Regular Languages and Regular Grammars

Wen-Guey Tzeng
Computer Science Department
National Chiao Tung University
Syllabus

• Give other formal representations for regular languages: regular expression, regular grammar

• Establish equivalence relations between regular expressions, regular grammars and finite acceptors
Regular Expressions

• A **formal notation** of describing a language.

• Recursive definition
  
  – Let $\Sigma$ be an alphabet. Then, $\emptyset$, $\lambda$, $a \in \Sigma$ are regular expressions.
  
  – If $r_1$ and $r_2$ are regular expressions, so are
    $r_1 + r_2$, $r_1 r_2$, $(r_1)$, $r_1 \ast$. 

2017 Spring
Languages associated with R.E.

• The language $L(r)$ denoted (defined) by $r$ is:
  
  - $\emptyset$
  - $\lambda$
  - $a$
  - $L(r_1+r_2)$
  - $L(r_1r_2)$
  - $L((r_1))$
  - $L(r_1^*)$
• Examples
  – L(a)
  – L(a*)
  – L(a+ab)
  – L(a*(a+b))
  – L((a+b)*(a+bb))
  – L((aa)*(bb)*b)
Design R.E. for languages

• $L = \{w \in \{a,b\}^* : w$ starts with a and ends with b$\}$

• $L = \{w \in \{a,b\}^* : w$ contains substring aba$\}$

• $L = \{w \in \{a,b\}^* : w$ does not contain substring aba$\}$

• $L = \{w \in \{a,b\}^* : |w| \text{ mod } 3 = 0 \}$
Design R.E. for languages

- \( L = \{ w \in \{0,1\}^* : w \) has no pair of consecutive zeros \}
  - \( r = (1^*011^*)^*(0+\lambda) + 1^*(0+\lambda) \)
  - \( r = (1+01)^*(0+\lambda) \)
  - \( r = 1^*(011^*)^*(0+\lambda) \)
Simplification of R.E.

- \((r^*)^* = r^*\)
- \(r_1^*(r_1 + r_2)^* = (r_1 + r_2)^*\)
- \(r_1(r_2 + r_3) = r_1r_2 + r_1r_3\)
- \(r_1\emptyset = \emptyset\)
- \(r + \emptyset = r\)
- \(\emptyset^* = \lambda\)
R.E. and Regular languages

• Every R.E. r denotes a regular language
  – For every r, there is an nfa M for accepting L(r), L(M)=L(r).

• Every regular language L is denoted by a R.E. r.
  – For every dfa M, there is a R.E. r denoting L(M), L(r)=L(M).
R.E. $r \Rightarrow$ nfa $M$ (one final state)

- Use the recursion property of R.E.

Figure 3.1: (a) nfa accepts $\emptyset$. (b) nfa accepts $\{\lambda\}$. (c) nfa accepts $\{a\}$. 
Figure 3.2: Schematic representation of an nfa accepting $L(r)$. 

$M(r)$
Figure 3.3: Automaton for $L(r_1 + r_2)$. 
Figure 3.4: Automaton for $L(r_1 r_2)$. 
Figure 3.5: Automaton for $L(r_1^*)$. 

2017 Spring
Examples

Figure 3.6: (a) $M_1$ accepts $L (a + bb)$. (b) $M_2$ accepts $L (ba^* + \lambda)$. 
Figure 3.7: Automaton accepts $L((a + bb)^* (ba^* + \lambda))$. 
• Generalized transition graphs (GTG)
  – An edge is labelled by a r.e.
• Complete GTG
  – Every edge is labeled.
• Simple two-state GTG $\Rightarrow$ R.E.
  – One is the initial state (non-final) and the other is a final state
• $r = r_1 * r_2 (r_4 + r_3 r_1 * r_2)^*$
• 3-state GTG: remove one state from GTG

• Example, remove $q_2$
  – Add an edge $q_1 \rightarrow q_1$
    • Labeled $e+af*b$
  – Add an edge $q_1 \rightarrow q_3$
    • Labeled $h+af*c$
  – Add an edge $q_3 \rightarrow q_1$
    • Labeled $i+df*b$
  – Add an edge $q_3 \rightarrow q_3$
    • Labeled $g+df*c$
Algorithm nfa-to-rex

