Start of Lecture 9: RSL: Logic, Λ-Calculus, Fctl. Specs.

1.3. The RSL Predicate Calculus

1.3.1. Propositional Expressions

- Let identifiers (or propositional expressions) a, b, ..., c designate Boolean values (**true** or **false** [or **chaos**]).
- Then:

false, **true**
a, b, ..., c ∼a, a∧b, a∨b, a⇒b, a=b, a≠b

- are propositional expressions having Boolean values.
- ∼, ∧, ∨, ⇒, = and ≠ are Boolean connectives (i.e., operators).
- They can be read as: *not*, *and*, *or*, *if then* (or *implies*), *equal* and *not equal*.

1.3.2. Simple Predicate Expressions

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

false, **true**
a, b, ..., c
∼a, a∧b, a∨b, a⇒b, a=b, a≠b
x=y, x≠y,
i<j, i≤j, i≥j, i≠j, i≥j, i>j

- are simple predicate expressions.

1.3.3. Quantified Expressions

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

$\forall$ x:X $\cdot$ $\mathcal{P}(x)$
$\exists$ y:Y $\cdot$ $\mathcal{Q}(y)$
$\exists$! z:Z $\cdot$ $\mathcal{R}(z)$

- are quantified expressions — also being predicate expressions.

Example 13 . . . . **Predicates Over Net Quantities:**

- From earlier examples we show some predicates:
- Example 1: Right hand side of function definition *is_two_way_link(l):*
  $\exists\ l\sigma{:}L\Sigma \cdot l\sigma \in obs\_H\Sigma(l) \wedge \textbf{card}\ l\sigma{=}2$

- Example 3:
  - The **Sorts + Observers + Axioms** part:
    - Right hand side of the wellformedness function *wf_N(n)* definition:
      $\forall\ n{:}N \cdot \textbf{card}\ obs\_Hs(n){\geq}2 \wedge \textbf{card}\ obs\_Ls(n){\geq}1 \wedge$ **axioms** 2.–3., 5.–6., and 8., (Page 13)
    - Right hand side of the wellformedness function *wf_N(hs,ls)* definition:
      $\textbf{card}\ hs{\geq}2 \wedge \textbf{card}\ ls{\geq}1 \ldots$

- The **Cartesians + Maps + Wellformedness** part:
  - Right hand side of the *wf_HUBS* wellformedness function definition:
    $\forall\ hi{:}HI \cdot hi \in \textbf{dom}\ hubs \Rightarrow obs\_HIhubs(hi){=}hi$
  - Right hand side of the *wf_LINKS* wellformedness function definition:
    $\forall\ li{:}LI \cdot li \in \textbf{dom}\ links \Rightarrow obs\_LIlinks(li){=}li$
  - Right hand side of the *wf_N(hs,ls,g)* wellformedness function definition:
    $[c]\ \textbf{dom}\ hs = \textbf{dom}\ g \wedge$
    $[d] \cup \{\textbf{dom}\ g(hi) | hi{:}HI \cdot hi \in \textbf{dom}\ g\} = \textbf{dom}\ links \wedge$
    $[e] \cup \{\textbf{rng}\ g(hi) | hi{:}HI \cdot hi \in \textbf{dom}\ g\} = \textbf{dom}\ g \wedge$
    $[f]\ \forall\ hi{:}HI \cdot hi \in \textbf{dom}\ g \Rightarrow \forall\ li{:}LI \cdot li \in \textbf{dom}\ g(hi) \Rightarrow (g(hi))(li){\neq}hi$
    $[g]\ \forall\ hi{:}HI \cdot hi \in \textbf{dom}\ g \Rightarrow \forall\ li{:}LI \cdot li \in \textbf{dom}\ g(hi) \Rightarrow$
    $\quad \exists\ hi'{:}HI \cdot hi' \in \textbf{dom}\ g \Rightarrow \exists\ !\ li{:}LI \cdot li \in \textbf{dom}\ g(hi) \Rightarrow$
    $\quad\quad (g(hi))(li) = hi' \wedge (g(hi'))(li) = hi$

. . . . **End of Example 13**

1.4. λ-Calculus + Functions

1.4.1. The λ-Calculus Syntax

```
type /∗ 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), ...
```

1.4.2. Free and Bound Variables

Let x, y be variable names and e, f be λ-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).

