Lecture 8

Deterministic Finite Automata

CS 241: Foundations of Sequential Programs
Winter 2018

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
- regular languages are composed of union, concatenation and repetition

Recognizers: Finite Automata

Regular languages can be recognized by *finite automata*.
We begin with *deterministic finite automata*, also called DFAs.

- states
- transitions
- start state
- final states

Finite Automata Example 1

Example: selected opcodes from MIPS assembly language, where alphabet is the ASCII characters.

Observations About Finite Automata

- ability to trace
- transitions out of a state are unique
- errors
- size of this language
- DFA M and language L(M)

Finite Automata Example 2

Example: MIPS labels, where the alphabet is ASCII characters.

More Finite Automata Examples

Let $\Sigma = \{a, b, c\}$.

- strings with exactly one a and exactly one b and no c's
- strings with one a, one b and one c
- strings with at least one a
- string with an even number of a's

More Finite Automata Examples

Let $\Sigma = \{a, b, c\}$.

- strings with an even number a's and odd number of b's

- strings with an even number a's or odd number of b's

DFA summary

- 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: $\delta : Q \times \Sigma \to Q$

DFA Interpreter Algorithm

Input: A word $w = w_1 w_2 ... w_n$ where each $w_i \in \Sigma$
Output: true if accepted, false if rejected

Implementing DFAs

Need to implement the transition function somehow

Where are DFAs used?

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

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 using a nice NFA

NFA definition

Same as a DFA with the following change:

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

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

NFA Interpreter Algorithm

Input: A word $w = w_1 w_2 ... w_n$ where each $w_i \in \Sigma$
Output: true if accepted, false if rejected

Implementing an NFA interpreter

Differences between NFAs and DFAs

A few words about the subset construction

Apply the subset construction on the example language $L = \{bba, baa, bbaa, bbbaa, bbbbaa, ...\}$ which is either 2 b's followed by an a, or 1 or more b's followed by 2 a's

Review

- An example comparing DFAs and NFAs: create an NFA (then a DFA) for all words over $\Sigma = \{a, b\}$ with the subword *aba* in them.

- Can convert between an NFA and DFA using the subset construction
  - define each set of states that can be occupied at the same step
  - one state in the DFA for each unique set of states in the NFA

Killer app for Finite Automata/Transducers

Scanner: see `asm.*`