categoria $\mathbf{Est}_L$ das estruturas sobre uma linguagem, há objeto inicial. Existe uma contrapartida de Álgebra Inicial, mas que diz respeito somente ao universo de indivíduos. Trata-se do *universo de Herbrand*, que é o conjunto de termos fechados, ou seja, sem variáveis da linguagem. Por exemplo, se a linguagem é composta por um símbolo funcional f e uma constante c, então o universo de Herbrand é como segue:

{c, f(c), f(f(c)), f(f(f(c))), ...}

Entretanto, as interpretações dos símbolos predicativos estão totalmente abertas e, portanto, não há como relacioná-las diretamente, mas todas têm relação com a estrutura para a linguagem que tem por domínio o universo de Herbrand e tem por interpretação, para cada símbolo predicativo, o conjunto vazio. Esse é o objeto inicial de $\mathbf{Est}_L$. Em se tratando de modelos de uma determinada apresentação de teoria sobre L (conjunto de axiomas), o cenário é o mesmo. Entretanto, sendo Δ uma axiomatização, então o conjunto de modelos Md(Δ) estabelece uma subcategoria de $\mathbf{Est}_L$.Nesse caso, essa subcategoria tem como objeto inicial o modelo de Δ sobre o universo de Herbrand.

4.3 Isomorfismo de Categorias: Conjuntos Finitos e Lógica Booleana

Como já foi visto, uma álgebra é, basicamente, um conjunto munido com algumas operações de um ou mais argumentos. Essas operações têm condições expressas em termos de equações. Por exemplo, o conjunto dos números naturais, munido com a soma (+, de aridade dois), a multiplicação ($\bullet$, de aridade dois) e as constantes 0 e 1 (de aridade zero), é uma álgebra que tem por equações o seguinte conjunto (variáveis são usadas para indicar que as equações devem valer para qualquer elemento da álgebra):

$x + 0 = x$ $\qquad$ $0 + x = x$

$x \bullet 1 = x$ $\qquad$ $1 \bullet x = x$

$x + y = y + x$

$x \bullet y = y \bullet x$

$x \bullet (y + z) = x \bullet y + x \bullet z$

Esse é um exemplo de uma *álgebra concreta.* Uma álgebra é dita *álgebra abstrata* quando não é indicado o conjunto sobre o qual as operações são definidas. Assim, uma álgebra abstrata é indicada pelas operações e constantes e pelas equações. As álgebras concretas que satisfazem as mesmas equações das álgebras abstratas às vezes são tomadas como um modelo (semântica) das últimas.

Um dos exemplos mais conhecidos de álgebras concretas é o conjunto das partes de um dado conjunto S, ou seja, 2^S, munido com as operações de união ($\cup$), intersecção ($\cap$), complemento ($\neg$) e as constantes $\bot$ (para o conjunto vazio) e T (para o conjunto S). As equações são:

$x \cup y = y \cup x$ $\qquad$ $x \cap y = y \cap x$

$x \cup (y \cup z) = (x \cup y) \cup z$ $\qquad$ $x \cap (y \cap z) = (x \cap y) \cap z$

$x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$ $\qquad$ $x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$

$x \cup \bot = x$ $\qquad$ $x \cap \bot = \bot$

$x \cup \neg x = T$ $\qquad$ $x \cap T = x$

Esses são exemplos de álgebras Booleanas concretas (há menção a um conjunto sobre o qual as operações são definidas). Um homomorfismo entre álgebras (concretas) é um mapeamento entre os respectivos conjuntos e as operações tal que as equações são preservadas. Assim, por exemplo, se h é um homomorfismo entre álgebras de Boole, então $h(x) \cup_2 h(y) = h(y) \cup_2 h(x)$, que foi obtida por aplicação de h à equação $x \cup_1 y = y \cup_1 x$, onde $\cup_1$ e $\cup_2$ são operações em cada álgebra respectivamente. Um isomorfismo

entre álgebras é um homomorfismo que tem uma inversa que também é um homomorfismo.

Um dos resultados mais interessantes sobre álgebra Booleana é um teorema de representação que afirma que toda álgebra de Boole finita é isomorfa a uma álgebra de Boole do conjunto das partes de um conjunto. Assim pode-se pensar em uma categoria de álgebras de Boole finitas **FinBoole** como tendo álgebras das partes de conjuntos como objetos e homomorfismos entre essas como morfismos.

Considere a categoria **FinSet** introduzida anteriormente. Sejam A e B dois conjuntos finitos e seja f : A→B uma função. Considere o mapeamento:

$$f^{-1} : 2^B \rightarrow 2^A$$

que associa a cada subconjunto S de B a sua imagem inversa $f^{-1}(S) = \{a \mid f(a) \in S\}$. Prova-se que f^{-1} é tal que:

$$f^{-1}(S_1 \cap S_2) = f^{-1}(S_1) \cap f^{-1}(S_2) \qquad \text{e} \qquad f^{-1}(\neg S_1) = \neg f^{-1}(S_1)$$

Então, f^{-1} é um homomorfismo entre as álgebras de Boole. Portanto, a categoria FinSetop, dual de **FinSet**, é isomorfa a **FinBoole**.

4.4 Álgebras e Lógicas de Primeira Ordem

O leitor deve ter notado o estreito relacionamento entre o conceito de assinatura de uma linguagem algébrica e o de uma linguagem de primeira ordem. Considerando somente os símbolos funcionais e as constantes de uma linguagem de primeira ordem, tem-se uma assinatura sobre um único sorte. Entretanto, o conceito algébrico de assinatura não leva em conta os símbolos predicativos.

Existe uma versão "sortida" da linguagem da lógica de primeira ordem. Nesse caso, incluem-se sortes para as variáveis, e os símbolos funcionais e constantes têm assinatura (na versão monossortida, somente a aridade é relevante). Cada símbolo predicativo de aridade n é associado a um string $s_1 s_2 ... s_n$ que indica os sortes dos termos que podem ser usados na formação de fórmulas atômicas.

Dessa forma, as variáveis também pertencem a seus respectivos sortes. Chama-se a esse tipo de linguagem de primeira ordem de linguagem "polissortida". Ela torna mais compactas e claras certas especificações, nas quais, em linguagem monossortida, faz-se necessário o uso de predicados para indicar a qual sorte indivíduos pertencem. Por exemplo, considere a linguagem do tipo de dado *Arranjo* (exemplo introduzido anteriormente) na qual existem três predicados, todos de aridade 1, como segue:

A denota o conjunto dos arranjos;

E denota o conjunto dos elementos de arranjos;

I denota o conjunto dos índices;

bem como os seguintes símbolos funcionais:

AT com aridade três, indica a função tal que, dado um arranjo, um elemento e um índice, retorna um novo arranjo com o elemento ocupando a posição indicada pelo índice, e nada mais é alterado;

CN com aridade dois, indica a função tal que, dado um arranjo e um índice, retorna o elemento (seu valor) que ocupa a posição indicada pelo índice no mesmo.

A especificação abaixo em lógica monossortida:

$\forall a_1\ \forall a_2\ \forall e\ \forall i\ (A(a_1) \wedge A(a_2) \wedge E(e) \wedge I(i) \rightarrow (AT(a_1, e, i) = a_2 \leftrightarrow (\forall y\ (y \neq i \rightarrow (CN(a_2, y) = CN(a_1, y)) \wedge (y = i \rightarrow CN(a_2, y) = e))))$

fica como abaixo em lógica polissortida:

a1, a2 : A

e : E

i, j : I

$\forall a_1\ \forall a_2\ \forall e\ \forall i\ (A(a_1, e, i) = a_2 \leftrightarrow (\forall j\ (j \neq i \rightarrow (CN(a_2, j) = CN(a_1, j)) \wedge (j = i \rightarrow CN(a_2, y) = e))))$

As estruturas para as linguagens de primeira ordem têm um universo de indivíduos para cada sorte além das interpretações de cada símbolo não-lógico da linguagem. O conceito de homomorfismo entre estruturas para a lógica polissortida de primeira ordem é, então, uma extensão do conceito usual, bastando incluir os mapeamentos entre os universos de indivíduos para cada sorte. Considerando linguagens com a mesma assinatura Σ (de primeira ordem), tem-se a categoria das estruturas para essas linguagens $\mathbf{Est}_\Sigma$. Dada uma apresentação de teoria, ou mesmo uma teoria T, tem-se a categoria de modelos de T, a qual é subcategoria (plena/larga?) de $\mathbf{Est}_\Sigma$. Para ambas as categorias, existe objeto inicial.

5 Lógica Intuicionista

5.1 Pensamento Axiomático versus Pensamento Intuicionista em Matemática

Parcialmente relacionado com os aspectos mais amplos do problema apresentado pelos paradoxos, questionamos agora sobre a natureza da matemática e o escopo dos métodos matemáticos. Para facilitar o trabalho do aluno ao estudar o conteúdo referente a este capítulo, há um esquema de estudo sobre Lógica Intuicionista no ANEXO 1 deste volume.

O método axiomático-dedutivo na matemática é conhecido por nós dos "Elementos" de Euclides (c. 330-320 a.C.) embora haja uma tradição que credita a Pitágoras (século VI a.C.) a apresentação do método. Pelo uso desse método, o corpo do conhecimento geométrico foi sistematizado. O sistema axiomático de Euclides pode ser descrito rudimentarmente assim: as "definições" de certos *termos primitivos* tais como "ponto", "linha", "plano" são dadas, o que pretende sugerir ao leitor o que é significado por esses termos; certas proposições relativas aos termos primitivos, que ficam para ser aceitáveis como imediatamente verdadeiras na base dos significados sugeridos pelas definições, são tomadas como *axiomas* ou *postulados*; então outros termos são definidos em termos dos termos primitivos; e outras proposições, chamadas *teoremas*, são deduzidas, pela lógica, dos axiomas. Axiomatização como a de Euclides, para os quais significados são dados para os termos primitivos desde o princípio, é chamada *axiomatização material*.

Um dos postulados de Euclides pareceu menos evidente do que os demais: o quinto postulado ou o "postulado paralelo". *[* Leia-se: Se duas linhas retas em um plano encontram uma outra linha reta no plano, tal que a soma dos ângulos internos no mesmo lado da última linha reta é menor que dois ângulos retos, então as duas linhas retas irão se encontrar naquele lado da última linha reta. *]* Ele usou isso para provar o teorema de que, através de um dado ponto P que não está em uma dada linha l, pode ser traçada exatamente uma linha paralela a l (isto é, que não encontra l em um ponto). Foram feitos esforços desde o tempo de Euclides para provar esse postulado a partir dos outros como um teorema. Nós agora sabemos que esses esforços não poderiam ter sucesso.

Por isso, Lobatchevski, em 1829, e Bolyai, em 1933, trabalharam sobre um sistema de geometria no qual, através de um dado ponto P que não está em uma dada linha l, infinitamente muitas linhas paralelas a l podem ser traçadas. Isso deixou aparente que o significado dos termos primitivos de Euclides em termos de espaço físico não possibilita alguém a decidir se o postulado paralelo de Euclides ou o postulado contrário de Lobatchevsky e Bolyai é verdadeiro. As diferenças nas outras geometrias podem ser pequenas demais para serem mostradas em quaisquer medidas que nós possamos fazer na porção do espaço acessível a nós, assim como algumas outras vezes as pessoas imaginaram a Terra plana a partir da porção dela que elas podiam ver.

Então se uma proposição da geometria de Euclides é exatamente verdadeira deve ser uma propriedade da geometria como um sistema lógico. Mas se a geometria de Euclides é uma estrutura lógica válida, então é a geometria de Lobatchevski. Por isso, como Felix Klein destacou em 1871, os axiomas da geometria plana Lobatchevskiana são todos verdadeiros quando os termos primitivos neles são reinterpretados para que "plano" seja tomado como significando o interior de um dado círculo no plano de Euclides, "ponto" signifique um ponto dentro desse círculo, "linha" signifique uma corda desse círculo, e

distâncias e ângulos são computados por fórmulas devidas a Cayley 1859. (Um outro modelo como o de Euclides, aplicável a uma porção limitada do plano não-Euclideano, foi dado em 1868 por Beltrami, que reinterpretou segmentos de linha como segmentos de trajetórias curtas entre pontos., ou "geodésias", sobre uma "superfície de curvatura constantemente negativa").

Nesses modelos, podemos observar que alguma coisa nova foi feita com os axiomas, não para ser encontrado no pensamento axiomático anterior: o significado dos termos primitivo variou, mantendo a estrutura dedutiva da teoria fixada. Então as axiomatizações formais surgiram, e nelas os significados dos termos primitivos, ao contrário de estarem especificados antecipadamente, são deixados não-especificados para as deduções dos teoremas a partir dos axiomas. Alguém fica então livre para escolher os significados dos termos primitivos de qualquer modo que torne os axiomas verdadeiros. Temos representado esse ponto de vista em nossa definição de "conseqüência válida" na teoria modelo (§§ 7, 20). Especialmente na álgebra moderna, foi provado muito frutífero desenvolver as conseqüências de sistemas de axiomas considerados formalmente, tais como os axiomas da teoria dos grupos abstratos (cf. § 39). Os resultados deduzidos a partir dos axiomas da teoria dos grupos, enquanto deixam não-especificado o conjunto de elementos e a operação de multiplicação, constitui um corpo de teoria pronto para usar para diversas aplicações.

Em axiomatizações formais, o sistema de axiomas pode ser investigado por propriedades tais como a independência de um axioma em relação aos outros (procurando uma interpretação dos termos primitivos que torne o axioma falso e os outros verdadeiros), categoricidade (isto é, que os elementos em quaisquer duas interpretações podem ser colocados em uma correspondência um para um, preservando todas as propriedades), etc. *[* Esse tipo de tratamento de sistemas axiomáticos é bem apresentado em J.W.Young 1911. Vários dos tópicos mencionados brevemente agora serão elaborados mais tarde: provas de independência no §57; categoricidade no §53; provas de consistência por interpretação no §52. *]*

Nesta abordagem para axiomatizações, surgem algumas questões. Por que escolhemos os axiomas que nós fazemos, e por que os sistemas resultantes deveriam nos interessar? A resposta é evidentemente que nós podemos aplicar a teoria resultante a sistemas de objetos que providenciaram desde o princípio os axiomas por uma interpretação dos termos primitivos. Algumas vezes, interpretações essencialmente diferentes são possíveis (então os axiomas não são categóricos, mas ambíguos); por exemplo, esse é um caso para os axiomas para grupos abstratos. Não devemos querer empregar um sistema de axiomas satisfeito sob nenhuma interpretação; chamamos um sistema assim de *vacuous*. Um dos problemas de axiomatização formal é mostrar sistemas de axiomas não-*vacuous*. Entretanto, um sistema de objetos usado como uma interpretação é freqüentemente traçado a partir de alguma outra teoria axiomática; então temos um regresso, o qual, ao invés disso, meramente nos traz a questão da significância da teoria axiomática. Se, em nenhum estágio, uma aplicação é feita, desde o princípio, de axiomatização formal, a atividade inteira deve parecer fútil. Portanto, concluímos que, se não vamos adotar um niilismo matemático, a matemática axiomatizada formalmente não deve ser a matemática inteira. Em algum lugar, deve haver significados, verdade e falsidade. E pelo menos, quando dizemos que, em uma dada teoria axiomática formal, uma certa proposição é um teorema, devemos acreditar que isso é verdade, isto é, que a proposição realmente segue de seus axiomas, embora se a própria proposição for verdade, está sendo deixado de contar, pois em axiomatizações formais as deduções são carregadas antes de determinar significados aos termos primitivos (ou sem respeitar

qualquer outra determinação).

