IIT CS595: Topics & Applications in Programming Languages

Homework 0: Rule induction, expressions, type safety

Dustin Van Tate Testa

September 17, 2021

1 Logistics

Writing up Solutions. If you're looking at this file, you're probably (at least thinking about) writing up your solutions in LaTeX. Nice! If you need help, see the writeup PDF for a link to resources and other options.

Collaboration. Please see the collaboration policy on the course website.

Submission. Submit your answers as a single .pdf or .doc file (see above) on Blackboard under the correct assignment.

2 Evaluation

Task 1. *Evaluate the following expression using the dynamics semantics from class, showing each step: (Put your answers in the ... and copy and paste the line to add more lines as necessary).*

$$
\begin{array}{ll}
 & \mathsf{let}\ x = \overline{2}\ \mathsf{in}\ x + (\mathsf{let}\ x = \overline{1}\ \mathsf{in}\ \mathsf{let}\ y = x + \overline{1}\ \mathsf{in}\ x + y) \\
\mapsto & \overline{2} + (\mathsf{let}\ x = \overline{1}\ \mathsf{in}\ \mathsf{let}\ y = x + \overline{1}\ \mathsf{in}\ x + y) \\
\mapsto & \overline{2} + (\mathsf{let}\ y = \overline{1} + \overline{1}\ \mathsf{in}\ \overline{1} + y \\
\mapsto & \overline{2} + (\mathsf{let}\ y = \overline{2}\ \mathsf{in}\ \overline{1} + y \\
\mapsto & \overline{2} + (\overline{1} + \overline{2}) \\
\mapsto & \overline{2} + \overline{3} \\
\mapsto & \overline{5}
\end{array}
$$

3 Red-Black Trees

In data structures, a red-black tree is a binary tree whose nodes are colored either red or black. Red-black trees must obey the following rules:

- Leaf nodes must be black.
- No red node has red children.
- Any path from the root to a leaf contains the same number of black nodes.

We can express red-black trees using the following small language (note that this syntax doesn't enforce the rules above; we'll do that later).

$$
\begin{array}{llll}
\mathit{Trees} & T & ::= & \mathsf{Leaf} \mid \mathsf{BNode}(T, T) \mid \mathsf{RNode}(T, T) \\
\mathit{Colors} & c & ::= & \mathsf{Black} \mid \mathsf{Red}
\end{array}
$$

where Leaf is a leaf and $\mathsf{BNode}(T_1, T_2)$ and $\mathsf{RNode}(T_1, T_2)$ represent black and red nodes, respectively, with children T_1 and T_2.

We will define a judgment T isTree c n, meaning that T is a valid red-black tree whose root node has the color c (which is either Black or Red) and all of whose paths from root to leaf have exactly n black nodes.

$$\frac{}{\mathsf{Leaf}\ \mathsf{isTree}\ \mathsf{Black}\ 1}\ (\text{RBT-1}) \qquad \frac{T_1\ \mathsf{isTree}\ c_1\ n \qquad T_2\ \mathsf{isTree}\ c_2\ n}{\mathsf{BNode}(T_1, T_2)\ \mathsf{isTree}\ \mathsf{Black}\ n+1}\ (\text{RBT-2})$$

$$\frac{T_1\ \mathsf{isTree}\ \mathsf{Black}\ n \qquad T_2\ \mathsf{isTree}\ \mathsf{Black}\ n}{\mathsf{RNode}(T_1, T_2)\ \mathsf{isTree}\ \mathsf{Red}\ n}\ (\text{RBT-3})$$

Rule (RBT-1) says that leaves are valid trees as long as they are black: a leaf by definition has 1 black node on any path from root to leaf. Rule (RBT-2) allows a black node with two children as long as both children are valid red-black trees with the same number of black nodes on any path (regardless of what color their roots are); the result is a tree with a black root and $n+1$ black nodes on any path. Rule (RBT-3) enforces that red nodes do not have red children by requiring in the premises that both children have black roots.

Define $\mathit{Nodes}(T)$ to be the number of nodes, including leaves, in a tree:

$$\begin{array}{lcl} \mathit{Nodes}(\mathsf{Leaf}) & = & 1 \\ \mathit{Nodes}(\mathsf{BNode}(T_1, T_2)) & = & 1 + \mathit{Nodes}(T_1) + \mathit{Nodes}(T_2) \\ \mathit{Nodes}(\mathsf{RNode}(T_1, T_2)) & = & 1 + \mathit{Nodes}(T_1) + \mathit{Nodes}(T_2) \end{array}$$

Task 2. *Prove by rule induction that for any tree T, color $c \in \{\mathsf{Red}, \mathsf{Black}\}$ and $n \geq 1$, if T isTree c n, then $\mathit{Nodes}(T) \geq 2^{n-1}$.*

You must *use rule induction, not any other proof technique.*

Note: *This, together with one or two other properties of red-black trees, proves that a valid red-black tree is approximately balanced.*

Answer:

