Regular Expressions

= Regular Languages

Mark Greenstreet, CpSc 421, Term 1, 2008/09
Lecture Outline

Regular Expressions

- Regular Expressions
- Equivalence of Regular Expressions and Finite Automata
Regular Madlibs

Once upon a ________, there was a ________ that ________

_______

zero or more adjectives  plural noun

- Let avocado denote the language \{avocado\}.
- Let noun = avocado ∪ beach ∪ carrot ∪ caterpillar ∪ pencil ∪ penguins ∪ zombie.
- Let pluralNoun = noun s.
- Let verb = add ∪ compile ∪ eat ∪ sing ∪ swim ∪ walk.
- Let pastVerb = verb ed.
- Let adjective = beautiful ∪ big ∪ cold ∪ considerable ∪ furry ∪ insipid ∪ yellow.

Now, our Madlib™ is

Once upon a noun , there was a noun , that pastVerb
(adjective)* pluralNoun.
Regular Madlibs

Once upon a ________, there was a _______ that _________
noun noun past tense verb

_______________ ________ •
zero or more adjectives plural noun

● Let avocado denote the language \{avocado\}.

● Let noun =
  avocado ∪ beach ∪ carrot ∪ caterpillar ∪ pencil ∪ penguins ∪ zombie.

● Let pluralNoun = noun s.

● Let verb = add ∪ compile ∪ eat ∪ sing ∪ swim ∪ walk.

● Let pastVerb = verb ed.

● Let adjective =
  beautiful ∪ big ∪ cold ∪ considerable ∪ furry ∪ insipid ∪ yellow.

● Now, our Madlib\textsuperscript{TM} is

  Once upon a pencil , there was a noun , that pastVerb (adjective)* pluralNoun.
Regular Madlibs

Once upon a ________, there was a _______ that _________

__________   ________  *
zero or more adjectives  plural noun

- Let *avocado* denote the language {avocado}.
- Let *noun* =
  avocado ∪ beach ∪ carrot ∪ caterpillar ∪ pencil ∪ penguins ∪ zombie.
- Let *pluralNoun* = noun s.
- Let *verb* = add ∪ compile ∪ eat ∪ sing ∪ swim ∪ walk.
- Let *pastVerb* = verb ed.
- Let *adjective* =
  beautiful ∪ big ∪ cold ∪ considerable ∪ furry ∪ insipid ∪ yellow.
- Now, our Madlib™ is
  Once upon a  pencil , there was a  carrot , that
  pastVerb  (adjective)*  pluralNoun.
Regular Madlibs

Once upon a ________, there was a _______ that ___________

__________  __________  __________  •

zero or more adjectives  plural noun

- Let avocado denote the language \{avocado\}.
- Let noun = 
avocado ∪ beach ∪ carrot ∪ caterpillar ∪ pencil ∪ penguins ∪ zombie.
- Let pluralNoun = noun s.
- Let verb = add ∪ compile ∪ eat ∪ sing ∪ swim ∪ walk.
- Let pastVerb = verb ed.
- Let adjective = 
  beautiful ∪ big ∪ cold ∪ considerable ∪ furry ∪ insipid ∪ yellow.

- Now, our Madlib\textsuperscript{TM} is
  Once upon a pencil , there was a carrot , that walked (adjective)* pluralNoun.
Regular Madlibs

Once upon a ______, there was a ______ that ____________

___________ zero or more adjectives ____________ plural noun

- Let avocado denote the language \{avocado\}.
- Let noun = avocado ∪ beach ∪ carrot ∪ caterpillar ∪ pencil ∪ penguins ∪ zombie.
- Let pluralNoun = noun s.
- Let verb = add ∪ compile ∪ eat ∪ sing ∪ swim ∪ walk.
- Let pastVerb = verb ed.
- Let adjective = beautiful ∪ big ∪ cold ∪ considerable ∪ furry ∪ insipid ∪ yellow.

Now, our Madlib™ is

Once upon a pencil , there was a carrot , that walked beautiful, (adjective)* pluralNoun.
Regular Madlibs

Once upon a _____, there was a _____ that ____________

noun noun past tense verb

_________________________ ________
zero or more adjectives plural noun

- Let avocado denote the language \{avocado\}.
- Let noun = avocado \cup beach \cup carrot \cup caterpillar \cup pencil \cup penguins \cup zombie.
- Let pluralNoun = noun s.
- Let verb = add \cup compile \cup eat \cup sing \cup swim \cup walk.
- Let pastVerb = verb ed.
- Let adjective = beautiful \cup big \cup cold \cup considerable \cup furry \cup insipid \cup yellow.

- Now, our Madlib\textsuperscript{TM} is

Once upon a pencil , there was a carrot , that walked beautiful, considerable pluralNoun.
Regular Madlibs

Once upon a ________, there was a _______ that __________

__________ ____________ ________
zero or more adjectives  plural noun

- Let *avocado* denote the language \{avocado\}.
- Let *noun* =
  avocado U beach U carrot U caterpillar U pencil U penguins U zombie.
- Let *pluralNoun* = noun s.
- Let *verb* = add U compile U eat U sing U swim U walk.
- Let *pastVerb* = verb ed.
- Let *adjective* =
  beautiful U big U cold U considerable U furry U insipid U yellow.