Como uma ilustração adicional de uma proposição matemática que não deve ser determinada meramente como uma conseqüência formal, mas significativa de axiomas, considere o teorema (provado na teoria dos números) de que, para dados inteiros a, b, c, podemos descobrir se existem ou não inteiros x e y tais que ax+by+c=0, isto é, o teorema de que há um método de decidir se a equação ax+by+c=0 (a, b, c inteiros) é ou não resolvível nos inteiros. Embora a teoria dos inteiros pode ter sido estabelecida axiomaticamente, essa prova pretende significar que, para alguns a, b, c particulares, *podemos descobrir* se há ou não soluções. Um estudante que pudesse meramente dar uma prova a partir de axiomas do teorema de que alguém pode descobrir se há soluções ou não, mas não pudesse fazer um problema no qual ele descobrisse, não teria assimilado o que o professor pretendeu ensinar. Apesar disso, ele estaria fazendo tudo o que deveria ser pedido para ele se o teorema (que alguém pode descobrir) fosse entendido apenas no sentido de axiomatizações formais.

Como o método menos drástico de encontrar a situação estabelecida pelos paradoxos, descrevemos a teoria dos conjuntos axiomáticos no final do §35. Aqui as axiomatizações devem ser entendidas no sentido formal, a menos que alguém esteja tentando reter uma concepção intuitiva de conjuntos, o que foi presumido que era exatamente o que os axiomas pretendiam suplantar. Apesar de as presentes considerações mostrarem que o recurso para uma teoria axiomática formal, embora possa oferecer consideráveis vantagens, deixa abertos problemas tais como por que os axiomas são significantes e se eles se aplicam ou não a qualquer sistema de objetos, não mera e similarmente postulados como os existentes para algumas outras teorias axiomáticas formais.

Hilbert ocupou-se com lidar com esses problemas. Ele admitiu que a matemática *clássica* (isto é, a matemática familiar, usando a lógica clássica) contém muito que vai além do que é claramente significativo e justificável em terrenos intuitivos, tais como os que de fato foram feitos por matemáticos geralmente para perceber quando, na teoria dos conjuntos, eles vão longe demais e encontram paradoxos. Mas ele propôs manter a matemática clássica (com poucos paradoxos) através de um programa que poderíamos descrever rudimentarmente como segue. A matemática clássica deveria ser formulada como uma teoria axiomática formal, e então deveria ser mostrado que a teoria é consistente, isto é, livre de contradições.

Antes desse propósito de Hilbert, feito primeiro em 1904, mas não tomado seriamente por ele e por seus colaboradores até depois de 1920, provas de consistência foram dadas para teorias axiomáticas formais por meios de um modelo ou uma interpretação, na qual todos os axiomas são descobertos verdadeiros quando os termos primitivos são interpretados em termos de outra teoria. Vimos um exemplo disso acima, pelo qual mostra-se que a geometria plana não-Euclideana de Lobatchevsky é consistente se a geometria de Euclides for consistente. *[* Esse tipo de tratamento de sistemas axiomáticos é bem apresentado em J.W.Young 1911. Vários dos tópicos mencionados brevemente agora serão elaborados mais tarde: provas de independência no §57; categoricidade no §53; provas de consistência por interpretação no §52. *]* Nesse caso, uma prova de consistência por um modelo mostra somente que uma teoria é consistente se alguma outra o for. Pelo método de geometria analítica de René Descartes (1619), a consistência de geometrias geralmente é reduzida àquela da teoria dos números reais, isto é, à análise. Mas como seria um método para estabelecer a consistência da análise? Certamente não usando um modelo geométrico; isso seria um círculo vicioso. Nem, de acordo com Hilbert e Bernays (1934), apelando ao mundo físico. Por limitações de

nossas medidas no mundo físico, impedimo-nos de dizer que o contínuo é realmente dado pela experiência; melhor do que isso é uma idéia de obtermos por extrapolação ou idealização o que é realmente dado. [KLE68]

Então a proposta de Hilbert de provar que a matemática clássica como incorporada em um sistema formal consistente chamado por um novo método no lugar do método de dar um modelo. Esse método consiste em uma aplicação direta da idéia da consistência, isto é, que não há contradição ou paradoxo consistindo de dois teoremas, um dos quais é a negação do outro. Para mostrar que isso não pode acontecer, Hilbert propôs fazer as provas na teoria axiomática, o objeto da investigação matemática, chamado *metamatemática* ou *teoria das provas*. Naturalmente, tal demonstração de consistência seria relativa aos métodos usados na metamatemática. Hilbert, portanto, almejava usar em sua matemática apenas métodos, que ele chamou "finitários", que eram convincentes intuitivamente. Especificamente, esses métodos deveriam evitar usar um infinito "real" ou "completo". A nova abordagem de Hilbert evitou o infinito completo nas declarações do problema de provar a consistência. Por isso, existe apenas uma infinidade contável de provas em uma dada teoria, e a proposição de consistência é relativa apenas a qualquer par de provas, e não ao conjunto de todas as provas como um objeto completo. (O que a teoria supõe estar acima disso deve ser muito menos elementar.) Então isso não parece razoável para esperar que o problema da consistência, agora que foi declarado em termos finitários, deva ser solucionado por métodos finitários.

Brouwer foi o campeão do pensamento intuitivo na matemática, assim como Hilbert foi o do pensamento axiomático. As abordagens de Brouwer e de Hilbert podem também ser chamadas de "genética" (ou "construtiva") e "existencial", respectivamente.

De acordo com Weyl (1946), "Brouwer tornou claro, como eu penso que vai além de qualquer dúvida, que não há evidências sustentando a crença no caráter existencial da totalidade de todos os números naturais... A seqüência dos números que cresce além de qualquer estágio já atingido por passar para o próximo número é um MANIFOLD de possibilidades abertas até o infinito; isso se mantém para sempre na situação de criação, mas não é um domínio fechado de coisas existentes nelas próprias."

Enquanto Hilbert propôs costear a estrutura da matemática clássica pela prova da consistência, Brouwer estava pronto para abandonar aquelas partes da matemática nas quais os matemáticos estiveram carregando por palavras que saem do significado claro. Brouwer propôs, em vez disso, desenvolver uma matemática "intuicionista", que iria apenas até onde a intuição pudesse levá-la. Para Brouwer, os sistemas de objetos para matemática deveriam ser gerados por alguns princípios de construção, e não trazidos à existência todos de uma vez como conjuntos satisfazendo uma lista de axiomas.

Como Brouwer tomou apenas o infinito "incompleto" ou "potencial" como intuitivo, ele não aceitou princípios lógicos que exigem, para sua justificativa, uma concepção de conjuntos infinitos com completos. Então, em um artigo intitulado "A UNTRUSTWORTHINESS dos princípios da lógica" (1908), ele desafiou a suposição de que as leis da lógica clássica tinham uma validade absoluta, independente do assunto-sujeito para o qual elas são aplicadas. Particularmente, ele criticou a lei do meio excluído, $Pv\sim P$. Considere um predicado $P(x)$ onde x classifica-se sobre algum conjunto D. Aplicada a $\exists xP(x)$ como P, a lei diz que, ou há um x em D tal que $P(x)$ ou não há um x em D tal que $P(x)$; em símbolos: $\exists xP(x)v\sim\exists xP(x)$. No caso de D ser um conjunto finito (e $P(x)$ ser um predicado tal que, para cada valor x em D, nós podemos testar se $P(x)$ é verdadeiro ou não), Brouwer encontra $\exists xP(x)v\exists\sim xP(x)$ verdadeiro. Por isso,

podemos descobrir se $\exists xP(x)$ ou $\sim\exists xP(x)$ testando, para cada membro x de D de cada vez, se P(x) é verdadeiro ou não para aquele x. Como D é finito, esse processo terminará (em princípio). Mas se D é um conjunto infinito, digamos um conjunto contável, tal como o conjunto dos números naturais, o processo de teste não pode humanamente ser completado. Se somos sortudos, podemos dividir o caminho através do teste encontrar um x tal que P(x). Mas se não há tal x, ou se tal x aparece muito tarde e o dia do juízo chegar mais cedo, nós poderemos procurar até o dia do juízo e ainda não ter uma resposta a nossa questão. De acordo com Brouwer, não se encontra nenhum embasamento para considerar que $\exists xP(x) \vee \sim\exists xP(x)$ seja sempre verdadeiro, quando D é infinito. Citando de Weyl 1946, "De acordo com sua visão e leitura de história, lógica clássica foi abstraída da matemática de conjuntos finitos e seus subconjuntos... Esquecidos de sua origem limitada, alguém mais tarde interpretou mal que a lógica para alguma coisa acima e prioridade para toda a matemática, e finalmente a aplicou, sem justificativa, à matemática de conjuntos infinitos."

Na maneira de Brouwer, assim como na de Hilbert, como vamos ver, tiveram que lidar com dificuldades. Uma matemática intuicionista foi desenvolvida (começando em 1918), e em parte ficou com pouco da matemática clássica nos resultados obtidos, em parte tomou uma direção diferente. Nas partes comuns à matemática clássica e à intuicionista, as provas intuicionistas (ou "construtivas"), embora freqüentemente mais difíceis, freqüentemente davam mais informações. Para provar uma declaração existencial $\exists xA(x)$, um intuicionista insiste que deve ser mostrado como encontrar um x tal que A(x). Uma "prova indireta" mostrando que a suposição $\sim\exists xA(x)$ leva a uma contradição não é aceita por ele como demonstração de que $\exists xA(x)$; ela estabelece apenas $\sim\sim\exists xA(x)$. *[* Demos um pequeno esboço mais completo em IM no §13. Apresentações excelentes estão em Heyting 1934, 1955 e 1956. Kleene e Vesley 1965 são para leitores familiarizados com os capítulos I-XIII (ou no mínimo IV-VIII) de IM ou o equivalente. *]*

Agora vamos tocar na controvérsia entre Brouwer e Hilbert. Brouwer argumentava que, mesmo se Hilbert pudesse ter sucesso ao dar a prova da consistência para a matemática clássica, isso não tornaria a matemática clássica correta. Então ele escreveu em 1923, "Uma teoria incorreta que não é parada por uma contradição não é nem um pouco a menos incorreta, assim como uma política criminal que não foi verificada por um tribunal censor não é nem um pouco a menos criminal." Hilbert rebateu em 1928, "Tirar fora a lei do meio excluído dos matemáticos seria como negar os astrônomos o telescópio ou ao boxeador o uso de seus punhos." Essa controvérsia entre os "formalistas", representados por Hilbert, e os "intuicionistas", representados por Brouwer, levou finalmente ao reconhecimento, pelos intuicionistas, de que o programa de Hilbert seria irrepreensível se e somente se os formalistas abstiverem-se de tomar a prova da consistência como uma justificativa para juntar um significado real àquelas partes da matemática que os intuicionistas rejeitam como não tendo bases intuitivas (Brouwer 1928).

Sobra então para os formalistas explicar, depois de ter admitido que a matemática clássica vai além da evidência intuitiva, como apesar disso suas partes não intuitivas podem ter valor. Dirigindo-se a esse problema, Hilbert 1926, 1928 traçou uma distinção entre *declarações reais* que têm um significado intuitivo, e *declarações ideais* (envolvendo o infinito completo) que não têm. É um instrumento comum da matemática moderna juntar "declarações ideais" a um sistema previamente constituído para atingir objetivos teóricos, tais como simplificar teoremas, compreendê-los sob um ponto de vista mais unificado, etc. Um exemplo ocorre na geometria projetiva, onde uma linha

até o infinito é unida à parte finita do plano para que quaisquer duas linhas paralelas (distintas) se interceptem em um ponto daquela linha. Desse modo, a exceção para linhas paralelas para as "relações de incidência" entre pontos e linhas é removida. Então, na geometria projetiva, não só realmente cada dois pontos distintos contêm uma única linha (que passa através de ambos), como dualmente cada duas linhas distintas contêm um único ponto (no qual elas se interceptam). Hilbert argumentou que apenas esse tipo de ganho teórico é atingido, juntando as declarações ideais às declarações reais na matemática clássica; é através desse procedimento que a matemática clássica atinge seu poder e sua elegância. Dessa forma, a matemática torna-se uma construção teórica na qual, Hilbert diz, não deveria ser esperado que cada declaração separada devesse ter um significado real, algo mais do que aquele que cada proposição em um sistema de natureza teórica deveria ser capaz de verificação experimental imediata; no último caso, é a teoria como um todo que é testada contra a realidade.

Um exemplo concreto de ganho teórico obtido, avançando através de declarações ideais no processo de provar declarações reais, é fornecido pela *teoria analítica dos números*, na qual os teoremas sobre inteiros são fornecidos pela teoria dos números reais e complexos. Muitas proposições da teoria elementar dos números têm sido provadas dessa forma, a qual nós nem sabemos como provar, nem podemos estabelecer apenas por provas muito mais complicadas se apenas métodos não analíticos são usados.

Proximamente relacionado a essa defesa da matemática clássica como um esquema de sistematização simples e elegante é a defesa fornecida por seu sucesso em aplicações às ciências teóricas, especialmente Física. Isso levou Weyl 1926 a pronunciar que Hilbert estava correto quando a matemática é misturada com a física no processo da construção do mundo teórico, enquanto ele tomou partido com Brouwer restringindo-se às verdades intuitivas quando a matemática se ocupa consigo mesma somente. [KLE68]

Ahora bien: si es preciso guardarse de confundir los distintos cálculos lógicos con las diversas aplicaciones de la lógica, es también necesario reconocer la estrecha relación existente entre una y otra cosa, entre necesidades de aplicación de la lógica formal y conveniencias de elaboración de lógicas no clásicas. Es evidente que no se trata de dos procesos separados: la aplicación dei análisis formal a una determinada materia puede revelar y de hecho ha revelado en multitud de ocasiones, y sigue revelando insuficiencias de la lógica en su actual estado, y ha constituido, por tanto, un estimulo para su despuegue para su ampliación, para su afinamiento en una u otra dirección.

