Regular Languages and Expressions

Lecture 2
January 19, 2017
Part I

Regular Languages
Regular Languages

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

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
- If $L_1, L_2$ are regular then $L_1 \cdot L_2$ is regular
- If $L$ is regular, then $L^* = \bigcup_{n \geq 0} L^n$ is regular

Regular languages are closed under the operations of union, concatenation and Kleene star.
Regular Languages

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

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language

Regular languages are closed under the operations of union, concatenation and Kleene star.
Regular Languages

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

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1

Regular languages are closed under the operations of union, concatenation and Kleene star.
Regular Languages

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

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
Regular Languages

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

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
- If $L_1, L_2$ are regular then $L_1 \cdot L_2$ is regular

Regular languages are closed under the operations of union, concatenation and Kleene star.
Regular Languages

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

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
- If $L_1, L_2$ are regular then $L_1 \cdot L_2$ is regular
- If $L$ is regular, then $L^* = \bigcup_{n \geq 0} L^n$ is regular
Regular Languages

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

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
- If $L_1, L_2$ are regular then $L_1 \cdot L_2$ is regular
- If $L$ is regular, then $L^* = \bigcup_{n \geq 0} L^n$ is regular

Regular languages are closed under the operations of union, concatenation and Kleene star.
Regular Languages

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

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
- If $L_1, L_2$ are regular then $L_1 \cdot L_2$ is regular
- If $L$ is regular, then $L^* = \cup_{n \geq 0} L^n$ is regular

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 \( \{abbabbbab\} \). Why?

\[
\{abc\} = \{a\}.\{b\}.\{c\}
\]
Some simple regular languages

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?
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}\}.

\[ \Sigma^* \cdot \{\text{cs374}\} \cdot \Sigma^* \]
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 $\{a\}$.
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

<table>
<thead>
<tr>
<th>Regular Languages</th>
<th>Regular Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset) regular</td>
<td>(\emptyset) denotes (\emptyset)</td>
</tr>
<tr>
<td>({\epsilon}) regular</td>
<td>(\epsilon) denotes ({\epsilon})</td>
</tr>
<tr>
<td>({a}) regular for (a \in \Sigma)</td>
<td>(a) denote ({a})</td>
</tr>
<tr>
<td>(R_1 \cup R_2) regular if both are</td>
<td>(r_1 + r_2) denotes (R_1 \cup R_2)</td>
</tr>
<tr>
<td>(R_1R_2) regular if both are</td>
<td>(r_1r_2) denotes (R_1R_2)</td>
</tr>
<tr>
<td>(R^*) is regular if (R) is</td>
<td>(r^<em>) denote (R^</em>)</td>
</tr>
</tbody>
</table>

Regular expressions denote regular languages — they explicitly show the operations that were used to form the language

\[
(0+1)^* = \left(\{0,3,0,3,1\}\right)^* = \Sigma^*
\]

\[
00110^* = \emptyset
\]
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 $\{0, 1\}$
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 $L(r_1) = L(r_2)$. 

Omit parenthesis by adopting precedence order: $\ast, \text{concat}, +$.

Example: 

$$r \ast s + t = ((r \ast s) + t) = (r + (s + t)) = (r + s) + t.$$

Omit parenthesis by associativity of each of these operations.

Example: 

$$(rs)t = (rs)t = r(st).$$

Superscript $+$. For convenience, define $r^+ = r \ast r$. 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 $r \cdot s$. 

Chandra Chekuri (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: $\ast$, concat, $\pm$.
  - **Example**: $r^\ast s + t = ((r^\ast)s) + t$
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 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: $\ast$, concat, $+$. 
  **Example:** $r^*s + t = ((r^*)s) + t$
- Omit parenthesis by associativity of each of these operations.
  **Example:** $rst = (rs)t = r(st)$, $r + s + t = r + (s + t) = (r + s) + t$. 
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 $L(r_1) = L(r_2)$.

Omit parenthesis by adopting precedence order: $\ast$, concat, $\plus$. 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 $\plus$. For convenience, define $r^+ = rr^\ast$. Hence if $L(r) = R$ then $L(r^+) = R^+$.
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 $L(r_1) = L(r_2)$.

Omit parenthesis by adopting precedence order: $\ast$, concat, $\oplus$. 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 $\oplus$. For convenience, define $r^+ = rr^*$. 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)
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 $\{0, 1\}$
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 $3$
- $\emptyset$: the empty set
- $(\epsilon + 1)(01)^*$: strings without two consecutive $0$s.
- $(\epsilon + 0)(1 + 10)^*$: strings without two consecutive $0$s.
(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
- $0^* + (0*10*10*10^*)^*$:
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
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 $3$
- $\emptyset$: $\{\}$
- $(\epsilon + 1)(01)^*(\epsilon + 0)$:
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with \(001\) as substring
- \(0^* + (01010101^*)^*\): 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
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
- \(\emptyset 0\): \(\{\}\)
- \((\epsilon + 1)(01)^*(\epsilon + 0)\): alternating 0s and 1s. Alternatively, no two consecutive 0s and no two consecutive 1s
- \((\epsilon + 0)(1 + 10)^*\):
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
- $(\epsilon + 0)(1 + 10)^*$: strings without two consecutive 0s.
Creating regular expressions

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

One answer:

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

One answer:

\[0^* + (0^* 10^* 10^* 10^*)^*\]

One answer:

\[0^* 1\]

Where \(r\) is the solution to the previous part

Bitstrings that do not contain 011 as a substring

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: \((0 + 1)^*001(0 + 1)^* + (0 + 1)^*100(0 + 1)^*\)
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
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: \(0^* + (0^*10^*10^*)^*\)
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
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: \(0^*1r\) 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: $0^*1r$ where $r$ is solution to previous part

- bitstrings that do not contain $011$ as a substring
Creating regular expressions

- bitstrings with the parttern 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: $0^*1r$ where $r$ is solution to previous part
- bitstrings that do not contain 011 as a substring
- Hard: bitstrings with an odd number of 1s and an odd number of 0s.
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 $r$, $L(r^* r^*) = L(r^*)$
- $(r^*)^* = r^*$
- $r r^* = r^* r$
- $(rs)^* r = r(sr)^*$
- $(r + s)^* = (r^* s^*)^* = (r^* + s^*)^* = (r + s^*)^* = \ldots$

**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.
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 $\overline{R_1}$ (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 \( L = \{ 0^n1^n \mid n \geq 0 \} = \{ \epsilon, 01, 0011, 000111, \ldots \} \).

**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?