CS 378 – (Introduction to) Theory of Computation

Professor Eric Aaron

Lecture – T R 2:30pm

Lecture Meeting Location: It’s complicated…

Business

• PS2 out
  – Due date Mar. 23 (sorta extended from Mar. 18)

• PS3-Lookahead out
  – PS3 may be due Mar. 30

• Read Sipser, pages 63-69, 77-82; also, Ch. 1.4

• Additional reading to be posted: A few pages from…
  – Introduction to Automata Theory, Languages, and Computation
    by Hopcroft, Motwani, and Ullman
  – I’ll refer to this text as HMU in lecture notes sometimes
Pumping Lemma

• Have you met Hans und Franz?

(The official lemma of Dwayne “The Rock” Johnson.)
(Not really.)

(Hans & Franz hosted the Arnold-inspired “Pumping Up with Hans & Franz” on Saturday Night Live, and they were responsible for some very non-regular language... but that’s not actually why we’re here....)

(They’ll be back....)

The Pumping Lemma

• Let \( L \) be a regular language. Then there exists \( n \) s.t. for all \( z \) in \( L \), if \( |z| \geq n \), \( z \) can be written as some \( uvw \) s.t.
  – \( uv^iw \) in \( L \) for all \( i \geq 0 \),
  – \( |uv| \leq n \), and
  – \( |v| \geq 1 \)
Using The Pumping Lemma

- Claim: \( \{0^i1^i \mid i \geq 0\} \) is not regular
- Claim: \( \{0^{i^2} \mid i \geq 0\} \) is not regular
- Claim \( \{ww \mid w \in \{0,1\}^* \} \) is not regular
- Claim \( \{0^p \mid p \text{ is prime}\} \) is not regular
Regular Expressions
(yet ANOTHER way of expressing the computation of regular languages)

- Regular expressions are commonly used in computational pattern matching
  - Each regular expression denotes a set (a language)
  - A regular expression is not a set—it is a piece of syntax, a string, but it denotes a set
- Uses operators you already know, with different notation (and very slightly different meaning)
- E.g.,
  - 0* (the star operator)
  - 0*10* + 1*01* (concatenation and union)
  - etc.

Getting Regularly Expressive

- Examples:
  - \( L = \{ w \mid w \in \{0,1\}* \text{ begins and ends with same character} \} \)
  - \( L = \{ w \mid w \in \{0,1\}* \text{ contains the pattern 010} \} \)
- Think about this one by blocking / segmenting the strings to show soundness and completeness:
  - \( L = \{ w \mid w \in \{0,1\}* \text{ has an even # of 0s} \} \)
Recursive Definition of Regular Expressions

- Inductive on structure of expressions
- Base cases—regular expressions and what they denote:
  - ∅ denotes the empty language
  - ε denotes the language {ε} containing only the empty string
  - Languages containing singleton characters: For each symbol a in Σ, a is a regular expression denoting the language {a}
- Inductive cases/constructors—if r, s are reg. exps. that denote languages R and S (resp.), then the following are regular expressions:
  - (r + s) denotes union \( R \cup S \)
  - (rs) denotes concatenation of languages RS
  - (r*) denotes \( R^* \), the Kleene star operation on language R

Order of operations is as if they were addition, multiplication, and exponentiation operators.

This syntax can become very intuitive after you use it a while!

Our Goal:

- Theorem:
The languages represented by regular expressions are exactly the regular languages(!)
  - Theorem 1: For every regular expression r, there exists a corresponding FA
    • (an NFA, actually, but that’s not important)
  - Theorem 2: For every DFA M, there exists a corresponding regular expression
Theorem 1

• For all regular expressions r, there exists a corresponding FA
  – (an NFA, actually, but it doesn’t matter)

• Pf: construct it, using structural induction
  – (Do the inductive cases seem familiar?)

• Example / Exercise: Apply this construction to build an FA for regular expression \((0 + 1)^*\)

See Sipser, pg. 67, 59-62

Hmmm… that procedure works, but it seems inefficient. A machine for \((0 + 1)^*\) doesn’t need all those states! Is there a way to optimize / minimize the number of states in a machine?

Theorem 2

• For all DFA M, there exists a corresponding regular expression

• Pf: build it step by step (dynamic programming)
  – Use construct \(L\)
    to stand for the language of strings that drive M from state i to state j while going through no state higher than k
    • Be sure to start numbering states with 1, not 0!
  – Recursive definition:
    • What are all the \(L\) when \(k = 0\)?
    • When \(k \neq 0\), recursion based on case analysis:
      Strings either go through state k, or they don’t…
    • Inductively build the set of accepted strings of M as a reg. exp.