1.4.3. Substitution

- **subst**([N/x]x) ≡ N;
- **subst**([N/x]a) ≡ a,

  for all variables a≠ x;
- **subst**([N/x](P Q)) ≡ (**subst**([N/x]P) **subst**([N/x]Q));
- **subst**([N/x]($\lambda x \cdot P$)) ≡ λ y·P;
- **subst**([N/x](λ y·P)) ≡ $\lambda y \cdot$ **subst**([N/x]P),

  if x≠y and y is not free in N or x is not free in P;
- **subst**([N/x](λy·P)) ≡ λz·**subst**([N/z]**subst**([z/y]P)),

  if y≠x and y is free in N and x is free in P

  (where z is not free in (N P)).

1.4.4. α-Renaming and β-Reduction

- α-renaming: λx·M

  If x, y are distinct variables then replacing x by y in λx·M results in λy·**subst**([y/x]M). We can rename the formal parameter of a λ-function expression provided that no free variables of its body M thereby become bound.
- β-reduction: (λx·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. (λx·M)(N) ≡ **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 λ–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 $\mathcal{MIN}$ and $\mathcal{MAX}$ functions are meta-linguistic.
– At any moment some vehicles, *v:V*, have a *pos:Pos*ition on the net and *VP* records those.
– A *Pos*ition 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 *T*ime to *NP* ("recordings").
– Modelling traffic in this way entails a ("serious") number of well-formedness conditions. These are defined in *wf_TF* (omitted: ...).

value
 n:N
type
 T, V
 TS = T-**infset**
axiom
 ∀ ts:TS · ∃ tmin,tmax:T: tmin ∈ ts ∧ tmax ∈ ts ∧ ∀ t:T · t ∈ ts ⇒ tmin ≤ t ≤ tmax
 [that is: ts = {$\mathcal{MIN}$(ts)..$\mathcal{MAX}$(ts)}]
type
 VP = V $\overrightarrow{m}$ Pos
 TF′ = T → VP, TF = {|tf:TF′·wf_TF(tf)(n)|}
 Pos = onL | atH
 onL == mkLPos(hi:HI,li:LI,f:F,hi:HI), atH == mkHPos(hi:HI)
value
 wf_TF: TF→ N → **Bool**
 wf_TF(tf)(n) ≡ ...
 $\mathcal{DOMAIN}$: TF → TS
 $\mathcal{MIN}$, $\mathcal{MAX}$: TS → 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'.

value
 insert_V: V × T × TF → TF → TF
 insert_V(v,t,vtf)(tf) **as** tf′

- The function *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.

pre: $\mathcal{MIN}$($\mathcal{DOMAIN}$(tf)≤t≤$\mathcal{MAX}$($\mathcal{DOMAIN}$(tf)) ∧
 ∀ t′:T · t′ ∈ $\mathcal{DOMAIN}$(tf) ⇒ v ∉ **dom** tf(t′) ∧
 $\mathcal{MIN}$($\mathcal{DOMAIN}$(vtf)) = t ∧
 ∀ t′:T·t′ ∈ $\mathcal{DOMAIN}$(vtf) ⇒ **dom** vtf(t′)={v}

- 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 λ-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′ = λt″·
        let ts = DOMAIN(tf) ∪ DOMAIN(vtf) in
        if MIN(ts) ≤ t″ ≤ MAX(ts)
          then
            ((t″<t) → tf(t″),
            (t″≥t) → [ v′↦ if v′≠v then (tf(t))(v′) else (vtf(t))(v) end
                          |v′:V·v′ ∈ vehicles(tf) ])
          else chaos end
        end
 assumption: wf_TF(vtf)∧wf_TF(tf)
 theorem: wf_TF(tf′)
value
 vehicles: TF → V-set
 vehicles(tf) ≡ {v|t:T,v:V·t ∈ DOMAIN(tf)∧v ∈ dom tf(t)}
```


1.4.5. Function Signatures

For sorts we may want to postulate some functions:

