Regular Expressions and Finite Automata

CSCI 3136: Principles of Programming Languages

Agenda

1. Motivation
2. Regular Expressions
3. Deterministic Finite Automata
4. Nondeterministic Finite Automata

Programming Language Translation

- Step 1: We must break the input (e.g., a text file) into a stream of "words" which represent the tokens of the language.
- We need some technique to specify these tokens.
- We started to examine Regular Languages because, for most (all?) programming languages, the set of tokens can be defined by a Regular Language.

Regular Languages

We use a recursive Definition ...

- Base Cases:
  - A (Empty language)
  - {ε} (Language consisting of the empty string)
  - {a}, a E Σ (Language consisting of one symbol)

- Inductive Step: If L1 and L2 are regular then so are
  - L1L2 = {στ | σ E L1,τ E L2}

  concatenation lets us have longer tokens

  - L1 ∨ L2 = {σ | σ E L1 λ σ E L2}

Specifying Regular Languages

- There are many ways to specify a regular language
  - {a,ab,abc}
  - {a}X
  - {a,ab,abb,abbb,...}X
  - {1n0 | n > 0}
  - Set of all positive integers (base 10)
- Problems:
  - The specification is not standard.
  - Hard for a program to interpret the specifications

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 of a RE:

  Base cases:

  - a, a E Σ is a RE
  - ε (the empty string) is a RE

  Inductive step: If R, R1, and R2, are REs:

  - R1 | R2 is a RE (union)

Regular Languages are the same as Regular Expressions

Regular Expression	Regular Language
	A (Empty language)
ε	{ε} (Empty String)
a, a Ε Σ	{a}, a Ε Σ
R1 | R2	L1 ν L2 = {σ | σ Ε L1 λ σ Ε L2}
R1R2	L1L2 = {στ | σ Ε L1,τ Ε L2}
RX	LX = {σi | σ Ε L, i ≥ 0}

Examples of Regular Expressions

	Corresponding Language
ab|c	{ab, c}
a(b|c)	{ab, ac}
aX	{a}X = {ε,a,aa,aaa,aaaa,...}
(a|b|c)X	{a,b,c}X = {a,b,c,aa,ab,ac,ba,bb,bc,...}
11X0	{1n0 |n > 0} = {10,110,1110,11110,...}
[1 − 9][0 − 9]X	Set of all positive integers
aaXbbbXccccX	{aibjck | i>0, j>1, k>2}
0X(100X)X(1|ε)	Binary strings with no adjacent 1s
[a−z][a−z]X@[a−z][a−z]X(.[a−z][a−z]X)X	email address

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

The Key Ideas

- Any character defines the language with that character
  - e.g., a defines {a}
- | means "or" and defines a language with what comes before the bar OR what comes after it
  - e.g., a|b defines {a,b}
- Concatenation defines the language that has two things side by side
  - e.g., ab defines {ab}
- * means zero or more copies of something

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...

Defining Tokens

- We can define a programming language's tokens using REs.
- if, for, while, ... keywords
- (+|-)(0-9)(0-9)* integers
- (a-zA-Z_$)(a-zA-Z_$0-9)* identifiers
- //(^\n)* comments
  - Note: (^\n) means anything except \n, i.e., "not a newline"
- "(^"\\)*" string
  - Note: (^"\\) means anything except " or \

How to Build a Scanner

- We now have a standard (machine-friendly) way to specify regular languages.
- So now 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).

Recognizers

- A "recognizer" for a language is a construct that determines if a given string is a member of the language.
- Every language has its own recognizer.
- For regular languages the recognizers are called "Finite State Automatas" (FA, FSA).
- That is for a language, L, there is a FAL such that for any string, s, used as input for FAL:

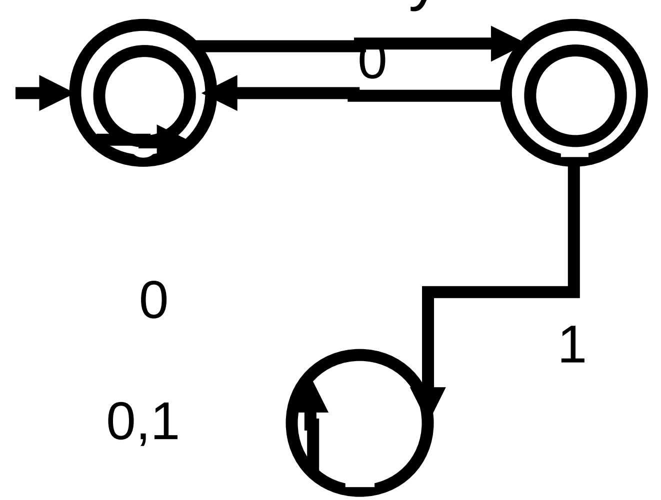
FAL(s) = true if s E L and FAL(s) = false otherwise

