The RAISE Specification Language Book Comments and Typos

Anne E. Haxthausen

Date: January 29, 2004

Doc.Id.: updated version of LACOS/CRI/AH/14/V2

This document contains a list of typos, comments and proposal for improvements to:

> The RAISE Language Group.
> *The RAISE Specification Language.*
> BCS Practitioner Series.
> Prentice-Hall International, 1992.

1 List of Comments and Typos

General: In most specifications, values are specified using typings + axioms and not using explicit definitions even that this in many cases would have been possible.

General: Some of the algebraic specifications have non-desired models, which could have been avoided by more axioms.

For instance:

sec. 7.12: LIST does not ensure induction and disjointness properties.

sec. 7.13: DATABASE does not ensure an induction property.

sec. 9.4: There should be an extra axiom:

inds all_natural_numbers $\equiv$ {i | i : **Nat** • i $\geq$ 1}

sec. 9.10: As new_record is introduced in order not to bother how records are represented, it would be more natural to express the last two axioms in RECORD in the following way:

key_of(new_record(k,d)) $\equiv$ k,
data_of(new_record(k,d)) $\equiv$ d

This would also make the comparison with RECORD on page 101 easier.

sec. 10.4-5: The formulas for **rng** m, m \ s, m / s and $m_1 \circ m_2$ do only hold, if the maps (m, s, m_1 and m_2) give deterministic results when applied.

sec. 10.7, sec. 11.4: The right-hand side of the first axiom in EQUIVALENCE_RELATION should be:

$(\forall\ e : \text{Element} \bullet e \in \textbf{dom}\ r \wedge (\exists !\ p : \text{Partition\_id} \bullet r(e) \equiv p))$

p. 79, l. -2: should be: *Element* into some unique partition identifier. Every element thus belongs to one partition.

p. 89, l. 16: Unt $\leadsto$ **Unit**

sec. 12.5-6: A better specification would have used a type NonNegReal instead of **Real**.

type NonNegReal = {| r : **Real** • r ≥ 0.0 |}

sec. 12.10, sec. 13.7: A better specification would also have had axioms about 'less_than' (as in section 9.10):

axiom forall e, e_1, e_2, e_3 : Elem •
[anti_reflexive] ∼ less_than(e,e)
[transitive] less_than(e_1,e_2) ∧ less_than(e_2,e_3) ⇒ less_than(e_1,e_3)
[total_ordering] less_than(e_1,e_2) ∨ less_than(e_2,e_1) ∨ $e_1 = e_2$

p. 108, l. -1: result is under-specified $\leadsto$ result is the value **swap**.

The explanation of **swap** on p. 189 should be moved to this place.

p. 110, l. 15: fomally $\leadsto$ formally

p. 116, l. -5: ... is under-specified $\leadsto$ ... is **swap**

p. 120: Last axiom on the page is not syntactically correct RSL. It should have been:

axiom forall r : Resource, p : Pool •
let (p1,r1) = obtain(p) **in** p1 = p\{r1} ∧ r1 ∈ p **end** ≡ **true**
pre p ≠ {}

sec. 17.6 and 28.2: RATIONAL is not statically correct – overloading is not resolvable. This problem can be resolved by changing the axioms to the following:

axiom forall n,n1,n2,d,d1,d2 : **Int**, r1,r2 : Rational •
(n1 / d1) : Rational + (n2 / d2) ≡
(n1 ∗ d2 + d1 ∗ n2) / (d1 ∗ d2),
r1 − r2 ≡ r1 + (r2 ∗ ((0−1)/1)),
(n1 / d1) : Rational ∗ (n2 / d2) ≡ (n1 ∗ n2) / (d1 ∗ d2),
(n1 / d1) : Rational / (n2 / d2) ≡ (n1 ∗ d2) / (d1 ∗ n2),
(**real**) ((n / d) : Rational) ≡ (**real** n) / (**real** d)
pre d ≠ 0

Furthermore, it would be better to replace the type definition with the following one and remove the declaration / : **Int** × **Int** → **Real**

type
Rational == /(**Int** × {| i : **Int** • i ≠ 0 |})

