Start of Lecture 7: RSL: Types

1. An RSL Primer
1.1. Types
1.1.1. Type Expressions

- Type expressions are expressions whose values are types, that is,
- possibly infinite sets of values (of "that" type).

1.1.1.1. Atomic Types

- Atomic types have (atomic) values.
- That is, values which we consider to have no proper constituent (sub-)values,
- i.e., cannot, to us, be meaningfully "taken apart".

type
[1] **Bool**
[2] **Int**
[3] **Nat**
[4] **Real**
[5] **Char**
[6] **Text**

1. The Boolean type of truth values **false** and **true**.
2. The integer type on integers ..., –2, –1, 0, 1, 2, ...
3. The natural number type of positive integer values 0, 1, 2, ...
4. The real number type of real values, i.e., values whose numerals can be written as an integer, followed by a period ("."), followed by a natural number (the fraction).
5. The character type of character values ″a″, ″b″, ...
6. The text type of character string values ″aa″, ″aaa″, ..., ″abc″, ...

Example 1 .... Basic Net Attributes:

- For safe, uncluttered traffic, hubs and links can 'carry' a maximum of vehicles.
- Links have lengths. (We ignore hub (traversal) lengths.)
- One can calculate whether a link is a two-way link.

type
  MAX = **Nat**
  LEN = **Real**
  is_Two_Way_Link = **Bool**
value
  obs_Max: (H|L) → MAX
  obs_Len: L → LEN
  is_two_way_link: L → is_Two_Way_Link
  is_two_way_link(l) ≡ ∃ lσ:LΣ · lσ ∈ obs_HΣ(l)∧**card** lσ=2

. . . . **End of Example 1**

1.1.1.2. Composite Types

- Composite types have composite values.
- That is, values which we consider to have proper constituent (sub-)values,
- i.e., can, to us, be meaningfully "taken apart".

[7] A-**set**
[8] A-**infset**
[9] A × B × ... × C
[10] A^*
[11] A^ω
[12] A $\overrightarrow{m}$ B
[13] A → B
[14] A $\stackrel{\sim}{\rightarrow}$ B
[15] (A)
[16] A | B | ... | C
[17] mk_id(sel_a:A,...,sel_b:B)
[18] sel_a:A ... sel_b:B

Example 2 . . . . **Composite Net Type Expressions:**

- The type clauses of function signatures:

  value
    f: A → B

- often have the type expressions *A* and/or *B*
- be composite type expressions:

value
  obs_HIs: L → HI-**set**
  obs_LIs: H → LI-**set**
  obs_HΣ: H → HT-**set**
  set_HΣ: H × HΣ → H

- Right-hand sides of type definitions often have composite type expressions:

type
  N = H-**set** × L-**set**
  HT = LI × HI × LI
  LT′ = HI × LI × HI

. . . . **End of Example 2**

1.1.2. Type Definitions

1.1.2.1. Concrete Types

- Types can be concrete
- in which case the structure of the type
- is specified by type expressions:

type
  A = Type_expr
schematic examples:
  A1 = B1-**set**, A2 = B1-**infset**
  A3 = B2 × C1 × D1
  B1 = E^*, B2 = E^ω
  C1 = F $\overrightarrow{m}$ G
  D1 = H → J, D2 = H $\xrightarrow{\sim}$ J
  K = L | M

Example 3 .... Composite Net Types:

- There are many ways in which nets can be concretely modelled:
- **Sorts + Observers + Axioms:** First we show an example of type definitions without right-hand side, that is, of sort definitions.

  From a net one can observe many things.

  Of the things we focus on are the hubs and the links.

  A net contains two or more hubs and one or more links.

type
  [sorts] N_α, H, L, HI, LI
value
  obs_Hs: N_α → H-**set**
  obs_Ls: N_α → L-**set**
axiom
  ∀ n:N_α · **card** obs_Hs(n)>0 ⇒ **card** obs_Ls(n)≥1 ∧ ...