Deterministic Finite Automata

- A DFA *M* is a machine that
  - Takes a string σ E ΣX as input
  - Either <u>*accepts*</u> σ if σ E L
  - Or <u>*rejects*</u> σ if σ Z L

  M <u>*recognizes*</u> L if it accepts σ if and only if σ E L

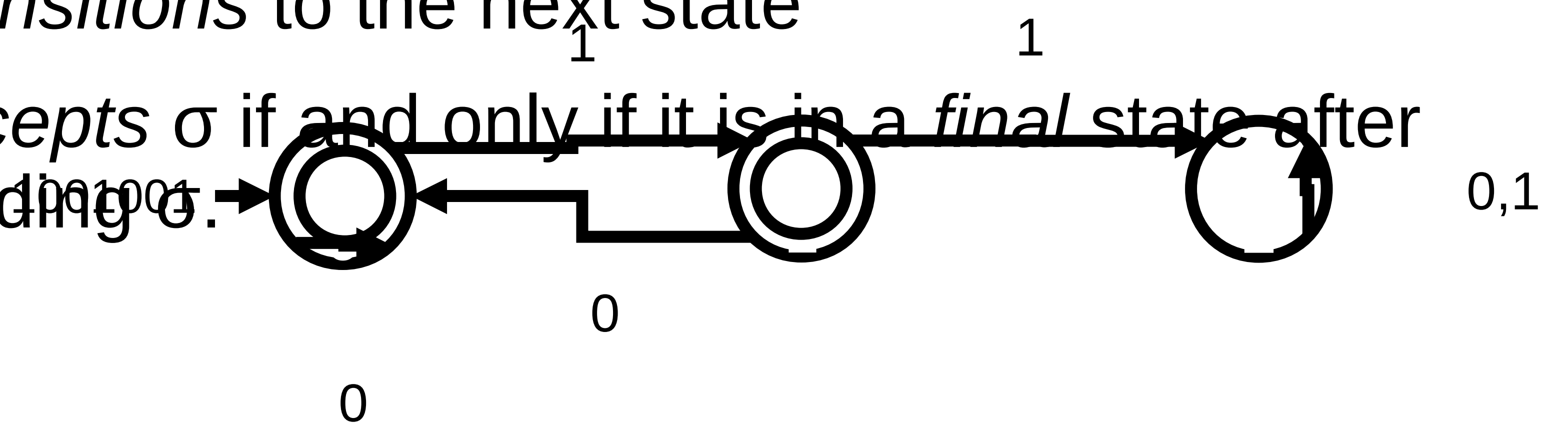
  A DFA consists of:
  - set of states
  - *start* state
  - set of *final* states
  - transition function



Operation of a DFA

A DFA

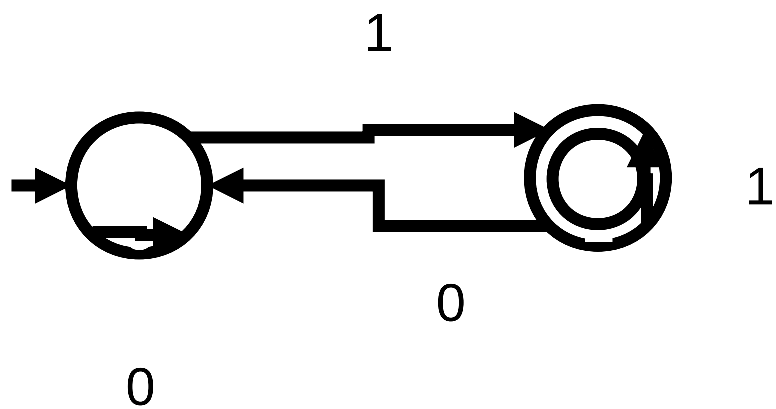
- Starts in the start *state*
- Reads in string σ one character at a time
- Computes the next state based on current state and character
- *Transitions* to the next state
- *Accepts* σ if and only if it is in a *final* state after reading σ.



Finite Automatas

- Set of states.
- The FA changes states when it "reads" a character from the input.
- Transitions are the arrows between states. Each transition is associated with a character.
- One state is special. This is the "start state" that the FA begins in before reading any input.
- Some states are "final." If the FA is in a final state when there is no more input to read, it accepts the input string as a member of L. If it is not in a final state, it must reject the string as

What Language Does this DFA Recognize?



