Problems: Regular Trees

1. A *non-deterministic, bottom-up, finite-state tree automaton* (which will be abbreviated as FSTA) is a formalism for recognizing (or "accepting") members of a language of trees. An FSTA $A = (Q, Q^F, \Sigma, \delta)$ has a set of *states* Q, a set of final states $Q^F \subseteq Q$, a *ranked alphabet* Σ, and a transition relation δ.

   A ranked alphabet means that each symbol has an *arity*, which indicates how many children it has. We will denote, e.g., a binary (arity-2) symbol *foo* by *foo*^{2}. Thus, if T_1 and T_2 are two trees, *foo*$^2(T_1, T_2)$ is another tree; it has the symbol *foo*^{2} at its root.

   The transition relation δ consists of rules of the form

   $$q(f^n) \leftarrow f^n(q_1, \ldots, q_n),$$

   where $q, q_1, \ldots, q_n \in Q$ and f^n is an n-ary symbol. We allow A to be non-deterministic; that is, one can have multiple result states (i.e., left-hand-side states) for a given combination of symbol and child states:

   $$\begin{array}{rcl} q(f^n) & \leftarrow & f^n(q_1, \ldots, q_n) \\ q'(f^n) & \leftarrow & f^n(q_1, \ldots, q_n) \end{array}$$

   An FSTA A accepts a language of trees $L(A)$. For a given tree T, T is accepted or rejected depending on the outcome of the possible *runs* of A over T. A run labels each leaf of T with a state, and then moves upward to successively label each node of T with a state, using the rules of δ. That is, if we have the rule

   $$q(f^n) \leftarrow f^n(q_1, \ldots, q_n)$$

   in δ and there is a subtree S whose root symbol is f^n and whose n children are labeled with $q_1, \ldots, q_n$, respectively, then the root of S can be labeled with q.

   An *accepting run* is one that labels the root of the tree with a state in Q^f. Because we allow A to be non-deterministic, only *one* of the possible runs of A over T needs to be an accepting run for T to be *accepted* (i.e., for $T \in L(A)$ to hold).

   An FSTA has no initial state, but the rules for 0-ary symbols cause certain states to act as initial states at a tree's various leaves. For instance, suppose that we have the rule

   $$q_{17}(a^0) \leftarrow a^0()$$

   Then if T has any instance of a^0 as a leaf, that leaf can be labeled with q_{17}, and serves as one of the "initial" states for runs of A over T. Note that we are permitted to have multiple rules for a 0-ary symbol:

   $$\begin{array}{rcl} q_{15}(a^0) & \leftarrow & a^0() \\ q_{17}(a^0) & \leftarrow & a^0() \end{array}$$

Example. Consider the FSTA A_{exp} defined as follows:

$$A_{exp} = (\{q_{int}, q_{\text{float}}, q_{\text{error}}\}, \{q_{int}, q_{\text{float}}\}, \{plus^2, a^0, m^0, x^0\},$$
$$\left\{ \begin{array}{rcl} q_{int}(a^0) & \leftarrow & a^0() \\ q_{int}(m^0) & \leftarrow & m^0() \\ q_{\text{float}}(m^0) & \leftarrow & m^0() \\ q_{\text{float}}(x^0) & \leftarrow & x^0() \\ q_{int}(plus^2) & \leftarrow & plus^2(q_{int}, q_{int}) \\ q_{\text{float}}(plus^2) & \leftarrow & plus^2(q_{\text{float}}, q_{\text{float}}) \\ q_{\text{error}}(plus^2) & \leftarrow & plus^2(q_{int}, q_{\text{float}}) \end{array} \quad \begin{array}{rcl} q_{\text{error}}(plus^2) & \leftarrow & plus^2(q_{\text{float}}, q_{int}) \\ q_{\text{error}}(plus^2) & \leftarrow & plus^2(q_{\text{error}}, q_{int}) \\ q_{\text{error}}(plus^2) & \leftarrow & plus^2(q_{\text{error}}, q_{\text{float}}) \\ q_{\text{error}}(plus^2) & \leftarrow & plus^2(q_{int}, q_{\text{error}}) \\ q_{\text{error}}(plus^2) & \leftarrow & plus^2(q_{\text{float}}, q_{\text{error}}) \\ q_{\text{error}}(plus^2) & \leftarrow & plus^2(q_{\text{error}}, q_{\text{error}}) \end{array} \right\}).$$

