A  An RSL Primer

This is an ultra-short introduction to the RAISE Specification Language, RSL. Examples follow and expand on the examples of earlier sections.

A.1  Types

The reader is kindly asked to study first the decomposition of this section into its sub-parts and sub-sub-parts.

A.1.1  Type Expressions

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

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”.

RSL has a number of built-in atomic types. There are the Booleans, integers, natural numbers, reals, characters, and texts.

Basic Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. The Boolean type of truth values \texttt{false} and \texttt{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 \texttt{“a”}, \texttt{“b”}, ...
6. The text type of character string values \texttt{“aa”}, \texttt{“aaa”}, ..., \texttt{“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

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".

From these one can form type expressions: finite sets, infinite sets, Cartesian products, lists, maps, etc.

Let A, B and C be any type names or type expressions, then:

**Composite Type Expressions**

- [8] A-infset
- [9] A × B × ... × C
- [10] A^*

7. The set type of finite cardinality set values.
8. The set type of infinite and finite cardinality set values.
9. The Cartesian type of Cartesian values.
10. The list type of finite length list values.
11. The list type of infinite and finite length list values.
12. The map type of finite definition set map values.
13. The function type of total function values.
14. The function type of partial function values.
15. In (A) A is constrained to be:
   - either a Cartesian B × C × ... × D, in which case it is identical to type expression kind 9,
   - or not to be the name of a built-in type (cf., 1-6) or of a type, in which case the parentheses serve as simple delimiters, e.g., (A B), or (A-set), or (A-set)list, or (A|B) (C|D|(E F)), etc.
16. The postulated disjoint union of types A, B, ..., and C.
An RSL Primer

17. The record type of \( \text{mk}_\text{id} \)-named record values
\( \text{mk}_\text{id}(av, ..., bv) \), where \( av, ..., bv \), are values of respective types. The distinct identifiers \( \text{sel}_a \), etc., designate selector functions.

18. The record type of unnamed record values
\( (av, ..., bv) \), where \( av, ..., bv \), are values of respective types. The distinct identifiers \( \text{sel}_a \), etc., designate selector functions.

Example 2

Composite Net Type Expressions:
The type clauses of function signatures:

\[
\text{value} \\
\text{f: A} \rightarrow \text{B}
\]

often have the type expressions \( A \) and/or \( B \) be composite type expressions:

\[
\text{value} \\
\text{obs}_\text{Hl}s: L \rightarrow \text{Hl}-set \\
\text{obs}_\text{Ll}s: H \rightarrow \text{Ll}-set \\
\text{obs}_\text{H}\Sigma: H \rightarrow \text{HT}-set \\
\text{set}_\text{H}\Sigma: H \times H\Sigma \rightarrow H
\]

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

\[
\text{type} \\
N = \text{H-set} \times \text{L-set} \\
\text{HT} = \text{Ll} \times \text{Hl} \times \text{Ll} \\
\text{LT}' = \text{Hl} \times \text{Ll} \times \text{Hl}
\]

End of Example 2

A.1.2 Type Definitions

Concrete Types

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

**Type Definition**

\[
\text{type} \\
A = \text{Type_expr} \\
\text{schematic examples:} \\
A1 = \text{B1-set}, A2 = \text{B1-infset} \\
A3 = \text{B2} \times \text{C1} \times \text{D1} \\
\text{B1} = \text{E}^*, \text{B2} = \text{E}\omega \\
\text{C1} = \text{F} \equiv \text{G} \\
\text{D1} = \text{H} \rightarrow \text{J}, \text{D2} = \text{H} \sim \text{J} \\
\text{K} = \text{L} \mid \text{M}
\]
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. Possibly other entities and net attributes may also be observable, but we shall not consider those here.

\[
\text{type} \quad [\text{sorts}] \quad N_{\alpha}, H, L, LI \\
\text{value} \quad \text{obs}_Hs: N_{\alpha} \rightarrow H\text{-set} \\
\text{obs}_Ls: N_{\alpha} \rightarrow L\text{-set} \\
\text{axiom} \quad \forall n: N_{\alpha} \cdot \text{card obs}_Hs(n) > 0 \Rightarrow \text{card obs}_Ls(n) \geq 1 \land ...
\]

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

\[
\text{type} \quad [\text{sorts}] \quad H, L \\
N_{\beta} = H\text{-set} \times L\text{-set} \\
\text{value} \quad \text{wf}_N_{\beta}: N_{\beta} \rightarrow \text{Bool} \\
\text{wf}_N_{\beta}(hs,ls) \equiv \text{card hs} > 1 \Rightarrow \text{card ls} > 0 ... \\
\text{inject}_N_{\beta}: N_{\alpha} \rightarrow \rightarrow N_{\beta} \text{ pre: } \text{wf}_N_{\beta}(hs,ls) \\
\text{inject}_N_{\beta}(n_{\alpha}) \equiv (\text{obs}_Hs(n_{\alpha}), \text{obs}_Ls(n_{\alpha}))
\]

- **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 identifiers to hubs),

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

  d a graph from hub \(h_i\) identifiers \(h_{i_{ij}}\) to maps from identifiers \(l_{ij}\) of hub \(h_i\) connected links \(l_{ij}\) to the identifiers \(h_{ji}\) of link connected hubs \(h_j\).
type
  [sorts] H, HI, L, LI
  [a] Nₜ = HUBS × LINKS × GRAPH
  [b] HUBS = H ⊗ H
  [c] LINKS = L ⊗ L
  [d] GRAPH = H ⊗ (L ⊗ H)

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

• Example 12 on page 117 defines the well-formedness predicates for the above map types.

Variety of Type Definitions

[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' • P(v) |}

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

Record 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)

Types A, B, ..., Z are disjoint, i.e., shares no values, provided all $mk_{id_k}$ are distinct and due to the use of the disjoint record type constructor ==.

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:

19. 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.

20. From a net one can remove a link.\textsuperscript{13} The removal command specifies a link identifier.

\textbf{type}

19  Insert == Ins(s_{ins}:Ins)

19  Ins = 2xHubs \mid 1x1nH \mid 2nHs

19a  2xHubs == 2oldH(s_{hi1}:HI,s_{l}:L,s_{hi2}:HI)

19b  1x1nH == 1oldH1newH(s_{hi}:HI,s_{l}:L,s_{h}:H)

19c  2nHs == 2newH(s_{h1}:H,s_{l}:L,s_{h2}:H)

20  Remove == Rmv(s_{li}:LI)

\textbf{axiom}

19d \ \forall \ 2oldH(hi',l,hi''):Ins \ • \ hi' \neq hi'' \wedge obs_{LIs}(l) = \{hi',hi''\} \land

\forall \ 1oldH1newH(hi,l,h):Ins \ • \ obs_{LIs}(l) = \{hi,obs_{HI}(h)\} \land

\forall \ 2newH(h',l,h''):Ins \ • \ obs_{LIs}(l) = \{obs_{HI}(h'),obs_{HI}(h'')\}

\textbf{RSL Explanation}

\begin{itemize}
\item 19: The type clause \textbf{type} Ins = 2xHubs \mid 1x1nH \mid 2nHs introduces the type name Ins and defines it to be the union (\mid) type of values of either of three types: 2xHubs, 1x1nH and 2nHs.

   \begin{itemize}
   \item 19a): The type clause \textbf{type} 2xHubs == 2oldH(s_{hi1}:HI, s_{l}:L, s_{hi2}:HI) defines the type 2xHubs to be the type of values of record type 2oldH(s_{hi1}:HI, s_{l}:L, s_{hi2}:HI), that is, Cartesian-like, or “tree”-like values with record (root) name 2oldH and with three sub-values, like branches of a tree, of types HI, L and HI. Given a value, \textit{cmd}, of type 2xHubs, applying the selectors \textit{s_{hi1}}, \textit{s_{l}} and \textit{s_{hi2}} to \textit{cmd} yield the corresponding sub-values.
   \end{itemize}
\end{itemize}

\textsuperscript{13} provided that what remains is still a proper net
– 19b): Reading of this type clause is left as exercise to the reader.
– 19c): Reading of this type clause is left as exercise to the reader.
– 19d): The axiom \textit{axiom} has three predicate clauses, one for each category of Insert commands.

\begin{itemize}
\item The first clause: \( \forall 2\text{oldH}(hi',l,hi''):\text{Ins} \cdot hi' \neq hi'' \land \text{obs}_HIs(l) = \{hi', hi''\} \) reads as follows:
  \begin{itemize}
  \item For all record structures, 2oldH(hi',l,hi''), that is, values of type Insert (which in this case is the same as of type 2xHubs),
  \item that is values which can be expressed as a record with root name 2oldH and with three sub-values (“freely”) named hi’, l and hi’’
  \item (where these are bound to be of type HI, L and HI by the definition of 2xHubs),
  \item the two hub identifiers hi’ and hi’’ must be different,
  \item and the hub identifiers observed from the new link, l, must be the two argument hub identifiers hi’ and hi’’.
\end{itemize}
\end{itemize}
\begin{itemize}
\item Reading of the second predicate clause is left as exercise to the reader.
\item Reading of the third predicate clause is left as exercise to the reader.
\end{itemize}

The three types 2xHubs, 1x1nH and 2nHs are disjoint: no value in one of them is the same value as in any of the other merely due to the fact that the record names, 2oldH, 1oldH1newH and 2newH, are distinct. This is no matter what the “bodies” of their record structure is, and they are here also distinct: \((s_{hi1}:HI,s_{l}:L,s_{hi2}:HI)\), \((s_{hi}:HI,s_{l}:L,s_{h}:H)\), respectively \((s_{h1}:H,s_{l}:L,s_{h2}:H)\).

\bullet 20; The type clause \textit{type} Remove == Rmv(s_{li}:LI)

\begin{itemize}
\item (as for Items 19b and 19c))
\item defines a type of record values, say rmv,
\item with record name Rmv and with a single sub-value, say li of type LI
\item where li can be selected from by rmv selector s_{li}.
\end{itemize}

\textbf{End of RSL Explanation}

Example 17 on page 128 presents the semantics functions for \textit{int} Insert and \textit{int} Remove.

\textbf{End of Example 4}

\textbf{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 \( P \), constitute the subtype \( A \):
Subtypes

type
A = {\mid b:B \cdot \mathcal{P}(b) \mid}

**Example 5**
In Example 3 on page 98 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
\[[\text{sorts}]\ N', H, L, H, L\]
N = {\mid n:N' \cdot \text{wf}_N(n)\mid}

value
wf_N: N' \rightarrow \text{Bool}
wf_N(n) \equiv
\forall n:N \cdot \text{card obs}_H(n) \geq 0 \land \text{card obs}_L(n) \geq 0 \land
\text{axioms 2.–3., 5.–6., and 8., (Page 13)}

- 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\text{-set} \times L\text{-set}
N = {\mid n:N' \cdot \text{wf}_N(n)\mid}

- In the Cartesians + Maps + Wellformedness part of Example 3
We now redefine this as follows:

\[
\text{type } \begin{align*}
\text{[sorts]} & \text{ L, H, LI, HI} \\
N' & = \text{HUBS} \times \text{LINKS} \times \text{GRAPH} \\
N & = \{ (hs,ls,g) : N' \land \text{wf\_HUBS}(hs) \land \text{wf\_LINKS}(ls) \land \text{wf\_GRAPH}(g)(hs,ls) \} \\
\text{HUBS}' & = \text{HI} \rightarrow^{m} \text{H} \\
\text{HUBS} & = \{ hs : \text{HUBS}' \land \text{wf\_HUBS}(hs) \} \\
\text{LINKS}' & = \text{LI} \rightarrow \text{L} \\
\text{LINKS} & = \{ ls : \text{LINKS}' \land \text{wf\_LINKS}(ls) \} \\
\text{GRAPH}' & = \text{HI} \rightarrow^{m} \text{LI} \rightarrow^{m} \text{HI} \\
\text{GRAPH} & = \{ g : \text{GRAPH}' \land \text{wf\_GRAPH}(g) \} \\
\text{value} & \begin{align*}
\text{wf\_GRAPH} : \text{GRAPH}' & \rightarrow (\text{HUBS} \times \text{LINKS}) \rightarrow \text{Bool} \\
\text{wf\_GRAPH}(g)(hs,ls) & \equiv \text{wf\_N}(hs,ls,g)
\end{align*}
\]

Example 12 on page 117 presents a definition of \text{wf\_GRAPH}.

\textbf{End of Example 5}

\section*{Sorts — Abstract Types}

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

\textbf{Sorts}

\textbf{type}

\begin{align*}
\text{A, B, ..., C}
\end{align*}

\textbf{Example 6} \hspace{1cm} \textbf{Net Sorts:}

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

\textbf{End of Example 6}

\section*{A.2 Concrete RSL Types: Values and Operations}

\subsection*{A.2.1 Arithmetic}
### Arithmetic

- **type**
  - Nat, Int, Real

- **value**
  - `+,−,*: Nat×Nat→Nat | Int×Int→Int | Real×Real→Real`
  - `/: Nat×Nat→Nat | Int×Int→Int | Real×Real→Real`
  - `<,≤,=,≠,≥,> (Nat|Int|Real) × (Nat|Int|Real) → Bool`

### A.2.2 Set Expressions

#### Set Enumerations

Let the below a’s denote values of type A, then the below designate simple set enumerations:

**Set Enumerations**

\[
\begin{align*}
\{\} , \{a\}, \{e_1,e_2,...,e_n\}, ... & \subseteq A\text{-set} \\
\{\} , \{a\}, \{e_1,e_2,...,e_n\}, ..., \{e_1,e_2,...\} & \subseteq A\text{-infset}
\end{align*}
\]

#### Example 7 .......................... Set Expressions over Nets:

We now consider hubs to abstract cities, towns, villages, etcetera. Thus with hubs we can associate sets of citizens.

