Regular Languages and Expressions

Lecture 2
Thursday, January 17, 2019
Part I

Regular Languages
Regular Languages

A class of simple but useful languages.
The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

1. $\emptyset$ is a regular language.
2. $\{\epsilon\}$ is a regular language.
3. $\{a\}$ is a regular language for each $a \in \Sigma$. Interpreting $a$ as a string of length 1.
Regular Languages

A class of simple but useful languages. The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

1. $\emptyset$ is a regular language.
2. $\{\epsilon\}$ is a regular language.
3. $\{a\}$ is a regular language for each $a \in \Sigma$. Interpreting $a$ as string of length 1.
4. If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular.
Regular Languages

A class of simple but useful languages. The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

1. $\emptyset$ is a regular language.
2. $\{\epsilon\}$ is a regular language.
3. $\{a\}$ is a regular language for each $a \in \Sigma$. Interpreting $a$ as string of length 1.
4. If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular.
5. If $L_1, L_2$ are regular then $L_1 L_2$ is regular.
Regular Languages

A class of simple but useful languages.

The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

1. $\emptyset$ is a regular language.
2. $\{\epsilon\}$ is a regular language.
3. $\{a\}$ is a regular language for each $a \in \Sigma$. Interpreting $a$ as a string of length 1.
4. If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular.
5. If $L_1, L_2$ are regular then $L_1L_2$ is regular.
6. If $L$ is regular, then $L^* = \bigcup_{n \geq 0} L^n$ is regular. The $^*$ operator name is Kleene star.
Regular Languages

A class of simple but useful languages. The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

1. $\emptyset$ is a regular language.
2. $\{\epsilon\}$ is a regular language.
3. $\{a\}$ is a regular language for each $a \in \Sigma$. Interpreting $a$ as string of length 1.
4. If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular.
5. If $L_1, L_2$ are regular then $L_1 L_2$ is regular.
6. If $L$ is regular, then $L^* = \cup_{n \geq 0} L^n$ is regular. The $\cdot^*$ operator name is Kleene star.

Regular languages are closed under the operations of union, concatenation and Kleene star.
Some simple regular languages

Lemma

If $w$ is a string then $L = \{ w \}$ is regular.

Example: $\{aba\}$ or $\{abbabbab\}$. Why?

Let $w$ be an arbitrary string in $\Sigma^*$.

For any $x \in \Sigma^*$ such that $1 \leq |x| < |w|$, $L = \{ x^3 \}$ is regular.

$w = \varepsilon$ \implies $L = \{ \varepsilon \}$ is regular.

$w = ax$ : $L = \{ \omega \in \Sigma^* : \omega x^3 \} = \{ a \} \cdot \{ x^3 \}$

\[ L = L \cup L \]

regular by def

regular by def

regular by IH
Lemma

If \( w \) is a string then \( L = \{ w \} \) is regular.

Example: \( \{aba\} \) or \( \{abbabbab\} \). Why?

Lemma

Every finite language \( L \) is regular.

Examples: \( L = \{a, abaab, aba\} \). \( L = \{w \mid |w| \leq 100\} \). Why?

\[
\{a3 \cup \{abaab3 \cup \{aba3 \}
\}
More Examples

- \{w \mid w\text{ is a keyword in Python program}\}
- \{w \mid w\text{ is a valid date of the form mm/dd/yy}\}
- \{w \mid w\text{ describes a valid Roman numeral}\}
  \{I, II, III, IV, V, VI, VII, VIII, IX, X, XI, \ldots\}\).
- \{w \mid w\text{ contains ”CS374” as a substring}\}.

\[\subseteq^* \{\text{CS374}\} \subseteq^*\]
\[\subseteq = \{a, b, \ldots, i, j, \ldots, A, B, C \ldots\}\]

\[AB \text{ CS374}\]
\[AB \cdot \text{CS374} - \varepsilon\]
Part II

Regular Expressions
Regular Expressions

A way to denote regular languages

- simple **patterns** to describe related strings
- useful in
  - text search (editors, Unix/grep, emacs)
  - compilers: lexical analysis
  - compact way to represent interesting/useful languages
  - dates back to 50’s: Stephen Kleene
    who has a star names after him.
Inductive Definition

A regular expression $r$ over an alphabet $\Sigma$ is one of the following:

**Base cases:**
- $\emptyset$ denotes the language $\emptyset$
- $\epsilon$ denotes the language $\{\epsilon\}$
- $a$ denote the language $\{a\}$
Inductive Definition

A regular expression \( r \) over an alphabet \( \Sigma \) is one of the following:

**Base cases:**
- \( \emptyset \) denotes the language \( \emptyset \)
- \( \epsilon \) denotes the language \( \{ \epsilon \} \)
- \( a \) denote the language \( \{ a \} \).