- **Cartesians + Wellformedness:** A net can be considered as a Cartesian of sets of two or more hubs and sets of one or more links.

type
  [sorts] H, L
  N_β = H-**set** × L-**set**
value
  wf_N_β: N_β → **Bool**
  wf_N_β(hs,ls) ≡ **card** hs>1 ⇒ **card** ls>0 ....
  inject_N_β: N_α $\xrightarrow{\sim}$ N_β **pre**: wf_N_β(hs,ls)
  inject_N_β(n_α) ≡ (obs_Hs(n_α),obs_Ls(n_α))

- **Cartesians + Maps + Wellformedness:** Or a net can be described

  a as a triple of b-c-d:

  b hubs (modelled as a map from hub identfiers to hubs),

  c links (modelled as a map from link identfiers to links), and

  d a graph from hub h_i identifiers h_{i_i} to maps from identfiers l_{ij_i} of hub h_i connected links l_{ij} to the identfiers h_{j_i} of link connected hubs h_j.

type
[sorts] H, HI, L, LI
[a] N_γ = HUBS × LINKS × GRAPH
[b] HUBS = HI $\overrightarrow{m}$ H
[c] LINKS = LI $\overrightarrow{m}$ L
[d] GRAPH = HI $\overrightarrow{m}$ (LI −m> HI)

– [b,c] *hs:HUBS* and *ls:LINKS* are maps from hub (link) identifiers to hubs (links) where one can still observe these identfiers from these hubs (link).

- Example 12 on page 323 defines the well-formedness predicates for the above map types.

. . . . **End of Example 3**

[1] Type_name = Type_expr /∗ without |s or subtypes ∗/
[2] Type_name = Type_expr_1 | Type_expr_2 | ... | Type_expr_n
[3] Type_name ==
mk_id_1(s_a1:Type_name_a1,...,s_ai:Type_name_ai) |
... |
mk_id_n(s_z1:Type_name_z1,...,s_zk:Type_name_zk)
[4] Type_name :: sel_a:Type_name_a ... sel_z:Type_name_z
[5] Type_name = {| v:Type_name′ • $\mathcal{P}$(v) |}

- where a form of [2–3] is provided by combining the types:

Type_name = A | B | ... | Z
A == mk_id_1(s_a1:A_1,...,s_ai:A_i)
B == mk_id_2(s_b1:B_1,...,s_bj:B_j)
...
Z == mk_id_n(s_z1:Z_1,...,s_zk:Z_k)

axiom
∀ a1:A_1, a2:A_2, ..., ai:Ai •
s_a1(mk_id_1(a1,a2,...,ai))=a1 ∧ s_a2(mk_id_1(a1,a2,...,ai))=a2 ∧
... ∧ s_ai(mk_id_1(a1,a2,...,ai))=ai ∧
∀ a:A • **let** mk_id_1(a1′,a2′,...,ai′) = a **in**
a1′ = s_a1(a) ∧ a2′ = s_a2(a) ∧ ... ∧ ai′ = s_ai(a) **end**

Example 4 . . . . Net Record Types: Insert Links:

7. To a net one can insert a new link in either of three ways:
   (a) Either the link is connected to two existing hubs — and the insert operation must therefore specify the new link and the identifiers of two existing hubs;
   (b) or the link is connected to one existing hub and to a new hub — and the insert operation must therefore specify the new link, the identifier of an existing hub, and a new hub;
   (c) or the link is connected to two new hubs — and the insert operation must therefore specify the new link and two new hubs.
   (d) From the inserted link one must be able to observe identifier of respective hubs.
8. From a net one can remove a link.[5] The removal command specifies a link identifier.

