Appendix C:
The Case Library

This appendix lists all the cases currently implemented for the constraint-understanding system, written in FIL (see Section 5.3). All the cases here are discussed in Appendix B, and this appendix supplements that with complete listings for reference. This appendix is organised as follows, divided at the top level by the kinds of higher-level constraints that the cases extract:

- Section C.1 contains the case for extracting `conjunction` (see Sections 6.2 and B.5);
- Section C.2 contains the cases for extracting `bound` (see Sections 6.2 and B.2);
- Section C.3 contains the case for extracting `range` (see Sections 6.2 and B.2);
- Section C.4 contains the cases for extracting `exactly_one` (see Section B.1);
- Section C.5 contains the cases for extracting `at_most_one` (see Section B.1);
- Section C.6 contains the cases for extracting `non_null` (see Section B.4);
- Section C.7 contains the cases for extracting `zero_or_more` (see Section B.4);
- Section C.8 contains the cases for extracting `equation` (see Section B.3);
- Section C.9 contains the cases for extracting `adder` (see Section B.3);
- Section C.10 contains the cases for extracting `multiplier` (see Section B.3);

C.1 Case for Extracting the `conjunction` HLC

```
(************************************)
(* conjunction.fil  1998:11:23 16:23 *)
(************************************)
(*      (?x AND ?y)                 *)
(************************************)
(* CONJUNCTION                      *)
(*      extracted_from = (?x AND ?y) *)
(************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = conjunction#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
```

```
     operator = and_operator;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE conjunction#1;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2 Cases for Extracting the `bound` HLC

C.2.1 Lower Inclusive Discrete Bound from `?variable >= ?limit`

```
(****************************************)
(* bound-lid1.fil       1998:11:23 17:07 *)
(****************************************)
(*   age : INTEGER;                     *)
(*   (age >= ?limit)                    *)
(****************************************)
(* BOUND                                *)
(*      kind = lower                    *)
(*      variable = age                  *)
(*      limit = ?limit                  *)
(*      inclusive = true                *)
(*      discrete = true;                *)
(*      extracted_from = (age >= ?limit) *)
(****************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = greater_than_or_equals;
     first_expression = explicit_attribute#1;
     second_expression = literal_expression#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(integer_type);
END_INSTANCE;
```

```
INSTANCE literal_expression#1;
     literal_value = integer_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = lower;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = true;
     discrete = true;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.2 Lower Inclusive Discrete Bound from `?limit <= ?variable`

```
(****************************************)
(* bound-lid2.fil       1998:11:23 17:10 *)
(****************************************)
(*   age : INTEGER;                      *)
(*   (?limit <= age)                     *)
(****************************************)
(* BOUND                                 *)
(*      kind = lower                     *)
(*      variable = age                   *)
(*      limit = ?limit                   *)
(*      inclusive = true                 *)
(*      discrete = true                  *)
(*      extracted_from = (?limit <= age) *)
(****************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = less_than_or_equals;
     first_expression = literal_expression#1;
     second_expression = explicit_attribute#1;
END_INSTANCE;
```

```
INSTANCE explicit_attribute#1;
     tipe = simple_type(integer_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value = integer_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = lower;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = true;
     discrete = true;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.3 Lower Exclusive Discrete Bound from `?variable > ?limit`

```
(**************************************)
(* bound-led1.fil      1998:11:23 17:12 *)
(**************************************)
(*   age : INTEGER;                    *)
(*   (age > ?limit)                    *)
(**************************************)
(* BOUND                               *)
(*      kind = lower                   *)
(*      variable = age                 *)
(*      limit = ?limit                 *)
(*      inclusive = false              *)
(*      discrete = true;               *)
(*      extracted_from = (age > ?limit) *)
(**************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = greater_than;
```

