DECISION PROBLEMS

What are decision problems

- A ***decision problem,*** or ***problem*** for short, consists of

  1. A set I of finite and discrete objects,
     called the problem's ***instances***; and

What are decision problems

- A ***decision problem,*** or ***problem*** for short, consists of

  1. A set I of finite and discrete objects,
     called the problem's ***instances***; and
  2. A ***property*** P that instances may satisfy or not.

What are decision problems

- A ***decision problem,*** or ***problem*** for short, consists of
  1. A set I of finite and discrete objects, called the problem's ***instances***; and
  2. A ***property*** P that instances may satisfy or not.
- A problem's ***algorithmic solution,*** or ***solution*** for short, is an algorithm that, given an instance w as input outputs the answer as to whether or not w satisfies the property P.

What are decision problems

- A **decision problem,** or ***problem*** for short, consists of

  1. A set I of finite and discrete objects, called the problem's **instances**; and
  2. A **property** P that instances may satisfy or not.

- A problem's **algorithmic solution,** or ***solution*** for short, is an algorithm that, given an instance w as input outputs the answer as to whether or not w satisfies the property P.

- A problem is **decidable** if it has an algorithmic solution, and **undecidable** otherwise.

Examples of decision problems

1. ***Primality.***

   I is the set of positive integers,

   P the property of being a prime number.

   A solution to the problem is an algorithm deciding primality.

Examples of decision problems

1. ***Primality.***

   I is the set of positive integers,

   P the property of being a prime number.

   A solution to the problem is an algorithm deciding primality.

2. ***Graph connectivity.***

   I is the set of finite undirected graphs,

   P is the property of being connected.

Examples of decision problems

1. ***Primality.***

   I is the set of positive integers,

   P the property of being a prime number.

   A solution to the problem is an algorithm deciding primality.

2. ***Graph connectivity.***

   I is the set of finite undirected graphs,

   P is the property of being connected.

3. ***Substring.***

   I consists of pairs (x, w) of binary strings.

   P is "x is a substring of w."

Examples of decision problems

1. ***Primality.***
   I is the set of positive integers,
   P the property of being a prime number.

   A solution to the problem is an algorithm deciding primality.

2. ***Graph connectivity.***
   I is the set of finite undirected graphs,
   P is the property of being connected.

3. ***Substring.***
   I consists of pairs (x, w) of binary strings.
   P is "x is a substring of w."

4. ***Termination.*** I consists of the programs p (in Python say).
   P is "p terminates for all legal inputs."

A mystery problem: Integer Partition

- Primality, Connectivity and Substring have each an efficient solution.
  Termination does not have any solution.
  Here is a problem for which there is a solution
  but we don't know if there is an efficient one.

A mystery problem: Integer Partition

- Primality, Connectivity and Substring have each an efficient solution.
  Termination does not have any solution.
  Here is a problem for which there is a solution
  but we don't know if there is an efficient one.
- Given a finite set S of positive integers,
  is there a set $P \subseteq S$ that adds up to half of ΣS.
- That is,
  - Instances: Finite sets S of positive integers
  - Property: Exists $P \subseteq S$ such that $\Sigma P = \Sigma(S - P)$

A mystery problem: Integer Partition

- Primality, Connectivity and Substring have each an efficient solution.
  Termination does not have any solution.
  Here is a problem for which there is a solution
  but we don't know if there is an efficient one.
- Given a finite set S of positive integers,
  is there a set $P \subseteq S$ that adds up to half of ΣS.
- That is,
  - Instances: Finite sets S of positive integers
  - Property: Exists $P \subseteq S$ such that $\Sigma P = \Sigma(S - P)$
- Instances are implicitly assumed to be given textually,
  e.g. list of binary numerals with a separator:
  $\{2, 4, 5\}$ given as $\mathbf{10\#100\#101}$.

A mystery problem: Integer Partition

- Primality, Connectivity and Substring have each an efficient solution.
  Termination does not have any solution.
  Here is a problem for which there is a solution
  but we don't know if there is an efficient one.
- Given a finite set S of positive integers,
  is there a set $P \subseteq S$ that adds up to half of ΣS.
- That is,
  - Instances: Finite sets S of positive integers
  - Property: Exists $P \subseteq S$ such that $\Sigma P = \Sigma(S - P)$
- Instances are implicitly assumed to be given textually,
  e.g. list of binary numerals with a separator:
  $\{2, 4, 5\}$ given as $\mathbf{10\#100\#101}$.
- Examples.
  - ⋆ For $\{2, 3, 4, 5\}$: *yes.*
  - ⋆ For $\{2, 3, 4, 6\}$: *no*, since the total is odd.
  - ⋆ For $\{2, 3, 4, 7\}$: *no*

