Automata, Grammars and Languages

Discourse 03

Finite Automata

Finite Automata / Switching Theory

(CS) /

(CE)

• Boolean operators / Gates (Elem. Switching Ops)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x \land y</th>
<th>x \lor y</th>
<th>x \oplus y</th>
<th>\lnot x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Boolean Functions / Combinatorial Circuits

• Circuit

H half adder

F full adder

Lecture 03
Boolean Functions / Comb. Circuits (cont’d)

• Table representing $F$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$z_1$</th>
<th>$z_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Boolean Functions / Comb. Circuits (cont’d)

• Equations representing $F$:
  - $z_1 = g_1(x) = x_1 \oplus x_2 \oplus x_3$
  - $z_2 = g_2(x) = (x_1 \oplus x_3) \land (x_1 \land x_2)$

• General scheme ($n$ inputs, $m$ outputs)
  \[
  \overline{z} = g(\overline{x})
  \]
  \[
  (z_1, z_2, \ldots, z_m) = g(x_1, x_2, \ldots, x_n) =
  \left( g_1(x_1, x_2, \ldots, x_n), \ldots, g_m(x_1, x_2, \ldots, x_n) \right)
  \]

Finite Automata / Sequential Circuits

• Add "memory" elements = delay elements

  $y(i) \xrightarrow{\text{combinatorial circuit}} y(i + 1)$

  $\overline{x} \xrightarrow{f} \overline{z}$

  $\overline{z}(i) = f(\overline{x}(i), \overline{x}(i-1), \overline{x}(i-2), \ldots)$

• Finite # of delay elements possible $\Rightarrow \exists d$

  $\overline{x}(i) = f(\overline{x}(i), \overline{x}(i-1), \ldots \overline{x}(i-d))$
Finite Automata / Sequential Circuits

Ex: sequential adder: add 2 binary numbers; low order bits received first

(a) sequential net (circuit):

\[
\begin{align*}
x_i(i) & \rightarrow \text{full adder } F \\
x_i(i) & \rightarrow z_i(i) \\
y_i(i + 1) & \rightarrow y_i(i)
\end{align*}
\]

(b) Next-State & Output Equations:
\[
y_i(i + 1) = (x_i(i) \oplus x_i(i)) \land y_i(i) \lor (x_i(i) \land x_i(i))
\]
\[
z_i(i) = x_i(i) \oplus x_i(i) \lor y_i(i)
\]

(c) Transition Table:

<table>
<thead>
<tr>
<th>(q_0)</th>
<th>(00)</th>
<th>(01)</th>
<th>(10)</th>
<th>(11)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>(q_0)</td>
<td>(q_1)</td>
<td>(q_0)</td>
<td>(q_0)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>(q_0)</td>
<td>(q_0)</td>
<td>(q_0)</td>
<td>(q_0)</td>
</tr>
</tbody>
</table>

(d) State Diagram:
Finite Automata / Sequential Circuits

- (e) Finite-State Transducer (Mealy Machine)
  - A 5-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0)$ where
    - $Q = \{ q_0, q_1 \}$ finite set of states
    - $q_0$ start state
    - $\Sigma = \{ (0,0), (0,1), (1,0), (1,1) \}$ input alphabet
    - $\Gamma = \{ 0, 1 \}$ output alphabet
    - $\delta: Q \times \Sigma \rightarrow Q \times \Gamma$ transition/output function
      - $\delta(q_0, (0,0)) = (q_1, 0)$
      - $\delta(q_0, (0,1)) = (q_1, 1)$
      - $\delta(q_0, (1,0)) = (q_1, 0)$
      - $\delta(q_0, (1,1)) = (q_1, 0)$
    - $e \in \mathbb{C}$

General Sequential Network

- $B = \{ 0, 1 \}
- \delta$ is a boolean function
- Input: $x_1, \ldots, x_n$
- Output: $y_1, \ldots, y_n$
- State space: $B^n$
- Transition function:
  - $\delta(q_i, (i, 0)) = (q_i, 0)$
  - $\delta(q_i, (i, 1)) = (q_i, 1)$

Three Types of Automata

- $a_i \in \Sigma$
- Transducer
-Recognizer (acceptor)
- Enumerator (generator)
Machines that Recognize

- Detection of an "event", i.e., a pattern in input
- Recognition of just those words in some language \( L \)
- Definition of a language
- Ex: detect \( abab \)—all non-overlapping occurrences

Ex: C Comments /* ... */

- Filter in the lexical scanner (transducer)
- Recognizer

Finite Automaton (Finite State Machine, FSA)

- Defn 1.5: A (deterministic) finite automaton is a 5-tuple \( M = (Q, \Sigma, \delta, q_0, F) \)
- \( Q \) is a finite set, the states
- \( \Sigma \) is a finite set, the alphabet
- \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function
- \( q_0 \in Q \) is the start state
- \( F \subseteq Q \) is the set of accepting (final) states
- Ex: \( M_0 = (Q_0, \Sigma, \delta, q_0, F_0) = (q_0, q_1, q_2) \)
  - \( Q = \{q_0, q_1, q_2\} \)
  - \( \Sigma = \{a, b\} \)
  - \( F = \{q_2\} \)
  - \( \delta(q_0, a) = q_1 \)
  - \( \delta(q_1, b) = q_2 \)
  - \( \delta(q_2, a) = q_2 \)
How FA Compute

- FA $M = (Q, \Sigma, \delta, q_0, F)$ is a finite structure—like a program—fixed and static
- Need to define the behavior of $M$ on input $w$
  - Sequence of configurations
  - Like trace of a program on given data
  - Dynamic and input-dependent
- Ex: start $M$ on input $w = ababa$ Look at sequence of “moves” determined by the transition function:
  - $(q_0, ababa) \rightarrow (q_1, baba)$
  - $(q_1, ba) \rightarrow (q_2, a) \rightarrow (q_3, \varepsilon)$
  - $(q_3, ababa) \rightarrow (q_4, \varepsilon)$
- Since in accepting state when input exhausted, $w$ is recognized by $M_0$

www.jflap.org

JFLAP is a package of graphical tools which can be used as an aid in learning the basic concepts of Formal Languages and Automata Theory.