Let c:C stand for a citizen value c being an element in the type C of all such. Let g:G stand for any (group) of citizens, respectively the type of all such. Let s:S stand for any set of groups, respectively the type of all such. Two otherwise distinct groups are related to one another if they share at least one citizen, the liaisons. A network nw:NW is a set of groups such that for every group in the network one can always find another group with which it shares liaisons.

Solely using the set data type and the concept of subtypes, we can model the above:

**type**

- C
  - G’ = C-set, G = { | g:G’ • g≠{ } | }
  - S = G-set

- L’ = C-set, L = { | ℓ:L’ • ℓ≠{} | }
  - NW’ = S, NW = { | s:S • wf_S(s) | }

**value**

- `wf_S: S → Bool`
  - `wf_S(s) ≡ ∀ g:G • g ∈ s ⇒ ∃ g’:G • g’ ∈ s ∧ share(g,g’)`

- `share: G×G → Bool`
  - `share(g,g’) ≡ g≠g’ ∧ g ∩ g’ ≠ {}`

- `liaisons: G×G → L`
  - `liaisons(g,g’) = g ∩ g’ pre share(g,g’)`
Annotations: L stands for proper liaisons (of at least one liaison). G’, L’ and N’ are the “raw” types which are constrained to G, L and N. \{ binding:type\_expr \bowtie bool\_expr \} is the general form of the subtype expression. For G and L we state the constraints “in-line”, i.e., as direct part of the subtype expression. For NW we state the constraints by referring to a separately defined predicate. \text{wf}_S(s) expresses — through the auxiliary predicate — that \textit{s} contains at least two groups and that any such two groups share at least one citizen. liaisons is a “truly” auxiliary function in that we have yet to “find an active need” for this function!

The idea is that citizens can be associated with more than one city, town, village, etc. (primary home, summer and/or winter house, working place, etc.). A group is now a set of citizens related by some “interest” (Rotary club membership, political party “grassroots”, religion, et.). The reader is invited to define, for example, such functions as: The set of groups (or networks) which are represented in all hubs [or in only one hub]. The set of hubs whose citizens partake in no groups [respectively networks]. The group [network] with the largest coverage in terms of number of hubs in which that group [network] is represented.

End of Example 7

**Set Comprehension**

The expression, last line below, to the right of the \(\equiv\), expresses set comprehension. The expression “builds” the set of values satisfying the given predicate. It is abstract in the sense that it does not do so by following a concrete algorithm.

**Set Comprehension**

\[ type \]
\[ A, B \]
\[ P = A \rightarrow \text{Bool} \]
\[ Q = A \sim \sim B \]

\[ value \]
\[ \text{comprehend}: A\text{-infset} \times P \times Q \rightarrow B\text{-infset} \]
\[ \text{comprehend}(s,P,Q) \equiv \{ Q(a) \mid a:A \cdot a \in s \land P(a) \} \]

**Example 8**

**Set Comprehensions:**

Item 48 on page 22, the \(\text{wf}_N(hs,ls,g)\) wellformedness predicate definition, includes:

\[ type \]
\[ 47a. \text{PLAN} = \text{HI} \overset{m}{\rightarrow} \text{LHIM} \]
\[ 47b. \text{LHIM} = \text{LI} \overset{m}{\rightarrow} \text{HI-set} \]

\[ value \]
\[ 48c. \text{no\_junk}: \text{PLAN} \rightarrow \text{Bool} \]
\[ 48c. \text{no\_junk}(plan) \equiv \text{dom plan} = \bigcup\{ \text{rng}(\text{plan}(hi)) \mid hi: \text{HI}\cdot hi \in \text{dom plan} \} \]
It expresses the distributed union of sets \((\text{rng}(\text{plan}(\text{li})))\) of hub identifiers (for each of the \(hi\) indexed maps from (definition set, \text{dom}) link identifiers to (range set, \text{rng}) hub identifiers, where \(hi:HI\) ranges over \text{dom plan}).

End of Example 8

A.2.3 Cartesian Expressions

Cartesian Enumerations Let \(e\) range over values of Cartesian types involving \(A, B, \ldots, C\), then the below expressions are simple Cartesian enumerations:

Cartesian Enumerations

type
\[
A, B, \ldots, C
\]
\[
A \times B \times \ldots \times C
\]

value
\[
(e_1, e_2, \ldots, e_n)
\]

Example 9 Cartesian Net Types:

So far we have abstracted hubs and links as sorts. That is, we have not defined their types concretely. Instead we have postulated some attributes such as: observable hub identifiers of hubs and sets of observable link identifiers of links connected to hubs. We now claim the following further attributes of hubs and links.

- **Concrete links have**
  - link identifiers,
  - link names – where two or more connected links may have the same link name,
  - two (unordered) hub identifiers,
  - lengths,
  - locations – where we do not presently defined what we main by locations,
  - etcetera

- **Concrete hubs have**
  - hub identifiers,
  - unique hub names,
  - a set of one or more observable link identifiers,
  - locations,
  - etcetera.
**An RSL Primer**

**A.2.4 List Expressions**

**List Enumerations**  Let \( a \) range over values of type \( A \), then the below expressions are simple list enumerations:

\[
\{\langle \rangle, \langle e \rangle, \ldots, \langle e_1, e_2, \ldots, e_n \rangle, \ldots\} \subseteq A^*
\]

\[
\{\langle \rangle, \langle e \rangle, \ldots, \langle e_1, e_2, \ldots, e_n\rangle, \ldots, \langle e_1, e_2, \ldots, e_m, \ldots \rangle, \ldots\} \subseteq A^\omega
\]

\[
\langle a_{i..j} \rangle
\]

The last line above assumes \( a_i \) and \( a_j \) to be integer-valued expressions. It then expresses the set of integers from the value of \( e_i \) to and including the value of \( e_j \). If the latter is smaller than the former, then the list is empty.

**List Comprehension**

The last line below expresses list comprehension.

\[
\langle a_{i..j} \rangle
\]

\[
\langle \langle \ldots \rangle \mid \text{some conditions} \rangle
\]

\[
\langle \ldots \rangle \cdot P(l(i))
\]

**Example 10**

**Routes in Nets:**

- A phenomenological (i.e., a physical) route of a net is a sequence of one or more adjacent links of that net.
- A conceptual route is a sequence of one or more link identifiers.
- An abstract route is a conceptual route.
for which there is a phenomenological route of the net for which the link identifiers of the abstract route map one-to-one onto links of the phenomenological route.

\[
\begin{align*}
\text{type} & \quad \text{N, H, L, HL, LI} \\
PR' &= L^* \\
PR &= \{ | \text{pr:PR' } \exists n: N \cdot \text{wf}_{PR}(pr)(n) \} \\
CR &= L^* \\
AR' &= L^* \\
AR &= \{ | \text{ar:AR' } \exists n: N \cdot \text{wf}_{AR}(ar)(n) \} \\
\text{value} & \quad \text{wf}_{PR}: PR' \rightarrow N \rightarrow \text{Bool} \\
\text{wf}_{PR}(pr)(n) & \equiv \\
& \quad \forall i: \text{Nat} \cdot \{ i, i+1 \} \subseteq \text{inds} \ pr \Rightarrow \text{obs}_{Ls}(l(i)) \cap \text{obs}_{Ls}(l(i+1)) \neq \{} \\
\text{wf}_{AR'}: AR' \rightarrow N \rightarrow \text{Bool} \\
\text{wf}_{AR}(ar)(n) & \equiv \\
& \quad \exists \text{pr:PR' } \text{pr} \in \text{routes}(n) \wedge \text{wf}_{PR}(pr)(n) \wedge \text{len}_{pr} = \text{len}_{ar} \wedge \\
& \quad \forall i: \text{Nat} \cdot i \in \text{inds} \ ar \Rightarrow \text{obs}_{Ll}(pr(i)) = ar(i) \\
\end{align*}
\]

- A single link is a phenomenological route.

- If \( r \) and \( r' \) are phenomenological routes
  - such that the last link \( r \)
  - and the first link of \( r' \)
  - share observable hub identifiers,

then the concatenation \( r \circ r' \) is a route.

This inductive definition implies a recursive set comprehension.

- A circular phenomenological route is a phenomenological route whose first and last links are distinct but share hub identifiers.

- A looped phenomenological route is a phenomenological route where two distinctly positioned (i.e., indexed) links share hub identifiers.

\[
\begin{align*}
\text{value} & \quad \text{routes}: N \rightarrow \text{PR-infset} \\
\text{routes}(n) & \equiv \\
& \quad \text{let prs} = \{ \langle l \rangle | L \cdot \text{obs}_{Ls}(n) \} \cup \\
\end{align*}
\]
\[ \cup \{ pr^r | pr', pr : PR \cup \{ pr, pr' \} \subseteq \text{prs} \land \text{obs\_Hls}(r(\text{len} pr)) \land \text{obs\_Hls}(pr'(1)) \neq \{ \} \} \]

\text{prs end}

\text{is\_circular: } PR \rightarrow \text{Bool}
\text{is\_circular}(pr) \equiv \text{obs\_Hls}(pr(1)) \land \text{obs\_Hls}(pr(\text{len} pr)) \neq \{ \}

\text{is\_looped: } PR \rightarrow \text{Bool}
\text{is\_looped}(pr) \equiv \exists i, j : \text{Nat} \cdot i \neq j \land \{ i, j \} \subseteq \text{index pr} \Rightarrow \text{obs\_Hls}(pr(i)) \land \text{obs\_Hls}(pr(j)) \neq \{ \}

- Straight routes are Phenomenological routes without loops.
- Phenomenological routes with no loops can be constructed from phenomenological routes by removing suffix routes whose first link give rise to looping.

\text{value}
\text{straight\_routes: } N \rightarrow \text{PR-set}
\text{straight\_routes}(n) \equiv
\text{let prs = routes}(n) \text{ in } \{ \text{straight\_route}(pr) | pr:PR \cup \text{ps} \in \text{prs} \} \text{ end}

\text{straight\_route: } PR \rightarrow PR
\text{straight\_route}(pr) \equiv \langle pr(i) | i : \text{Nat} \cdot i \in [1..\text{len} pr] \land pr(i) \notin \text{elems}(pr(j) | j : \text{Nat} \cdot j \in [1..i]) \rangle

\text{End of Example 10}

A.2.5 \hspace{1cm} \textbf{Map Expressions} \hspace{1cm} 307

**Map Enumerations** Let (possibly indexed) \( u \) and \( v \) range over values of type \( T_1 \) and \( T_2 \), respectively, then the below expressions are simple map enumerations:

\text{Map Enumerations}

\text{type}
\text{T}_1, \text{T}_2
M = T_1 \rightarrow m T_2

\text{value}
u, u_1, u_2, ..., u_n : T_1, v, v_1, v_2, ..., v_n : T_2
\{ [], [ u \mapsto v ], ... , [ u_1 \mapsto v_1, u_2 \mapsto v_2, ..., u_n \mapsto v_n ], ... \} \subseteq M

**Map Comprehension** 308

The last line below expresses map comprehension:
Map Comprehension

type
U, V, X, Y
M = U → m V
F = U ⊳ X
G = V ⊳ Y
P = U → Bool

value
comprehend: M × F × G × P → (X m Y)
comprehend(m,F,G,P) ≡
[F(u) ↦ G(m(u)) | u:U • u ∈ dom m ∧ P(u)]

Example 11
Concrete Net Type Construction:

- We Define a function con[struct]_N_γ_ (of the Cartesians + Maps + Wellformedness part of Example 3).
  - The base of the construction is the fully abstract sort definition of N_α in the Sorts + Observers + Axioms part of Example 3 — where the sorts of hub and link identifiers are taken from earlier examples.
  - The target of the construction is the N_γ of the Cartesians + Maps + Wellformedness part of Example 3.
- First we recall the essential types of that N_γ.

```
type
  N_γ = HUBS × LINKS × GRAPH
  HUBS = HI m H
  LINKS = LI m L
  GRAPH = HI m (LI m HI)

value
  con_N_γ: N_α → N_γ
  con_N_γ(n_α) ≡
  let hubs = [obs_HI(h) ↦ h | h:H • h ∈ obs_Hs(n_α)],
  links = [obs_HI(h) ↦ l | l:L • l ∈ obs_Ls(n_α)],
  graph = [obs_HI(h) ↦ [obs_HI(l) ↦ τ(obs_HI(l) \{obs_HI(h)}) | l:L • l ∈ obs_Ls(n_α) ∧ li ∈ obs_Lls(h)]]
  in
  (hubs,links,graph)
```

τ: A-set ⊳ A [A could be LI-set]
τ(as) ≡ if card as=1 then let {a}=as in a else chaos end end
theorem:
\( n_\alpha \) satisfies axioms 2.–3., 5.–6., and 8. (Page 13) \( \Rightarrow \) \( w_\gamma N (\text{con} N_\gamma (n_\alpha)) \)

