McMaster University
Department of Computing and Software
Dr. W. Kahl

SFWR ENG 3E03
Exercise Sheet 8

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) $\{x \geq -5\}\ z := 5 - x\ \{z \leq 11 \wedge x \geq -3\}$

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

"Proving a counterexample" for the Hoare triple

$$\{pre\}Prog\{post\}$$

means to derive an assertion

$$\sigma_1(Prog) \Rightarrow \sigma_2$$

involving
– a state σ_1 for which *pre* **holds**, and
– a state σ_2 for which *post* **does not hold**.

Solution Hints

(a) Using operational semantics, we can prove a counterexample:

$$\dfrac{\dfrac{\{x \mapsto -5\}(5) \Rightarrow 5 \qquad \{x \mapsto -5\}(x) \Rightarrow -5}{\{x \mapsto -5\}(5 - x) \Rightarrow 10}}{\{x \mapsto -5\}(z := 5 - x) \Rightarrow \{x \mapsto -5, z \mapsto 10\}}$$

This last state clearly does not satisfy $\{z \leq 11 \wedge x \geq -3\}$

(b) For $\{x \geq -5\}\ z := 5 - x\ ;\ x := x + 2\ \{z \leq 11 \wedge x \geq -3\}$, we again use operational semantics (expression evaluation not shown) to prove a counterexample:

$$\dfrac{\dfrac{\{x \mapsto 20\}(5 - x) \Rightarrow -15}{\{x \mapsto 20\}(z := 5 - x) \Rightarrow \{x \mapsto 20, z \mapsto -15\}} \quad \dfrac{\{x \mapsto 20, z \mapsto -15\}(z + 2) \Rightarrow -13}{\{x \mapsto 20, z \mapsto -15\}(x := x + z) \Rightarrow \{x \mapsto -13, z \mapsto -15\}}}{\{x \mapsto 20\}(z := 5 - x\ ;\ x := z + 2) \Rightarrow \{x \mapsto -13, z \mapsto -15\}}$$

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

Exercise 8.2 — Semantics of Exceptions

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

Stmt ::=	**skip**	*Expr* ::=	*Id*
	| *Id* **:=** *Expr*		| *Num*
	| *Stmt* **;** *Stmt*		| *Bool*
	| **if** *Expr* **then** *Stmt* **else** *Stmt*		| *Expr Op Expr*
	| **while** *Expr* **do** *Stmt*		
	| **throw** *Expr*	*Op* ::=	+ | − | * | / | ≤ | ≥ | < | >
	| **try** *Stmt* **catch(** *Id* **)** *Stmt*		

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

We still have the following basic semantic domains:

Val	$= Bool + Num$	values
$Store$	$= Id \rightharpoonup Val$	(simple) stores

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: ■ $\{b \mapsto \{\mathsf{True}\}, n \mapsto 0\}$
2. True: ☐ False: ■ $\{k \mapsto 7, b \mapsto 42, m \mapsto 1001, n \mapsto 1, b \mapsto \mathsf{False}\}$
3. True: ■ False: ☐ $\{b \mapsto 42, k \mapsto \mathsf{True}\}$
4. True: ☐ False: ■ $\{k \mapsto 5, b \mapsto \mathsf{True}, s \mapsto \textbf{skip}\}$
5. True: ■ False: ☐ $\{\} \times Val$
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:

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

Solution Hints

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