```
     first_expression = explicit_attribute#1;
     second_expression = literal_expression#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(integer_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value = integer_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = lower;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = false;
     discrete = true;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.4 Lower Exclusive Discrete Bound from `?limit < ?variable`

```
(***************************************)
(* bound-led2.fil      1998:11:23 17:15 *)
(***************************************)
(*   age : INTEGER;                    *)
(*   (?limit < age)                    *)
(***************************************)
(* BOUND                               *)
(*      kind = lower                   *)
(*      variable = age                 *)
(*      limit = ?limit                 *)
(*      inclusive = false              *)
(*      discrete = true;               *)
(*      extracted_from = (?limit < age) *)
(***************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
```

```
(***********)

INSTANCE binary_operation#1;
     operator = less_than;
     first_expression = literal_expression#1;
     second_expression = explicit_attribute#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(integer_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value = integer_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = lower;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = false;
     discrete = true;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.5 Upper Inclusive Discrete Bound from `?variable <= ?limit`

```
(****************************************)
(* bound-uid1.fil       1998:11:23 17:22 *)
(****************************************)
(*   age : INTEGER;                      *)
(*   (age <= ?limit)                     *)
(****************************************)
(* BOUND                                 *)
(*      kind = upper                     *)
(*      variable = age                   *)
(*      limit = ?limit                   *)
(*      inclusive = true                 *)
(*      discrete = true                  *)
(*      extracted_from = (age <= ?limit) *)
(****************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
```

```
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = less_than_or_equals;
     first_expression = explicit_attribute#1;
     second_expression = literal_expression#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(integer_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value= integer_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = upper;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = true;
     discrete = true;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.6 Upper Inclusive Discrete Bound from `?limit >= ?variable`

```
(*****************************************)
(* bound-uid2.fil       1998:11:23 17:25 *)
(*****************************************)
(*   age : INTEGER;                      *)
(*   (?limit >= age)                     *)
(*****************************************)
(* BOUND                                 *)
(*      kind = upper                     *)
(*      variable = age                   *)
(*      limit = ?limit                   *)
(*      inclusive = true                 *)
(*      discrete = true                  *)
(*      extracted_from = (?limit >= age) *)
(*****************************************)
```

```
INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = greater_than_or_equals;
     first_expression = literal_expression#1;
     second_expression = explicit_attribute#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(integer_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value= integer_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = upper;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = true;
     discrete = true;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.7 Upper Exclusive Discrete Bound from `?variable < ?limit`

```
(***************************************)
(* bound-ued1.fil      1998:11:23 17:28 *)
(***************************************)
(*   age : INTEGER;                     *)
(*   (age < ?limit)                     *)
(***************************************)
(* BOUND                                *)
(*      kind = upper                    *)
(*      variable = age                  *)
(*      limit = ?limit                  *)
(*      inclusive = false               *)
```

```
(*      discrete = true;                *)
(*      extracted_from = (age < ?limit) *)
(***************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = less_than;
     first_expression = explicit_attribute#1;
     second_expression = literal_expression#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(integer_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value= integer_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = upper;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = false;
     discrete = true;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.8 Upper Exclusive Discrete Bound from `?limit > ?variable`

```
(***************************************)
(* bound-ued2.fil      1998:11:23 17:30 *)
(***************************************)
(*   age : INTEGER;                    *)
(*   (?limit > age)                    *)
(***************************************)
(* BOUND                               *)
```

```
(*      kind = upper                     *)
(*      variable = age                   *)
(*      limit = ?limit                   *)
(*      inclusive = true                 *)
(*      discrete = true                  *)
(*      extracted_from = (?limit > age) *)
(*************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = greater_than;
     first_expression = literal_expression#1;
     second_expression = explicit_attribute#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(integer_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value= integer_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = upper;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = true;
     discrete = true;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.9 Lower Inclusive Continuous Bound from `?variable >= ?limit`

```
(****************************************)
(* bound-lic1.fil       1998:11:24 10:26 *)
(****************************************)
```