End of Example 11

A.2.6 Set Operations

Set Operator Signatures

Set Operations

value

21 \( \in \): \( A \times A\)-infset \( \rightarrow \) Bool
22 \( \notin \): \( A \times A\)-infset \( \rightarrow \) Bool
23 \( \cup \): \( A\)-infset \( \times \) \( A\)-infset \( \rightarrow \) \( A\)-infset
24 \( \cup \): \( (A\)-infset\)-infset \( \rightarrow \) \( A\)-infset
25 \( \cap \): \( A\)-infset \( \times \) \( A\)-infset \( \rightarrow \) \( A\)-infset
26 \( \cap \): \( (A\)-infset\)-infset \( \rightarrow \) \( A\)-infset
27 \( \setminus \): \( A\)-infset \( \times \) \( A\)-infset \( \rightarrow \) \( A\)-infset
28 \( \subseteq \): \( A\)-infset \( \times \) \( A\)-infset \( \rightarrow \) Bool
29 \( \subseteq \): \( A\)-infset \( \times \) \( A\)-infset \( \rightarrow \) Bool
30 \( \neq \): \( A\)-infset \( \times \) \( A\)-infset \( \rightarrow \) Bool
31 \( \neq \): \( A\)-infset \( \times \) \( A\)-infset \( \rightarrow \) Bool
32 \( \card \): \( A\)-infset \( \rightarrow \) Nat

Set Examples

Set Examples

elements

\( a \in \{a,b,c\} \)
\( a \notin \{\}, a \notin \{b,c\} \)
\( \{a,b,c\} \cup \{a,b,d,e\} = \{a,b,c,d,e\} \)
\( \cup \{a\}, \{a,b\}, \{a,d\} = \{a,b,d\} \)
\( \{a,b,c\} \cap \{c,d,e\} = \{c\} \)
\( \cap \{a\}, \{a,b\}, \{a,d\} = \{a\} \)
\( \{a,b,c\} \setminus \{c,d\} = \{a\} \)
\( \{a,b\} \subset \{a,b,c\} \)
\( \{a,b,c\} \subseteq \{a,b,c\} \)
\( \{a,b,c\} = \{a,b,c\} \)
\( \{a,b,c\} \neq \{a,b\} \)
\( \card \{\} = 0, \card \{a,b,c\} = 3 \)
Informal Explication

21. $\in$: The membership operator expresses that an element is a member of a set.

22. $\notin$: The non-membership operator expresses that an element is not a member of a set.

23. $\cup$: The infix union operator. When applied to two sets, the operator gives the set whose members are in either or both of the two operand sets.

24. $\bigcup$: The distributed prefix union operator. When applied to a set of sets, the operator gives the set whose members are in some of the operand sets.

25. $\cap$: The infix intersection operator. When applied to two sets, the operator gives the set whose members are in both of the two operand sets.

26. $\bigcap$: The prefix distributed intersection operator. When applied to a set of sets, the operator gives the set whose members are in some of the operand sets.

27. $\setminus$: The set complement (or set subtraction) operator. When applied to two sets, the operator gives the set whose members are those of the left operand set which are not in the right operand set.

28. $\subseteq$: The proper subset operator expresses that all members of the left operand set are also in the right operand set.

29. $\subset$: The proper subset operator expresses that all members of the left operand set are also in the right operand set, and that the two sets are not identical.

30. $=:=$: The equal operator expresses that the two operand sets are identical.

31. $\neq$: The non-equal operator expresses that the two operand sets are not identical.

32. $\text{card}$: The cardinality operator gives the number of elements in a finite set.

Set Operator Definitions

The operations can be defined as follows ($\equiv$ is the definition symbol):

Set Operation Definitions

value

\begin{align*}
s' \cup s'' & \equiv \{ a : a:A \land a \in s' \lor a \in s'' \} \\
s' \cap s'' & \equiv \{ a : a:A \land a \in s' \land a \in s'' \} \\
s' \setminus s'' & \equiv \{ a : a:A \land a \in s' \land a \notin s'' \} \\
s' \subseteq s'' & \equiv \forall a:A : a \in s' \Rightarrow a \in s'' \\
s' \subset s'' & \equiv s' \subseteq s'' \land \exists a:A : a \in s'' \land a \notin s'
\end{align*}
$s' = s'' \equiv \forall a: A \rightarrow a \in s' \equiv a \in s'' \equiv s' \subseteq s' \wedge s' \subseteq s$

$s' \neq s'' \equiv s' \cap s'' \neq \{\}$

$\text{card } s \equiv$

\[
\begin{align*}
\text{if } s = \{\} \text{ then } 0 \text{ else} \\
\text{let } a: A \rightarrow a \in s \text{ in } 1 + \text{card } (s \setminus \{a\}) \text{ end end}
\end{align*}
\]

$\text{pre } s /* \text{is a finite set */}$

$\text{card } s \equiv \text{chaos } /* \text{tests for infinity of } s */$

### A.2.7 Cartesian Operations

#### Cartesian Operations

**type**

- $A, B, C$
- $g0: G0 = A \times B \times C$
- $g1: G1 = (A \times B \times C)$
- $g2: G2 = (A \times B) \times C$
- $g3: G3 = A \times (B \times C)$

**value**

- $va:A, vb:B, vc:C, vd:D$
- $(va, vb, vc):G0,$
- $(va, vb, vc):G1$
- $((va, vb), vc):G2$
- $(va3, (vb3, vc3)):G3$

**decomposition expressions**

- let $(a1, b1, c1) = g0,$
- $(a1', b1', c1') = g1$ in .. end
- let $((a2, b2), c2) = g2$ in .. end
- let $(a3, (b3, c3)) = g3$ in .. end

### A.2.8 List Operations

#### List Operator Signatures

**List Operations**

**value**

- $\text{hd: } A^\omega \rightarrow A$
- $\text{tl: } A^\omega \rightarrow A^\omega$
- $\text{len: } A^\omega \rightarrow \text{Nat}$
- $\text{inds: } A^\omega \rightarrow \text{Nat-infset}$
- $\text{elems: } A^\omega \rightarrow A\text{-infset}$
- $\text{(.): } A^\omega \times \text{Nat} \rightarrow A$
- $\hat{\cdot}: A^* \times A^\omega \rightarrow A^\omega$
- $=: A^\omega \times A^\omega \rightarrow \text{Bool}$
- $\neq: A^\omega \times A^\omega \rightarrow \text{Bool}$
List Examples

examples

\begin{align*}
\text{hd}(a_1,a_2,\ldots,a_m) &= a_1 \\
\text{tl}(a_1,a_2,\ldots,a_m) &= (a_2,\ldots,a_m) \\
\text{len}(a_1,a_2,\ldots,a_m) &= m \\
\text{inds}(a_1,a_2,\ldots,a_m) &= \{1,2,\ldots,m\} \\
\text{elems}(a_1,a_2,\ldots,a_m) &= \{a_1,a_2,\ldots,a_m\} \\
\langle a_1,a_2,\ldots,a_m\rangle(i) &= a_i \\
\langle a,b,c\rangle \hat{\langle} a,b,d \rangle &= \langle a,b,c,a,b,d \rangle \\
\langle a,b,c\rangle &= \langle a,b,c \rangle \\
\langle a,b,c\rangle &\neq \langle a,b,d \rangle
\end{align*}

Informal Explication

- **\text{hd}:** Head gives the first element in a nonempty list.
- **\text{tl}:** Tail gives the remaining list of a nonempty list when Head is removed.
- **\text{len}:** Length gives the number of elements in a finite list.
- **\text{inds}:** Indices give the set of indices from 1 to the length of a nonempty list. For empty lists, this set is the empty set as well.
- **\text{elems}:** Elements gives the possibly infinite set of all distinct elements in a list.
- **\ell(i):** Indexing with a natural number, \( i \) larger than 0, into a list \( \ell \) having a number of elements larger than or equal to \( i \), gives the \( i \)th element of the list.
- **\hat{\langle}:** Concatenates two operand lists into one. The elements of the left operand list are followed by the elements of the right. The order with respect to each list is maintained.
- **=:** The equal operator expresses that the two operand lists are identical.
- **\neq:*** The non-equal operator expresses that the two operand lists are not identical.

The operations can also be defined as follows:

**List Operator "Definitions"**

\begin{align*}
\text{value} \\
\text{is\_finite\_list: } & A^\omega \rightarrow \text{Bool} \\
\text{len } q &\equiv \text{case } \text{is\_finite\_list}(q) \text{ of }
\end{align*}
true → if q = ⟨⟩ then 0 else 1 + len tl q end,
false → chaos end

inds q ≡
case is_finite_list(q) of
  true → { i : i:Nat • 1 ≤ i ≤ len q },
  false → { i : i:Nat • i≠0 } end

elems q ≡ { q(i) | i:Nat • i ∈ inds q }

q(i) ≡
case (q,i) of
  ⟨⟩,1 → chaos,
  (_1) → let a:A,q′:Q • q=(a)ˆq′ in a end
  _ → q(i-1)

fq ˆ iq ≡
  ⟨ if 1 ≤ i ≤ len fq then fq(i) else iq(i - len fq) end |
  i:Nat • if len iq≠chaos then i ≤ len fq+len end ⟩

pre is_finite_list(fq)

iq′ = iq” ≡
inds iq’ = inds iq” ∧ ∀ i:Nat • i ∈ inds iq’ ⇒ iq’(i) = iq”(i)

iq’ ≠ iq” ≡ ∼(iq’ = iq”)

A.2.9 Map Operations

Map Operator Signatures and Map Operation Examples

value
m(a): M → A ∼ B, m(a) = b

dom: M → A-infset [domain of map]
dom [a1→b1,a2→b2,...,an→bn] = {a1,a2,...,an}

rng: M → B-infset [range of map]
rng [a1→b1,a2→b2,...,an→bn] = {b1,b2,...,bn}

†: M × M → M [override extension]
[a→b,a′→b′,a''→b’’] † [ a’’→b”,a’’→b’” ] = [ a→b,a′’→b”,a’’→b’”]
\[ \cup : M \times M \to M [ \text{merge } \cup ] \\
= \left[ a \mapsto b, a' \mapsto b', a'' \mapsto b'' \right] \cup \left[ a'''' \mapsto b''''' \right] = \left[ a \mapsto b, a' \mapsto b', a'' \mapsto b'', a'''' \mapsto b''''' \right] \]

\[ \setminus : M \times A-\text{infset} \to M [ \text{restriction by} ] \\
= \left[ a \mapsto b, a' \mapsto b', a'' \mapsto b'' \right] \setminus \{ a \} = \left[ a' \mapsto b', a'' \mapsto b'' \right] \]

\[ / : M \times A-\text{infset} \to M [ \text{restriction to} ] \\
= \left[ a \mapsto b, a' \mapsto b', a'' \mapsto b'' \right] / \{ a', a'' \} = \left[ a' \mapsto b', a'' \mapsto b'' \right] \]

\[ =, \neq : M \times M \to \text{Bool} \]

\[ \circ : (A \overline{m} B) \times (B \overline{m} C) \to (A \overline{m} C) [ \text{composition} ] \\
= \left[ a \mapsto b, a' \mapsto b' \right] \circ \left[ b \mapsto c, b' \mapsto c', b'' \mapsto c'' \right] = \left[ a \mapsto c, a' \mapsto c' \right] \]

**Map Operation Explication**

- **\( m(a) \)**: Application gives the element that \( a \) maps to in the map \( m \).
- **\( \text{dom} \)**: Domain/Definition Set gives the set of values which maps to in a map.
- **\( \text{rng} \)**: Range/Image Set gives the set of values which are mapped to in a map.
- **\( \dagger \)**: Override/Extend. When applied to two operand maps, it gives the map which is like an override of the left operand map by all or some “pairings” of the right operand map.
- **\( \cup \)**: Merge. When applied to two operand maps, it gives a merge of these maps.
- **\( \setminus \)**: Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements that are not in the right operand set.
- **\( / \)**: Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements of the right operand set.
- **\( =, \neq \)**: The equal operator expresses that the two operand maps are identical.
- **\( \neq \)**: The non-equal operator expresses that the two operand maps are not identical.
- **\( \circ \)**: Composition. When applied to two operand maps, it gives the map from definition set elements of the left operand map, \( m_1 \), to the range elements of the right operand map, \( m_2 \), such that if \( a \) is in the definition set of \( m_1 \) and maps into \( b \), and if \( b \) is in the definition set of \( m_2 \) and maps into \( c \), then \( a \), in the composition, maps into \( c \).
Example 12: Miscellaneous Net Expressions: Maps