Let T_1 and T_2 be two trees defined as follows:

$$\begin{array}{rcl} T_1 & = & plus^2(plus^2(m^0(), a^0()), a^0()) \\ T_2 & = & plus^2(plus^2(a^0(), b^0()), x^0()) \end{array}$$

Note that there is both an accepting run for T_1, namely,

$$\begin{array}{rl} & plus^2(plus^2(m^0(), a^0()), a^0()) \\ \Rightarrow & plus^2(plus^2(q_{int}(m^0), q_{int}(a^0)), q_{int}(a^0)) \\ \Rightarrow & plus^2(q_{int}(plus^2(m^0(), a^0())), q_{int}(a^0)) \\ \Rightarrow & q_{int}(plus^2(plus^2(m^0(), a^0()), a^0())) \end{array}$$

and a non-accepting run for T_1,

$$\begin{array}{rl} & plus^2(plus^2(m^0(), a^0()), a^0()) \\ \Rightarrow & plus^2(plus^2(q_{\text{float}}(m^0), q_{int}(a^0)), q_{int}(a^0)) \\ \Rightarrow & plus^2(q_{\text{error}}(plus^2(m^0(), a^0())), q_{int}(a^0)) \\ \Rightarrow & q_{\text{error}}(plus^2(plus^2(m^0(), a^0()), a^0())) \end{array}$$

In contrast, there is only a non-accepting run for T_2, namely,

$$\begin{array}{rl} & plus^2(plus^2(a^0(), b^0()), x^0()) \\ \Rightarrow & plus^2(plus^2(q_{int}(a^0), q_{int}(b^0)), q_{\text{float}}(x^0)) \\ \Rightarrow & plus^2(q_{int}(plus^2(a^0(), b^0())), q_{\text{float}}(x^0)) \\ \Rightarrow & q_{\text{error}}(plus^2(plus^2(a^0(), b^0()), x^0)) \end{array}$$

Consequently, $T_1 \in L(A_{exp})$ but $T_2 \notin L(A_{exp})$. □
Abbreviations:

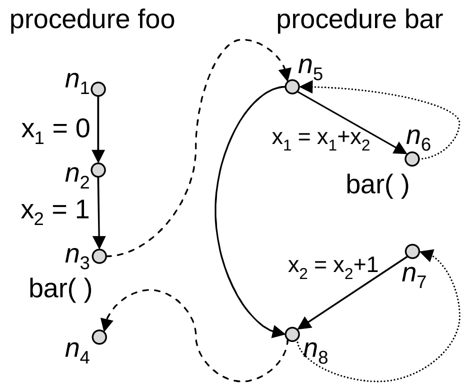
- You may drop superscripts on alphabet symbols.
- Although we wrote out all of the possible transitions involving q_{error}, it would have been convenient to treat q_{error} as a "stuck" state—in which case, in the set of rules for A_{exp} we would have omitted the last two rules in the first column and all the rules in the second column. Such rules would be implicit: an occurrence of q_{error} in any child of an arity-k symbol results in the symbol being labeled with q_{error}.

Part (a)

Explain how ordinary non-deterministic finite-state (string) automata are a degenerate case of FSTAs.

We now describe ideas needed for Parts (b), (c), and (d).
FSTAs are useful in dataflow-analysis and model-checking problems because they can be used to describe the matched call-and-return structure of paths in multi-procedure programs. Parts (b) and (c) concern how to define an FSTA to specify a tree-language that captures the matched paths specific to a particular program, namely, the one shown below. (The "if(*)" denotes a non-deterministic branch.)

```
void foo() {                 void bar() {
   n1: x1 = 0;                  n5: if (*) {
   n2: x2 = 1;                          x1 = x1 + x2;
   n3: bar();                       n6: bar();
   n4: ;                            n7: x2 = x2+1;
}                               }
                                n8: ;
                             }
```



