ECE251: Programming Languages & Translators Fall 2010

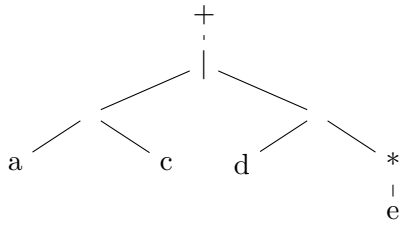
Lecture 4 — September 21, 2010

Patrick Lam *version 1*

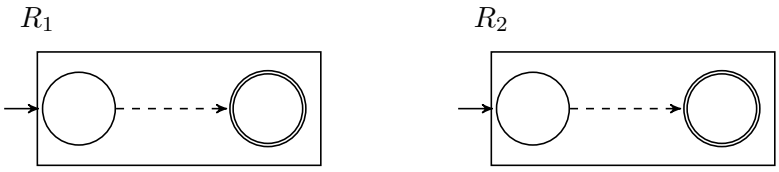
Implementation Details for using Regular Expressions

So far, we have seen NFAs, DFAs, and regular expressions. We want to execute regular expressions, but we only know how to execute NFAs and DFAs.

We will use an abstract syntax tree for the regular expression, e.g. $(ac|de^*)^+$.

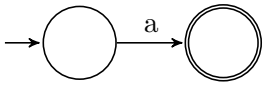


We will walk this tree and emit an NFA, using the following rules. When we have two regular expressions R_1 and R_2, assume that they look like this:

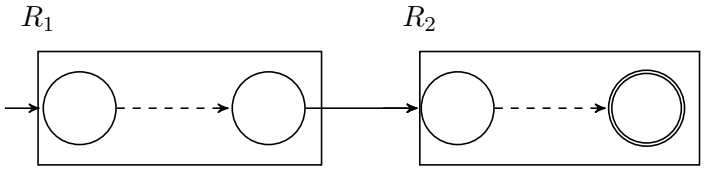


The rules:

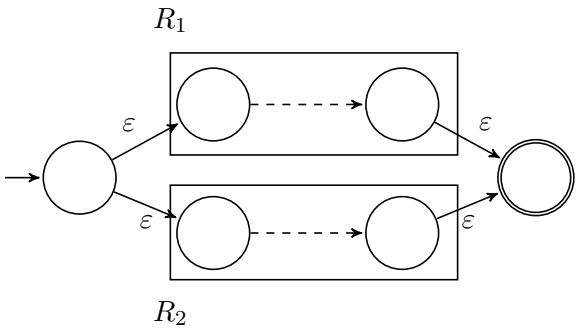
Literal `a`



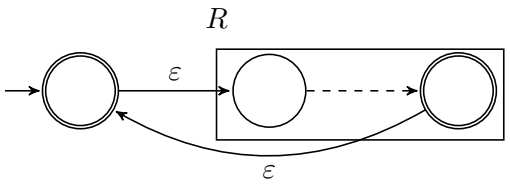
Concatenation R_1R_2



Union $R_1 \mid R_2$



Kleene Star R^*



Example: draw the finite automaton for this regular expression,

$$(1|0)^*1.$$

Summary.

Recall that our original goal was to investigate domain-specific languages for text processing. I first gave some example applications. We looked at manually-written scanner code, trying to extract the "what" (declarations) from the "how" (plumbing code). We saw that we were actually implementing a finite automaton.

OK, so how do we specify a finite automaton? Regular expressions! I introduced the first syntax and semantics of a language here, the language of regular expressions.

We then introduced the notions of deterministic and nondeterministic automata. NFA, DFA, and regular expressions have the same expressive power. (What does that mean, and how do you prove it?)

We will now continue with real-world string manipulation languages.