Example 3 on page 98 left out defining the well-formedness of the map types:

type
\[
\text{GRAPH} = HI \overset{m}{\to} (LI \overset{m}{\to} HI)\text{-set}
\]
\[
\text{HUBS} = HI \overset{m}{\to} H
\]
\[
\text{LINKS} = LI \overset{m}{\to} L
\]
\[
\text{N}_\gamma = \text{HUBS} \times \text{LINKS} \times \text{GRAPH}
\]

value
\[
\begin{align*}
\text{wf}_\text{HUBS}: H\text{-set} &\to \text{Bool} \\
\text{wf}_\text{HUBS}(hubs) &\equiv \forall hi:HI \mid hi \in \text{dom} \; \text{hubs} \Rightarrow \text{obs}_\text{HI}(\text{hubs}(hi)) = hi \\
\text{wf}_\text{LINKS}: L\text{-set} &\to \text{Bool} \\
\text{wf}_\text{LINKS}(links) &\equiv \forall li:LI \mid li \in \text{dom} \; \text{links} \Rightarrow \text{obs}_\text{LI}(\text{links}(li)) = li \\
\text{wf}_\text{N}_\gamma: \text{N}_\gamma &\to \text{Bool} \\
\text{wf}_\text{N}_\gamma(hs,ls,g) &\equiv \\
&\quad \text{dom} \; hs = \text{dom} \; g \\
&\quad \bigcup \{\text{dom} \; g(hi)|hi:HI \mid hi \in \text{dom} \; g\} = \text{dom} \; \text{links} \\
&\quad \bigcup \{\text{rng} \; g(hi)|hi:HI \mid hi \in \text{dom} \; g\} = \text{dom} \; g \\
&\quad \forall hi:HI \mid hi \in \text{dom} \; g \Rightarrow \forall li:LI \mid li \in \text{dom} \; g(hi) \Rightarrow (g(hi))(li) \neq hi \\
&\quad \forall hi:HI \mid hi \in \text{dom} \; g \Rightarrow \exists li:LI \mid li \in \text{dom} \; g(hi) \Rightarrow (g(hi))(li) = hi' \land (g(hi'))(li) = hi
\end{align*}
\]

- Hubs record the same hubs as do the net corresponding \text{GRAPH} (\text{dom} \; hs = \text{dom} \; g)\land).
- \text{GRAPH} record the same links as do the net corresponding \text{LINKS} (\bigcup \{\text{dom} \; g(hi)|hi:HI \mid hi \in \text{dom} \; g\} = \text{dom} \; \text{links}).
- The target (or range) hub identifiers of graphs are the same as the domain of the graph (\bigcup \{\text{rng} \; g(hi)|hi:HI \mid hi \in \text{dom} \; g\} = \text{dom} \; g), that is none missing, no new ones!
- No links emanate from and are incident upon the same hub (\forall hi:HI \mid hi \in \text{dom} \; g \Rightarrow \forall li:LI \mid li \in \text{dom} \; g(hi) \Rightarrow (g(hi))(li) = hi').
- If there is a link from one hub to another in the \text{GRAPH}, then the same link also connects the other hub to the former (\forall hi:HI \mid hi \in \text{dom} \; g \Rightarrow \forall li:LI \mid li \in \text{dom} \; g(hi) \Rightarrow \exists hi':HI \mid hi' \in \text{dom} \; g \Rightarrow \exists li:LI \mid li \in \text{dom} \; g(hi) \Rightarrow (g(hi))(li) = hi' \land (g(hi'))(li) = hi).

End of Example 12

Map Operation “Redefinitions”

The map operations can also be defined as follows:
value

\[ \text{rng } m \equiv \{ m(a) \mid a:A \cdot a \in \text{dom } m \} \]

\[ m_1 \uparrow m_2 \equiv \begin{bmatrix} a \rightarrow b \mid a:A,b:B \cdot \text{a } \in \text{dom } m_1 \setminus \text{dom } m_2 \land b=m_1(a) \lor a \in \text{dom } m_2 \land b=m_2(a) \end{bmatrix} \]

\[ m_1 \cup m_2 \equiv \begin{bmatrix} a \rightarrow b \mid a:A,b:B \cdot \text{a } \in \text{dom } m_1 \land b=m_1(a) \lor a \in \text{dom } m_2 \land b=m_2(a) \end{bmatrix} \]

\[ m \setminus s \equiv \begin{bmatrix} a \rightarrow m(a) \mid a:A \cdot a \in \text{dom } m \setminus s \end{bmatrix} \]

\[ m / s \equiv \begin{bmatrix} a \rightarrow m(a) \mid a:A \cdot a \in \text{dom } m \cap s \end{bmatrix} \]

\[ m_1 = m_2 \equiv \text{dom } m_1 = \text{dom } m_2 \land \forall a:A \cdot a \in \text{dom } m_1 \Rightarrow m_1(a) = m_2(a) \]

\[ m_1 \neq m_2 \equiv \sim(m_1 = m_2) \]

\[ m \circ n \equiv \begin{bmatrix} a \rightarrow c \mid a:A,c:C \cdot a \in \text{dom } m \land c = n(m(a)) \end{bmatrix} \]

\[ \text{pre rng } m \subseteq \text{dom } n \]

A.3 **The RSL Predicate Calculus**

A.3.1 **Propositional Expressions**

Let identifiers (or propositional expressions) \( a, b, \ldots, c \) designate Boolean values (\textbf{true} or \textbf{false} [or \textbf{chaos}]). Then:

**Propositional Expressions**

- \textbf{false}, \textbf{true}
- \( a, b, \ldots, c \sim a, a \land b, a \lor b, a \Rightarrow b, a \equiv b \)

are propositional expressions having Boolean values. \( \sim, \land, \lor, \Rightarrow, = \) and \( \neq \) are Boolean connectives (i.e., operators). They can be read as: \textit{not}, \textit{and}, \textit{or}, \textit{if then} (or \textit{implies}), \textit{equal} and \textit{not equal}.

A.3.2 **Simple Predicate Expressions**

Let identifiers (or propositional expressions) \( a, b, \ldots, c \) designate Boolean values, let \( x, y, \ldots, z \) (or term expressions) designate non-Boolean values and let \( i, j, \ldots, k \) designate number values, then:
Simple Predicate Expressions

false, true
a, b, ..., c
\(~a, a\land b, a\lor b, a\Rightarrow b, a=b, a\neq b\)
x=y, x\neq y,
i<j, i\leq j, i\geq j, i\neq j, i\geq j, i>j

are simple predicate expressions.

A.3.3 Quantified Expressions

Let X, Y, ..., C be type names or type expressions, and let P(x), Q(y) and R(z) designate predicate expressions in which x, y and z are free. Then:

Quantified Expressions

\[\forall x:X \cdot P(x)\]
\[\exists y:Y \cdot Q(y)\]
\[\exists! z:Z \cdot R(z)\]

are quantified expressions — also being predicate expressions.

They are “read” as: For all x (values in type X) the predicate P(x) holds; there exists (at least) one y (value in type Y) such that the predicate Q(y) holds; and there exists a unique z (value in type Z) such that the predicate R(z) holds.

Example

Predicates Over Net Quantities:

From earlier examples we show some predicates:

- Example 1: Right hand side of function definition is_two_way_link(l):
  \[\exists l:\Sigma \cdot l\sigma \in \text{obs}_H\Sigma(l) \land \text{card} l\sigma = 2\]

- Example 3:
  - The \textbf{Sorts + Observers + Axioms} part:
    * Right hand side of the wellformedness function wf_{N}(n) definition:
      \[\forall n:N \cdot \text{card obs}_{Hs}(n)\geq 2 \land \text{card obs}_{Ls}(n)\geq 1 \land \text{axioms 2.–3., 5.–6., and 8., (Page 13)}\]
    * Right hand side of the wellformedness function wf_{N}(hs,ls) definition:
      \[\text{card } hs\geq 2 \land \text{card } ls\geq 1 \ldots\]
  - The \textbf{Cartesians + Maps + Wellformedness} part:
    * Right hand side of the \textit{wf}_{HUBS} wellformedness function definition:
      \[\forall hi:HI \cdot hi \in \text{dom hubs} \Rightarrow \text{obs}_Hhi\text{hubs}(hi)=hi\]
Right hand side of the $\text{wf LINKS}$ wellformedness function definition:

\[
\forall li:LI \cdot li \in \text{dom links} \Rightarrow \text{obs LINKS}(li) = li
\]

Right hand side of the $\text{wf } N(hs,ls,g)$ wellformedness function definition:

\[
\begin{align*}
[c] & \quad \text{dom } hs = \text{dom } g \\
[d] & \quad \cup \{ \text{dom } g(hi) | hi:HL \cdot hi \in \text{dom } g \} = \text{dom links} \\
[e] & \quad \cup \{ \text{rng } g(hi) | hi:HL \cdot hi \in \text{dom } g \} = \text{dom } g \\
[f] & \quad \forall hi:HL \cdot hi \in \text{dom } g \Rightarrow \forall li:LI \cdot li \in \text{dom } g(hi) \Rightarrow (g(hi))(li) \neq hi \\
[g] & \quad \forall hi:HL \cdot hi \in \text{dom } g \Rightarrow \forall li:LI \cdot li \in \text{dom } g(hi) \Rightarrow \\
 & \quad \exists hi':HL \cdot hi' \in \text{dom } g \Rightarrow \exists ! li:LI \cdot li \in \text{dom } g(hi) \Rightarrow \\
 & \quad (g(hi))(li) = hi' \land (g(hi'))(li) = hi
\end{align*}
\]

End of Example 13

A.4 $\lambda$-Calculus + Functions

A.4.1 The $\lambda$-Calculus Syntax

$\lambda$-Calculus Syntax

\begin{verbatim}
| * A BNF Syntax: */
| (L) ::= ⟨V⟩ | ⟨F⟩ | ⟨A⟩ | ⟨(A)⟩
| (V) ::= /* variables, i.e. identifiers */
| (F) ::= λ⟨V⟩ • ⟨L⟩
| (A) ::= ⟨(L)(L)⟩

value /* Examples */

| (L): e, f, a, ...
| (V): x, ...
| (F): λ x • e, ...
| (A): f a, (f a), f(a), (f)(a), ...
\end{verbatim}

A.4.2 Free and Bound Variables

Free and Bound Variables Let $x, y$ be variable names and $e, f$ be $\lambda$-expressions.

- (V): Variable $x$ is free in $x$.
- (F): $x$ is free in $\lambda y \cdot e$ if $x \neq y$ and $x$ is free in $e$.
- (A): $x$ is free in $f(e)$ if it is free in either $f$ or $e$ (i.e., also in both).
A.4.3 Substitution

In RSL, the following rules for substitution apply:

Substitution of an expression $N$ for all free free $x$ in $M$ is expressed: $\text{subst}([N/x]M)$.

**Substitution**

- $\text{subst}([N/x]x) \equiv N$;
- $\text{subst}([N/x]a) \equiv a$, for all variables $a \neq x$;
- $\text{subst}([N/x](P \ Q)) \equiv (\text{subst}([N/x]P) \ \text{subst}([N/x]Q))$;
- $\text{subst}([N/x](\lambda x \cdot P)) \equiv \lambda y \cdot \text{subst}([N/x]P)$, if $x \neq y$ and $y$ is not free in $N$ or $x$ is not free in $P$;
- $\text{subst}([N/x](\lambda y \cdot P)) \equiv \lambda z \cdot \text{subst}([N/z]\text{subst}([z/y]P))$, if $y \neq x$ and $y$ is free in $N$ and $x$ is free in $P$ (where $z$ is not free in $(N \ P)$).

A.4.4 $\alpha$-Renaming and $\beta$-Reduction

**$\alpha$ and $\beta$ Conversions**

- $\alpha$-renaming: $\lambda x \cdot M$
  
  If $x$, $y$ are distinct variables then replacing $x$ by $y$ in $\lambda x \cdot M$ results in $\lambda y \cdot \text{subst}([y/x]M)$. We can rename the formal parameter of a $\lambda$-function expression provided that no free variables of its body $M$ thereby become bound.

- $\beta$-reduction: $(\lambda x \cdot M)(N)$
  
  All free occurrences of $x$ in $M$ are replaced by the expression $N$ provided that no free variables of $N$ thereby become bound in the result. $(\lambda x \cdot M)(N) \equiv \text{subst}([N/x]M)$

**Example 14** .................................................................

**Network Traffic:**

We model traffic by introducing a number of model concepts. We simplify, without loosing the essence of this example, namely to show the use of $\lambda$–functions, by omitting consideration of dynamically changing nets. These are introduced next:

- Let us assume a net, $n:N$.  

• There is a dense set, $T$, of times – for which we omit giving an appropriate definition.

• There is a sort, $V$, of vehicles.

• $TS$ is a dense subset of $T$.

• For each $ts:TS$ we can define a minimum and a maximum time.

• The $MIN$ and $MAX$ functions are meta-linguistic, that is, are not defined in our formal specification language RSL, but can be given a satisfactory meaning.

• At any moment some vehicles, $v:V$, have a $pos:Position$ on the net and $VP$ records those.

• A $Position$ is either on a link or at a hub.

• An $onL$ink position can be designated by the link identifier, the identifiers of the from and to hubs, and the fraction, $f:F$, of the distance down the link from the from hub to the to hub.

• An $atH$ub position just designates the hub (by its identifier).

• Traffic, $tf:TF$, is now a continuous function from $Time$ to $NP$ ("recordings").

• Modelling traffic in this way, in fact, in whichever way, entails a ("serious") number of well-formedness conditions. These are defined in $wf\_TF$ (omitted: ...).

value
  $n:N$

type
  $T, V$
  $TS = T\text{-}\infset$

axiom
  $\forall ts:TS \cdot \exists tmin, tmax: T: tmin \in ts \land tmax \in ts \land \forall t:T \cdot t \in ts \Rightarrow tmin \leq t \leq tmax$
  [that is: $ts = \{MIN(ts) .. MAX(ts)\}]

type
  $VP = V \mid Pos$
  $TF' = T \rightarrow VP,$
  $TF = \{[tf:TF' \cdot wf\_TF(tf)(n)]\}$
  $Pos = onL \mid atH$
  $onL == mkLPos(hi:HI, li:LI, f:F, hi:HI),$  $atH == mkHPos(hi:HI)$

value
  $wf\_TF: TF \rightarrow N \rightarrow Bool$
  $wf\_TF(tf)(n) \equiv ...$
  $DOMAIN: TF \rightarrow TS$
  $MIN', MAX': TS \rightarrow T$
We have defined the continuous, composite entity of traffic. Now let us define an operation of inserting a vehicle in a traffic.