```
(*   age : REAL;                          *)
(*   (age >= ?limit)                      *)
(******************************************)
(* BOUND                                  *)
(*      kind = lower                      *)
(*      variable = age                    *)
(*      limit = ?limit                    *)
(*      inclusive = true                  *)
(*      discrete = false;                 *)
(*      extracted_from = (age >= ?limit) *)
(******************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = greater_than_or_equals;
     first_expression = explicit_attribute#1;
     second_expression = literal_expression#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(real_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value = real_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = lower;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = true;
     discrete = false;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.10 Lower Inclusive Continuous Bound from `?limit <= ?variable`

```
(**************************************)
(* bound-lic2.fil       1998:11:24 10:29 *)
(**************************************)
(*    age : REAL;                       *)
(*    (?limit <= age)                   *)
(**************************************)
(* BOUND                                *)
(*      kind = lower                    *)
(*      variable = age                  *)
(*      limit = ?limit                  *)
(*      inclusive = true                *)
(*      discrete = false                *)
(*      extracted_from = (?limit <= age) *)
(**************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = less_than_or_equals;
     first_expression = literal_expression#1;
     second_expression = explicit_attribute#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(real_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value = real_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = lower;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = true;
     discrete = false;
```

```
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.11 Lower Exclusive Continuous Bound from `?variable > ?limit`

```
(****************************************)
(* bound-lec1.fil      1998:11:24 10:30 *)
(****************************************)
(*   age : REAL;                        *)
(*   (age > ?limit)                     *)
(****************************************)
(* BOUND                                *)
(*      kind = lower                    *)
(*      variable = age                  *)
(*      limit = ?limit                  *)
(*      inclusive = false               *)
(*      discrete = false;               *)
(*      extracted_from = (age > ?limit) *)
(****************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = greater_than;
     first_expression = explicit_attribute#1;
     second_expression = literal_expression#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(real_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value = real_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = lower;
```

```
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = false;
     discrete = false;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.12 Lower Exclusive Continuous Bound from `?limit < ?variable`

```
(**************************************)
(* bound-lec2.fil      1998:11:24 10:32 *)
(**************************************)
(*   age : REAL;                       *)
(*   (?limit < age)                    *)
(**************************************)
(* BOUND                               *)
(*      kind = lower                   *)
(*      variable = age                 *)
(*      limit = ?limit                 *)
(*      inclusive = false              *)
(*      discrete = false;              *)
(*      extracted_from = (?limit < age) *)
(**************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = less_than;
     first_expression = literal_expression#1;
     second_expression = explicit_attribute#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(real_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value = real_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
```

```
(************)

INSTANCE bound#1;
     kind = lower;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = false;
     discrete = false;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.13 Upper Inclusive Continuous Bound from `?variable <= ?limit`

```
(****************************************)
(* bound-uic1.fil       1998:11:24 10:33 *)
(****************************************)
(*   age : REAL;                         *)
(*   (age <= ?limit)                     *)
(****************************************)
(* BOUND                                 *)
(*      kind = upper                     *)
(*      variable = age                   *)
(*      limit = ?limit                   *)
(*      inclusive = true                 *)
(*      discrete = false                 *)
(*      extracted_from = (age <= ?limit) *)
(****************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = less_than_or_equals;
     first_expression = explicit_attribute#1;
     second_expression = literal_expression#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(real_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value= real_literal(?limit);
```

```
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = upper;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = true;
     discrete = false;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.14 Upper Inclusive Continuous Bound from `?limit >= ?variable`

```
(*****************************************)
(* bound-uic2.fil       1998:11:24 10:35 *)
(*****************************************)
(*   age : REAL;                         *)
(*   (?limit >= age)                     *)
(*****************************************)
(* BOUND                                 *)
(*      kind = upper                     *)
(*      variable = age                   *)
(*      limit = ?limit                   *)
(*      inclusive = true                 *)
(*      discrete = false                 *)
(*      extracted_from = (?limit >= age) *)
(*****************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = greater_than_or_equals;
     first_expression = literal_expression#1;
     second_expression = explicit_attribute#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(real_type);
```

```
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value= real_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = upper;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = true;
     discrete = false;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.15 Upper Exclusive Continuous Bound from `?variable < ?limit`