This ensures that rational numbers can only be constructed from / and non-zero denominators.

p. 152, l. 21: ... zero for zero n ... ⇝ ... 0.0 for $n = 0.0$ ...

chap. 22: Some of the specifications in this section could be changed such that they follow the style found in the rest of the book:

1. In the axiom parts of the specifications on p. 158, 165, 166-167:
   = ⇝ ≡.
2. In the signature of is_empty on page 159, 160, 161, 162, 166 and in the signature of is_empty_a on page 165, 166:
   $\xrightarrow{\sim}$ ⇝ →.

sec. 22.2: The two first axioms ought also be explained.

sec. 22.6: A development step could be to go from the applicative LIST_A to the imperative LIST. However LIST is not a refinement of LIST_A, but there is a formal relationship between LIST_A and LIST, which is expressed in LIST_B.

p. 169 , l. -14: replced ⇝ replaced

p. 173, l. 17: That is, ..., communicates with ... ⇝ That is, ..., may communicate with ...

p. 182, l. -6: ... It often occurs proofs ... ⇝ ... It often occurs in proofs ...

chap. 26: Note, the infix combinators ';' and '⫴' may not be used in a comprehended expression as they are not associative and commutative.

p. 189, l. 4: get(i)(t) ⇝ get(i)

p. 189: In the end of the section there should be added:

"If the set in a comprehended expression is infinite the comprehended expression is equivalent to **chaos**."

There should also be added some information about side-effects.

chap. 27: Some of the specifications in this section could be changed such that they follow the style found in the rest of the book:

1. In the axiom parts of the specifications on p. 190, 196, 201, 202: $= \rightsquigarrow \equiv$
2. In the signature of is_empty on page 197 and 201, in the signature of is_empty_a on page 201 and 202, in the signature of add on page 197, 198 and 201, and in the signature of insert and defined on page 199:
   $\xrightarrow{\sim} \rightsquigarrow \rightarrow$.
3. p. 195 and 197: $\Box$ is not necessary
4. The handling of 'empty' could be changed: In 27.1-2 there could be a channel called 'empty' which the list process may input from, and in 27.3-5 there could be an interface function called 'empty'.

Below, the resulting specifications are listed:

Section 27.1:

```
LIST =
  extend LIST_A with
  class
    channel
      empty : Unit,
      is_empty : Bool,
      add, head : Int,
      tail : Unit
    value
      list : List → in empty, add, tail out is_empty, head Unit
    axiom forall l : List •
      list(l) ≡
        empty? ; list(empty_a)
        □
        is_empty!(l = empty_a) ; list(l)
        □
        let i = add? in list(add_a(i,l)) end
        □
        if ∼(l = empty_a) then head!(head_a(l)) ; list(l) else stop end
        □
        if ∼(l = empty_a) then tail? ; list(tail_a(l)) else stop end
  end
```

Section 27.2:

```
LIST =
  class
    type
      List
    channel
      empty : Unit,
      is_empty : Bool,
      add, head : Int,
      tail : Unit
    variable
      is_empty_res : Bool,
      head_res : Int
    value
      list : List → in empty, add, tail out is_empty, head Unit
    axiom forall i : Int, l : List •
      [is_empty_empty]
        (list(l) ‖ empty!()) ‖ is_empty_res := is_empty? ≡
          (list(l) ‖ empty!()) ‖ is_empty_res := true,
      [is_empty_add]
        (list(l) ‖ add!i) ‖ is_empty_res := is_empty? ≡
          (list(l) ‖ add!i) ‖ is_empty_res := false,
      [head_add]
        (list(l) ‖ add!i) ‖ head_res := head? ≡
          (list(l) ‖ add!i) ‖ head_res := i,
      [tail_add]
        (list(l) ‖ add!i) ‖ tail!() ≡ list(l),
      [add_list]
        ∃ l′ : List • list(l) ‖ add!i ≡ list(l′)
      [empty_list]
        ∃ l′ : List • list(l) ‖ empty!() ≡ list(l′)
  end
```