**Inductive cases:** If \( r_1 \) and \( r_2 \) are regular expressions denoting languages \( R_1 \) and \( R_2 \) respectively then,
- \( (r_1 + r_2) \) denotes the language \( R_1 \cup R_2 \)
- \( (r_1r_2) \) denotes the language \( R_1R_2 \)
- \( (r_1)^* \) denotes the language \( R_1^* \)
## Regular Languages vs Regular Expressions

### Regular Languages

- $\emptyset$ regular
- $\{\varepsilon\}$ regular
- $\{a\}$ regular for $a \in \Sigma$
- $R_1 \cup R_2$ regular if both are
- $R_1R_2$ regular if both are
- $R^*$ is regular if $R$ is

### Regular Expressions

- $\emptyset$ denotes $\emptyset$
- $\varepsilon$ denotes $\{\varepsilon\}$
- $a$ denote $\{a\}$
- $r_1 + r_2$ denotes $R_1 \cup R_2$
- $r_1r_2$ denotes $R_1R_2$
- $r^*$ denote $R^*$

Regular expressions denote regular languages — they explicitly show the operations that were used to form the language.
For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!  
**Example:** $(0 + 1)$ and $(1 + 0)$ denote same language $\{0, 1\}$
For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language! **Example:** $(0 + 1)$ and $(1 + 0)$ denote same language $\{0, 1\}$

Two regular expressions $r_1$ and $r_2$ are **equivalent** if $L(r_1) = L(r_2)$. 

---

**Notation and Parenthesis**

- Omit parenthesis by adopting precedence order: $\ast$, concatenate, $\circ$.
- Example: $r \ast s + t = ((r \ast s) + t)$.
- Omit parenthesis by associativity of each of these operations.
- Example: $rst = (rs)t = r(st)$, $r + s + t = r + (s + t) = (r + s) + t$.
- Superscript $\circ$. For convenience, define $r^\circ = rr^\ast$. Hence if $L(r) = R$ then $L(r^\circ) = R + R$.
- Other notation: $r + s$, $r \cup s$, $r \mid s$ all denote union. $rs$ is sometimes written as $r \cdot s$. 

---

Chan, Har-Peled, Hassanieh (UIUC)
Notation and Parenthesis

- For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language! **Example:** $(0 + 1)$ and $(1 + 0)$ denote the same language $\{0, 1\}$.
- Two regular expressions $r_1$ and $r_2$ are equivalent if $L(r_1) = L(r_2)$.
- Omit parenthesis by adopting precedence order: $\ast$, concatenate, $\ast$.
  **Example:** $r^\ast s + t = ((r^\ast)s) + t$.
- Omit parenthesis by associativity of each of these operations.
  **Example:** $rst = (rs)t = r(st)$, $r + s + t = r + (s + t) = (r + s) + t$.
- Superscript $\ast$. For convenience, define $r^\ast r^\ast = rr^\ast$. Hence if $L(r) = R$ then $L(r^\ast) = R + r^\ast$.
- Other notation: $r + s$, $r \cup s$, $r \mid s$ all denote union. $rs$ is sometimes written as $r \cdot s$. 
For a regular expression \( r \), \( L(r) \) is the language denoted by \( r \). Multiple regular expressions can denote the same language!

**Example:** \((0 + 1)\) and \((1 + 0)\) denote the same language \(\{0, 1\}\)

Two regular expressions \( r_1 \) and \( r_2 \) are equivalent if 
\[ L(r_1) = L(r_2). \]

Omit parenthesis by adopting precedence order: \(*\), concatenate, \(+\).

**Example:** \( r^* s + t = ((r^*)s) + t \)

Omit parenthesis by associativity of each of these operations.

**Example:** \( rst = (rs)t = r(st) \), 
\[ r + s + t = r + (s + t) = (r + s) + t. \]
Notation and Parenthesis

- For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!
  