```
(**************************************)
(* bound-uec1.fil      1998:11:24 10:36 *)
(**************************************)
(*   age : REAL;                      *)
(*   (age < ?limit)                   *)
(**************************************)
(* BOUND                              *)
(*      kind = upper                  *)
(*      variable = age                *)
(*      limit = ?limit                *)
(*      inclusive = false             *)
(*      discrete = false;             *)
(*      extracted_from = (age < ?limit) *)
(**************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = less_than;
     first_expression = explicit_attribute#1;
     second_expression = literal_expression#1;
```

```
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(real_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value= real_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = upper;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = false;
     discrete = false;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.2.16 Upper Exclusive Continuous Bound from `?limit > ?variable`

```
(***************************************)
(* bound-uec2.fil      1998:11:24 10:38 *)
(***************************************)
(*   age : REAL;                        *)
(*   (?limit > age)                     *)
(***************************************)
(* BOUND                                *)
(*      kind = upper                    *)
(*      variable = age                  *)
(*      limit = ?limit                  *)
(*      inclusive = false               *)
(*      discrete = false                *)
(*      extracted_from = (?limit > age) *)
(***************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = bound#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)
```

```
INSTANCE binary_operation#1;
     operator = greater_than;
     first_expression = literal_expression#1;
     second_expression = explicit_attribute#1;
END_INSTANCE;

INSTANCE explicit_attribute#1;
     tipe = simple_type(real_type);
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value= real_literal(?limit);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE bound#1;
     kind = upper;
     variable = explicit_attribute#1;
     limit = ?limit;
     inclusive = false;
     discrete = false;
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.3 Case for Extracting the `range` HLC

```
(*****************************************)
(* range.fil              1998:11:24 10:43 *)
(*****************************************)
(* CONJUNCTION                           *)
(*      conjuncts = {bound#1,bound#2}      *)
(* BOUND#1                               *)
(*      kind = lower                     *)
(*      variable = age                   *)
(* BOUND#2                               *)
(*      kind = upper                     *)
(*      variable = age                   *)
(*****************************************)
(* RANGE                                 *)
(*      lower_bound = bound#1;           *)
(*      upper_bound = bound#2;           *)
(*      extracted_from = {bound#1,bound#2} *)
(*      variable = age                   *)
(*****************************************)

INSTANCE case_frame#1;
```

```
     problem = conjunction#1;
     solution = range#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE conjunction#1;
     conjuncts = {bound#1,bound#2};
END_INSTANCE;

INSTANCE bound#1;
     kind = lower;
     variable = ?variable;
END_INSTANCE;

INSTANCE bound#2;
     kind = upper;
     variable = ?variable;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE range#1;
     lower_bound = bound#1;
     upper_bound = bound#2;
     variable = ?variable;
     extracted_from = {bound#1,bound#2};
     conjunct_of = conjunction#1;
END_INSTANCE;
```

C.4 Cases for Extracting the `exactly_one` HLC

C.4.1 `exactly_one` from an Enumeration Type Attribute

```
(************************************************)
(* exactly_one-enum.fil        1998:11:24 10:49 *)
(************************************************)
(* ENTITY person;                               *)
(*    sex : gender;                             *)
(* END_ENTITY;                                  *)
(*                                              *)
(* TYPE gender = ENUMERATION OF (male,female); *)
(* END_TYPE;                                    *)
(************************************************)
(* EXACTLY_ONE                                  *)
```

```
(*      choice_set = {male,female}              *)
(*      extracted_from = gender                 *)
(*      constrainee = sex                       *)
(************************************************)