Info on JFLAP

- Downloads
- Tutorial
- Lectura (linux) install
  - cd /usr/local/jflap
  - java -jar JFLAP.jar
- X11 forwarding (graphics)
  - ssh -X lectura
How FA Compute (cont’d)

- Given a FA  $M = (Q, \Sigma, \delta, q_0, F)$
- Defn: configuration of $M$ is an element of $Q \times \Sigma^*$
- Defn: yields in one step (or moves) relation $\overset{*}{\Rightarrow}$ between configurations is defined by
  
  $$(q, aw) \overset{a}{\Rightarrow} (q', w') \iff \delta(q, a) = q'$$

  where $a \in \Sigma, w \in \Sigma^*, q, q' \in Q$.

- Notes: $\Rightarrow$ is a function, since $\delta$ is.
  $(q, \varepsilon) \overset{*}{\Rightarrow}$ is undefined.

- Defn: yields is the relation $\overset{*}{\Rightarrow}$
  - Means "moves in zero or more steps to"
  - $(q, w) \overset{*}{\Rightarrow} (q', w)$

- Defn: A string $w$ is recognized (accepted) by $M \iff (q_0, w) \overset{*}{\Rightarrow} (F, \varepsilon)$

How FA Compute (cont’d)

- Defn: The language recognized (accepted) by $M$ is
  
  $L(M) = \{ w : M \text{ accepts } w \}$
  
  $= \{ w : (\exists f \in F) (q, w) \overset{*}{\Rightarrow} (f, \varepsilon) \}$

- Defn 1.16: A language $S$ is regular iff there is some FA that recognizes it, i.e., $L(M) = S$ \iff $S \in L(FA)$

- Ex: In FA $M_0$
  
  $(q_0, a) \overset{*}{\Rightarrow} (q_0, \varepsilon)$
  
  $(q_0, a) \overset{a}{\Rightarrow} (q_1, a)$
  
  $(q_0, ab) \overset{a}{\Rightarrow} (q_1, b)$
  
  $\vdots$
  
  $(q_i, (ab)^i a) \overset{a}{\Rightarrow} (q_i, \varepsilon)$

  $\vdash L(M_0) = \{ (ab)^i a : i \geq 0 \}$

Example:

- Coin checker for 30¢ coffee.
  
  $\Sigma = \{n, d, q\}$

  ![Diagram of coin checker FA]

  - Make change for $i - 30$¢ & vend coffee

  $i = \text{make change for } i - 30\text{¢ }\& \text{ vend coffee}$

  $0, 10, 20, 25, 30, 40, 50$
Regular Operations & Regular Expressions

- The regular operations on languages are:
  - union (\( \cup \)), concatenation (\( \cdot \)) and Kleene star (\( * \)).
- So called because the class of regular languages are closed under them—i.e., applying these operators to regular languages results in a regular language. (We will prove these closure results later.)
- In fact, these three operations (\( \cup \), \( \cdot \), \( * \)) actually characterize what it means to be a regular language: any regular language can be built up from alphabet symbols and a finite number of these regular operations.
- This motivates the notion of regular expression: a sequence of symbols, like an arithmetic expression, that defines a regular language using regular ops.

Regular Expressions

- A syntax for describing sets of strings (languages)
  - Terse
  - Eliminates fussy \( \{ \cdot \} \)
  - Reminiscent of arithmetic expressions
  - Obeys some useful "algebra", e.g., \( (E^*F)^* = (E+F)^* \)
- Syntax for regular expressions over \( \Sigma, +, \cdot, * , (, ) \)
  - \( E \rightarrow (E+E) \) (text uses \( \cup \) not +; some authors use \( | \) )
  - \( E \rightarrow (EE) \) (usually suppress the \( \cdot \) in \( E \cdot E \))
  - \( E \rightarrow (E^*) \)
  - \( E \rightarrow \epsilon \) (some authors use \( \lambda \) )
  - \( E \rightarrow \emptyset \)
  - \( E \rightarrow a \) for each \( a \) in \( \Sigma \)
  - suppress (\( (, ) \) where possible: \( (a+b)^*a \) not \( ( (a + b)^* ) \cdot a \) )

Regular Expressions (cont’d)

- Meaning rules for the syntax
  - The meaning (denotation) of an expression, \( L(E) \), is a set of strings (a language)
- Rules

\[
\begin{align*}
\text{expression } E & \quad \text{language } L(E) \\
\emptyset & = \{ \} \\
a & = \{a\} \\
\epsilon & = \{\epsilon\} \\
(E+F) & = L(E) \cup L(F) \\
(EF) & = L(E) \cdot L(F) \\
(E^*) & = L(E)^* \\
\end{align*}
\]
Reg. Expr.: Examples, Equivalence(=)

- \((a+b)^*a\)  
- \((a^b)^*\)  
- \(=(a+b)^*\)  
- \((a+b)^*a(a+b)^*a(a+b)^*\)  
- \((b^a)^{ab^*ab^*}\)  

\[w \in \Sigma : w \text{ has } \geq 2 \text{ a's} \]

PASCAL unsigned numbers. \(d=\{0,1,\ldots,9\}\)

- \(dd(\varepsilon+dd)(\varepsilon+E(\varepsilon+E)dd)\)  
- \(a\Sigma^*a+b\Sigma^*a+a+b\)  

Defn: \(E=F \iff L(E)=L(F)\)

- \(\varepsilon=\varepsilon\)  
- \(E\cdot F=\varepsilon\)  
- \(F=\varepsilon\cdot F \)

Nondeterminism

- Real computing devices are deterministic: the current configuration and instruction determines the next configuration. The \(\vdash\) relation is a function.

- Why the concept of nondeterminism?
  - Provides powerful, economical descriptive ability
  - Provides a way to specify languages without over-specifying and complex handling of cases
  - Can be algorithmically converted to a deterministic description (at the sacrifice of some economy and with added complexity)
  - Generalization of determinism

Ex: \(abab\) occurs somewhere in \(w\): \(\ldots abab \ldots\)

- \(w \in \Sigma^*\) has penultimate symbol \(b\): \(w = \ldots b \)

- \(w \in \Sigma^*\) has \(\geq 2\) a's: \(w = \ldots a \ldots a \ldots\)
**ε-Moves Can Be Useful**

- SNOBOL arithmetic constants (no floating ε)
  - Use to specify "optional characters" like Unix command line [opt]

```
ε
-ε
d
```

**Nondeterministic Finite Automaton**

- Defn 1.5: A nondeterministic finite automaton is a 5-tuple
  
  \[ M = (Q, Σ, δ, q_0, F) \]

  - \( Q \) is a finite set, the states
  - \( Σ \) is a finite set, the alphabet
  - \( δ : Q × (Σ ∪ \{ε\}) \rightarrow 2^Q \) transition function
  - \( q_0 \in Q \) is the start state
  - \( F \subseteq Q \) is the set of accepting (final) states

- Ex: \( M_1 = (Q, Σ, δ, q_0, F) \quad Q = \{q_0, q_1\} \quad Σ = \{a, b\} \quad F = \{q_1\} \)

  \[
  δ(q_0, a) = \{q_0, q_1\}
  \]

  \[
  δ(q_0, b) = \{q_1\}
  \]

**DFA vs NFA**

- DFA δ
  - For each state \( q \) and input symbol \( a \), there is exactly one choice of new state (or no transition is defined at all). Each transition "consumes" an input symbol
  - Special case of NFA!