- To insert a vehicle, \( v \), in a traffic, \( tf \), is prescribable as follows:
  - the vehicle, \( v \), must be designated;
  - a time point, \( t \), "inside" the traffic \( tf \) must be stated;
  - a traffic, \( vtf \), from time \( t \) of vehicle \( v \) must be stated;
  - as well as traffic, \( tf \), into which \( vtf \) is to be "merged".

- The resulting traffic is referred to as \( tf' \).

\[
\text{value} \\
\text{insert}_V : V \times T \times TF \rightarrow TF \\
\text{insert}_V(v,t,vtf)(tf) \text{ as } tf'
\]

- The function \( \text{insert}_V \) is here defined in terms of a pair of pre/post conditions.

- The pre-condition can be prescribed as follows:
  - The insertion time \( t \) must be within to open interval of time points in the traffic \( tf \) to which insertion applies.
  - The vehicle \( v \) must not be among the vehicle positions of \( tf \).
  - The vehicle must be the only vehicle "contained" in the "inserted" traffic \( vtf \).

\[
\text{pre: } \begin{align*}
  & \text{MIN}(\text{DOMAIN}(tf)) \leq t \leq \text{MAX}(\text{DOMAIN}(tf)) \land \\
  & \forall t' : T \cdot t' \in \text{DOMAIN}(tf) \Rightarrow v \not\in \text{dom } tf(t') \land \\
  & \text{MIN}(\text{DOMAIN}(vtf)) = t \land \\
  & \forall t' : T \cdot t' \in \text{DOMAIN}(vtf) \Rightarrow \text{dom } vtf(t') = \{v\}
\end{align*}
\]

- The post condition "defines" \( tf' \), the traffic resulting from merging \( vtf \) with \( tf \):
  - Let \( ts \) be the time points of \( tf \) and \( vtf \), a time interval.
  - The result traffic, \( tf' \), is defines as a \( \lambda \)-function.
  - For any \( t'' \) in the time interval
  - if \( t'' \) is less than \( t \), the insertion time, then \( tf' \) is as \( tf \);
  - if \( t'' \) is \( t \) or larger then \( tf' \) applied to \( t'' \), i.e., \( tf'(t'') \)
    - for any \( v' : V \) different from \( v \) yields the same as \( (tf(t))(v') \),
    - but for \( v \) it yields \( (vtf(t))(v) \).
post: \( tf' = \lambda t'' \cdot \)

\[
\begin{align*}
\text{let } ts &= \text{DOMAIN}(tf) \cup \text{DOMAIN}(vtf) \text{ in} \\
\text{if } \text{MIN}(ts) \leq t'' \leq \text{MAX}(ts) \\
\text{then} \\
\quad &((t'' < t) \rightarrow tf(t''), \\
&\quad (t'' \geq t) \rightarrow [v' \mapsto \begin{cases} \\
&\text{if } v' \neq v \text{ then } (tf(t))(v) \text{ else } (vtf(t))(v) \end{cases}]) \\
\quad \text{else chaos end end}
\end{align*}
\]

assumption: \( \text{wf}_{\text{TF}}(vtf) \land \text{wf}_{\text{TF}}(tf) \)

theorem: \( \text{wf}_{\text{TF}}(tf') \)

value

\[
\begin{align*}
\text{vehicles: } TF &\rightarrow V\text{-set} \\
\text{vehicles}(tf) &\equiv \{v|t:T, v:V \cdot t \in \text{DOMAIN}(tf) \land v \in \text{dom } tf(t)\}
\end{align*}
\]

We leave it as an exercise for the reader to define functions for: removing a vehicle from a traffic, changing to course of a vehicle from an originally (or changed) vehicle traffic to another. etcetera.

End of Example 14

A.4.5 **Function Signatures**

For sorts we may want to postulate some functions:

**Sorts and Function Signatures**

\[
\begin{align*}
type &\quad A, B, ..., C \\
\text{value} &\quad \text{obs}_B: A \rightarrow B \\
&\quad ... \\
&\quad \text{obs}_C: A \rightarrow C
\end{align*}
\]

These functions cannot be defined. Once a domain is presented in which sort \( A \) and sorts or types \( B, ..., C \) occurs these observer functions can be demonstrated.

**Example 15**  

**Hub and Link Observers:**

Let a net with several hubs and links be presented. Now observer functions \( \text{obs}_H \)s and \( \text{obs}_L \)s can be demonstrated: one simply “walks” along the net, pointing out this hub and that link, one-by-one until all the net has been visited.

The observer functions \( \text{obs}_H \) and \( \text{obs}_L \) can be likewise demonstrated, for example: when a hub is “visited” its unique identification can be postulated (and “calculated”) to be the unique
geographic position of the hub one which is not overlapped by any other hub (or link), and likewise for links.

End of Example 15

A.4.6 Function Definitions

Functions can be defined explicitly:

\[
\text{type} \quad \begin{align*}
A, B \\
\text{value} \\
f: A \to B \quad \text{[a total function]} \quad g: A \rightsquigarrow B \quad \text{[a partial function]} \\
f(a_{\text{expr}}) \equiv b_{\text{expr}} \quad g(a_{\text{expr}}) \equiv b_{\text{expr}} \\
\text{pre } P(a_{\text{expr}}) \quad \text{pre } P(a_{\text{expr}}) \\
P: A \to \text{Bool} \quad P': A \to \text{Bool}
\end{align*}
\]

\(a_{\text{expr}}, b_{\text{expr}}\) are \(A\), respectively \(B\) valued expressions of any of the kinds illustrated in earlier and later sections of this primer.

Or functions can be defined implicitly:

\[
\text{value} \quad \begin{align*}
f: A \to B \\
f(a_{\text{expr}}) \text{ as } b' \\
\text{post } P(a_{\text{expr}}, b) \\
\text{pre } P'(a_{\text{expr}}) \\
P: A \times B \to \text{Bool} \quad P': A \to \text{Bool}
\end{align*}
\]

where \(b'\) is just an identifier.

The symbol \(\rightsquigarrow\) indicates that the function is partial and thus not defined for all arguments. Partial functions should be assisted by preconditions stating the criteria for arguments to be meaningful to the function.

Finally functions, \(f, g, ..., h\), can be defined in terms of axioms over function identifiers, \(f, g, ..., h\), and over identifiers of function arguments and results.

\[
\text{type} \quad \begin{align*}
A, B, ..., C, D \\
\text{value} \\
f: A \to B, g: B \to C, ..., h: C \to D
\end{align*}
\]

\[
\text{axiom} \quad \begin{align*}
\forall a:A, b:B, ..., c:C, d:D \\
P_1(f, g, ..., h, a, b, ..., c, d) \land ... \land P_n(f, g, ..., h, a, b, ..., c, d)
\end{align*}
\]

\(P_1, \ldots, P_m\) and \(Q_1, \ldots, Q_n\) designate suitable predicate expressions.
Example 16 .......................... Axioms over Hubs, Links and Their Observers:
The axioms displayed in Items 2–3 and 5–8 on Page 13 of Sect. 2.1 demonstrates how a number of entities and observer functions are constrained (that is, partially defined) by function signatures.

End of Example 16

A.5 Other Applicative Expressions

A.5.1 Simple let Expressions

Simple (i.e., nonrecursive) let expressions:

Let Expressions

\[ \text{let } a = E_d \text{ in } E_b(a) \text{ end} \]

is an “expanded” form of:

\[ (\lambda a. E_b(a))(E_d) \]

A.5.2 Recursive let Expressions

Recursive let expressions are written as:

Recursive let Expressions

\[ \text{let } f = \lambda a \cdot E(f,a) \text{ in } B(f,a) \text{ end} \]
\[ \text{let } f = (\lambda g \cdot \lambda a \cdot E(g,a))(f) \text{ in } B(f,a) \text{ end} \]
\[ \text{let } f = F(f) \text{ in } E(f,a) \text{ end where } F \equiv \lambda g \cdot \lambda a \cdot E(g,a) \]
\[ \text{let } f = YF \text{ in } B(f,a) \text{ end where } YF = F(YF) \]

We read \( f = YF \) as “\( f \) is a fix point of \( F \)”.

A.5.3 Non-deterministic let Clause

The non-deterministic let clause:

\[ \text{let } a : A \cdot P(a) \text{ in } B(a) \text{ end} \]

expresses the non-deterministic selection of a value \( a \) of type \( A \) which satisfies a predicate \( P(a) \) for evaluation in the body \( B(a) \). If no \( a : A \cdot P(a) \) the clause evaluates to \text{chaos}.
A.5.4 Pattern and "Wild Card" let Expressions

Patterns and wild cards can be used:

Patterns

let \{a\} \cup s = \text{set} ... end
let \{a,\_\} \cup s = \text{set} ... end

let (a,b,\ldots,c) = \text{cart} in ... end
let (a,\_,\ldots,c) = \text{cart} in ... end

let \langle a \rangle \hat{\ell} = \text{list} in ... end
let \langle a,\_b \rangle \hat{\ell} = \text{list} in ... end

let \lfloor a \mapsto b \rfloor \cup m = \text{map} in ... end
let \lfloor a \mapsto \_ \rfloor \cup m = \text{map} in ... end

A.5.5 Conditionals

Various kinds of conditional expressions are offered by RSL:

Conditionals

if b_{\text{expr}} \text{ then } c_{\text{expr}} \text{ else } a_{\text{expr}} \text{ end}

if b_{\text{expr}} \text{ then } c_{\text{expr}} \text{ end} \equiv /\ast \text{ same as: } /\ast/
    \text{ if } b_{\text{expr}} \text{ then } c_{\text{expr}} \text{ else skip end}

if b_{\text{expr}}_{1} \text{ then } c_{\text{expr}}_{1}
\text{ elsif } b_{\text{expr}}_{2} \text{ then } c_{\text{expr}}_{2}
\text{ elsif } b_{\text{expr}}_{3} \text{ then } c_{\text{expr}}_{3}
\ldots
\text{ elsif } b_{\text{expr}}_{n} \text{ then } c_{\text{expr}}_{n} \text{ end}

case expr of
    \text{choice}_{\text{pattern}}_{1} \rightarrow \text{expr}_{1},
    \text{choice}_{\text{pattern}}_{2} \rightarrow \text{expr}_{2},
    \ldots
    \text{choice}_{\text{pattern}}_{n} \text{ or } \text{wild card} \rightarrow \text{expr}_{n} \text{ end}
Example 17 .......................... Choice Pattern Case Expressions: Insert Links:
We consider the meaning of the Insert operation designators.

33. The insert operation takes an Insert command and a net and yields either a new net or chaos for the case where the insertion command “is at odds” with, that is, is not semantically well-formed with respect to the net.

34. We characterise the “is not at odds”, i.e., is semantically well-formed, that is:

- pre_int_Insert(op)(hs,ls),

as follows: it is a propositional function which applies to Insert actions, op, and nets, (hs,ls), and yields a truth value if the below relation between the command arguments and the net is satisfied. Let (hs,ls) be a value of type N.

35. If the command is of the form 2oldH(hi’,l,hi”) then

- *1 hi’ must be the identifier of a hub in hs,
- *2 l must not be in ls and its identifier must (also) not be observable in ls, and
- *3 hi” must be the identifier of a(nother) hub in hs.

36. If the command is of the form 1oldH1newH(hi,l,h) then

- *1 hi must be the identifier of a hub in hs,
- *2 l must not be in ls and its identifier must (also) not be observable in ls, and
- *3 h must not be in hs and its identifier must (also) not be observable in hs.

37. If the command is of the form 2newH(h’,l,h”’) then

- *1 h’ — left to the reader as an exercise (see formalisation !),
- *2 l — left to the reader as an exercise (see formalisation !), and
- *3 h”’ — left to the reader as an exercise (see formalisation !).

Conditions concerning the new link (second *s, *2, in the above three cases) can be expressed independent of the insert command category.

value

33 int_Insert: Insert → N ~ N
34’ pre_int_Insert: Ins → N → Bool
34” pre_int_Insert(Ins(op))(hs,ls) ≡
   \( s\_l(op) \notin ls \land \text{obs}_L(s\_l(op)) \notin \text{iols}(ls) \land \text{case op of} \)
35) 2oldH(hi’,l,hi”’) → \{hi’,hi”’\} ∈ \text{iohs}(hs),
36) \[ 1 \text{oldH1newH}(hi,l,h) \rightarrow \]
\[ hi \in \text{iohs}(hs) \land h \notin hs \land \text{obs_Hl}(h) \notin \text{iohs}(hs), \]

37) \[ 2 \text{newH}(h',l,h'') \rightarrow \]
\[ \{h',h''\} \cap hs = \{\} \land \{\text{obs_Hl}(h'),\text{obs_Hl}(h'')\} \cap \text{iohs}(hs) = \{\} \]

end

RSL Explanation

• 33: The value clause value \text{int}\text{Insert}: Insert \rightarrow N \rightsquigarrow N names a value, \text{int}\text{Insert}, and defines its type to be Insert \rightarrow N \rightsquigarrow N, that is, a partial function (\rightsquigarrow) from Insert commands and nets (N) to nets. (\text{int}\text{Insert} is thus a function. What that function calculates will be defined later.)