A mystery problem: Integer Partition

- Primality, Connectivity and Substring have each an efficient solution.
  Termination does not have any solution.
  Here is a problem for which there is a solution
  but we don't know if there is an efficient one.
- Given a finite set S of positive integers,
  is there a set $P \subseteq S$ that adds up to half of ΣS.
- That is,
  - Instances: Finite sets S of positive integers
  - Property: Exists $P \subseteq S$ such that $\Sigma P = \Sigma(S - P)$
- Instances are implicitly assumed to be given textually,
  e.g. list of binary numerals with a separator:
  $\{2, 4, 5\}$ given as $\mathbf{10\#100\#101}$.
- Examples.
  - ⋆ For $\{2, 3, 4, 5\}$: *yes.*
  - ⋆ For $\{2, 3, 4, 6\}$: *no*, since the total is odd.
  - ⋆ For $\{2, 3, 4, 7\}$: *no*
- What algorithm is a solution?

A mystery problem: Integer Partition

- Primality, Connectivity and Substring have each an efficient solution.
  Termination does not have any solution.
  Here is a problem for which there is a solution
  but we don't know if there is an efficient one.
- Given a finite set S of positive integers,
  is there a set $P \subseteq S$ that adds up to half of ΣS.
- That is,
  - Instances: Finite sets S of positive integers
  - Property: Exists $P \subseteq S$ such that $\Sigma P = \Sigma(S - P)$
- Instances are implicitly assumed to be given textually,
  e.g. list of binary numerals with a separator:
  $\{2, 4, 5\}$ given as $\mathbf{10\#100\#101}$.
- Examples.
  - ⋆ For $\{2, 3, 4, 5\}$: *yes.*
  - ⋆ For $\{2, 3, 4, 6\}$: *no*, since the total is odd.
  - ⋆ For $\{2, 3, 4, 7\}$: *no*
- What algorithm is a solution?
  What's wrong with it?

LANGUAGES

Language = set of strings

- Given an alphabet Σ a ***Σ-language*** (or ***language over*** Σ) is a set of Σ-strings.
- Don't confuse strings and languages:
  Languages are sets, and can be finite or infinite.
  Strings are data-objects, and are finite by defnition.

Examples

- The set of all strings over Σ, denoted Σ^*.

Examples

- The set of all strings over Σ, denoted Σ^*.
- $\{\mathtt{a}, \mathtt{b}, \mathtt{ab}, \mathtt{bb}\}$

Examples

- The set of all strings over Σ, denoted Σ^*.
- $\{\mathtt{a}, \mathtt{b}, \mathtt{ab}, \mathtt{bb}\}$
- The empty language.

Examples

- The set of all strings over Σ, denoted Σ^*.
- $\{\mathtt{a}, \mathtt{b}, \mathtt{ab}, \mathtt{bb}\}$
- The empty language.
- The alphabet Σ itself (each string is a single symbol).

Examples

- The set of all strings over Σ, denoted Σ^*.
- $\{$a, b, ab, bb$\}$
- The empty language.
- The alphabet Σ itself (each string is a single symbol).
- $\{$a, aa, aaa, aaaa,...$\}$

Examples

- The set of all strings over Σ, denoted Σ^*.
- $\{\mathtt{a}, \mathtt{b}, \mathtt{ab}, \mathtt{bb}\}$
- The empty language.
- The alphabet Σ itself (each string is a single symbol).
- $\{\mathtt{a}, \mathtt{aa}, \mathtt{aaa}, \mathtt{aaaa}, \ldots\}$
- The decimal numerals for prime numbers:
  $\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29 \ \ldots\ \}$.

Examples

- The set of all strings over Σ, denoted Σ^*.
- $\{a, b, ab, bb\}$
- The empty language.
- The alphabet Σ itself (each string is a single symbol).
- $\{a, aa, aaa, aaaa, ...\}$
- The decimal numerals for prime numbers: $\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29 \ \ldots\}$.
- The binary numerals for prime numbers:
$\{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11101, \ldots\}$.

More examples

- The single-sylable English words

More examples

- The single-sylable English words
- The passwords used at IUB

More examples

- The single-sylable English words
- The passwords used at IUB
- The books in the Library of Congress (each book is a string!)