```
type
 A, B, ..., C
value
 obs_B: A → B
 ...
 obs_C: A → C
```

- These functions cannot be defined.
- Once a domain is presented
  - in which sort A and sorts or types B, ... and 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
  - obs_Hs and
  - obs_Ls

  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
  - obs_HI and
  - obs_LI

  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**

1.4.6. Function Definitions

Functions can be defined explicitly:

type
  A, B
value
  f: A → B [a total function]
  f(a_expr) ≡ b_expr

  g: A $\stackrel{\sim}{\rightarrow}$ B [a partial function]
  g(a_expr) ≡ b_expr
  pre P(a_expr)
  P: A → **Bool**

- a_expr, b_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:

value
  f: A→B
  f(a_expr) **as** b
  post P(a_expr,b)
  P: A×B→**Bool**

  g: A$\stackrel{\sim}{\rightarrow}$B
  g(a_expr) **as** b
  pre P′(a_expr)
  post P(a_expr,b)
  P′: A→**Bool**

where *b* is just an identifier.

- Finally functions, f, g, ..., can be defined in terms of axioms
- over function identifiers, f, g, ..., and over identiers of function arguments and results.

type
  A, B, C, D, ...
value
  f: A → B
  g: C → D
  ...
axiom
  ∀ a:A, b:B, c:C, d:D, ...
    $\mathcal{P}_1$(f,a,b) ∧ ... ∧ $\mathcal{P}_m$(f,a,b)
    ...
    $\mathcal{Q}_1$(g,c,d) ∧ ... ∧ $\mathcal{Q}_n$(g,c,d)

Example 16 . . **Axioms over Hubs, Links and Their Observers:**

- The axioms displayed in Items 2–3 and 5–8 on Page 12 of Sect.
- demonstrates how a number of entities and observer functions are constrained
- (that is, partially defined) by function signatures.

. . . . **End of Example 16**

1.5. Other Applicative Expressions

1.5.1. Simple let Expressions

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

$$\textbf{let } \mathrm{a} = \mathcal{E}_d \textbf{ in } \mathcal{E}_b(\mathrm{a}) \textbf{ end}$$

is an "expanded" form of:

$$(\lambda \mathrm{a}.\mathcal{E}_b(\mathrm{a}))(\mathcal{E}_d)$$

1.5.2. Recursive let Expressions

Recursive **let** expressions are written as:

$$\textbf{let } \mathrm{f} = \lambda \mathrm{a}{\cdot}\mathrm{E}(\mathrm{f,a}) \textbf{ in } \mathrm{B}(\mathrm{f,a}) \textbf{ end}$$
$$\textbf{let } \mathrm{f} = (\lambda \mathrm{g}{\cdot}\lambda \mathrm{a}{\cdot}\mathrm{E}(\mathrm{g,a}))(\mathrm{f}) \textbf{ in } \mathrm{B}(\mathrm{f.a}) \textbf{ end}$$
$$\textbf{let } \mathrm{f} = \mathrm{F}(\mathrm{f}) \textbf{ in } \mathrm{E}(\mathrm{f,a}) \textbf{ end where } \mathrm{F} \equiv \lambda \mathrm{g}{\cdot}\lambda \mathrm{a}{\cdot}\mathrm{E}(\mathrm{g,a})$$
$$\textbf{let } \mathrm{f} = \mathbf{Y}\mathrm{F} \textbf{ in } \mathrm{B}(\mathrm{f,a}) \textbf{ end where } \mathbf{Y}\mathrm{F} = \mathrm{F}(\mathbf{Y}\mathrm{F})$$

- We read $\mathsf{f} = \mathbf{Y}\mathsf{F}$ as "*f is a fix point of F*".

1.5.3. Non-deterministic let Clause

- The non-deterministic **let** clause:

$$\textbf{let } \mathrm{a{:}A} \cdot \mathcal{P}(\mathrm{a}) \textbf{ in } \mathcal{B}(\mathrm{a}) \textbf{ end}$$

- expresses the non-deterministic selection of a value a of type A
- which satisfies a predicate $\mathcal{P}$(a) for evaluation in the body $\mathcal{B}$(a).
- If no a:A • P(a) the clause evaluates to **chaos**.

1.5.4. Pattern and "Wild Card" let Expressions

Patterns and *wild cards* can be used:

let {a} ∪ s = set **in** ... **end**
let {a,_} ∪ s = set **in** ... **end**

let (a,b,...,c) = cart **in** ... **end**
let (a,_,...,c) = cart **in** ... **end**

