Regular Languages and Expressions

Lecture 2

January 23, 2025

Background

Fix some finite alphabet Σ.

- Σ^* is the set of all strings over Σ
- A language over Σ is a subset of strings. That is, $L \subseteq \Sigma^*$
- Σ^* is countably infinite. Set of all languages $= \mathcal{P}(\Sigma^*)$ is uncountably infinite
- Each machine/program can be described by a string. Hence set of machines/programs is countably infinite
- Implies many/most languages that are too "complex" for machines/programs

Background

Fix some finite alphabet Σ.

- Σ^* is the set of all strings over Σ
- A language over Σ is a subset of strings. That is, $L \subseteq \Sigma^*$
- Σ^* is countably infinite. Set of all languages $= \mathcal{P}(\Sigma^*)$ is uncountably infinite
- Each machine/program can be described by a string. Hence set of machines/programs is countably infinite
- Implies many/most languages that are too "complex" for machines/programs

Question: What languages are easy? What languages should we focus on? Can we *classify* them via various features?

Languages

Study of languages motivated by (among many others)

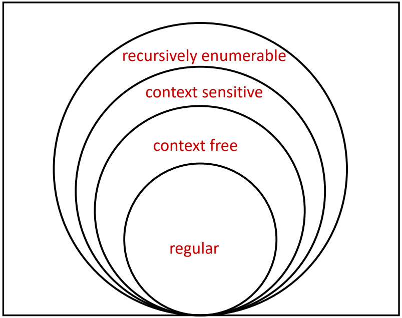
- linguistics and natural language understanding
- programming languages and logic
- computation and machines

Intution: As ability of a language to *express/model* increases the more *complex/computationally hard* it becomes.

Chomsky Hierarchy and Machines

Grammars		Machines
phrase structured		Turing machine (TMss)
context sensitive		linear bounded automata (LBAs)
context free		pushdown automata (PDAs)
regular expressions		finite state automata (DFAs)

All Languages



Part I

Regular Languages

Regular Languages

A class of simple but very useful languages.

The set of regular languages over some alphabet Σ is defined inductively/recursively via the following rules:

- $\emptyset$ is a regular language

Regular Languages

A class of simple but very useful languages.

The set of regular languages over some alphabet Σ is defined inductively/recursively via the following rules:

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

Regular Languages

A class of simple but very useful languages.

The set of regular languages over some alphabet Σ is defined inductively/recursively via the following rules:

- $\emptyset$ 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

A class of simple but very useful languages.

The set of regular languages over some alphabet Σ is defined inductively/recursively via the following rules:

- $\emptyset$ 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 Σ is defined inductively/recursively via the following rules:

- $\emptyset$ 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 $L_1 L_2$ is regular

Regular Languages

A class of simple but very useful languages.
The set of regular languages over some alphabet Σ is defined inductively/recursively via the following rules:

- $\emptyset$ 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 $L_1 L_2$ is regular
- If L is regular, then $L^* = \cup_{n \geq 0} L^n$ is regular

Regular Languages

A class of simple but very useful languages.
The set of regular languages over some alphabet Σ is defined inductively/recursively via the following rules:

- $\emptyset$ 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 $L_1 L_2$ is regular
- If L is regular, then $L^* = \cup_{n \geq 0} L^n$ is regular

Regular Languages

A class of simple but very useful languages.

The set of regular languages over some alphabet Σ is defined inductively/recursively via the following rules:

- $\emptyset$ 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 $L_1 L_2$ 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 $\{abbabbab\}$. Why?

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

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

Regular Languages

- How expressive are these languages?
- What can we use them for?
- What are limitations? That is, what can be *not* express as regular languages?

Part II

Regular Expressions

Regular Expressions

A way to denote/describe/represent 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 named after him.

Inductive Definition

A regular expression r over an alphabhe Σ is one of the following:

Base cases:

- $\emptyset$ denotes the language $\emptyset$
- ϵ denotes the language $\{\epsilon\}$.
- a denote the language $\{a\}$.