- Notice use of shorthand $|T|$ to indicate the value of n (black height) for tree T

Base Case:

- all cases when n = 1:
  - T= Leaf: Rule (RBT-1) confirms that when T is a single black node, n = 1.
    The proposition holds for this case as $\mathit{Nodes}(\mathsf{Leaf}) = 1 \geq 2^{n-1} = 2^{1-1} = 1$.
  - T= RNode(Leaf, Leaf): rule (RBT-3) confirms that when T is a red node, n is the same as it's children, which when they're both leaves makes n = 1.
    The proposition holds for this case as follows:
    LHS: $\mathit{Nodes}(\mathsf{RNode}(\mathsf{Leaf}, \mathsf{Leaf})) = 1 + \mathit{Nodes}(\mathsf{Leaf}) + \mathit{Nodes}(\mathsf{Leaf}) = 1 + 1 + 1 = 3$
    RHS: $2^{n-1} = 2^{1-1} = 2^0 = 1$
    giving $3 \geq 1$

Inductive case assuming $\mathit{Nodes}(T) \geq 2^{n-1}$:

- Case when $T = \mathsf{BNode}(..., ...)$ – tree's top element is a black node

$\mathit{Nodes}(\mathsf{BNode}(T_1, T_2))$	$\geq$	$2^{\lvert T\rvert-1}$	the proposition for this case
$1 + \mathit{Nodes}(T_1) + \mathit{Nodes}(T_2)$	$\geq$	$1 + 2^{\lvert T_1\rvert-1} + 2^{\lvert T_2\rvert-1}$	apply rule (RBT-2)
1	$\geq$	1	apply inductive hypothesis

- Case when $T = \mathsf{RNode}(..., ...)$ – tree's top element is a red node

$\mathit{Nodes}(\mathsf{RNode}(T_1, T_2))$	$\geq$	$2^{\lvert T\rvert-1}$	the proposition for this case
$1 + \mathit{Nodes}(T_1) + \mathit{Nodes}(T_2)$	$\geq$	$2^{\lvert T_1\rvert-1} + 2^{\lvert T_2\rvert-1}$	apply rule (RBT-3)
1	$\geq$	0	apply inductive hypothesis

4 Booleans

In this task, you'll extend the E language from class with Booleans:

$$\begin{array}{rrcl} \textit{Types} & \tau & ::= & \mathsf{int} \mid \mathsf{string} \mid \mathsf{bool} \\ \textit{Expressions} & e & ::= & x \mid \overline{n} \mid \text{“}s\text{”} \mid \mathsf{true} \mid \mathsf{false} \mid e = e \mid e + e \mid e \mathbin{\hat{}} e \mid |e| \mid \mathsf{let}\ x = e\ \mathsf{in}\ e \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \end{array}$$

The expressions true and false are values and have their usual meanings. The expression if e_1 then e_2 else e_3 should evaluate e_1. If it evaluates to true, it should continue evaluating e_2, otherwise it should continue evaluating e_3. The other branch *should not* be evaluated (this isn't possible to express in E, but consider the code if $x = \overline{0}$ then $\overline{0}$ else $\overline{42}/x$: we certainly don't want to evaluate the else branch when the condition is true!). We've also added an integer equality test $e_1 = e_2$ to have an interesting way of producing Booleans (this is an operation on integers only, not on Booleans or strings). Recall the dynamic semantics from class, now extended with the rules for $e_1 = e_2$.

Note: There's a lot here, but don't panic: the only new rules here are (S-11) through (S-14). The rest are just there as a reminder. Your job is only to add Booleans to the language.

$$\frac{}{\overline{n}\ \mathsf{val}}\ \text{(V-1)} \qquad \frac{}{\text{“}s\text{”}\ \mathsf{val}}\ \text{(V-2)} \qquad \frac{}{\overline{n_1} + \overline{n_2} \mapsto \overline{n_1 + n_2}}\ \text{(S-1)} \qquad \frac{}{\text{“}s_1\text{”} \mathbin{\hat{}} \text{“}s_2\text{”} \mapsto \text{“}s_1 s_2\text{”}}\ \text{(S-2)}$$

$$\frac{}{|\text{“}s\text{”}| \mapsto \overline{|s|}}\ \text{(S-3)} \qquad \frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2}\ \text{(S-4)} \qquad \frac{e_2 \mapsto e_2'}{\overline{n_1} + e_2 \mapsto \overline{n_1} + e_2'}\ \text{(S-5)} \qquad \frac{e_1 \mapsto e_1'}{e_1 \mathbin{\hat{}} e_2 \mapsto e_1' \mathbin{\hat{}} e_2}\ \text{(S-6)}$$

$$\frac{e_2 \mapsto e_2'}{\text{“}s_1\text{”} \mathbin{\hat{}} e_2 \mapsto \text{“}s_1\text{”} \mathbin{\hat{}} e_2'}\ \text{(S-7)} \qquad \frac{e \mapsto e'}{|e| \mapsto |e'|}\ \text{(S-8)} \qquad \frac{e_1 \mapsto e_1'}{\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mapsto \mathsf{let}\ x = e_1'\ \mathsf{in}\ e_2}\ \text{(S-9)}$$