(0|1)*1

Formal Definition of a DFA

- A DFA, M, is a 5-tuple: *M = (Q,Σ,δ,q0,F)*
  - Q set of states
  - Σ alphabet
  - δ transition function (complete): δ : Q × Σ → Q
  - q0 start state, q0 E Q
  - F set of final states, F ë Q
- A DFA *M* accepts a string *σ E ΣX* 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*

Examples of DFAs

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

M = (Σ, Q, δ, q0, F)

- Σ={0,1}
- Q = {q0,q1,q2}
- F =
- δ:

State	Symbol	New State
q0	0	q0
q0	1	q1
q1	0	q0
q1	1	q2
q2	0	q2
q2	1	q2

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

M = (Σ, Q, δ, q0, F)

- Σ={0,1}
- Q = {q0,q1}
- F =
- δ:

State	Symbol	New State
q0	0	q0
q0	1	q1
q1	0	q0
q1	1	q1

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

Example of an NFA

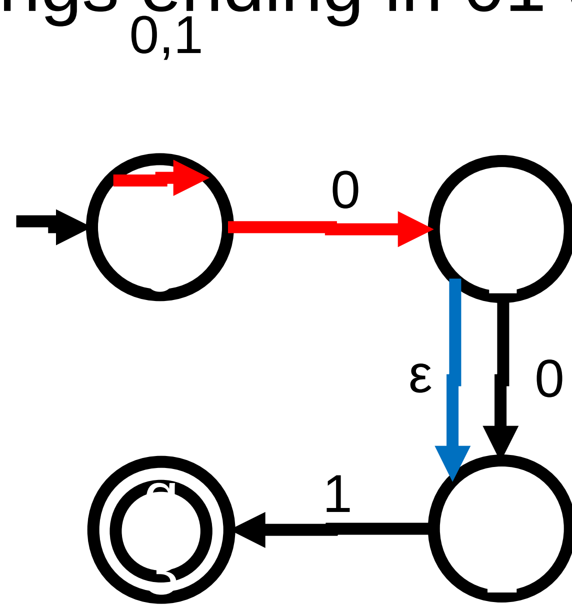
NFA that accepts binary strings ending in 01 or 001

L = (0|1) X 0(1|01)

M = (Σ, Q, δ, q0, F)

- Σ=
- Q
- F
- δ:

State	Symbol	New State
q0	0	q0
q0	0	q1
q0	1	q0
q1	0	q2
q1	ε	q2
q2	1	q3



Formal Definition of an NFA

- An NFA is a 5-tuple *M = (Q,Σ,δ,q0,F)*
  - Q set of states
  - Σ alphabet
  - δ transition function: δ : Q × (Σ ν {ε}) → 2Q
  - q0 start state, q0 E Q
  - F set of final states, F ë 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

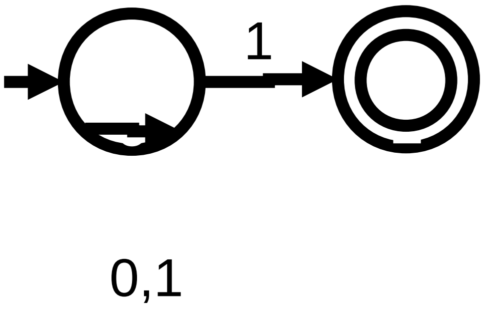
NFA Example 1

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

M = (Σ, Q, δ, q0, F)

- Σ={0,1}
- Q = {q0,q1}
- F = {q1}
- δ:

State	Symbol	New State
q0	0	q0
q0	1	q0
q0	1	q1



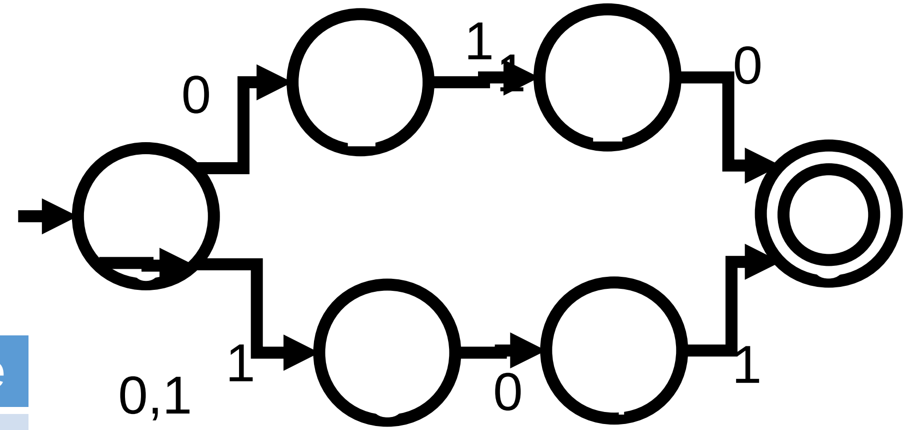
NFA Example 2

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

M = (Σ, Q, δ, q0, F)

- Σ={0,1}
- Q = {q0,q1,q2}
- F =
- δ:

State	Symbol	New State
q0	0	q0
q0	1	q0
q0	0	q1
q0	1	q3
q1	1	q2
q2	0	q5
q3	0	q4
q4	1	q5



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

- Theorem: 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

Operation	Regular Language	Regular Expression
Empty Language	A	A
Empty String	{ε}	ε
Single character	{a}, a E Σ	a
Disjunction	L1 ν L2	R1|R2
Concatenation	L1L2	R1R2
Kleene-closure	LX	R*