Regular Expressions

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science
Objectives

You should be able to...

- Explain the syntax of regular expressions.
- Explain the limitations of regular expressions.
- Know how to convert a regular expression into an NFA.
Motivation

- Regular Languages were developed by Noam Chomsky in his quest to describe human languages.
- Computer Scientists like them because they are able to describe “words” or “tokens” very easily.

Examples:

- **Integers**  a bunch of digits
  - **Reals**  an integer, a dot, and an integer
- **Past Tense English Verbs**  a bunch of letters ending with “ed”
- **Proper Nouns**  a bunch of letters, the first of which must be capitalized
A bunch of digits?!

- We need something a bit more formal if we want to communicate properly.
- We will use a *pattern* (or a *regular expression*) to represent the kinds of words we want to describe.
- As it will turn out, these expressions will correspond to NFAs.
- Kinds of patterns we will use:
  - Single letters
  - Repetition
  - Grouping
  - Choices
Single Letters

- To match a single character, just write the character.
- To match the letter “a”...
  - Regular Expression: a
  - State machine:

```
state machine:
q0  a  q1
start  q0  q1
```

- To match the character “8”...
  - Regular Expression: 8
  - State machine:

```
state machine:
q0  8  q1
start  q0  q1
```
Juxtaposition

- To match longer things, just put two regular expressions together.
- To match the character “a” followed by the character “8”...
  - Regular expression: a8
  - State machine:

  ![State machine diagram 1](image1)

- To match the string “hello”...
  - Regular expression: hello
  - State machine:

  ![State machine diagram 2](image2)
Repetition

- Zero or more copies of $A$, add $\ast$
  - Regular expression $A^*$
  - State machine:

- One or more copies of $A$, add $+$
  - Regular expression $A^+$
  - State machine:
Grouping

- To group things together, use parenthesis.
- To match one or more copies of the word “hi”...
  - Regular expression: (hi)+
  - State machine:

```
start  → q0    ε    → q1    h    → q2    i    → q3    ε    → q4
      q0        q1        q2        q3        q4
```

Choice

- To make a choice, use the vertical bar (also called “pipe”).
- To match \( A \) of \( B \).
  - Regular expression: \( A | B \)
  - State machine:
## Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>(Some) Matches</th>
<th>(Some) Rejects</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ab*a</code></td>
<td><code>aa, aba, abbba</code></td>
<td><code>ba, aaba, abaa</code></td>
</tr>
<tr>
<td>`(0</td>
<td>1)`*</td>
<td>any binary number, ( \epsilon )</td>
</tr>
<tr>
<td>`(0</td>
<td>1)`+</td>
<td>any binary number</td>
</tr>
<tr>
<td>`(0</td>
<td>1)`*0</td>
<td>even binary numbers</td>
</tr>
<tr>
<td><code>(aa)*a</code></td>
<td>odd number of as</td>
<td></td>
</tr>
<tr>
<td><code>(aa)*a(aa)</code>*</td>
<td>odd number of as</td>
<td></td>
</tr>
<tr>
<td>`(aa</td>
<td>bb)*((ab</td>
<td>ba)(aa</td>
</tr>
</tbody>
</table>
Some Notational Shortcuts

- A range of characters: \([Xa–z]\) matches X and between a and z (inclusively).
- Any character at all: .
- Escape: \`

<table>
<thead>
<tr>
<th>Expression</th>
<th>(Some) Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>([0–9])+</td>
<td>integers</td>
</tr>
<tr>
<td>X.*Y</td>
<td>anything at all between an X and a Y</td>
</tr>
<tr>
<td>([0–9])<em>.[0–9]</em></td>
<td>floating point numbers (positive, without exponents)</td>
</tr>
</tbody>
</table>
Things to know...

