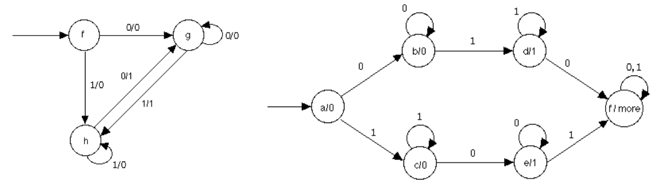
Regular Expressions and NFAs

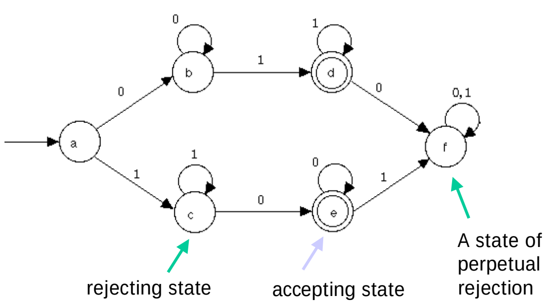
September 7, 2001

Review: Transducers and Classifiers



- Input is given as a sequence.
- At each step, we have a state change
- Outputs may depend on the transition or on the state

Review: An Acceptor



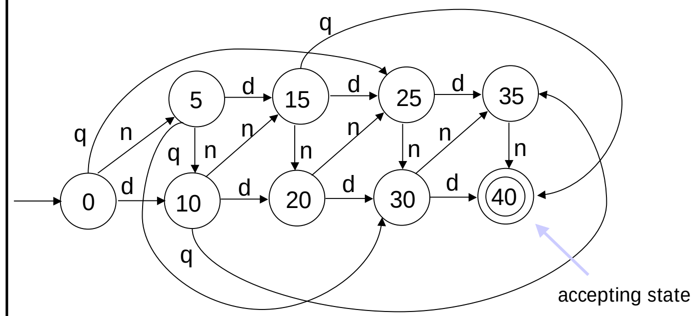
Equivalence

- Every classifier can be represented as a "gang" of acceptors
  - e.g., an acceptor for sequences with no edges, an acceptor for sequences with exactly one edge, and an acceptor for sequences with "many" edges
- Every transducer can be represented as an "equivalent" classifier
- Therefore, studying acceptors, the simplest model, yields insight for all finite-state machines

What can an Acceptor Do?

- Consider the Pepsi Machine near B101.
- Coins of 5, 10, and 25 cents can be entered
  - Referred to by input symbols n, d, q, respectively.
- Accepts when a total of 40 cents (or more) has been entered.

Pepsi Acceptor (partial)



This specification is “partial” because we have not said what happens when q is input to states 20, 25, 30, 35, etc. (The default is that this means going to an always-rejecting state.)

Languages

- A convenient way to characterize an acceptor is by its **language**, the set of all input sequences it accepts.
- Typically the language will be infinite, although there are also cases of finite languages.

Language Examples

- The set of all strings over `{0,1}` such that every `0`, if followed by any symbol, is followed by a `1`.
- The set of all strings over `{0,1}` such that the number of symbols is a multiple of 4.
- The set of all binary numerals that, m.s.b. first, are multiples of 3:
  `{0,11,110,1001,1100,1111, …}`
  (corresponding to 0, 3, 6, 9, 12, 15, ...)

Language Examples

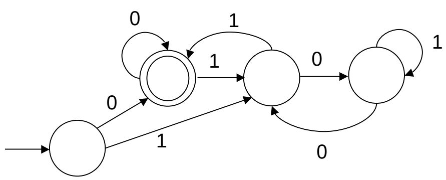
- The set of all strings over {0, 1} such that every 0, if followed by any symbol, is followed by a 1.

Language Examples

- The set of all strings over {0, 1} such that the number of symbols is a multiple of 4.