The FSTA you will define should accept a language, each tree of which represents a properly matched path from n_1 to n_4. For each properly matched path ρ from n_1 to n_4, the FSTA should accept a tree that represents ρ. The FSTA should reject trees that either (i) do not represent a path, or (ii) represent a path that violates matched call-and-return structure in the graph given above.
The alphabet Σ consists of three kinds of symbols:

(a) Nine 0-ary symbols for the nine edges in the graph given above:

$$\{e^0_{n_1\to n_2}, e^0_{n_2\to n_3}, e^0_{n_3\to n_5}, e^0_{n_5\to n_8}, e^0_{n_5\to n_6}, e^0_{n_6\to n_5}, e^0_{n_7\to n_8}, e^0_{n_8\to n_7}, e^0_{n_8\to n_4}\}.$$

(b) Sixty-four binary symbols for possible start/end positions in a sub-path:

$$\{p^2_{n_i\to n_j} \mid 1 \le i, j \le 8\}.$$

(c) Sixty-four ternary symbols for subtrees that represent possible matched call-and-return sub-paths: $\{c^3_{n_i\to n_j} \mid 1 \le i, j \le 8\}$. (The symbols are ternary so that the three children can represent a call-edge from caller-to-callee, an edge or a matched path from the entry node to the exit node of the callee, and a return-edge from callee-to-caller.)

Part (b)
The idea is that the frontier of each tree (i.e., its sequence of leaves in left-to-right order) represents a candidate path. Draw the three trees that represent the following paths:

(a) $[n1 \to n2, n2 \to n3, n3 \to n5, n5 \to n8, n8 \to n4]$

(b) $[n1 \to n2, n2 \to n3, n3 \to n5, n5 \to n6, n6 \to n5, n5 \to n6, n6 \to n5, n5 \to n8, n8 \to n7, n7 \to n8, n8 \to n7, n7 \to n8, n8 \to n4]$

(c) $[n1 \to n2, n2 \to n3, n3 \to n5, n5 \to n8, n8 \to n7]$

(Note: the first two trees should be accepted by the FSTA that you will define in Part (c); the third tree should be rejected by the FSTA from Part (c).)

Part (c)

The alphabet Σ of the FSTA has been defined above. The set of states Q of the FSTA consists of a stuck state, q_{error}, together with 64 states that are indexed by a pair of node names: $Q = \{q_{\text{error}}\} \cup \{q_{n_i \to n_j} \mid 1 \leq i, j \leq 8\}$. The set of final states is defined as follows: $Q^F = \{q_{n_1 \to n_4}\}$. Using Q, Q^F, and Σ as defined above, *sketch* the definition of an FSTA that accepts the language of trees that represent all properly matched paths from n_1 to n_4. The intention is that state $q_{n_i \to n_j}$ only arises in a run when there exists a matched path from n_i to n_j.

Note: There are 65 different states and 137 alphabet symbols. We do not expect you to write out the full transition relation; however, it should be clear from your answer what the essential features are and what the intended pattern is.

Explain why your FSTA accepts the first two trees from your answer to Part (b), and why it rejects the third tree from Part (b).

Part (d)

Given an FSTA $A = (Q, \Sigma, \delta, q^0, Q^F)$, give an algorithm for determining whether $L(A) = \emptyset$.

2. A *regular tree grammar* is a formalism for specifying languages of trees. For instance, the following grammar G

$$\begin{array}{rcl} exp & ::= & PlusExp(exp, exp) \\ & \mid & TimesExp(exp, exp) \\ & \mid & IntExp(natNum) \\ & \mid & Variable(ident) \end{array}$$

where $natNum = \{0, 1, 2, \ldots\}$ and $ident$ is some finite or infinite set of allowable identifiers (e.g., $\{A, B, \ldots, X, Y, Z\}$), defines a language $L(G)$ of trees (or terms). $L(G)$ includes the trees

$$\begin{array}{l} Variable(A),\ Variable(B), \ldots,\ Variable(Z), \\ IntExp(0),\ IntExp(1), \ldots, \\ PlusExp(Variable(A), IntExp(0)),\ PlusExp(Variable(B), IntExp(0)), \ldots, \\ TimesExp(Variable(A), IntExp(0)),\ TimesExp(Variable(B), IntExp(0)), \ldots, \\ PlusExp(IntExp(0), Variable(A)),\ PlusExp(IntExp(0), Variable(B)), \ldots, \\ TimesExp(IntExp(0), Variable(A)),\ TimesExp(IntExp(0), Variable(B)), \ldots, \end{array}$$

