Regular Expressions

- Another means to describe languages accepted by Finite Automata.

- In some books, regular languages, **by definition**, are described using regular expressions.

Specifying Languages

- Recall: how do we specify languages?
  - If language is finite, you can list all of its strings.
    - $L = \{a, aa, aba, aca\}$
  - Descriptive:
    - $L = \{ x \mid n_a(x) = n_b(x) \}$
  - Using basic Language operations
    - $L = (aa, ab)^* \cup (b)(bb)^*$
  - Regular languages are described using this last method

Regular Languages

- A regular expression describes a language using only the set operations of:
  - Union
  - Concatenation
  - Kleene Star

Kleene Star Operation

- The set of strings that can be obtained by concatenating any number of elements of a language $L$ is called the Kleene Star, $L^*$

$$L^* = \bigcup_{i=0}^{\infty} L^i \cup L^1 \cup L^2 \cup L^3 \cup \ldots$$

- Note that since, $L^*$ contains $L^0$, $\lambda$ is an element of $L^*$

Regular Expressions

- Regular expressions are the mechanism by which regular languages are described:
  - Take the "set operation" definition of the language and:
    - Replace $\cup$ with $+$
    - Replace $\{\}$ with $(\cdot)$
  - And you have a regular expression
Regular expressions

<table>
<thead>
<tr>
<th>{λ}</th>
<th>λ</th>
</tr>
</thead>
<tbody>
<tr>
<td>{011}</td>
<td>011</td>
</tr>
<tr>
<td>{0, 1}</td>
<td>0 + 1</td>
</tr>
<tr>
<td>{0, 01}</td>
<td>0 + 01</td>
</tr>
<tr>
<td>{110}*(0,1)</td>
<td>(110)∗(0+1)</td>
</tr>
<tr>
<td>{10, 11, 01}*</td>
<td>(10 + 11 + 01)*</td>
</tr>
<tr>
<td>(0, 11)∗((11)∗ ∪ {101, λ})</td>
<td>(0 + 11)∗((11)∗ + 101 + λ)</td>
</tr>
</tbody>
</table>

Regular Expression

- Recursive definition of regular languages / expression over Σ:
  1. ∅ is a regular language and its regular expression is ∅
  2. {λ} is a regular language and λ is its regular expression
  3. For each a ∈ Σ, {a} is a regular language and its regular expression is a

Regular Expressions

- Some shorthand
  - If we apply precedents to the operators, we can relax the full parenthesized definition:
    - Kleene star has highest precedent
    - Concatenation has mid precedent
    - + has lowest precedent
  - Thus
    - a + b′c is the same as (a + (b′)c))
    - (a + b)′ is not the same as a + b′

Regular Expressions

- More shorthand
  - Equating regular expressions.
    - Two regular expressions are considered equal if they describe the same language
      - 1"1' = 1'
      - (a + b)′ = a + b′

Regular Expressions

- Even more shorthand
  - Sometimes you might see in the book:
    - r^n where n indicates the number of concatenations of r (e.g. r^n)
    - r^+ to indicate one or more concatenations of r.
  - Note that this is only shorthand!
  - r^0 and r^+ are not regular expressions.
Regular Expressions

- Important thing to remember
  - A regular expression is not a language
  - A regular expression is used to describe a language.
    - It is incorrect to say that for a language \( L \),
      - \( L = (a + b + c)^* \)
    - But it's okay to say that \( L \) is described by
      - \( (a + b + c)^* \)

Examples of Regular Languages

- All finite languages can be described by regular expressions
  - Can anyone tell me why?
Examples of Regular Languages
- \( L = \{ x \in \{0,1\}^* \mid x \text{ contains an odd number of } 0 \text{s} \} \)
- Express \( x = yz \)
- \( y \) is a string of the form \( y=1^i01 \)
- In \( z \), there must be an even number of additional \( 0 \)s or \( z = (01^i01^m)^* \)
- \( x \) can be described by \( (1^i01^j)^* \)
- Questions?

