CMSC 330: Organization of Programming Languages

Regular Expressions and Finite Automata
How do regular expressions work?

What we’ve learned

• What regular expressions are
• What they can express, and cannot
• Programming with them

What’s next: how they work

• A great computer science result
Languages and Machines

![Diagram illustrating the hierarchy of languages and machines (CMSC 330 Spring 2021)]
A Few Questions About REs

- How are REs implemented?
  - Given an arbitrary RE and a string, how to decide whether the RE matches the string?

- What are the basic components of REs?
  - Can implement some features in terms of others
    - E.g., e+ is the same as ee*

- What does a regular expression represent?
  - Just a set of strings
    - This observation provides insight on how we go about our implementation

- ... next comes the math!
Definition: Alphabet

- An **alphabet** is a finite set of symbols
  - Usually denoted $\Sigma$

Example alphabets:
- Binary: $\Sigma = \{0,1\}$
- Decimal: $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$
- Alphanumeric: $\Sigma = \{0-9,a-z,A-Z\}$
Definition: String

A string is a finite sequence of symbols from $\Sigma$

- $\epsilon$ is the empty string (""" in Ruby)
- $|s|$ is the length of string $s$
  - $|\text{Hello}| = 5$, $|\epsilon| = 0$
- Note
  - $\emptyset$ is the empty set (with 0 elements)
  - $\emptyset \neq \{ \epsilon \}$ (and $\emptyset \neq \epsilon$)

Example strings over alphabet $\Sigma = \{0,1\}$ (binary):

- 0101
- 0101110
- 0101110
- $\epsilon$
Definition: Language

- A language $L$ is a set of strings over an alphabet

- Example: All strings of length 1 or 2 over alphabet $\Sigma = \{a, b, c\}$ that begin with $a$
  - $L = \{ a, aa, ab, ac \}$

- Example: All strings over $\Sigma = \{a, b\}$
  - $L = \{ \varepsilon, a, b, aa, bb, ab, ba, aaa, bba, aba, baa, \ldots \}$
  - Language of all strings written $\Sigma^*$

- Example: All strings of length 0 over alphabet $\Sigma$
  - $L = \{ s | s \in \Sigma^* \text{ and } |s| = 0 \}$
  - “the set of strings $s$ such that $s$ is from $\Sigma^*$ and has length 0”
  - $= \{ \varepsilon \} \neq \emptyset$
Definition: Language (cont.)

Example: The set of phone numbers over the alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 9, (, ), -\}$

- Give an example element of this language: $\text{(123) 456-7890}$
- Are all strings over the alphabet in the language? No
- Is there a Ruby regular expression for this language? No

Example: The set of all valid (runnable) Ruby programs

- Later we’ll see how we can specify this language
- (Regular expressions are useful, but not sufficient)
Operations on Languages

- Let $\Sigma$ be an alphabet and let $L, L_1, L_2$ be languages over $\Sigma$

- **Concatenation** $L_1 L_2$ creates a language defined as
  - $L_1 L_2 = \{ xy \mid x \in L_1 \text{ and } y \in L_2 \}$

- **Union** creates a language defined as
  - $L_1 \cup L_2 = \{ x \mid x \in L_1 \text{ or } x \in L_2 \}$

- **Kleene closure** creates a language is defined as
  - $L^* = \{ x \mid x = \varepsilon \text{ or } x \in L \text{ or } x \in LL \text{ or } x \in LLL \text{ or } \ldots \}$
Operations Examples

Let $L_1 = \{a, b\}$, $L_2 = \{1, 2, 3\}$ (and $\Sigma = \{a, b, 1, 2, 3\}$)

- **What is $L_1L_2$?**
  - $\{a1, a2, a3, b1, b2, b3\}$

- **What is $L_1 \cup L_2$?**
  - $\{a, b, 1, 2, 3\}$

- **What is $L_1^*$?**
  - $\{\epsilon, a, b, aa, bb, ab, ba, aaa, aab, bba, bbb, aba, abb, baa, bab, \ldots\}$
Quiz 1: Which string is **not** in $L_3$

$L_1 = \{a, \, ab, \, c, \, d, \, \varepsilon\}$ \hspace{1cm} \text{where} \Sigma = \{a,b,c,d\}$

$L_2 = \{d\}$

$L_3 = L_1 \cup L_2$

A. a
B. ad
C. $\varepsilon$
D. d
Quiz 1: Which string is not in $L_3$

$L_1 = \{a, ab, c, d, \varepsilon\}$  \hspace{1cm} where  $\Sigma = \{a,b,c,d\}$

$L_2 = \{d\}$

$L_3 = L_1 \cup L_2$

A. a  
B. ad  
C. $\varepsilon$  
D. d
Quiz 2: Which string is not in $L_3$

$L_1 = \{a, ab, c, d, \varepsilon\}$ where $\Sigma = \{a,b,c,d\}$
$L_2 = \{d\}$
$L_3 = L_1(L_2^*)$