Section 27.3:

```
LIST =
  class
    type
      List
    value
      empty : Unit → in any out any Bool,
      is_empty : Unit → in any out any Bool,
      add : Int → in any out any Unit,
```

```
      head : Unit ~→ in any out any Int,
      tail : Unit ~→ in any out any Unit,
      list : List → in any out any Unit
    axiom forall i : Int, l : List,
                    test_bool : Bool ~→ Unit, test_int : Int ~→ Unit •
      [is_empty_empty]
        (list(l) ∥ empty()) ∥ test_bool(is_empty()) ≡
          (list(l) ∥ empty()) ∥ test_bool(true),
      [is_empty_add]
        (list(l) ∥ add(i)) ∥ test_bool(is_empty()) ≡
          (list(l) ∥ add(i)) ∥ test_bool(false),
      [head_add]
        (list(l) ∥ add(i)) ∥ test_int(head()) ≡
          (list(l) ∥ add(i)) ∥ test_int(i),
      [tail_add]
        (list(l) ∥ add(i)) ∥ tail() ≡ list(l),
      [add_list]
        ∃ l′ : List • list(l) ∥ add(i) ≡ list(l′)
      [empty_list]
        ∃ l′ : List • list(l) ∥ empty() ≡ list(l′)
  end
```

Section 27.4:

```
LIST =
  class
    type
      List = {| l : Unit → in any out any write any Unit • is_list(l) |}
    value
      is_list : (Unit → in any out any write any Unit) → Bool,
      empty : Unit → in any out any Unit,
      is_empty : Unit → in any out any Bool,
      add : Int → in any out any Unit,
      head : Unit ~→ in any out any Int,
      tail : Unit ~→ in any out any Unit
    axiom forall i : Int, l : List,
                    test_bool : Bool ~→ Unit, test_int : Int ~→ Unit •
      [is_empty_empty]
        (l() ∥ empty()) ∥ test_bool(is_empty()) ≡
          (l() ∥ empty()) ∥ test_bool(true),
      [is_empty_add]
        (l() ∥ add(i)) ∥ test_bool(is_empty()) ≡
          (l() ∥ add(i)) ∥ test_bool(false),
      [head_add]
```

(l() ∥ add(i)) ∥ test_int(head()) ≡
(l() ∥ add(i)) ∥ test_int(i),
[tail_add]
(l() ∥ add(i)) ∥ tail() ≡ l(),
[add_list]
∃ l′ : List • □ l() ∥ add(i) ≡ l′()
[empty_list]
∃ l′ : List • □ l() ∥ empty() ≡ l′()
end

Section 27.5-6: similarly.

p. 199, l. -19: look_up : Key $\xrightarrow{\sim}$ **in any out** Data $\rightsquigarrow$
look_up : Key $\xrightarrow{\sim}$ **in any out any** Data

sec. 30.4: With the current definition it would not be possible to sort those lists which contain elements which are not in any order of each other.

Either change the parameter requirement, PARTIAL_ORDER, to TOTAL_ORDER, or relax the definition of is_ordered from:

... ⇒ T.leq(list(idx_1),list(idx_2))
to:
... ⇒ (T.eq(list(idx_2),list(idx_1)) ∨ ∼ T.leq(list(idx_2),list(idx_1)))

scheme TOTAL_ORDER =
extend PARTIAL_ORDER **with**
class axiom ∀ e_1, e_2 : Elem • leq(e_1,e_2) ∨ leq(e_2,e_1) **end**

p. 206, l. 15: All variables defined ... are initialized to their initial value $\rightsquigarrow$
All variables defined ... are initialized to their initial value by the expression **initialise**

sec. 28.4: The scope rules for an extending class expression could appropriately be mentioned here.

p. 226, l. 17: However, in class_expr_2, the maximal type of a value is allowed to be a subtype of the maximal type of the corresponding value in class_expr_1. For instance,

value v : T_1 $\xrightarrow{\sim}$ T_2 statically implements **value** v : T_1 $\xrightarrow{\sim}$ **write** x T_2

sec. 30.6.2: The type of a new variable or channel must be a subtype of the type of the old variable or channel
$\rightsquigarrow$
The type of a new variable or channel must be equivalent to the type of the old variable or channel