Useful properties of regular expressions
- Commutative
  - \( L + M = M + L \)
- Associative
  - \( (L + M) + N = L + (M + N) \)
  - \( (LM)N = L(MN) \)
- Identities
  - \( \emptyset + L = L + \emptyset = L \)
  - \( \lambda L = L \lambda = L \)
  - \( \emptyset L = L \emptyset = \emptyset \)

Useful properties of regular expressions
- Distributed
  - \( L (M + N) = LM + LN \)
  - \( (M + N)L = ML + NL \)
- Idempotent
  - \( L + L = L \)

Useful properties of regular expressions
- Closures
  - \( (L^*)^* = L^* \)
  - \( \emptyset^* = \lambda \)
  - \( \lambda^* = \lambda \)
  - \( L^+ = LL^* \)
  - \( L^* = L^+ + \lambda \)
- Questions?

Practical uses for regular expressions
- grep
  - Global (search for) Regular Expressions and Print
  - Finds patterns of characters in a text file.
  - grep man foo.txt
  - grep [ab]*c[de]? foo.txt

Practical uses for regular expressions
- How a compiler works
Practical uses for regular expressions

How a compiler works
- The Lexical Analyzer (lexer) reads source code and generates a stream of tokens
- What is a token?
  - Identifier
  - Keyword
  - Number
  - Operator
  - Punctuation

Examples of Regular Languages

L = set of valid C keywords
- This is a finite set
- L can be described by
  - if + then + else + while + do + goto + break + switch + ...

Examples of Regular Languages

L = set of valid C identifiers
- A valid C identifier begins with a letter or _
- A valid C identifier contains letters, numbers, and _
- If we let:
  - l = \{a, b, ..., z, A, B, ..., Z\}
  - d = \{1, 2, ..., 9, 0\}
- Then a regular expression for L:
  - (l + _)(l + d + _)*

Practical uses for regular expressions

lex
- Program that will create a lexical analyzer.
- Input: set of valid tokens
- Tokens are given by regular expressions.

Questions?
For next time

- Chicken or the egg?
  - Which came first, the regular expression or the finite automata?
    - McCulloch/Pitts -- used finite automata to model neural networks (1943)
    - Kleene (mid 1950s) -- Applied to regular sets
    - Ken Thompson/ Bell Labs folk (1970s) -- QED / ed / grep / lex / awk / ...
  - Recall:
    - Princeton dudes (1937)

The bottom line

- Regular expressions and finite automata are equivalent in their ability to describe languages.
  - Every regular expression has a FA that accepts the language it describes
  - The language accepted by an FA can be described by some regular expression.
  - The Kleene Theorem! (1956)
  - But that's next time....

One last note:
Apps Using regular expressions

<table>
<thead>
<tr>
<th>Program</th>
<th>Original Author</th>
<th>Version</th>
<th>Expr Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>awk</td>
<td>Alex, Wenchang, Zhangsha</td>
<td>1.9Aa</td>
<td>YAP</td>
</tr>
<tr>
<td>lex</td>
<td>John, Raymond, Wang</td>
<td>1.4Aa</td>
<td>YAP</td>
</tr>
<tr>
<td>yacc</td>
<td>Jack, Bob, Alan</td>
<td>1.2A</td>
<td>PERTURBA</td>
</tr>
<tr>
<td>sed</td>
<td>Jean, John, Mark</td>
<td>1.0</td>
<td>YAP</td>
</tr>
<tr>
<td>lex</td>
<td>Jack, Bob, Alan</td>
<td>1.2A</td>
<td>YAP</td>
</tr>
<tr>
<td>awk</td>
<td>Alex, Wenchang, Zhangsha</td>
<td>1.9Aa</td>
<td>YAP</td>
</tr>
<tr>
<td>yacc</td>
<td>Jack, Bob, Alan</td>
<td>1.2</td>
<td>PERTURBA</td>
</tr>
<tr>
<td>sed</td>
<td>Jean, John, Mark</td>
<td>1.2</td>
<td>YAP</td>
</tr>
<tr>
<td>lex</td>
<td>Jack, Bob, Alan</td>
<td>1.1</td>
<td>YAP</td>
</tr>
<tr>
<td>awk</td>
<td>Alex, Wenchang, Zhangsha</td>
<td>1.9Aa</td>
<td>YAP</td>
</tr>
<tr>
<td>yacc</td>
<td>Jack, Bob, Alan</td>
<td>1.2</td>
<td>PERTURBA</td>
</tr>
<tr>
<td>sed</td>
<td>Jean, John, Mark</td>
<td>1.1</td>
<td>YAP</td>
</tr>
<tr>
<td>lex</td>
<td>Jack, Bob, Alan</td>
<td>1.0</td>
<td>YAP</td>
</tr>
</tbody>
</table>