(A. A.5. A.5.6.)

Start of Lecture 10: RSL: IMPERATIVE & PARALLEL CONSTRUCTS

(A. A.5. A.5.6.)

A.6. Imperative Constructs

A.6.1. Statements and State Changes

Unit
value
stmt: **Unit** → **Unit**
stmt()

- 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. A.6. Imperative Constructs A.6.1. Statements and State Changes)

A.6.2. Variables and Assignment

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

A.6.3. Statement Sequences and skip

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. A.6. Imperative Constructs A.6.4. Imperative Conditionals)

A.6.5. Iterative Conditionals

6. **while** expr **do** stm **end**
7. **do** stmt **until** expr **end**

A.6.6. Iterative Sequencing

8. **for** e **in** list_expr · P(b) **do** S(b) **end**

A.7. Process Constructs

A.7.1. Process Channels

Let A, B and D stand for two types of (channel) messages and i:KIdx for channel array indexes, then:

channel
  c,c′:A
channel
  {k[i]|i:KIdx}:B
  {ch[i]i:KIdx}:B

Example 45 – Modelling Connected Links and Hubs:

- Examples (45–48) 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.

- We assume a net, $n : N$, and a set, vs, of vehicles.
- Each vehicle can potentially interact
  - with each hub and
  - with each link.
- Array channel indices *(vi,hi):IVH* and *(vi,li):IVL* serve to effect these interactions.
- Each hub can interact with each of its connected links and indices *(hi,li):IHL* serves these interactions.

type
  N, V, VI
value
  n:N, vs:V-**set**
  ωVI: V → VI
type
  H, L, HI, LI, M
  IVH = VI×HI, IVL = VI×LI, IHL = HI×LI

- 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, $vs : VS$, 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.

value
  hs:H-**set** = ωHs(n), ls:L-**set** = ωLs(n)
  his:HI-**set** = {ωHI(h)|h:H•h ∈ hs}, lis:LI-**set** = {ωLI(h)|l:L•l ∈ ls},
  ivhs:IVH-**set** = {(ωVI(v),ωHI(h))|v:V,h:H•v ∈ vs∧h ∈ hs}
  ivls:IVL-**set** = {(ωVI(v),ωLI(l))|v:V,l:L•v ∈ vs∧l ∈ ls}
  ihls:IHL-**set** = {(hi,li)|h:H,(hi,li):IHL• h ∈ hs∧hi=ωHI(h)∧li ∈ ωLIs(h)}

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

channel
{vh[i] | i:IVH · i ∈ ivhs} : M
{vl[i] | i:IVL · i ∈ ivls} : M
{hl[i] | i:IHL · i ∈ ihls} : M

■ End of Example 45

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.

value
P: **Unit** → **in** c **out** k[i] **Unit**
Q: i:KIdx → **out** c **in** k[i] **Unit**
P() ≡ ... c ? ... k[i] ! e ... ; P()
Q(i) ≡ ... k[i] ? ... c ! e ... ; Q(i)

R: **Unit** → **out** c **in** k[i] B
S: i:KIdx → **out** c **in** k[i] D
R() ≡ ... c′ ? ... ch[i] ! e ... ; B_Val_Expr
S(i) ≡ ... ch[i] ? ... c ! e ...; D_Val_Expr

Example 46 – 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.

```
value
  hub: hi:HI × h:H → in,out {hl[(hi,li)|li:LI•li ∈ ωLIs(h)]}
                     in,out {vh[(vi,hi)|vi:VI•vi ∈ vis]} Unit
  link: li:LI × l:L → in,out {hl[(hi,li)|hi:HI•hi ∈ ωHIs(l)]}
                     in,out {vh[(vi,li)|vi:VI•vi ∈ vis]} Unit
  vehicle: vi:VI → (Pos × Net) → v:V →
                     in,out {vh[(vi,hi)|hi:HI•hi ∈ his]}
                     in,out {vl[(vi,li)|li:LI•li ∈ lis]} Unit
```

■ End of Example 46

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 $\mathcal{P}$ and $\mathcal{Q}$ stand for process expressions,
- and let $\mathcal{P}_i$ stand for an indexed process expression, then:

$\mathcal{P} \parallel \mathcal{Q}$	Parallel composition
$\mathcal{P} \,\square\, \mathcal{Q}$	Nondeterministic external choice (either/or)
$\mathcal{P} \,\sqcap\, \mathcal{Q}$	Nondeterministic internal choice (either/or)
$\mathcal{P} \,\#\, \mathcal{Q}$	Interlock parallel composition
$\mathcal{O}\ \{\ \mathcal{P}_i \mid \text{i:Idx}\ \}$	Distributed composition, $\mathcal{O} = \parallel, \square, \sqcap, \#$

