## 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 \((\text{Abs } e)\).

\[
\begin{align*}
\text{integers} & \quad n & \quad \text{values} & \quad v \equiv n \\
\text{expressions} & \quad e \equiv n & & | (+ \, e_1 \, e_2) \\
& & & | (\text{Abs } e) \\
\end{align*}
\]

\(e \Downarrow v\) expression \(e\) evaluates to value \(v\)

\[
\begin{align*}
\frac{n \Downarrow n}{\text{eval-const}} & \quad \frac{e_1 \Downarrow n_1 \quad e_2 \Downarrow n_2}{(+ \, e_1 \, e_2) \Downarrow n_1 + n_2} & \quad \frac{e \Downarrow n}{\text{eval-abs}} \\
\end{align*}
\]

**Question 1(a).** Roughly following the structure of eval-add, design a rule “eval-abs” such that \((\text{Abs } e)\) 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 \(n\).

\[
\frac{e \Downarrow n}{(\text{Abs } e) \Downarrow |n|}
\]

**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 \((\text{Abs } e)\):

(i) Add a “variant” (also called a “constructor”) for \((\text{Abs } e)\) 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 \((\text{Abs } e)\) 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.
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 1), 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 1), 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 $e \rightarrow e'$ and $e \rightarrow e''$

then $e' = e''$.

**Proof.** Consider cases of $e$.

- Case $e = n$
- Case $e = (+ e_1 e_2)$
- Case $e = (Abs e_1)$

[Writing $e = (Abs e)$ doesn’t make sense: it’s like saying that $x + 1 = x$. The name $e$ can’t be used for two different things at the same time. We need to rename the $e$ from the grammar so it doesn’t clash with the $e$ mentioned in the conjecture.]
§2 Proof techniques

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

\[(+ 0 (+ e_{21} e_{22})) \Downarrow 0\]

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Expression</th>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.</td>
<td>(0 \Downarrow 0)</td>
<td>By eval-const</td>
</tr>
<tr>
<td>1.</td>
<td>((+ e_{21} e_{22}) \Downarrow n_{21} + n_{22})</td>
<td>By eval-add</td>
</tr>
<tr>
<td>2.</td>
<td>((+ 0 (+ e_{21} e_{22})) \Downarrow 0 + (n_{21} + n_{22}))</td>
<td>By eval-add</td>
</tr>
<tr>
<td>3.</td>
<td>(n_{21} = 0)</td>
<td>Above</td>
</tr>
<tr>
<td>4.</td>
<td>(n_{22} = 0)</td>
<td>Above</td>
</tr>
<tr>
<td>5.</td>
<td>(0 + (n_{21} + n_{22}) = 0)</td>
<td>By arithmetic</td>
</tr>
<tr>
<td>6.</td>
<td>((+ 0 (+ e_{21} e_{22})) \Downarrow 0)</td>
<td>By above equation ([0 + (n_{21} + n_{22}) = 0])</td>
</tr>
</tbody>
</table>
§2 Proof techniques

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.)

\[(\text{Abs } e_1) \Downarrow n_1 \quad \text{Given}\]
\[n_1 = |n| \quad \text{By inversion on rule eval-abs}\]
\[e_1 \Downarrow n \quad "\]

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

[Giving this question was a mistake. I should have given a simpler question.]

Conjecture.
For all $C, M, M'$ and $D_1$ such that $D_1$ derives $M \mapsto_R M'$, there exists $D_2$ such that $D_2$ derives $C[M] \mapsto_R C[M']$.

- Suppose that I suggest you use structural induction on $D_1$. Write the appropriate induction hypothesis:

[Refer to the three-step process given in a1sol.pdf.]

For all $C_H, M_H, M'_H$ and $D_{1H}$ such that $D_{1H} \prec D_1$ and $D_{1H}$ derives $M_H \mapsto_R M'_H$, there exists $D_{2H}$ such that $D_{2H}$ derives $C_H[M_H] \mapsto_R C_H[M'_H]$.

This was nasty, because the prime on $M'$ in the conjecture makes systematically adding primes difficult, if not impossible: $M'$ can become $M''$ but then $M$ becomes $M'$, which clashes with the $M'$ in the conjecture.

[My workaround was to add a subscript $H$ to every meta-variable.]

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

For all $C_H, M_H, M'_H$ and $D_{1H}$ such that $M_H \prec M$ and $D_{1H}$ derives $M_H \mapsto_R M'_H$, there exists $D_{2H}$ such that $D_{2H}$ derives $C_H[M_H] \mapsto_R C_H[M'_H]$.

[Note that the only difference from the IH for inducting on $D_1$ is in the $\prec$ part.]
Proof techniques

3 Typing

types \( \Lambda ::= \text{int} \)

\( e : \Lambda \) expression \( e \) has type \( \Lambda \)

\[
\begin{array}{lll}
\text{type-const} & \text{e : int} & \text{type-abs} \\
\text{n : int} & \text{Abs e : int} & \text{e_1 : int e_2 : int} \\
& \text{type-add} & \text{(+ e_1 e_2) : int}
\end{array}
\]

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 \( e : \text{int} \).

**Proof.** By structural induction on \( e \). [The only thing given is \( e \); we are trying to construct the derivation of \( e : \text{int} \), 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 \( e' : \text{int} \).

**Consider cases of \( e \).**

- **Case: \( e = n \)**
  
  \[
  \begin{array}{ll}
  n : \text{int} & \text{By applying rule type-const} \\
  e : \text{int} & \text{By above equation \( [e = n] \)}
  \end{array}
  \]

- **Case: \( e = (\text{Abs } e_1) \)**
  
  \[
  \begin{array}{ll}
  e_1 \prec e & \text{[By above equation \( e = (\text{Abs } e_1) \)]} \\
  e_1 : \text{int} & \text{By IH [with \( e_1 \) as \( e' \)]} \\
  (\text{Abs } e_1) : \text{int} & \text{By applying rule type-abs} \\
  e : \text{int} & \text{By above equation \( [e = (\text{Abs } e_1)] \)}
  \end{array}
  \]

- **Case: \( e = (+ e_1 e_2) \)**
  
  \[
  \begin{array}{ll}
  e_1 \prec e & \text{[By above equation \( e = (+ e_1 e_2) \)]} \\
  e_1 : \text{int} & \text{By IH [with \( e_1 \) as \( e' \)]} \\
  e_2 \prec e & \text{[By above equation \( e = (+ e_1 e_2) \)]} \\
  e_2 : \text{int} & \text{By IH [with \( e_2 \) as \( e' \)]} \\
  (+ e_1 e_2) : \text{int} & \text{By applying rule type-add} \\
  e : \text{int} & \text{By above equation \( [e = (+ e_1 e_2)] \)}
  \end{array}
  \]

\( \square \)