Objectives
You should be able to...

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.
  - Regular Expression: a
  - State machine:

- To match the letter “a”...
  - Regular Expression: a
  - State machine:

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

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:

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

Repetition

- Zero or more copies of A, add *
  - Regular expression A*
  - State machine:

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

Grouping

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

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

```
Objectives
Regular Expressions
Syntax of Regular Expressions
Conversion to Right Linear Grammar

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>ab*a</td>
<td>aa, aba, abbb</td>
<td>ba, aaba, abaa</td>
</tr>
<tr>
<td>(0</td>
<td>1)*</td>
<td>any binary number, ϵ</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>(aa)*a</td>
<td>odd number of as</td>
<td></td>
</tr>
<tr>
<td>(aa)<em>a(aa)</em></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: \`

**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 | aaaaaaaaa
  - 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 2

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

\[
\begin{align*}
S_0 & \rightarrow aS_1 \\
S_1 & \rightarrow aS_2 \\
& \quad \mid dS_3 \\
S_2 & \rightarrow aS_2 \\
& \quad \mid dS_3 \\
& \quad \mid fS_4 \\
S_3 & \rightarrow aS_2 \\
& \quad \mid dS_3 \\
& \quad \mid fS_4 \\
\end{align*}
\]

Going from Regular Expression to Right Linear Grammar

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

Characters

To convert a single character \( a \), we make a simple production.

\[ S \rightarrow a \]

where \( S \) is the start symbol.

Concatenation

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

\[
\begin{align*}
\text{Regexp: } a & \rightarrow S_1 \\
S_1 & \rightarrow aS_2 \\
\text{Regexp: } ab & \rightarrow S_2 \\
S_2 & \rightarrow b \\
\end{align*}
\]
Choice and Repetition

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

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Syntax</th>
<th>Conversion to Right Linear Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$S_1$</td>
<td>$S_1 \rightarrow a$</td>
</tr>
<tr>
<td>$b$</td>
<td>$S_2$</td>
<td>$S_2 \rightarrow b$</td>
</tr>
<tr>
<td>$a \lor b$</td>
<td>$S$</td>
<td>$S \rightarrow S_1 \mid S_2$</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Syntax</th>
<th>Conversion to Right Linear Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \lor b$</td>
<td>$S$</td>
<td>$S \rightarrow S_1 \mid S_2$</td>
</tr>
<tr>
<td>$a$</td>
<td>$S_1$</td>
<td>$S_1 \rightarrow a$</td>
</tr>
<tr>
<td>$b$</td>
<td>$S_2$</td>
<td>$S_2 \rightarrow b$</td>
</tr>
<tr>
<td>$(a \lor b)^+$</td>
<td>$S$</td>
<td>$S \rightarrow S_1 \mid S_2 \mid aS \mid bS$</td>
</tr>
</tbody>
</table>

**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