Regular Expressions

Problem Set Four is due using a late period in the box up front.
Concatenation

- The **concatenation** of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

  \[ L_1L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \} \]

- Intuitively, the set of all strings formed by concatenating some string from $L_1$ and some string from $L_2$.

- Conceptually similar to the Cartesian product of two sets, only with strings.
Language Exponentiation

- We can define what it means to “exponentiate” a language as follows:
  - $L^0 = \{ \varepsilon \}$
    - The set containing just the empty string.
    - Idea: Any string formed by concatenating zero strings together is the empty string.
  - $L^{n+1} = LL^n$
    - Idea: Concatenating $(n+1)$ strings together works by concatenating $n$ strings, then concatenating one more.
The Kleene Closure

• An important operation on languages is the **Kleene Closure**, which is defined as

\[ L^* = \bigcup_{i=0}^{\infty} L^i \]

• Mathematically:

\[ w \in L^* \iff \exists n \in \mathbb{N}. \ w \in L^n \]

• Intuitively, all possible ways of concatenating any number of copies of strings in \( L \) together.
Closure Properties

• The regular languages are closed under the following operations:
  • Complementation
  • Union
  • Intersection
  • Concatenation
  • Kleene closure
Another View of Regular Languages
Rethinking Regular Languages

- We currently have several tools for showing a language is regular.
  - Construct a DFA for it.
  - Construct an NFA for it.
  - Apply closure properties to existing languages.
- We have not spoken much of this last idea.
Constructing Regular Languages

• **Idea**: Build up all regular languages as follows:
  
  • Start with a small set of simple languages we already know to be regular.
  
  • Using closure properties, combine these simple languages together to form more elaborate languages.

• **A bottom-up approach to the regular languages.**
Regular Expressions

- **Regular expressions** are a family of descriptions that can be used to capture the regular languages.

- Often provide a compact and human-readable description of the language.

- Used as the basis for numerous software systems (Perl, `flex`, `grep`, etc.)
Atomic Regular Expressions

- The regular expressions begin with three simple building blocks.
- The symbol $\emptyset$ is a regular expression that represents the empty language $\emptyset$.
- The symbol $\epsilon$ is a regular expression that represents the language $\{ \epsilon \}$.
- This is not the same as $\emptyset$!
- For any $a \in \Sigma$, the symbol $a$ is a regular expression for the language $\{ a \}$.
Compound Regular Expressions

- We can combine together existing regular expressions in four ways.
- If $R_1$ and $R_2$ are regular expressions, $R_1 R_2$ is a regular expression for the **concatenation** of the languages of $R_1$ and $R_2$.
- If $R_1$ and $R_2$ are regular expressions, $R_1 \mid R_2$ is a regular expression for the **union** of the languages of $R_1$ and $R_2$.
- If $R$ is a regular expression, $R^*$ is a regular expression for the **Kleene closure** of the language of $R$.
- If $R$ is a regular expression, $(R)$ is a regular expression with the same meaning as $R$. 
Operator Precedence

- Regular expression operator precedence:
  
  $(R)$
  
  $R^*$
  
  $R_1R_2$
  
  $R_1 | R_2$

- So $ab^*c|d$ is parsed as $((a(b^*))c) | d$
Regular Expression Examples

- The regular expression `trick|treat` represents the regular language `{ trick, treat }

- The regular expression `booo*` represents the regular language `{ boo, booo, boooo, ... }

- The regular expression `candy!(candy!)*` represents the regular language `{ candy!, candy!candy!, candy!candy!candy!, ... }`
Regular Expressions, Formally

- The **language of a regular expression** is the language described by that regular expression.
- Formally:
  - \( L(\varepsilon) = \{ \varepsilon \} \)
  - \( L(\emptyset) = \emptyset \)
  - \( L(a) = \{ a \} \)
  - \( L(R_1 R_2) = L(R_1) L(R_2) \)
  - \( L(R_1 | R_2) = L(R_1) \cup L(R_2) \)
  - \( L(R^*) = L(R)^* \)
  - \( L((R)) = L(R) \)

