CO3408 Advanced Software Engineering Techniques

Tutorial Exercise Book 3: Formal Specification ***with answers***

Requirements Specification

Assume the existence of an array of strings (names):

names

jim,	fred,	joan,	jean,	allan,	bert

1. Are there any faults with the following informal specification of a function, find, to find the position of a string?

   The routine will take a string and an array of strings. It will return the position in the array of the string.

   It doesn't define the behaviour if the string is not present in the array.
   You could query whether the maximum lengths of strings and arrays should be specified

2. Are there any faults with the following more formal specifications of the function, find.

```
i.int find(string target, stringArray theArray) = r
  pre-condition: none
  post-condition: theArray[r] == target
```

   Impossible to satisfy the post-condition if the target isn't present.
   Maybe you could modify theArray by inserting the target into it, e.g. at position 0. This would allow the pre-condition to be satisfied. But almost certainly isn't what the user of this function wants.

```
ii.int find(string target, stringArray theArray) = r
   pre-condition: none
   post-condition: theArray[r] == target || r == -1
```

   This allows -1 to be returned if the item is not present, but it doesn't stop -1 being returned all the time, which will satisfy the post-condition, but is useless.

```
iii.int find(string target, stringArray theArray) = r
     pre-condition: theArray.length() > 0
     post-condition: theArray[r] == target
                     && theArray is unchanged
```

   Better than (i) but can't handle an item that's not present.
   The best solution is

```
int find(string target, stringArray theArray) = r
pre-condition: theArray.length() > 0
post-condition: (theArray[r] == target || There is no value of
x such that theArray[x] == target and r == -1)
&& theArray is unchanged
```

   We haven't looked at the notation for "there is no value such that" yet

3. Using the above notation, specify the following functions
   i. A function to return the larger of two numbers

```
int bigger(int x, int y) = r
pre-condition: true
post-condition: x == r && x >= y || y == r && x < y
```

Note: pre-condition true means there are no constraints on the parameters. Double rather than int might be better.

ii. A function to return the perimeter (the sum of the length of the sides) of a triangle given the lengths of the three sides

```
int perimeter(int x, int y, int z) = r
pre-condition: x > 0 && y > 0 && z > 0
                && x+y > z && x+z > y && z+y > x

post-condition: x + y + z == r
```

4. Define test cases for the functions specified in [3]. Does it help to have a formal specification? Alternatively, would it be easier to develop the formal specification given the test cases?

The answer to both is probably yes: the pre-condition and the post-condition can help to check that the test cases are valid (the input must satisfy the pre-condition) and the output must satisfy the post condition. Similarly test cases can help to suggest pre-conditions and to identify features of the post-condition. A test case where x = y would help to ensure that the post-condition to 1 is correct (an easily made mistake would be:

```
x == r && x > y || y == r && x < y)
```

5. Simplify the following expressions (Hint: draw truth tables)

Symbol	Meaning
$\vee$	or
$\wedge$	and
$\neg$	not

(a) $x \vee \neg x$ *true, since one or other must be true*
(b) $x \wedge \neg x$ *false, since one or other must be false*
(c) $x \vee x$ *x – draw a truth table to see the expression has the same values as x*
(d) $x \wedge x$ *x*
(e) $x \vee (y \vee x) \vee \neg y$ *true since at least one of y or $\neg y$ must be true*
(f) $\neg (x \wedge y) \Leftrightarrow \neg x \vee \neg y$ *true – draw the truth table to show that $\neg (x \wedge y)$ and $\neg x \vee \neg y$ have the same truth values so the bi-implication will be true*

6. Show that $a \Leftrightarrow b$ is the same as $(a \Rightarrow b) \wedge (b \Rightarrow a)$ (Hint: draw a truth table)

a	*b*	*a ⇒ b*	*b ⇒ a*	*(a ⇒ b) ∧ (b ⇒ a)*	*a ⇔ b*
T	*T*	*T*	*T*	*T*	*T*
T	*F*	*F*	*T*	*F*	*F*
F	*T*	*T*	*F*	*F*	*F*
F	*F*	*T*	*T*	*T*	*T*

7. Comment on the following predicates over the Boolean type (B):

(a) $\forall x,y \in B \bullet (x \wedge y)$ $\forall x,y \in B \bullet (x \vee y)$

$\forall x,y \in B \bullet (x \Leftrightarrow y)$ $\forall x,y \in B \bullet (x \Rightarrow y)$

These are all false since there is at least one combination of x and y making the bound sub-expression false.

