

## Design and Selection of Programming Languages

9 November 2005

### Exercise 9.1 — Partial Correctness

Consider the following program:

```
(i, j, s) := (0, 0, 0);  
while i ≠ n do  
  if i = j  
  then (i, j, s) := (i + 1, 0, s + 1)  
  else (j, s) := (j + 1, s + 2) fi  
od
```

- (a) Show partial correctness of this program with respect to the precondition  $\{\text{True}\}$  and the postcondition  $\{s = n^2 + 2j\}$ .
- (b) Which precondition is necessary for obtaining total correctness?

### Solution Hints

- (a) First the annotated program:

```
{True} ⇒ {0 = 02 + 2 · 0}  
(i, j, s) := (0, 0, 0);  
{s = i2 + 2j}  
while i ≠ n do  
  {s = i2 + 2j ∧ i ≠ n}  
  if i = j  
  then  
    {s = i2 + 2j ∧ i ≠ n ∧ i = j} ⇒ {s + 1 = (i + 1)2 + 2 · 0}  
    (i, j, s) := (i + 1, 0, s + 1)  
    {s = i2 + 2j}  
  else  
    {s = i2 + 2j ∧ i ≠ n ∧ i ≠ j} ⇒ {s + 2 = i2 + 2(j + 1)}  
    (j, s) := (j + 1, s + 2)  
    {s = i2 + 2j}  
  fi  
  {s = i2 + 2j}  
od  
{s = i2 + 2j ∧ i = n} ⇒ {s = n2 + 2j}
```

(b) *Detailed proof* —  $B$  is the body of the **while**-loop:

$$\begin{aligned}
& \{\text{True}\} (i, j, s) := (0, 0, 0) ; \mathbf{while} \ i \neq n \ \mathbf{do} \ B \ \mathbf{od} \ \{s = n^2 + 2j\} \\
& \Leftarrow \langle \text{right consequence} \rangle \\
& \quad \{\text{True}\} (i, j, s) := (0, 0, 0) ; \mathbf{while} \ i \neq n \ \mathbf{do} \ B \ \mathbf{od} \ \{s = i^2 + 2j \wedge i = n\} \\
& \quad \wedge (s = i^2 + 2j \wedge i = n \Rightarrow s = n^2 + 2j) \\
& \Leftarrow \langle \text{sequence, logic} \rangle \\
& \quad \{\text{True}\} (i, j, s) := (0, 0, 0) \ \{s = i^2 + 2j\} \\
& \quad \wedge \{s = i^2 + 2j\} \ \mathbf{while} \ i \neq n \ \mathbf{do} \ B \ \mathbf{od} \ \{s = i^2 + 2j \wedge i = n\} \\
& \quad \wedge \text{True} \\
& \Leftarrow \langle \text{left consequence, while-rule} \rangle \\
& \quad (\text{True} \Rightarrow 0 = 0^2 + 2 \cdot 0) \wedge \{0 = 0^2 + 2 \cdot 0\} (i, j, s) := (0, 0, 0) \ \{s = i^2 + 2j\} \\
& \quad \wedge \{s = i^2 + 2j \wedge i \neq n\} \ \mathbf{if} \ i = j \ \mathbf{then} \ (i, j, s) := (i + 1, 0, s + 1) \\
& \quad \quad \quad \mathbf{else} \ (j, s) := (j + 1, s + 2) \quad \mathbf{fi} \ \{s = i^2 + 2j\} \\
& \Leftarrow \langle \text{arithmetic, assignment, conditional} \rangle \\
& \quad \text{True} \wedge \text{True} \\
& \quad \wedge \{s = i^2 + 2j \wedge i \neq n \wedge i = j\} (i, j, s) := (i + 1, 0, s + 1) \ \{s = i^2 + 2j\} \\
& \quad \wedge \{s = i^2 + 2j \wedge i \neq n \wedge i \neq j\} (j, s) := (j + 1, s + 2) \ \{s = i^2 + 2j\} \\
& \Leftarrow \langle \text{left consequence, left consequence} \rangle \\
& \quad (s = i^2 + 2j \wedge i \neq n \wedge i = j \Rightarrow s + 1 = (i + 1)^2 + 2 \cdot 0) \\
& \quad \wedge \{s + 1 = (i + 1)^2 + 2 \cdot 0\} (i, j, s) := (i + 1, 0, s + 1) \ \{s = i^2 + 2j\} \\
& \quad \wedge (s = i^2 + 2j \wedge i \neq n \wedge i \neq j \Rightarrow s + 2 = i^2 + 2 \cdot (j + 1)) \\
& \quad \wedge \{s + 2 = i^2 + 2 \cdot (j + 1)\} (j, s) := (j + 1, s + 2) \ \{s = i^2 + 2j\} \\
& \Leftarrow \langle \text{arithmetic, assignment, arithmetic, assignment} \rangle \\
& \quad \text{True} \wedge \text{True} \wedge \text{True} \wedge \text{True}
\end{aligned}$$

(c)  $\{n \geq 0\}$ , since  $i \geq 0$ .