Por ejemplo, como es bien sabido, la filosofia intuicionista de la matematica constituyó una respuesta a ia crisis de fundamentos de esa disciplina. Los intuicionistas habiaron de una «lógica matemática» en ei sentido de una sistematización a *posteriori* de las regias dei razonamiento efectivamente empleadas por ei matemático en sus construcciones. Seria, pues, una *lógica de la matemática*, un registro de las regias admitidas como válidas en esta ciencia. Ahora bien: en la medida en que ei intuicionismo supone una contracción astringente de la matemática, la *lógica intuicionista* será asimismo una lógica más restrictiva que la lógica clásica. Asi, por ejemplo, como ei intuicionismo no admite que en matemática todo enunciado haya de ser o bien verdadero o bien falso (sin por elio, dicho sea de paso, admitir la posjbilidad de valores intermedios), la lógica intuicionista no admitirá ei principio de bivalencia. Otras peculiaridades de esta lógica se refieren a sus concepciones de la negación y de ia existencia (es decir, dei cuantificador existencial). Asi pues, en resumidas cuentas, ia lógica intuicionista es una restricción de la lógica ciásica en ei campo de la matemática.

Pero aunque esto era asi en ei intuicionismo tradicional, es decir, aunque autores como Brower pretendían sustituir la lógica clâsica por la lógica intuicionista tan sólo en ei campo de las entidades matemáticas, no faltan quienes más recientemente han propuesto lievar a cabo esa sustitución en todos los campos, extrapolando ias exigencias intuicionistas iniciales aí todo deI razonamiento humano. Dummett, por ejemplo.

De hecho, hoy en dia la concepción intuicionista de la lógica desempena, en cualquier caso y como minimo, un papel de sobre-exigencia, de desafio. Puesto que, como hemos dicho, sus criterios son mas estric-tos de lo normal, ei atenerse a elios o ei sutiple tenerlos presentes en la construcción de cálculos o en ei diseno de demostraciones supone someter a la teoria de la argumentación humana a una prueba cuya severidad es un acicate dei rigor. [DEA 78]

Uma interpretação radical dos paradoxos foi defendida por Brouwer e sua escola intuicionista [HEY56]. Eles se recusaram a aceitar a universalidade de certas leis lógicas básicas, tais como a lei do meio excluído: P ou não-P. Tal lei, eles defendem, é verdadeira para conjuntos finitos, mas é inválido estendê-la por atacado como base para todos os conjuntos. Dessa forma, eles dizem que é inválido concluir que "há um objeto x tal que não-P(x)" segue de "não-(para-todo-x, P(x))"; somos justos ao afirmar a existência de um objeto que tem uma certa propriedade somente se nós conhecemos um método efetivo para construir (ou encontrar) tal objeto. Os paradoxos, naturalmente, não são deriváveis (ou mesmo significativos) se obedecermos às estruturas intuicionistas, mas, ao contrário, muitos deles são teoremas queridos do dia-a-dia matemático, e, por essa razão, os intuicionistas encontraram poucos convertidos entre os matemáticos. [MEN64]

Um outro exemplo de cálculo proposicional não clássico é o "cálculo proposicional intuicionista", no qual a lei do meio excluído $A \vee \neg A$ e a lei da dupla negação $\neg\neg A \rightarrow A$ não são afirmados. O ponto de vista a partir do qual o sistema intuicionista de lógica surgiu e é de interesse será considerado mais tarde (parágrafo 36). Nós não tentamos aqui uma descrição modelo-teórica do cálculo proposicional intuicionista. Uma formulação prova-teórica é obtida substituindo nosso esquema de axioma 8 ($\neg\neg A \rightarrow A$) por 8^{I}. $\neg A \rightarrow (A \rightarrow B)$. Como qualquer axioma, por esse esquema, é provável no cálculo proposicional clássico (cf. * 10^{a}), o cálculo proposicional intuicionista é um subsistema do clássico, isto é, todas as fórmulas prováveis no sistema intuicionista são prováveis no clássico. Aquelas de nossos resultados, enunciados oficialmente envolvendo "|--" (incluindo aquelas que nós primeiro declaramos co "|=") que não estão prontamente estabelecidas para o sistema intuicionista também são identificadas por "o". [KLE68]

5.2 Intuicionismo: teoria de tipos de Martin-Löf

Este capítulo é dedicado à lógica (e à matemática) intuicionista e a uma aplicação particular dela à ciência da computação. A aplicação mencionada é a teoria de tipos de Martin-Löf. De fato, é um tanto quanto estranho referir-se à teoria de tipos como uma aplicação pois ele ela nos apresenta um FRAMEWORK unificado no qual implementar a atividade de especificação, construção e verificação de programas. Em grande parte deste capítulo, é mostrada por cima uma exposição da teoria de Martin-Löf, mas primeiro parece prudente falar um pouco sobre a concepção intuicionista de lógica e matemática.

Tanto para Frege como para Russell, e para os lógicos clássicos em geral, a lógica é a mais fundamental e geral de todas as teorias. Essa idéia é básica, por exemplo, para o programa LOGISTIC pois, de acordo com ela, a matemática estava carente de uma justificativa por baixo, e essa justificativa estava para ser fornecida pela própria lógica. Os intuicionistas, por outro lado, DRAW uma MORAL diferente do fato de que a matemática clássica estava carente de justificativa. Sob o seu ponto de vista, a matemática é primária, e a lógica é secundária: a lógica é nada mais do que uma coleção de regras descobertas, *a posteriori*, as quais acontecem de caracterizar os padrões de inferência implícitos na matemática quando corretamente PRACTISED. A expressão "corretamente PRACTISED" é significativa pois, adicionalmente as suas visões radicais sobre a relação entre matemática e lógica, eles mantêm uma visão um tanto não-usual sobre a natureza da própria matemática. Primeiro, números são entidades mentais. De acordo com Brouwer, eles são construídos fora da "sensação de tempo". Matemática é essencialmente uma atividade mental. Segundo, somente objetos "construíveis" são legítimos na matemática intuicionista. Isso exclui, por exemplo, quase todas as hierarquias cumulativas de conjuntos. De fato, o intuicionismo não admite totalidades infinitas completas que não sejam, em algum sentido, construíveis. Além disso, todas as provas das asserções matemáticas devem ser construtivas. Essa visão tem um pouco de

efeito debilitante sobre a matemática; certamente, nem toda a matemática clássica é intuicionistamente aceitável. De fato, antes da contribuição de Bishop (Bishop 1967), muitos dos resultados de análises clássicas parecem ser intuicionistamente não-obtíveis.

Que impacto essa visão tem sobre a própria lógica? Dado que a lógica é vista como um outro empreendimento, e que a lógica tem como significado refletir corretamente a prática matemática, não é surpreendente que o impacto seja considerável. Por exemplo, considere a asserção

(∃n) (n é um número ímpar perfeito)

Essa asserção será provável somente se nós formos capazes de *construir* um número que seja tanto ímpar quanto perfeito. Ela será não-provável se essa suposição levar, via uma *prova construtiva*, a uma contradição. Mas como nós nem temos um meio de construir tal número, nem podemos estabelecer por meios construtivos que essa suposição leva a uma contradição, a asserção é intuicionistamente problemática. Tal consideração leva os intuicionistas a rejeitar a lei do terceiro excluído, A∨¬A, que é uma lei fundamental da lógica clássica.

Tanto em seu livro como em outros escritos seus, Dummett forneceu-nos uma perspectiva um tanto quanto diferente sobre a natureza da discordância entre lógicos intuicionistas e lógicos clássicos. A exposição de Dummett relata as diferentes teorias de significado empregado pelas duas escolas. Para o matemático clássico, o significado de uma sentença é para ser provado por suas condições-verdade. Isso, por exemplo, é refletido na avaliação de Tarski da semântica para o cálculo de predicados. A suposição que baseia a matemática e a lógica clássicas é que as asserções matemáticas são determinadamente ou verdadeiras ou falsas. Para os intuicionistas, o significado de uma declaração reside não em suas condições-verdade, mas preferivelmente em seu significado de verificação ou prova. Essa distinção é refletida na avaliação dos operadores lógicos fornecidos na próxima seção. Seu significado não é fornecido por condições-verdade, como em quase todos os outros capítulos deste livro, mas pela especificação de o que é para contar como uma prova de uma sentença contendo-os.

Esta seção não pretende ser, obviamente, uma exposição dos fundamentos filosóficos do intuicionismo. Nós apenas tentamos dar ao leitor conhecimento suficiente para seguir nossa exposição da teoria de Martin-Löf. Para uma discussão detalhada dos fundamentos filosóficos do intuicionismo, o leitor pode fazer não mais do que consultar o excelente livro de Dummett [TUR84].

5.3 A Interpretação Intuicionista das Constantes Lógicas

Classicamente, o significado de cada constante lógica é fornecido pela especificação de suas condições-verdade para qualquer sentença em que essa constante seja o conectivo principal. Intuicionistamente, o significado de cada constante lógica é para ser GLEANED de uma especificação de o que é contar como uma *prova* para qualquer sentença envolvendo a constante como seu conectivo principal. Nós resumimos o significado intuicionista das várias constantes lógicas na seguinte tabela.

UMA PROVA DE	CONSISTE DE
A∨B	Uma prova de A ou uma prova de B

A&B	Uma prova de A e uma prova de B
$A \rightarrow B$	Uma construção que transforma qualquer dada prova de A em uma prova de B
$\neg A$	Uma prova de $A \rightarrow \perp$, em que $\perp$ é alguma declaração absurda, por exemplo, (0=1)
$\exists x\ A(x)$	Uma construção de A(c) para algum indivíduo c
$\forall x\ A(x)$	Uma construção que, quando aplicada a qualquer indivíduo c, produz uma prova de A(c)

Esse guia é bastante sucinto, então vamos ver se podemos melhorar o assunto um pouco.

(i) *Disjunção*

Uma prova de $A \vee B$ é para consistir de uma prova de A ou uma prova de B junto, presumivelmente, com uma indicação de qual. Desse modo, o conjunto de provas de $A \vee B$, $P(A \vee B)$, pode ser representado como a *união disjunta* de P(A) e P(B), em que P(A) é o conjunto de provas de A, etc. Nós escreveremos a união disjunta de P(A) e P(B) como P(A)+P(B). Os elementos de P(A)+P(B) são, portanto, da forma i(a) e j(b) em que $a \in P(A)$ e $b \in P(B)$, e i, j são as funções injetoras, $i:P(A) \rightarrow P(A)+P(B)$ e $j:P(B) \rightarrow P(A)+P(B)$.

(ii) *Conjunção*

Uma prova de A&B é para consistir de um par cuja primeira componente é uma prova de A e cuja segunda componente é uma prova de B. Em outras palavras, o conjunto de provas de A&B é para consistir do *produto cartesiano* de P(A) e P(B), denotado por $P(A) \times P(B)$.

(iii) *Implicação*

De acordo com a tabela, uma prova de $A \rightarrow B$ é para consistir de uma construção que, quando aplicada a uma prova de A, produz uma prova de B. Colocado diferentemente, $P(A \rightarrow B)$ é o conjunto das construções (funções construtivas) que, quando aplicadas a elementos de P(A), produzem elementos de P(B). O conjunto $P(A \rightarrow B)$ é, portanto, a *função espaço* das funções construtivas de P(A) para P(B), que nós denotamos por $P(A) \rightarrow P(B)$.

(iv) *Negação*

A negação é definida em termos da implicação. Para provar $\neg A$, é suficiente provar que A leva a uma contradição. Portanto, $P(\neg A)$ é o conjunto das funções construtivas de P(A) para $P(\perp)$, em que $P(\perp)$ é o conjunto de provas vazio.

(v) *Quantificação existencial*

Uma prova de $\exists x\ A(x)$ consiste de um indivíduo c e de uma prova de A(c). (Estritamente falando, alguém deveria também insistir para que c seja dado por alguma construção. Nós retornaremos a esse ponto mais tarde.) $P(\exists x\ A(x))$, portanto, consiste de pares ordenados <c,d> em que c é um indivíduo e d é a prova de A(c). Alternativamente, $P(\exists x\ A(x))$ é a união disjunta da família P(A(c)), indexada pelo

domínio de indivíduos C. Nós escrevemos essa união disjunta como $(\Sigma c \in C)P(A(c))$.

(vi) *Quantificação universal*

Uma prova de $\forall x\ A(x)$ é uma construção ou função efetiva que, quando aplicada a um indivíduo c produz uma prova de A(c). O conjunto $P(\forall x\ A(x))$ é, portanto, o produto cartesiano da família P(A(c)), indexado pelo domínio de indivíduos C. Nós escrevemos esse produto cartesiano como $(\Pi c \in C)P(A(c))$.

Nós podemos resumir tudo isso bastante sucintamente como segue:

$$P(A \vee B) = P(A)+P(B)$$

$$P(A\&B) = P(A)\times P(B)$$

$$P(A \rightarrow B) = P(A) \rightarrow P(B)$$

$$P(\exists x\ A(x)) = (\Sigma c \in C)P(A(c))$$

$$P(\forall x\ A(x)) = (\Pi c \in C)P(A(c))$$

Isso deveria fazer as coisas um tanto quanto mais explícitas, mas há ainda duas áreas de VAGUENESS. Primeiramente, o que, exatamente, é uma construção ou uma função construtiva, e o que constitui um domínio de indivíduos ou um domínio de quantificação? Martin-Löf fornece uma resposta para essas questões em termos de sua teoria de tipos.

A teoria de tipos foi desenvolvida como uma formalização da matemática construtiva. Ela pretende ser um sistema de escala-completa para formalizar o desenvolvimento da matemática intuicionista, por exemplo, em Bishop (1967). A linguagem da teoria confirma a ocorrência de provas com proposições, e como conseqüência, proposições podem expressar propriedades de provas. Essa facilidade tem importantes conseqüências para a teoria de tipos como uma linguagem de programação. Nós agora reexaminamos o significado intuicionista das constantes lógicas e DEPLOY a discussão para apresentar a teoria de Martin-Löf.

Seja A uma proposição qualquer. O conjunto P(A) é o conjunto das provas de A que, segundo Martin-Löf, nós podemos referenciar como *os objetos do tipo A*. Martin-Löf, influenciado pelo trabalho de Curry, Howard e Scott, aproveitou o passo muito natural de *identificar tipos com proposições*. Quais implicações essa identificação tem para o significado das constantes lógicas?