$$\frac{v\ \mathsf{val}}{\mathsf{let}\ x = v\ \mathsf{in}\ e_2 \mapsto [v/x]e_2}\ \text{(S-10)} \qquad \frac{e_1 \mapsto e_1'}{e_1 = e_2 \mapsto e_1' = e_2}\ \text{(S-11)} \qquad \frac{e_2 \mapsto e_2'}{\overline{n_1} = e_2 \mapsto \overline{n_1} = e_2'}\ \text{(S-12)}$$

$$\frac{}{\overline{n} = \overline{n} \mapsto \mathsf{true}}\ \text{(S-13)} \qquad \frac{n_1 \neq n_2}{\overline{n_1} = \overline{n_2} \mapsto \mathsf{false}}\ \text{(S-14)}$$

Task 3. *Write the new inference rules for the dynamic semantics of Booleans and the if-else construct. You should have 2 new rules for the judgment* $e\ \mathsf{val}$ *and 3 new rules for the judgment* $e \mapsto e$*.*
*(**Hint:** only one of these will be a search rule.)*

Answer:

$$\frac{}{\mathsf{false}\ \mathsf{val}}\ \text{(V-3)} \qquad \frac{}{\mathsf{true}\ \mathsf{val}}\ \text{(V-4)} \qquad \frac{e_1 \mapsto e_1'}{\mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto \mathsf{if}\ e_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2}\ \text{(S-15)}$$

$$\frac{}{\mathsf{if}\ \mathsf{true}\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_1}\ \text{(S-16)} \qquad \frac{}{\mathsf{if}\ \mathsf{false}\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_2}\ \text{(S-17)}$$

We also extend the typing rules with the new rule for equality testing and Booleans.

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 = e_2 : \mathsf{bool}}\ \text{(T-8)} \qquad \frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}\ \text{(T-9)} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}\ \text{(T-10)}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}\ \text{(T-11)}$$

Note that rule (T-11) doesn't require that the two branches of the conditional have a particular type (e.g. int). The use of the same metavariable τ *does* however, mean that the two branches e_2 and e_3 must have the *same* type, which is then the type of the whole expression (this makes sense: how could we possibly give a type to the expression $\mathsf{if}\ n = \overline{0}\ \mathsf{then}\ \overline{42}\ \mathsf{else}$ "Oops"?).

Task 4. *Prove the cases of the Preservation theorem for the new rules you added in Task 3.*

Answer:
Proposition: if $\Gamma \vdash e : \tau$ and $e \mapsto e'$ then $\Gamma \vdash e' : \tau$

- S-16: When $e = \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
  $\mapsto e' = e_1$
  by inversion on rule (T-11) $\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$ and $\Gamma \vdash e_1 : \tau$
  with induction on $e_1 \mapsto e_1'$ this confirms that the type of the stepped value is preserved.
- S-17: When $e = \mathsf{if}\ \mathsf{false}\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
  $\mapsto e' = e_2$
  by inversion on rule (T-11) $\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau$ and $\Gamma \vdash e_2 : \tau$
  with induction on $e_2 \mapsto e_2'$ this confirms that the type of the stepped value is preserved.
- S-15: When $e = \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
  $\mapsto e' = \mathsf{if}\ e_0'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
  by inversion on rule (T-11), $\Gamma \vdash e_0 : \mathsf{bool}$, $\Gamma \vdash e_1 : \tau$, and $\Gamma \vdash e_2 : \tau$
  by induction we can assume $e_0 \mapsto e_0'$ with $\Gamma \vdash e_0' : \mathsf{bool}$
  thus we can apply (T-11) to assert that the type is the same.

Task 5. *State (you do not have to prove it) the new case of the Canonical Forms lemma for Booleans.*

Answer: if e val and $\vdash e : \mathsf{bool}$ then e is either true or false.

Task 6. *Prove the cases of the Progress theorem for the new rules (T-9) through (T-11). You may (and should) use the new case of Canonical Forms from Task 5.*

Answer:
Proposition: if $\vdash e : \tau$ then e val or $e \mapsto e'$

- T-9: $e = \mathsf{true}$, $\tau = \mathsf{bool}$
  by V-4 e val
- T-10: $e = \mathsf{false}$, $\tau = \mathsf{bool}$
  by V-3 e val
- T-11: $e = \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3$
  by inversion on rule T-11 $\vdash e_1 : \mathsf{bool}$
  by induction, either e_1 val or $e_1 \mapsto e_1'$
  by the Canonical Forms lemma for from Task 5 e_1 can hold either trueor false
  - true: S-16 confirms that $\mathsf{if}\ \mathsf{true}\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \mapsto e_2$ and thus $e' = e_2$
  - false: S-17 confirms that $\mathsf{if}\ \mathsf{false}\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \mapsto e_3$ and thus $e' = e_3$