INSTANCE case_frame#1;
     problem = explicit_attribute#1;
     solution = exactly_one#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE explicit_attribute#1;
     tipe = defined_type#1;
     is_optional = false;
END_INSTANCE;

INSTANCE defined_type#1;
     tipe = enumeration_type(?list);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE exactly_one#1;
     choice_set = ?list;
     extracted_from = {defined_type#1};
     constrainee = explicit_attribute#1;
END_INSTANCE;
```

C.4.2 `exactly_one` from a Select Type Attribute

```
(**********************************************)
(* exactly_one-sele.fil     1998:11:24 10:52 *)
(**********************************************)
(* ENTITY person;                             *)
(*   sex : gender;                            *)
(* END_ENTITY;                                *)
(*                                            *)
(* TYPE gender = SELECT (manhood,womanhood); *)
(* END_TYPE;                                  *)
(*                                            *)
(* ENTITY manhood;                            *)
(* END_ENTITY;                                *)
(*                                            *)
(* ENTITY womanhood;                          *)
```

```
(* END_ENTITY;                                *)
(******************************************)
(* EXACTLY_ONE                                *)
(*      choice_set = {manhood,womanhood}      *)
(*      extracted_from = gender               *)
(*      constrainee = sex                     *)
(******************************************)

INSTANCE case_frame#1;
     problem = explicit_attribute#1;
     solution = exactly_one#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE explicit_attribute#1;
     tipe = defined_type#1;
     is_optional = false;
END_INSTANCE;

INSTANCE defined_type#1;
     tipe = select_type(?set);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE exactly_one#1;
     choice_set = ?set;
     extracted_from = {defined_type#1};
     constrainee = explicit_attribute#1;
END_INSTANCE;
```

C.4.3 **`exactly_one`** from a **`ONEOF`** Supertype Expression

```
(*******************************************************)
(* exactly_one-inh.fil                 1998:11:24 10:55 *)
(*******************************************************)
(* ENTITY person ABSTRACT SUPERTYPE OF (ONEOF (man,woman)); *)
(* END_ENTITY;                                            *)
(*                                                        *)
(* ENTITY man SUBTYPE OF (person);                        *)
(* END_ENTITY;                                            *)
(*                                                        *)
(* ENTITY woman SUBTYPE OF (person);                      *)
(* END_ENTITY;                                            *)
```

```
(*********************************************************)
(* EXACTLY_ONE                                           *)
(*      choice_set = {man,woman}                         *)
(*      extracted_from = ONEOF(man,woman)                *)
(*      constrainee = person                             *)
(*********************************************************)

INSTANCE case_frame#1;
     problem = entity_type#1;
     solution = exactly_one#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE entity_type#1;
     is_abstract = true;
     supertype_of = oneof_expression#1;
END_INSTANCE;

INSTANCE oneof_expression#1;
     expressions = ?set;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE exactly_one#1;
     choice_set = ?set;
     extracted_from = {oneof_expression#1};
     constrainee = entity_type#1;
END_INSTANCE;
```

C.5 Cases for Extracting the `at_most_one` HLC

C.5.1 `at_most_one` from an Enumeration Type Attribute

```
(*********************************************)
(* at_most_one-enum.fil       1998:11:24 10:58 *)
(*********************************************)
(* ENTITY person;                            *)
(*   sex : OPTIONAL gender;                  *)
(* END_ENTITY;                               *)
(*                                           *)
(* TYPE gender = ENUMERATION OF (male,female); *)
(* END_TYPE;                                 *)
(*********************************************)
```

```
(* AT_MOST_ONE                                  *)
(*      choice_set = {male,female}              *)
(*      extracted_from = gender                 *)
(*      constrainee = sex                       *)
(***********************************************)

INSTANCE case_frame#1;
     problem = explicit_attribute#1;
     solution = at_most_one#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE explicit_attribute#1;
     tipe = defined_type#1;
     is_optional = true;
END_INSTANCE;

