Regular Expressions and Finite Automata

CSCI 3136: Principles of Programming Languages

Agenda

- Announcements
- Readings:
  - Today: 2.2.1 (From optional text)
  - Next: 2.2.1
  - Note: I recommend using alternative texts for this part of the course:
  - E.g., Hopcroft et al, “Introduction to Automata Theory”
- Lecture Contents
  - Motivation
  - Regular Expressions
  - Deterministic Finite Automata
  - Nondeterministic Finite Automata

Specifying Regular Languages

- There are many ways to specify a regular language
  - {a,ab,abc}
  - $\{a\}^*$
  - $\{a,b,c\}^*$
  - $\{1^n0 \mid n > 0\}$
  - Set of all positive integers (base 10)
  - $\{\Sigma\Sigma^*@\ \Sigma\Sigma^*(.\ \Sigma\Sigma^*)^n \mid \Sigma = \{a,b,c,\ldots z\}, n \geq 0\}$
- Problems:
  - The specification is not standard
  - Hard for a program to interpret the specifications above.
- Does some of the notation above look familiar?

Regular Expressions

- Idea: Regular expressions (REs) are a concise way to specify regular languages
- **Theorem**: L is regular if and only if there is a regular expression R that specifies L
- Recursive definition:

  Base cases:
  - Ø defines the empty language (no words)
  - a, $a \in \Sigma$ defines the language {a}
  - ε defines the language {ε}

  Inductive step: If R, R_1, and R_2, are REs:
  - $R_1|R_2$ defines $L_1 \cup L_2$, (union) where R_i specifies L_i
  - R_1R_2 defines L_1L_2, (concatenation) where R_i specifies L_i
  - R^* defines L^*, (Kleene-*) where R specifies L

Examples of Regular Expressions

	<u>Corresponding Language</u>
• ab|c	{ab, c}
• a(b|c)	{ab, ac}
• a∗	$\{a\}^*$
• (a|b|c)∗	$\{a,b,c\}^*$
• 11∗0	$\{1^n0 \mid n>0\}$
• 0*[1 − 9][0 − 9]∗	Set of all positive integers
• aa∗bbb∗cccc∗	$\{a^ib^jc^k \mid i>0,j>1,k>2\}$
• 0 ∗ (100∗) ∗ (1|ε)	Binary strings with no adjacent 1s
• [a–z][a–z]∗@[a–z][a–z]∗(.[a–z][a–z]∗)∗	email address

Note: Notation [a − z] = (a|b|c|d|…|z)

Applications and History

- Applications:
  - Search (and replace)
  - editors, string manipulation libraries,
  - scanners
  - specification of tokens.
- History
  - Stephen Cole Kleene, 1956

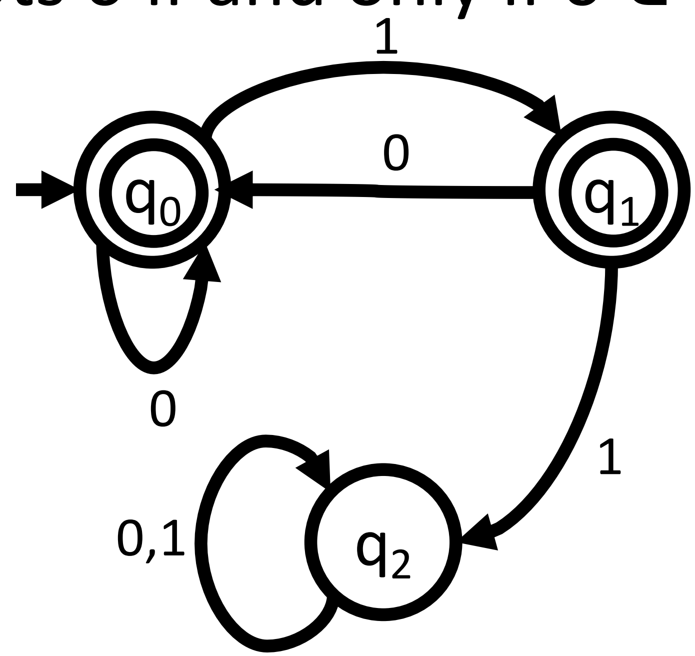
    “Representation of events in nerve nets and finite automata”
  - Ken Thompson developed editors: QUE, ed, grep
  - Used in awk, emacs, vi, lex, etc...
  - Henry Spencer, 1986, C regex library used in Tcl, Perl, etc...