### Exercise 9.2 — Partial Correctness Proof — 24% of Final 2003

Consider the following program fragment in a language providing a Java-like printing statement, given an  $n$ -element Java-like array  $a$ :

```

(i, m) := (0, 0) ;
while i ≠ n do
    (i, m) := (i + 1, (m * i + a[i]) / (i + 1)) ;
    println(i + " " + m)
od

```

- (a) What is the output of this program for  $n = 5$  and  $a$  containing the sequence 4, 2, 9, 1, 4?
- (b) What does this program do? (Short verbal description.)
- (c) For this program **without the println** statement, **prove partial correctness** with respect to the **precondition**  $\{n \geq 0\}$  and the **postcondition**  $\{m \cdot n = \sum_{j=0}^{n-1} a[j]\}$ .

**Important:** *Justify* implications you use, and pay attention to **definedness of operations!**

### Solution Hints

$\{n \geq 0\} (i, m) := (0, 0) ; \mathbf{while} \ i \neq n \ \mathbf{do} \ (i, m) := (i + 1, (m * i + a[i]) / (i + 1)) \ \mathbf{od} \ \{m \cdot n = \sum_{j=0}^{n-1} a[j]\}$

(a) 

1	4.00000
2	3.00000
3	5.00000
4	4.00000
5	4.00000

(b) Incremental calculation of average

$\{n \geq 0\} P \{m \cdot n = \sum_{j=0}^{n-1} a[j]\}$

$\Leftarrow \langle (\text{right consequence}) \rangle$

$\{n \geq 0\} \text{Init} ; W \{m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n \wedge i = n\}$

$\wedge (m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n \wedge i = n \Rightarrow m \cdot n = \sum_{j=0}^{n-1} a[j] \wedge i \geq 0 \wedge i \leq n)$

$\Leftarrow \langle (\text{sequence, logic}) \rangle$

$\{n \geq 0\} (i, m) := (0, 0) \{m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n\}$

$\wedge \{m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n\} \mathbf{while} \ i \neq n \ \mathbf{do} \ B \ \mathbf{od} \ \{m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n \wedge i = n\}$

$\wedge \text{True}$

$\Leftarrow \langle (\text{left consequence, while}) \rangle$

$(n \geq 0 \Rightarrow 0 \cdot 0 = \sum_{j=0}^{0-1} a[j]) \wedge 0 \geq 0 \wedge 0 \leq n$

$\wedge \{m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n \wedge i \neq n\} (i, m) := (i + 1, (m * i + a[i]) / (i + 1)) \{m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n\}$

$\Leftarrow \langle (\text{logic, assignment, left consequence}) \rangle$

$\text{True} \wedge \text{True}$

$\wedge (m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n \wedge i \neq n \Rightarrow ((m * i + a[i]) / (i + 1)) \cdot (i + 1) = \sum_{j=0}^{(i+1)-1} a[j] \wedge i + 1 \geq 0 \wedge i + 1 \leq n)$

$\wedge \{((m * i + a[i]) / (i + 1)) \cdot (i + 1) = \sum_{j=0}^{(i+1)-1} a[j] \wedge i + 1 \geq 0 \wedge i + 1 \leq n\} (i, m) := (i + 1, (m * i + a[i]) / (i + 1))$

$\{m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n\}$

$\Leftarrow \langle (\text{logic and arithmetic, assignment}) \rangle$

$(m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n \wedge i \neq n \Rightarrow ((m * i + a[i]) / (i + 1)) \cdot (i + 1) = m \cdot i + a[i])$

$\wedge (m \cdot i = \sum_{j=0}^{i-1} a[j] \wedge i \geq 0 \wedge i \leq n \wedge i \neq n \Rightarrow m \cdot i + a[i] = \sum_{j=0}^i a[j])$

