LING/C SC/PSYC 438/538

Lecture 13

Sandiway Fong
From last time

Ungraded Homework 9

• apply the set-of-states construction technique to the two machines on the $\varepsilon$-transition slide (repeated below)
• self-check your answer:
  • verify in each case that the machine produced is deterministic and accurately simulates its $\varepsilon$-transition counterpart

![Diagram](image-url)
Ungraded Homework 9 Review

• Converting a NDFSA into a DFSA

\[\begin{array}{c}
\text{Note: this machine with an } \varepsilon\text{-transition is non-deterministic} \\
\end{array}\]

\[\begin{array}{c}
\text{Note: this machine is deterministic} \\
\end{array}\]

[Powerpoint animation]
Ungraded Homework 9 Review

• Converting a NDFSA into a DFSA

Note: this machine with an \( \varepsilon \)-transition is non-deterministic

Note: this machine is deterministic

[Powerpoint animation]
Perl Regular Expressions

- Perl regex can include backreferences to groupings (i.e. \1, etc.)
  - backreferences give Perl regexs expressive power beyond regular languages:
    ```perl
    /the (.*)er they (.*)/, the \1er we \2/
    ```
    will match *The faster they ran, the faster we ran* but not *The faster they ran. the faster we ate. These numbered memories are called registers (e.g. \$1).*
    - the set of prime numbers is **not** a regular language
      \[L_{prime} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\}\]
      can be proved using the Pumping Lemma for regular languages (later)
    - can have regular Perl code inside a regex
      ```perl
      $s = $ARGV[0];
      $string = "$s" x $s;
      if ($string ~ /\^(11+7)\1+$/) { print "$s factored using \2, length($s), \"not a prime\n"
    } else { print "$s is a prime\n"
    }
    ```
      ```perl
      $x = "This is a slightly simplified version of a rather complicated piece of Perl code."
      $x =~ s/\([^/]/chars\{\1\}\++;\1/eg;
      @list = sort {chars{sb} <=> chars{sa}} keys %chars;
      foreach $key (@list) { print "frequency of \'$key\' is \$chars{\$key}\n";
    ```
Backreferences and FSA

• Deep question:
  • why are backreferences impossible in FSA?

  Example: Suppose you wanted a machine that accepted /(a+b+)/\1/
  One idea: link two copies of the machine together

• Perl implementation:
  – how to modify it get the backreference effect?

```perl
$delta =
1   ("s", {"a", "x"}, "x", {"a", "x", "b", "y"}, "y", {"b","y", "a","x2"},
2   "x2", {"a","x2", "b","y2"}, "y2", {"b","y2"});
3
4 $state = "s";
5
6 foreach $c (split(//, ARGV[0])) {
7   $state = $delta{$state}{$c};
8 }
9 print (($state eq "y2") ? "Accept\n" : "Reject\n");
```
Regular Languages and FSA

• Formal (constructive) set-theoretic definition of a regular language

  1. \( \emptyset \) is a regular language
  2. \( \forall a \in \Sigma \cup \epsilon, \{a\} \) is a regular language
  3. If \( L_1 \) and \( L_2 \) are regular languages, then so are:
     (a) \( L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\} \), the concatenation of \( L_1 \) and \( L_2 \)
     (b) \( L_1 \cup L_2 \), the union or disjunction of \( L_1 \) and \( L_2 \)
     (c) \( L_1^* \), the Kleene closure of \( L_1 \)

• Correspondence between REs and Regular Languages
  • concatenation (juxtaposition)
  • union (also [ ])
  • Kleene closure (*) Note: \( x^+ = xx^* \)

• Note:
  • backreferences are memory devices and thus are too powerful
  • e.g. \( L = \{ww\} \) and prime number testing (see earlier lectures)
Regular Languages and FSA

- Closure properties: i.e. *do we still have a regular language afterwards?*

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersection</td>
<td>If $L_1$ and $L_2$ are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both $L_1$ and $L_2$.</td>
</tr>
<tr>
<td>Difference</td>
<td>If $L_1$ and $L_2$ are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in $L_1$ but not $L_2$.</td>
</tr>
<tr>
<td>Complementation</td>
<td>If $L_1$ is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren’t in $L_1$.</td>
</tr>
<tr>
<td>Reversal</td>
<td>If $L_1$ is a regular language, then so is $L_1^R$, the language consisting of the set of reversals of all the strings in $L_1$.</td>
</tr>
</tbody>
</table>

- Properties not necessarily preserved higher up: *e.g. context-free grammars as we’ll see later*
Equivalence: FSA and Regexs

*Textbook gives one direction only*

- case by case:
  - a) Empty string
  - b) Empty set
  - c) Any character from the alphabet

**Figure 2.22** Automata for the base case (no operators) for the induction showing that any regular expression can be turned into an equivalent automaton.
Equivalence: FSA and Regexs

- **Concatenation:**

  3. If $L_1$ and $L_2$ are regular languages, then so are:

     (a) $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$, the concatenation of $L_1$ and $L_2$