(b) $\exists x,y \in B \bullet (x \wedge y)$ $\exists x,y \in B \bullet (x \vee y)$

$\exists x,y \in B \bullet (x \Leftrightarrow y)$ $\exists x,y \in B \bullet (x \Rightarrow y)$

These are all true since there is at least one combination of x and y making the bound sub-expression true.

(c) $\exists! x,y \in B \bullet (x \wedge y)$ *True: only T, T* $\exists! x,y \in B \bullet (x \vee y)$ *False: e.g. T,T and T,F*

$\exists! x,y \in B \bullet (x \Leftrightarrow y)$ *False: T,T and F,F* $\exists! x,y \in B \bullet (x \Rightarrow y)$ *False: e.g. T,T and F,F*

8. Consider the following predicates defined over the domain of vegetables:

   R(x) : x is a root vegetable

   T(x) : x tastes nice

   Additionally the proposition P is defined as:

   P : Peas are blue

   Express the following statements symbolically:

   (a) If peas are blue then cabbage tastes nice. $P \Rightarrow T(cabbage)$

   (b) There is a root vegetable that tastes nice. $\exists x \in vegetable \bullet R(x) \wedge T(x)$

   (c) Lettuce is a root vegetable and peas are blue. $R(Lettuce) \wedge P$

   (d) All root vegetables taste nice. $\forall x \in vegetable \bullet R(x) \Rightarrow T(x)$

   (e) Peas are blue or there is a vegetable which is a root vegetable and which does not taste nice. $P \wedge \exists x \in vegetable \bullet R(x) \wedge \neg T(x)$

9. Write a specification (pre-condition and post-condition) for a function that tests if an array of integers is sorted.

   boolean isSorted(int[] arr)

   requires true

   ensures Result== ForAll(1, arr.Length, x => arr[x] >= arr[x-1]

   Note: the informal specification is incomplete: could be sorted into ascending or descending order – a literal interpretation might be that the function will return true in either case – if so add another clause to the above specification "$\vee \forall x \in \{1..len\ arr-1\} \bullet arr[x] \leq arr[x-1]$". Assume that duplicate items are ok.

   What about arrays of size 0 or 1? The above assumes that they are regarded as sorted – the domain of x in the quantifier will be empty, so the quantifier will automatically be true

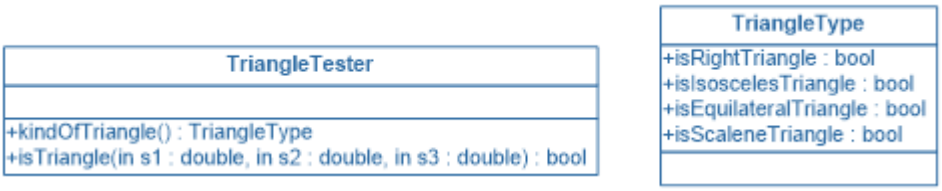
   Make sure that you have included "r=" in the post-condition. It's easy to forget it when thinking about how to derive an expression that will be true when the array is sorted.

   Make sure that $\geq$ is used.

10. Formal Specification of a classification operation

context: TriangleTester::kindOfTriangle(s1, s2, s3 :: Real) : TriangleType
pre: TriangleTester::isTriangle(s1, s2, s3)
post: if $((s1^2 = s2^2 + s3^2)$ or $(s2^2 = s1^2 + s3^2)$ or $(s3^2 = s2^2 + s1^2))$
then
result.isRightTriangle = true
else
result.isRightTriangle = false
endif
if ((s1 = s2) or (s1 = s3) or(s2 = s3))
then
result.isIsoscelesTriangle = true
else
result.isIsoscelesTriangle = false
endif
if ((s1 = s2) and (s2 = s3))
then
result.isEquilateralTriangle = true
else
result.isEquilateralTriangle = false
endif
if result.isIsoscelesTriangle = false
then
result.isScaleneTriangle = true
else
result.isScaleneTriangle = false
endif

a) Draw a simple class diagram showing the classes TriangleTester and TriangleType and the methods and properties required for the above OCL.



b) Define TriangleTester::isTriangle.

context: *TriangleTester::isTriangle(s1, s2, s3 :: Real) : Boolean*
pre: *true*
post: *(s1 < s2 + s3 and s2 < s1 + s3) and s3 < s2 + s1 and s1 > 0 and s2 > 0 and s3 > 0) = result*

c) Why can the OCL specification use isTriangle?