How to Build a Scanner

- We now have a standard (machine-friendly) way to specify regular languages.
- So what?
- We now need a way to decide if a given string **σ** is in a given regular language **L**.
- How do we do this?
- We use a *Deterministic Finite Automata* (DFA).

Deterministic Finite Automata

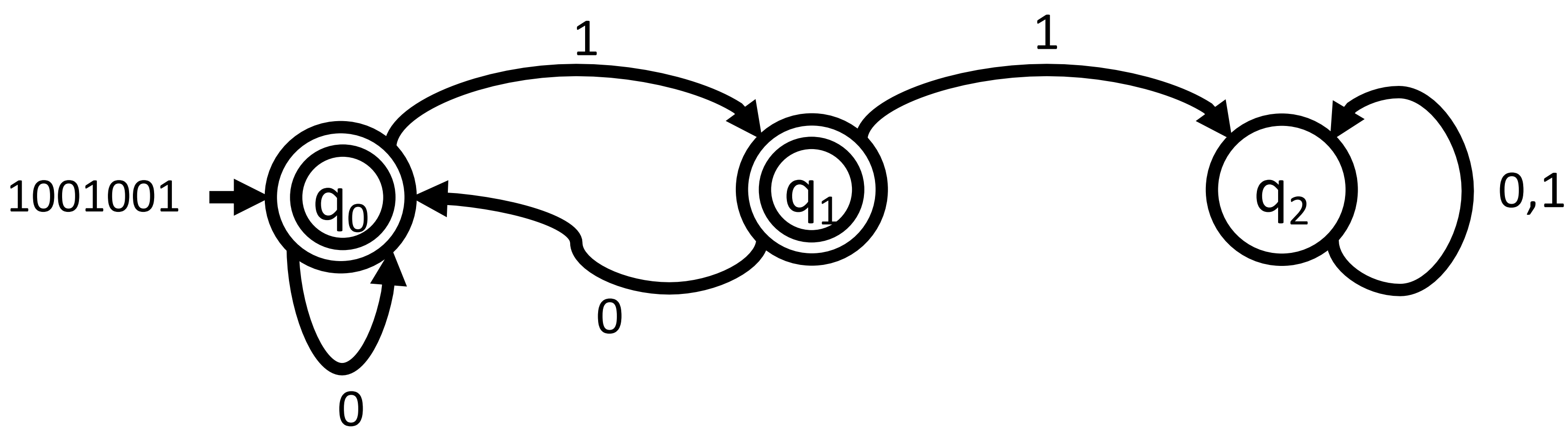
- A DFA *M* is a machine that
  - Takes a string $\sigma \in \Sigma^*$ as input
  - Either <u>*accepts*</u> σ if $\sigma \in L$
  - Or <u>*rejects*</u> σ if $\sigma \notin L$
- *M* <u>*recognizes*</u> L if it accepts σ if and only if $\sigma \in L$
- A DFA consists of:
  - set of states
  - *start* state
  - set of *final* states
  - transition function



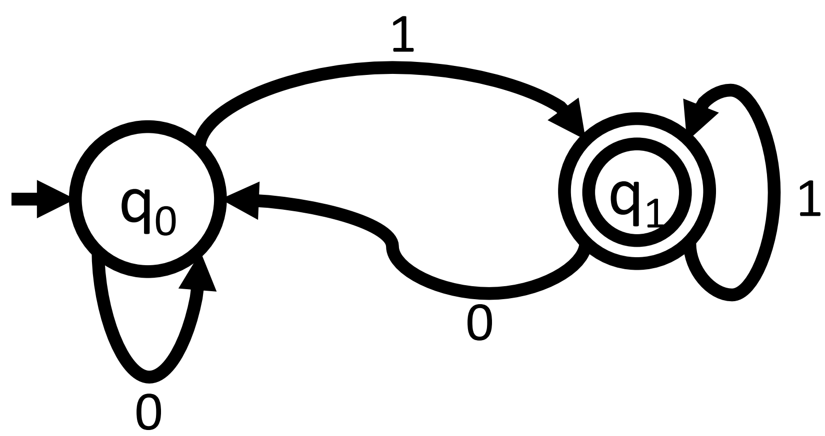
Operation of a DFA

A DFA

- Starts in the start *state*
- Reads in string σ one character at a time
- Computes the next state based on current state and character
- *Transitions* to the next state
- *Accepts* σ if and only if it is in a *final* state after reading σ.



What Language Does this DFA Recognize?



