Exercises

1 Logic

1. Below are some equivalences which always hold (are true) in classical logic. Which of them hold in RAISE's conditional logic?

   (a) $\sim(\sim a) \equiv a$

   (b) **true** $\vee$ a $\equiv$ **true**

   (c) a $\vee$ **true** $\equiv$ **true**

   (d) a $\Rightarrow$ b $\equiv \sim$a $\vee$ b

   (e) a $\vee \sim$a $\equiv$ **true**

   (f) (a $\wedge$ b) $\wedge$ c $\equiv$ a $\wedge$ (b $\wedge$ c)

   (g) (a $\vee$ b) $\vee$ c $\equiv$ a $\vee$ (b $\vee$ c)

   (h) (a = a) $\equiv$ **true**

   (i) (a $\equiv$ a) $\equiv$ **true**

2. Reduce the following expressions to simpler expressions:

   (a) **if true then false else chaos end** $\equiv$?

   (b) **if** a **then** $\sim$(a $\equiv$ **chaos**) **else false end** $\equiv$?

   Hint:
   Use the following equivalences:

   if true then a **else** b **end** $\equiv$ a (1)

   if false then a **else** b **end** $\equiv$ b (2)

   if a **then** e1 **else** e2 **end** $\equiv$ **if** a **then** e1[**true**/a] **else** e2[**false**/a] **end** (3)

   if a **then true else false end** $\equiv$ a (4)

3. Which of the following expressions are true:

   (a) $\forall$ i : **Int** • $\exists$ j : **Int** • i + j = 0

   (b) $\forall$ i : **Int** • $\exists$ j : **Nat** • i + j = 0

   (c) $\exists$ i : **Int** • $\forall$ j : **Int** • i + j = 0

4. Write an RSL value expression which expresses the fact that there is not a largest integer.

5. Complete the following definition of a function which tests whether a natural number is even.

   is_even : **Nat** $\rightarrow$ **Bool**
   is_even(n) $\equiv$ ...

2 Products

1. (a) Define a type, ‘Complex’, with an appropriate representation for complex numbers.
   (b) Define a value, ‘zero’, which represents the complex number 0 + 0i.
   (c) Define a value, ‘c’, which represents a complex number of the form x + xi, i.e. one where the real and imaginary parts are equal in magnitude.
   (d) Define functions, ‘add’ and ‘mult’, for addition and multiplication of complex numbers.
   (e) Define a function, ‘f’, which takes a complex number as argument and returns some complex number which is different.

   Hint: part (b) can be done with an explicit value definition, part (c) with an implicit value definition, part (d) with explicit function definitions, and part (e) with an implicit function definition.

3 Functions

1. Suggest types for the operators
   (a) $+$ (addition)
   (b) $*$ (multiplication)
   (c) $\backslash$ (remainder)
   (d) $\uparrow$ (exponentiation)

   Remember that they may have both integer and real versions.
2. Define a function, ‘max’, that returns the maximum of two integers in each of the following styles:
   (a) explicit function
   (b) implicit function
   (c) signature/axiom
3. Define a function, ‘approx_sqrt’, that for a given tolerance (positive real number) finds an approximation to the square root of non-negative real numbers. The approximation, approx_sqrt(x,eps), must be such that the mathematical square root, square_root(x), lies in the half open interval [approx_sqrt(x,eps),approx_sqrt(x,eps) + eps[.

4 Sets

1. Write value expressions representing the set of odd numbers between 0 and 10, as an enumerated set and as a comprehended set.
2. Do the following identities hold for sets A, B and C?
   (a) $(A \cup B) \setminus C = (A \setminus C) \cup (B \setminus C)$
   (b) $(A \setminus C) \cap B = (A \cap B) \setminus C$
3. Define a function, ‘dunion’, that takes a set, ss, of sets of elements as argument and returns the set of all those elements which are elements of some set in ss.

   Example:

   dunion({ {1,2}, {7,1}, {5} }) = {1,2,7,5}

5 Lists

1. Define the following functions:
   (a) 'length' which calculates the length of a list, without using the built-in operator **len**,
   (b) 'rev', that reverses the elements of a list,
   (c) 'drev', that takes a list of lists of elements as argument, and doubly reverses it.
   Example:
   drev(⟨⟨e11, ..., e1i⟩,...,⟨en1, ..., enm⟩⟩) ≡
   ⟨⟨enm, ..., en1⟩,...,⟨e1i, ..., e11⟩⟩

2. Define a function, 'pascal', which generates Pascal triangles up to the order n. That is a list of n lists, the latter being the rows of the triangle. The first rows of such a triangle looks like:

⟨1⟩
⟨1 , 1⟩
⟨1 , 2 , 1⟩
⟨1 , 3 , 3 , 1⟩
⟨1 , 4 , 6 , 4 , 1⟩

   The nth row, for n>1, starts, and ends with 1, and its ith element, for 1<i<n, is the sum of the (i-1)st and ith element of the (n-1)st row.

3. A system for checking words on pages etc. is to be specified. A page consists of lines of words. Punctuation marks should be ignored. Define a module, 'PAGE', providing
   (a) types 'Page', 'Line' and 'Word',
   (b) a function, 'is_on', which checks that a given word is on a given page,
   (c) a function, 'number_of', which gives the number of occurrences of a particular word on a given page,
   (d) a type, 'Dict', of dictionaries of correctly spelled words,
   (e) a function, 'spell_check', that for a given page returns the words which are not correctly spelled with respect to a given dictionary of correct words.