Worthwhile activity: Apply this recursive definition to \( a(b|c)((d)) \) and see what you get.
Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains $00$ as a substring $\}$
  
  $(0 \mid 1)^*00(0 \mid 1)^*$

11011100101
0000
1111101111001111

Regular Expressions are Awesome

• Let $\Sigma = \{0, 1\}$

• Let $L = \{ w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring } \}$
Regular Expressions are Awesome

Let $\Sigma = \{0, 1\}$

Let $L = \{ w \in \Sigma^* | |w| = 4 \}$

The length of a string $w$ is denoted $|w|$
Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$
Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

\[
\Sigma^4
\]

\[
\begin{align*}
0000 \\
1010 \\
1111 \\
1000
\end{align*}
\]
Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* | \text{w contains at most one 0} \}$

$1^* (0 \mid \varepsilon) 1^*$

$11110111$
$111111$
$0111$
$0$
Regular Expressions are Awesome

• Let $\Sigma = \{0, 1\}$
• Let $L = \{ w \in \Sigma^* \mid w \text{ contains at most one } 0 \}$
Regular Expressions are Awesome

- Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”

- Regular expression for email addresses:
Regular Expressions are Awesome

• Let $\Sigma = \{a, ., @\}$, where $a$ represents “some letter.”

• Regular expression for email addresses:

\[
\text{aa*(.aa*)*@aa*.aa*(.aa*)*)}
\]

\[
\text{cs103@cs.stanford.edu}
\]
\[
\text{first.middle.last@mail.site.org}
\]
\[
\text{barack.obama@whitehouse.gov}
\]
Regular Expressions are Awesome

• Let $\Sigma = \{ \text{a, ., @} \}$, where $\text{a}$ represents “some letter.”

• Regular expression for email addresses:

$$a^+ (.a^+)* @ a^+ (.a^+)^+$$

$$\text{cs103@cs.stanford.edu}$$
$$\text{first.middle.last@mail.site.org}$$
$$\text{barack.obama@whitehouse.gov}$$
Regular Expressions are Awesome

\[ a^+(.a^+)^*@a^+(.a^+)^+ \]
Shorthand Summary

- $R^n$ is shorthand for $RR \ldots R$ ($n$ times).
- $\Sigma$ is shorthand for “any character in $\Sigma$.”
- $R?$ is shorthand for $(R \mid \varepsilon)$, meaning “zero or one copies of $R$.”
- $R^+$ is shorthand for $RR^*$, meaning “one or more copies of $R$.”
Break for Announcements!
Midterm Logistics

• Midterm is tomorrow, October 29, from 7PM - 10PM

• Room determined by last name:
  • A – G: Go to **Gates B01**
  • H – K: Go to **Gates B03**
  • L – P: Go to **200-002**
  • Q – V: Go to **420-041**
  • W – Z: Go to **Herrin T175**
Your Questions
When writing a logic statement, do you have to include the universal or existential quantifier for every variable that you state? I thought you had to, but this one from lecture doesn't:

\[ Tallest(x) \rightarrow \forall y. \ (x \neq y \rightarrow IsShorterThan(y, x)) \]

This example is a “sentence fragment” in first-order logic; without a definition of \( x \), this isn't a valid statement. All variables need to be quantified.
"When writing first-order logic statements with quantifiers, which one out of the following would be correct?

\[ \forall x \ P(x) \cdot \exists y \ R(y) \]

or

\[ \forall x. \ (P(x) \rightarrow \exists y. \ R(y)) \]
If you find that the function $f: A \rightarrow B$ is not surjective, have you proven that $|A| < |B|$? Or do you still need to do additional proof steps?

$$f : \mathbb{N} \rightarrow \mathbb{N}$$

$$f(n) = 137$$
“What is the best thing to do to prepare for the exam between now and 7PM tomorrow?”
“Is there some mathematical automaton that can determine whether or not two first-order logical statements are equivalent?”