• Input: nfa M;
  1. Start with an nfa with $Q=\{q_0, q_1, \ldots, q_n\}$ and a final state $q \neq q_0$
  2. Convert the nfa to a complete GTG. Let $r_{ij}$ be the label from $q_i$ to $q_j$
  3. If the GTG has only two states, construct the regular expression directly.
  4. If the GTG has three states ($q_i$: initial, $q_j$: final, $q_k$: to be removed),
      • Add labels $r_{pq} + r_{pk} r_{kk} * r_{kq}$ for $p=i, j$ and $q=i, j$
      • Remove $q_k$ from the GTG
5. Whenever the GTG has 4 or more states, pick a state $q_k$ to be removed. For every $(q_i, q_j, q_k)$, $i \neq k$, $j \neq k$, do step 4.

6. Repeat steps (3)-(5) until a regular expression is obtained.

• Note: do as much simplification as possible
  • $r+\emptyset = r$
  • $r\emptyset = \emptyset$
  • $\emptyset^* = \lambda$
Example

Figure 3.13
Figure 3.14
Figure 3.15
R.E. for simple patterns

• Used in describing patterns
  – In programming languages, the set of integers: sdd*, s is from \{+, -, \lambda\}, d is from \{0, 1, ..., 9\}
  – In Unix, search a file name with pattern aba*c
    • How does UNIX do it?
grep "([A-Za-z]*)" GPL-3

Copyright (C) 2007 Free Software Foundation, Inc. distribution (with or without modification), making available to the than the work as a whole, that (a) is included in the normal form of Component, and (b) serves only to enable use of the work with that (if any) on which the executable work runs, or a compiler used to (including a physical distribution medium), accompanied by the (including a physical distribution medium), accompanied by a place (gratis or for a charge), and offer equivalent access to the ...
...

2017 Spring
Regular Grammars

- A grammar is a *formal way* of describing languages
- A regular grammar describes a regular language
  - linear grammars
  - left-linear and right-linear grammars
Linear grammar

• A grammar \( G=(V, T, S, P) \) is **right-linear** if the productions are of form
  \[
  A \rightarrow xB \quad \text{or} \quad A \rightarrow x,
  \]
  where \( A, B \in V, x \in T^* \).

• A grammar \( G=(V, T, S, P) \) is **left-linear** if the productions are of form
  \[
  A \rightarrow Bx \quad \text{or} \quad A \rightarrow x,
  \]
  where \( A, B \in V, x \in T^* \).
Examples

• $G_1 = (\{S\}, \{a, b\}, S, \{S \to abS | a\})$

• $G_2 = (\{S, S_1, S_2\}, \{a, b\}, S,$
\[\{S \to S_1ab, S_1 \to S_1ab | S_2, S_2 \to a\}\])

• Not linear: $G_3 = (\{S, A, B\}, \{a,b\}, S,$
\[\{S \to A, A \to aB | \lambda, B \to Ab\}\})$
Right-linear grammar $\rightarrow$ nfa

- Variables are depicted as states.
- Add a final state $V_f$
- Productions are transferred into transitions.
- Example, $P=\{V_0 \rightarrow aV_1, V_1 \rightarrow abV_0 | b\}$
dfa $\rightarrow$ right-linear grammar

- States $q_0, q_1, \ldots$ are depicted as variables $V_0, V_1$.
- $\delta(q_i, a)=q_j$ is denoted as $V_i \rightarrow aV_j$
- Add $V_f \rightarrow \lambda$
- Example
Equivalence of right- and left-linear grammars

- Left-linear grammar $G$, $L(G) = A$
  
  $(\text{reverse } G_1) \rightarrow$ Right linear grammar $G_1$, $L(G_1) = A^R$
  
  $\rightarrow \text{nfa } M_1$, $L(M_1) = A^R$

  $(\text{reverse } M_1) \rightarrow \text{nfa } M_2$, $L(M_2) = (A^R)^R = A$

  $\rightarrow \text{dfa } M_3$, $L(M_3) = A$

  $\rightarrow$ Right-linear grammar $G_2$, $L(G_2) = A$
Sum up

• Regular languages