Example 47 – Modelling Transport Nets:

- The net, with vehicles, potential or actual, is now considered a process.
- It is the parallel composition of
  - all hub processes,
  - all link processes and
  - all vehicle processes.

```
value
  net: N → V-set → Unit
  net(n)(vs) ≡
    ‖ {hub( ωHI(h))(h)|h:H•h ∈ ωHs(n)} ‖
    ‖ {link( ωLI(l))(l)|l:L•l ∈ ωLs(n)} ‖
    ‖ {vehicle(ωVI(v))(ωPN(v))(v)|v:V•v ∈ vs}

  ωPN: V → (Pos×Net)
```


- We illustrate a schematic definition of simplified hub processes.
- The hub process alternates, internally non-deterministically, ⊓, between three sub-processes
  - a sub-process which serves the link-hub connections,
  - a sub-process which serves thos 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 !

$$
\begin{array}{l}
\mathsf{hub(hi)(h)} \equiv \\
\quad \square\{\mathbf{let}\ \mathsf{m} = \mathsf{hl[(hi,li)]\ ?}\ \mathbf{in}\ \mathsf{hub(hi)}(\mathcal{E}_{h_\ell}\mathsf{(li)(m)(h)})\ \mathbf{end}|\mathsf{i{:}LI{\cdot}li} \in \omega\mathsf{LI(h)}\} \\
\ \sqcap\ \square\{\mathbf{let}\ \mathsf{m} = \mathsf{vh[(vi,hi)]\ ?}\ \mathbf{in}\ \mathsf{hub(vi)}(\mathcal{E}_{h_v}\mathsf{(vi)(m)(h)})\ \mathbf{end}|\mathsf{vi{:}VI{\cdot}vi} \in \mathsf{vis}\} \\
\ \sqcap\ \mathsf{hub(hi)}(\mathcal{E}_{h_{own}}\mathsf{(h)})
\end{array}
$$

- The three auxiliary processes:
  - $\mathcal{E}_{h_\ell}$ update the hub with respect to (wrt.) connected link, *li*, information *m*,
  - $\mathcal{E}_{h_v}$ update the hub with wrt. vehicle, *vi*, information *m*,
  - $\mathcal{E}_{h_{own}}$ 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.

$\mathcal{E}_{h_\ell}$: LI → M → H → H
$\mathcal{E}_{h_v}$: VI → M → H → H
$\mathcal{E}_{h_{own}}$: H → H

- The student is encouraged to sketch/define similarly schematic link and vehicle processes. ■ End of Example 47

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]?	input A value
[2] c!e, k[i]!e	output A value

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.

Example 48 – 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 $\overrightarrow{m}$ (LI $\overrightarrow{m}$ HI)
Pos = atH | onL
atH == μatH(hi:HI)
onL == μonL(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, *li*,
    - [4] among the links, *lis*, emanating from the hub,
    - [6] and towards a next hub, *hi′*.
  - [4,6] The *lis* and *hi′* quantities are obtained from the vehicles 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]).

```
type
  M == μL_H(li:LI,hi:HI) | μH_L(hi:HI,li:LI)
value
  vehicle: VI → (Pos × Net) → V → Unit
  vehicle(vi)(μatH(hi),net)(v) ≡
  [1]  (wait ;
  [2]   vehicle(vi)(μatH(hi),net)(v))
  [3]  ⊓
  [4]  (let lis=dom net(hi) in
  [5]   let li:LI•li ∈ lis in
  [6]   let hi′=(net(hi))(li) in
  [7]   (vh[ (vi,hi) ]!μH_L(hi,li)‖vl[ (vi,li) ]!μH_L(hi,li));
  [8]   vehicle(vi)(μonL(hi,li,0,hi′),net)(v)
  [9]   end end end)
```


- 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 == μL_H(li:LI,hi:HI) | μH_L(hi:HI,li:LI)
value
  δ:Real = move(h,f) axiom 0<δ≪1
  vehicle(vi)( μonL(hi,li,f,hi′),net)(v) ≡
  [1′]  (wait ;
  [2′]   vehicle(vi)(μonL(hi,li,f,hi′),net)(v))
  [3′]  ⊓
  [4′]  (case f of
  [5′]      1 → ((vl[ vi,hi′ ]!μL_H(li,hi′)‖vh[ vi,li ]!μL_H(li,hi′));
  [6′]             vehicle(vi)(μatH(hi′),net)(v)),
  [7′]      _ → vehicle(vi)(μonL(hi,li,f+δ,hi′),net)(v)
  [8′]   end)
  move: H × F → F
```

■ End of Example 48

End of Lecture 10: RSL IMPERATIVE & PARALLEL CONSTRUCTS