Inductive Definition

A regular expression r over an alphabhe Σ is one of the following:

Base cases:

- $\emptyset$ denotes the language $\emptyset$
- ϵ 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 $R_1 \cup R_2$
- $(r_1 r_2)$ denotes the language $R_1 R_2$
- $(r_1)^*$ denotes the language R_1^*

Regular Languages vs Regular Expressions

Regular Languages	**Regular Expressions**
$\emptyset$ regular	$\emptyset$ denotes $\emptyset$
$\{\epsilon\}$ regular	ϵ denotes $\{\epsilon\}$
$\{a\}$ regular for $a \in \Sigma$	a denote $\{a\}$
$R_1 \cup R_2$ regular if both are	$\mathrm{r}_1 + \mathrm{r}_2$ denotes $R_1 \cup R_2$
$R_1 R_2$ regular if both are	$\mathrm{r}_1\mathrm{r}_2$ denotes $R_1 R_2$
R^* is regular if R is	r^* denote R^*

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

Regular Languages vs Regular Expressions

Regular Languages	Regular Expressions
$\emptyset$ regular	$\emptyset$ denotes $\emptyset$
$\{\epsilon\}$ regular	ϵ denotes $\{\epsilon\}$
$\{a\}$ regular for $a \in \Sigma$	a denote $\{a\}$
$R_1 \cup R_2$ regular if both are	$\mathrm{r}_1 + \mathrm{r}_2$ denotes $R_1 \cup R_2$
$R_1 R_2$ regular if both are	$\mathrm{r}_1\mathrm{r}_2$ denotes $R_1 R_2$
R^* is regular if R is	r^* denote R^*

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

Examples: $(0 + 1)^*$, $010^* + (110)^*$, $(10 + 110)^*(11 + 10)$

Notation and Parenthesis

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

Notation and Parenthesis

- For a regular expression r, $\boldsymbol{L}(\mathrm{r})$ 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 $\boldsymbol{L}(\mathrm{r}_1) = \boldsymbol{L}(\mathrm{r}_2)$.

Notation and Parenthesis

- For a regular expression r, $\boldsymbol{L}(\mathrm{r})$ 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 $\boldsymbol{L}(\mathrm{r}_1) = \boldsymbol{L}(\mathrm{r}_2)$.
- Omit parenthesis by adopting precedence order: $*$, concat, $+$.
  Example: $\boldsymbol{r}^*\boldsymbol{s} + \boldsymbol{t} = ((\boldsymbol{r}^*)\boldsymbol{s}) + \boldsymbol{t}$

Notation and Parenthesis

- For a regular expression r, $\boldsymbol{L}(\mathrm{r})$ 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 $\boldsymbol{L}(\mathrm{r}_1) = \boldsymbol{L}(\mathrm{r}_2)$.
- Omit parenthesis by adopting precedence order: $*$, concat, $+$.
  Example: $r^*s + t = ((r^*)s) + t$
- Omit parenthesis by associativity of each of these operations.
  Example: $rst = (rs)t = r(st)$,
  $r + s + t = r + (s + t) = (r + s) + t$.

Notation and Parenthesis

- For a regular expression r, $\boldsymbol{L}(\mathrm{r})$ 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 $\boldsymbol{L}(\mathrm{r}_1) = \boldsymbol{L}(\mathrm{r}_2)$.
- Omit parenthesis by adopting precedence order: $*$, concat, $+$.
  Example: $\boldsymbol{r}^*\boldsymbol{s} + \boldsymbol{t} = ((\boldsymbol{r}^*)\boldsymbol{s}) + \boldsymbol{t}$
- Omit parenthesis by associativity of each of these operations.
  Example: $\boldsymbol{rst} = (\boldsymbol{rs})\boldsymbol{t} = \boldsymbol{r}(\boldsymbol{st})$,
  $\boldsymbol{r} + \boldsymbol{s} + \boldsymbol{t} = \boldsymbol{r} + (\boldsymbol{s} + \boldsymbol{t}) = (\boldsymbol{r} + \boldsymbol{s}) + \boldsymbol{t}$.
- Superscript $+$. For convenience, define $\mathrm{r}^+ = \mathrm{rr}^*$. Hence if $\boldsymbol{L}(\mathrm{r}) = \boldsymbol{R}$ then $\boldsymbol{L}(\mathrm{r}^+) = \boldsymbol{R}^+$.