- NFA δ
  - There may be multiple choices for the same input symbol
  - There may be ε-moves that do not "consume" an input character
  - There can be "chains" of ε-moves
  - ε-moves can create even more choice for the next input character
How NFA Compute

• Given a NFA \( M = (Q, \Sigma, \delta, q_0, F) \)
• Defn: configuration – \((q, w) \in Q \times \Sigma^*\)
• Defn: yields in one step (or moves) relation between configurations
  \((q, aw) \rightarrow (q', w) \Leftrightarrow q' \in \delta(q, a)\)
  \((q, w) \rightarrow (q', w) \Leftrightarrow q' \in \delta(q, \varepsilon)\) (\(\varepsilon\)-move)
• Defn: yields =
  • Means "COULD move in zero or more steps to"
• Defn: \( w \) is recognized (accepted) by \( M \) \( \Leftrightarrow \)
  \((\exists f \in F) \ (q_0, w) \rightarrow^* (f, \varepsilon)\)
  • Same as before, but has the meaning "if there exists some sequence of moves from the start config to some accepting config"

How NFA Compute (cont’d)

• Defn: The language recognized (accepted) by \( M \) is
  \( L(M) = \{ w : M \text{ accepts } w \} \)
  \( = \{ w : (\exists f \in F) \ (q_0, w) \rightarrow^* (f, \varepsilon) \} \)
• Ex: In NFA \( M \)
  \((q_1, aabbba) \rightarrow^* (q_1, \varepsilon)\)
  • This provides no "evidence" that aabbba is accepted (or not)
  • However, also via a separate computation sequence:
    \((q_1, aabbba) \rightarrow^* (q_1, \varepsilon)\)
    • And so aabbba is recognized!

“Tree” of Computations

• Ex: NFA \( M \)
  \((q_1, aabbba)\)
    \((q_1, abba)\)
      \((q_1, bba)\)
        \((q_1, ba)\)
          \((q_1, a)\)
            \((q_1, \varepsilon)\)
              null "evidence"
    \((q_1, bba)\)
      \((q_1, bba)\)
        \((q_1, ba)\)
          \((q_1, a)\)
            \((q_1, \varepsilon)\)
              \(q_1 \in F\) \(\exists\) accepting Computation \(\Rightarrow w \in L(M)\)
Computation Tree: Example

- Ex: $L = \{ w : w$ begins & ends same $\}$

\[
\begin{align*}
(1, \text{ababa}) & \quad 11 \quad \text{X} & \quad \text{accept } \text{ababe} \\
(2, \text{babab}) & \quad \text{X} & \quad \text{3 paths to } F \\
(3, \text{aba}) & \quad \text{X} & \quad \text{x} = F \\
(2, \text{a}) & \quad \text{X} & \quad \text{x} = F \\
(4, \text{ε}) & \quad \text{X} & \quad \text{x} = F \\
(5, \text{ε}) & \quad \text{3εF} & \quad \text{3εF} \\
(6, \text{ε}) & \quad \text{εF} & \quad \text{εF} \\
(7, \text{ε}) & \quad \text{εF} & \quad \text{εF} \\
(8, \text{ε}) & \quad \text{εF} & \quad \text{εF} \\
(9, \text{ε}) & \quad \text{εF} & \quad \text{εF} \\
(10, \text{ε}) & \quad \text{εF} & \quad \text{εF} \\
(11, \text{ε}) & \quad \text{εF} & \quad \text{εF} \\
\end{align*}
\]

Example with ε-Moves

- String length a multiple of 2 or 3

\[
\begin{align*}
(0, \text{a.a.a}) & \quad \text{ε-moves} \\
(2, \text{a.a.a}) & \quad 0 \quad \text{ε-moves} \\
(3, \text{a.a}) & \quad 2 \quad \text{ε-moves} \\
(4, \text{ε}) & \quad 4 \quad \text{ε-moves} \\
(5, \text{ε}) & \quad 5 \quad \text{ε-moves} \\
(6, \text{ε}) & \quad 6 \quad \text{ε-moves} \\
(7, \text{ε}) & \quad 7 \quad \text{ε-moves} \\
\end{align*}
\]

Example with ε-Moves

- $a^*b^*$

\[
\begin{align*}
(1, \text{ab}) & \quad \text{ε-moves } \text{"consume" no input symbols} \\
(2, \text{b}) & \quad \text{ε-moves } \text{"consume" no input symbols} \\
(3, \text{ε}) & \quad \text{ε-moves } \text{"consume" no input symbols} \\
\end{align*}
\]
Equivalence of NFA to DFA

- There is an algorithm to convert any NFA into a DFA
  - We show basic idea assuming NFA has no ε-moves
  - Then (later) modify the construction for NFAs with ε-moves
- Ex: \( L = \{ x : \text{last symbol of } x \text{ appeared previously} \} \) \( \Sigma = \{a, b\} \)

- Idea: given input string, keep track of all possible reached states after reading each letter. At end of input, see if a final state is among those reached

Equivalence of NFA and DFA (cont’d)

- Computation paths through NFA \( N_0 \) on \( w = abba \)

Equivalence of NFA and DFA (cont’d)

- Idea: keep a list of all possible states reachable by each prefix of \( w \) (“parallel worlds”). For NFA \( N_0 \):
  \[
  \begin{align*}
  (p) & \rightarrow \{ p_x \} \rightarrow \{ p_x \} \rightarrow \{ p_y \} \\
  q & \rightarrow q_x \\
  r & \rightarrow r_x \\
  s & \rightarrow s_x
  \end{align*}
  \]

  \[ \vdash (p) \rightarrow \{ p_x, q_x, r_x, s \} \]