[5]— provided that what remains is still a proper net

type
7 Insert == Ins(s_ins:Ins)
7 Ins = 2xHubs | 1x1nH | 2nHs
7(a) 2xHubs == 2oldH(s_hi1:HI,s_l:L,s_hi2:HI)
7(b) 1x1nH == 1oldH1newH(s_hi:HI,s_l:L,s_h:H)
7(c) 2nHs == 2newH(s_h1:H,s_l:L,s_h2:H)
8 Remove == Rmv(s_li:LI)
axiom
7(d) ∀ 2oldH(hi′,l,hi″):Ins · hi′≠hi″ ∧ obs_LIs(l)={hi′,hi″} ∧
∀ 1old1newH(hi,l,h):Ins · obs_LIs(l)={hi,obs_HI(h)} ∧
∀ 2newH(h′,l,h″):Ins · obs_LIs(l)={obs_HI(h′),obs_HI(h″)}

Example 17 on page 356 presents the semantics functions for *int_Insert* and *int_Remove*.

. . . . **End of Example 4**

1.1.2.2. Subtypes

- In RSL, each type represents a set of values. Such a set can be delimited by means of predicates.
- The set of values b which have type B and which satisfy the predicate $\mathcal{P}$, constitute the subtype A:

type
A = {| b:B · $\mathcal{P}$(b) |}

Example 5 . . . . **Net Subtypes:**

- In Example 3 on page 273 we gave three examples.
  - For the first we gave an example, **Sorts + Observers + Axioms**, “purely” in terms of sets, see *Sorts — Abstract Types* below.
  - For the second and third we gave concrete types in terms of Cartesians and Maps.

- In the **Sorts + Observers + Axioms** part of Example 3
  - a net was defined as a sort, and so were its hubs, links, hub identifiers and link identifiers;
  - axioms – making use of appropriate observer functions - make up the wellformedness condition on such nets.

We now redefine this as follows:

type
[sorts] N′, H, L, HI, LI
N = {| n:N′ · wf_N(n) |}
value
wf_N: N′ → **Bool**
wf_N(n) ≡
∀ n:N · **card** obs_Hs(n)≥0 ∧ **card** obs_Ls(n)≥0 ∧
axioms 2.–3., 5.–6., and 8., (Page 14)

- In the **Cartesians + Wellformedness** part of Example 3
  - a net was a Cartesian of a set of hubs and a set of links
  - with the wellformedness that there were at least two hubs and at least one link
  - and that these were connected appropriately (treated as ...).

  We now redefine this as follows:

type
N′ = H-**set** × L-**set**
N = {| n:N′ · wf_N(n) |}

- In the **Cartesians + Maps + Wellformedness** part of Example 3
  - a net was a triple of hubs, links and a graph,
  - each with their wellformednes predicates.

  We now redefine this as follows:

type
[sorts] L, H, LI, HI
N′ = HUBS × LINKS × GRAPH
N = {| (hs,ls,g):N′ · wf_HUBS(hs)∧wf_LINKS(ls)∧wf_GRAPH(g)(hs,ls) |}
HUBS′ = HI $\overrightarrow{m}$ H
HUBS = {| hs:HUBS′ · wf_HUBS(hs) |}
LINKS′ = LI → L
LINKS = {| ls:LINKS′ · wf_LINKS(ls) |}
GRAPH′ = HI $\overrightarrow{m}$ (LI $\overrightarrow{m}$ HI)
GRAPH = {| g:GRAPH′ · wf_GRAPH(g) |}
value
wf_GRAPH: GRAPH′ → (HUBS × LINKS) → **Bool**
wf_GRAPH(g)(hs,ls) ≡ wf_N(hs,ls,g)

- Example 12 on page 323 presents a definition of *wf_GRAPH*.

. . . . **End of Example 5**

1.1.2.3. Sorts — Abstract Types

- Types can be (abstract) sorts
- in which case their structure is not specified:

type
 A, B, ..., C

Example 6 . . . . **Net Sorts:**

- In formula lines of Examples 3–5
- we have indicated those **type** clauses which define *sorts*,
- by bracketed [sorts] literals.

. . . . **End of Example 6**

End of Lecture 7: RSL: Types