INSTANCE defined_type#1;
     tipe = enumeration_type(?list);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE at_most_one#1;
     choice_set = ?list;
     extracted_from = {defined_type#1};
     constrainee = explicit_attribute#1;
END_INSTANCE;
```

C.5.2 `at_most_one` from a Select Type Attribute

```
(*********************************************)
(* at_most_one-sele.fil     1998:11:24 11:00 *)
(*********************************************)
(* ENTITY person;                            *)
(*   sex : OPTIONAL gender;                  *)
(* END_ENTITY;                               *)
(*                                           *)
(* TYPE gender = SELECT (manhood,womanhood); *)
(* END_TYPE;                                 *)
(*                                           *)
(* ENTITY manhood;                           *)
(* END_ENTITY;                               *)
(*                                           *)
```

```
(* ENTITY womanhood;                          *)
(* END_ENTITY;                                *)
(*********************************************)
(* AT_MOST_ONE                                *)
(*      choice_set = {manhood,womanhood}      *)
(*      extracted_from = gender               *)
(*      constrainee = sex                     *)
(*********************************************)

INSTANCE case_frame#1;
     problem = explicit_attribute#1;
     solution = at_most_one#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE explicit_attribute#1;
     tipe = defined_type#1;
     is_optional = true;
END_INSTANCE;

INSTANCE defined_type#1;
     tipe = select_type(?set);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE at_most_one#1;
     choice_set = ?set;
     extracted_from = {defined_type#1};
     constrainee = explicit_attribute#1;
END_INSTANCE;
```

C.5.3 `at_most_one` from a `ONEOF` Supertype Expression

```
(***************************************************)
(* at_most_one-inh.fil            1998:11:24 11:02 *)
(***************************************************)
(* ENTITY person SUPERTYPE OF (ONEOF (man,woman)); *)
(* END_ENTITY;                                     *)
(*                                                 *)
(* ENTITY man SUBTYPE OF (person);                 *)
(* END_ENTITY;                                     *)
(*                                                 *)
(* ENTITY woman SUBTYPE OF (person);               *)
```

```
(* END_ENTITY;                                      *)
(***************************************************)
(* AT_MOST_ONE                                      *)
(*      choice_set = {man,woman}                    *)
(*      extracted_from = ONEOF(man,woman)           *)
(*      constrainee = person                        *)
(***************************************************)

INSTANCE case_frame#1;
     problem = entity_type#1;
     solution = at_most_one#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE entity_type#1;
     is_abstract = false;
     supertype_of = oneof_expression#1;
END_INSTANCE;

INSTANCE oneof_expression#1;
     expressions = ?set;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE at_most_one#1;
     choice_set = ?set;
     extracted_from = {oneof_expression#1};
     constrainee = entity_type#1;
END_INSTANCE;
```

C.6 Cases for Extracting the `non_null` HLC

C.6.1 `non_null` from a Mandatory Attribute

```
(************************************)
(* non_null-attr.fil 1998:11:24 11:08 *)
(************************************)
(* ENTITY e;                          *)
(*   a : t;                           *)
(* END_ENTITY;                        *)
(************************************)
(* NON_NULL                           *)
(*      extracted_from = a            *)
```

```
(*      attribute = a                 *)
(************************************)

INSTANCE case_frame#1;
     problem = explicit_attribute#1;
     solution = non_null#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE explicit_attribute#1;
     is_optional = false;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE non_null#1;
     extracted_from = {explicit_attribute#1};
     attribute = explicit_attribute#1;
END_INSTANCE;
```

C.6.2 non_null from the EXISTS Predicate

```
(**************************************)
(* non_null-exists.fil 1998:11:24 11:10 *)
(**************************************)
(* ENTITY e;                          *)
(*   a : OPTIONAL t;                  *)
(* WHERE                              *)
(*   EXISTS(a);                       *)
(* END_ENTITY;                        *)
(**************************************)
(* NON_NULL                           *)
(*      extracted_from = EXISTS(a)    *)
(*      attribute = a                 *)
(**************************************)

