[CSC3304]

NEWCASTLE UNIVERSITY

SEMESTER 1 2010/11

SOFTWARE VERIFICATION TECHNOLOGY

Time allowed – 1½ Hours

Instructions to candidates:

Answer ALL questions

Marks shown for subsections are indicative only

[Turn over

Question 1.

Answer ALL parts of this question.

a) Briefly explain the difference between partial correctness and total correctness in software verification. Explain the role of a variant in proving the total correctness of a while loop, illustrating your answer with a total correctness rule for such a loop. [4 marks]

b) Consider a simple programming language defined using the Floyd/Hoare rules printed opposite. Using these rules, prove:

```
{y = x+z and z > 0}
x := x+1; z := z-1
{y = x+z and z >= 0}
```

[5 marks]

c) Suppose that we wish to add an **`until`** loop statement to the programming language. The statement has the following form:

do S **until** b

where S is a statement and b is a Boolean expression. The meaning of the loop is as follows: execute S and then evaluate b; if b is true, exit the loop, otherwise repeat. Note that S is always executed at least once.

Suggest a Floyd/Hoare rule for partial correctness of **`until`** loops. Explain your reasoning. [6 marks]

Floyd-Hoare Rules for Partial Correctness

Assignment:

$$\frac{}{\texttt{\{P[E/x]\} x := E \{P\}}}$$

Sequential composition:

$$\frac{\texttt{\{P\} S1 \{Q\}, \{Q\} S2 \{R\}}}{\texttt{\{P\} S1;S2 \{R\}}}$$

Conditional:

$$\frac{\texttt{\{P \textbf{and} C\} S1 \{R\}, \{P \textbf{and} \textbf{not} C\} S2 \{R\}}}{\texttt{\{P\} \textbf{if} C \textbf{then} S1 \textbf{else} S2 \{R\}}}$$

Consequence:

$$\frac{\texttt{P' => P, \{P\} S \{Q\}, Q => Q'}}{\texttt{\{P'\} S \{Q'\}}}$$

While Loop:

$$\frac{\texttt{\{I \textbf{and} C\} S \{I\}}}{\texttt{\{I\} \textbf{while} C \textbf{do} S \textbf{end} \{I \textbf{and} \textbf{not} C\}}}$$

Question 2.

Answer ALL parts of this question. For reference, a basic VDM language summary is printed opposite.

a) An abstract design for a database system views the database as a set of records, with each record containing two fields: a key field and the associated data. This is specified in VDM as follows (the definitions of the types `Identifier` and `Data` are not significant):

```
DBA = set of Record

Record :: key : Identifier
          d   : Data
```

The type `Record` is not constrained by any invariants. It is decided to realise the design as a mapping from keys to data:

```
DBC = map Identifier to Data
```

Define and informally explain the totality and adequacy proof obligations that must be met by this design. [4 marks]

b) The following retrieve function is proposed:

```
retr: DBC -> DBA

retr(dbc) == {mk_Record(k,dbc(k))|k in set dom dbc}
```

The function creates a set consisting of records, each of which contains a key and the data to which that key mapped in the concrete model.

Do you believe this retrieve function to be *adequate*? If so, give an argument in support of your claim. If not, give a counter-example and suggest what additional conditions would have to be met in order to ensure adequacy. [6 marks]

c) The following operation adds a new record to the abstract database of type DBA. Give a concrete refinement of this operation over the type DBC and define the domain and result proof obligations arising from your proposed refinement.

```
NewRecA(k:Identifier,d:Data)
ext wr dba:DBA
pre not exists r in set dba & r.key = k
post dba = dba~ union {mk_Record(k,d)}
```

[5 marks]

Basic VDM Language Summary

Base Types

`nat, nat1`	Natural numbers (from 0 or 1)
`bool`	Booleans
`char`	Characters
`token`	Structureless tokens

Records

```
R :: f1: T1
     ...
     fn: Tn
```

Record Type (n fields)

`mk_Record(x1,...,xn)`	record constructor
`r.f1`	field selector

Collections

`set of` `T`	Finite sets of values from type `T` (duplicates are immaterial)
`seq of` `T`	Finite sequences of values from type `T` (duplicates are significant; indexing starts at 1)
`map` `T1` **`to`** `T2`	Finite mappings from elements of `T1` to `T2` (mappings are many-to-one, not one-to-many)

Basic operators

Sets: Consider `s, s1, s2:` **`set of`** `X`

`card` `s`	cardinality of `s`
`union`	set formed by union of `s1` and `s2`
`inter`	set formed by intersection of `s1` and `s2`
`s1\s2`	set formed by removing elements of `s2` from `s1`

Sequences. Consider `s, s1, s2:` **`seq of`** `X`

`s(n)`	nth element of sequence `s`
`hd`	first element of `s`
`tl`	sequence formed from `s(2,...,len s)`
`elems`	set of elements of `s`
`len`	length of `s`
`s1^s2`	concatenation of sequences `s1` and `s2`

Mappings: Consider `m, m1,m2: map X to Y`

`m(x)`	range value (of type `Y`) mapped to by domain value `x`
`dom` `m`	domain of `m` (a set of type `set of X`)
`rng`	range of `m` (a set of type `set of Y`)
`m1 ++ m2`	`m1` overridden by `m2`.

[Turn over

Question 3.

Answer ALL parts of this question

a) Use a code example of your own choosing to explain the *Design by Contract* approach for Software Development. Give two advantages for this approach, illustrated by your example. [6 marks]

In the code fragment shown below, a, b, x and w are positive integer variables. Line numbers are shown on the left for ease of reference.

```
1.    read(a);
2.    read(b);
3.    if a > b
4.    then read(x);
5.    endif
6.    a:= a+1;
7.    w:= a;
8.    if a-b > 1
9.    then w:=x;
10.   endif
```

b) Use the code fragment to show how a Control-Flow Graph (CFG) is used to model a piece of code. [3 marks]

c) Use the code fragment and the CFG to show how a static analyser checks code for variable usage [4 marks]

d) Explain why the process of static analysis described in your answer to c) may not always produce a definite answer. [2 marks]