Let us now introduce some terminology: *exp*, *natNum*, and *ident* are called *nonterminals* (or *types*); *PlusExp*, *TimesExp*, *IntExp*, and *Variable* are called *operators*. It is useful to consider $0, 1, 2, \ldots$ as nullary operators of type *natNum* (in which case we might write them as $0(), 1(), 2(), \ldots$) and $A, B, \ldots, X, Y, Z$ as nullary operators of type *ident* (in which case we might write them as A(), B(), ..., X(), Y(), Z()). Hence, with this notation one of the trees in $L(G)$ is $PlusExp(Variable(A()), IntExp(0()))$.

Note that each operator has a fixed *arity* that specifies the number of children that it has. For instance, the arities of some of the operators of G are as follows:

Operator	*Arity*
PlusExp	2
TimesExp	2
IntExp	1
Variable	1
0	0
1	0
2	0
⋮	⋮
A	0
B	0
⋮	⋮
Z	0

Let the children of an arity-k operator be numbered $1, \ldots, k$.

A *path* in a tree can be described by a string over an alphabet of (compound) symbols of the form

$$(nonterminal :: Operator.childNum)$$

(By convention, if *Operator* is a nullary operator, *childNum* is 0.) For instance, the set of root-to-leaf paths in the tree $PlusExp(Variable(A()), IntExp(0()))$ is

$$\left\{ \begin{array}{l} (exp :: PlusExp.1)(exp :: Variable.1)(ident :: A.0), \\ (exp :: PlusExp.2)(exp :: IntExp.1)(natNum :: 0.0) \end{array} \right\}.$$

Part (a)
Describe how to create an ordinary finite-state automaton that accepts the language of root-to-leaf paths in given a regular tree grammar H. That is, given a regular tree grammar H, your construction should produce the automaton A_H that accepts

$$\{p \mid p \text{ is a root-to-leaf path in some tree T } \in L(H)\}.$$

Part (b)
Give the automaton that would be produced by your construction for the regular tree grammar G

$$\begin{array}{rcl} \mathit{exp} & ::= & \mathit{PlusExp}(\mathit{exp}, \mathit{exp}) \\ & | & \mathit{TimesExp}(\mathit{exp}, \mathit{exp}) \\ & | & \mathit{IntExp}(\mathit{natNum}) \\ & | & \mathit{Variable}(\mathit{ident}) \end{array}$$

Part (c)
Regular tree grammars are related to context-free grammars in the following way: Suppose that you normalize a context-free grammar F by introducing additional nonterminals so that terminal symbols only appear in leaf productions of the form *nonterminal* $\rightarrow$ *terminal*; then, by introducing an operator symbol for each production (and treating each terminal symbol as a nullary operator), one has a regular tree grammar whose language is the set of parse trees for the context-free grammar F.

A context-free grammar can have two kinds of useless nonterminals:

Useless 1 : nonterminal n is useless if there is no derivation $\mathit{root} \rightarrow^* \alpha\ n\ \beta$
Useless 2 : nonterminal n is useless if there is no finite parse tree derivable from n

Describe *two* algorithms, both working on finite automata of the kind described earlier for the language of root-to-leaf paths of a regular tree grammar:

Part (c.i) The algorithm for this part returns the set of nonterminals that are useless because of reason "Useless 1".

Part (c.ii) The algorithm for this part returns the set of nonterminals that are useless because of reason "Useless 2". (For this part, you may assume that all "Useless 1" nonterminals were removed from the context-free grammar before the automaton was constructed.)

Part (d)
Give an example of a *context-free grammar* that has both kinds of useless nonterminals, and illustrate the two algorithms on it.