- Now, our Madlib™ is
  Once upon a _____, there was a _____ that ________
  walked beautiful, considerable ___ penguins.
Regular Expressions

- A regular expression, $\alpha$, is

<table>
<thead>
<tr>
<th>$R$</th>
<th>$L(R)$</th>
<th>where</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>${\epsilon}$</td>
<td></td>
</tr>
<tr>
<td>$c$</td>
<td>${c}$</td>
<td>$c \in \Sigma$</td>
</tr>
<tr>
<td>$R_1 \cup R_2$</td>
<td>$L(R_1) \cup L(R_2)$</td>
<td>$R_1$ and $R_2$ are regular expressions</td>
</tr>
<tr>
<td>$R_1 \cdot R_2$</td>
<td>$L(R_1) \cdot L(R_2)$</td>
<td>$R_1$ and $R_2$ are regular expressions</td>
</tr>
<tr>
<td>$R_1^*$</td>
<td>$L(R_1)^*$</td>
<td>$R_1$ is a regular expression</td>
</tr>
</tbody>
</table>

- Language union, concatenation, and asteration were defined in the Sept. 10 notes and Sipser p. 44.
Regular Expressions Examples

Let $\Sigma = \{a, b\}$.

- $a^*b^*$ — the set of all string with zero or more $a$’s followed by zero or more $b$’s. For example, the strings $\epsilon$, $a$, $aaab$, $bb$, and $aabbb$ are in this language. The strings $aba$ and $ba$ are not.

- $(aaa)^*(bb)^*b$ — the set of all strings consisting of a number of $a$’s that is divisible by three followed by an odd number of $b$’s. For example, the strings $b$, $aaabbb$, and $aaaaaaaaaaaabbbbb$ are in this language, but the strings $\epsilon$, $baaa$, and $aabbb$ are not.

- $a\Sigma^*b$ — the set of all strings that begin with an $a$ and end with a $b$. For example, the strings $ab$, $ababab$ and $abbbabaababab$ are in this language, but the strings $a$, $aba$, and $babbab$ are not.
A Few More Remarks

- We’ll write $\Sigma$ as a regular language that generates the language of all strings in $\Sigma^1$.
- From the definition of $L^*$, we note that $\epsilon \in L^*$ for any language $L$. In particular, note that $\emptyset^* = \{\epsilon\}$.
- Regular expressions and programming languages.
  The following regular expressions describe various lexical pieces of Java:
  - The keyword class: `class`.
  - Identifiers: $([A - Z] \cup [a - z] \cup _) \cup $ $([A - Z] \cup [a - z] \cup _ \cup $ $\cup [0 - 9])^*$, where $[A - Z]$ denotes all characters from A to Z, and likewise for $[a - z]$ and $[0 - 9]$.
  - Floating point numbers:
    $$(((0 - 9)^+ \cdot (0 - 9)^*) \cup ((0 - 9)^* \cdot (0 - 9)^+)) (\epsilon \cup (e (+ \cup _ \cup e) (0 - 9)^+)) \cup$$
    $$[0 - 9]^+ e (+ _ \cup _ \epsilon) [0 - 9]^+, $$
  where $[0 - 9]^+ = [0 - 9][0 - 9]^*$. 

We will show that every language described by a regular expression is recognized by an NFA.

We will then show that every language recognized by a DFA has a corresponding regular expression.
From REs to NFAs – strategy

- Regular expressions are defined inductively (see slide 4).
- Our proof is by induction on the structure of the regular expression.
- One case for each way to form a regular expression:
  - The empty language: $\emptyset$
  - The empty string: $\epsilon$
  - A single symbol: $c$
  - Union of two REs: $R_1 \cup R_2$
  - Concatenation of two REs: $R_1 \cdot R_2$
  - Kleene star: $R^*$
From REs to NFAs

- $R = \emptyset$: 

- $R = \epsilon$: 

- $R = c$: 

- $R = R_1 \cup R_2$: 

$N_1$ recognizes $R_1$

$N_2$ recognizes $R_2$
From REs to NFAs (cont.)

- $R = R_1 \cdot R_2$

- $R = R_1^*$
An Example

\[ R = (b \cup c \cup ab)^* \]

\[ \begin{align*}
\text{a} & \equiv \quad \text{a} \quad \text{a} \quad \text{a} \\
\text{ab} & \equiv \quad \text{a} \quad \varepsilon \quad \varepsilon \quad \varepsilon \\
\text{b} & \equiv \quad \text{b} \quad \text{b} \quad \text{b} \\
\text{c} & \equiv \quad \text{c} \quad \text{c} \quad \text{c} \\
\text{b} \cup \text{c} & \equiv \quad \epsilon \quad \epsilon \quad \epsilon \\
\text{b} \cup \text{c} \cup \text{ab} & \equiv \quad \epsilon \quad \epsilon \quad \epsilon \quad \epsilon \\
(b \cup c \cup ab)^* & \equiv \quad \epsilon \quad \epsilon \quad \epsilon \quad \epsilon \quad \epsilon \\
\end{align*} \]
From DFAs to REs

- Given a DFA, we want to construct a regular expression that for the DFA’s language.