More examples

- The single-sylable English words
- The passwords used at IUB
- The books in the Library of Congress (each book is a string!)
- The syntactically correct JavaScript programs

More examples

- The single-sylable English words
- The passwords used at IUB
- The books in the Library of Congress (each book is a string!)
- The syntactically correct JavaScript programs
- The Python programs that terminate for all input

TEXTUAL DECISION PROBLEMS

Each language is a decision problem

- Each language $L \subseteq \Sigma^*$ is a decision problem:
  - Instances: $w \in \Sigma^*$
  - Property: $w \in L$

Conversely: Each decision problem is a language

- The instances of a decision problem P are finite and discrete, so they are codable as text.

Conversely: Each decision problem is a language

- The instances of a decision problem P are finite and discrete, so they are codable as text.
- The codes of the instances satisfying P form a language!

Conversely: Each decision problem is a language

- The instances of a decision problem P are finite and discrete, so they are codable as text.
- The codes of the instances satisfying P form a language!
- Example: Using binary numerals the Primality problem becomes the language $\mathtt{10, 11, 101, 111, 1011, \ldots}$.

Conversely: Each decision problem is a language

- The instances of a decision problem P are finite and discrete,
  so they are codable as text.
- The codes of the instances satisfying P form a language!
- Example: Using binary numerals the Primality problem
  becomes the language $\mathtt{10}, \mathtt{11}, \mathtt{101}, \mathtt{111}, \mathtt{1011}, \ldots$.
- Once we set a coding method,
  we write $a^{\#}$ for the code of a
  (at least when distinguishing between a and $a^{\#}$ matters).

DELINEATING LANGUAGES

Operations on languages

- If L and M are languages then
  we obtain new languages by basic set operations, such as
  union ($L \cup M$), intersection ($L \cap M$), and difference ($L - M$).
- Those operations work for any sets.
  We consider next operations that are specific to languages.

String operations applied pointwise

- Let $rev : \Sigma^* \to \Sigma^*$ be the function that reverses its input.
  Example: $rev(\texttt{ab123}) = \texttt{321ba}$.

String operations applied pointwise

- Let $rev : \Sigma^* \to \Sigma^*$ be the function that reverses its input.
  Example: $rev(\texttt{ab123}) = \texttt{321ba}$.
- Applying rev to each string in a language L
  we obtain a new language $\{\, rev(w) \mid w \in L \,\}$.

String operations applied pointwise

- Let $rev : \Sigma^* \to \Sigma^*$ be the function that reverses its input.
  Example: $rev(\texttt{ab123}) = \texttt{321ba}$.

- Applying rev to each string in a language L
  we obtain a new language $\{ rev(w) \mid w \in L \}$.

  From function rev over strings
  we obtain a function over languages:

  $$\widehat{rev}(L) = \{ rev(w) \mid w \in L \}$$

String operations applied pointwise

- Let $rev : \Sigma^* \to \Sigma^*$ be the function that reverses its input.
  Example: $rev(\texttt{ab123}) = \texttt{321ba}$.
- Applying rev to each string in a language L
  we obtain a new language $\{ rev(w) \mid w \in L \}$.

  From function rev over strings
  we obtain a function over languages:

  $$\widehat{rev}(L) = \{ rev(w) \mid w \in L \}$$
- Generally, a function $f : \Sigma^* \to \Sigma^*$
  induces a function $\hat{f}$ on Σ-languages:

  $$\hat{f}(L) = \{f(w) \mid w \in L\}$$

Language concatenation

- From concatenation on strings
  we get a concatenation operation on languages:
  The **concatenation** of languages L, M is

$$L \cdot M \; =_{\mathrm{df}} \; \{u \cdot v \mid u \in L,\; v \in M\}$$

Language concatenation

- From concatenation on strings
  we get a concatenation operation on languages:
  The **concatenation** of languages L, M is

$$L \cdot M =_{\text{df}} \{u \cdot v \mid u \in L,\ v \in M\}$$

- Examples:
  - $\{\mathtt{a}, \mathtt{b}\} \cdot \{\mathtt{b}, \mathtt{c}\} = \{\mathtt{ab}, \mathtt{ac}, \mathtt{bb}, \mathtt{bc}\}$

Language concatenation

- From concatenation on strings
  we get a concatenation operation on languages:
  The **concatenation** of languages L, M is

$$L \cdot M =_{\text{df}} \{u \cdot v \mid u \in L,\ v \in M\}$$

- Examples:
  - $\{\mathtt{a}, \mathtt{b}\} \cdot \{\mathtt{b}, \mathtt{c}\} = \{\mathtt{ab}, \mathtt{ac}, \mathtt{bb}, \mathtt{bc}\}$
  - $\{\mathtt{1}, \mathtt{11}\} \cdot \{\mathtt{1}, \mathtt{11}\} = \{\mathtt{11}, \mathtt{111}, \mathtt{1111}\}$