let ⟨a⟩^ℓ = list **in** ... **end**
let ⟨a,_,b⟩^ℓ = list **in** ... **end**

let [a↦b] ∪ m = map **in** ... **end**
let [a↦b,_] ∪ m = map **in** ... **end**

1.5.5. Conditionals

if b_expr **then** c_expr **else** a_expr
end

if b_expr **then** c_expr **end** ≡ /∗ same as: ∗/
  if b_expr **then** c_expr **else skip end**

if b_expr_1 **then** c_expr_1
elsif b_expr_2 **then** c_expr_2
elsif b_expr_3 **then** c_expr_3
...
elsif b_expr_n **then** c_expr_n **end**

case expr **of**
  choice_pattern_1 → expr_1,
  choice_pattern_2 → expr_2,
  ...
  choice_pattern_n_or_wild_card → expr_n **end**

Example 17 . **Choice Pattern Case Expressions: Insert Links:** We consider the meaning of the Insert operation designators.

21. 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.
22. 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.

23. If the command is of the form 2oldH(hi′,l,hi′) then
    - $\star 1$ hi′ must be the identifier of a hub in hs,
    - $\star s2$ l must not be in ls and its identifier must (also) not be observable in ls, and
    - $\star 3$ hi″ must be the identifier of a(nother) hub in hs.
24. If the command is of the form 1oldH1newH(hi,l,h) then
    - $\star 1$ hi must be the identifier of a hub in hs,
    - $\star 2$ l must not be in ls and its identifier must (also) not be observable in ls, and
    - $\star 3$ h must not be in hs and its identifier must (also) not be observable in hs.

25. 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

21 int_Insert: Insert → N $\stackrel{\sim}{\to}$ N
22′ pre_int_Insert: Ins → N → **Bool**
22″ pre_int_Insert(Ins(op))(hs,ls) ≡
⋆2 s_l(op)∉ ls ∧ obs_LI(s_l(op)) ∉ iols(ls) ∧
case op **of**
23) 2oldH(hi′,l,hi″) → {hi′,hi″}∈ iohs(hs),
24) 1oldH1newH(hi,l,h) →
hi ∈ iohs(hs) ∧ h∉ hs ∧ obs_HI(h)∉ iohs(hs),
25) 2newH(h′,l,h″) →
{h′,h″}∩ hs={} ∧ {obs_HI(h′),obs_HI(h″)}∩ iohs(hs)={}
end

26. 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.
27. 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

26: xtr_H: HI → N $\stackrel{\sim}{\to}$ H
26: xtr_H(hi)(hs,_) ≡ **let** h:H•h ∈ hs ∧ obs_HI(h)=hi **in** h **end**
pre hi ∈ iohs(hs)
27: xtr_L: HI → N $\stackrel{\sim}{\to}$ H
27: xtr_L(li)(_,ls) ≡ **let** l:L•l ∈ ls ∧ obs_LI(l)=li **in** l **end**
pre li ∈ iols(ls)

28. 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.
29. 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

aLI: H × LI → H, rLI: H × LI $\stackrel{\sim}{\to}$ H
28: aLI(h,li) **as** h′
pre li ∉ obs_LIs(h)
post obs_LIs(h′) = {li} ∪ obs_LIs(h) ∧ non_l_eq(h,h′)
29: rLI(h′,li) **as** h
pre li ∈ obs_LIs(h′) ∧ **card** obs_LIs(h′)≥2
post obs_LIs(h) = obs_LIs(h′) \ {li} ∧ non_l_eq(h,h′)

30. If the Insert command is of kind 2newH(h',l,h'') then the updated net of hubs and links, has
    - the hubs hs joined, ∪, by the set {h′,h″} and
    - the links ls joined by the singleton set of {l}.
31. If the Insert command is of kind 1oldH1newH(hi,l,h) then the updated net of hubs and links, has
    - 31.1 : the hub identified by hi updated, hi′, to reflect the link connected to that hub.
    - 31.2 : The set of hubs has the hub identified by hi replaced by the updated hub hi′ and the new hub.
    - 31.2 : The set of links augmented by the new link.
32. If the Insert command is of kind 2oldH(hi',l,hi'') then
    - 32.1–.2 : the two connecting hubs are updated to reflect the new link,
    - 32.3 : and the resulting sets of hubs and links updated.