$\wedge (i \geq 0 \Rightarrow i + 1 \geq 0)$

$\wedge (i \leq n \wedge i \neq n \Rightarrow i + 1 \leq n)$

$\wedge \text{True}$

$\Leftarrow \langle (\text{logic and arithmetic}) \rangle$

$\text{True}$

### Exercise 9.3— Semantics of Exceptions

We consider a simple imperative programming language with exceptions, with the following **abstract syntax**:

$$\begin{array}{ll}
 \text{Stmt} ::= & \mathbf{skip} \\
 & | \text{Id} := \text{Expr} \\
 & | \text{Stmt} ; \text{Stmt} \\
 & | \mathbf{if} \text{Expr} \mathbf{then} \text{Stmt} \mathbf{else} \text{Stmt} \\
 & | \mathbf{while} \text{Expr} \mathbf{do} \text{Stmt} \\
 & | \mathbf{throw} \text{Expr} \\
 & | \mathbf{try} \text{Stmt} \mathbf{catch}(\text{Id}) \text{Stmt} \\
 \text{Expr} ::= & \text{Id} \\
 & | \text{Num} \\
 & | \text{Bool} \\
 & | \text{Expr Op Expr} \\
 \text{Op} ::= & + \mid - \mid * \mid / \mid \leq \mid \geq \mid < \mid >
 \end{array}$$

(a) Define Haskell datatypes for the abstract syntax of this language.

We still have the following basic semantic domains:

$$\begin{array}{lll}
 \text{Val} & = \text{Bool} + \text{Num} & \text{values} \\
 \text{Store} & = \text{Id} \mapsto \text{Val} & \text{(simple) stores}
 \end{array}$$

We denote the elements of *Val* by True, False, 0, 1, 2, ...

(b) For each of the following, indicate whether it denotes an element of the set *Store*, i.e., a possible *Store* (the notation “ $a \mapsto b$ ” means exactly the pair “ $(a, b)$ ”):

1. True:  False:   $\{b \mapsto \{\text{True}\}, n \mapsto 0\}$
2. True:  False:   $\{k \mapsto 7, b \mapsto 42, m \mapsto 1001, n \mapsto 1, b \mapsto \text{False}\}$
3. True:  False:   $\{b \mapsto 42, k \mapsto \text{True}\}$
4. True:  False:   $\{k \mapsto 5, b \mapsto \text{True}, s \mapsto \mathbf{skip}\}$
5. True:  False:   $\{\} \times \text{Val}$
6. True:  False:   $\{n\} \times \{0\}$
7. True:  False:   $\{n\} \times \{0, 1, 2\}$
8. True:  False:   $\{k, m, n\} \times \{0\}$

From an operational point of view, assuming that the expression  $e$  evaluates to the number  $k$ , the statement “**throw**  $e$ ” raises exception  $k$ .

We allow **only numbers** as exceptions.

If a statement raising an exception is not enclosed by any “**try \_ catch**” construct, then this exception immediately leads to program termination with an *uncaught exception*.

If there is an enclosing “**try \_ catch**” construct, then this is of the shape “**try \_ catch**(  $i$  )  $s_2$ ” for some identifier  $i$  and a statement  $s_2$ . In that case, execution proceeds immediately to  $s_2$  in an environment where the identifier  $i$  is bound to the numerical value of the caught exception.

(c) Write down the *Store* that the statement  $s_2$  executes from when control arrives at  $s_2$  in the following program:

$$k := 100 ; \mathbf{try} \ q := 42 ; \mathbf{throw} \ 14 ; s := q + 1 \mathbf{catch}(n) \ s_2$$

#### Solution Hints

The store is:  $\{k \mapsto 100, q \mapsto 42, n \mapsto 14\}$

The statement semantics needs to accommodate the possibility of locally uncaught exceptions. Therefore, the lecture introduced an additional assertion schema for operational semantics:

$\sigma_1(s) \stackrel{!}{\Rightarrow} (\sigma_2, x)$  — execution of statement  $s$  starting in state  $\sigma_1$  can terminate in state  $\sigma_2$  **raising exception**  $x$

The lecture also showed that in the Haskell interpreter, statement interpretation **with exceptions** can be implemented

via:

$interpStmtExc :: Statement \rightarrow State1 \rightarrow Maybe (Either State1 (State1, Exc))$

This function corresponds to the operational semantics in the following way:

$$\begin{array}{lll} \sigma_1(s) \Rightarrow \sigma_2 & \mathbf{iff} & interpStmt\ s\ \sigma_1 = Just\ (Left\ \sigma_2) \\ \sigma_1(s) \overset{!}{\Rightarrow} (\sigma_2, x) & \mathbf{iff} & interpStmt\ s\ \sigma_1 = Just\ (Right\ (\sigma_2, x)) \\ \neg \exists \sigma_2, x \bullet \sigma_1(s) \Rightarrow \sigma_2 \vee & & \\ \sigma_1(s) \overset{!}{\Rightarrow} (\sigma_2, x) & \mathbf{iff} & interpStmt\ s\ \sigma_1 = Nothing \end{array}$$

- (d) Extend operational semantics of expression evaluation to allow for the possibility that expression evaluation raises exceptions. (In particular, division by zero should be defined to raise exception 24.)
- (e) Adapt also the definition of the Haskell interpreter functions accordingly.