Assignment 2

Joshua Dunfield

due: Tuesday, 5 February 2019

Name: **Estimated time spent:**

Submit your modified `a2.rkt` by email (joshuad@cs.queensu.ca) as `a2-`*`yourname`*`.rkt`.

Please fill in the "Estimated time spent" (above). Your answer will not change your mark, but helps me to design reasonable assignments.

1 Language Extension

Consider the following language, defined by a grammar and a big-step evaluation judgment. The big-step evaluation given is incomplete, in the informal sense that it has no rules saying how to evaluate `(Abs` e`)`.

$$
\begin{array}{llll}
\text{integers} & n & & \\
\text{values} & v & ::= & n \\
\text{expressions} & e & ::= & n \\
 & & | & (\texttt{+}\ e_1\ e_2) \\
 & & | & (\texttt{Abs}\ e)
\end{array}
$$

$\boxed{e \Downarrow v}$ expression e evaluates to value v

$$
\dfrac{}{n \Downarrow n}\ \text{eval-const}
\qquad\qquad
\dfrac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{(\texttt{+}\ e_1\ e_2) \Downarrow n_1 + n_2}\ \text{eval-add}
$$

Question 1(a). Roughly following the structure of eval-add, design a rule "eval-abs" such that `(Abs` e`)` will compute the absolute value of e. Similar to how we used the standard notation for addition, $n_1 + n_2$, in eval-add, you may use the notation $|n|$ for the absolute value of n.

$$
\dfrac{\qquad\qquad\qquad\qquad\qquad\qquad}{(\texttt{Abs}\ e) \Downarrow \qquad\qquad}\ \text{eval-abs}
$$

Question 1(b). Extend the Racket code a2.rkt (which is nearly identical to the expr1.rkt file that I showed in lecture) to support the construct `(Abs` e`)`:

(i) Add a "variant" (also called a "constructor") for `(Abs` e`)` to the `define-type` declaration.

This will cause Racket to yell at you, because EXPR will then have three variants instead of two, which makes the `type-case` in `big-step` become incomplete. The simplest fix (until you extend `big-step`, below) is to add an `else` branch to the `type-case` that uses `error`.

(ii) Extend the procedure `parse` so that it accepts S-expressions that match the production `(Abs` e`)` in our grammar, and returns your `Abs` variant. Write at least two test cases for `parse` that involve absolute value.

(iii) Extend the procedure `big-step` to support absolute value, following your rule eval-abs. Note that the Racket standard library has a procedure `abs`. Write at least three test cases for `big-step` that involve absolute value.

2 Proof techniques

These questions are not about complete proofs. In some of the questions, the conjecture is not even true, or you have not been given enough information to do a complete proof. Instead, they ask you to make progress on several different proofs by using a specific proof technique.

In all of these questions, the grammar of expressions is the *extended* grammar (Section **??**), and the system of rules deriving $e \Downarrow v$ includes the *three* rules eval-const, eval-add, eval-abs.

Question 2(a). Using the extended grammar of expressions (Section **??**), list the cases produced by *case analysis on* e. The cases must correspond to the grammar. Do not attempt to complete the cases to show $e' = e''$.

Conjecture.
For all expressions e*,* e' *and* e''*,*
if $e \mapsto e'$ *and* $e \mapsto e''$
then $e' = e''$.

Proof. Consider cases of e.

- Case

□

Question 2(b). In this question, your goal is to derive

$$(\texttt{+}\ 0\ (\texttt{+}\ e_{21}\ e_{22})) \Downarrow 0$$

The following are given. Use equations (and the fact that $0 + 0 = 0$) and apply the rules eval-const, eval-add to derive the goal.

$e_{21} \Downarrow n_{21}$	Given
$e_{22} \Downarrow n_{22}$	Given
$n_{21} = 0$	Given
$n_{22} = 0$	Given

Question 2(c). In this question, use *inversion*: write down all the facts given by inverting on rule eval-abs. (I can't give you a specific goal because what you get depends on your rule, eval-abs.)

$(\texttt{Abs}\ e_1) \Downarrow n_1$ Given

Question 2(d). This question uses notation you have never seen, but it can still be answered.

Conjecture.
For all $\mathcal{C}$, M_1, M_2 and $\mathcal{D}_1$ such that $\mathcal{D}_1$ derives $M_1 \mapsto_R M_2$,
there exists $\mathcal{D}_2$ such that $\mathcal{D}_2$ derives $\mathcal{C}[M_1] \mapsto_R \mathcal{C}[M_2]$.

- Suppose that I suggest you use structural induction on $\mathcal{D}_1$. Write the appropriate induction hypothesis:

- Suppose that I suggest you use structural induction on M_1. Write the appropriate induction hypothesis (using the subexpression ordering $\prec$ for M_1):

3 Typing

$$\text{types} \quad A ::= \mathsf{int}$$

$\boxed{e : A}$ expression e has type A

$$\frac{}{n : \mathsf{int}}\ \text{type-const} \qquad \frac{e : \mathsf{int}}{(\mathtt{Abs}\ e) : \mathsf{int}}\ \text{type-abs} \qquad \frac{e_1 : \mathsf{int} \qquad e_2 : \mathsf{int}}{(+\ e_1\ e_2) : \mathsf{int}}\ \text{type-add}$$

This is not a terribly interesting type system: every possible expression has the same type, int. Prove the following conjecture:

Conjecture 3.1.
For all expressions e*,*
it is the case that $e : \mathsf{int}$.

Proof. By structural induction on e. [The only thing given is e; we are trying to construct the derivation of $e : \mathsf{int}$, but it doesn't exist yet. So we have to induct on e.]

Induction hypothesis: For all expressions e' such that $e' \prec e$, it is the case that $e' : \mathsf{int}$.

□