COMS W4115
Programming Languages and Translators
Lecture 5: Implementing a Lexical Analyzer
February 6, 2013

Outline

1. Finite automata
2. Converting an NFA to a DFA
3. Equivalence of regular expressions and finite automata
4. Simulating an NFA
5. The pumping lemma for regular languages
6. Closure and decision properties of regular languages

1. Finite Automata

- Variants of finite automata are commonly used to match regular expression patterns.
- A nondeterministic finite automaton (NFA) consists of
  - A finite set of states *S*.
  - An input alphabet consisting of a finite set of symbols Î£.
  - A transition function Î´ that maps *S* Ã— (Î£ âˆª {Îµ}) to subsets of *S*. This transition function can be represented by a transition graph in which the nodes are labeled by states and there is a directed edge labeled *a* from node *w* to node *v* if Î´(*w*, *a*) contains *v*.
  - An initial state s_0 in *S*.
  - *F*, a subset of *S*, called the final (or accepting) states.
- An NFA accepts an input string *x* iff there is a path in the transition graph from the initial state to a final state that spells out *x*.
- The language defined by an NFA is the set of strings accepted by the NFA.
- A deterministic finite automaton (DFA) is an NFA in which

1. There are no Îµ moves, and
2. For each state *s* and input symbol *a* there is exactly one transition out of *s* labeled *a*.

2. Converting an NFA to a DFA

- Every NFA can be converted to an equivalent DFA using the subset construction (Algorithm 3.20, ALSU, pp. 153-154).
- Every DFA can be converted into an equivalent minimum-state DFA Using Algorithm 3.39, ALSU, pp. 181-183. All equivalent minimum-state DFAs are isomorphic up to state renaming.

3. Equivalence of Regular Expressions and Finite Automata

- Regular expressions and finite automata define the same class of languages, namely the regular sets.
- Every regular expression can be converted into an equivalent NFA using the McNaughton-Yamada-Thompson algorithm (Algorithm 3.23, ALSU, pp. 159-161).
- Every finite automaton can be converted into a regular expression using Kleene's algorithm.

4. Simulating an NFA

- Two-stack simulation of an NFA: Algorithm 3.22, ALSU, pp. 156-159.

5. The Pumping Lemma for Regular Languages

- The pumping lemma allows us to prove certain languages, like { $a^n b^n$ | *n* â‰¥ 0 }, are not regular.

  The pumping lemma. If L is a regular language, then there exists a constant *n* associated with L such that for every string *w* in L where |*w*| â‰¥ *n*, we can partition *w* into three strings *xyz* (i.e., *w* = *xyz*) such that
  - *y* is not the empty string,
  - the length of *xy* is less than or equal to *n*, and
  - for all *k* â‰¥ 0, the string xy^kz is in L.

6. Closure and Decision Properties of Regular Languages

- The regular languages are closed under the following operations:
  - union
  - intersection
  - complement

- reversal
- Kleene star
- homomorphism
- inverse homomorphism

- Decision properties
  - Given a regular expression *r* and a string *w*, it is decidable whether *r* matches *w*.
  - Give a finite automaton A, it is decidable whether L(A) is empty.
  - Given two finite automata A and B, it is decidable whether L(A) = L(B).

7. Practice Problems

1. Write down deterministic finite automata for the following regular expressions:
   a. `(a*b*)*`
   b. `(aa|bb)*((ab|ba)(aa|bb)*(ab|ba)(aa|bb)*)*`
   c. `a(ba|a)*`
   d. `ab(a|b*c)*bb*a`
2. Construct a deterministic finite automaton that will recognize all strings of 0's and 1's representing integers that are divisible by 3. Assume the empty string represents 0.
3. Use the McNaughton-Yamada-Thompson algorithm to convert the regular expression `a(a|b)*a` into a nondeterministic finite automaton.
4. Convert the NFA of (3) into a DFA.
5. Minimize the number of states in the DFA of (4).

8. Reading Assignment

- ALSU Chapter 3, all sections except 3.9.
- Russ Cox's article [Regular Expression Matching Can Be Simple and Fast (but is slow in Java, Perl, PHP, Python, Ruby, ...)] has a good historical account on the evolution of regular expression matching programs.

---

aho@cs.columbia.edu