A. a  
B. abd  
C. adad  
D. abdd
Quiz 2: Which string is not in $L_3$?

$L_1 = \{a, \text{ab}, c, d, \varepsilon\}$ \hspace{1cm} where $\Sigma = \{a,b,c,d\}$
$L_2 = \{d\}$
$L_3 = L_1(L_2^*)$

A. $a$
B. $\text{abd}$
C. $\text{adad}$
D. $\text{abdd}$
Regular Expressions: Grammar

- We can define a grammar for regular expressions \( R \)

\[ R ::= \emptyset \quad \text{The empty language} \]
\[ | \varepsilon \quad \text{The empty string} \]
\[ | \sigma \quad \text{A symbol from alphabet } \Sigma \]
\[ | R_1 R_2 \quad \text{The concatenation of two regexps} \]
\[ | R_1 | R_2 \quad \text{The union of two regexps} \]
\[ | R^* \quad \text{The Kleene closure of a regexp} \]
Regular Languages

- Regular expressions denote languages. These are the regular languages
  - *aka* regular sets
- Not all languages are regular
  - Examples (without proof):
    - The set of palindromes over $\Sigma$
    - $\{a^n b^n \mid n > 0\}$ ($a^n$ = sequence of $n$ a’s)
- Almost all programming languages are not regular
  - But aspects of them sometimes are (e.g., identifiers)
  - Regular expressions are commonly used in parsing tools
Semantics: Regular Expressions (1)

Given an alphabet $\Sigma$, the regular expressions over $\Sigma$ are defined inductively as follows.

**Constants**

<table>
<thead>
<tr>
<th>regular expression</th>
<th>denotes language</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>${\varepsilon}$</td>
</tr>
<tr>
<td>each symbol $\sigma \in \Sigma$</td>
<td>${\sigma}$</td>
</tr>
</tbody>
</table>

*Ex: with $\Sigma = \{a, b\}$, regex $a$ denotes language $\{a\}$*

*regex $b$ denotes language $\{b\}$*
Semantics: Regular Expressions (2)

- Let $A$ and $B$ be regular expressions denoting languages $L_A$ and $L_B$, respectively. Then:

  **Operations**

<table>
<thead>
<tr>
<th>regular expression</th>
<th>denotes language</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AB$</td>
<td>$L_A L_B$</td>
</tr>
<tr>
<td>$A</td>
<td>B$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>$L_A^*$</td>
</tr>
</tbody>
</table>

- There are no other regular expressions over $\Sigma$
Terminology etc.

- Regexps apply operations to symbols
  - Generates a set of strings (i.e., a language)
    - (Formal definition shortly)
  - Examples
    - a generates language \{a\}
    - a|b generates language \{a\} \cup \{b\} = \{a, b\}
    - a* generates language \{\epsilon\} \cup \{a\} \cup \{aa\} \cup \ldots = \{\epsilon, a, aa, \ldots \}

- If \(s \in\) language \(L\) generated by a RE \(r\), we say that \(r\) accepts, describes, or recognizes string \(s\)
Precedence

Order in which operators are applied is:

•kleene closure * > concatenation > union |
•ab|c = ( a b ) | c → {ab, c}
•ab* = a ( b* ) → {a, ab, abb ...}
•a|b* = a | ( b* ) → {a, ε, b, bb, bbb ...}

We use parentheses ( ) to clarify

•E.g., a(b|c), (ab)*, (a|b)*
•Using escaped \(\) if parens are in the alphabet
Ruby Regular Expressions

Almost all of the features we’ve seen for Ruby REs can be reduced to this formal definition

- /Ruby/ – concatenation of single-symbol REs
- /(Ruby|Regular)/ – union
- /(Ruby)*/ – Kleene closure
- /(Ruby)+/ – same as (Ruby)(Ruby)*
- /(Ruby)?/ – same as (ε|(Ruby))
- /[a-z]/ – same as (a|b|c|...|z)
- /[^0-9]/ – same as (a|b|c|...) for a,b,c,... ∈ Σ - {0..9}
- ^, $ – correspond to extra symbols in alphabet

Think of every string containing a distinct, hidden symbol at its start and at its end – these are written ^ and $
Implementing Regular Expressions

- We can implement a regular expression by turning it into a finite automaton
  - A “machine” for recognizing a regular language
Finite Automaton

Elements
- States $S$ (start, final)
- Alphabet $\Sigma$
- Transition edges $\delta$
Finite Automaton

- Machine starts in **start** or **initial** state
- Repeat until the end of the string \( s \) is reached
  - Scan the next symbol \( \sigma \in \Sigma \) of the string \( s \)
  - Take transition edge labeled with \( \sigma \)
- String \( s \) is **accepted** if automaton is in **final** state when end of string \( s \) is reached

**Elements**
- States \( S \) (\( start, final \))
- Alphabet \( \Sigma \)
- Transition edges \( \delta \)
Finite Automaton: States

- **Start state**
  - State with incoming transition from no other state
  - Can have only one start state

- **Final states**
  - States with double circle
  - Can have zero or more final states
  - Any state, including the start state, can be final
Finite Automaton: Example 1

0 0 1 0 1 1

Accepted?