• 34': The predicate pre\text{int}\text{Insert}: Insert \rightarrow N \rightarrow \text{Bool} function (which is used in connection with \text{int}\text{Insert} to assert semantic well-formedness) applies to Insert commands and nets and yield truth value true if the command can be meaningfully performed on the net state.

• 34'': The action pre\text{int}\text{Insert}(op)(hs,ls) (that is, the effect of performing the function pre\text{int}\text{Insert} on an Insert command and a net state is defined by a case distinction over the category of the Insert command. But first we test the common property:

• \star2: s_L(op)\notin ls \land \text{obs_Ll}(s_L(op)) \notin \text{iols}(ls), namely that the new link is not an existing net link and that its identifier is not already known.

  – 35): If the Insert command is of kind \text{2oldH}(hi',l,hi'') then \{hi',hi''\} \subseteq \text{iohs}(hs), that is, then the two distinct argument hub identifiers must not be in the set of known hub identifiers, i.e., of the existing hubs hs.

  – 36): If the Insert command is of kind \text{1oldH1newH}(hi,l,h) then ... exercise left as an exercises to the reader.

  – 37): If the Insert command is of kind \text{2newH}(h',l,h'') ... exercise left as an exercises to the reader. The set intersection operation is defined in Sect. A.2.6 on page 111 Item 25 on page 112.

End of RSL Explanation

38. Given a net, (hs,ls), and given a hub identifier, (hi), which can be observed from some hub in the net, xtr\_H(hi)(hs,ls) extracts the hub with that identifier.

39. Given a net, (hs,ls), and given a link identifier, (li), which can be observed from some link in the net, xtr\_L(li)(hs,ls) extracts the hub with that identifier.
value
38: \text{xtr}\_H: H \rightarrow N \xrightarrow{\_} H
38: \text{xtr}\_H(hi)(hs,\_)= \text{let}\ h: H \in hs \land \text{obs}\_H(h)=hi \text{ in } h \text{ end}
\text{pre } hi \in iohs(hs)
39: \text{xtr}\_L: H \rightarrow N \xrightarrow{\_} H
39: \text{xtr}\_L(li)(\_,ls)= \text{let}\ l: L \in ls \land \text{obs}\_L(l)=li \text{ in } l \text{ end}
\text{pre } li \in iols(ls)

RSL Explanation

• 38: Function application \text{xtr}\_H(hi)(hs,\_) yields the hub \(h\), i.e. the value \(h\) of type \(H\), such that (\_\_\_) \(h\) is in \(hs\) and \(h\) has hub identifier \(hi\).

• 38: The wild-card, \_\_ expresses that the extraction (\text{xtr}\_H) function does not need the \(L\)-set argument.

• 39: Left as an exercise for the reader.

End of RSL Explanation

40. When a new link is joined to an existing hub then the observable link identifiers of that hub must be updated to reflect the link identifier of the new link.

41. When an existing link is removed from a remaining hub then the observable link identifiers of that hub must be updated to reflect the removed link (identifier).

value
\text{aLI}: H \times L \rightarrow H, \text{rLI}: H \times L \xrightarrow{\_} H
40: \text{aLI}(h,li) \text{ as } h'
\text{pre } li \notin \text{obs}\_Lls(h)
\text{post } \text{obs}\_Lls(h') = \{li\} \cup \text{obs}\_Lls(h) \land \text{non}\_I\_eq(h,h')
41: \text{rLI}(h',li) \text{ as } h
\text{pre } li \in \text{obs}\_Lls(h') \land \text{card } \text{obs}\_Lls(h')\geq 2
\text{post } \text{obs}\_Lls(h) = \text{obs}\_Lls(h') \setminus \{li\} \land \text{non}\_I\_eq(h,h')

RSL Explanation

• 40: The add link identifier function \text{aLI}:
  - The function definition clause \text{aLI}(h,li) \text{ as } h' defines the application of \text{aLI} to a pair (h,li) to yield an update, h' of h.
  - The pre-condition \text{pre } li \notin \text{obs}\_Lls(h) expresses that the link identifier li must not be observable h.
The post-condition \( \text{post obs}_LIs(h) = \text{obs}_LIs(h') \setminus \{li\} \land \text{non}_L\text{eq}(h,h') \) expresses that the link identifiers of the resulting hub are those of the argument hub except \( \setminus \) that the argument link identifier is not in the resulting hub.

- 41: The remove link identifier function \( rLI \):

- The function definition clause \( rLI(h',li) \) as \( h \) defines the application of \( rLI \) to a pair \( (h',li) \) to yield an update, \( h' \).
- The pre-condition clause \( \text{pre li} \in \text{obs}_LIs(h') \land \text{card obs}_LIs(h') \geq 2 \) expresses that the link identifier \( li \) must not be observable \( h \).
- Post-condition clause \( \text{post obs}_LIs(h) = \text{obs}_LIs(h') \setminus \{li\} \land \text{non}_L\text{eq}(h,h') \) expresses that the link identifiers of the resulting hub are those of the argument hub except that the argument link identifier is not in the resulting hub.

End of RSL Explanation

42. If the Insert command is of kind \( 2\text{newH}(h',l,h'') \) then the updated net of hubs and links, has

- the hubs \( hs \) joined, \( \bigcup \), by the set \( \{h',h''\} \) and
- the links \( ls \) joined by the singleton set of \( \{l\} \).

43. If the Insert command is of kind \( 1\text{oldH1newH}(hi,l,h) \) then the updated net of hubs and links, has

43.1 : the hub identified by \( hi \) updated, \( hi' \), to reflect the link connected to that hub.

43.2 : The set of hubs has the hub identified by \( hi \) replaced by the updated hub \( hi' \) and the new hub.

43.2 : The set of links augmented by the new link.

44. If the Insert command is of kind \( 2\text{oldH}(hi',l,hi'') \) then

44.1–.2 : the two connecting hubs are updated to reflect the new link.

44.3 : and the resulting sets of hubs and links updated.

\[
\text{int_Insert(op)}(hs,ls) \equiv
\]
\[
\begin{cases}
2\text{newH}(h',l,h'') & \rightarrow (hs \cup \{h',h''\},ls \cup \{l\}).
\end{cases}
\]
\[
\begin{cases}
1\text{oldH1newH}(hi,l,h) & \rightarrow
\end{cases}
\]
\[
\begin{cases}
43.1 & \text{let } h' = \text{aLI}(\text{xtr}_H(hi,hs),\text{obs}_L\text{L}(l)) \text{ in }
\end{cases}
\]
\[
\begin{cases}
43.2 & (hs \setminus \{\text{xtr}_H(hi,hs)\} \cup \{h',l\},ls \cup \{l\}) \text{ end,}
\end{cases}
\]
\[
\begin{cases}
44 & 2\text{oldH}(hi',l,hi'') & \rightarrow
\end{cases}
\]
44.1 let \( \text{hs}\delta = \{\text{aLI}(\text{xtr}_H(\text{hi}',\text{hs}),\text{obs}_L(l)), \text{aLI}(\text{xtr}_H(\text{hi}'',\text{hs}),\text{obs}_L(l)))\} \) in
44.2 (\( \text{hs}\setminus\{\text{xtr}_H(\text{hi}',\text{hs}),\text{xtr}_H(\text{hi}'',\text{hs})\} \cup \text{hs}\delta,\text{ls} \cup \{l\} \)) end
44.3 \( \star_j \) end
44.4 \( \star_k \) pre \text{pre}_\text{int}\text{Insert}(\text{op})(\text{hs,ls})

RSL Explanation

- \( \star_{i-j} \): The clause case \( \text{op} \) of \( \text{p}_1 \rightarrow \text{c}_1, \text{p}_2 \rightarrow \text{c}_2, \ldots \text{p}_n \rightarrow \text{c}_n \) end is a conditional clause.
- \( \star_k \): The pre-condition expresses that the insert command is semantically well-formed — which means that those reference identifiers that are used are known and that the new link and hubs are not known in the net.
- \( \star_i + 42 \): If \( \text{op} \) is of the form 2newH(\( \text{hi}'',\text{l},\text{hi}'' \)) then — the narrative explains the rest;
  - else
    - \( \star_i + 43 \): If \( \text{op} \) is of the form 1oldH1newH(\( \text{hi},\text{l},\text{h} \)) then
      - 43.1: \( \text{hi}' \) is the known hub (identified by \( \text{hi} \)) updated to reflect the new link being connected to that hub,
      - 43.2: and the pair \( [\{\text{updated hs,updated ls}\}] \) reflects the new net: the hubs have the hub originally known by \( \text{hi} \) replaced by \( \text{hi}' \), and the links have been simple extended \((\cup)\) by the singleton set of the new link;
  - else
    - \( \star_i + 44 \): If \( \text{op} \) is of the form 2oldH(\( \text{hi}',\text{l},\text{hi}'' \)) then
      - 44.1: the first element of the set of two hubs \( (\text{hs}\delta) \) reflect one of the updated hubs,
      - 44.2: the second element of the set of two hubs \( (\text{hs}\delta) \) reflect the other of the updated hubs,
      - 44.3: the set of two original hubs known by the argument hub identifiers are removed and replaced by the set \( \text{hs}\delta \);
  - else — well, there is no need for a further ‘else’ part as the operator can only be of either of the three mutually exclusive forms !

End of RSL Explanation

45. The remove command is of the form Rmv(\( \text{li} \)) for some \( \text{li} \).

46. We now sketch the meaning of removing a link:
a) The link identifier, li, is, by the \texttt{pre}\texttt{int\_Remove} pre-condition, that of a link, l, in the net.

b) That link connects to two hubs, let us refer to them as $h'$ and $h''$.

c) For each of these two hubs, say h, the following holds wrt. removal of their connecting link:
   
i. If l is the only link connected to h then hub h is removed. This may mean that
      • either one
      • or two hubs
   
   are also removed when the link is removed.
   
   ii. If l is not the only link connected to h then the hub h is modified to reflect that
      it is no longer connected to l.

d) The resulting net is that of the pair of adjusted set of hubs and links.

\begin{align*}
\text{\texttt{int\_Remove}} &: \text{Rmv} \to N \xrightarrow{\sim} N \\
\text{\texttt{int\_Remove(Rmv(li))(hs,ls)}} &: \equiv \\
46a) &: \text{let } l = \text{extr}(\text{li})(\text{ls}), \{h',h''\} = \text{obs\_Hls(l)} \text{ in} \\
46b) &: \text{let } \{h',h''\} = \{\text{extr\_H(h',hs)},\text{extr\_H(h'',hs)}\} \text{ in} \\
46c) &: \text{let } hs' = \text{cond\_rmv}(h',hs) \cup \text{cond\_rmv}(h'',hs) \text{ in} \\
46d) &: (hs\{\{h',h''\} \cup hs',ls\{\{l\}\}\} \text{ end end end} \\
46a) &: \text{pre } li \in \text{iols(ls)}
\end{align*}

\begin{align*}
\text{\texttt{cond\_rmv}} &: \text{LI} \times H \times H\text{-set} \to H\text{-set} \\
\text{\texttt{cond\_rmv}(li,h,hs)} &: \equiv \text{cond\_rmv}\texttt{Rmv}(li,h,hs) \\
46(c)i) &: \text{if } \text{obs\_Hls}(h) \{\{li\}\} \text{ then } \{\} \\
46(c)ii) &: \text{else } \{\text{sL}(\{li,h\})\} \text{ end} \\
\text{\texttt{pre } li \in \text{obs\_Hls}(h)}
\end{align*}

\textbf{RSL Explanation}

\begin{itemize}
\item 45: The \texttt{int\_Remove} operation applies to a remove command \texttt{Rmv(li)} and a net (hs,ls) and yields a net — provided the remove command is semantically well-formed.
\item 46: To Remove a link identifier by li from the net (hs,ls) can be formalised as follows:
   
   - 46a): obtain the link l from its identifier li and the set of links ls, and
   
   - 46a): obtain the identifiers, \{hi',hi''\}, of the two distinct hubs to which link l is connected;
   
   - 46b): then obtain the hubs \{h',h''\} with these identifiers;
   
   - 46c): now examine \texttt{cond\_rmv} each of these hubs (see Lines 46(c)i)–46(c)ii)
\end{itemize}
The examination function cond\_rmv either yields an empty set or the singleton set of one modified hub (a link identifier has been removed).

- 46c) The set, hs', of zero, one or two modified hubs is yielded.
- That set is joined to the result of removing the hubs \{h', h''\}
- and the set of links that result from removing l from ls.

The conditional hub remove function cond\_rmv
- 46(c)i): either yields the empty set (of no hubs) if li is the only link identifier inh,
- 46(c)ii): or yields a modification of h in which the link identifier li is no longer observable.

End of RSL Explanation

End of Example 17

A.5.6 Operator/Operand Expressions

Operator/Operand Expressions

\langle Expr \rangle ::= 
\langle Prefix\_Op \rangle \langle Expr \rangle 
| \langle Expr \rangle \langle Infix\_Op \rangle \langle Expr \rangle 
| \langle Expr \rangle \langle Suffix\_Op \rangle 
| ...

\langle Prefix\_Op \rangle ::= 
- \sim | \cup | \cap | card | len | inds | elems | hd | tl | dom | rng

\langle Infix\_Op \rangle ::= 
= | \neq | \equiv | + | - | \ast | \uparrow | / | \leq | \leq | \geq | \geq | \land | \lor | \Rightarrow 
| \in | \notin | \cup | \cap | \setminus | \subseteq | \subseteq | \supseteq | \supseteq | \hat{\bullet} \circ

\langle Suffix\_Op \rangle ::= !