- The “hard” part is keeping track of all of the possible paths from the start state to an accepting state, especially because there can be many possible loops.

- The key observation is that the symbols that label edges in a DFA are simple regular expressions.

  - We’ll generalize this idea and allow arbitrary regular expressions on edges.

  - We’ll use the flexibility of regular expressions to allow us to eliminate one state from the DFA at a time. We’ll modify the REs for the remaining edges to account for the deleted states. Thus, our new DFA will recognize the same language as the original one.

  - By successively deleting states, we’ll eventually get to a DFA with a start state, an accept state, and a single edge from the start state to the accept state. The label for this edge is the RE corresponding to the original DFA.
Consider paths from state 1 to state 4 that go through state 0.

Any such path must begin with a string that takes it to state 0 for the first time. $\alpha_1$ describes such strings.

Then, the path can visit state 0 several times. The expression $\beta^*$ describes all such looping.

Finally, the path has visited state 0 for the last time and goes to state 4. The expression $\gamma_4$ describes that part of the path.

Thus, the set of strings that start in state 1, pass through state 0 at least once, and end in state 4 are described by the expression $\alpha_1 \beta^* \gamma_4$. 
We can replace all edges in and out of state 0 in the same way as we replaced the edge from state 1.

Once we’ve done this, we can eliminate state 0 from the machine.

The resulting machine accepts the same language as the original machine.

We continue, until the we have eliminated all states except for the start and accept states. The final machine accepts the same language as the original machine. The final machine has one edge whose label is the regular expression corresponding to the original DFA.
From DFAs to REs (proof 1/3)

To make a complete proof out of the preceding observations, we define the automata that we use that have regular expressions for edge labels.

- A GNFA, $G$, is a 5-tuple $(Q, \Sigma, E, s, t)$.
- $Q$ is a finite set of states.
- $\Sigma$ is a finite set of symbols.
- $E : Q \times Q \rightarrow \text{regular expression}$, is the edge labeling.
- $s$ is the start state, there are no edges going into $s$.
- $t$ is the accepting state, there are no edges going out of $t$.
- $G$ accepts $w$ iff there are strings $x_1, x_2, \ldots x_k$ and states $q_1, q_1, \ldots q_{k-1}$ such that $x_1$ matches the regular expression for $(s, q_1)$, $x_i$ matches the label for $(q_{i-1}, q_i)$, and $x_k$ matches the label for $(q_{k-1}, t)$. 
From DFAs to REs (proof 2/3)

Given a DFA, \( M = (Q_D, \Sigma, \delta_D, q_0, F_D) \), we construct a GNFA with \( G = (Q_G, \Sigma, E, q_{\text{start}}, q_{\text{accept}}) \) where

- \( Q_G = Q_D \cup \{q_{\text{start}}, q_{\text{accept}}\} \) – we require \( q_{\text{start}}, q_{\text{accept}} \not\in Q_D \).

- If for each \( c \in C_{i,j} \), \( \delta(q_i, c) = q_j \), then \( E \) has an edge from \( q_i \) to \( q_j \) labeled with the regular expression \( \bigcup_{c \in C_{i,j}} c \).

- There is an edge from \( q_{\text{start}} \) to \( q_{0,D} \) labeled with \( \epsilon \).

- There is an edge from each state in \( F_D \) to \( q_{\text{accept}} \), and each such edge is labeled with \( \epsilon \).

- By this construction, \( L(G) = L(M) \).
From DFAs to REs (proof 3/3)

- k–state DFA
- Add $q_{\text{start}}$ and $q_{\text{accept}}$.
- k+2–state GNFA
- Eliminate a state.
- k+1–state GNFA
- Regular expression
- 2–state GNFA
The coming week

Reading: Note: this is different than the schedule in the Sept. 3 notes – we’re nearly two lectures ahead of schedule.

September 17 (Today): Regular Expressions
   Read Sipser 1.3.

September 19 (Friday): Nonregular Languages – Read Sipser 1.4.
   Lecture will cover through Example 1.73 (i.e. pages 77-80).

September 22 (Monday): Pumping Lemma Examples.
   The rest of Sipser 1.4 (i.e. pages 80–82).

September 24 (A week from today): Introduction to Context Free Languages – Sipser 2.1.
   Lecture will cover through “Designing Context-Free Grammars” (i.e. pages 99-105).

Homework:

   September 19 (Friday): Homework 1 due. Homework 2 goes out (due Sept. 26).

Midterm: Oct. 8