No caso da disjunção, nós devemos identificar $A \vee B$ com P(A+B), mas como P(A+B)=P(A)+P(B), e, por indução, P(A) é para ser identificado com A, e P(B) com B, nós estamos para identificar a proposição $A \vee B$ com a união disjunta A+B. Similarmente, A&B é para ser identificada com o produto cartesiano $A \times B$. Para a implicação nós devemos identificar a proposição $A \rightarrow B$ com $A \rightarrow B$, o tipo de funções de A para B. A respeito da negação, a proposição $\neg A$ é identificada com $A \rightarrow \varnothing$, em que $\varnothing$ é o tipo vazio.

Isso nos leva a considerar os quantificadores. Mais cedo, nós levantamos uma questão sobre a natureza dos domínios da quantificação. Na matemática intuicionista, a quantificação é restrita àqueles domínios que são 'GRASPABLE' (emprestando uma expressão de Dummett). Para Martin-Löf, domínios de quantificação são tipos. Subseqüentemente, nós nunca empregamos um quantificador irrestrito, mas preferivelmente quantificações da forma $\exists x \in A$ ou $\forall x \in A$, para ser entendido como "existe um objeto do tipo A" e "para todos os objetos do tipo A", respectivamente. Nós devemos agora modificar nossa avaliação dos quantificadores para permitir esse refinamento.

Uma prova de $(\exists x\in A)B(x)$ consiste de um par <a,b> cuja primeira componente é um objeto do tipo A e cuja segunda é uma prova de B(a). Em outras palavras, $P((\exists x\in A)B(x))$ é a união disjunta da família P(B(a)), indexada pelos elementos do tipo A. Sob a identificação de proposições com tipos isso significa que nós estamos para identificar $(\exists x\in A)B(x)$ com a *união disjunta* $(\Sigma x\in A)B(x)$. A quantificação universal é para sofrer uma transformação similar. Uma prova de $(\forall x\in A)B(x)$ é uma função efetiva que para cada objeto a do tipo A produz um objeto do tipo B(a). Colocado diferentemente, $P((\forall x\in A)B(x))$ é o produto cartesiano da família P(B(a)), indexado pelo tipo A. Empregando a identificação, nós estamos instruídos para identificar a proposição $(\forall x\in A)B(x)$ com o *produto cartesiano* $(\prod x\in A)B(x)$.

Nós discutimos a questão concernente aos domínios de quantificação, mas nada dissemos sobre a definição de construções/funções construtivas. Martin-Löf fornece uma avaliação precisa de tais funções empregando regras de inferência (em um sistema de dedução natural) que especificam o que os objetos de cada tipo são. Nós retornaremos rapidamente a ISSUE de como os tipos e os objetos de vários tipos são especificados.

O leitor deve agora ter alguma idéia de o que a teoria de tipos envolve. Nas duas próximas seções, nós cobrimos o básico com mais detalhes, e na seção final nós discutimos a linguagem da teoria de tipos. Isso vem em duas partes intimamente relacionadas: a linguagem de tipos mesmo e a linguagem dos objetos dos vários tipos. [TUR84]

5.4 A Linguagem da Teoria de Tipos

Essencialmente, expressões na teoria de tipos são construídas a partir de variáveis e constantes por meio de aplicação e abstração funcional. A linguagem não é, entretanto, livre de tipos no sentido do cálculo lambda. Há restrições adicionais sobre quais formas de expressão são bem-formadas. Nordström & Smith (1983) usam a noção de "aridade" para impor as restrições apropriadas sobre as quais as formas de aplicação são legítimas. Grosso modo, a aridade de uma expressão descreve a funcionalidade da expressão, na qual ela indica a aridade da expressão para a qual pode ser significativamente aplicada. Uma expressão *saturada* é aquela que não pode ser aplicada a coisa alguma.

Nós agora descrevemos a linguagem de expressões um pouco mais precisamente. Fazendo isso nós seguimos a apresentação de Nordström & Smith (1983).

1. Uma variável é uma expressão
2. Uma constante é uma expressão
3. Se $e_1, ..., e_n$ são expressões de aridades apropriadas, então a aplicação $e(e_1, ..., e_n)$ é uma expressão. A intenção aqui é que a aridade de e deve ser tal que ela possa ser aplicada a expressões que tenham a aridade das expressões e1, ..., e_n.
4. Se $x_1, ..., x_n$ são variáveis e e é uma expressão, então a abstração $(x_1, ..., x_n)e$ é uma expressão. A expressão resultante possui uma aridade que é tal que, quando é aplicada a expressões que tenham a mesma aridade que as variáveis $x_1, ..., x_n$, ela produz uma expressão com a mesma aridade que e.

Constantes vêm em dois tipos: constantes primitivas e constantes definidas. Exemplos das anteriores são o, succ, λ (λ-operador-de-abstração). Constantes definidas são introduzidas por definições explícitas como

$$f(x_1, ..., x_n) = b$$

em que a constante f é introduzida, e f, quando aplicada às variáveis x_1, ..., x_n, é por definição igual a b. Isso é muito parecido com definição funcional em linguagens de programação.

Será construtivo considerar uns poucos exemplos em mais detalhe. Considere as expressões $(\Sigma x \in A)B$ e $(\prod x \in A)B$ introduzidas na seção 4.2. Para conformar com a descrição acima, nós precisamos apresentar estas como aplicações e/ou abstrações. A expressão $(\Sigma x \in A)B$ de fato consiste de duas partes: uma expressão denotando um tipo A e uma expressão da forma (x)B(x), na qual a variável x é amarrada. Em conformidade com a sintaxe acima, a expressão deveria ser escrita como Σ(A,(x)B x)). Similarmente, $(\prod x \in A)B$ deveria ser relançada como P(A, (x)B(x)). No que segue, entretanto, freqüentemente empregaremos a notação mais familiar para muitas expressões.

Nós agora fornecemos uma lista de algumas das formas de expressão mais comuns de serem encontradas na teoria dos tipos.

Canônica	**Não-canônica**
$\prod$(A,(x)B(x)),(λx)b	c(a)
Σ(A,(x)B(x)),(a,b)	(Exy)(c,d)
+(A,b),i(a),j(b)	(Dxy)(c,d,e)
I(A,a,b),r	J(c,d)

Os termos “canônica” e “não-canônica” empregados na tabela acima se relacionam ao significado de expressões na teoria dos tipos. Tanto em lógica e matemática clássica como em ciência da computação, o significado de uma expressão ou de um programa é fornecido por uma definição semântica associada. Na lógica clássica de primeira ordem, por exemplo, a semântica é fornecida via a noção de um modelo da noção de satisfação de Tarski. Em linguagens de programação, uma técnica de definição padrão emprega os domínios semânticos da teoria da computação de Dana Scott. Está implícita em ambos a suposição de que o significado de um objeto sintático é para ser fornecida via certas estruturas matemáticas que definitivamente têm sua existência garantida pela ONTOLOGY de padrão da teoria dos conjuntos de Zermelo-Fraenkel (Krivine 1971). A teoria de tipos pretende ser uma fundamentação para a matemática construtiva e é, portanto, simplesmente perversa ao exigir uma explanação da linguagem da teoria de tipos em termos da teoria de conjuntos. Ao invés disso, na teoria de tipos o significado de uma expressão é fornecido por uma regra de computação.

O procedimento mecânico de computar o valo de uma expressão é chamado *avaliação*. Uma expressão que tem ela própria como valor é chamada *canônica* ou normal. Uma expressão que não é canônica é chamada *não-canônica*. O valor de uma expressão não-canônica $f(e_1, ..., e_n)$ é uma expressão canônica obtida seguindo as regras de computação para a constante f. Na teoria de tipos, somente a forma f de uma expressão $f(e_1, ..., e_n)$ determina se a expressão é canônica ou não. Então, por exemplo, Succ(e) é canônica independentemente da forma de e. A razão para isso concerne à natureza da avaliação. Em geral, não é viável exigir que todas as partes $e_1, ..., e_n$ de uma expressão canônica sejam avaliadas pois algumas partes da expressão podem não ter valor. Uma expressão está *completamente avaliada* se é canônica e todas as partes saturadas estão

completamente avaliadas.

Para ilustrar o processo de avaliação, nós indicamos como as expressões não-canônicas em nossa tabela podem ser avaliadas.

Para executar uma expressão de e1(e2), nós primeiro executamos e1. Se nós obtivermos (λx)b como resultado, continuamos executando b(e2|x) (isto é, b com e2 substituído por x). Para executar ou avaliar (Exy)(c,d), nós primeiro executamos c. Se resultar (a,b), executamos d(a,b|x,y) (substituir a por x e b por y em d). Para avaliar D(x,y)(c,d,e) nós primeiro executamos c. Se obtivermos i(a) como resultado, executamos d(a|x). Se, alternativamente, obtivermos j(b) como o resultado da execução de c, executamos e(b|y) ao invés disso. Para executar J(c,d), primeiro executamos c. Se resultar r, executamos d.

Resumindo, expressões na teoria de tipos são construídas a partir de variáveis e constantes por meio de abstração e aplicação. O significado não é fornecido por meio de uma definição semântica associada, mas por meio de regras de avaliação. [TUR84]

5.5 Julgamentos e Regras de Inferência

O cerne da teoria de Martin-Löf concerne a fazer julgamentos da forma

`(i)` A é um tipo

`(ii)` A e B são tipos iguais

`(iii)` a é um objeto do tipo A

`(iv)` a e b são objetos iguais do tipo A

que são abreviados como (i) tipo a, (ii) A=B, (iii) a∈A, e (iv) a=b∈A respectivamente. Os tipos podem ser identificados com conjuntos, proposições ou tarefas. Conseqüentemente, cada um dos quatro tipos de julgamento admite três leituras intuitivas diferentes. O terceiro julgamento, por exemplo, pode ser entendido como

a é um elemento do conjunto A,

a é uma prova da proposição A,

a é um programa para a tarefa A.

É claro que a terceira interpretação é a relevante para a ciência da computação. Essa leitura é significada apenas como uma ajuda ao nosso entendimento; elas não pretendem fixar os significados das várias formas de julgamento em nenhum sentido formal. Isso é alcançado pelas regras de inferência do sistema e, em particular, as tão chamadas regras de introdução.

Martin-Löf, em 1982, definiu tipos canônicos da seguinte forma:

Um tipo canônico A é definido prescrevendo como um objeto canônico de A é formado, assim também como de que maneira dois objetos iguais de um tipo A são formados.

Tais prescrições são fornecidas pelas regras de introdução para os vários tipos. Por exemplo, considere a construção do produto cartesiano sobre tipos introduzido na seção 4.2. De acordo com a explanação acima, nós precisamos fornecer uma regra para prescrever como um objeto canônico de A é formado, além de como alguém que mostra como dois objetos canônicos do tipo A são formados. Martin-Löf chamou as regras relevantes de ∏-introdução.

(x∈A)	(x∈A)

$b \in B$	$b=d \in B$
$(\lambda x)b \in (\Pi x \in A)B$	$(\lambda x)b=(\lambda x)d \in (\Pi x \in A)B$

A expressão entre parênteses $(x \in A)$ representa uma premissa ou hipótese, e, em geral, os julgamentos na teoria dos tipos são feitos em relação a tais hipóteses. Nós fazemos referência, para o leitor, a Martin-Löf (1982) para detalhes.

Como uma ilustração adicional, considere a(s) regra(s) para a construção da união disjunta (Σ-introdução):

$a \in A \quad B \in b(a\|x)$	$a=c \in A \quad b=d \in B(a\|x)$
$(a,b) \in (\Sigma x \in A)B$	$(a,b)=(c,d) \in (\Sigma x \in A)B$

A primeira regra estipula como um objeto canônico de $(\Sigma x \in A)B$ é formada, enquanto a segunda informa-nos como dois objetos de tipos iguais são formados. Em todas essas regras, a relação de igualdade entre objetos é exigida para ser uma relação de equivalência.

Tais definições para os tipos canônicos habilitam-nos a fornecer uma avaliação mais formal dos vários julgamentos. Por exemplo, um julgamento da forma do tipo A significa que A tem um tipo canônico como valor, enquanto o julgamento $a=b \in A$ significa que a e b, quando avaliados, têm como valores objetos canônicos iguais do tipo canônico denotado por A. Essa definição faz sentido perfeito, pois, por hipótese, A tem um tipo canônico como valor (é isso que significa ser um tipo pela definição anterior) e pela definição de tipos canônicos nós sabemos como dois objetos canônicos do mesmo tipo são formados.

Um segundo tipo de regra de inferência no sistema é uma "imagem-espelhada" da regra de introdução. Novamente, ilustramos por referência à construção do produto cartesiano para tipos.

Π-eliminação

$$\frac{c \in (\Pi x \in A)B \qquad a \in A}{c(a) \in B(a|x)} \qquad \frac{c=f \in (\Pi x \in A)B \qquad a=d \in A}{c(a)=f(d) \in B(a|x)}$$

Como antes, há duas partes para a eliminação das regras, a primeira informa-nos como formar uma aplicação, e a segunda, como duas aplicações iguais são formadas. Essas regras estavam implícitas em nossa discussão da quantificação universal: a expressão c denota uma função que, para cada elemento do tipo A, associa um objeto c(a) do tipo B(a|x).

Há regras similares governando a união disjunta e o produto cartesiano de dois tipos, de tipos finitos e do tipo dos números naturais. Se tipos adicionais são adicionados (por exemplo, tipos enumerados, listas, etc.), então as regras que governam esses tipos serão adicionadas quando exigidas. Adicionalmente a essas regras para os vários tipos, o sistema de Martin-Löf inclui regras mais gerais relativas à igualdade de objetos e tipos (por exemplo, elas têm que ser relações de equivalência).

Como uma ilustração da sua teoria, Martin-Löf oferece uma prova do axioma da

escolha. Ele começa com as premissas

tipo A

tipo B (x∈A)

tipo C (x∈A, y∈B)

Usando a abreviação A→B para (Πx∈A)B (x não livre em B), a expressão E,

(Πx∈A)(Σy∈B)C→(Σf∈(Πx∈A)B)(Πx∈A)C(f(x)|y)

é um tipo que, quando entendido como uma proposição, expressa o axioma da escolha. Usando as regras de prova e as suposições acima, ele termina com uma conclusão da forma

(λz)((λx)p(z(x)),(λx)q(z(x)))∈E

Basicamente, a estratégia de prova é aplicar as regras de eliminação para separar o antecedente da expressão E, e as regras de introdução para construir o conseqüente. A expressão

(λz)((λx)p(z(x)),(λx)q(z(x)))

constitui a prova exigida para o axioma da escolha. [TUR84]

5.6 A Teoria de Tipos como uma Linguagem de Programação

A relevância da teoria de Per Martin-Löf para a ciência da computação relaciona-se com a terceira forma de julgamento na teoria. Considere o julgamento a∈A. Sob a terceira interpretação, A é para ser interpretado como um problema ou tarefa (ou uma *especificação* desses) e a é um *programa* para sua solução. A especificação é uma definição de tipo, e o programa é um objeto que encontra/conhece a especificação. Por isso, a separação natural entre expressões que denotam tipos e aquelas que se referem a objetos dos vários tipos define uma linguagem de especificação de um lado e uma linguagem de programação do outro.