  \( abba \in L(N) \) since \( \{ p_x, r_x, s \} \cap F \neq \emptyset \)
Equivalence of NFA and DFA (cont’d)

- Equivalent DFA \( M \) will have:
  - State set \( \mathcal{Q} \)
  - Alphabet \( \Sigma \)
  - Start state “set” \( \{q_0\} \)
  - Accepting states \( \{X \subseteq \mathcal{Q} : X \cap F \neq \emptyset\} \)
  - Deterministic transition function \( \delta' : \mathcal{Q} \times \Sigma \rightarrow \mathcal{P}(\mathcal{Q}) \)

- Ex: For NFA \( N_0 \):
  \[
  \delta'(\{p, q\}, a) = \{p, q, s\} \\
  \delta'(\{p, q\}, b) = \{p, q, r\} \\
  \delta'(p), a) = \{p, q\} \\
  \delta'(p), b) = \{p, r\} \\
  \ldots
  \]

Equivalence of NFA and DFA (cont’d)

- Thm: [Rabin-Scott Construction]. Let \( L = L(N) \) for some NFA \( N \) with no \( \epsilon \)-moves. There is an algorithm to construct a DFA \( M \) equivalent to \( N \), i.e. with \( L(M) = L(N) \).

  Pf: Given \( N \) we construct a DFA \( M \) and then verify that it recognizes the same set as \( N \).

  Construction: Given NFA \( N = (Q, \Sigma, \delta, s, F) \) construct \( M = (Q', \Sigma, \delta', s', F') \) where
  \[
  Q' = \mathcal{P}(Q), s' = (s), F' = \{X \in Q' : X \cap F \neq \emptyset\}
  \]
  and \( \delta' : Q' \times \Sigma \rightarrow Q' \) is defined as:
  \[
  (\forall S \subseteq Q, a \in \Sigma) \delta'(S, a) = (q \in Q : (\exists p \in S) q \in \delta(p, a))
  \]

Equivalence of NFA and DFA (cont’d)

- Picture of \( \delta' \)

- \( \delta'(S, a) = S' \)
Equivalence of NFA and DFA (cont’d)

- Verification: Show (1) $M$ is a DFA and (2) $L(M) = L(N)$.
  1. $\delta$ is a function by the construction, and $Q'$ is finite: $|Q'| = 2^{|Q|}$. So $M$ is a DFA.
  2. To show equivalence we prove the
    - Lemma:
      \[(p, w) \overset{*}{\rightarrow} (q, \varepsilon) \iff \exists q \in Q \land (\{(p), w\})^{*}_{M} (Q, \varepsilon)\]
      
      \[P:\text{By induction on the length of the input string } w.\]
      
      \[\text{Base } |w| = 0.\]
      
      \[(p, \varepsilon) \overset{*}{\rightarrow} (q, \varepsilon) \iff p = q \iff (\{(p), \varepsilon\})^{*}_{N} (Q, \varepsilon).
      \]
      
      \[\text{Step Suppose (IH) the lemma is true } \forall w. \ |w| \leq k.\]
      
      \[\text{Let } \ |w| = k + 1, \ w = ua, \ |u| = k. \ \text{To show:} \]
      \[(p, ua \overset{*}{\rightarrow} (q, \varepsilon) \iff \exists q \in Q \land (\{(p), ua\})^{*}_{M} (Q, \varepsilon).\]

Equivalence of NFA and DFA (cont’d)

\[\Rightarrow. \ \text{Assume} \ (p, ua) \overset{*}{\rightarrow} (q, \varepsilon). \ \text{Then} \ \exists \ \text{state} \ r \ \text{with} \]
\[(p, u) \overset{*}{\rightarrow} (r, \varepsilon) \ \text{and} \ q \in \delta(r, a). \]

Then \[(p, u) \overset{*}{\rightarrow} (r, \varepsilon). \ \text{By (IH)} \]
\[\exists r, x \in R \land (\{(p), u\})^{*}_{N} (R, \varepsilon) \tag{\text{(*)}}\]

By construction of $M$ $q \in \delta'(R, a)$. Let $Q = \delta'(R, a)$. Then $\exists q \in Q \land (R, a)^{*}_{N} (Q, \varepsilon)$.

Using this with (*) results in:
\[\exists q \in Q \land (\{(p), u\})^{*}_{N} (R, a)^{*}_{N} (Q, \varepsilon). \]

So
\[\exists q \in Q \land (\{(p), ua\})^{*}_{M} (Q, \varepsilon). \]

Equivalence of NFA and DFA (cont’d)

\[\Leftarrow. \ \text{Assume} \ \exists q \in Q \land (\{(p), u\})^{*}_{M} (Q, \varepsilon). \]

Then $\exists \text{state } r \ \text{with} \ \delta'(R, a) = Q$. So
\[\{(p), u\}^{*}_{N} (R, a)^{*}_{N} (Q, \varepsilon) \tag{1}\]

By construction $\exists x \in R, q \in \delta(x, a)$ and $(x, a)^{*}_{N} (q, \varepsilon). \tag{2}$

Since $(\{(p), u\})^{*}_{N} (R, \varepsilon)$ we have from (IH)
\[(p, u) \overset{*}{\rightarrow} (x, \varepsilon) \tag{3}\]

Combining (2) & (3):
\[(p, ua) \overset{*}{\rightarrow} (x, a) \overset{*}{\rightarrow} (q, \varepsilon)\]

So
\[(p, ua) \overset{*}{\rightarrow} (q, \varepsilon). \ \ \square\]

Equivalence of NFA and DFA (cont’d)
Equivalence of NFA and DFA (cont’d)

We now finish the verification proof. Let $F \in F$. From the Lemma

$$\exists Q, F \in Q \land (\{s\}, w) \xrightarrow{\epsilon} (Q, \epsilon) \iff (s', w) \xrightarrow{\epsilon} (Q, \epsilon) \land Q \cap F \neq \emptyset.$$  

That is, $(s, w) \xrightarrow{\epsilon} (F, \epsilon)$ for some $F \in F \iff (s', w) \xrightarrow{\epsilon} (Q, \epsilon)$ for some $Q \in F'$.

