Start of Lecture 11: RSL SPECIFICATIONS

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:


```
type
  A, B, C, D, E, F, G
  Hf = A-set, Hi = A-infset
  J = B×C×...×D
  Kf = E*, Ki = E^ω
  L = F →m G
  Mt = J → Kf, Mp = J ~→ Ki
  N == alpha | beta | ... | omega
  O == μHf(as:Hf)
     | μKf(el:Kf) | ...
  P = Hf | Kf | L | ...
variable
  vhf:Hf := ⟨⟩
channel
  chf:F, chg:G, {chb[i]|i:A}:B
value
  va:A, vb:B, ..., ve:E
  f1: A → B, f2: C ~→ D
  f1(a) ≡ ℰf1(a)
  f2: E → in|out chf F
  f2(e) ≡ ℰf2(e)
  f3: Unit → in chf out chg Unit
  ...
axiom
  𝒫i(f1,va),
  𝒫j(f2,vb),
  ...
  𝒫k(f3,ve)
```


- 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 decalred 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
  A

value
  a:A

  * non-deterministically binds a to a value of type A.
  * Thuis includes, for example

type
  A, B

value
  f: A → B

  * which non-deterministically binds f to a function value of type A→B.

- The **axiom** clause is usually expressed as several "comma (,) separated" predicates:

$$\mathcal{P}_i(\overline{A_i},\overline{f_i},\overline{v_i}),\mathcal{P}_j(\overline{A_j},\overline{f_j},\overline{v_j}),\ldots,\mathcal{P}_k(\overline{A_k},\overline{f_k},\overline{v_k})$$

- where $(\overline{A_k, f_\ell}, \overline{v\ell})$ is an abbreviation for A_{ℓ_1}, A_{ℓ_2}, …, A_t, f_{ℓ_1}, f_{ℓ_2}, …, f_{ℓ_f}, v_{ℓ_1}, v_{ℓ_2}, …, v_{ℓ_v}.
- 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) $\mathcal{P}$.

Example 49 – A Neat Little "System":

- We present a self-contained specification of a simple system:
  – The system models
    * vehicles moving along a net, *vehicle*,
    * the recording of vehicles entering links, *enter_sensor*,
    * the recording of vehicles leaving links, *leave_sensor*, and
    * the *road_pricing payment* of a vehicle having traversed (*entered* and *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 *road_pricing payment* is imposed on vehicles entering, staying-in (or at) and leaving hubs.

  – We assume the following:
    * that each *link* is somehow 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;
    and
    * a *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.

- *ves* stand for vehicle entering (link) sensor channels,
- *vls* stand for vehicle leaving (link) sensor channels,
- *rp* stand for 'road pricing' channel
- *enter_sensor(hi,li)* stand for vehicle entering [sensor] process from hub *hi* to link (li).
- *leave_sensor(li,hi)* stand for vehicle leaving [sensor] process from link *li* to hub (hi).
- *road_pricing()* stand for the unique 'road pricing' process.
- *vehicle(vi)(...)* stand for the vehicle *vi* process.


```
type
  N, H, HI, LI, VI
  RPM == μEnter_L(vi:VI,li:LI) | μLeave_L(vi:VI,li:LI)
value
  n:N
channel
  {ves[ωHI(h),li]|h:H•h ∈ ωHs(n)∧li ∈ ωLIs(h)}:VI
  {vls[li,ωHI(h)]|h:H•h ∈ ωHs(n)∧li ∈ ωLIs(h)}:VI
  rp:RPM
type
  Fee, Bal
  LVS = LI →m VI-set, FEE = LI →m Fee, ACC = VI →m Bal
value
  link: (li:LI × L) → Unit
  enter_sensor: (hi:HI × li:LI) → in ves[hi,li],out rp Unit
  leave_sensor: (li:LI × hi:HI) → in vls[li,hi],out rp Unit
  road_pricing: (LVS×FEE×ACC) → in rp Unit
```


- To understand the sensor behaviours let us review the vehicle behaviour.
- In the *vehicle* behaviour defined in Example 48, in two parts, Slide 297 and Slide 299 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
  δ:Real = move(h,f) axiom 0<δ≪1
  move: H × F → F
```



```
  vehicle: VI → (Pos × Net) → V → Unit
  vehicle(vi)(pos,net)(v) ≡
  [1] (wait ;
  [2]  vehicle(vi)(pos,net)(v))
  [3]  ⌈⌉
     case pos of
       μatH(hi) →
  [4–6]   (let lis=dom net(hi) in let li:LI•li ∈ lis in let hi′=(net(hi))(li) in
  [7]      ves[hi,li]!vi;
  [8]      vehicle(vi)(μonL(hi,li,0,hi′),net)(v)
  [9]      end end end)
       μonL(hi,li,f,hi′) →
  [4′]     (case f of
  [5′–6′]     1 → (vls[li,hi]!vi; vehicle(vi)(μatH(hi′),net)(v)),
  [7′]        _ → vehicle(vi)(μonL(hi,li,f+δ,hi′),net)(v)
  [8′]      end)
     end
```

- As mentioned on Slide 306 *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) ≡
  let {hi,hi′} = ωHIs(l) **in**
  enter_sensor(hi,li) ∥ leave_sensor(li,hi) ∥
  enter_sensor(hi′,li) ∥ leave_sensor(li,hi′) **end**
enter_sensor(hi,li) ≡
  let vi = ves[hi,li]? **in** rp!μEnter_LI(vi,li); enter_sensor(hi,li) **end**
leave_sensor(li,hi) ≡
  let vi = ves[li,hi]? **in** rp!μLeave_LI(vi,li); enter_sensor(li,hi) **end**

- The *LVS* component of the *road_pricing* behaviour serves,
  - among other purposes that are not mentioned here,
  - to record whether the movement of a vehicles "originates" along a link or not.
- Otherwise we leave it to the student to carefully read the formulas.

value
payment: VI × LI → (ACC × FEE) → ACC
payment(vi,li)(fee,acc) ≡
  let bal′ = **if** vi ∈ **dom** acc **then** add(acc(vi),fee(li)) **else** fee(li) **end**
  in acc † [vi ↦ bal′] **end**
add: Fee × Bal → Bal [add fee to balance]

road_pricing(lvs,fee,acc) ≡ **in** rp
  let m = rp? **in**
  case m **of**
    μEnter_LI(vi,li) →
      road_pricing(lvs†[li↦lvs(li)∪{vi}],fee,acc),
    μLeave_LI(vi,li) →
      let lvs′ = **if** vi ∈ lvs(li) **then** lvs†[li↦lvs(li)\{vi}] **else** lvs **end**,
        acc′ = payment(vi,li)(fee,acc) **in**
      road_pricing(lvs′,fee,acc′)
  end end end

■ End of Example 49

End of Lecture 11: RSL SPECIFICATIONS