Lecture 10

Deterministic Finite Automata

CS 241: Foundations of Sequential Programs
Fall 2009

Troy Vasiga et al
University of Waterloo

Review

- formal languages give a theoretical basis for communication and organizing processes
- terminology (alphabet, word, language)
- specification vs. recognition
- studying language levels that increase in power/complexity
- a language is regular if there exists a DFA that specifies/recognizes it

DFAs

- a.k.a. finite state machines
- start state
- final/accepting states
- implicit error state
- accepted and rejected words
- $L(M)$ – the language recognized by DFA M
- Notice that $L(M) = L(M')$ even though $M \neq M'$

Formal Definition

A DFA is a 5-tuple $(\Sigma, Q, q_0, A, \delta)$ where

- finite alphabet Σ
- finite set of states Q
- start state q_0
- set of final/accepting states $A \subseteq Q$
- transition function: $\delta : Q \times \Sigma \rightarrow Q$

DFA Interpreter

Input: A word $w = w_1 w_2 ... w_n$
Output: true if accepted, false if rejected

Implementing DFAs

Need to implement the transition function somehow

NFAs

- $L = \{bba, baa, bbaa, bbbaa, bbbbaa, ...\}$ which is either 2 b's followed by an a, or 1 or more b's followed by 2 a's
- try to derive this using a DFA
- a nice NFA?

NFA definition

Same as a DFA with the following change:

$$T : Q \times \Sigma \rightarrow 2^Q$$

That is, we can be in a set of states, and thus T is a *relation* instead of a function.

NFA interpreter

Implementing an NFA interpreter

Differences between NFAs and DFAs

Transducers

Add an output alphabet, and each transition can output a single character from that output alphabet.