$\therefore \ L(M) = L(N)$. □

Example: $\epsilon$-Free NFA → DFA

Consider the previous NFA $N_0 = (Q_0, \Sigma, \delta_0, p_0, \{s\})$

$$M_0 = (\mathcal{P}(Q), \Sigma, \delta', (p), \mathcal{F})$$

NFA with $\epsilon$-Moves

- $\epsilon$-closure($R$) = $\mathcal{E}(R)$ for a set of states $R$
**ε-closure of a set of states**

- Coalesce all nodes reachable from \{4,5\} by ε-moves:

  ![Diagram of ε-closure](image)

  Note: still an NFA

\[ \varepsilon \text{-closure}(\{4,5\}) = \varepsilon(\{4,5\}) = \{1, 2, 3, 4, 5, 6, 7, 8\} \]

---

**Conversion: NFA → DFA**

- **Thm**: There is an algorithm to convert any NFA to an equivalent DFA.

- **Pf**: Construction: Given NFA \( N = (Q, \Sigma, \delta, s_0, F) \) construct new NFA \( M = (2^Q, \Sigma, \delta', s_0', F') \) where

  \[ F' = \{ S \subseteq Q \mid S \cap F' \neq \emptyset \} \]

  \[ s_0' = E(s) \]

  \[ \delta'(S, a) = \bigcup \{ E(p) \mid p \in \delta(q, a) \text{ for some } q \in S \} \]

  **Verification**

  \[ (q, \omega) \vdash_M (p, \varepsilon) \iff (E(q), \omega) \vdash_m (E(p), \varepsilon) \]

  for some set \( P \) containing \( p \)

- **Pf**: By induction on \( |w| \)

---

**Conversion: NFA → DFA**

- **Thm 1.39**: [Rabin-Scott Theorem]: There is an algorithm to convert any NFA into an equivalent DFA.

- **Corollary 1.40**: A language is regular \( \iff \) some NFA recognizes it.

- **Ex**: Start with an NFA \( N_1 \) as follows:

  ![Diagram of NFA](image)
Conversion: NFA → DFA

Ex: $N_i$

$\begin{array}{c}
\rightarrow 1 \\
b \rightarrow 2 \\
e \rightarrow 3 \\
d \rightarrow 4
\end{array}$

**Useful summary**

$E(1) = \{1\}$  $E(2) = \{1, 2, 3\}$  
$E(3) = \{1, 3\}$  $E(4) = \{4\}$  
$1 \rightarrow 2$  $2 \rightarrow b$  $3 \rightarrow 4$  $4 \rightarrow b$  
$1 \rightarrow \emptyset$  $2 \rightarrow 2$  $3 \rightarrow \emptyset$  $4 \rightarrow \emptyset$  
$1 \rightarrow \emptyset$  $2 \rightarrow \emptyset$  $3 \rightarrow \emptyset$  $4 \rightarrow 3$

---

Ex: NFA → DFA

$\begin{array}{c}
\rightarrow 1 \\
b \rightarrow 2 \\
e \rightarrow 3 \\
d \rightarrow 4
\end{array}$

$\delta' = E(1) = \{1\}$  
$\delta'(s', b) = E(2) = \{1, 2, 3\}$  
$\delta'(s', \varepsilon) = \emptyset$  
$\delta'(s', d) = \emptyset$  
$\delta'(1, 2, 3, b) = E(1) \cup E(2) \cup E(3) \cup E(4) = \{1, 2, 3, 4\}$  
$\delta'(1, 2, 3, \varepsilon) = E(3) = \{1, 3\}$  
$\delta'(1, 2, 3, d) = \emptyset$

---

Ex: NFA → DFA (cont’d)

$\begin{array}{c}
\rightarrow 1 \\
b \rightarrow 2 \\
e \rightarrow 3 \\
d \rightarrow 4
\end{array}$

$\delta'(1, 2, 3, 4, b) = E(2) \cup E(4) = \{1, 2, 3, 4\}$  
$\delta'(1, 2, 3, 4, \varepsilon) = E(3) = \{1, 3\}$  
$\delta'(1, 2, 3, 4, d) = E(3) = \{1, 3\}$  
$\delta'(1, 3, b) = E(2) \cup E(4) = \{1, 2, 3, 4\}$  
$\delta'(1, 3, \varepsilon) = \emptyset$  
$\delta'(1, 3, d) = \emptyset$
Conversion: NFA → DFA (cont’d)

Regular Expression → NFA

• Thm 1.55: There is an algorithm that, given a regular expression $E$, constructs an NFA $N$ such that $L(E) = L(M)$.
  
Pf: Induction on the # of operator symbols in $E$.
  
  Base: $E = \varepsilon \emptyset a \in \Sigma$

  Step: Assume (IH) the result is true of all expressions with $\leq$ operator symbols $(+, \cdot, *)$. Let $E$ have $k+1$ ops.

  Three cases:
  
  Case $E = (E_1 + E_2)$. By IH, $\exists$ FA $M_1, M_2$ with $L(E_1) = L(M_1)$ and $L(E_2) = L(M_2)$. Construct the following NFA $M$.

Case $+$

$L(M) = L(M_1) \cup L(M_2)$
Regular Expression → NFA (cont’d)

• Case $E = (E_1 E_2)$. By IH, $\exists$ FA $M_1, M_2$ with $L(E_1) = L(M_1)$ and $L(E_2) = L(M_2)$. Construct the following NFA $M$.

\[
\begin{align*}
M_1 & \quad M_2 \\
F_1 & \quad F_2
\end{align*}
\]

Unmark final states in $M_1$

$L(M) = L(M_1) \cdot L(M_2)$

Regular Expression → NFA (cont’d)

• Case $E = (E_1)^*$. By IH, $\exists$ FA $M_1$ with $L(E_1) = L(M_1)$. Construct the following NFA $M$.
Lecture 03

Case *

\[ M \]

\[ F = F_1 \cup \{s\} \]

\[ L(M) = L(M_1)^* \]

Example: Reg. Exp. \(\rightarrow\) NFA

- \((b+aa)^*\)

\[ (b+aa)^* \]

Not very economical

Regular Expressions—Applications

- Regexp used in various development tools
  - qed – interactive text editor. 1st version Lampson & Deutsch 1967
  - Regexp added by Ken Thompson, Bell Labs, ca. 1968
  - Regexp compiled into NFA in machine code
  - Rabin-Scott idea used to scan “on the fly”
  - One of the first software patents
  - Offspring ed by Ken for Unix
  - Many others followed: em, vi / ex, sam, qedx, ...
  - grep, egrep – pattern search in a file
  - shell – command line interpreter
  - lex – lexical analyzer generator
  - sed – non-interactive stream editor
  - awk – pattern scanning and processing language
  - perl – pattern-driven programming language
Applications (cont’d)