Note that

variable id : T := e

is equivalent to

variable id : T
axiom initialise ; id := e ≡ **initialise**

sec. 30.6.2, l. 9: There is a contradiction if a variable is assigned a value outside the type of the variable. However, there is no contradiction if a value outside the type of a channel is communicated on that channel – the effect of the expression is just under-specified.

sec. 32.1:

let head_{-1} = L_1.head() **in**
  L_1.tail() ; L_2.add(head_{-1})
end

could be written simpler:

L_2.add(L_1.head()) ; L_1.tail()

Similar simplification for the case (2,1).

sec. 32.2 and 32.3: Simplifications in TWO_LISTS and MANY_LISTS as in TWO_LISTS in section 32.1.

p. 261: Full stop ('.') is missing at the end of 1. paragraph

p. 262, last spec: remove comma after 3

p. 274, l. -6: repl_value $\leadsto$ repl_val

p. 274, l. -4 and p. 275 l. 9 and p. 277, l. -6: constant_variant $\leadsto$ constructor

p. 275, l. -13: constant_variants and record_variants $\leadsto$ variants

p. 276: The second axiom declaration could be changed such that it follows the style found in the rest of the book:

$= \leadsto \equiv$.

sec. 38.5.3: The paragraph starting with "Assume the meta-function ..." can be removed.

p. 294: Text of last bullet (•) should be changed to:

For each non-axiom definition in the old class expression there is a definition in the new class expression of the same kind statically implementing it, and if this new definition is hidden then the old definition must be hidden.

p. 295, l. -13 and p. 296, l. 2: is equal to $\rightsquigarrow$ is a supertype of

p. 297, l. 3-4: ... provided that the maximal class ... is a static implementation ...
$\rightsquigarrow$
... provided that the class ... is an implementation ...

p. 300: If the array is applied to a value which is not within the index type then the result is under-specified.

p. 303, item 3.(a): ...and $\rightsquigarrow$ ... and

sec. 41.9: ... makes the restriction hold in all states ... $\rightsquigarrow$ ... makes the restriction hold ...

sec. 42.19, p. 332: 3 last lines in paragraph are not RSL and should be replaced with:

the comprehended_expr represents the same effect as **stop** (in the case of $\Box$), **swap** (in the case of $\sqcap$) and **skip** (in the case of $\|$).

sec. 42.24.2, p. 336, l. 8-9: The *value_expr* is evaluated in each of the models and a non-deterministic choice is made between the resulting effects.
$\rightsquigarrow$
If the set of models is empty or infinite then the let_expr is equivalent to **swap** and **chaos**, respectively. Otherwise, the value_expr is evaluated in each of the models and a non-deterministic choice is made between the resulting effects.

sec. 45.1: innner_pattern $\rightsquigarrow$ inner_pattern

sec. 46.3: In a qualified_op in which no qualification ... are visible $\rightsquigarrow$ In a qualified_op in which no qualification ... are hidden

p. 353, l. -13: , axiom_naming or id_or_wildcard. $\rightsquigarrow$ or axiom_naming.

p. 366, l. 3: ... does not involve communication and if ... $\rightsquigarrow$ ... does not involve communication and does not access x and if ...

p. 381: $\Box$, $\sqcap$, $\|$ and $\#$ should have associativity Left and not Right

appendix B, p. 385: The title of the appendix is misleading since the appendix states precedence of many other things than operators.

"Syntactic Precedence and Associativity"

would be more precise.

p. 384-385: The title "ASCII Forms of Greek Letters" heads a sub-paragraph of "Varying Tokens" and so should be set in a smaller font. The title "RSL Keywords" heads a sub-paragraph of "Fixed Tokens" and so should be set in a smaller font.

The current form is confusing because the structure indicates 4 categories of tokens instead of 2.

p. 385: $\equiv$ is missing from the list of reserved words

p. 390: ':' ought to be added to the list of symbols