  **Example:** $(0 + 1)$ and $(1 + 0)$ denote the same language $\{0, 1\}$

- Two regular expressions $r_1$ and $r_2$ are equivalent if $L(r_1) = L(r_2)$.

- Omit parenthesis by adopting precedence order: $\ast$, concatenate, $+$. 
  
  **Example:** $r^\ast s + t = ((r^\ast)s) + t$

- Omit parenthesis by associativity of each of these operations.
  
  **Example:** $rst = (rs)t = r(st)$,
  
  $r + s + t = r + (s + t) = (r + s) + t$.

- **Superscript $+$**. For convenience, define $r^+ = rr^\ast$. Hence if $L(r) = R$ then $L(r^+) = R^+$.
Notation and Parenthesis

- For a regular expression \( r \), \( L(r) \) is the language denoted by \( r \). Multiple regular expressions can denote the same language!
  
  **Example:** \( (0 + 1) \) and \( (1 + 0) \) denote the same language \( \{0, 1\} \)

- Two regular expressions \( r_1 \) and \( r_2 \) are equivalent if 
  \[ L(r_1) = L(r_2). \]

- Omit parenthesis by adopting precedence order: \( *, \) concatenate, \( + \).
  
  **Example:** \( r^*s + t = ((r^*)s) + t \)

- Omit parenthesis by associativity of each of these operations.
  
  **Example:** \( rst = (rs)t = r(st) \),
  
  \( r + s + t = r + (s + t) = (r + s) + t \).

- **Superscript** \( + \). For convenience, define \( r^+ = r^*r \). Hence if 
  \( L(r) = R \) then \( L(r^+) = R^+ \).

- **Other notation:** \( r + s, r \cup s, r|s \) all denote union. \( rs \) is sometimes written as \( r \cdot s \).
Given a language $L$ “in mind” (say an English description) we would like to write a regular expression for $L$ (if possible).
Skills

- Given a language $L$ “in mind” (say an English description) we would like to write a regular expression for $L$ (if possible)
- Given a regular expression $r$ we would like to “understand” $L(r)$ (say by giving an English description)
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
  
  \[0^* + 1^*\]
  
  \(\epsilon\) or string of all 0s or string of all 1s
Understanding regular expressions

- $(0 + 1)^*$: set of all strings over $\{0, 1\}$
- $(0 + 1)^*001(0 + 1)^*$: strings with $001$ as substring
- $0^* + (0^*10^*10^*10^*10^*)^*$: strings with number of $1$'s divisible by $3$
Understanding regular expressions

- $(0 + 1)^*$: set of all strings over $\{0, 1\}$
- $(0 + 1)^*001(0 + 1)^*$: strings with $001$ as substring
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
- \(0^* + (0^*10^*10^*10^*)^*\): strings with number of 1's divisible by 3
- \(\emptyset\): set of all strings over \(\{\}\)
- \(\epsilon\) + 1\((01)^*\) + 0\((1 + 10)^*\)\: alternating 0s and 1s. Alternatively, no two consecutive 0s and no two consecutive 1s.
Understanding regular expressions

- $(0 + 1)^*$: set of all strings over $\{0, 1\}$
- $(0 + 1)^*001(0 + 1)^*$: strings with 001 as substring
- $0^* + (0^*10^*10^*10^*)^*$: strings with number of 1’s divisible by 3

\[
\begin{align*}
001010 & \in \mathcal{L}(0^*10^*10^*) \\
110 & \in \mathcal{L}(010^*10^*) \\
1110 & \notin \mathcal{L}(010^*10^*)
\end{align*}
\]
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
- \(0^* + (0^*10^*10^*10^*)^*\): strings with number of 1’s divisible by 3
- \(\emptyset\):
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
- \(0^* + (0^*10^*10^*10^*)^*\): strings with number of 1’s divisible by 3
- \(\emptyset\): \(\{\}\)
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
- \(0^* + (0^*10^*10^*10^*)^*\): strings with number of 1’s divisible by 3
- \(\emptyset0\): \(\{\}\)
- \((\epsilon + 1)(01)^*(\epsilon + 0)\):
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with \(001\) as substring
- \(0^* + (0^*10^*10^*10^*)^*\): strings with number of 1's divisible by 3
- \(\emptyset\): \(\{\}\)
- \((\epsilon + 1)(01)^*(\epsilon + 0)\): alternating 0s and 1s. Alternatively, no two consecutive 0s and no two consecutive 1s