- Regular expressions = “patterns”

<table>
<thead>
<tr>
<th>meaning</th>
<th>awk regexp</th>
</tr>
</thead>
<tbody>
<tr>
<td>matches &gt;=1 r</td>
<td>r+</td>
</tr>
<tr>
<td>matches &gt;=0 r</td>
<td>r*</td>
</tr>
<tr>
<td>matches 0 or 1 r</td>
<td>r?</td>
</tr>
<tr>
<td>matches r then s</td>
<td>rs</td>
</tr>
<tr>
<td>matches r or s</td>
<td>r</td>
</tr>
<tr>
<td>match literal “c”</td>
<td>\c</td>
</tr>
<tr>
<td>match begin/end line</td>
<td>^</td>
</tr>
<tr>
<td>match any char</td>
<td>.</td>
</tr>
<tr>
<td>group exprs</td>
<td>(s)</td>
</tr>
<tr>
<td>character list</td>
<td>[abc... ]</td>
</tr>
<tr>
<td>negated char list</td>
<td>[^abc... ]</td>
</tr>
</tbody>
</table>

Applications--Examples

-?[0-9]+ nonempty digit strings, optional sign
[^0-9] any char except digit
[^\[\].*\]] reference citations in a paper
g/[^ ]*$d delete blank lines
g/[ ]*/d delete lines with a blank

Ex: match is always (1) leftmost and (2) longest
file: abcddddef
  vi: s/d*/s/ xabcddddef
  s/d*/s/ abcxef

Ex: csh: sort roll[1-5] | egrep “C SC|MATH” | pr

Applications--Examples

Ex: traditional spelling mnemonic
- “i before e, except after c,
or when pronounced ‘a’,
as in ‘neighbor’ and ‘weigh’
--except for ‘weird’ examples.”
- grep “[^c]el” /usr/share/dict/words > foo
cat foo
  abseil Aeneid ageing Alamein albeit atheist
  Boeing Budweiser caffeine canoeist deice deictic
dilettantism dreidl ...
- if you think this spelling rule is sufficient, you will be deficient,
inefficient, unscientific and far from omniscient
Applications--Examples

Ex: lex - generates a lexical analyzer yylex(). Example: wordcount (wc)

```c
int nchar, nword, nline;
%
\n\n% {  nline++; nchar++; }
(\^t\n)+ { nword++, nchar += yyleng; }
\n// yyleng = length of matched string
\n% { nchar++;}
%
int main(void) {
    yylex();   // invoke generated lexer
    printf("%d	%d	%d
" , nchar, nword, nline);
    return 0;
}
```

Regular ⇔ L. Denoted by a Reg. Exp.

- We’ve defined “regular” as meaning: recognized by a DFA (equiv. to rec. by an NFA)
- This equivalence result is known as “Kleene’s Theorem”
- We’ve already shown the ⇐ direction—we constructed an NFA from a regular expression (Using Rabin-Scott we could convert this NFA to a DFA.)
- Now we show the ⇒ direction: given a DFA \( M \) construct a regular expr. \( E \) with \( L(M) = L(E) \).
- Thm (Kleene): There is an algorithm that, given a DFA \( M \), computes a regular expression \( E \) such that \( L(M) = L(E) \).
- Pf: Given the graph of the DFA, use the “node elimination algorithm” to gradually eliminate all nodes in favor of expressions on the edges of the graph.

Kleene’s Thm: use “Generalized NFA”

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>B</td>
</tr>
</tbody>
</table>

Order: CAB
Ex: Node Elimination Algorithm

Add $\epsilon$-moves:

Ex: Node Elimination Algorithm (ACB)

Elim. A:

Elim. C:

Elim. B:

Ex: Node Elimination: other orders
Ex: Other elimination orders (CAB)

Elim. C:

Elim. A:

CA = AC (above)

Elim. B:

CAB = ACF

Ex: Other elimination orders (CBA)

Elim. C:

Elim. B:

CB

Elim. A:

CBA

Ex: Other elimination orders (BAC)

Elim. B:

Elim. A:

BA = AB

Elim. C:

B-BC = ABC
Ex: All elimination orders equiv \((BAC = ACB)\)

\[
\begin{align*}
BAC & \quad \left( b(b+a)^*ab \right)^* \\
ACB & \quad \left( b(b+a)^*a \right)^* \\
\end{align*}
\]

Easy to prove by induction: for any expression \(E\), \(b(\epsilon+E)b^* = b(\epsilon+a^*a)b\).

Using this identity: \(b(bb+a^*ab)^* = b(b(b+a^*ab))^* = b(b(a^*a)a)b^* = b(ba^*)^*\)

Further regular expression simplification is possible: \(b(bb+a^*ab)^* = b(b(b+a^*ab))^* = b(b(a^*a)a)b^* = b(ba^*)^*\)

Good exercise: show results of all other elimination orders are equivalent to these, using regular expression algebra

---

Algebra of Regular Expressions

- \(\exists\) an algebra for simplifying regular expressions
- Can use this algebra to construct RegExps from FSA

\[
\begin{align*}
(r+s)t & = r(st) + s(t) \\
r+rr & = r \\
r+0 & = r \\
(r^*)^* & = r^* \\
(r^*) & = r + r^* \\
(r+s)^* & = (r+s)^* \\
\end{align*}
\]

---

Solving Regular Expr Equations

- Can solve “linear equations” with regexp variables

\[
\begin{align*}
X & = aX + b \\
& = a(aX + b) + b = a^2X + ab + b \\
& = a^2(aX) + ab + b = a^3X + a^2b + ab + b \\
& \vdots \\
X & = a^*b \\
\end{align*}
\]

Check: \(a[a^*b] + b = aa^*b + b = (aa^*+\epsilon)b = a^*b\)

Ex:

\[
\begin{align*}
X & \rightarrow aX + bY \\
Y & \rightarrow \epsilon \rightarrow X = a^*b
\end{align*}
\]
Solving RegExp Equations (cont’d)

