COMS 6998 TLC: Homework 1

Stephen A. Edwards, Columbia University

Due Sunday, March 5 at 11:59 PM, 2023

Please submit this as a PDF file on Courseworks. You may handwrite and scan or create PDF on the computer, e.g., using LaTeX. For the latter, check out the *semantic*, *syntax*, and *libertine* packages.

1. String Concatenation

   Assume you know for any character c whether $c \in \Sigma$ and for any sequence s of zero-or-more characters in Σ that "s" $\in \Sigma^*$.

   Use the following rules for string concatenation

   $$\frac{\text{“}s\text{”} \in \Sigma^*}{\text{“”} \mathbin{+\!\!+} \text{“}s\text{”} = \text{“}s\text{”}}\ \text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} \mathbin{+\!\!+} \text{“}s_2\text{”} = \text{“}s_3\text{”}}{\text{“}cs_1\text{”} \mathbin{+\!\!+} \text{“}s_2\text{”} = \text{“}cs_3\text{”}}\ \text{concat}$$

   to prove that

   $$\textbf{“con”} \mathbin{+\!\!+} \textbf{“cat”} = \textbf{“concat”}$$

   when $\Sigma = \{\mathbf{a}, \mathbf{c}, \mathbf{n}, \mathbf{o}, \mathbf{t}\}$.

2. Binary Search Trees

First, assume you can conclude whether something is a natural number, i.e., $n \in \mathbb{N}$, and that given two $n_1, n_2 \in \mathbb{N}$, you can conclude whether $n_1 < n_2$ or $n_1 = n_2$, i.e., you do not have to offer proofs for, say, $5 \in \mathbb{N}$, $3 < 4$, or $2 = 2$. You may not assume any other properties of natural numbers.

Consider the following axiomatization of binary search trees with unique elements written syntactically. Something like "leaf 4" represents a tree with a single leaf holding 4 whereas "branch(leaf 2, leaf 4, 3)" represents the tree 

. The judgments "min(t, n)" and "max(t, n)" are meant to indicate n is the minimum and maximum values in tree t, respectively.

$$\frac{n \in \mathbb{N}}{\max(\text{leaf } n, n)}\ \text{max-leaf} \qquad \frac{n \in \mathbb{N} \quad \max(t_2, n_2) \quad n < n_2}{\max(\text{branch}(t_1, t_2, n), n_2)}\ \text{max-branch}$$

$$\frac{n \in \mathbb{N}}{\min(\text{leaf } n, n)}\ \text{min-leaf} \qquad \frac{n \in \mathbb{N} \quad \min(t_1, n_1) \quad n_1 < n}{\min(\text{branch}(t_1, t_2, n), n_1)}\ \text{min-branch}$$

$$\frac{n \in \mathbb{N}}{\text{leaf } n : \text{BST}}\ \text{BST-leaf}$$

$$\frac{n \in \mathbb{N} \quad \max(t_1, n_1) \quad \min(t_2, n_2) \quad n_1 < n \quad n < n_2}{\text{branch}(t_1, t_2, n) : \text{BST}}\ \text{BST-branch}$$

(a) In the style shown in class (i.e., a stack of rules), prove that

$$\text{branch}(\text{leaf } 1, \text{leaf } 5, 3) : \text{BST}$$

is true under these rules.

(b) Unfortunately, these rules don't correctly capture the usual constraints on the values in a binary tree (e.g., everything on the left branch is less than the value; everything on the right is greater). Devise a counterexample by replacing each ? below with numbers such that t violates the usual properties of a binary tree, yet $t : \text{BST}$ under the rules above, i.e., show a proof of $t : \text{BST}$.

$$t = \text{branch}(\text{branch}(\text{leaf } ?, \text{leaf } ?, 8), \text{branch}(\text{leaf } ?, \text{leaf } ?, 12), 10)$$

(c) Propose a corrected BST-branch rule (i.e., modify the preconditions above the line, not the conclusion below it) that correctly reflects the usual properties of a binary search tree. Don't add or change any other existing rules at this point.

(d) Now that your BST-branch rule works, can you remove any preconditions in the other rules without breaking the overall tree correctness property?

(e) Add searching rules, i.e., that can conclude $n \in t$ when n is a natural number that appears in the tree t.

Here is the base case:

$$\frac{n \in \mathbb{N}}{n \in \text{leaf } n} \text{ in-leaf}$$

Your other rules should look like

$$\frac{?}{n_1 \in \text{branch}(t_1, t_2, n_2)} \text{ in-branch-?}$$

Do not introduce unnecessary variables or preconditions.

Your rules must be algorithmic (i.e., they could be implemented in Haskell or some other language) and unambiguous (there should only be one way to prove each valid statement).

Don't forget that t must be a binary search tree; assert this in your rules.

3. Top-Down (Predictive) Parsing

   The left-factored grammar

   dig ::= **0** | **1** | … | **9**

   expr ::= *dig exprt*

   exprt ::= **+** *dig exprt* | **-** *dig exprt* |

   (note that *exprt* may be the empty string) can be expressed as the following set of rules:

$$\frac{}{\mathbf{0} \xrightarrow[]{\text{dig}} \text{lit}(0)}\ \text{zero} \qquad \cdots \qquad \frac{}{\mathbf{9} \xrightarrow[]{\text{dig}} \text{lit}(9)}\ \text{nine}$$

$$\frac{d \xrightarrow[]{\text{dig}} d' \qquad e \xrightarrow[d']{\text{exprt}} e'}{d\ e \xrightarrow[]{\text{expr}} e'}\ \text{expr} \qquad \frac{}{\xrightarrow[e']{\text{exprt}} e'}\ \text{exprt}$$

$$\frac{d \xrightarrow[]{\text{dig}} d' \qquad e \xrightarrow[\text{add}(e',d')]{\text{exprt}} e''}{\mathbf{+}\ d\ e \xrightarrow[e']{\text{exprt}} e''}\ \text{add} \qquad \frac{d \xrightarrow[]{\text{dig}} d' \qquad e \xrightarrow[\text{sub}(e',d')]{\text{exprt}} e''}{\mathbf{-}\ d\ e \xrightarrow[e']{\text{exprt}} e''}\ \text{sub}$$

   Prove that under these rules,

$$\mathbf{1 - 2 - 3} \xrightarrow{\text{expr}} \text{sub}(\text{sub}(\text{lit}(1), \text{lit}(2)), \text{lit}(3))$$

   Hint: for a judgment of the form $e \xrightarrow[e']{\text{exprt}} e''$, think of e and e' as inputs and e'' as the output. Ultimately, the exprt rule tells us that if e is empty, e'' is just e'.