\((\epsilon + 0)(10)^*(\epsilon + 1)\)
(0 + 1)*: set of all strings over \{0, 1\}
(0 + 1)*001(0 + 1)*: strings with 001 as substring
0* + (0*10*10*10*)*: strings with number of 1’s divisible by 3
∅0: {} 
(ε + 1)(01)*(ε + 0): alternating 0s and 1s. Alternatively, no two consecutive 0s and no two consecutive 1s
(ε + 0)(1 + 10)*:
(0 + 1)*: set of all strings over {0, 1}

(0 + 1)*001(0 + 1)*: strings with 001 as substring

0* + (0*10*10*10*)*: strings with number of 1’s divisible by 3

∅0: {} 

(ɛ + 1)(01)*(ɛ + 0): alternating 0s and 1s. Alternatively, no two consecutive 0s and no two consecutive 1s

(ɛ + 0)(1 + 10)*: strings without two consecutive 0s.
Creating regular expressions

- bitstrings with the pattern 001 or the pattern 100 occurring as a substring

\[ (0+1)^* (001 + 100) (0+1)^* \]
\[ (0+1)^* 001 (0+1)^* + (0+1)^* 100 (0+1)^* \]
\[ (3+0+1)^* 0 01 (2+0+1)^* \]

- \( r = 1001 \)
- \( \rightarrow 0100 \not\in L(r) \)
Creating regular expressions

- bitstrings with the pattern 001 or the pattern 100 occurring as a substring
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)
Creating regular expressions

- bitstrings with the pattern 001 or the pattern 100 occurring as a substring
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)
- bitstrings with an even number of 1’s
  \(0^* + (0^*1010^*)^*\)

Hard: bitstrings with an odd number of 1s and an odd number of 0s.
Creating regular expressions

- bitstrings with the pattern 001 or the pattern 100 occurring as a substring
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)
- bitstrings with an even number of 1's
  one answer: \(0^* + (0^*10^*10^*)^*\)

Hard: bitstrings with an odd number of 1s and an odd number of 0s.
Creating regular expressions

- bitstrings with the pattern **001** or the pattern **100** occurring as a substring
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)

- bitstrings with an even number of 1’s
  one answer: \(0^* + (0*10*10*)^* = r\)

- bitstrings with an odd number of 1’s
  \(r1r\)
  \(1rX\)
  \(010\)
  \(1rX\)
  \(r1X\)
Creating regular expressions

- bitstrings with the pattern 001 or the pattern 100 occurring as a substring
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)

- bitstrings with an even number of 1’s
  one answer: \(0^* + (0^*10^*10^*)^*\)

- bitstrings with an odd number of 1’s
  one answer: \(r1r\) where \(r\) is solution to previous part
Creating regular expressions

- bitstrings with the pattern **001** or the pattern **100** occurring as a substring
  
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)

- bitstrings with an even number of **1**’s
  
  one answer: \(0^* + (0^*10^*10^*)^*\)

- bitstrings with an odd number of **1**’s
  
  one answer: \(r1r\) where \(r\) is solution to previous part

- bitstrings that do not contain **01** as a substring
  
  \(1^*01^*\)
Creating regular expressions

- bitstrings with the pattern 001 or the pattern 100 occurring as a substring
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)

- bitstrings with an even number of 1’s
  one answer: \(0^* + (0^*10^*10^*)^*\)

- bitstrings with an odd number of 1’s
  one answer: \(r1r \text{ where } r \text{ is solution to previous part}\)

- bitstrings that do \textit{not} contain 01 as a substring
  one answer: \(1^*0^*\)
Creating regular expressions

- bitstrings with the pattern **001** or the pattern **100** occurring as a substring
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)
- bitstrings with an even number of **1**’s
  one answer: \(0^* + (0^*10^*10^*)^*\)
- bitstrings with an odd number of **1**’s
  one answer: \(r1r\) where \(r\) is solution to previous part
- bitstrings that do not contain **01** as a substring
  one answer: \(1^*0^*\)
- bitstrings that do not contain **011** as a substring
  \[1^*0^*10^* + \varepsilon\]
  \[10101\]
Creating regular expressions