*Because it has no side-effects (In mathematical terms, it is a **pure** method. However, pure in C++ means something else)*

d) What do the four different classifications mean in English?

A right-angled triangle has a right angle (it satisfies Pythagoras' theorem), an Isosceles triangle has two equal sides, an equilateral triangle has three equal sides, a Scalene triangle has three sides of different lengths. Note that a right angle triangle can be isosceles or scalene, but not equilateral.

e) Simplify the post-condition.

result.isRightTriangle = ($s1^2 = s2^2 + s3^2$) or ($s2^2 = s1^2 + s3^2$) or ($s3^2 = s2^2 + s1^2$)

result.isEquilateralTriangle = (s1 = s2) and (s2 = s3)

result.isIsoscelesTriangle = (s1 = s2) or (s1 = s3) or (s2 = s3)

result.isScaleneTriangle = not result.isIsoscelesTriangle

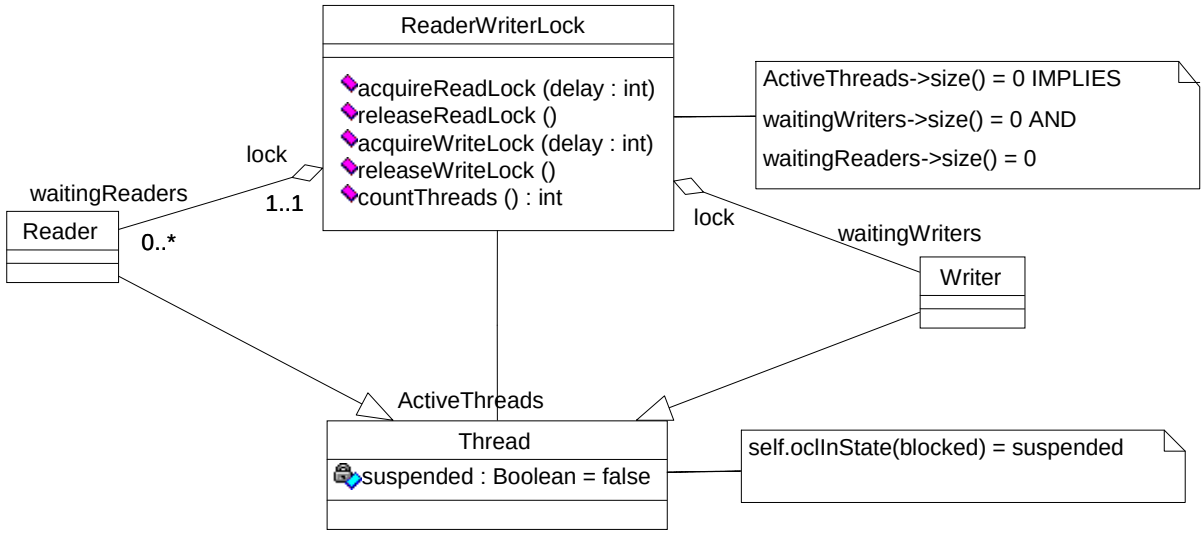
f) Can the specification be useful to a tester?

Yes, it you can use the pre- and post- conditions to help pick test cases. If nothing else, you would want to ensure that each alternative in the post-condition had been used in testing.

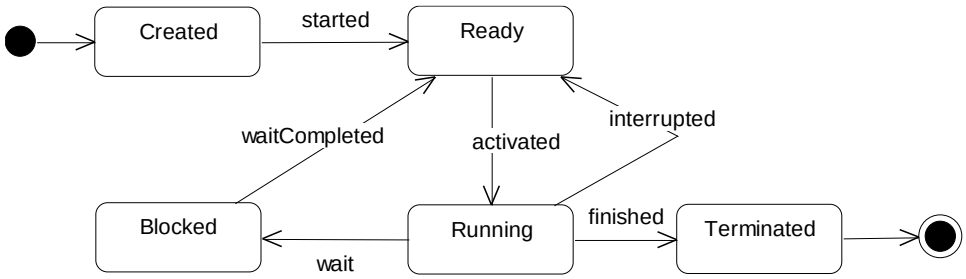
g) Does the formal specification have any advantages over a natural language specification?

For discussion. Possible points: more precise and less ambiguous. In this case it's hard to say that an informal specification would be simpler to understand.

11. Applying OCL



Class diagram for a ReaderWriterLock



State diagram for the Thread class

a) What do the OCL annotations on the class diagram mean?

If there are no ActiveThreads, there must be no waitingReaders or waitingWriters
The suspended property of a Thread indicates whether it is in a blocked state.

b) Check the OCL annotations. Are they correct? Can they be improved? Is it possible to improve a correct invariant? [Hints: You may find the following OCL predicates (effectively boolean functions) and functions useful:

item.oclisType(TypeName) [true if the item is of the given type],

aCollection->forAll(variablelist | boolExpression), aCollection->forAll(| boolExpression),

aCollection->Exists(variablelist | boolExpression), aCollection->Exists(boolExpression),

aCollection->select(theTest) [items from aCollection meeting a test], aCollection->size()

aCollection->includes(anItem)

aColl = aColl@pre->including(anItem) – aColl is the old collection plus anItem

Note that X.b is used when X is an object. X->b is used when X is a collection]

They are correct in the sense that they are not wrong and are consistent with our understanding of a ReaderWriterLock.

They could be improved e.g. if there are any activethreads, either there is precisely one writer active or there are any number of readers.

c) There is no role at the ReaderWriterLock end of its association with Thread. How would this be handled if you wanted to refer to the ReaderWriterLock end of this association in an OCL expression? What if the waitingWriters role had been omitted?

You would refer to it as from a Thread as self.ReaderWriterLock since only one is possible.

Refer to it from a Thread as self.ReaderWriterLock.Writers since it is a collection.

d) Complete the multiplicities on the diagram

The waitingWriters and the Threads should have the same multiplicity as waitingReaders

e) Define pre and post conditions for countThreads, which calculates the number of threads currently involved with a ReadWriteLock.

context: *ReadWriteLock::countThreads() : int*
pre: *true*
post: *result = waitingWriters->size() + waitingReaders->size() + ActiveThreads->size()*

f) Express pre & post conditions for releaseReadLock() in English.

Pre: The current thread must be an ActiveThread and must be a Reader
Post: The current thread has been removed from the ActiveThreads. If the current thread was the only reader on the ActiveThreads and there was a thread in waitingWriters, a thread must be transferred between waitingWriters and ActiveThreads.

g) Define pre and post conditions for releaseReadLock().

context: *ReadWriteLock:: releaseReadLock ()*
pre: *ActiveThreads.includes(currentThread)*
post: *ActiveThreads = ActiveThreads@pre->excluding(currentThread) and*
(ActiveThreads@pre->size()=1 and waitingWriters@pre->size()>0 implies
ActiveThreads@pre->size()=1 and
waitingWriters@pre->size() = waitingWriters->size()+1
and waitingWriters@pre = waitingWriters.union(ActiveThreads)

h) Have you identified any issues because of the formal specification?
i) Define pre and post conditions for acquireReadLock()

context: *ReadWriteLock:: acquireReadLock ()*
pre: *not ActiveThreads->includes(currentThread) -- this implies this isn't a re-entrant lock*
post: *if (ActiveThreads@pre->size() = 0 or ActiveThreads@pre->forAll(self.oclisType(Reader))*
then
ActiveThreads= ActiveThreads@pre->including(currentThread))
else
waitingReaders = waitingReaders@pre->including(currentThread)

12. Evaluation
    a) Has the formal modelling increased your understanding?
    b) Could you have gained that understanding (or more understanding) by other methods?

Both of these are open to discussion.