Lecture 3: Closure Properties & Regular Expressions

Jim Hook
Tim Sheard
Portland State University
Last Time

- Defined DFA, regular languages
- Defined NFA, showed equivalent to DFA
- Showed closure properties of Regular Languages
Why do we care about closure properties?

- One course objective is to “map the world”
- Closure properties tell us how to build new regular languages from old
Can properties define a Class of Languages?

What is the smallest class of languages:

That contains

the empty language
the universal language
every singleton character of the alphabet

And is closed under

union
concatenation
iteration (Kleene star)
Regular languages?

- Can the regular languages from last lecture be this smallest class?
- Since they meet the requirements they must at least contain this smallest class
  - Discuss
- How can we tell if there are regular languages not in this class?
Regular expressions

- Kleene introduced regular expressions (REGEXP) to name the languages in this “smallest class”
  - $a$ is a REGEXP for every $a$ in $\Sigma$
  - $\varepsilon$ is a REGEXP
  - $\emptyset$ is a REGEXP
  - if $R_1$ and $R_2$ are REGEXPs then the following are REGEXPs
    - $R_1 + R_2$
    - $R_1 \cdot R_2$
    - $R_1^*$
Regular expressions and Regular Languages

• Thm [1.54] A language is regular iff it is described by a regular expression
• Lemma [1.55] If a language is described by a regular expression then it is regular
• Proof sketch:
  – For each REGEXP we must show that a corresponding NFA can be constructed
  – We’ve done the hard work by proving the closure properties
  – We just have to complete the base cases for \{a\}, \{\varepsilon\}, and \emptyset.
Regular expressions and Regular Languages

• More interesting: can we convert a DFA $M$ into a regular expression?

• Lemma [1.60] If a language is regular then it is described by a regular expression
  - How do we prove this?
  - Can we calculate a REGEXP from a DFA?
DFA -> REGEXP

• One construction:
  – Draw a graph labeled essentially like the DFA
  – Find a way to remove states from the DFA systematically, replacing labels with regular expressions
  – Set things up so that when we are done the resulting regular expression describes the language accepted by the DFA
Generalized NFAs

• Generalize an NFA to have regular expressions labeling transitions
• Goal is to simplify an automaton to:

```
• Helpful to have single start and final state
```
Simplification (remove $q_{\text{rip}}$)

\[ q_i \xrightarrow{R_1} q_{\text{rip}} \xrightarrow{R_3} q_j \]

\[ q_i \xrightarrow{R_4} q_j \]

\[ R_1(R_2^*)R_3 + R_4 \]
Complete the transition relation by adding

1. epsilon transitions from start to all initial DFA states, and from all DFA final states to accept.
2. null transitions between all unlabeled DFA states,
Example (DFA)
Complete the transition relation by adding epsilon transitions from start to all initial DFA states, and from all DFA final states to accept.
null transitions between all unlabeled DFA states,
As Table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>ε</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>0</td>
<td>Ø</td>
<td>a+b</td>
<td>ε</td>
</tr>
<tr>
<td>1</td>
<td>a+b</td>
<td>Ø</td>
<td>Ø</td>
</tr>
</tbody>
</table>
Cut 1 (w.r.t all pairs)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>$\varepsilon$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td>$a+b$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>1</td>
<td>$a+b$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c|c}
\text{start} & 0 & 1 & \text{accept} \\
\hline
\varepsilon & \emptyset & \emptyset & \emptyset \\
\hline
\emptyset & a+b & \varepsilon & \emptyset \\
\hline
a+b & \emptyset & \emptyset & \emptyset \\
\end{array}
\]
Simplifying

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>$0$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$\varepsilon + \emptyset \emptyset^*(a+b)$</td>
<td>$\emptyset + \emptyset \emptyset^* \emptyset$</td>
</tr>
<tr>
<td>0</td>
<td>$\emptyset + (a+b) \emptyset^*(a+b)$</td>
<td>$\varepsilon + (a+b) \emptyset^* \emptyset$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>$0$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$\varepsilon$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>0</td>
<td>$(a+b)(a+b)$</td>
<td>$\varepsilon$</td>
</tr>
</tbody>
</table>
Cut 0 and simplify