INSTANCE case_frame#1;
     problem = built_in_function_call#1;
     solution = non_null#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)
```

```
INSTANCE built_in_function_call#1;
     function_called = exists_function;
     actual_parameters = [explicit_attribute#1];
END_INSTANCE;

INSTANCE explicit_attribute#1;
     is_optional = true;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE non_null#1;
     extracted_from = {built_in_function_call#1};
     attribute = explicit_attribute#1;
END_INSTANCE;
```

C.7 Cases for Extracting the `zero_or_more` HLC

C.7.1 `zero_or_more` from `?aggregate[0:?]`

```
(**************************************)
(* zero_or_more-empty 1998:10:25 22:38 *)
(**************************************)
(* a : AGGREGATE [0 : ?] OF t;         *)
(**************************************)
(* ZERO_OR_MORE                        *)
(*      attribute = a                  *)
(**************************************)

INSTANCE case_frame#1;
     problem = explicit_attribute#1;
     solution = zero_or_more#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE explicit_attribute#1;
     tipe = aggregation_type#1;
     is_optional = false;
END_INSTANCE;

INSTANCE aggregation_type#1;
     lower_bound = literal_expression#1;
END_INSTANCE;
```

```
INSTANCE literal_expression#1;
     literal_value = integer_literal(0);
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE zero_or_more#1;
     attribute = explicit_attribute#1;
     extracted_from = {explicit_attribute#1};
END_INSTANCE;
```

C.7.2 `zero_or_more` from `OPTIONAL ?aggregate[1:?]`

```
(****************************************)
(* zero_or_more-optional 1998:11:24 11:16 *)
(****************************************)
(* a : OPTIONAL AGGREGATE [1 : ?] OF t;   *)
(****************************************)
(* ZERO_OR_MORE                           *)
(*      attribute = a                     *)
(****************************************)

INSTANCE case_frame#1;
     problem = explicit_attribute#1;
     solution = zero_or_more#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE explicit_attribute#1;
     tipe = aggregation_type#1;
     is_optional = true;
END_INSTANCE;

INSTANCE aggregation_type#1;
     lower_bound = literal_expression#1;
END_INSTANCE;

INSTANCE literal_expression#1;
     literal_value = integer_literal(1);
END_INSTANCE;

(************)
(* Solution *)
(************)
```

```
INSTANCE zero_or_more#1;
     attribute = explicit_attribute#1;
     extracted_from = {explicit_attribute#1};
END_INSTANCE;
```

C.8 Cases for Extracting the `equation` HLC

C.8.1 `equation` from a Domain Rule

```
(****************************************)
(* equation-equals.fil 1998:10:23 02:38 *)
(****************************************)
(* ?x = ?y                              *)
(****************************************)
(* EQUATION                             *)
(*      equals = {?x,?y}                *)
(*      extracted_from = (?x = ?y)      *)
(****************************************)

INSTANCE case_frame#1;
     problem = binary_operation#1;
     solution = equation#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE binary_operation#1;
     operator = equals;
     first_expression = ?x;
     second_expression = ?y;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE equation#1;
     equals = {?x, ?y};
     extracted_from = {binary_operation#1};
END_INSTANCE;
```

C.8.2 `equation` from a Derived Attribute

```
(****************************************)
(* equation-derive.fil  1998:10:23 13:15 *)
```

```
(****************************************)
(* DERIVE ?x := ?y                      *)
(****************************************)
(* EQUATION                             *)
(*      equals = {?x,?y}                *)
(*      extracted_from = DERIVE ?x := ?y *)
(****************************************)

INSTANCE case_frame#1;
     problem = derived_attribute#1;
     solution = equation#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE derived_attribute#1;
     formula = ?z;
END_INSTANCE;