     (b) $L_1 \cup L_2$, the union or disjunction of $L_1$ and $L_2$

     (c) $L_1^*$, the Kleene closure of $L_1$

- Link final state of FSA$_1$ to initial state of FSA$_2$ using an empty transition

  ![Diagram of two FSAs concatenated with an empty transition](image)

  **Figure 2.23** The concatenation of two FSAs.

  **Note:** empty transition $\varepsilon$ can be deleted using the set of states construction
Equivalence: FSA and Regexs

• **Kleene closure:**

  3. If $L_1$ and $L_2$ are regular languages, then so are:

  - (a) $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$, the **concatenation** of $L_1$ and $L_2$
  - (b) $L_1 \cup L_2$, the **union** or **disjunction** of $L_1$ and $L_2$
  - (c) $L_1^*$, the **Kleene closure** of $L_1$

  • repetition operator: zero or more times
  • use empty transitions for loopback and bypass

![Diagram of FSA and Kleene closure](image-url)
Equivalence: FSA and Regexs

- **Union:** aka disjunction

  3. If \( L_1 \) and \( L_2 \) are regular languages, then so are:
     
     (a) \( L_1 \cdot L_2 = \{ xy | x \in L_1, y \in L_2 \} \), the concatenation of \( L_1 \) and \( L_2 \)
     
     (b) \( L_1 \cup L_2 \), the union or disjunction of \( L_1 \) and \( L_2 \)
     
     (c) \( L_1^* \), the Kleene closure of \( L_1 \)

- Non-deterministically run both FSAs at the same time, accept if either one accepts
Regular Languages and FSA

• Other closure properties:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersection</td>
<td>If $L_1$ and $L_2$ are regular languages, then so is $L_1 \cap L_2$, the</td>
</tr>
<tr>
<td></td>
<td>language consisting of the set of strings that are in both $L_1$ and $L_2$.</td>
</tr>
<tr>
<td>difference</td>
<td>If $L_1$ and $L_2$ are regular languages, then so is $L_1 - L_2$, the</td>
</tr>
<tr>
<td></td>
<td>language consisting of the set of strings that are in $L_1$ but not $L_2$.</td>
</tr>
<tr>
<td>complementation</td>
<td>If $L_1$ is a regular language, then so is $\Sigma^* - L_1$, the set of all</td>
</tr>
<tr>
<td></td>
<td>possible strings that aren’t in $L_1$.</td>
</tr>
<tr>
<td>reversal</td>
<td>If $L_1$ is a regular language, then so is $L_1^R$, the language consisting</td>
</tr>
<tr>
<td></td>
<td>of the set of reversals of all the strings in $L_1$.</td>
</tr>
</tbody>
</table>

Let’s consider building the FSA machinery for each of these guys in turn...
Regular Languages and FSA

• Other closure properties:

| intersection | if $L_1$ and $L_2$ are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both $L_1$ and $L_2$. |

Final state?
Regular Languages and FSA

• Other closure properties:

| difference | if $L_1$ and $L_2$ are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in $L_1$ but not $L_2$. |

Final state?
Regular Languages and FSA

• Other closure properties:

Example: $\Sigma^* = \{a,b\}$
need arcs for each character in $\Sigma$
## Regular Languages and FSA

- **Other closure properties:**

  - **reversal**: If $L_1$ is a regular language, then so is $L_1^R$, the language consisting of the set of reversals of all the strings in $L_1$.

  **reverse arrows and swap initial/final**
Regular Expressions from FSA

Recall textbook Exercise: find a RE for

4. the set of all strings from the alphabet $a,b$ such that each $a$ is immediately preceded by and immediately followed by a $b$;

Examples (* denotes string not in the language):
- $ab$  $ba$
- $bab$
- $\lambda$ (empty string)
- $bb$
- $*baba$
- $babab$
Regular Expressions from FSA

• Draw a FSA and convert it to a RE:

\[ b^* \ b \ (ab^+)^+ \]

\[ = b+(ab^+)^* | \varepsilon \]
Regular Expressions from FSA

• **Example Perl implementation:**
  
  ```perl
  $s = "ab ba bab bb baba babab";
  while ($s =~ /\b(b+(ab+))*\b/g) {
      print "<$1> match!\n";
  }
  ```

  **Note:** this doesn’t include the empty string case

• **Output:**

  perl test.perl
  <bab> match!
  <bb> match!
  <babab> match!

  **Note:** recall /../g global flag for multiple matches
Converting FSA to REs

• Example:
  • Give a RE for the FSA:

```
1 0 1
0 1 1
1 1 1
```

• State by-pass method:
  1. Delete one state at a time
  2. Calculate the possible paths passing through the deleted state
  3. Add the regex calculated at each stage as an arc
  • e.g.
    • eliminate state 3
    • then 2...
Converting FSA to REs

- eliminate state 3
- eliminate state 2

Answer: \((0(1^*0|1)*1^+1 \mid 1)^*\)