

Programming Languages

Qualifying Examination
Spring 2007

January 17, 2007

Problem 1 [Axiomatic Semantics – 30pts]

Consider the following code fragment

```
i := 1;
while (i <= n) do
  if (a[i-1] > a[i]) then
    a[i-1] := a[i-1]+a[i];
    a[i] := a[i-1] - a[i];
    a[i-1] := a[i-1] - a[i]
  endif;
  i := i+1
endwhile
```

Assume a precondition of the type

$$n \geq 1 \wedge \forall I(0 \leq I \leq n \rightarrow a[I] \geq 0)$$

- describe the total effect of the code on the array
- develop the appropriate post-condition (i.e., it should capture your intuitive description from the previous point) and loop invariant
- use Hoare's axioms and rules to prove partial correctness

Problem 2 [Denotational Semantics – 70pts]

We intend to model a basic language for parallel computing. The language is used to program SIMD architectures—where a single stream of instructions is executed operating on (possibly) different data.

Part A [40pts]

Let us start with the following syntax for programs:

```
<program>      ::= fork <number> : <command>
<command>     ::= <command> ; <command>
                |   [<expression>]<identifier> := <expression>
<expression> ::= <number>
                |   <identifier>
                |   own
                |   [<expression>]<identifier>
                |   <expression> + <expression>
```

Execution starts with a specification (`fork`) of how many threads should be concurrently running (`number`). All the threads are executing the same command. Each thread has a unique id (a non-negative integer); if there are n threads, then they are assigned the ids from 0 to $n-1$. The identifier called `own` allows a thread to retrieve its own id. All threads execute synchronously the same program. The variables used in the program are local to the threads; the notation

[`<threadid>`] `identifier`

means we want to access the variable `identifier` that belongs to the thread `threadid`; e.g., `[own]x` means that the thread wants to access its own variable named `x`, while `[5]x` means that the thread wants to access the variable `x` that belongs to thread 5.

For example, the program

```
fork 3 :
  [own]x := own;
  [own]x := [own+1]x + [own]x
```

starts 3 threads, each initializes its own x to its own id (i.e., thread 0 assigns 0 to its x , thread 1 assigns 1 to its x , and thread 2 assigns 2 to its x), and then each thread computes the sum between its own x and the x of the next thread (e.g., thread 0 will compute $0 + 1$ and store it in its own x).¹

Provide the denotational semantics for this language.

Hint: consider keeping multiple stores, and remember that each thread is numbered.

Part B [20pts]

Let us extend the previous language by adding a new command:

```
<command> ::= where <expression> do <command>
```

The intuition is that each thread executing this statement will compute the expression, and if this is true, then it will execute the command, otherwise the command is skipped.

E.g.,

```
where (own % 2 == 0) do
  [own] x = [own/2]x
```

only the threads with an even id will execute the statement.

Extend the denotational semantics from the previous point to include this new command.

Part C [10pts]

Let us introduce a new type of variable to the language; these are *global* variables, which are not replicated in the threads, instead each global variable is unique and accessible by all threads. The syntax is extended by a new command:

```
<command> ::= <identifier> := <expression>
```

and a new type of expression

```
<expression> ::= <identifier>
```

¹You can make any assumptions you want regarding indexing a variable of a non-existing thread.

For example, we can now write statements like

```
where (own == 0)
  y := 5;
[own]x := y
```

where the thread 0 initializes the global variable y to 5, and then each thread copies the value of y in its own variable x .

Extend the denotational semantics to include this new type of variables. *Hint:* there is only one copy of each global variable, and global variables should be kept separately from the local ones.