- Ex: NFA → RegExp

\[
\begin{align*}
A &= 0A + 1B + \varepsilon \\
B &= 1B + 0C \\
C &= 0A + 1B \\
\text{elim.} B &: B = 1'0C \\
\text{elim.} C &: C = (0'1)'0A \\
A &= 0A + 1'0C + \varepsilon \\
C &= 0A + 1'0C \\
A &= 0A + 1'0(1'1)'0A + \varepsilon \\
&= (0 + 1'0)(1'1)'0A + \varepsilon \\
&= (0 + 1'0)(1'1)'0'0' \\
\text{Simplify using reg. algebra:} \\
0 + 1'0(1'1)'0'0' &= (0 + 1'0)(1'1)'0'0' \\
&= 0'1'00' \\
\therefore A &= (0'1)'0' \\
\end{align*}
\]

Ex: Node Elimination Example via Algebra

- Want B. C is accept state.

\[
\begin{align*}
A &= aA + aB \\
B &= bA + bB + bC + \varepsilon \\
C &= bA + ab + bb + \varepsilon \\
\text{elim. A:} \\
A &= aA + aB \\
B &= bA + bB + bC + \varepsilon \\
\text{elim. B:} \\
B &= bA + bB \\
C &= (ba + ab + bb) * \\
\text{elim. C:} \\
B &= bA + bB + bC + \varepsilon \\
\therefore B &= bA + bB + bC + \varepsilon \\
\text{Simplifies to} B &= bA + bB + bC + \varepsilon \\
\end{align*}
\]

Closure Properties

- A class of languages is said to be closed under an operation if applying that operation to members of the class results in a language that is again a member of the class. Example: the regular languages are closed under the operations of union, concatenation and Kleene star.

- Thm: The regular languages are closed under intersection and complementation.

**Proof:** Let \( L = L(M) \) where \( M = (Q, \Sigma, \delta, s_0, F) \) is a DFA. Then the FA \( \mathcal{R} = (Q', \Sigma, \delta', s_0, F') \) is also deterministic, and \( (s_0, w) \sim (Q_0, \varepsilon) \iff (s_0, w) \sim (Q_0, \varepsilon) \). So \( w \) leads to a non-accepting state in \( M \) \( \iff \) \( w \) leads to an accepting state in \( \mathcal{R} \). So \( L(\mathcal{R}) = \Sigma^* - L(M) \).
Closure Properties (cont’d)

Intersection. Let \( L_1, L_2 \) be regular. By DeMorgan’s law
\[
L_1 \cap L_2 = (L_1 \cup L_2)'
\]
Since the regular languages are closed under complementation and union, the result follows. \( \square \)

Closure Properties (cont’d)

- Another proof of closure under \( \cap \) illustrates the technique called “cross-product construction”. See Sipser text, Theorem 1.25.
- Thm: The class of regular languages is closed under the intersection operation.
  
  Pf: Assume \( L_1 = L(M_1) \), \( L_2 = L(M_2) \) and
  
  where the automata are deterministic.
  
  Construction. Construct a “cross-product machine” \( M \) as follows:
  
  \[
  \delta \times \delta' \left( q_1, q_2 \right), a = \left( \delta (q_1, a), \delta' (q_2, a) \right),
  \]
  
  Machine \( M \) simulates the two given machines “in parallel”, keeping each machine state in one component of \( \left( q_1, q_2 \right) \).

Verification. By an easy induction on \( |x| \), can show that
\[
\left( (q_1, q_2), x \right) \vdash_{M} \left( (p_1, p_2), \varepsilon \right) \Leftrightarrow \left( (q_1, x) \right) \vdash_{M_1} \left( p_1, \varepsilon \right) \land \left( (q_2, x) \right) \vdash_{M_2} \left( p_2, \varepsilon \right)
\]
Therefore, for a pair of final states \( q_1 \in F_1, q_2 \in F_2 \)
\[
\left( (q_1, q_2), x \right) \vdash_{M} \left( (p_1, p_2), \varepsilon \right) \Leftrightarrow \left( (q_1, x) \right) \vdash_{M_1} \left( p_1, \varepsilon \right) \land \left( (q_2, x) \right) \vdash_{M_2} \left( p_2, \varepsilon \right)
\]
This says that
\[
x \in L(M) \Leftrightarrow x \in L(M_1) \land x \in L(M_2)
\]
i.e., that
\[
L(M) = L(M_1) \cap L(M_2).
\]
Closure Properties (cont’d)

- **Defn:** A **homomorphism** \( h \) is a function that maps each symbol of \( \Sigma = \{ a_1, a_2, \ldots, a_n \} \) to a string over some alphabet \( \Delta \), i.e.,
  \[ h(a_1) = w_1, \ h(a_2) = w_2, \ldots, \ h(a_n) = w_n. \]
- The homomorphism is extended to operate on strings character-by-character, i.e.,
  \[ h(c_1 c_2 \ldots c_n) = h(c_1) h(c_2) \ldots h(c_n). \]
- It is further extended to languages element-wise, i.e.,
  \[ h(L) = \{ h(w) : w \in L \}. \]
- **Thm:** If \( L \) is regular and \( h \) is a homomorphism, then \( h(L) \) is regular.
  
  **Pf:** Assume \( L \) is recognized by a DFA \( M = (Q, \Sigma, \delta, s_0, F) \).  

---

Closure Properties (cont’d)

- **Construction:** Construct the machine \( M^h = (Q, \Sigma, \delta^h, s_0, F) \) where for each transition in \( M: \)
  \[ p \xrightarrow{a} q \]
  put into \( M^h \) the transition \[ p \xrightarrow{h(a)} q. \]
  
  An easy induction establishes that
  \[ (s, w) \xrightarrow{a} (q, \varepsilon) \iff (s, h(a)w) \xrightarrow{a} (q, \varepsilon), \]
  from which it follows that
  \[ L(M^h) = h(L(M)) \].  \( \square \)

---

What is **Not** Regular?

- **FA** have a very limited computing ability. They cannot, for example, recognize strings of well-nested parentheses, or well-formed arithmetic expressions, or even the language of strings of the form \( w^2 \), having two copies of the same substring.
- How can we show some languages are **not** regular? We will give a property that all regular languages **must have** (called the pumping property). Then, to show that a language \( L \) is not regular, we argue that it lacks this pumping property.
Pumping Lemma

- Thm [Pumping Lemma for Regular Languages]. Suppose that $L$ is an infinite⁴ regular language. Then

$$(\exists p) \ (\forall w) \ [w \in L \land |w| \geq p \Rightarrow (\exists x, y, z) \ (w = xyz \land y \neq \varepsilon \land |xy| \leq p \land (\forall i \geq 0) \ xy^i z \in L)]$$

⁴All finite languages are regular, so only infinite languages are of interest.