A real linguagem de programação é uma linguagem funcional pura, e na aparência assemelha-se a uma versão do cálculo lambda. As regras de inferência, que entre outras coisas, definem a gramática dessa linguagem, não permitem auto-aplicações, e então a linguagem tem como base de versão polimorficamente-tipada do cálculo lambda.

Uma outra diferença significativa entre a teoria de tipos como linguagem de programação e linguagens mais convencionais é que é permitida apenas recursão primitiva. Na teoria de tipos, todas as funções são *totais*. Apesar disso, é permitido que alguém defina funções de ordem mais alta por recursão primitiva (veja Smith (1982)).

Para ilustrar o uso da teoria de tipos tanto como linguagem de especificação quanto como linguagem de programação, nós aplicamos um exemplo devido a Nordström & Smith (1983). Em seus exemplos, o programa alvo deve gerar um índice KWIC (palavra-chave no contexto). O programa, dada uma lista de títulos e uma lista de palavras não-significativas, deve produzir uma lista das rotações significativas dos títulos ordenada alfabeticamente, em que uma rotação significativa é um rearranjo cíclico das expressões nas quais a primeira palavra é significativa. Nordström & Smith produzem uma especificação formal dessa tarefa da seguinte maneira:

Primeiro eles assumem que temos certos tipos enumerados definidos, denominados *CaracterImprimível*, que é um subconjunto do tipo enumerado *ASCII*. Eles então definem via definição explícita

Título = Lista(Palavra)

Palavra = Lista(CaracterImprimível)

Uma *rotação* de uma lista é um rearranjo cíclico dos elementos na lista, os quais podem ser definidos como

rotação(y, t) = $(\exists n \in N)(Shift^n(y) = t)$

em que

shift(nil) = nil

shift(a,s) = s <> (a,nil)

e

$f^0(x) = x$

$f^{n+1}(x) = f(f^n(x))$

e <> é o operador de concatenação óbvio. Eles então apresentam a noção de *rotação significativa*

rotSign(y,t,n) = rotação(y,t) & primeiro(y) $em_{PALAVRA}$ n)

em que

a em_A nil = $\perp$

a em_A b.s = $[a =_A b] \vee (a\ em_A\ s)$

e $\perp$ quer dizer absurdo, isto é, uma proposição sem prova.

No exemplo, a saída é ordenada com respeito à ordem lexicográfica, $<_A$, entre os títulos, por

$lex(A)(<_A)(nil,nil) = T$

$lex(A)(<_A)(a.s,nil) = \perp$

$lex(A)(<_A)(nil,b.t) = t$

$lex(A)(<_A)(a.s,b.t) = [a =_A b]$ & $lex(A)(<_A)(s,t) \vee [a =_A b]$ & $(a <_A b)$

em que T = {.}.

Em outras palavras, se < é uma ordem entre os caracteres imprimíveis, então lex(CaracterImprimível)(<) é uma ordem sobre as palavras, e lex(palavra)(lex(CaracterImprimível)) é uma ordem sobre os títulos. Acima, T é a proposição que é sempre verdadeira e é representada por um conjunto com um elemento.

A proposição Ordenado'(x) declara que a lista de títulos está ordenada de acordo com a ordem lexicográfica, e Nordström & Smith definem isso como segue:

Ordenado'(x) = Ordenado(Lex(Palavra)(Lex(CaracterImprimível)(<)),x)

em que

Ordenado(0,x)

é definido por

Ordenado(0,nil) = T

Ordenado(0,a.nil) = T

Ordenado(0,a.b.s) = 0(a,b) & Ordenado(0,b.s)

Com tudo isso, alcançar a especificação real é relativamente direto:

$$\exists \left\{ \begin{array}{l} \lceil \quad (\forall t \in \text{Lista(Título)})(\forall n \in \text{Lista(Palavra)}) \\ \quad\quad (\exists x \in \text{Lista(Título)}) \\ \lfloor \quad (\text{Ordenado'}(x) \;\&\; (\forall y \in \text{Título})(y \text{ em } x \leftrightarrow \\ \qquad\qquad (\exists z \in \text{Título})((z \text{ em } t) \;\&\; \text{RotSign}(y,z,n)))). \end{array} \right.$$

Se nós aplicamos a maquinaria de inferência da teoria de tipos para provar essa proposição/especificação, nós obtemos uma função f cujo tipo é dado pela especificação acima. Em outras palavras, nós terminamos com um julgamento da forma $f \in E$. Além disso, a função f, quando aplicada a uma lista de títulos t e a uma lista n de palavras, dá um par, a primeira componente do que é o resultado exigido, isto é, p(ap(ap(f,t),n)). Isso fornece o índice KWIC de t com respeito à lista n de palavras não-significativas.

Os benefícios da abordagem de Martin-Löf são, em princípio, muito consideráveis. Debaixo do guarda-chuva de uma teoria, existe uma linguagem de especificação, uma linguagem de programação (que é puramente funcional em essência), e um meio formal de derivar programas de suas especificações. Esse processo de derivação não é mecânico, naturalmente; alguém tem que construir uma prova de um julgamento da forma $a \in A$. Apesar de tudo, o processo de construir a prova não só facilita a derivação do programa a partir de sua especificação, mas também fornece uma prova de que o programa derivado encontra as especificações. Tudo isso deveria ser visto em contraste com a metodologia adotada nas provas de correção de tipos de Hoare. Neste início, não só há duas linguagens relativamente distintas (a linguagem de programação e a linguagem lógica na qual as provas de correção são executadas), mas também há uma disparidade maior entre o processo de construção de programa e a verificação. Para ser favorável aos discípulos do estilo de construção e de verificação de programas de Hoare, deveríamos adicionar que a filosofia por trás de sua abordagem é trazer junto esses dois processos (por exemplo, o uso de invariantes na construção de programas), mas parece que tanto as ferramentas formais disponíveis quanto as linguagens de programação nas quais elas estão aplicadas tendem a aliviar contra essa aspiração. Como conclusão, a teoria de Martin-Löf oferece uma perspectiva animadora para uma visão unificada da especificação, construção e verificação de programas. [TUR84]

5.7 Teoria de Tipos como Categorias

Desejamos formalizar a teoria de Martin-Löf usando teoria das categorias. Definimos a categoria **TML** como [CAM2001] :

Objetos Conjuntos de provas;

Morfismos Mapeamentos entre provas (funções construtivas);

Composição Composição de funções construtivas;

Associatividade Trivial, pela associatividade da composição de funções.

Observe que **TML** é uma subcategoria não plena e não larga de **Set**.

As Figuras 5.1, 5.2, 5.3, 5.4 e 5.5 ilustram essas provas expressas em teoria das categorias.

Na Figura 5.1, representamos as provas de $P \vee Q$ como a soma de Provas(P) e Provas(Q), onde i e j são as respectivas funções de imersão.

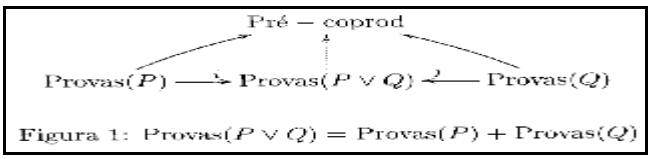


Figura 5.1 Provas(*P* Ú *Q*) = Provas(*P*) + Provas(*Q*)

Na Figura 5.2, representamos as provas de $P \wedge Q$ como o produto de Provas(P) e Provas(Q), onde π_1 e π_2 são as respectivas projeções.

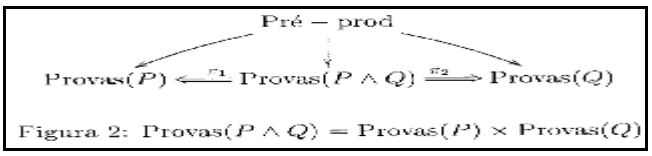


Figura 5.2 Provas(*P* Ù *Q*) = Provas(*P*) ´ Provas(*Q*)

A Figura 5.3 ilustra as provas de $P \rightarrow Q$ como morfismos f (funções construtivas) de Provas(P) para Provas(Q).

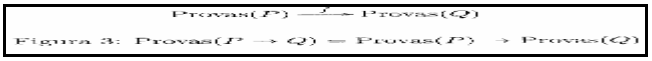


Figura 5.3 Provas(*P* ® *Q*) = Provas(*P*) ® Provas(*Q*)

A Figura 5.4 ilustra as provas de $(\exists x \in D)\ P(x)$ como a soma finita indexada pelo domínio D.

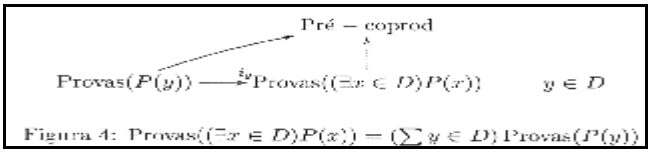


Figura 5.4 Provas((Sx Î *D*) *P*(*x*)) = (S*y* Î *D*) Provas(*P*(*y*))

Finalmente, a Figura 5.5 ilustra as provas de $(\forall x \in D)P(x)$ como o produto finito indexado pelo domínio D.

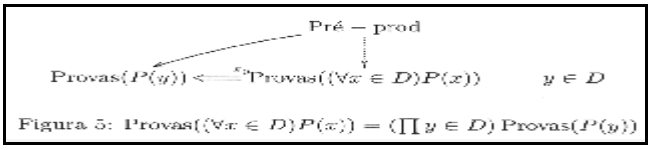


Figura 5.5 Provas((" *x* Î *D*) *P*(*x*)) = (P*y* Î *D*) Provas(*P*(*y*))

Um exemplo de diagrama é ilustrado na Figura 5.6. Nesse caso, o exemplo mostra a prova de $((\forall x \in D)P(x)) \rightarrow P(a)$, com $a \in D$ (instanciação universal). Na Figura, f é o morfismo que representa as funções construtivas que transformam provas de $((\forall x \in D)P(x))$ em provas de $P(a)$.

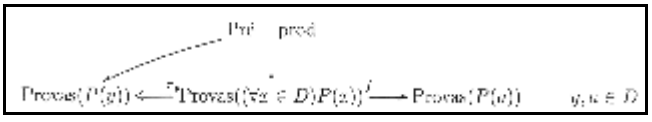


Figura 5.6 Exemplo de inferência

Este capítulo apresentou os fundamentos e uma aplicação da lógica intuicionista em ciência da computação. Essa lógica é usada como suporte para a teoria de tipos de Martin-Lõf, cuja principal aplicação é a especificação, construção e verificação formal de programas. Além disso, propomos uma forma diagramática de apresentar as inferências usando-se teoria das categorias. Concluímos que o uso de categorias resultou em uma forma mais intuitiva de trabalhar com esta teoria. Sua aplicação parece ser de valor, pois permite uma melhor visualização das inferências feitas.

6 Programação Concorrente Formalizada por Lógica Categorial

6.1 Relação da Especificação com os Comandos

Nesta seção são apresentadas as relações entre a linguagem de descrição utilizada e os três formalismos escolhidos.

Comando Condicional

Linguagem de descrição

se <condição> **então**

<comando1>

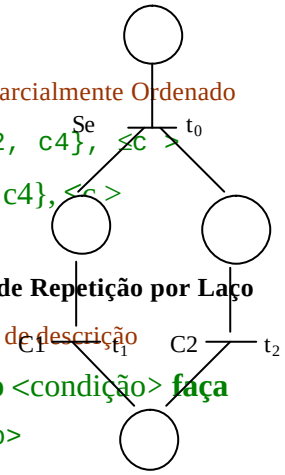
[**senão** <comando2>]

CCS

if E *then* P *else* Q

onde E é a condição, P o comando1 e Q o comando2.

Rede de Petri



Conjunto Parcialmente Ordenado

<{c1, c2, c4}, ≤c >

<{c1, c3, c4}, ≤c >

Comando de Repetição por Laço

Linguagem de descrição

enquanto <condição> **faça**

<comando>

CCS

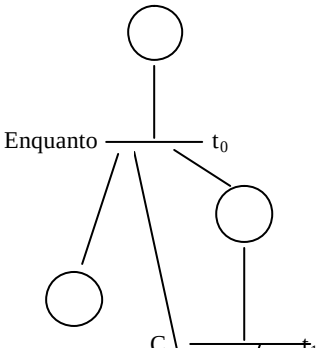
enquanto = expr.enquanto

expr = E.comando + E.fim

fim = 0

E é a condição

Rede de Petri



Conjunto Parcialmente Ordenado

<{c1, c2, c4}, ≤c >

<{c1, c4}, ≤c >

Comando de Repetição Iterativo

Linguagem de descrição

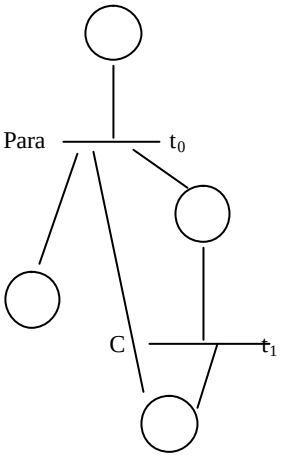
para <variável> = valor_inicial **até** valor_final **faça**

<comando>

CCS

$$\sum_{i \in I} E_i$$

Rede de Petri



Conjunto Parcialmente Ordenado

<{c1, c2, c4}, ≤c >

<{c1, c4}, ≤c >

Comando de Comunicação - Envio

Linguagem de descrição

envia (X, i)

CCS

$E \overset{def}{=} \overline{X} . E$

Rede de Petri

Conjunto Parcialmente Ordenado

Comando de Comunicação – Recebimento

Linguagem de descrição

recebe (Y, j)

CCS

$E \overset{def}{=} X . E$

Rede de Petri

Conjunto Parcialmente Ordenado

Comando de Laço Paralelo

Linguagem de descrição

para valor_inicial ≤ variável ≤ valor_final **faça-paralelo**

<comando>

CCS

Rede de Petri

Conjunto Parcialmente Ordenado

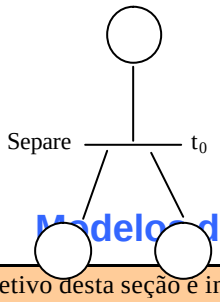
Comando de Subdivisão de Processo

Linguagem de descrição

separe (X, X_1, X_2)

CCS

Rede de Petri



Conjunto Parcialmente Ordenado

$X = X_1 + X_2$

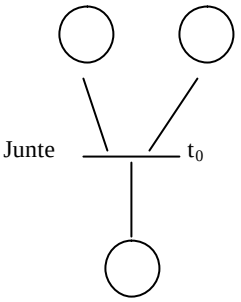
Comando de Junção de Processos

Linguagem de descrição

junte (X, Y, Z)

CCS

Rede de Petri



Conjunto Parcialmente Ordenado

$X = X_1 \times X_2$

Comando de Execução de Tarefa - Subrotina

Linguagem de descrição

compute <nome-de-tarefa>

compute [**recursivamente**] <nome-de-tarefa>

CCS

$E \overset{def}{=} \overline{X} .E$

Rede de Petri

Conjunto Parcialmente Ordenado

6.2 Modelos de Máquinas Paralelas

O objetivo desta seção é introduzir alguns modelos de máquinas paralelas. Ao contrário do caso seqüencial, não existe um modelo padrão e universal para processamento paralelo, uma vez que o desempenho de algoritmos paralelos depende de um conjunto de fatores que são dependentes de máquina. Entre esses fatores estão concorrência, alocação, alocação, organização e sincronização.