A.6 Imperative Constructs

A.6.1 Statements and State Changes

Often, following the RAISE method, software development starts with highly abstract, sorts and applicative constructs which, through stages of refinements, are turned into concrete types and imperative constructs.

Imperative constructs are thus inevitable in RSL.

Unit value

\begin{verbatim}
stmt: Unit → Unit
stmt()
\end{verbatim}
• The Unit clause, in a sense, denotes “an underlying state”
  – which we, for simplicity, can consider as
  – a mapping from identifiers of declared variables into their values.

• Statements accept no arguments and, usually, operate on the state
  – through “reading” the value(s) of declared variables and
  – through “writing”, i.e., assigning values to such declared variables.

• Statement execution thus changes the state (of declared variables).

• Unit → Unit designates a function from states to states.

• Statements, stmt, denote state-to-state changing functions.

• Affixing () as an “only” arguments to a function “means” that () is an argument of
type Unit.

A.6.2 Variables and Assignment

Variables and Assignment

0. variable v:Type := expression
1. v := expr

A.6.3 Statement Sequences and skip

Sequencing is expressed using the ‘;’ operator. skip is the empty statement having no
value or side-effect.

2. skip
3. stm_1;stm_2;...;stm_n

A.6.4 Imperative Conditionals

4. if expr then stm_c else stm_a end
5. case e of: p_1→S_1(p_1),...,p_n→S_n(p_n) end
A.6.5 Iterative Conditionals

6. \textbf{while} expr \textbf{do} stm \textbf{end}
7. \textbf{do} stmt \textbf{until} expr \textbf{end}

A.6.6 Iterative Sequencing

8. \textbf{for} i \textbf{in} list \cdot P(list(i)) \textbf{do} S(list(i)) \textbf{end}
9. \textbf{for} e \textbf{in} set \cdot P(e) \textbf{do} S(e) \textbf{end}

A.7 Process Constructs

A.7.1 Process Channels

Let A, B and C stand for three types of (channel) messages and i:IIdx, j:IIdx for channel array indexes, then:

\begin{description}
\item[Process Channels]
\begin{verbatim}
channel c:A
channel {k[i]:IIdx}:B
\{ch[i,j]:IIdx,j:IIidx}\}:C
\end{verbatim}
\end{description}

declare a channel, c, and a set (an array) of channels, k[i], capable of communicating values of the designated types (A and B).

Example 18 Modelling Connected Links and Hubs:

Examples (18–21) of this section, i.e., Sect. A.7 are building up a model of one form of meaning of a transport net. We model the movement of vehicles around hubs and links. We think of each hub, each link and each vehicle to be a process. These processes communicate via channels.

\begin{itemize}
\item We assume a net, $n : N$, and a set, vs, of vehicles.
\item Each vehicle can potentially interact
  \begin{itemize}
  \item with each hub and
  \item with each link.
  \end{itemize}
\item Array channel indices $(vi,hi):IVH$ and $(vi,li):IVL$ serve to effect these interactions.
\item Each hub can interact with each of its connected links and indices $(hi,li):IHL$ serves these interactions.
\end{itemize}
An RSL Primer

137

\textbf{value}

\begin{align*}
n \colon & N, \quad vs \colon V \text{-set} \\
\text{obs}_V & \colon V \rightarrow VI \\
\text{type} & = H, L, HI, LI, M \\
IVH & = VI \times HI, \quad IVL = VI \times LI, \quad IHL = HI \times LI
\end{align*}

- We need some auxiliary quantities in order to be able to express subsequent channel declarations.

- Given that we assume a net, \( n : N \) and a set of vehicles, \( vs : V S \), we can now define the following (global) values:
  - the sets of hubs, \( hs \), and links, \( ls \) of the net;
  - the set, \( ivhs \), of indices between vehicles and hubs,
  - the set, \( ivls \), of indices between vehicles and links, and
  - the set, \( ihls \), of indices between hubs and links.

\textbf{value}

\begin{align*}
hs & \colon H \text{-set} = \text{obs}_H(s(n), \quad ls \colon L \text{-set} = \text{obs}_L(s(n)) \\
his & \colon HI \text{-set} = \{ \text{obs}_H(h) \mid h : H \in hs \}, \quad lis \colon LI \text{-set} = \{ \text{obs}_L(l) \mid l : L \in ls \}, \\
ivhs & \colon IVH \text{-set} = \{ (\text{obs}_V(v), \text{obs}_H(h)) \mid v : V \in vs \land h : H \in hs \}, \\
ivls & \colon IVL \text{-set} = \{ (\text{obs}_V(v), \text{obs}_L(l)) \mid v : V \in vs \land l : L \in ls \}, \\
ihls & \colon IHL \text{-set} = \{ (hi, li) \mid h : H \in hs \land hi = \text{obs}_H(h) \land li = \text{obs}_L(l) \}
\end{align*}

- We are now ready to declare the channels:
  - a set of channels, \( \{ vh[i] \mid i : IVH \cdot i \in ivhs \} \) between vehicles and all potentially traversable hubs;
  - a set of channels, \( \{ vh[i] \mid i : IVH \cdot i \in ivhs \} \) between vehicles and all potentially traversable links; and
  - a set of channels, \( \{ hl[i] \mid i : IHL \cdot i \in ihls \} \), between hubs and connected links.

\textbf{channel}

\begin{align*}
\{ vh[i] \mid i : IVH \cdot i \in ivhs \} & : M \\
\{ vl[i] \mid i : IVL \cdot i \in ivls \} & : M \\
\{ hl[i] \mid i : IHL \cdot i \in ihls \} & : M
\end{align*}

End of Example 18
A.7.2 Process Definitions

A process definition is a function definition. The below signatures are just examples. They emphasise that process functions must somehow express, in their signature, via which channels they wish to engage in input and output events.

Processes $P$ and $Q$ are to interact, and to do so “ad infinitum”. Processes $R$ and $S$ are to interact, and to do so “once”, and then yielding $B$, respectively $D$ values.

\begin{align*}
&\text{value} \\
&P: \text{Unit} \rightarrow \text{in c out } \{k[i] | i: \text{Int}\} \text{ Unit} \\
&Q: i: \text{Int} \rightarrow \text{out c in } k[i] \text{ Unit} \\
&P() \equiv \ldots \text{c ? } \ldots \text{k[i] ! e } \ldots ; P() \\
&Q(i) \equiv \ldots \text{c ! e } \ldots \text{k[i] ? } \ldots ; Q(i)
\end{align*}

\[\text{Figure 9: The } P \longrightarrow Q \text{ Process}\]

\textbf{Example 19} Communicating Hubs, Links and Vehicles:

- Hubs interact with links and vehicles:
  - with all immediately adjacent links,
  - and with potentially all vehicles.
- Links interact with hubs and vehicles:
  - with both adjacent hubs,
  - and with potentially all vehicles.
Vehicles interact with hubs and links:
  – with potentially all hubs.
  – and with potentially all links.

\[
\text{value}
\]

\[
\begin{align*}
\text{hub: } & hi:HI \times h:H \rightarrow \text{in, out } \{ hl[(hi,li)\mid li:LI \in \text{obs}_LIs(h)] \} \\
\text{link: } & li:LI \times l:L \rightarrow \text{in, out } \{ hl[(hi,li)\mid hi:HI \cdot hi \in \text{obs}_HIs(l)] \} \\
\text{vehicle: } & vi:VI \rightarrow (\text{Pos} \times \text{Net}) \rightarrow v:V \rightarrow \text{in, out } \{ vh[(vi,hi)\mid hi:HI \cdot hi \in \text{his}] \} \\
\end{align*}
\]

\text{Unit}

\text{End of Example 19}

\section*{A.7.3 Process Composition}

Let \( P \) and \( Q \) stand for names of process functions, i.e., of functions which express willingness to engage in input and/or output events, thereby communicating over declared channels. Let \( P \) and \( Q \) stand for process expressions, and let \( P_i \) stand for an indexed process expression, then:

\[
\begin{align*}
\mathcal{P} \parallel \mathcal{Q} & \quad \text{Parallel composition} \\
\mathcal{P} \mid \mathcal{Q} & \quad \text{Nondeterministic external choice (either/or)} \\
\mathcal{P} \sqcap \mathcal{Q} & \quad \text{Nondeterministic internal choice (either/or)} \\
\mathcal{P} \parallel \mathcal{Q} & \quad \text{Interlock parallel composition} \\
\mathcal{O} \{ P_i \mid i:Idx \} & \quad \text{Distributed composition, } \mathcal{O} = ||, [, ], \parallel
\end{align*}
\]

express the parallel (\( || \)) of two processes, or the nondeterministic choice between two processes: either external (\( [] \)) or internal (\( [ ] \)). The interlock (\( \parallel \)) composition expresses that the two processes are forced to communicate only with one another, until one of them terminates.

\text{Example 20} \quad \text{Modelling Transport Nets:}

\begin{itemize}
  \item The net, with vehicles, potential or actual, is now considered a process.
  \item It is the parallel composition of
    \begin{itemize}
      \item all hub processes,
      \item all link processes and
      \item all vehicle processes.
    \end{itemize}
\end{itemize}
value

\[
\text{net: } N \rightarrow V\text{-set } \rightarrow \text{Unit}
\]

\[
\text{net}(n)(vs) \equiv \left\| \text{hub}(\text{obs}_H(h))(h)|H \cdot h \in \text{obs}_H(n)\right\| \\
\left\| \text{link}(\text{obs}_L(l))(l)|L \cdot l \in \text{obs}_L(n)\right\| \\
\left\| \text{vehicle}(\text{obs}_V(v))|\text{obs}_P(v)(v)|V \cdot v \in vs\right\|
\]

\[
\text{obs}_P: V \rightarrow (\text{Pos} \times \text{Net})
\]

- We illustrate a schematic definition of simplified hub processes.
- The hub process alternates, internally non-deterministically, \(\left\lfloor \right\rfloor\), between three sub-processes
  - a sub-process which serves the link-hub connections,
  - a sub-process which serves those vehicles which communicate that they somehow wish to enter or leave (or do something else with respect to) the hub, and
  - a sub-process which serves the hub itself — whatever that is!

\[
\text{hub}(hi)(h) \equiv \\
\left\lfloor \right\rfloor \left\{ \text{let } m = h[hi, li] ? \text{in} \text{hub}(hi)(E_{hi}(li)(m))(h) \text{ end} | i : L \cdot li \in \text{obs}_L(h) \right\| \\
\left\lfloor \right\rfloor \left\{ \text{let } m = v[hi, vi] ? \text{in} \text{hub}(vi)(E_{hi}(vi)(m))(h) \text{ end} | vi : V \cdot vi \in \text{vis} \right\| \\
\left\lfloor \right\rfloor \text{hub}(hi)(E_{hi}(hi)(h))
\]

- The three auxiliary processes:
  - \(E_{hi}\) update the hub with respect to (wrt.) connected link, \(li\), information \(m\),
  - \(E_{hv}\) update the hub with wrt. vehicle, \(vi\), information \(m\),
  - \(E_{hown}\) update the hub with wrt. whatever the hub so decides. An example could be signalling dependent on previous link-to-hub communicated information, say about traffic density.

\[
E_{hi}: \text{LI} \rightarrow M \rightarrow H \rightarrow H \\
E_{hv}: \text{VI} \rightarrow M \rightarrow H \rightarrow H \\
E_{hown}: H \rightarrow H
\]

The reader is encouraged to sketch/define similarly schematic link and vehicle processes.

\text{End of Example 20}
A.7.4 Input/Output Events

Let c and k[i] designate channels of type A and e expression values of type A, then:

1. c?, k[i]?
2. c!e, k[i]!e

value
3. P: ... → out c ..., P(...) ≡ ... c!e ... offer an A value,
4. Q: ... → in c ..., Q(...) ≡ ... c? ... accept an A value
5. S: ... → ..., S(...) = P(...) || Q(...) synchronise and communicate

[5] expresses the willingness of a process to engage in an event that [1,3] “reads” an input, respectively [2,4] “writes” an output. If process P reaches the c!e “program point before” process Q ‘reaches program point’ c? then process P “waits” on Q — and vice versa. Once both processes have reached these respective program points they “synchronise while communicating the message value e.

The process function definitions (i.e., their bodies) express possible [output/input] events.

Example 21 Modelling Vehicle Movements:

- Whereas hubs and links are modelled as basically static, passive, that is, inert, processes we shall consider vehicles to be “highly” dynamic, active processes.
- We assume that a vehicle possesses knowledge about the road net.
  - The road net is here abstracted as an awareness of
    - which links, by their link identifiers,
    - are connected to any given hub, designated by its hub identifier,
    - the length of the link,
    - and the hub to which the link is connected “at the other end”, also by its hub identifier
- A vehicle is further modelled by its current position on the net in terms of either hub or link positions
  - designated by appropriate identifiers
  - and, when “on a link” “how far down the link”, by a measure of a fraction of the total length of the link, the vehicle has progressed.