- bitstrings with the pattern 001 or the pattern 100 occurring as a substring
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)

- bitstrings with an even number of 1's
  one answer: \(0^* + (0^*10^*10^*)^*\)

- bitstrings with an odd number of 1's
  one answer: \(r1r\) where \(r\) is solution to previous part

- bitstrings that do not contain 01 as a substring
  one answer: \(1^*0^*\)

- bitstrings that do not contain 011 as a substring
  one answer: \(1^*0^*(100^*)^*(1 + \varepsilon)\)
Creating regular expressions

- bitstrings with the pattern 001 or the pattern 100 occurring as a substring
  one answer: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)

- bitstrings with an even number of 1’s
  one answer: \(0^* + (0^*10^*10^*)^*\)

- bitstrings with an odd number of 1’s
  one answer: \(r1r\) where \(r\) is solution to previous part

- bitstrings that do not contain 01 as a substring
  one answer: \(1^*0^*\)

- bitstrings that do not contain 011 as a substring
  one answer: \(1^*0^*(100^*)^*(1 + \epsilon)\)

- Hard: bitstrings with an odd number of 1s and an odd number of 0s.
Bit strings with odd number of 0s and 1s

The regular expression is

$$(00 + 11)^* (01 + 10)$$

$$\left(00 + 11 + (01 + 10)(00 + 11)^* (01 + 10)\right)^*$$

(Solved using techniques to be presented in the following lectures...)
Regular expression identities

- $r^* r^* = r^*$ meaning for any regular expression $r$,
  $L(r^* r^*) = L(r^*)$
- $(r^*)^* = r^*$
- $rr^* = r^* r$
- $(rs)^* r = r(sr)^*$
- $(r + s)^* = (r^* s^*)^* = (r^* + s^*)^* = (r + s^*)^* = \ldots$
Regular expression identities

- $r^* r^* = r^*$ meaning for any regular expression $r$, $L(r^* r^*) = L(r^*)$
- $(r^*)^* = r^*$
- $rr^* = r^* r$
- $(rs)^* r = r(sr)^*$
- $(r + s)^* = (r^* s^*)^* = (r^* + s^*)^* = (r + s^*)^* = \ldots$

**Question:** How does one prove an identity?
Regular expression identities

- $r^* r^* = r^*$ meaning for any regular expression $r$, $L(r^* r^*) = L(r^*)$
- $(r^*)^* = r^*$
- $rr^* = r^* r$
- $(rs)^* r = r(sr)^*$
- $(r + s)^* = (r^* s^*)^* = (r^* + s^*)^* = (r + s^*)^* = \ldots$

**Question:** How does one prove an identity?
By induction. On what?
Regular expression identities

- $r^* r^* = r^*$ meaning for any regular expression $r$,
  $L(r^* r^*) = L(r^*)$
- $(r^*)^* = r^*$
- $rr^* = r^* r$
- $(rs)^* r = r(sr)^*$
- $(r + s)^* = (r^* s^*)^* = (r^* + s^*)^* = (r + s^*)^* = \ldots$

**Question:** How does one prove an identity? 
By induction. On what? Length of $r$ since $r$ is a string obtained from specific inductive rules.
A non-regular language and other closure properties

Consider \( L = \{0^n1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\} \).

Theorem \( L \) is not a regular language.

How do we prove it?

Other questions:

Suppose \( R_1 \) is regular and \( R_2 \) is regular. Is \( R_1 \cap R_2 \) regular?

Suppose \( R_1 \) is regular is \( \bar{R}_1 \) (complement of \( R_1 \)) regular?
Consider $L = \{0^n1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\}$.

**Theorem**

$L$ is not a regular language.
A non-regular language and other closure properties

Consider \( L = \{0^n1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\} \).

**Theorem**

\( L \) is not a regular language.

How do we prove it?
A non-regular language and other closure properties

Consider $L = \{0^n1^n | n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\}$.

**Theorem**

* $L$ is not a regular language.

How do we prove it?

Other questions:

- Suppose $R_1$ is regular and $R_2$ is regular. Is $R_1 \cap R_2$ regular?
- Suppose $R_1$ is regular is $\overline{R_1}$ (complement of $R_1$) regular?