Yes
Finite Automaton: Example 2

0 0 1 0 1 0

Accepted?
No
Quiz 3: What Language is This?

A. All strings over \{0, 1\}
B. All strings over \{1\}
C. All strings over \{0, 1\} of length 1
D. All strings over \{0, 1\} that end in 1
Quiz 3: What Language is This?

A. All strings over \{0, 1\}
B. All strings over \{1\}
C. All strings over \{0, 1\} of length 1
D. All strings over \{0, 1\} that end in 1

regular expression for this language is (0|1)*1
Finite Automaton: Example 3

(a,b,c notation shorthand for three self loops)
Finite Automaton: Example 3

(a,b,c notation shorthand for three self loops)

<table>
<thead>
<tr>
<th>string</th>
<th>state at end</th>
<th>accept s?</th>
</tr>
</thead>
<tbody>
<tr>
<td>aabcc</td>
<td>S2</td>
<td>Y</td>
</tr>
</tbody>
</table>
Finite Automaton: Example 3

(a,b,c notation shorthand for three self loops)
Finite Automaton: Example 3

(a,b,c notation shorthand for three self loops)

<table>
<thead>
<tr>
<th>string</th>
<th>state at end</th>
<th>accept s?</th>
</tr>
</thead>
<tbody>
<tr>
<td>acca</td>
<td>S3</td>
<td>N</td>
</tr>
</tbody>
</table>
Finite Automaton: Example 3

(a,b,c notation shorthand for three self loops)

<table>
<thead>
<tr>
<th>string</th>
<th>state at end</th>
<th>accept s?</th>
</tr>
</thead>
<tbody>
<tr>
<td>aacbbb</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CMSC 330 Spring 2021
Finite Automaton: Example 3

(a,b,c notation shorthand for three self loops)

<table>
<thead>
<tr>
<th>string</th>
<th>state at end</th>
<th>accept s?</th>
</tr>
</thead>
<tbody>
<tr>
<td>aacbbb</td>
<td>S3</td>
<td>N</td>
</tr>
</tbody>
</table>
Quiz 4: Which string is **not** accepted?

(a, b, c notation shorthand for three self loops)

A. bcca  
B. abbbc  
C. ccc  
D. ε
Quiz 4: Which string is **not** accepted?

(a,b,c notation shorthand for three self loops)

A. bcca
B. abbbc
C. ccc
D. $\varepsilon$
What language does this FA accept?

S3 is a dead state – a nonfinal state with no transition to another state
- aka a trap state

a*b*c*
Finite Automaton: Example 4

Language?

$\text{a}^*\text{b}^*\text{c}^*$ again, so FAs are not unique
If a transition is omitted, assume it goes to a dead state that is not shown.

Language?
- Strings over \{0,1,2,3\} with alternating even and odd digits, beginning with odd digit.
Finite Automaton: Example 5

Description for each state

- S0 = “Haven't seen anything yet” OR “Last symbol seen was a b”
- S1 = “Last symbol seen was an a”
- S2 = “Last two symbols seen were ab”
- S3 = “Last three symbols seen were abb”
Finite Automaton: Example 5

- **Language as a regular expression?**
  - \((a|b)^*abb\)
Over $\Sigma=\{a,b\}$, this FA accepts only:

A. A string that contains a single b.
B. Any string in $\{a,b\}$.
C. A string that starts with b followed by a’s.
D. One or more b’s, followed by zero or more a’s.
Over $\Sigma=\{a,b\}$, this FA accepts only:

A. A string that contains a single $b$.
B. Any string in $\{a,b\}$.
C. A string that starts with $b$ followed by $a$’s.
D. One or more $b$’s, followed by zero or more $a$’s.
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings containing two consecutive 0s followed by two consecutive 1s
- That accepts strings with an odd number of 1s
- That accepts strings containing an even number of 0s and any number of 1s
- That accepts strings containing an odd number of 0s and odd number of 1s
- That accepts strings that **DO NOT** contain odd number of 0s and an odd number of 1s
Exercises: Define an FA over $\Sigma = \{0, 1\}$

- That accepts strings with an odd number of 1s
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings with an odd number of 1s
Exercises: Define an FA over $\Sigma = \{a,b\}$

- That accepts strings containing an even number of a’s and any number of b’s
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings containing an even number of 0s and any number of 1s
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings containing two consecutive 0s followed by two consecutive 1s
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings containing two consecutive 0s very immediately (right after, no other things in between) followed by two consecutive 1s
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings end with two consecutive 0s followed by two consecutive 1s
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings end with two consecutive 0s followed by two consecutive 1s
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings containing an odd number of 0s and odd number of 1s
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings containing an odd number of 0s and odd number of 1s

4 states:

0s  1s
e  e  e
o  e  e
o  o  o
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings that DO NOT contain odd number of 0s and an odd number of 1s
Exercises: Define an FA over $\Sigma = \{0,1\}$

- That accepts strings that **DO NOT** contain odd number of 0s and an odd number of 1s

Flip each state