**type**

Net = HI m> (LI m> HI)
Pos = atH | onL
atH == mk_atH(hi:HI)
onL == mk_onL(fhi:HI,li:LI,f:F,thi:HI)
F = \{f:Real\ 0 ≤ f ≤ 1\}
• We first assume that the vehicle is at a hub.
• There are now two possibilities (1–2] versus [4–8]).
  – Either the vehicle remains at that hub
    * [1] which is expressed by some non-deterministic wait
    * [2] followed by a resumption of being that vehicle at that location.
  – [3] Or the vehicle (driver) decides to “move on”:
    * [5] Onto a link, \( l_i \),
    * [4] among the links, \( l_i s \), emanating from the hub,
    * [6] and towards a next hub, \( h'_i \).
  – [4,6] The \( l_i s \) and \( h'_i \) quantities are obtained from the vehicle’s own knowledge of the net.
  – [7] The hub and the chosen link are notified by the vehicle of its leaving the hub and entering the link,
  – [8] whereupon the vehicle resumes its being a vehicle at the initial location on the chosen link.

• The vehicle chooses between these two possibilities by an internal non-deterministic choice ([3]).

\[
\text{type} \quad M \triangleq mk\_L\_H(l_i:L_1,h_i:H_1) \mid mk\_H\_L(h_i:H_1,l_i:L_1)
\]

\[
\text{value} \quad \text{vehicle} : \forall l \rightarrow (\text{Pos} \times \text{Net}) \rightarrow V \rightarrow \text{Unit}
\]

\[
\begin{align*}
\text{vehicle}(v)(mk\_atH(h_i),\text{net})(v) & \equiv \\
[1] & (\text{wait} ;) \\
[2] & \text{vehicle}(v)(mk\_atH(h_i),\text{net})(v)) \\
[3] & | \\
[4] & (\text{let lis} = \text{dom net}(h_i) \text{ in} ) \\
[5] & \text{let } l_i : L_1 l_i \in lis \text{ in} \\
[6] & \text{let } h'_i = (\text{net}(h_i))(l_i) \text{ in} \\
[7] & (vh[(v,hi)] ! mk\_H\_L(h_i,l_i)||vl[(v,li)] ! mk\_H\_L(h_i,l_i)); \\
[8] & \text{vehicle}(v)(mk\_onL(h_i,l_i,0,h'_i),\text{net})(v) \\
[9] & \text{end end end})
\end{align*}
\]

• We then assume that the vehicle is on a link and at a certain distance “down”, \( f \), that link.
There are now two possibilities ([1–2] versus [4–7]).

- Either the vehicle remains at that hub
  - [1'] which is expressed by some non-deterministic wait
  - [2'] followed by a resumption of being that vehicle at that location.
- [3'] Or the vehicle (driver) decides to “move on”.
- [4'] Either
  - [5'] The vehicle is at the very end of the link and signals the link and the hub of its leaving the link and entering the hub,
  - [6'] whereupon the vehicle resumes its being a vehicle at hub \( h' \).
- [7'] or the vehicle moves further down, some non-zero fraction down the link.

The vehicle chooses between these two possibilities by an internal non-deterministic choice ([3]).

type
\[ M == mk_{L \to H}(li:L,l;hi:H) \mid mk_{H \to L}(hi:H,li:L) \]

value
\[ \delta: \text{Real} = \text{move}(h,f) \]
axiom \[ 0 < \delta \ll 1 \]
vehicle(\( vi \))( mk_{onL}(hi,li,f,hi'),net)(v) \equiv
\[ [1'] (\text{wait} ; \]
\[ [2'] \text{vehicle}(vi)(mk_{onL}(hi,li,f,hi'),net)(v)) \]
\[ [3'] \]
\[ [4'] (\text{case } f \text{ of} \]
\[ [5'] 1 \to ((vl[ vi,hi']!mk_{L \to H}(li,hi')|vh[ vi,li ]!mk_{L \to H}(li,hi')); \]
\[ [6'] \text{vehicle}(vi)(mk_{atH}(hi'),net)(v)), \]
\[ [7'] 0 \to \text{vehicle}(vi)(mk_{onL}(hi,li,f+\delta,hi'),net)(v) \]
\[ [8'] \text{end} \]
move: \( H \times F \to F \)

End of Example 21

A.8 Simple RSL Specifications

Besides the above constructs RSL also possesses module-oriented scheme, class and object constructs. We shall not cover these here. An RSL specification is then simply a sequence of one or more clusters of zero, one or more sort and/or type definitions, zero, one or more variable declarations, zero, one or more channel declarations, zero, one or more value definitions (including functions) and zero, one or more and axioms. We can illustrate these specification components schematically:

Simple RSL Specifications
The ordering of these clauses is immaterial. Intuitively the meaning of these definitions and declarations are the following.

The **type** clause introduces a number of user-defined type names; the type names are visible anywhere in the specification; and either denote sorts or concrete types.

The **variable** clause declares some variable names; a variable name denote some value of declared type; the variable names are visible anywhere in the specification: assigned to (‘written’) or values ‘read’.

The **channel** clause declares some channel names; either simple channels or arrays of channels of some type; the channel names are visible anywhere in the specification.

The **value** clause bind (constant) values to value names. These value names are visible anywhere in the specification. The specification

type

value

A, B, C, D, E, F, G

va:A, vb:B, ..., ve:E

Hf = A-set, Hi = A-infset

J = B×C×...×D

Kf = E', Ki = E'

L = F → m

Mt = J → Kf, Mp = J ∼ Ki

N == alpha | beta | ... | omega

O == mk_Hf(as:Hf)

V == mk_Kf(el:Kf) | ...

P = Hf | Kf | L | ...

variable

vhf: Hf := ⟨⟩

channel

chf:F, chg:G, {chb[i]|i:A}:B

The indexed sort or type names, A and the indexed function names, d, are defined elsewhere in the specification. The index value names, v are usually names of bound ‘variables’ of universally or existentially quantified predicates of the indexed (“comma”-separated) P.
Example 22 ......................................................... A Neat Little “System”:

We present a self-contained specification of a simple system: The system models vehicles moving along a net, \textit{vehicle}, the recording of vehicles entering links, \textit{enter\_sensor}, the recording of vehicles leaving links, \textit{leave\_sensor}, and the \textit{road\_pricing payment} of a vehicle having traversed (\textit{entered} and \textit{left}) a link. Note that vehicles only pay when completing a link traversal; that ‘road pricing’ only commences once a vehicle enters the first link after possibly having left an earlier link (and hub); and that no \textit{road\_pricing payment} is imposed on vehicles entering, staying-in (or at) and leaving hubs.

We assume the following: that each \textit{link} is somehow associated with two pairs of \textit{sensors}: a pair of \textit{enter} and \textit{leave sensors} at one end, and a pair of \textit{enter} and \textit{leave sensors} at the other end; and a \textit{road pricing} process which records pairs of link enterings and leavings, first one, then, after any time interval, the other, with leavings leading to debiting of traversal fees; Our first specification define types, assume a net value, declares channels and state signatures of all processes.

- \textit{ves} stand for vehicle entering (link) sensor channels,
- \textit{vls} stand for vehicle leaving (link) sensor channels,
- \textit{rp} stand for ‘road pricing’ channel
- \textit{enter\_sensor(hi,li)} stand for vehicle entering [sensor] process from hub \textit{hi} to link \textit{li}.
- \textit{leave\_sensor(li,hi)} stand for vehicle leaving [sensor] process from link \textit{li} to hub \textit{hi}.
- \textit{road\_pricing()} stand for the unique ‘road pricing’ process.
- \textit{vehicle(vi)\ldots} stand for the vehicle \textit{vi} process.

\begin{verbatim}

| type | N, H, HI, LI, VI
| RPM = mk\_Enter\_L(vi:VI,li:LI) | mk\_Leave\_L(vi:VI,li:LI) |
| value | n:N
| channel | {ves[obs\_H(l),li]|li:LI,hi:H ∈ obs\_H(n)^\land li ∈ obs\_Lls(h)}:VI
| | {vls[li,obs\_H(l)]|li:LI,hi:H ∈ obs\_H(n)^\land li ∈ obs\_Lls(h)}:VI
| | rp:RPM

| type | Fee, Bal
| LVS = LI \rightarrow \text{VI-set}, \text{ FEE = LI \rightarrow Bal}, \text{ ACC = VI \rightarrow Bal |
| value | link: (li:LI \times L) \rightarrow Unit
| enter\_sensor: (hi:HI \times li:LI) \rightarrow in ves[hi,li], out rp Unit

\end{verbatim}
To understand the sensor behaviours let us review the vehicle behaviour. In the vehicle behaviour defined in Example 21, in two parts, Page 142 and Page 143 we focus on the events [7] where the vehicle enters a link, respectively [5'] where the vehicle leaves a link. These are summarised in the schematic reproduction of the vehicle behaviour description. We redirect the interactions between vehicles and links to become interactions between vehicles and enter and leave sensors.

value
\[
\delta: \text{Real} = \text{move}(h,f) \ \text{axiom} \ 0 < \delta < 1
\]
move: \(H \times F \rightarrow F\)
vehicle: \(VI \rightarrow (\text{Pos} \times \text{Net}) \rightarrow V \rightarrow \text{Unit}\)
vehicle\((vi)\)(pos,net)(v) ≡
\[
\begin{align*}
1 & \quad \text{(wait)}; \\
2 & \quad \text{vehicle}(vi)(\text{pos,net})(v)) \\
3 & \quad \text{case pos of} \\
4 & \quad \text{mk}\_\text{atH}(hi) \rightarrow \\
5 & \quad (\text{let lis}=\text{dom}(net(hi)) \text{ in let li:LI} \subseteq \text{lis in let hi'}=(\text{net}(hi))(li) \text{ in} \\
6 & \quad \text{ves}[hi,li]!vi; \\
7 & \quad \text{vehicle}(vi)(\text{mk}\_\text{onL}(hi,li,0,hi'),\text{net})(v)) \\
8 & \quad \text{end end end end) \\
9 & \quad \text{mk}\_\text{onL}(hi,li,f,hi') \rightarrow \\
10 & \quad \text{mk}\_\text{atH}(hi') \rightarrow \\
11 & \quad (\text{case f of} \\
12 & \quad 1 \rightarrow (\text{vls}[li,hi]!vi; \text{vehicle}(vi)(\text{mk}\_\text{atH}(hi'),\text{net})(v)), \\
13 & \quad \text{vehicle}(vi)(\text{mk}\_\text{onL}(hi,li,f+\delta,hi'),\text{net})(v) \\
14 & \quad \text{end) \\
15 & \quad \text{end})
\end{align*}
\]

- As mentioned on Page 145 link behaviours are associated with two pairs of sensors:
  - a pair of enter and leave sensors at one end, and
  - a pair of enter and leave sensors at the other end;

value
link\((li)\)(l) ≡
\[
\begin{align*}
\text{let } \{hi,hi'\} &= \text{obs}\_\text{Hls}(l) \text{ in} \\
\text{enter}\_\text{sensor}(hi,li) \ || \ \text{leave}\_\text{sensor}(li,hi) \ || \\
\text{enter}\_\text{sensor}(hi',li) \ || \ \text{leave}\_\text{sensor}(li,hi') \ \text{end} \\
\text{enter}\_\text{sensor}(hi,li) &\equiv \\
\text{let } vi = \text{ves}[hi,li]? \ \text{in } \text{rp!mk}\_\text{Enter}\_\text{LI}(vi,li); \text{enter}\_\text{sensor}(hi,li) \ \text{end} \\
\text{leave}\_\text{sensor}(li,hi) &\equiv \\
\text{let } vi = \text{ves}[li,hi]? \ \text{in } \text{rp!mk}\_\text{Leave}\_\text{LI}(vi,li); \text{enter}\_\text{sensor}(li,hi) \ \text{end}
\end{align*}
\]
• The LVS component of the road\textsuperscript{\textregistered} pricing behaviour serves,
  
  – among other purposes that are not mentioned here,
  
  – to record whether the movement of a vehicle’s “originates” along a link or not.

• Otherwise we leave it to the reader to carefully read the formulas.

value

\[
\begin{align*}
\text{payment: } & VI \times LI \to (ACC \times FEE) \to ACC \\
\text{payment}(vi, li)(fee, acc) & \equiv \\
& \quad \text{let } bal' = \text{if } vi \in \text{dom acc then } \text{add}(acc(vi), fee(li)) \text{ else } fee(li) \text{ end} \\
& \quad \text{in } acc[v] \mapsto bal' \text{ end} \\
\text{add: } & \text{Fee } \times \text{Bal} \to \text{Bal} \quad \text{[add fee to balance]} \\
\text{road\textsuperscript{\textregistered} pricing}(lvs, fee, acc) & \equiv \text{in rp} \\
& \quad \text{let } m = \text{rp}? \text{ in} \\
& \quad \text{case } m \text{ of} \\
& \quad \quad \text{mk\textunderscore Enter}_L(vi, li) \to \\
& \quad \quad \quad \text{road\textsuperscript{\textregistered} pricing}(lvs[v] \cup \{vi\}, fee, acc), \\
& \quad \quad \text{mk\textunderscore Leave}_L(vi, li) \to \\
& \quad \quad \quad \text{let } lvs' = \text{if } vi \in lvs(li) \text{ then } lvs[v] \cup \{vi\} \text{ else } lvs \text{ end,} \\
& \quad \quad \quad \text{acc} = \text{payment}(vi, li)(fee, acc) \text{ in} \\
& \quad \quad \quad \text{road\textsuperscript{\textregistered} pricing}(lvs', fee, acc') \\
& \quad \end{align*}
\]

End of Example 22