Notation and Parenthesis

- For a regular expression r, $\boldsymbol{L}(\mathrm{r})$ 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 $\boldsymbol{L}(\mathrm{r}_1) = \boldsymbol{L}(\mathrm{r}_2)$.
- Omit parenthesis by adopting precedence order: $*$, concat, $+$.
  Example: $\boldsymbol{r}^*\boldsymbol{s} + \boldsymbol{t} = ((\boldsymbol{r}^*)\boldsymbol{s}) + \boldsymbol{t}$
- Omit parenthesis by associativity of each of these operations.
  Example: $\boldsymbol{rst} = (\boldsymbol{rs})\boldsymbol{t} = \boldsymbol{r}(\boldsymbol{st})$,
  $\boldsymbol{r} + \boldsymbol{s} + \boldsymbol{t} = \boldsymbol{r} + (\boldsymbol{s} + \boldsymbol{t}) = (\boldsymbol{r} + \boldsymbol{s}) + \boldsymbol{t}$.
- Superscript $+$. For convenience, define $\mathrm{r}^+ = \mathrm{rr}^*$. Hence if $\boldsymbol{L}(\mathrm{r}) = \boldsymbol{R}$ then $\boldsymbol{L}(\mathrm{r}^+) = \boldsymbol{R}^+$.
- Other notation: $\boldsymbol{r} + \boldsymbol{s}$, $\boldsymbol{r} \cup \boldsymbol{s}$, $\boldsymbol{r}|\boldsymbol{s}$ all denote union. $\boldsymbol{rs}$ is sometimes written as $\boldsymbol{r} \bullet \boldsymbol{s}$.

Skills

- 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" $L(\mathrm{r})$ (say by giving an English description)

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots, \}$

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots, \}$
- $(0+1)^*$: set of all strings over $\{0, 1\}$

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots,\}$
- $(0+1)^*$: set of all strings over $\{0, 1\}$
- $(0+1)^*001(0+1)^*$:

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots,\}$
- $(0+1)^*$: set of all strings over $\{0, 1\}$
- $(0+1)^*001(0+1)^*$: strings with 001 as substring

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots, \}$
- $(0+1)^*$: 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^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots, \}$
- $(0+1)^*$: 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^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots,\}$
- $(0+1)^*$: 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
- $\emptyset 0$:

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots,\}$
- $(0+1)^*$: 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
- $\emptyset 0$: $\{\}$

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots, \}$
- $(0+1)^*$: 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
- $\emptyset 0$: $\{\}$
- $(\epsilon + 1)(01)^*(\epsilon + 0)$:

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots, \}$
- $(0+1)^*$: 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
- $\emptyset 0$: $\{\}$
- $(\epsilon + 1)(01)^*(\epsilon + 0)$: alteranting 0s and 1s. Alternatively, strings with no two consecutive 0s and no two conescutive 1s

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots,\}$
- $(0+1)^*$: 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
- $\emptyset 0$: $\{\}$
- $(\epsilon + 1)(01)^*(\epsilon + 0)$: alteranting 0s and 1s. Alternatively, strings with no two consecutive 0s and no two conescutive 1s
- $(\epsilon + 0)(1 + 10)^*$:

Understanding regular expressions

- 0^*: set of all strings over $\{0\}$, $\{\epsilon, 0, 00, 000, \ldots,\}$
- $(0+1)^*$: 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
- $\emptyset 0$: $\{\}$
- $(\epsilon + 1)(01)^*(\epsilon + 0)$: alteranting 0s and 1s. Alternatively, strings with no two consecutive 0s and no two conescutive 1s
- $(\epsilon + 0)(1 + 10)^*$: strings without two consecutive 0s.