Serão abordados os modelos mais conhecidos e usados para desenvolvimento e análise de algoritmos. Estes modelos estão baseados em grafos acíclicos direcionados, memória compartilhada e redes.

- O modelo de grafos acíclicos direcionados - podem ser utilizados para representar

computações paralelas de forma natural e podem providenciar um modelo simples que não inclui nenhuma característica da arquitetura.

- O modelo de memória compartilhada, onde um número de processadores se comunicam através de uma memória global, proporciona o desenvolvimento de técnicas para

computação paralela.

- O modelo de redes, diferentemente dos anteriores, absorve a comunicação em sua topologia de interconexão.

6.3 Grafos Acíclicos Direcionados – (GAD)

Muitas computações podem ser expressas por grafos acíclicos direcionados de uma forma natural. Cada entrada é representada por um nodo que não tem vértices chegando nele (posição das entradas). Cada operação é representada por um nodo que tem vértices chegando nele, vindo dos nodos que representam seus operandos. O número máximo de vértices que podem chegar em um mesmo nodo é dois. Nodos de onde não saem vértices representam as saídas. Será assumido o critério do custo unitário onde cada nodo representa uma operação que consome uma unidade de tempo.

Um grafo acíclico direcionado com n nodos de entrada representa uma computação que não tem instruções ramos e que tem uma entrada de tamanho n. Portanto um algoritmo é representado por uma família de GAD { Gn }, onde Gn corresponde ao algoritmo com entrada de tamanho n.

Este modelo é particularmente útil para análise de computações numéricas, onde as instruções intermediárias são utilizadas para executar um certo número de vezes uma seqüência de operações, tantas vezes quanto forem necessárias, para representar apropriadamente o número de vezes.

Um GAD especifica as operações que devem ser executadas pelo algoritmo e implica em restrições de precedência na ordem em que estas operações devem ser executadas. É totalmente independente da arquitetura.

Exemplo

Considere o problema de computar a soma 5 de n=2k elementos de um array A. Dois possíveis algoritmos são representados para n=8. O primeiro algoritmo calcula somas parciais consecutivas, iniciando com A(l)+A(2), este resultado somado com A(3) e, assim, sucessivamente.

O segundo algoritmo calcula segundo uma árvore binária, a qual dois a dois, A(l)+A(2), A(3)+A(4) A(n-l)+A(n), obtendo parcelas. somadas duas a duas, tendo n/2 elementos. Repete-se o processo até que nodo raiz.

Neste capítulo são discutidas brevemente algumas questões relativas aos sistemas concorrentes. Apresentam-se a seguir dois exemplos extraídos da Literatura de lógicas temporais que se propõem a modelar o comportamento de sistemas concorrentes. Os conceitos básicos subjacentes a essas lógicas foram introduzidos nos capítulos anteriores deste trabalho.

6.4 Formalização de Sistemas Concorrentes

O estudo de sistemas concorrentes e distribuídos é uma área de pesquisa importante na Ciência da Computação. Um sistema concorrente consiste de um certo número de componentes autônomos que interagem entre si a fim de realizarem uma tarefa conjunta.

Sistemas concorrentes reais são usualmente compostos de vários processadores independentes, cada um deles executando um programa próprio. Em tais sistemas, a execução dos comandos nos diferentes processadores comumente se sobrepõe em vez de se intercalar. Informalmente, pode-se dizer que dois eventos são "concorrentes" se eles ocorrem sem uma ordem prévia estabelecida para suas ocorrências. Isso contrasta com os sistemas seqüenciais, em que quaisquer dois eventos que ocorrem em uma

computação estão obrigatoriamente ordenados.

A teoria dos sistemas concorrentes consiste da formulação de modelos matemáticos abstratos e do estudo das propriedades desses modelos. Uma das principais características dos sistemas concorrentes é a concorrência, ou seja, o fato de que há vários processos envolvidos cujos eventos podem ocorrer simultaneamente. Além da concorrência, dois outros aspectos de interesse da teoria dos sistemas distribuídos são a causalidade e a escolha:

- ⇨ a *causalidade* refere-se ao fato de que certos eventos em um sistema distribuído podem somente ocorrer em uma ordem fixa. Por exemplo, uma mensagem somente pode ser recebida após ter sido enviada. O recebimento da mensagem é dito ser *causalmente* dependente do envio da mesma.
- ⇨ a *escolha* captura o fato de que sistemas podem comportar-se de uma maneira indeterminada. Ou seja, em certos pontos da computação, o sistema pode escolher entre eventos alternativos, levando a diferentes comportamentos.

Considerando-se uma *linguagem de especificação* como sendo um formalismo no qual se especificam comportamentos dos sistemas sob estudo, evitando-se referências ao método ou detalhes de implementação, temos que uma linguagem de especificação para sistemas distribuídos é aquela em que podem ser descritas propriedades comportamentais dos sistemas distribuídos. É necessário que a linguagem de especificação permita que se combinem especificações simples a fim de construir especificações mais complexas – o que reflete a intuição sobre sistemas grandes poderem ser partidos em subsistemas menores e mais bem gerenciáveis.

Além disso, tratando-se de sistemas distribuídos, há de se esperar que a linguagem de especificação possa descrever propriedades como causalidade, escolha e concorrência. É necessário, pois, que se possam especificar os relacionamentos válidos entre os estados do sistema enquanto a computação procede. Vendo-se um programa como um gerador de um conjunto de computações, espera-se que a especificação de um programa forneça uma caracterização alternativa, preferencialmente mais descritiva e menos operacional, do conjunto de comutações geradas pelo programa. Esses requisitos sugerem o uso da lógica formal com conectivos booleanos e modalidades temporais como a linguagem de especificação desejada.

Sistemas de transições são meios adequados para a definição dos significados de programas. Entretanto, para muitos sistemas computacionais, especialmente os que envolvem concorrência, seu comportamento em andamento como seqüências de ações ou mudanças de estados é mais importante. Lógicas Temporais são projetadas para raciocinar sobre tal comportamento. [SOU99]

6.5 Aplicação da Teoria das Categorias à Especificação Formal de Sistemas Concorrentes

Os conceitos abordados até aqui formam a base teórica que permite estudar as aplicações da Teoria das Categorias à especificação formal de Sistemas Concorrentes. Serão apresentados alguns exemplos de tais aplicações.

CONJUNTOS PARCIALMENTE ORDENADOS ´ CONCORRÊNCIA

Exemplo 6.1 – *Conjuntos Parcialmente Ordenados* × *Concorrência* - Considere o seguinte trecho de programa seqüencial, baseado em linguagens de programação do tipo Pascal, na qual o símbolo **;** representa uma relação de dependência causal dos comandos c1, c2 e c3.

c1; c2; c3

Uma semântica de tal programa pode ser dada pelo seguinte conjunto parcialmente ordenado: ⟨{c1, c2, c3}, ≤c? onde c1 ≤c c2 e c2 ≤c c3 e, portanto, c1 ≤c c3. Mais precisamente, a relação de ordem parcial ≤c é tal que:

≤c = {(c1, c1), (c2, c2), (c3, c3), (c1, c2), (c2, c3), (c1, c3)}

De forma análoga, considere os seguintes trechos de programas:

p1 ; p2

q1 ; q2 ; q3

e as seguintes correspondentes semânticas;

⟨{p1, p2}, ≤p? onde p1 ≤p p2

⟨{q1, q2, q3}, ≤q? onde q1 ≤q q2 e q2 ≤q q3

Suponha que os três trechos de programa são concorrentes sem qualquer sincronização (são *independentes*). Então, a semântica de tal programa concorrente seria simplesmente o seguinte conjunto parcialmente ordenado (induzido pela operação de união disjunta de conjuntos): ⟨{c1, c2, c3} ∪ {p1, p2} ∪ {q1, q2, q3}, ≤c ∪ ≤p ∪ ≤q? ?

PRODUTO E COPRODUTO EM CONJUNTO PARCIALMENTE ORDENADO VISTO COMO CATEGORIA (WAIT, EXIT, FORK)

Produto em um Conjunto Parcialmente Ordenado ´ Sistemas Concorrentes e Primitiva fork

Conjuntos parcialmente ordenados constituem um modelo semântico para sistemas concorrentes. Assim, a existência de um produto entre dois elementos de um conjunto parcialmente ordenado (visto como uma categoria) pode ser interpretada como segue:

a) *Dependência causal*: ambos são dependentes causais de um mesmo elemento, pois possuem um limitante inferior. Por exemplo, no conjunto parcialmente ordenado ⟨{a, b, c, p, q, x, y, z}, ≤⟩ (parcialmente) ilustrado na Figura 6.1, embora *b* seja independente de *c*, ambos dependem de *q* para ocorrerem, Adicionalmente, é o "maior" elemento dos quais *b* e *c* são dependentes causais, ou seja, é o objeto resultante do produto de *b* e *c*. Note-se que ambos também dependem de *p*, mas este não é o "maior" (limitante inferior). Adicionalmente, não existe produto entre *y* e *p*. Isso significa que não possuem qualquer dependência causal comum;

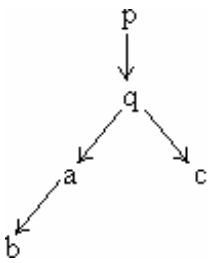


Figura 6.1 Produto em conjunto parcialmente ordenado: dependência causal/concorrência

```
p:      ...
q:      fork a, c
                exit
a:      ...
b:      ...
                exit
c:      ...
                exit
```

Figura 6.2 Trecho de programa usando as primitivas fork e exit

b) *Concorrência*: além da dependência causal, *b* e *c* são concorrentes, Ou seja, não só dependem da ocorrência de *q*, como também pertencem a partes independentes do sistema, com origem em *q*.

Tal interpretação pode ser usada para dar semântica a uma *primitiva* (operação atômica, em geral implementada no *núcleo* de um *sistema operacional*) denominada *fork*, que possui como função o "disparo" de partes concorrentes (independentes) de um sistema.

Vale recordar que as operações *fork*, *wait* e *exit* foram definidas por Conway em 1963, tendo sido implementadas em diversos sistemas operacionais e linguagens de programação. A execução da instrução *fork x* por um processo *f* faz com que um novo processo *g* seja criado e inicie sua execução pela instrução com rótulo *x*. A partir daí, os processos *f* e *g* são executados em simultâneo. Se um processo executa uma instrução *exit*, ele termina.

A Figura 6.2 ilustra um trecho de programa em uma linguagem de descrição correspondente ao subconjunto parcialmente ordenado ilustrado na Figura 6.1. Nesse caso, os rótulos das instruções

correspondem aos elementos do conjunto parcialmente ordenado. O trecho de programa ilustra o disparo e o controle de fim de processamento (usando uma outra primitiva denominada *exit*) das partes concorrentes. Existe, ainda, uma terceira primitiva, denominada *wait*, que objetiva controlar sincronizações de partes independentes.

Coproduto em um Conjunto Parcialmente Ordenado ´ Sistemas Concorrentes e Primitiva wait

Relativamente aos conjuntos parcialmente ordenados como um modelo semântico para sistemas concorrentes (e analogamente ao produto), a existência do coproduto entre dois elementos também possui uma importante interpretação, que é a de caracterizar a existência e o momento em que ocorre a sincronização de duas partes independentes (concorrentes) de um sistema. Tal sincronização pode ser usada para dar semântica a uma primitiva denominada wait. Por exemplo, no conjunto parcialmente ordenado na Figura 6.3, w é o objeto resultante do coproduto de b e u. Assim, as partes independentes do sistema onde se encontram b e u serão sincronizadas em w, o que significa que existe um wait em w, como ilustrado no trecho de programa na Figura 6.4 (o número 2 após o wait significa a sincronização de duas partes independentes). Vale lembrar que a instrução *wait t* tem a semântica: t:=t−1; while t≠0 do {}. Note-se a dificuldade de expressar um conjunto parcialmente ordenado (como um modelo para sistemas concorrentes) usando as primitivas fork, exit e wait introduzidas. *Fork*, *wait* e *exit* podem tornar os programas confusos, principalmente quando usados dentro de loops ou de outras estruturas de controle.

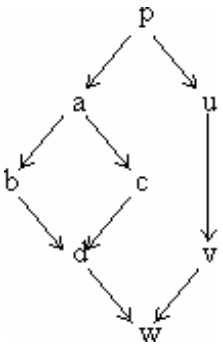


Figura 6.3 Conjunto parcialmente ordenado

```
p:    fork a, u
      exit
a:    fork b, c
      exit
b:    goto d
c:    goto d
d:    wait 2
      goto w
u:    ...
v:    ...
w:    wait 2
      exit
```

Figura 6.4 Trecho de programa usando as primitivas fork, exit e wait

PRODUTO E COPRODUTO EM GRAFOS REFLEXIVOS

A abordagem de grafos exemplifica formalismos análogos aos autômatos e sistemas de transições os quais são tipicamente seqüenciais. Destaque-se que formalismos seqüenciais podem ser usados para dar semântica a sistemas concorrentes, usando a abordagem denominada de *intercalação*. Nesse caso, resumidamente, duas ações *a* e *b* são ditas concorrentes se a ordem de ocorrência é irrelevante. Assim, em termos de grafos como um modelo seqüencial, existem dois caminhos alternativos que refletem as possíveis seqüencializações de *a* e *b*, ou seja, um caminho constituído por *a* seguido de *b* e outro, por *b* seguido de *a*, como ilustrado na Figura 6.5. O “quadrado” representado é usualmente denominado de *quadrado da independência*.

