Exam Program Verification 2017/2018

12th Oct 2017, 11:00–12:45

Lecturer: Wishnu Prasetya

1. **Program Semantic** [1.5 pt].

   Consider a simple programming language L_0 where we can write a program like this:

   > **vars** `x`, `y`, `z` ;
   > **init** `x`> `0` && `10`>`y` ;
   > { `z:=0` ; **if** `x/y` > `2` **then** `z:=2` **else** `z:=1` }

   The **init**-part specifies allowed initial states: the program can only execute on an initial state on which the **init**-predicate would evaluate to true. If the program is invoked on such a state, it will then execute its body-statement. At the end, the program's final state will be returned. It will furthermore mark the state as either 'normal' or 'exceptional'. A state is marked as exceptional if the program terminates by throwing an exception. In the above case, this would happen if for example the value of `y` is 0 in the division `x/y`.

   The full syntax of our programming language is as follows:

Program	$\rightarrow$	**vars** *variables* ;	(variables declaration)
		init *expression* ;	(allowed initial state)
		body;	
variables	$\rightarrow$	one or more identifiers (variable-name) separated by ","	
body	$\rightarrow$	"{" one or more statements separated by ";" "}"	
statement	$\rightarrow$	$identifier := expression$	(assignment)
	|	**if** *expression* **then** *body* **else** *body*	
expression	$\rightarrow$	integer constants like 0,1,2,...	
	|	"`undefined`"	
	|	$identifier$	
	|	$expression\ +\ expression$	
	|	$expression\ /\ expression$	(the div operator)
	|	$expression == expression$	(testing equality)
	|	$expression > expression$	(greater than)
	|	$expression\ \&\&\ expression$	('and' operator)

   - e_1/e_2 results in `undefined` if e_2 evaluates to zero.
   - All the binary operators above, e_1 *op* e_2, result in `undefined` if one of its arguments evaluates to `undefined`.
   - An assignment $x{:=}e$ throws an exception if e evaluates to `undefined`. The program will then break its execution, and its state is left as it was just before the assignment.
   - The statement **if** e **then** ... **else** ... throws an exception if e evaluates to `undefined`. The program will then break its execution, and its state is left as it was just before the **if**.

Your tasks:

(a) *Provide a denotational semantic* for the above programming language. You will need to provide the functions $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{P}$ that describe the semantic of respectively expressions, statements, and programs. Hint: choose a proper semantical domain for each.

(b) Suppose we extend the syntax so that we can also write a post-condition. A post-condition will be written as a pair **post** p **exceptional** q to mean that the program should either terminate normally in a state satisfying p, or terminate exceptionally in a state satisfying q. For example, the post-condition below is a valid one:

> **vars** x, y, z ;
> **init** x> 0 && 10>y ;
> **post** z> 0 **exceptional** z==0
> { z:=0 ; **if** x/y > 2 **then** z:=2 **else** z:=1 }

Extend your denotational semantic so that the above semantic of post-condition is defined (and reasonable).

2. **Loop Invariant** [1.5 pt].

Give an invariant for each of the GCL loops below. It should be an invariant that is consistent, strong enough to realize the asked post-condition, and realistic to be established by the pre-condition or initialization of the loop. Use the *partial correctness* interpretation of Hoare triples.

Below, `a` is an infinite array of `int`; `b` is of type `bool`; other variables are of type `int`.

(a) { x = 100 } **while** x>0 **do** {x := x−2 } { x=0 }

(b) { x=10 ∧ y=0 } **while** x>0 **do** {x := x−1 ; y := y+10 } { x+y=100 }

(c) { x=100 ∧ y=1 } **while** x>y **do** {y := y∗2 } { y=128 }

(d) Here is a program to check if an array consists of only 0's:

> { k=0 ∧ allzeros=**true** }
>
> **while** k<N ∧ allzeros **do** { allzeros := (a[k]=0) ; k:=k+1 }
>
> { allzeros = (∀i : 0≤i<N : a[i] = 0) }

(e)

> { **true** }
>
> k, found := 0, **false** ;
> **while** ¬found **do** { found := (a[k]=0) ; k:=k+1 }
>
> { k ≥ 0 ∧ a[k−1]=0 }

3. **Weakest pre-condition** [1.5 pt].

   (a) Consider the loop below, with the given post-condition; x is of type integer:

   $$\textbf{while } \texttt{x>0} \textbf{ do } \{ \textbf{ assert } \texttt{even(x)} \; ; \; \texttt{x := x}-\texttt{2} \} \;\; \{ \texttt{even(x)} \}$$

   where $even(x)$ is a predicate that means that x is an even integer.

   Calculate the wlp of the loop above using the fix-point iteration.

   (b) Suppose we want to have a non-deterministic conditional statement in our programming language. We will denote it with the following multi-armed **if**, with $n \geq 1$:

   $$\begin{array}{lll} \textbf{if } g_1 & \rightarrow & S_1 \\ \quad ... & & \\ \quad g_n & \rightarrow & S_n \end{array}$$

   $g_1...g_n$ are 'guards'; these are boolean expressions. $S_1...S_n$ are statements.

   This is how the above statement works. Suppose we execute it on a state s. If there are multiple guards that evaluate to true on s, one will be selected non-deterministically, e.g. g_k, and the corresponding S_k is then executed.

   If no guard evaluates to true on s, the whole statement simply does a skip.

   Give a reasonable definition of the wlp of such a statement.

   (c) Give the definition of **repby** and propose a definition of the wlp of assignments that target a two dimensional array.

4. **Basic HOL** [1 pt].

   (a) In HOL, a tactic is a function of the type:

   $$goal \rightarrow (goal \; \texttt{list} \; \# \; proofFunction)$$

   where $goal = (\texttt{term list} \; \# \; \texttt{term})$ and $proofFunction = \texttt{thm list} \rightarrow \texttt{thm}$.

   The combinator `THEN` : $tactic \rightarrow tactic \rightarrow tactic$ applies two tactics one after another. That is, t_1 `THEN` t_2 applies t_1 on the given goal, then it applies t_2 on all the subgoals produced by t_1. Note that `THEN` produces a new tactic (you can see it in its type!) that internally does what is said in the previous sentence.

   Give the definition of `THEN`. You can give the definition in terms of a pseudo-code (it does not have to be in ML).

   (b) Show how the quantifiers $\forall$ and $\exists$ are defined in the primitive HOL. If you use operators other than function application, λ, $=$, $\Rightarrow$, and `T` define your operators as well.

5. **Hoare Logic** [0.5 pt, challenging].

   Consider again the language L_0 in the question No. 1. Propose how to calculate the wlp of the statements in L_0. Keep in mind that we have defined a post-condition in L_0 to be a pair of predicates Q_N, Q_E specifying the program final state when it ends normally, and when it ends exceptionally.

6. **HOL** [4 subquestions for total 4 pt, time: 48 hrs].

   From the PV website, you can download the file xxx.smx .... REMOVED in this sample version.