Homework 2 (Written): Semantics

17-355/17-665/17-819: Program Analysis
Jonathan Aldrich, Rohan Padhye

Due: Thursday, February 18, 2021 (11:59 PM) 100 points total

Assignment Objectives:

- Precisely specify language features using both small- and big-step semantic rules.
- Carefully consider the benefits of small- versus big-step rules for specifying language features.
- Practice and demonstrate the use of induction on the structure of derivations to prove conjectures about the semantic rules for WHILE.

Handin Instructions *(5 points)*. Please submit your assignment through the Gradescope link on Canvas (supports PDF and jpgs/photos) by the due date. When submitting, please indicate which pages of the PDF correspond to each homework question. Putting page breaks between questions makes this simpler. Typesetting is not required, but is strongly suggested; you may submit photos of handwritten answers, but they must be clear and legible. Feel free to typeset your proofs using your favorite LaTeX package. If you do not have one, you may be interested in `mathpartir`, which you can find on the website. To see how to write some inference rules, compile the example `mathpartir.tex` with, e.g., `pdflatex`. To use it for your assignment, include `mathpartir.sty` in your tex file (i.e., (`\usepackage {mathpartir}`). Alternatively, you can also modify `mathpartir.tex` directly.

Question 1, `let` Statement, *(20 points).* Consider the WHILE language (not WHILE3ADDR!) extended with a new statement "`let` x = e `in` s". The informal semantics of this construct is that the expression e is evaluated; a new local variable x is created with lexical scope c; and x is initialized with the result of evaluating e. Then the statement s is evaluated in c. For exposition/convenience, we also extend WHILE with statement "`print` e" which evaluates the e and "displays the result" in some un-modeled manner (it is otherwise similar to `skip`). We therefore expect the following code to display "3 2 1 5" (the curly braces are syntactic sugar):

$$
\begin{array}{l}
x := 1; \\
y := 2; \\
\text{let } x = 3 \text{ in } \{ \\
\quad \text{print } x; \\
\quad \text{print } y; \\
\quad x := 4; \\
\quad y := 5; \\
\}; \\
\text{print } x;\ \text{print } y
\end{array}
$$

Part (a): Extend the big-step operational semantics judgment $\langle E, s\rangle \Downarrow E'$ with one new rule for dealing with the *let* statement. Pay careful attention to the scope of the newly declared variable and to changes to other variables.

Part (b): Extend the small-step operational semantics judgment $\langle E, s\rangle \rightarrow \langle E', s'\rangle$ to account for the *let* statement.

Question 2, Exceptional semantics, *(25 points).* One way to handle error situations (like divide-by-zero, mentioned in class) generally is to explicitly introduce error handling into the language. We thus add to WHILE integer-valued *exceptions* (or *run-time errors*), as in Java, ML or C#. We introduce a new sort T to represent command terminations, which can either be normal or exceptional (with an exception value $n \in \mathbb{Z}$):

$$
\begin{array}{lcll}
T & ::= & E & \text{"normal termination"} \\
  & | & E \ \texttt{exc}\ n & \text{"exceptional termination"}
\end{array}
$$

We use t to range over T. We then redefine our big-step operational semantics judgment:

$$\langle E, S\rangle \Downarrow T$$

The interpretation of

$$\langle E, S\rangle \Downarrow E' \ \texttt{exc}\ n$$

is that statement S terminated abruptly by throwing an exception with value $n \in \mathbb{Z}$ at a point in S's execution when the state was E'. We only model one type of exception, but every exception has an integer "argument" n (or "payload" or "value") that is set when the exception is thrown and available when the exception is caught.

Our previous statement rules must now be updated to account for exceptions, as in:

$$
\frac{\langle E, S_1\rangle \Downarrow E' \ \texttt{exc}\ n}{\langle E, S_1; S_2\rangle \Downarrow E' \ \texttt{exc}\ n}\ \textsf{seq1}
\qquad
\frac{\langle E, S_1\rangle \Downarrow E' \quad \langle E', S_2\rangle \Downarrow t}{\langle E, S_1; S_2\rangle \Downarrow t}\ \textsf{seq2}
$$

We also introduce two new statements:

- `throw` e: raise an exception with argument e.
- `try` S_1 `catch` x S_2: execute S_1. If S_1 terminates normally (i.e., without an uncaught exception), the `try` statement also terminates normally. If S_1 raises an exception with value e, the variable $x \in L$ is assigned the value e, and then S_2 is executed.