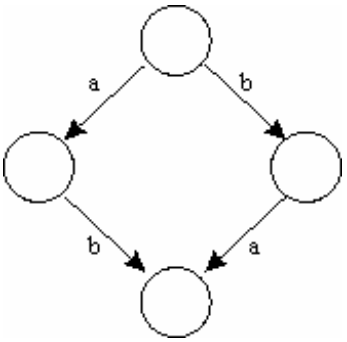


Figura 6.5 Quadrado da independência

Relativamente à categoria dos grafos reflexivos, tem-se que:

- o objeto resultante do coproduto pode ser visto como a justaposição, lado a lado, dos grafos componentes;
- no objeto resultante do produto, as transições são pares de transições dos grafos componentes. Entretanto, quando visto como o comportamento conjunto de sistemas, pode ser interpretado de forma completamente diferente, refletindo todas as combinações síncronas e assíncronas das transições dos sistemas componentes.

Exemplo 6.2 *– Produto e coproduto de grafos reflexivos* - Considere os grafos reflexivos G_1 e G_2 ilustrados na Figura 6.6. Então:

a) *Coproduto*. O objeto resultante do coproduto é um grafo reflexivo distribuído constituído pelos grafos G_1 e G_2 "justapostos";

b) *Produto*. O objeto resultante do produto é uma categoria na qual objetos e morfismos são pares dos objetos e morfismos das categorias componentes, com ilustrado na Figura 6.7.

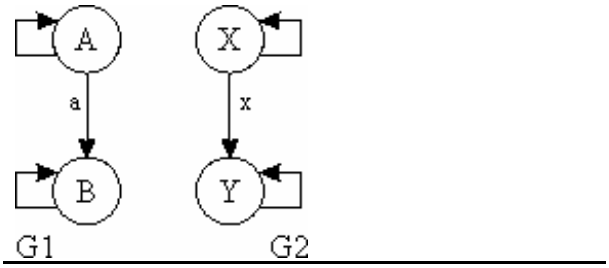


Figura 6.6 Grafos reflexivos

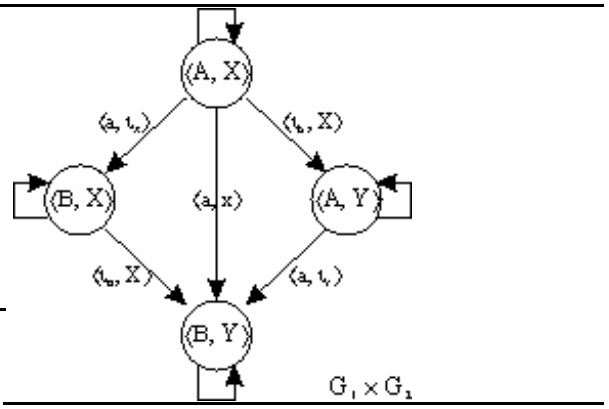


Figura 6.7 Produto de grafos reflexivos

Uma aplicação usual para o coproduto e para o produto de grafos reflexivos é a representação do comportamento conjunto de sistemas baseados em grafos reflexivos. Nesse caso, a interpretação do *coproduto* é a de um combinador *assíncrono*. Adicionalmente, no caso de autômatos ou sistemas de transições, o coproduto caracteriza um *não-determinismo* ou *escolha*, no sentido em que cada um dos grafos componentes (do objeto resultante do coproduto) representa uma alternativa.

Por outro lado, o *produto* pode ser interpretado como um combinador simultaneamente síncrono e assíncrono. Por exemplo, considere os grafos ilustrados na Figura 6.6. Uma transição identidade de um nodo é interpretada como uma transição do tipo *sem operação* (*"no operation"*), a qual não altera o estado do sistema. Então, o objeto resultante do produto ilustrado na Figura 6.7 é um sistema no qual as transições

representam todas as combinações *síncronas* e *assíncronas* possíveis entre as transições dos sistemas componentes.

Este capítulo descreveu os conceitos básicos da Teoria das Categorias, visando a uma abordagem voltada à especificação formal de sistemas concorrentes. Inicialmente, foram abordadas as construções básicas dessa teoria, em especial seus diagramas e o conceito de limite, o qual deriva diversos outros conceitos básicos, além de alguns exemplos ilustrativos. Todos esses conceitos formam o embasamento que permite estudar de maneira mais aprofundada as aplicações da Teoria das Categorias à especificação formal de sistemas concorrentes. Essa é uma área de pesquisa que se preocupa com a qualidade do *software* a ser desenvolvido, tanto em seu aspecto de correção como de documentação e desempenho. Espera-se, portanto, que este trabalho possa constituir um auxílio aos primeiros passos na busca de soluções alternativas através da Teoria das Categorias para a melhoria da qualidade das especificações de programas.

7 Conclusões

O presente trabalho consistiu em uma coleção de conceitos, exemplos, construções e resultados referentes à aplicação de lógicas categóricas, especialmente a lógica intuicionista, à especificação formal de sistemas concorrentes. O objetivo foi a exposição sintética das idéias encontradas, bem como sua análise crítica e comparativa, consoante com a bibliografia consultada, para fins didáticos.

Preocupou-nos apresentar todos os tópicos de uma forma acessível ao aluno, dentro de um esquema de ensino objetivo e prático. Por essa razão, as características deste livro são eminentemente didáticas. Foram evitadas demonstrações, sendo apresentados comentários e análises objetivas dos assuntos. O estudo é complementado por exercícios em abundância, nos quais procuramos trabalhar com as situações práticas.

Este livro introduz alguns dos principais conceitos de Teoria das Categorias, de forma didática e acessível, tanto para graduação como para pós-graduação em Ciência da Computação. Sempre que possível, os principais exemplos apresentados são baseados em casos aplicados à computação em geral.

Procurou-se destacar, ao longo do texto, importantes características da Lógica Categorial aplicada à Formalização de Sistemas Concorrentes as quais motivam o seu uso na Ciência da Computação, com destaque para:

a) Independência de Implementação.

b) Dualidade.

c) Herança de Resultados e Comparação da Expressividade de Formalismos.

d) Notação Gráfica.

e) Expressividade de suas Construções.

Talvez a principal motivação para o uso da Lógica Categorial em Ciência da Computação seja a expressividade das suas construções, pois permite formalizar idéias mais complexas de forma mais simples bem como propicia um novo ou melhor entendimento das questões relacionadas com toda a Ciência da Computação. Tal característica é especialmente relevante quando se considera a complexidade dos sistemas computacionais atuais nos quais o desenvolvimento de soluções é limitado pela capacidade do ser humano de entender e expressar os problemas e suas soluções.

Assim, o leitor que acompanhou a abordagem desenvolvida ao longo deste livro, além de ter desenvolvido a sua capacidade de raciocínio abstrato (lógico-matemático) como um todo, deve ser capaz de aplicar os conceitos básicos da Teoria das Categorias como uma ferramenta Matemática unificada para investigações em Ciência da Computação;

Ao longo de todo o texto, foram explorados alguns aspectos relativos a Modelos para Concorrência. Concorrência é um problema crítico em Ciência da Computação, incluindo o seu correto entendimento e a sua representação computacional.

Já a representação de sistemas concorrentes complexos, possivelmente compostos a partir de outros sistemas, bem como a definição da relação entre as partes componentes, pode ser explicada usando, por exemplo, operações de limites e de colimites dentro da Lógica Categorial.

Em relação à Lógica Categorial, pode-se deduzir que, em matéria de sistemas lógicos, a seguinte associação é válida para uma determinada categoria C:

- *fórmulas são objetos;*

- *deduções são morfismos;*
- *conectivos/quantificadores são funtores;*
- *regras são transformações naturais.*

Essas equações foram discutidas no decorrer do texto para a lógica intuicionista. No entanto, existe uma quantidade significativa de trabalhos, relacionando diferentes lógicas do ponto de vista da lógica categorial.

Por meio deste texto, procurou-se apresentar uma visão geral inicial sobre o estudo de lógicas categoriais e sua aplicação à formalização de sistemas concorrentes, suprindo uma lacuna existente principalmente em língua portuguesa. Uma versão anterior deste trabalho foi apresentada no XIII Salão de Iniciação Científica da UFRGS em março de 2002, obtendo o prêmio Destaque do Evento pela qualidade da pesquisa.

Entre os trabalhos futuros dos quais esta pesquisa mostrou a existência da necessidade, estão a validação da característica do texto de ser autocontido, a qual deverá feita através de um seminário interno do GMCPAD; através de uma disciplina de tópicos a ser proposta pelos professores responsáveis com aulas sobre partes do assunto lógica categorial, as quais teriam como assistência professores do instituto de informática e alunos de disciplinas que fazem uso de lógica categorial (como, por exemplo, a disciplina de Categorias Computacionais do curso de bacharelado em Ciência da Computação da UFRGS); e através de miniaulas para público diferenciado, como alunos de graduação de outros cursos além do curso de Ciência da Computação. A disponibilização do texto para a comunidade acadêmica via o site do Grupo de Matemática da Computação e Processamento de Alto Desempenho também deve ser alcançada com vistas ao mesmo objetivo.

Espera-se que esta obra consiga incentivar os pesquisadores a escreverem livros-textos sobre os seus temas de estudo para todos os níveis de escolaridade, e não apenas para os mais entendidos, proporcionando, assim, aos temas de pesquisa hoje exclusivamente universitária um maior alcance em abrangência e disseminação para o grande público.

Bibliografia

[AND91] ANDREWS, G. Concurrent Programming: Principles and Practice. Benjamin Cummings, 1991.

[BAV2001] BAVARESCO, Simone; MENEZES, Paulo Fernando Blauth. **Lógica Intuicionista**: Relato de Atividades. Arquivo gerado em PowerPoint 97 apresentado no evento Iniciação Científica em Relatos, promovido pela PROPESQ/UFRGS, 2001, Porto Alegre.

[BEE84] BEESON, M. Proving programs e programming proofs, in: Proc. 7th. Int. Cong. of Logic, Methodology and Philosophy of Science. Salzburg. Forthcoming in 'Logic, methodology and the philosophy of science', North Holland Studies in Logic, 1984.

[BIS71] BISHOP, E. 'Mathematics as a numberical language', in: Intuitionism and proof theory, ed. Myhill, Kino, & Vesley. North Holland Studies in Logic, Amsterdam, 1970. p. 53-71.

[BRU80] BRUJIN, N. G. 'A survey of the project automath', in: To H. B. Curry: Essays on combinatory logic, lambda calculus and formalism, ed. Seldin, J. P. & Hindley, J. R., Academic Press, London, 1980.

[CAM2001] CAMPANI, Carlos Antônio Pereira. **Lógica Categórica Aplicada à Semântica de Programas**. Trabalho para a disciplina Categorias Computacionais, 2001. Porto Alegre.

[CAM2001a] CAMPANI, Carlos Antônio Pereira. **Lógica intuicionista**: a teoria de tipos de Martin-Löf. Apostila de palestra com duração de 2 horas proferida no Instituto de Informática da UFRGS em 26 mar.2001.

[COF71] COFFMAN, E. G.; ELPHICK, M. J.; SHOSHANI, A. System Deadlocks. ACM Computing Surveys, p. 67-68, june 1971.

[CON78] CONSTABLE, R. L. & O'Donnell, M. J. (1978) A programming logic, Winthrop, Cambridge, Mass.

[CON71] CONSTABLE, R. L. (1971) 'Constructive mathematics and automatic program writers', in: Proc. IFIP Congress 1971, North Holland.

[CON83] CONSTABLE, R. L. (1983) 'Mathematics as programming': in: logics of programs, Springer Lecture Notes in Computer Science. vol 164. ed. E.Clarke & d. Kozen, 116-129.

[COR90] CORRADINI, An Algebraic semantics for transition systems and logic programming, PhD Thesis, technical report TD – 8/90, Università di Pisa, 1990.

[COS92] COSTA, Marcos Mota do Carmo. **Introdução à lógica modal aplicada à computação**. Porto Alegre: Instituto de Informática da UFRGS, 1992. 210 p. p.31-85. Trabalho apresentado na Escola de Computação, 8., 1992, Gramado, RS.

[DAL94] DALEN, D. Logic and structure. 3.ed., Springer Verlag, 1994.

[DEA78] DEAÑO, Alfredo. **Introducción a la lógica formal**. Madri: Alianza Universidad Textos, 1978. 424p.

[DEI84] DEITEL, H. M. Operanting Systems. Addison-Wesley, 1984.

[DIM88] DIMURO, Graçaliz Pereira. **Um estudo de extensões à programação em lógica baseadas na lógica modal**. Porto Alegre: CPGCC da UFRGS, 1988. 48 p.

[DIV2000] DIVERIO, Tiarajú Asmuz.; MENEZES, Paulo Fernando Blauth. **Teoria da computação**: máquinas universais e computabilidade. 2.ed. Porto Alegre: Instituto de Informática da UFRGS: Sagra Luzzatto, 2000.

[DON72] DONOVAN, J. Systems Programming. McGraw-Hill International Book Company, 1972.

[DOT96] D'OTTAVIANO, Itala Maria Loffredo, SILVA, J. J., SETTE, Antonio Mario A, CARNIELLI, Walter Alexandre. Traduções entre Sistemas Dedutivos: Um Tratamento Categorial. In: XI ENCONTRO BRASILEIRO DE LOGICA, 1996, Salvador, BA. Anais do XI EBL. SALVADOR, BA: 1996. p.26-26.

[DOT97] D'OTTAVIANO, Itala Maria Loffredo, SILVA, J. J., SETTE, Antonio Mario A, CARNIELLI, Walter Alexandre. Tranlations between Deductive Systems: a Categorial Approach. Journal of the Interest Group in Pure and Applied Logics (IGPL), 1997.

[DUM77] DUMMETT, M. (1977) Elements of intuitionism, Oxford U. P.

[GOL87] GOLDBLATT, Robert. **Logics of time and computation**. Standford: CSLI, 1987. n.7. 131p. p. 1-47, 70-85.

[HEY56] HEYTING, A. Intuitionism: an introduction. Amsterdam: North-Holland, 1956.

[HOA84] HOARE, C. A. R. Communicating sequential process. Prentice-Hall, 1984.

[HOW80] HOWARD, W. A. (1980) 'The formulae-as-type notion of construction', in: To H. B. Curry: essays on combinatory logic, lambda calculus and formalism, ed. Seldin, J. P. & Hindley, J. R., Academic Press, London, 429-441.

[HUG96] HUGHES, G.; CRESSWELL, M. **A new introduction to modal logic**. [S.l.: s.n.], 1996.

[KLE68] KLEENE, S. C. Mathematical logic. Nova Iorque: John Wiley & Sons, 1968. pp.191-198.

[KLE96] KLEIMAN, S.; SHAH, D.; SMAALDERS, B. Programming with Threads. Sunsoft Press, 1996.

[KRI71] KRIVINE, J. L. (1971) 'Introduction to axiomatic set theory', D. Reidel Pub. Company.