(0|1)*1

Formal Definition of a DFA

- A DFA is a 5-tuple $M = (Q,\Sigma,\delta,q_0,F)$
  - Q set of states
  - Σ alphabet
  - δ transition function (complete): $\delta : Q \times \Sigma \rightarrow Q$
  - q_0 start state, $q_0 \in Q$
  - F set of final states, $F \subseteq Q$
- A DFA *M* accepts a string $\sigma \in \Sigma*$ if and only if it is in a final state after reading σ.
- A DFA *M* recognizes language *L* if and only if it only accepts all the strings in *L*

  $L(M) = \{\sigma \in \Sigma^* \mid M \text{ accepts } \sigma\}$

Examples of DFAs

DFA that accepts all binary strings that have no consecutive 1s.

$M = (\Sigma, Q, \delta, q_0, F)$

- $\Sigma=\{0,1\}$
- $Q = \{q_0,q_1,q_2\}$
- $F = \{q_0,q_1\}$
- δ:

State	Symbol	New State
q_0	0	q_0
q_0	1	q_1
q_1	0	q_0
q_1	1	q_2
q_2	0	q_2
q_2	1	q_2

DFA that accepts L = (0|1)*1

$M = (\Sigma, Q, \delta, q_0, F)$

- $\Sigma=\{0,1\}$
- $Q = \{q_0,q_1\}$
- $F = \{q_1\}$
- δ:

State	Symbol	New State
q_0	0	q_0
q_0	1	q_1
q_1	0	q_0
q_1	1	q_1

More Examples

- $L \subseteq \{a, b\}^*$: all strings containing an odd number of b’s
- $L \subseteq [0 - 9]^*$: all integers divisible by 100
- $L \subseteq [a - z, @.]^*$: all valid email addresses
- $L \subseteq \{0, 1\}^*$: all binary numbers not divisible by 3

Nondeterministic Finite Automata (NFA)

- A DFA is deterministic in that it has a single transition for each symbol and state

  I.e., A DFA traces a single path for each input
- An NFA is like a DFA except it may have a choice of transitions for a given state and character.

  I.e., An NFA may trace multiple paths for an input
- Two kinds of nondeterministic choices:
  - **ε transitions:** transition to another state without reading a character
  - **multiple successor states:** multiple transitions to different states from same state and same character
- An NFA *accepts* a string σ if one of the paths ends in a final state

Example of an NFA

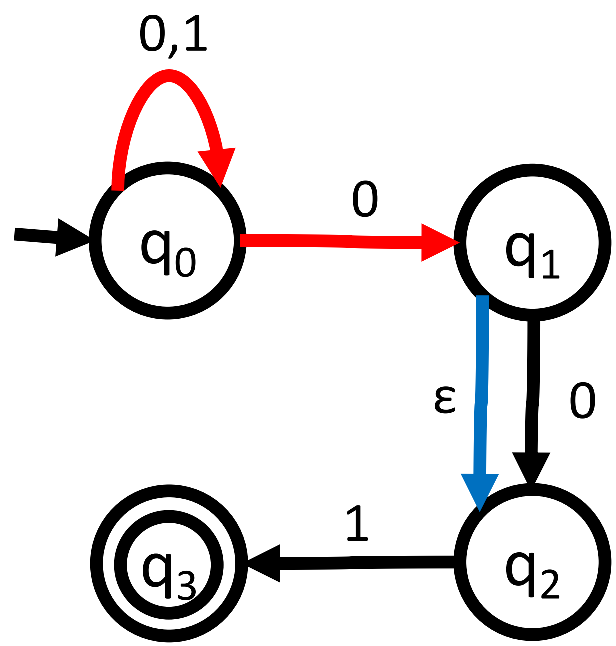
NFA that accepts binary strings ending in 01 or 001

$L = (0|1) * 0(1|01)$

$M = (\Sigma, Q, \delta, q_0, F)$

- $\Sigma = \{0,1\}$
- $Q = \{q_0, q_1, q_2, q_3\}$
- $F = \{q_3\}$
- δ:

State	Symbol	New State
q_0	0	q_0
q_0	0	q_1
q_0	1	q_0
q_1	0	q_2
q_1	ε	q_2
q_2	1	q_3



Formal Definition of an NFA

- An NFA is a 5-tuple $M = (Q,\Sigma,\delta,q_0,F)$
  - Q set of states
  - Σ alphabet
  - δ transition function: $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$
  - q_0 start state, $q_0 \in Q$
  - F set of final states, $F \subseteq Q$
- Every input σ induces a set of paths traced by δ as σ is read
- NFA M accepts a σ if and only if one of the paths ends in a final state
- NFA M *recognizes* $L(M) = \{\sigma \in \Sigma^* \mid M \text{ accepts } \sigma\}$
- Question: Are NFAs more powerful than DFAs?