Pumping Lemma (English)

- Thm [Pumping Lemma for Regular Languages]. Suppose that $L$ is an infinite⁴ regular language. Then there is some number $p$ (called the pumping length) such that:

  if $w$ is any string in $L$ with $|w| \geq p$, then $w$ can be factored into 3 substrings, $w = xyz$, that satisfy the following 3 conditions:

  (i) $y \neq \varepsilon$ [y is not empty]
  (ii) $|xy| \leq p$ [the prefix and pumped part are short]
  (iii) for every $i \geq 0$, $xy^i z \in L$ [“pumped up” and “pumped down” ($i = 0$) versions of the string must also be in $L$]

⁴All finite languages are regular, so only infinite languages are of interest.

Pumping Lemma (cont’d)

- Pf: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing $L$ and let $p$ be the number of its states.

  Let $w = a_1 a_2 \cdots a_n$ be an input string of length $n$ where $n \geq p$.

  Let $x_1, x_2, \cdots, x_n, x_{n+1}$ be the sequence of states that $M$ enters while processing $w$ so that $x_a = \delta(x_{a-1}, a)$ for $1 \leq a \leq n$. This state sequence has length $n+1 \geq p+1$.

  Among the first $p+1$ states of this sequence, at least 2 must be the same state [“pigeonhole principle”]. Call the first of these 2 $x_j$ and the second $x_k = x_j$. Because $x_j = x_k$ occurs among the first $p+1$ places in the sequence $x_1, x_2, \cdots, x_n, x_{n+1}$, we have that $k \leq p+1$.

  Define the following substrings of $w$:

  $X = a_1 \cdots a_{j-1}, y = a_j \cdots a_{k-1}, z = a_k \cdots a_n$
Pumping Lemma (cont’d)

- Picture:

- From the picture, we see that there is an accepting path from \( r_i \) to a final state \( r_k \) for all the strings of the form \( x y^i z \), \( i \geq 0 \). Also, since \( j \neq k \) it must be that \( y \neq z \). Furthermore, \( |y| > k \) so \( |xy| \leq p \). □

Non-regular Examples

- Ex: \( L = \{a^n : k \geq 0\} \) is not regular.
  
  \( \text{Pf:} \) By contradiction. Suppose \( L \) is regular. Then by the Pumping Lemma,
  \( (\exists x, y, z) x = a^k, y = a^l (q > 0), z = a^m \) \( (\forall n \geq 0) a^p \cdot (a^q)^n \cdot a^r \in L \).
  
  Then it follows that \( (\forall n \geq 0) p + q \cdot n + r \) is a perfect square. This is impossible. For suppose \( p + r + q \cdot n_i = k_i^2 \) for a \( k_i \) so large that \( 2k_i + 1 > q \). Then \( p + r + q \cdot (n_i + 1) = k_i^2 + q < k_i^2 + 2k_i + 1 = (k_i + 1)^2 \).
  
  Hence \( p + r + q \cdot (n_i + 1) \) falls “in between” perfect squares—a contradiction. □

Non-regular Examples (cont’d)

- Ex: \( L = \{w : w \in \{a, b\}^*\} \) is not regular.
  
  \( \text{Pf:} \) By contradiction. Suppose \( L \) is regular. Then by closure properties of the regular languages
  \( L_1 = L \cap a \cdot b \cdot a \) is regular. Now \( L_1 = \{a^n bba : n \geq 0\} \). We show \( L_1 \) cannot be regular, which provides a contradiction.
  
  If \( L_1 \) is regular, then there are substrings \( x, y, z \) with \( y \neq \varepsilon \) such that \( (\forall n \geq 0) xy^n z \in L \).
  
  Case 1: \( y \) is entirely in the \( a \)’s. Assume it is in the \( a \)’s before the \( 2 \) \( b \)’s (The other subcase is symmetric). Then \( x = a^p, y = a^q, z = a^r bba^s (q > 0) \)
  
  and \( p + q + r = s \). But then \( a^p a^q bba^s \in L \)
  
  where \( p + r < s \). This is a contradiction.
Non-regular Examples (cont’d)

- Case 2. \( y \) contains a b. Then \( xy^i z \) has more than 2 b’s, and so \( xy^i z \notin L_A \). This is a contradiction.

Contradictions in all cases \( \Rightarrow \) contradiction to the assumption that \( L_A \) is regular. So \( L_A \) is not regular. \( \Box \)

- Ex: \( B = \{ w : w \) is a well-nested string of parentheses\) is not regular.
  
  Pf: Suppose \( B \) is regular. Then so is \( B' = B \cap (\cdot)^* = \{ (\cdot)^n : n \geq 0 \} \) as its homomorphic image \( \{a^nb^n : n \geq 0 \} \). Contradiction. \( \Box \)

Decision Problems

- For a property/predicate \( P \) the decision problem for \( P \) is:
  - Given: \( x \)
  - Question: Is \( P(\alpha) \) true?

- Ex: Given DFA \( M \), is it true that \( L(M) = \emptyset \)?

- Thm: For DFA \( M \) all the following decision problems are solvable, i.e. there exists an algorithm to decide the question for any input:

  Given \( \quad \) Question
  1. \( M, w \) \( w \in L(M) ? \)
  2. \( M \) \( L(M) = \emptyset ? \)
  3. \( M \) \( L(M) = \Sigma^* ? \)
  4. \( M, M' \) \( L(M) \subseteq L(M')? \)
  5. \( M, M' \) \( L(M) = L(M')? \)

Decision Problems (cont’d)

Pf: Assume given DFAs for inputs.

1. Trace \( w \) through \( M \). “Yes” if leads from \( s \) to some \( q \in F \)
2. “Yes” if there is some \( q \in F \) reachable from \( s \)
3. Convert \( M \to \Pi \). \( L(M) = \Sigma^* \Leftrightarrow \Pi(M) = \emptyset \)
4. \( L(M_1) \subseteq L(M_2) \Leftrightarrow \Pi(M_1) \cap \Pi(M_2) = \emptyset \)
5. Use (4) twice