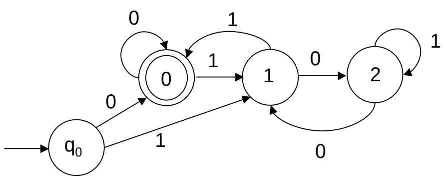
Language Examples

- The set of all binary numerals that, m.s.b. first, are multiples of 3:
  {0, 11, 110, 1001, 1100, 1111, ...}
  (corresponding to 0, 3, 6, 9, 12, 15, ...)



Correctness of the Multiples-of-3 Example

Let n be the numeral input so far. For every n, there is a k and an r < 3, such that n = 3k+r (r = n%3). States, other than the starting state, are identified with with r.

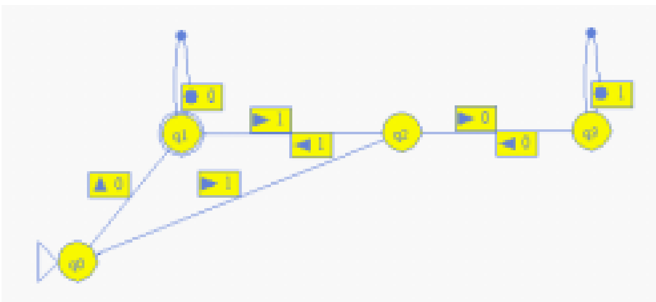


Inputting a 0 takes n to 2n, and
inputting a 1 takes n to 2n+1.
So inputting a 0 takes 3k+r to 6k+2r, while
Inputting a 1 takes 3k+r to 6k+2r+1.

r	(2r) % 3	(2r+1) % 3
0	0	1
1	2	0
2	1	2

Acceptor as Entered in JFLAP

(Applet-based tool by Susan Rodger:
http://www.cs.duke.edu/~rodger/tools/jflap)

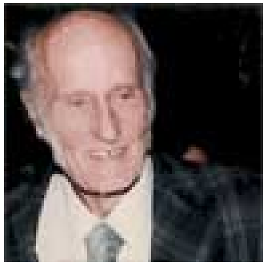


Characterization of Finite-State Machines by "Regular Expressions"

- **Regular expressions** are a *machine-independent* way of specifying a language.
- They are often used in textual **pattern-matching** applications.
- They are closely related to **grammars**, but the form of recursion is limited to "iterative" forms only.

Regular Expressions

- Discovered by the mathematical-logician S.C. Kleene (1909-1994, Prof. at U. of Wisconsin) in studying "nerve nets" in 1956.
- Kleene was also a principal developer of the field of recursion (computability) theory



About Mr. Kleene

Kleene pronounced his last name /klay'nee/.

/klee'nee/ and /kleen/ are extremely common mispronunciations. His first name is /steev'n/, not /stef'n/.

His son, Ken Kleene <kenneth.kleene@umb.edu>, wrote: "As far as I am aware this pronunciation is incorrect in all known languages. I believe that this novel pronunciation was invented by my father."

Regular Expressions Defined

- A regular expression (RE) is always defined with respect to a finite alphabet of symbols, Σ.
- The definition is *inductive*:
- Basis:
  - Any symbol in Σ by itself is an RE.
  - The special symbol λ is an RE
    - Often ε is used instead of λ.
  - The special symbol ∅ is an RE.
- Induction step: If R and S are RE's, then so are:
  - `RS`
  - `R | S`
  - `R*`

Regular Expression Examples

- Take Σ = `{0, 1}`.
- Basis:
  - Any symbol in Σ is an RE: `0` `1`
  - The special symbol λ is an RE: λ
  - The special symbol ∅ is an RE: ∅
- Induction step: If R and S are RE's, then so are:
  - RS: `00` `01` `0001` `1010` `1(00|11)*0`
  - R | S `00|11` `0 | 1 | l`
  - R* `0*` `01*0` `(00 | 11)*`

Meaning of Regular Expressions(1)

- Each regular expression R denotes a **language** (set of strings) L(R) over its alphabet:
- Basis:
  - A symbol σ in Σ denotes the language of one string of one letter: L(σ) = {σ}.
  - The special symbol λ denotes the empty string (no letters): L(λ) = {λ}.
  - The special symbol ∅ denotes the empty set (no strings): L(∅) = ∅.