[LAM94] LAMB, Luís da Cunha. **Introdução à teoria das categorias**. 1994. 45p. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[LAM88] LAMBEK, J. & SCOTT, P. J. Introduction to higher order categorial logic. Cambridge: Cambridge University Press, 1988. 293p.

[LEA97] LEA, D. Concurrent Programming in Java: Design Principles and Patterns. Addison-Wesley, 1997.

[MAL2001] MALANOVICZ, Aline *et alii*. *Aplicação da Teoria das Categorias à especificação de sistemas concorrentes*. In: VII Congresso Argentino de Ciência da Computação. Argentina: CACIC, 2001.

[MAL2002] MALANOVICZ, Aline Vieira. **Lógicas modais**: fundamentos e aplicações. 2002. 55p. Projeto de diplomação (Curso de Bacharelado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul,

Porto Alegre.

[MAL2001a] MALANOVICZ, Aline Vieira.; DIVERIO, Tiarajú Asmuz.; MENEZES, Paulo Fernando Blauth. Teoria da Computação Concorrente formalizada por Lógica Categorial: material didático. In: SALÃO DE INICIAÇÃO CIENTÍFICA, 13., 2002, Porto Alegre. **Livro de Resumos**. Porto Alegre: UFRGS, 2001. 673p. p.278-79.

[MAN92] MANNA Z.; PNUELI, A. The temporal logic of reactive and concurrent systems. Vol. 1 Specification. Springer Verlag, 1992.

[MAR2001] MARTINI, Alfio Ricardo. **An introduction to categorical logic**: via algebraic theories. Apostila sobre minicurso ministrado no Instituto de Informática da UFRGS em 06 e 27 abr.2001.

[MAR2001a] MARTINI, Alfio Ricardo. Comunicação por e-mail. abril. 2001.

[MAR75] MARTIN-LÖF, P. (1975) 'An intuitionistic theory of types: predicative part', in: Logic colloquium 1973, ed. Rose, H. E. & Shephedson, J. C., North Holland, Amsterdam, 73-118.

[MAR82] MARTIN-LÖF, P. (1982) 'Constructive mathematics and computer programming', in Logic, methodology and philosophy of science, VI (Proc. of the 6th Int. Cong., Hannover, 1979). North Holland Publishing Company, Amsterdam, 153-175.

[MEN64] MENDELSON, E. *Introduction to Mathematical Logic.* New Jersey: Van Nostrand Company, 1964. 310p.

[MEN2001] MENEZES, Paulo Fernando Blauth; HAEUSLER, Edward Hermann. **Teoria das Categorias para Ciência da Computação**. Porto Alegre: Instituto de Informática da UFRGS : Sagra Luzzatto, 2001. 324p. (Série Livros Didáticos, n. 12).

[MIL89] MILNER, R. Communication and concurrency. Prentice-Hall, 1989.

[MOO93] Moortgat, Michael. Categorial grammar : logical parameters and linguistic variation. Lisboa: Universidade de Lisboa, 1993. 1 v. (várias paginações) - INF FL 2823

[OLI2000] OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistemas Operacionais**. Porto Alegre: Instituto de Informática da UFRGS : Sagra Luzzatto, 2000. 233p. p. 29-58. (Série Livros Didáticos, n. 11).

[PRE73] PREPARATA, Franco P.; YEH, Raymond T. **Introduction to discrete structures for computer science and engineering**. [S.l.]: Addison-Wesley, 1973.

[PRI2000] PRICE, A.; TOSCANI, S. S. Implementação de Linguagens de Programação: Compiladores. Sagra-Luzzatto, 2000.

[RAM??] RAMOS, Henrique. Palavras cruzadas Coquetel (problemas de lógica e raciocínio) e Edição especial: jogos e desafios. nº 1 e nº 9. ISSN 1517-2287. Ediouro.

[SER98] SÉRATES, Jonofon. **Raciocínio lógico**: lógico matemático, lógico quantitativo, lógico numérico, lógico analítico, lógico crítico. 7. ed. Brasilia: Jonofon, 1998. 2v.

[SIL2000] SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G. Applied Operating System Concepts. John-Wiley & Sons, 2000.

[SIL85] SILBERSCHATZ, A.; PETERSON, J. Operating Systems Concepts. 2.ed. Addison-Wesley, 1985.

[SMI82] SMITH, J. (1982) 'The identification of propositions and types in Martin-Löf's type theory: a programming example', in Foundations of computation theory, Springer Lecture Notes in Computer Science. No 158. ed. M. Karpinski, 445-454.

[SOU99] SOUZA, Sandro. **Um estudo introdutório de lógicas modais e temporais na formalização de sistemas concorrentes**. 1999. 70 p. Projeto de Diplomação (Curso de Bacharelado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[STA97] STALLINGS, W. Operating Systems: Internals and Design Principles. 3.ed. Prentice-Hall, 1997.

[TAN97] TANENBAUM, A. S.; WOODHULL, A. S. Operating Systems Design and Implementation. 2.ed. Prentice-Hall, 1997.

[TAN92] TANENBAUM, A.S. Sistemas operacionais modernos. Rio de Janeiro: Prentice-Hall do Brasil, 1992. p. 264.

[TUR84] TURNER, R. Logics for artificial intelligence. Chichester: Ellis Horwood, 1984. Cap. 4, p. 43-58: Intuitionistic logic: Martin-Löfs theory of types.

Endereços *web*

http://bocc.ubi.pt/pag/amaral-campelo-otto-apel-1.html

http://www.cs.math.ist.utl.pt/cs/courses/eap1.html

http://www.cs.math.ist.utl.pt/cs/courses/eap2.html

http://www.cs.math.ist.utl.pt/s84.www/cs/courses/fa.html

http://www.cs.math.ist.utl.pt/s84.www/cs/courses/faep.html

http://www.cs.math.ist.utl.pt/s84.www/cs/courses/tcl.html

http://www.cs.math.ist.utl.pt/s84.www/cs/courses/tclc.html

http://www.cs.math.ist.utl.pt/s84.www/cs/lcg/seminar9899.html

http://www.di.fc.ul.pt/~llf/fmc.html

http://www.inf.ufrgs.br/~wmf98

http://www.inf.ufrgs.br/pos/html/pep_96.htm

http://www.inf.ufsc.br/sbes99/anais/SBES-Completo/20.pdf

http://www.mat.uc.pt/~picado/publicat/Pref.html

http://www.math.ist.utl.pt/mma/tcl.html

http://www-nt.inf.puc-rio.br/cgilua/cgilua.exe/html/pessoa.htm?cx%5Fid=prof%5Fpubaa&cx%5Forig=827&id=HAEUSLER%2C+E%2EH%2E

ANEXO 1 – Esquema de Estudo para Lógica Intuicionista

Lógica	Clássica (Hilbert)	Intuicionista (Brouwer)
Pensamento	Axiomático	Intuitivo
Abordagem	Existencial	Genética, Construtiva
Matemática	Abstrata	Concreta

FINITO *versus* INFINITO

- Brouwer não acreditava na existência da totalidade dos números naturais

  → números que crescem ao infinito pertencem ao estado da criação, não são coisas existentes no mundo real.

- Números reais (abordagem clássica):

  Sejam X e Y reais.

  X = Y se as representações decimais de seus dígitos são iguais.

  Problema: 2,4 = 2,3999999999999999999...

  Solução: escolher uma das representações!

  Brouwer não aceitava essa idéia!

Paradigmas da Lógica Intuicionista

- O desenvolvimento da **Matemática** deve ir até aonde a **intuição** chegar.
- Os objetivos da **Matemática** devem ser gerados por alguns princípios de **construção**, **não** tornados **válidos** de uma só vez por satisfazer um **conjunto de axiomas**.
- Temos que **negar** princípios cuja justificativa depende da **concepção de conjuntos infinitos** como completos.
- Vista pelos matemáticos intuicionistas apenas como uma **ferramenta** e não como uma base.

Princípio do Terceiro Excluído (PÚ ØP)

- Considere um predicado P(x) definido sobre algum conjunto D. Então vale
  - $\exists x\, P(x) \vee \neg\exists x\, P(x)$
  - 1º) Se D é um conjunto finito, então existe um procedimento finito (construtivo) que prova $\exists x\, P(x)$ ou $\neg\exists x\, P(x)$, e, portanto, $\exists x\, P(x) \vee \neg\exists x\, P(x)$ vale.
  - Exemplo: os intuicionistas não teriam dificuldade em aceitar a seguinte conclusão.

Imaginemos um conjunto de 10 pessoas em que uma é um assassino. Dividimos esse conjunto em dois conjuntos de cinco pessoas e escolhemos ao acaso um deles. Logo, o assassino está nesse conjunto ou o assassino não está nesse conjunto. (O conjunto é finito - ou pode ser testado finitamente.)

Lógica	Clássica	Intuicionista
Finito	Aceita	Aceita
Infinito	Aceita	Não aceita, pois não há procedimento finito para testar.

  - 2º) Segundo Brouwer, não existe razão para se aceitar $\exists x\, P(x) \vee \neg\exists x\, P(x)$ se D é infinito (pois não existe procedimento finito para testar).
    - $\exists x\, P(x)$ significa, para o intuicionista, que alguém é capaz de encontrar um y tal que P(y) vale.
    - $\neg\exists x\, P(x)$ significa, para o intuicionista, que ninguém é capaz de encontrar um y tal que P(y).
  - Ou seja, a matemática clássica e a matemática intuicionista coincidem quando os conjuntos envolvidos (domínios) são finitos.

Absurdo Clássico

$$\frac{\begin{array}{c}[\text{Ø } A]\\ \vdots\\ \wedge\end{array}}{A} \qquad \text{Þ} \qquad \frac{\begin{array}{c}[\text{Ø } A]\\ \vdots\\ \wedge\end{array}}{\text{ØØ } A}$$

A prova do absurdo clássico, para os intuicionistas, é uma prova de ¬¬ A.

ØØ *A* ® *A* NÃO VALE

Não se pode provar A *pela negação de* ¬ A.

A *deve ser provado diretamente (construtivamente).*

Conseqüências Diretas

- A lógica intuicionista é mais empobrecida que a lógica clássica.

  Intuicionista ⊆ Clássica E

  Clássica ⊄ Intuicionista

- É necessário usar todas as estruturas (conetivos) na lógica intuicionista, pois algumas "abreviaturas" não valem mais.
- As provas, em lógica intuicionista, são mais difíceis de serem obtidas (têm menos recursos).
- Gödel (incompletude dos sistemas formais): os intuicionistas não aceitam os sistemas formais e a idéia de completude associada.

A Lógica Intuicionista na Ciência da Computação

- Provas construtivas pressupõem **funções totais**, o que **evitaria** o problema de **looping**.
- Muitos resultados importantes para Ciência da Computação não fazem sentido na matemática intuicionista.
- Turing (problema da parada): Essa prova é feita por redução ao absurdo clássica, o que não é aceito pelos intuicionistas. Para um intuicionista, a questão estaria "em aberto" até que alguém apontasse um algoritmo capaz de testar a parada de qualquer programa.
- Construção de programas: **semântica** operacional na linguagem de **programação funcional** ® Teoria de Tipos de Per Martin-Löff

ANEXO 2 – Respostas aos Exercícios Propostos

CALOUROS E VETERANOS

O primeiro aluno tinha de responder “não” à pergunta do desconhecido, fosse ou não de fato um calouro. Portanto, a resposta do segundo aluno foi verdadeira. Isso significa que o segundo aluno é veterano. Se o terceiro aluno estiver dizendo a verdade, então ele não é calouro. Por outro lado, se o terceiro aluno estiver mentindo, ele é calouro e o primeiro aluno é veterano. Portanto, o primeiro ou o terceiro aluno podiam ser calouros, mas não ambos. Assim, o grupo de três alunos terá de consistir de um calouro e dois veteranos.

OS TRÊS ERROS

1º erro: “contém”, e não “contén”
2º erro: “três”, e não “tres”
3º erro: Esta frase tem só dois erros.

TAMPAS TROCADAS

Da caixa com a inscrição PB. Nesta caixa só podem estar duas camisas da mesma cor, ou seja, PP ou BB. Supondo-se que a camisa retirada seja B, então estão aqui as BB. Na caixa com a tampa PP não podem estar duas camisas brancas, como já sabemos. A única possibilidade que resta para esta caixa é PB, e a caixa com a tampa BB deve conter as duas camisas pretas (PP).

3.4.1

a) Não é proposição, pois não é uma sentença declarativa.
b) É uma proposição. V
e) E uma proposição. V
d) Não é proposição, pois não é sentença declarativa e sim interrogativa.
e) É uma proposição. V
f) É unia proposição. F

3.4.2

São sentenças abertas: a e e.
São proposições declarativas: b e d.

3.4.3

a) O empregado não foi demitido.
b) O patrão não indenizou o empregado.
c) O empregado foi demitido e o patrão indenizou o empregado.
d) O empregado foi demitido ou o patrão indenizou o empregado.
e) O empregado não foi demitido e o patrão indenizou o empregado.
f) O empregado foi demitido ou o patrão não indenizou o empregado.

3.4.4

a) $\neg p$ b) $\neg q$ c) $\neg(\neg p)$ d) $p \vee q$ e) $\neg p \wedge q$

3.4.5

a) p: l998 é um número ímpar. v(p) = F
¬p: Não é verdade que 1998 é um número par. V(¬p) = V
b) v(3 + 4 = 7) = V
v(2 + 2 = 5) = F
v(3 + 4 = 7 e 2 + 2 = 5) = F

v[¬(3 + 4 = 7 e 2 + 2 = 5)] = V

c) v(3 + 3 = 7) = F
v(4 + 4 = 9) = F
v(3 +.3 = 7 ou 4 + 4 = 9)
v[¬(3 + 3 = 7 e 4 + 4 = 9)] = V

3.4.6

Seja p: x = 2 e q: y = 6.
Afirmar "se p, então q" é o mesmo que afirmar "se ¬q, então ¬p".
Temos: ¬p: x ≠ 2 e ¬q: y ≠ 6
Portanto, "se x = 2, então y = 6" é o mesmo que "se y ≠ 6, então, x ≠ 2".
Logo, a alternativa correta é c.

3.4.7

Seja p: 9 + 7 = 17 e q: eu sou a rainha da Inglaterra.
v(p) = F. v(q) = F
Como não ocorreu VF, a proposição é verdadeira.

3.4.8

De acordo com as relações entre implicações, temos:
a) Se ela é pobre, então, ela é uma boa cozinheira; (p→q e q→p)
b) Se ela não é uma boa cozinheira, então ela não é pobre. (p→q e ¬p→¬q)
c) Se ela não é pobre, então, ela não é uma boa cozinheira. (p→q e ¬q→¬p)

ANEXO 3 – ESQUEMA DE ESTUDO PARA LÓGICA CATEGORIAL