

IIT CS595: Topics & Applications in Programming Languages

Homework 0: Rule induction, expressions, type safety

Your name here

September 10, 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).

$$\begin{array}{l} \text{let } x = \bar{2} \text{ in } x + (\text{let } x = \bar{1} \text{ in let } y = x + \bar{1} \text{ in } x + y) \\ \mapsto \dots \\ \mapsto \dots \end{array}$$

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).

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

where **Leaf** is a leaf and **BNode**(T_1, T_2) and **RNode**(T_1, T_2) represent black and red nodes, respectively, with children T_1 and T_2 .

We will define a judgment $T \text{ isTree } c \ n$, 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.

$$\begin{array}{c}
\frac{}{\text{Leaf isTree Black } 1} \text{ (RBT-1)} \qquad \frac{T_1 \text{ isTree } c_1 \ n \quad T_2 \text{ isTree } c_2 \ n}{\text{BNode}(T_1, T_2) \text{ isTree Black } n+1} \text{ (RBT-2)} \\
\\
\frac{T_1 \text{ isTree Black } n \quad T_2 \text{ isTree Black } n}{\text{RNode}(T_1, T_2) \text{ isTree Red } n} \text{ (RBT-3)}
\end{array}$$

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

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

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

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:

4 Booleans

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

$$\begin{array}{ll}
\text{Types } \tau & ::= \text{int} \mid \text{string} \mid \text{bool} \\
\text{Expressions } e & ::= x \mid \bar{n} \mid "s" \mid \text{true} \mid \text{false} \mid e = e \mid e + e \mid e \wedge e \mid |e| \mid \text{let } x = e \text{ in } e \mid \text{if } e \text{ then } e \text{ else } e
\end{array}$$

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** $x = \bar{0}$ **then** $\bar{0}$ **else** $\bar{42}/x$: 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.

$$\begin{array}{c}
\overline{\overline{n} \text{ val}} \text{ (V-1)} \quad \overline{\overline{“s”} \text{ val}} \text{ (V-2)} \quad \overline{\overline{\overline{n_1} + \overline{n_2} \mapsto \overline{n_1} + \overline{n_2}}} \text{ (S-1)} \quad \overline{\overline{“s_1” \wedge “s_2” \mapsto “s_1 s_2”}} \text{ (S-2)} \\
\overline{\overline{|“s”| \mapsto |\overline{s}|}} \text{ (S-3)} \quad \frac{e_1 \mapsto e'_1}{e_1 + e_2 \mapsto e'_1 + e_2} \text{ (S-4)} \quad \frac{e_2 \mapsto e'_2}{\overline{n_1} + e_2 \mapsto \overline{n_1} + e'_2} \text{ (S-5)} \quad \frac{e_1 \mapsto e'_1}{e_1 \wedge e_2 \mapsto e'_1 \wedge e_2} \text{ (S-6)} \\
\frac{e_2 \mapsto e'_2}{“s_1” \wedge e_2 \mapsto “s_1” \wedge e'_2} \text{ (S-7)} \quad \frac{e \mapsto e'}{|e| \mapsto |e'|} \text{ (S-8)} \quad \frac{e_1 \mapsto e'_1}{\text{let } x = e_1 \text{ in } e_2 \mapsto \text{let } x = e'_1 \text{ in } e_2} \text{ (S-9)} \\
\frac{v \text{ val}}{\text{let } x = v \text{ in } e_2 \mapsto [v/x]e_2} \text{ (S-10)} \quad \frac{e_1 \mapsto e'_1}{e_1 = e_2 \mapsto e'_1 = e_2} \text{ (S-11)} \quad \frac{e_2 \mapsto e'_2}{\overline{n_1} = e_2 \mapsto \overline{n_1} = e'_2} \text{ (S-12)} \\
\overline{\overline{\overline{n} = \overline{n} \mapsto \text{true}}} \text{ (S-13)} \quad \frac{n_1 \neq n_2}{\overline{n_1} = \overline{n_2} \mapsto \text{false}} \text{ (S-14)}
\end{array}$$

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 $e \text{ val}$ and 3 new rules for the judgment $e \mapsto e$. (*Hint: only one of these will be a search rule.*)

Answer:

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

$$\begin{array}{c}
\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 = e_2 : \text{bool}} \text{ (T-8)} \quad \overline{\overline{\Gamma \vdash \text{true} : \text{bool}}} \text{ (T-9)} \quad \overline{\overline{\Gamma \vdash \text{true} : \text{bool}}} \text{ (T-10)} \\
\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau} \text{ (T-11)}
\end{array}$$

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 `if $n = \overline{0}$ then $\overline{42}$ else “Oops”`?).

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

Answer:

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

Answer:

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 ??.

Answer: