Design and Selection of Programming Languages

4 November 2005

Exercise 8.1 — Using Operational Semantics to Prove Incorrectness

The following Hoare triples do not hold.
For each of these Hoare triples, present a derivation in the operational semantics that proves a counterexample to the statement.

(a) \{x \geq -5\} z := 5 - x \{z \leq 11 \land x \geq -3\}
(b) \{x \geq -5\} z := 5 - x ; x := z + 2 \{z \leq 11 \land x \geq -3\}

“Proving a counterexample” for the Hoare triple
\(\{pre\}Prog\{post\}\)

means to derive an assertion
\(\sigma_1(Prog) \Rightarrow \sigma_2\)

involving
– a state \(\sigma_1\) for which \(pre\) holds, and
– a state \(\sigma_2\) for which \(post\) does not hold.

Solution Hints

(a) Using operational semantics, we can prove a counterexample:
\[
\frac{x \mapsto -5}(5) \Rightarrow 5 \quad \frac{x \mapsto -5}(x) \Rightarrow -5
\]
\[
\frac{x \mapsto -5}(5 - x) \Rightarrow 10
\]
\[
\frac{x \mapsto -5}(z := 5 - x) \Rightarrow \{x \mapsto -5, z \mapsto 10\}
\]

This last state clearly does not satisfy \(\{z \leq 11 \land x \geq -3\}\).

(b) For \(\{x \geq -5\} z := 5 - x ; x := x + 2 \{z \leq 11 \land x \geq -3\}\), we again use operational semantics (expression evaluation not shown) to prove a counterexample:
\[
\frac{x \mapsto 20}(5 - x) \Rightarrow -15
\]
\[
\frac{x \mapsto 20}(z := 5 - x) \Rightarrow \{x \mapsto 20, z \mapsto -15\}
\]
\[
\frac{x \mapsto 20, z \mapsto -15}(x := x + z) \Rightarrow \{x \mapsto -13, z \mapsto -15\}
\]

Although \(\{x \mapsto 20\}\) satisfies the precondition \(x \geq -5\), the final state \(\{x \mapsto -13, z \mapsto -15\}\) does not satisfy the postcondition \(\{z \leq 11 \land x \geq -3\}\).

Exercise 8.2 — Semantics of Exceptions
We consider a simple imperative programming language with exceptions, with the following **abstract syntax**:

\[
Stmt \ ::= \ \text{skip} \mid \text{Id} := \text{Expr} \mid Stmt ; Stmt \mid \text{if} \ \text{Expr} \ \text{then} \ Stmt \ \text{else} \ Stmt \mid \text{while} \ \text{Expr} \ \text{do} \ Stmt \mid \text{throw} \ \text{Expr} \mid \text{try} \ Stmt \ \text{catch( Id )} \ Stmt
\]

\[
Expr \ ::= \ \text{Id} \mid \text{Num} \mid \text{Bool} \mid \text{Expr} \ \text{Op} \ \text{Expr}
\]

\[
\text{Op} \ ::= + | - | \ast | / | \leq | \geq | < | >
\]

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

We still have the following basic semantic domains:

\[
\text{Val} = \text{Bool} + \text{Num} \quad \text{values}
\]

\[
\text{Store} = \text{Id} \rightarrow \text{Val} \quad \text{(simple) stores}
\]

We denote the elements of \( \text{Val} \) by True, False, 0, 1, 2, …

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

1. True: False: \{b \rightarrow \{True\}, n \rightarrow 0\}
2. True: False: \{k \rightarrow 7, b \rightarrow 42, m \rightarrow 1001, n \rightarrow 1, b \rightarrow \text{False}\}
3. True: False: \{b \rightarrow 42, k \rightarrow \text{True}\}
4. True: False: \{k \rightarrow 5, b \rightarrow \text{True}, s \rightarrow \text{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 \( \text{Store} \) that the statement \( s_2 \) executes from when control arrives at \( s_2 \) in the following program:

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

**Solution Hints**

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