- They are *greedy*. 
  \( X .* Y \) will match \( XabaaYaababY \) entirely, not just \( XabaaY \).
- They *cannot count* very well.
  - They can only count as high as you have states in the machine.
  - This regular expression matches some primes:
    - \( aa|aaa|aaaaa|aaaaaaa \)
  - You cannot match an infinite number of primes.
  - You cannot match “nested comments”. \((\*.*\*)\)
Right Linear Grammars

A Right Linear Grammar is one in which every production has the form

\[ A \rightarrow x \]

or

\[ A \rightarrow xB \]

or

\[ A \rightarrow B \]

where \( A \) and \( B \) are arbitrary (possibly identical) nonterminal symbols, and \( x \) is an arbitrary terminal symbol.

- “At most one non-terminal symbol in the right hand side.”
- It turns out these are equivalent to NFAs!
- Have one nonterminal symbol for each state, one terminal symbol for each production.
Example 1

- Regular Expression: asdf
- State machine:

```
start → q0 → a → q1 → s → q2 → d → q3 → f → q4
```

- Grammar:

```
S_0 \rightarrow aS_1
S_1 \rightarrow sS_2
S_2 \rightarrow dS_3
S_3 \rightarrow fS_4
S_4 \rightarrow \epsilon
```
Example 2

- Regular Expression: \( a(s \mid d) + f \)

\[
\begin{align*}
S_0 &\rightarrow aS_1 \\
S_1 &\rightarrow sS_2 \\
&\quad | \quad dS_3 \\
S_2 &\rightarrow sS_2 \\
&\quad | \quad dS_3 \\
&\quad | \quad fS_4 \\
S_3 &\rightarrow sS_2 \\
&\quad | \quad dS_3 \\
&\quad | \quad fS_4
\end{align*}
\]
Going from Regular Expression to Right Linear Grammar

- One way: Regular Expression → NFA → DFA → RLG
- Another way: direct conversion. We’ll use a “bottom up” strategy.

**Characters** To convert a single character a, we make a simple production.

<table>
<thead>
<tr>
<th>Regexp: a</th>
<th>Regexp: b</th>
<th>Regexp: ab</th>
</tr>
</thead>
</table>
| $S_1 \rightarrow a$ | $S_2 \rightarrow b$ | $S_1 \rightarrow aS_2$
|             |           | $S_2 \rightarrow b$ |

**Concatenation** To concatenate two regular expressions, add the second start symbol to the end of any “accepting” states from the first grammar.
Choice and Repetition

**Choice**  To choose between two regular expressions, add a new start symbol that “picks” one of the choices.

Regexp: \(a\)

\[
S_1 \rightarrow a
\]

Regexp: \(b\)

\[
S_2 \rightarrow b
\]

Regexp: \(a \mid b\)

\[
S \rightarrow S_1 \mid S_2
\]

\[
S_1 \rightarrow a
\]

\[
S_2 \rightarrow b
\]

**Kleene Plus**  If \(S\) is the start symbol, then for every rule of the form \(A \rightarrow x\) (“accepting states”) add another rule of the form \(A \rightarrow xS\). You may have to remove \(\epsilon\) productions first.

Regexp: \((a \mid b)^+\)

\[
S \rightarrow S_1 \mid S_2
\]

\[
S_1 \rightarrow a \mid aS
\]

\[
S_2 \rightarrow b \mid bS
\]
Choice and Repetition

**Kleene Star**  If $S$ is the start symbol, then for every rule of the form $A \rightarrow x$ (“accepting states”) add another rule of the form $A \rightarrow xS$. Also add an $\epsilon$ rule.

Regexp: $a | b$

\[
\begin{align*}
S & \rightarrow S_1 | S_2 \\
S_1 & \rightarrow a \\
S_2 & \rightarrow b
\end{align*}
\]

Regexp: $(a | b)^*$

\[
\begin{align*}
S & \rightarrow S_1 | S_2 | \epsilon \\
S_1 & \rightarrow a | aS \\
S_2 & \rightarrow b | bS
\end{align*}
\]
Credits

The algorithm for converting a regular expression to a right linear grammar is based partly on the discussion here:
http://vasy.inria.fr/people/Gordon.Pace/Research/Software/Relic/Transformations/RE/toRG.html