```
int_Insert(op)(hs,ls) ≡
⋆i   case op of
30      2newH(h′,l,h″) → (hs ∪ {h′,h″},ls ∪ {l}),
31      1oldH1newH(hi,l,h) →
31.1       let h′ = aLI(xtr_H(hi,hs),obs_LI(l)) in
31.2       (hs\{xtr_H(hi,hs)}∪{h,h′},ls ∪{l}) end,
32      2oldH(hi′,l,hi″) →
32.1       let hsδ = {aLI(xtr_H(hi′,hs),obs_LI(l)),
32.2                  aLI(xtr_H(hi″,hs),obs_LI(l))} in
32.3       (hs\{xtr_H(hi′,hs),xtr_H(hi″,hs)}∪ hsδ,ls ∪{l}) end
⋆j   end
⋆k   pre pre_int_Insert(op)(hs,ls)
```


33. The remove command is of the form Rmv(li) for some li.
34. We now sketch the meaning of removing a link:
    (a) The link identifier, li, is, by the pre_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.


```
value
33  int_Remove: Rmv → N ~→ N
34  int_Remove(Rmv(li))(hs,ls) ≡
34(a))  let l = xtr_L(li)(ls), {hi′,hi″} = obs_HIs(l) in
34(b))  let {h′,h″} = {xtr_H(hi′,hs),xtr_H(hi″,hs)} in
34(c))  let hs′ = cond_rmv(h′,hs) ∪ cond_rmv_H(h″,hs) in
34(d))  (hs\{h′,h″} ∪ hs′,ls\{l}) end end end
34(a))  pre li ∈ iols(ls)

cond_rmv: LI × H × H-set → H-set
cond_rmv(li,h,hs) ≡
34((c))i)   if obs_HIs(h)={li} then {}
34((c))ii)   else {sLI(li,h)} end
pre li ∈ obs_HIs(h)
```

.... **End of Example 17**

1.5.6. 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$\rangle$ ::=
  $- \mid \sim \mid \cup \mid \cap \mid$ **card** | **len** | **inds** | **elems** | **hd** | **tl** | **dom** | **rng**
$\langle$Infix_Op$\rangle$ ::=
  $= \mid \neq \mid \equiv \mid + \mid - \mid * \mid \uparrow \mid / \mid < \mid \leq \mid \geq \mid > \mid \wedge \mid \vee \mid \Rightarrow$
  $\mid \in \mid \notin \mid \cup \mid \cap \mid \backslash \mid \subset \mid \subseteq \mid \supseteq \mid \supset \mid \widehat{\ } \mid \dagger \mid \circ$
$\langle$Suffix_Op$\rangle$ ::= !

End of Lecture 9: RSL: Logic, Λ-Calculus, Fctl. Specs.

© Dines Bjørner 2010, Fredsvej 11, DK–2840 Holte, Denmark