************)
(* Solution *)
(************)

INSTANCE equation#1;
     equals = {derived_attribute#1, ?z};
     extracted_from = {derived_attribute#1};
END_INSTANCE;
```

C.9 Cases for Extracting the adder HLC

C.9.1 adder from $z = x + y$

```
(**********************************)
(* adder-plus.fil 1998:11:12 17:00 *)
(**********************************)
(* EQUATION                       *)
(*      ?x + ?y = ?z              *)
(**********************************)
(* ADDER                          *)
(*      addends = {?x,?y}         *)
(*      sum = ?z                  *)
(*      extracted_from = EQUATION  *)
(**********************************)

INSTANCE case_frame#1;
     problem = equation#1;
     solution = adder#1;
```

```
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE equation#1;
     equals = {binary_operation#1, ?z};
END_INSTANCE;

INSTANCE binary_operation#1;
     operator = add;
     first_expression = ?x;
     second_expression = ?y;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE adder#1;
     addends = {?x, ?y};
     sum = ?z;
     extracted_from = {equation#1};
END_INSTANCE;
```

C.9.2 **adder** from $x = z - y$

```
(*************************************)
(* adder-minus.fil 1998:11:12 17:00 *)
(*************************************)
(* EQUATION                         *)
(*      ?x = ?z - ?y                *)
(*************************************)
(* ADDER                            *)
(*      addends = {?x,?y}           *)
(*      sum = ?z                    *)
(*      extracted_from = EQUATION   *)
(*************************************)

INSTANCE case_frame#1;
     problem = equation#1;
     solution = adder#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)
```

```
INSTANCE equation#1;
     equals = {binary_operation#1, ?x};
END_INSTANCE;

INSTANCE binary_operation#1;
     operator = subtract;
     first_expression = ?z;
     second_expression = ?y;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE adder#1;
     addends = {?x, ?y};
     sum = ?z;
     extracted_from = {equation#1};
END_INSTANCE;
```

C.10 Cases for Extracting the `multiplier` HLC

C.10.1 `multiplier` from $z = x * y$

```
(******************************************)
(* multiplier-multiply.fil 1998:11:05 13:32 *)
(******************************************)
(* EQUATION                               *)
(*      ?x * ?y = ?z                      *)
(******************************************)
(* MULTIPLIER                             *)
(*      multiplicands = {?x,?y}           *)
(*      product = ?z                      *)
(*      extracted_from = EQUATION         *)
(******************************************)

INSTANCE case_frame#1;
     problem = equation#1;
     solution = multiplier#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE equation#1;
     equals = {binary_operation#1, ?z};
END_INSTANCE;
```

```
INSTANCE binary_operation#1;
     operator = multiply;
     first_expression = ?x;
     second_expression = ?y;
END_INSTANCE;

(************)
(* Solution *)
(************)

INSTANCE multiplier#1;
     multiplicands = {?x, ?y};
     product = ?z;
     extracted_from = {equation#1};
END_INSTANCE;
```

C.10.2 `multiplier` from $x = z / y$

```
(****************************************)
(* multiplier-divide.fil 1998:10:23 15:57 *)
(****************************************)
(* EQUATION                             *)
(*      ?x = ?z / ?y                    *)
(****************************************)
(* MULTIPLIER                           *)
(*      multiplicands = {?x,?y}         *)
(*      product = ?z                    *)
(*      extracted_from = EQUATION       *)
(****************************************)

INSTANCE case_frame#1;
     problem = equation#1;
     solution = multiplier#1;
END_INSTANCE;

(***********)
(* Problem *)
(***********)

INSTANCE equation#1;
     equals = {binary_operation#1, ?x};
END_INSTANCE;

INSTANCE binary_operation#1;
     operator = divide;
     first_expression = ?z;
     second_expression = ?y;
END_INSTANCE;
```

```
(************)
(* Solution *)
(************)

INSTANCE multiplier#1;
     multiplicands = {?x, ?y};
     product = ?z;
     extracted_from = {equation#1};
END_INSTANCE;
```