91.304 Foundations of (Theoretical) Computer Science

Chapter 1 Lecture Notes (Section 1.1: DFA's)

David Martin
dm@cs.uml.edu

With some modifications by Prof. Karen Daniels, Fall 2012

Chapter 1: Regular Languages

- Simple model of computation
- Input a string, and either accept or reject it
  - Models a very simple type of function, a predicate on strings:
    $$f : \Sigma^* \rightarrow \{0,1\}$$
  - See example of a **state-transition diagram**

Syntax of DFA

- A **deterministic finite automaton (DFA)** is a 5-tuple (Q,Σ,δ,q_0,F) such that
  1. Q is a *finite* set of states
  2. Σ ("sigma") is an alphabet (finite set)
  3. $\delta:Q\times\Sigma\rightarrow Q$ ("delta") is the transition function
  4. $q_0 \in Q$ ("q naught") is the start state
  5. $F \subseteq Q$ is the set of accepting states
- Usually these names are used, but others are possible as long as the role is clear

DFA syntax

- It is deterministic because for every input (q,c), the next state is a uniquely determined member of Q.

DFA computation

- This definition is different from but equivalent to the one in the text
- Let $M=(Q,\Sigma,\delta,q_0,F)$ be a DFA. We define the *extended transition function*

  $\delta^*:Q\times\Sigma^*\to Q$

  inductively as follows. For all $q\in Q$,

  $\delta^*(q,\varepsilon) = q.$

  If $w\in\Sigma^*$ and $c\in\Sigma$, let

  $\delta^*(q,wc) = \delta(\delta^*(q,w),c)$
- According to this definition, δ^***(q,x) is the state of the machine after starting in state q and reading the entire *string* x**
  - See example

Measuring DFA space complexity

- Space complexity: the amount of memory used
  - But a DFA has no extra memory; it only remembers what state it is in
  - Can’t look back or forward
  - So a DFA **always** uses the same amount of memory, namely the amount of memory required to remember what state it’s in
    - Needs to remember current element of Q
    - Can write down that number in $\log_2 |Q|$ bits

Language recognized by DFA

- The **language recognized by** the DFA M is written L(M) and defined as
  $L(M) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in F\}$
- Think of L() as an operator that turns a program into the language it specifies
  - We will use L() for other types of machines and grammars too
- Example 1.7, textbook p. 37

Example

- Let $L_2=\{x\in\{0,1\}^* \mid$ either x is the empty string, or the binary number x is a multiple of 2 $\}$ and build a DFA M_2 such that $L(M_2) = L_2$
  - Remember this means $L(M_2) \subseteq L_2$ **and** L_2 $L_2 \subseteq L(M_2)$
  - This is Example 1.9 from textbook, p. 38

Definition of regular languages

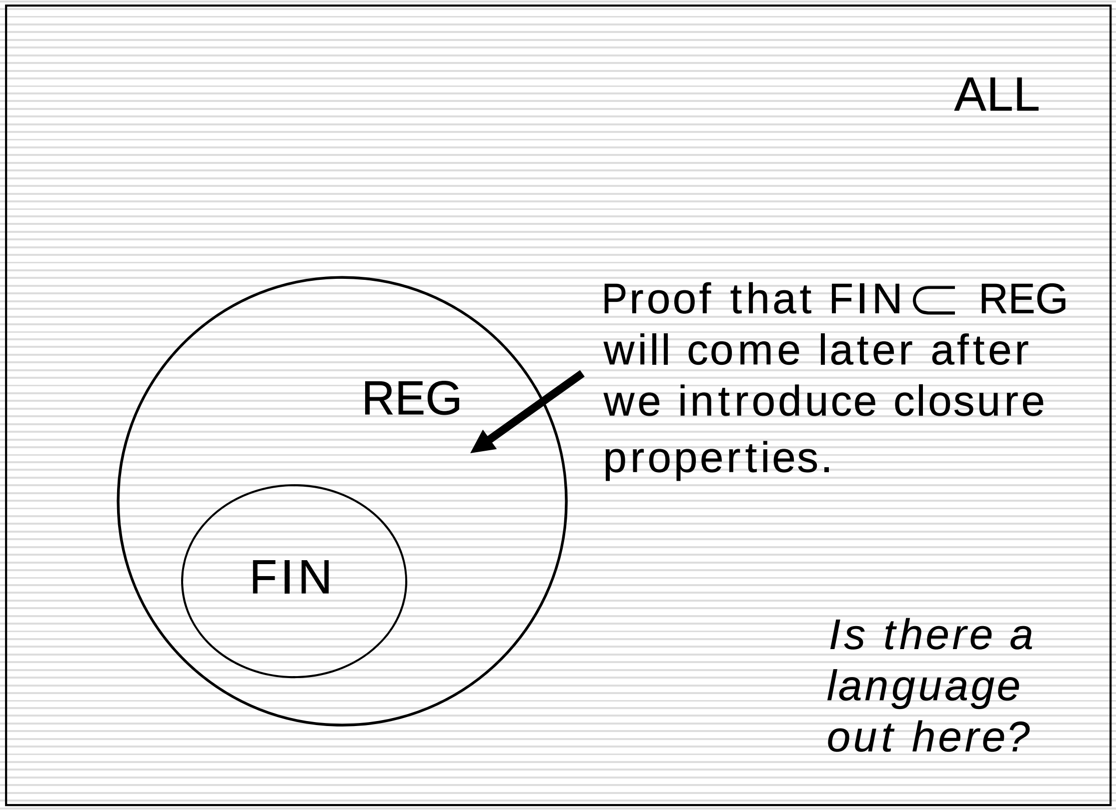
- A language L is **regular** if there exists a DFA M such that L = L(M)
- The **class of regular languages** over the alphabet Σ is called REG and defined

$$\mathrm{REG} = \{ L \subseteq \Sigma^* \mid L \text{ is regular} \}$$
$$= \{ L(M) \mid M \text{ is a DFA over } \Sigma \}$$

- Now we know 4 classes of languages: $\emptyset$, FIN, REG, and ALL (see Lecture 0)

Picture so far

Each point is a language in this Venn diagram



REG = L(DFA)

≠ FIN

"the class of languages generated by DFAs"

Problems

- For all $k \geq 1$, let $A_k = \{0^{kn} \mid n \geq 0\}$. Prove that $(\forall\ k \geq 1)\ A_k \in REG$
  - Solution is a scheme, not a single DFA
- (Harder) Build a DFA for $L_3 = \{x \in \{0,1\}^* \mid$ the binary number x is a multiple of 3 $\}$ *similar to Example 1.13*
- Build a DFA for $L_{3'} = \{x \in \{a,b\}^* \mid$ x does not contain 3 consecutive 'b's$\}$
- Build a DFA for $L_4 = \{x \in \{a,b\}^* \mid$ x contains an odd # of 'a's and an even # of 'b's$\}$ *homework from 2009*

Is REG reasonable?

- We should be able to combine computations as subroutines in simple ways
  - logical OR ($A \cup B$) $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ *example*
  - logical AND ($A \cap B$) *homework 2010*
  - concatenation ($A \cdot B$) and star (A^*)
    - hard to prove!! *motivation for NFA*
  - complement (A^c) *Problem 1.14 in textbook*
  - reversal (A^R) *homework 2010*
  - All above are easy to do as logic circuits
- *Closure under these language operations*