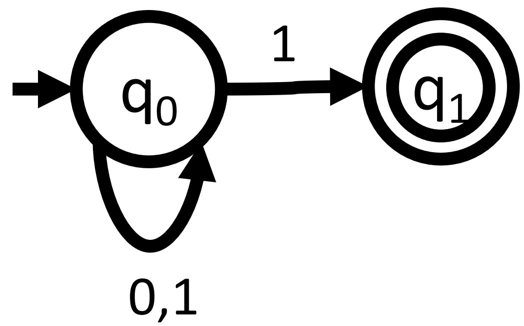
NFA Example 1

NFA that accepts L = (0|1)*1

$M = (\Sigma, Q, \delta, q_0, F)$

- $\Sigma=\{0,1\}$
- $Q = \{q_0,q_1\}$
- $F = \{q_1\}$
- δ:

State	Symbol	New State
q_0	0	q_0
q_0	1	q_0
q_0	1	q_1



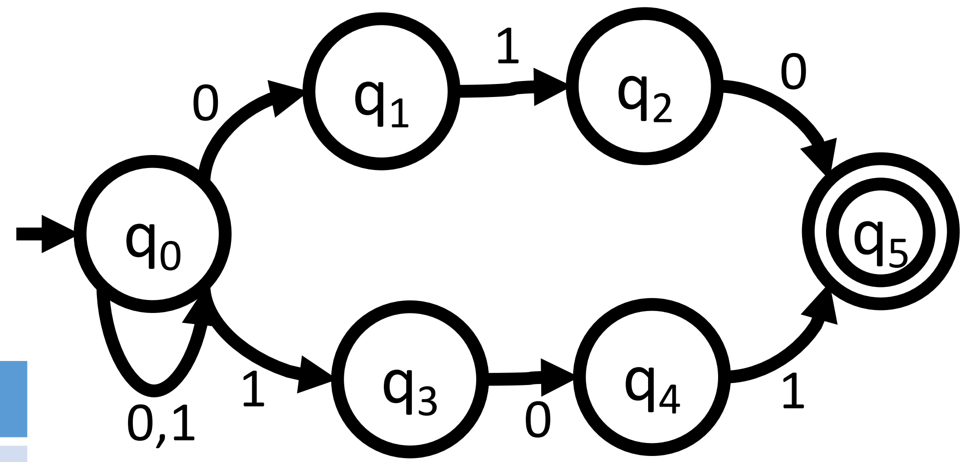
NFA Example 2

NFA that accepts L = (0|1)*(010|101)

$M = (\Sigma, Q, \delta, q_0, F)$

- $\Sigma=\{0,1\}$
- $Q = \{q_0,q_1,q_2\}$
- $F = \{q_0,q_1\}$
- δ:

State	Symbol	New State
q_0	0	q_0
q_0	1	q_0
q_0	0	q_1
q_0	1	q_3
q_1	1	q_2
q_2	0	q_5
q_3	0	q_4
q_4	1	q_5



More NFA Examples

- $L \subset \{a, b\}^*$: all strings containing an odd number of a's or b's
- $L \subset \{0, 1\}^*$: all binary strings containing the substring 101 or 010

Are these all the same?

- We have discussed a variety of specifications: RLs, RE, DFAs, NFAs
  - RLs: a class of languages
  - RE a way to specify RLs
  - DFAs: a way to implement scanners for RLs
  - NFAs: a simpler way to implement scanners for RLs
- Questions:
  - Are these all of equal power?
  - Are NFAs same as DFAs?
  - Do REs specify only regular languages?

Regular Languages Equivalence Theorem

- Thm: The following statements are equivalent:
    i. L is a regular language.
    ii. L is the language described by a regular expression.
    iii. L is recognized by an NFA.
    iv. L is recognized by a DFA.
- We will prove: (i) ≡ (ii) ≡ (iii) ≡ (iv)

Regular Languages are equivalent to Regular Expressions

- Every regular language can be specified by a regular expression.
- Every regular expression specifies a regular language.
- Idea: There is a one-to-one correspondence between definitions of RLs and REs
- Apart from notation, the recursive definitions are identical.

Operation	Regular Language	Regular Expression
Empty Language	$\emptyset$	$\emptyset$
Empty String	$\{\varepsilon\}$	ε
Single character	$\{a\}, a \in \Sigma$	a
Disjunction	$L_1 \cup L_2$	$R_1 \mid R_2$
Concatenation	L_1L_2	R_1R_2
Kleene-*	L^*	R*