These are intended to have the standard exception semantics from languages like Java, C#, or OCaml *except* that the catch block merely assigns to x, it does not bind it to a local scope. So, catch does not behave like a let (simplifying the specification of the construct, if not its actual use!). We thus expect:

```
x := 0 ;
{ try
    if x <= 5 then throw 33 else throw 55
  catch x
    print x } ;
while true do {
  x := x - 15 ;
  print x ;
  if x <= 0 then throw (x*2) else skip
}
```

to output "`33 18 3 -12`" and then terminate with an uncaught exception with value `-24`.

> Give the big step operational semantics inference rules (using our new judgment) for the two new statements listed above.

Question 3, Big or small?, *(15 points).* Argue for or against the claim that it would be more natural to describe "WHILE with exceptions" using small-step semantics. You may use "simpler" or "more elegant" instead of "more natural" if you prefer. Do not exceed two paragraphs (one should suffice). Both the ideas and the clarity with which they are expressed (i.e., prose) matter.

Question 4, Induction, *(35 points).* In the lecture notes, we observed that we can use induction on the structure of expressions to prove that the big- and small-step semantics for `Aexp` obtain equivalent results. For the syntactic categories in WHILE, we can express this claim formally as:

$$\begin{array}{llll}
\forall a \in \texttt{AExp}. & \forall E.\, \forall n \in \mathbb{Z}. & \langle E, a\rangle \rightarrow_a^* n & \Leftrightarrow \quad \langle E, a\rangle \Downarrow n \\
\forall P \in \texttt{Bexp}. & \forall E.\, \forall b \in \{\texttt{true}, \texttt{false}\}. & \langle E, P\rangle \rightarrow_b^* b & \Leftrightarrow \quad \langle E, P\rangle \Downarrow b \\
\forall S \in \texttt{Stmt}. & \forall E, E' \in \texttt{Var} \rightarrow \mathbb{Z}. & \langle E, S\rangle \rightarrow^* \langle E', \texttt{skip}\rangle & \Leftrightarrow \quad \langle E, S\rangle \Downarrow E'
\end{array}$$

> Prove by induction on the structure of derivations that, if a statement terminates, the big- and small-step semantics for WHILE will obtain equivalent results (equation (3) above). You may assume (1) and (2) have been proven. Show (a) the base case(s), and (b) the inductive case for `let` (using the semantics you developed in question (1)) and (c) two more representative inductive cases. If you cannot show this for `let`, choose some other third inductive case for partial credit. Also, make sure your proof is sufficiently detailed.

If needed, here are the rules which define $\langle E, S\rangle \rightarrow^* \langle E', S'\rangle$ (the reflexive, transitive, closure of $\langle E, S\rangle \rightarrow \langle E', S'\rangle$):

$$\frac{}{\langle E, S\rangle \rightarrow^* \langle E, S\rangle}\ R \qquad\qquad \frac{\langle E, S_1\rangle \rightarrow \langle E', S_2\rangle \quad \langle E', S_2\rangle \rightarrow^* \langle E'', S_3\rangle}{\langle E, S_1\rangle \rightarrow^* \langle E'', S_3\rangle}\ T$$

If needed, you may assume the following Lemmas hold:

Lemma 1.

Transitivity of $\langle E, S\rangle \rightarrow^* \langle E', S'\rangle$

$$\frac{\langle E, S_1\rangle \rightarrow^* \langle E', S_2\rangle \quad \langle E', S_2\rangle \rightarrow^* \langle E'', S_3\rangle}{\langle E, S_1\rangle \rightarrow^* \langle E'', S_3\rangle}$$

Lemma 2.

a) *Small-step assignment congruence of* $\langle E, S\rangle \rightarrow^* \langle E', S'\rangle$

$$\frac{\langle E, a\rangle \rightarrow_a^* a'}{\langle E, x := a\rangle \rightarrow^* \langle E, x := a'\rangle}$$

b) *Small-step sequence congruence of* $\langle E, S\rangle \rightarrow^* \langle E', S'\rangle$

$$\frac{\langle E, S_1\rangle \rightarrow^* \langle E', S_1'\rangle}{\langle E, S_1; S_2\rangle \rightarrow^* \langle E', S_1'; S_2\rangle}$$

c) *Small-step if congruence of* $\langle E, S\rangle \rightarrow^* \langle E', S'\rangle$

$$\frac{\langle E, P\rangle \rightarrow_b^* \ P'}{\langle E, \texttt{if } P \texttt{ then } S_1 \texttt{ else } S_2\rangle \rightarrow^* \langle E, \texttt{if } P' \texttt{ then } S_1 \texttt{ else } S_2\rangle}$$

d) *Small-step let congruence rule(s) of* $\langle E, S\rangle \rightarrow^* \langle E', S'\rangle$ *(applies your answer to Question 1(b))*