Meaning of Regular Expressions (2)

- Induction step: Suppose R and S are regular expressions and L(R) and L(S) have been defined. Then (concatenation of two strings)
  - $L(RS) = \{xy \mid x \in L(R) \text{ and } y \in L(S)\}$
  - $L(R \mid S) = L(R) \cup L(S)$
  - $L(R^*) = \{\lambda\} \cup L(R) \cup L^2(R) \cup L^3(R) \quad \ldots$

    where $L^k(R)$ means the language formed by concatenating k strings, each one from L(R).

Note on Precedence in Regular Expressions

- It is common to omit parentheses.
- The binding order is:

  * binds most tightly

  juxtaposition is next

  | binds most weakly

Examples of RE's, with Meanings

- `0101`
  The set of one string "0101".
- `0101 | 1010`
  The set of two strings, "0101" and "1010".
- `1(0101 | 1010)0`
  The set of two strings, "101010" and "110100".
- `01*0`
  The set of strings that begin and end with 0 and contain a continuous run of 1's (of length 0 or more).

Examples of RE's, with Meanings

- `0*1*`
  The set of strings in which no 1 is followed by a 0.
- `0*1*0*1*`
  The set of strings in which at most one 1 is immediately followed by a 0.
- `0*(100*)*`
  The set of strings in which every one is followed by a 0.

Try These

`(0*10*1)*0*`

`((0 | 1)(0 | 1))*`

`0*10*  |  1*01*`

`(0*1*)*`

Give Regular Expressions (over alphabet {0, 1}) for

- The set of strings with at most two 0's
- The set of strings with more than two 0's
- The set of strings in which 0's and 1's strictly alternate

Kleene's Remarkable Result

- The languages accepted by finite-state acceptors and the languages denoted by regular expressions are the same thing!

In other words:

- Part I: The language accepted by any finite-state acceptor can be expressed as a regular expression.
- Part II: For every regular expression, there is a finite state acceptor that accepts the language denoted by the expression.

Non-Deterministic Finite Automata

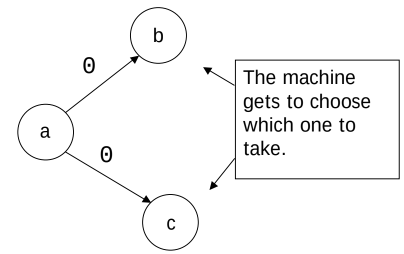
- The easiest way to prove part II is to appeal to the idea of a non-deterministic finite-state acceptor (NFA):
  - Part IIa: For every regular expression R, there is an **N**FA that accepts L(R).
  - Part IIb: For every NFA N there is a (deterministic) finite-state acceptor that accepts L(N).

Non-Deterministic Finite Automata

- A non-deterministic finite-state acceptor (NFA) is a finite-state acceptor with free-choice of transitions:
  - A given state may have more than one transition leaving with the same symbol, or
  - A state may be left spontaneously via a λ transition.

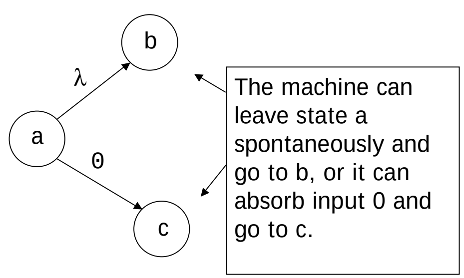
Non-Deterministic Finite Automata

- A given state may have more than one (or even no) transition leaving with a given symbol.



Non-Deterministic Finite Automata

- A state may be left spontaneously via a λ transition.



Acceptance Notion for NFAs

- An NFSA accepts an input sequence iff there is ***some*** path from ***some*** initial state (an NFSA can have more than one) to ***some*** accepting state.