More on that later in the quarter...
Back to Regular Expressions!
The Power of Regular Expressions

**Theorem:** If $R$ is a regular expression, then $\mathcal{L}(R)$ is regular.

**Proof idea:** Show how to convert a regular expression into an NFA.
A Marvelous Construction

• The following theorem proves the language of any regular expression is regular:

  **Theorem:** For any regular expression $R$, there is an NFA $N$ such that

  \[ L(R) = L(N) \]

  • $N$ has exactly one accepting state.
  • $N$ has no transitions into its start state.
  • $N$ has no transitions out of its accepting state.
Base Cases

Automaton for $\varepsilon$

Automaton for $\emptyset$

Automaton for single character $a$
Construction for $R_1R_2$
Construction for $R_1 \mid R_2$

Machine for $R_1$

Machine for $R_2$
Construction for $R^*$

Machine for $R$

start $\varepsilon$ $\varepsilon$ $\varepsilon$ $\varepsilon$
Why This Matters

• Many software tools work by matching regular expressions against text.

• One possible algorithm for doing so:
  • Convert the regular expression to an NFA.
  • (Optionally) Convert the NFA to a DFA using the subset construction.
  • Run the text through the finite automaton and look for matches.

• Runs extremely quickly!
The Power of Regular Expressions

**Theorem:** If \( L \) is a regular language, then there is a regular expression for \( L \).

This is not obvious!

**Proof idea:** Show how to convert an arbitrary NFA into a regular expression.
From NFAs to Regular Expressions

Regular expression: \((s_1 \mid s_2 \mid \ldots \mid s_n)^*\)

Key idea: Label transitions with arbitrary regular expressions.
From NFAs to Regular Expressions

Key idea: If we can convert any NFA into something that looks like this, we can easily read off the regular expression.

Regular expression: $R$
From NFAs to Regular Expressions
From NFAs to Regular Expressions
From NFAs to Regular Expressions

Could we eliminate this state from the NFA?
From NFAs to Regular Expressions

Note: We’re using concatenation and Kleene closure in order to skip this state.
Note: We’re using union to combine these transitions together.
From NFAs to Regular Expressions

\[ R_{11} \ast R_{12} \ (R_{22} \mid R_{21} R_{11} \ast R_{12}) \ast \varepsilon \]
From NFAs to Regular Expressions

\[ R_{11}^* R_{12} \left( R_{22} \mid R_{21} R_{11}^* R_{12} \right)^* \]
From NFAs to Regular Expressions

\[ R_{11} * R_{12} (R_{22} | R_{21} R_{11} * R_{12})^* \]
The Construction at a Glance

- Start with an NFA for the language $L$.
- Add a new start state $q_s$ and accept state $q_f$ to the NFA.
  - Add $\varepsilon$-transitions from each original accepting state to $q_f$, then mark them as not accepting.
- Repeatedly remove states other than $q_s$ and $q_f$ from the NFA by “shortcutting” them until only two states remain: $q_s$ and $q_f$.
- The transition from $q_s$ to $q_f$ is then a regular expression for the NFA.
Our Transformations

DFA → NFA, subset construction
NFA → Regexp, recursive transform
DFA → Regexp, direct conversion
NFA → DFA, state elimination
**Theorem:** The following are all equivalent:

- $L$ is a regular language.
- There is a DFA $D$ such that $\mathcal{L}(D) = L$.
- There is an NFA $N$ such that $\mathcal{L}(N) = L$.
- There is a regular expression $R$ such that $\mathcal{L}(R) = L$. 
Next Time

- **Applications of Regular Languages**
  - Answering “so what?”
- **Intuiting Regular Languages**
  - What makes a language regular?
- **The Pumping Lemma**
  - Proving languages aren't regular.