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>$\varepsilon$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>0</td>
<td>$(a+b)(a+b)$</td>
<td>$\varepsilon$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>$\emptyset + \varepsilon((a+b)(a+b))^*$ $\varepsilon$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>$((a+b)(a+b))^*$</td>
</tr>
</tbody>
</table>
Example (conclusion)

\[(a+b)(a+b)^*\]
Proof Sketch

• Formalize GNFA
  – adjust delta to give REGEXP
  – define acceptance for GNFA
    • this will give a sequence of states visited on acceptance
• Show “ripping a state” preserves language accepted
  – Let G’ be obtained from G by ripping qrip
  – Show: $w \in L(G') \Rightarrow w \in L(G)$
  – Show: $w \in L(G) \Rightarrow w \in L(G')$
Proof Sketch (cont)

• Let $G'$ be obtained from $G$ by ripping $q_{\text{rip}}$
• Show: $w \in L(G') \Rightarrow w \in L(G)$
  – $w \in L(G')$ implies there is a sequence of states: $q_{\text{start}}$, $q_1$, ..., $q_{\text{accept}}$
    and substrings $w_1$, $w_2$, ..., $w_n$ satisfying the acceptance conditions
  – Look at each $w_i$, either
    • $w_i$ comes from an “R4” rule, or
    • $w_i$ comes from an R1 R2* R3 rule
  – If $w_i$ comes from an R4 rule then $G$ can make a corresponding step
  – If $w_i$ comes from an R1 R2* R3 rule, then $w_i$ is of the form $y_1$ ... $y_m$, where
    • $y_1 \in R1$,
    • $y_i \in R2$ ($1 < i < m$)
    • $y_m \in R3$
  – In this case $G$ transitions from $q_{i-1}$ to $q_i$ with $m-2$ intermediate instances of $q_{\text{rip}}$ on input $w_i = y_1$ ... $y_m$
Proof Sketch (cont)

• Let $G'$ be obtained from $G$ by ripping $q_{\text{rip}}$
• Show: $w \in L(G) \Rightarrow w \in L(G')$
• $w \in L(G)$ implies there are states $q_{\text{start}}$, $q_1$, …, $q_{\text{accept}}$ and strings $w_1$, …, $w_n$ satisfying conditions of acceptance
• Cases:
  – $q_{\text{rip}}$ not used in computation: $w$ clearly in $L(G')$ (use only R4 rules)
  – $q_{\text{rip}}$ is used:
    • every occurrence of $q_{\text{rip}}$ is in a context of the form:
      – $q_i$ $q_{\text{rip}}$ $q_{\text{rip}}$ … $q_{\text{rip}}$ $q_j$, in which there is one or more occurrences of $q_{\text{rip}}$ between non rip states $i$ and $j$.
      – In this case
        » $w_{i+1}$ will be an “R1” string
        » $w_{i+2}$, …, $w_{j-1}$ will be “R2” strings (there may be 0 of these)
        » $w_j$ will be an “R3” string
Proof Sketch (cont)

• Let $G'$ be obtained from $G$ by ripping $q_{rip}$
• Show: $w \in L(G) \Rightarrow w \in L(G')$
• Cases:
  – $q_{rip}$ is used:
    • every occurrence of $q_{rip}$ is in a context of the form:
      – $q_i q_{rip} q_{rip} \ldots q_{rip} q_j$, in which there is one or more occurrences of $q_{rip}$ between non rip states $i$ and $j$.
      – In this case
        » $w_{i+1}$ will be an “R1” string
        » $w_{i+2}, \ldots, w_{j-1}$ will be “R2” strings (there may be 0 of these)
        » $w_j$ will be an “R3” string
      – Consequently, $G'$ will transition from $q_i$ to $q_j$ on $w_{i+1} \ldots w_j$ by an R1 R2* R3 transition
Next time

• Non-regular languages (pumping lemma)