6 Maps

1. In MAP_DATABASE amend 'insert' so that it also reports if the new key was already present, and 'remove' so that it reports if the key to be deleted was not previously present.

2. Generalise the new version of insert to one that merges two databases, the second taking precedence, but reporting on keys that are common to both databases.

3. Add to MAP_DATABASE a function to report on keys in a database that satisfy some predicate of type 'Key → **Bool**' (supplied as a parameter).

7 Subtypes

1. Write subtype expressions for
   (a) finite integer lists of at least two elements

(b) finite integer lists with no repetitions

(c) finite non-empty sets of natural numbers

2. For a given type T, write a subtype expression which represents T-**set** as a subtype of T-**infset** and a subtype expression which represents T^* as a subtype of T^ω.

3. In each of the following cases decide what subtype relations there are between the Ts:

   (a) $T_1 = \{|\ i : \textbf{Nat} \bullet \text{is\_even(i)}\ |\}$
   $T_2 = \{|\ i : \textbf{Nat} \bullet \text{is\_even(i)} \wedge i \geq 5\ |\}$

   (b) $T_1 = \{|\ i : \textbf{Int} \bullet i/2 = 0\ |\}$
   $T_2 = \{|\ i : \textbf{Int} \bullet i/2 = 1\ |\}$

   (c) $T_1 = A \rightarrow B$
   $T_2 = A \xrightarrow{\sim} B$

   (d) $T_1 = \textbf{Int} \rightarrow \textbf{Int}$
   $T_2 = \textbf{Nat} \rightarrow \textbf{Int}$
   $T_3 = \textbf{Int} \rightarrow \textbf{Nat}$
   $T_4 = \textbf{Nat} \rightarrow \textbf{Nat}$

8 Type definitions

These exercises are based on the type Tree defined by

type
Tree == nil | node(left : Tree, val : **Int**, right : Tree)

1. Define a function ‘depth’ that returns the depth of a tree.

2. Define a function ‘is_in’ to find if an integer is in a tree.

3. Define the subtype ‘Ordered_tree’. The subtype should not allow repetitions, so that an ordered tree models a set.

4. Define a function ‘is_in_ordered’ to find if an integer is in an ordered tree.

5. Define a total function ‘add’ to add an integer to an ordered tree.

6. Define a total function ‘remove’ to remove an integer from an ordered tree.

9 Imperative Specification

1. Given a variable definition:

   variable x : **Int** := 0

   Reduce the following expressions:

   (a) x := 1; x := 2 ≡

(b) x := x + 1 ≡

(c) **initialise**; x ≡

(d) ((x := 1 ; x) ≡ (x := 0 ; x+1)) ≡

(e) ((x := 1 ; x) = (x := 0 ; x+1)) ≡

2. Write a model-oriented, imperative specification of stacks. The specification must provide a function 'empty' which makes the stack empty, a function 'push' which pushes an element on the stack, a function 'is_empty' which tests whether the stack is empty, a function 'top' which gives the top element (last inserted element) of the stack (without removing it) and a function 'pop' which removes the top element of the stack.

   Use explicit function definitions.

3. Write a model-oriented, imperative specification of stacks. Use implicit function definitions (i.e. use post expressions – don't use equivalence expressions).

10 Concurrency

1. Define a process that continuously inputs two integers from two channels *l1* and *l2*, one from each channel, and outputs their maximum on a channel *r*.

2. Define a process that continuously inputs two integers from two channels *l1* and *l2*, one from each channel, and outputs their maximum on one channel *r1* and their minimum on another channel *r2*.

3. Consider the following specification:

```
scheme SEMAPHORE =
  hide get, release, semaphore in
  class
    type
      Process = Unit ~→ in any out any Unit
   channel get, release : Unit
    value
      system, p1, p2, p3, f1, f2, f3, semaphore : Process
    axiom
      p1() ≡ f1() ; p1(),
      p2() ≡ f2() ; p2(),
      p3() ≡ f3() ; p3(),
      semaphore() ≡ skip,
      system() ≡ p1() ‖ p2() ‖ p3() ‖ semaphore()
  end
```

   We want to use the semaphore to ensure that only one of the processes 'f1', 'f2', and 'f3' can run at any one time, regardless of how they are implemented. Change the first 4 axioms only of 'SEMAPHORE' to achieve this.

   Hint: The accesses of 'p1', 'p2', 'p3' and 'semaphore' mean that they can input from or output to the channels 'get' and 'release'.

11 Implementation

1. Which of the following type definitions are implementations of others?

(a) **type** t

(b) **type** t = **Int**

(c) **type** t = **Nat**

2. Which of the following value definitions are implementations of others?

(a) **value** x : **Int**

(b) **value** x : **Int** • $x > 2$

(c) **value** x : **Int** $= 2$

(d) **value** x : **Nat**

(e) **value** x : **Nat** • $x > 2$

(f) **value** x : **Nat** $= 2$

(g) **value** x : **Nat** • $x < 0$

3. Which of the following function definitions are implementations of others?

(a) **value**
f : **Int** $\xrightarrow{\sim}$ **Int**
f(x) $\equiv$ x
pre $x \geq 0$

(b) **value**
f : **Int** $\xrightarrow{\sim}$ **Int**
f(x) $\equiv$ x
pre $x \geq 2$

(c) **value**
f : **Int** $\rightarrow$ **Int**
f(x) $\equiv$ x