Semantics of Exceptions (mostly from Final 2002)

We consider again the simple imperative programming language with exceptions from Exercise 7.2.

The abstract syntax of this programming language is the following:

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

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

\[
Op ::= + | - | ∗ | / | ≤ | ≥ | < | >
\]

We still have the following basic semantic domains:

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

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

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

(a) 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: ☐ \( \{k \mapsto 5, b \mapsto \text{True}, s \mapsto \text{skip}\} \)
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.

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

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

Solution Hints
The semicolon before the “catch” that was originally on the assignment needs to be removed!
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, it produces a state transition function that returns either just a Store or a Store together with an exception number, i.e., statement semantics $\llbracket$ $\rrbracket_S$ has the following type (the state transition function may be partial to accommodate non-termination):

$$\llbracket$ $\rrbracket_S : \text{Stmt} \rightarrow (\text{Store} \rightarrow (\text{Store} + (\text{Store} \times \text{Num})))$$

(c) Define $\llbracket$skip$\rrbracket_S$.

Solution Hints
We use Left and Right for the injection into the result type Store + (Store × Num) of statement semantics.

$\llbracket$skip$\rrbracket_S = \text{Left}$

(d) Define $\llbracket s_1 ; s_2 \rrbracket_S$ for arbitrary statements $s_1, s_2 : \text{Stmt}$.

Solution Hints

$$\llbracket s_1 ; s_2 \rrbracket_S(s) = \begin{cases} \llbracket s_2 \rrbracket_S(t) & \text{if } \llbracket s_1 \rrbracket_S(s) = \text{Left } t \\ \text{Right}(t, e) & \text{if } \llbracket s_1 \rrbracket_S(s) = \text{Right } (t, e) \end{cases}$$

(e) Define $\llbracket \text{try } s_1 \text{ catch( } i \text{ ) } s_2 \rrbracket_S$ for arbitrary statements $s_1, s_2 : \text{Stmt}$ and an arbitrary identifier $i : \text{Id}$. 
Solution Hints

\[
\begin{aligned}
[\text{try } s_1 \text{ catch(} i \text{) } s_2]_S (s) &= \begin{cases}
  t & \text{if } [s_1]_S (s) = \text{Left } t \\
  [s_2]_S (t \oplus \{i \mapsto e\}) & \text{if } [s_1]_S (s) = \text{Right } (t, e) \\
  \bot & \text{if } s \not\in \text{dom } [s_1]_S
\end{cases}
\end{aligned}
\]

Since we already have exceptions in our language, we want primitive operations to raise exceptions if they cannot execute properly, and \textbf{never} produce \( \bot \).

(f) Propose and explain a type for expression semantics \([::]_E\):

Solution Hints

An expression either evaluates to a Value, or it raises an exception, which is a Number. All expression evaluation terminates, so expression semantics produces \textbf{total} functions.

\[
\text{Expr} \rightarrow (\text{Store} \rightarrow (\text{Val} + \text{Num}))
\]

(g) Define \( [\text{throw } e]_S \) for an arbitrary expression \( e : \text{Expr} \).

Solution Hints

\[
[\text{throw } e]_S (s) = \begin{cases}
  \text{Right } (s, \text{val}) & \text{if } [e]_E (s) = \text{Left } \text{val} \\
  \text{Right } (s, \text{exc}) & \text{if } [e]_E (s) = \text{Right } \text{exc}
\end{cases}
\]

(h) Define \( [v := e]_S \) for an arbitrary identifier \( v : \text{Id} \) and an arbitrary expression \( e : \text{Expr} \).

Solution Hints

\[
[v := e]_S (s) = \begin{cases}
  \text{Left } (s \oplus \{v \mapsto \text{val}\}) & \text{if } [e]_E (s) = \text{Left } \text{val} \\
  \text{Right } (s, \text{exc}) & \text{if } [e]_E (s) = \text{Right } \text{exc}
\end{cases}
\]

(i) Define \( [\text{if } b \text{ then } s_1 \text{ else } s_2]_S \) for an arbitrary expression \( b : \text{Expr} \) and arbitrary statements \( s_1, s_2 : \text{Stmt} \).

Solution Hints

\[
[\text{if } b \text{ then } s_1 \text{ else } s_2]_S (s) = \begin{cases}
  [s_1]_S (s) & \text{if } [b]_E (s) = \text{Left } \text{true} \\
  [s_2]_S (s) & \text{if } [b]_E (s) = \text{Left } \text{false} \\
  \text{Right } (s, \text{exc}_{IB}) & \text{if } [b]_E (s) = \text{Left } n \land n \in \text{Num} \\
  \text{Right } (s, \text{exc}) & \text{if } [b]_E (s) = \text{Right } \text{exc}
\end{cases}
\]

\( \text{exc}_{IB} \) is the exception number for “Boolean value expected”.