Creating regular expressions

- bitstrings with 001 or 100 occurring as a substring

Creating regular expressions

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

Creating regular expressions

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

Creating regular expressions

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

Creating regular expressions

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

Creating regular expressions

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

Creating regular expressions

- bitstrings with 001 or 100 occurring as a substring
  one answer: $(0+1)^*001(0+1)^* + (0+1)^*100(0+1)^*$
- bitstrings with 001 as a subsequence
  one answer: $(0+1)^*0(0+1)^*0(0+1)^*1(0+1)^*$
- 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 001 or 100 occurring as a substring
  one answer: $(0+1)^*001(0+1)^* + (0+1)^*100(0+1)^*$
- bitstrings with 001 as a subsequence
  one answer: $(0+1)^*0(0+1)^*0(0+1)^*1(0+1)^*$
- bitstrings with an even number of 1's
  one answer: $0^* + (0^*10^*10^*)^*$
- bitstrings with an odd number of 1's
  one answer: $0^*1\boldsymbol{r}$ where $\boldsymbol{r}$ is solution to previous part

Creating regular expressions

- bitstrings with 001 or 100 occurring as a substring
  one answer: $(0+1)^*001(0+1)^* + (0+1)^*100(0+1)^*$
- bitstrings with 001 as a subsequence
  one answer: $(0+1)^*0(0+1)^*0(0+1)^*1(0+1)^*$
- bitstrings with an even number of 1's
  one answer: $0^* + (0^*10^*10^*)^*$
- bitstrings with an odd number of 1's
  one answer: $0^*1\boldsymbol{r}$ where $\boldsymbol{r}$ is solution to previous part
- bitstrings that do *not* contain 011 as a substring

Creating regular expressions

- bitstrings with 001 or 100 occurring as a substring
  one answer: $(0+1)^*001(0+1)^* + (0+1)^*100(0+1)^*$
- bitstrings with 001 as a subsequence
  one answer: $(0+1)^*0(0+1)^*0(0+1)^*1(0+1)^*$
- bitstrings with an even number of 1's
  one answer: $0^* + (0^*10^*10^*)^*$
- bitstrings with an odd number of 1's
  one answer: $0^*1\boldsymbol{r}$ where $\boldsymbol{r}$ is solution to previous part
- bitstrings that do *not* contain 011 as a substring
  one answer: $1^*0^*(10^+)^*(1+\epsilon)$

Creating regular expressions

- bitstrings with 001 or 100 occurring as a substring
  one answer: $(0+1)^*001(0+1)^* + (0+1)^*100(0+1)^*$
- bitstrings with 001 as a subsequence
  one answer: $(0+1)^*0(0+1)^*0(0+1)^*1(0+1)^*$
- bitstrings with an even number of 1's
  one answer: $0^* + (0^*10^*10^*)^*$
- bitstrings with an odd number of 1's
  one answer: $0^*1\boldsymbol{r}$ where $\boldsymbol{r}$ is solution to previous part
- bitstrings that do *not* contain 011 as a substring
  one answer: $1^*0^*(10^+)^*(1+\epsilon)$
- Hard: bitstrings with an odd number of 1s *and* an odd number of 0s.

Creating regular expressions

- bitstrings with 001 or 100 occurring as a substring
  one answer: $(0+1)^*001(0+1)^* + (0+1)^*100(0+1)^*$
- bitstrings with 001 as a subsequence
  one answer: $(0+1)^*0(0+1)^*0(0+1)^*1(0+1)^*$
- bitstrings with an even number of 1's
  one answer: $0^* + (0^*10^*10^*)^*$
- bitstrings with an odd number of 1's
  one answer: $0^*1\boldsymbol{r}$ where $\boldsymbol{r}$ is solution to previous part
- bitstrings that do *not* contain 011 as a substring
  one answer: $1^*0^*(10^+)^*(1+\epsilon)$
- Hard: bitstrings with an odd number of 1s *and* an odd number of 0s.
- Hard: English strings with all occurrences of "CS173" as a substring are before any occurence of "CS374" as a substring

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 on 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 on 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 on 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 $L = \{0^n 1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\}$.

A non-regular language and other closure properties

Consider $L = \{0^n 1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\}$.

Theorem

L is **not** *a regular language.*

A non-regular language and other closure properties

Consider $L = \{0^n 1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\}$.

Theorem

L is **not** *a regular language.*

How do we prove it?

A non-regular language and other closure properties

Consider $L = \{0^n 1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\}$.

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?

Summary and Skills

Regular languages and expressions defined inductively via simple base cases and three operations: union, concatenation, Kleene star

Skills:

- Given a laguage L described in English, design a regular expression r such that $L = L(r)$
- Given a regular expression r, give an English description of the language $L(r)$

Later:

- see equivalence with DFAs, NFAs
- technique to prove that languages are not regular