Language concatenation

- From concatenation on strings
  we get a concatenation operation on languages:
  The **concatenation** of languages L, M is

$$L \cdot M \ =_{\text{df}} \ \{u \cdot v \mid u \in L,\ v \in M\}$$

- Examples:
  - $\{\mathtt{a}, \mathtt{b}\} \cdot \{\mathtt{b}, \mathtt{c}\} = \{\mathtt{ab}, \mathtt{ac}, \mathtt{bb}, \mathtt{bc}\}$
  - $\{\mathtt{1}, \mathtt{11}\} \cdot \{\mathtt{1}, \mathtt{11}\} = \{\mathtt{11}, \mathtt{111}, \mathtt{1111}\}$
  - $\{\mathtt{play},\ \mathtt{dress}\} \cdot \{\mathtt{er},\ \mathtt{ing}\}$
    $= \{\mathtt{player}, \mathtt{playing}, \mathtt{dresser}, \mathtt{dressing}\}$

Language concatenation

- From concatenation on strings
  we get a concatenation operation on languages:
  The ***concatenation*** of languages L, M is

$$L \cdot M =_{\mathrm{df}} \{u \cdot v \mid u \in L,\ v \in M\}$$

- Examples:
  - $\{\mathtt{a}, \mathtt{b}\} \cdot \{\mathtt{b}, \mathtt{c}\} = \{\mathtt{ab}, \mathtt{ac}, \mathtt{bb}, \mathtt{bc}\}$
  - $\{\mathtt{1}, \mathtt{11}\} \cdot \{\mathtt{1}, \mathtt{11}\} = \{\mathtt{11}, \mathtt{111}, \mathtt{1111}\}$
  - $\{\mathtt{play}, \mathtt{dress}\} \cdot \{\mathtt{er}, \mathtt{ing}\}$
    $= \{\mathtt{player}, \mathtt{playing}, \mathtt{dresser}, \mathtt{dressing}\}$
  - $\Sigma^* \cdot \Sigma^* = \Sigma^*$

Puzzles

- $\{A, B, C\} \cdot \{1, 2\} =$

Puzzles

- $\{\mathtt{A}, \mathtt{B}, \mathtt{C}\} \cdot \{1, 2\} = \{\mathtt{A1}, \mathtt{A2}, \mathtt{B1}, \mathtt{B2}, \mathtt{C1}, \mathtt{C2}\}$

Puzzles

- $\{\texttt{A}, \texttt{B}, \texttt{C}\} \cdot \{\texttt{1}, \texttt{2}\} = \{\texttt{A1}, \texttt{A2}, \texttt{B1}, \texttt{B2}, \texttt{C1}, \texttt{C2}\}$
- $L \cdot \emptyset =$

Puzzles

- $\{\mathtt{A}, \mathtt{B}, \mathtt{C}\} \cdot \{\mathtt{1}, \mathtt{2}\} = \{\mathtt{A1}, \mathtt{A2}, \mathtt{B1}, \mathtt{B2}, \mathtt{C1}, \mathtt{C2}\}$
- $L \cdot \emptyset =$

  $\emptyset$

Puzzles

- $\{\texttt{A}, \texttt{B}, \texttt{C}\} \cdot \{\texttt{1}, \texttt{2}\} = \{\texttt{A1}, \texttt{A2}, \texttt{B1}, \texttt{B2}, \texttt{C1}, \texttt{C2}\}$
- $L \cdot \emptyset =$

  $\emptyset$
- $L \cdot \{\varepsilon\} =$

Puzzles

- $\{\mathtt{A}, \mathtt{B}, \mathtt{C}\} \cdot \{\mathtt{1}, \mathtt{2}\} = \{\mathtt{A1}, \mathtt{A2}, \mathtt{B1}, \mathtt{B2}, \mathtt{C1}, \mathtt{C2}\}$
- $L \cdot \emptyset =$

  $\emptyset$
- $L \cdot \{\varepsilon\} =$

  L

  So $\{\varepsilon\}$ is the unit of language concatenation,
  just as $0, 1$ and ε are the units
  of addition, multiplication, and string concatenation.

Puzzles

- $\{\mathtt{A},\mathtt{B},\mathtt{C}\} \cdot \{\mathtt{1},\mathtt{2}\} = \{\mathtt{A1},\mathtt{A2},\mathtt{B1},\mathtt{B2},\mathtt{C1},\mathtt{C2}\}$
- $L \cdot \emptyset =$

  $\emptyset$
- $L \cdot \{\varepsilon\} =$

  L

  So $\{\varepsilon\}$ is the unit of language concatenation,
  just as $0, 1$ and ε are the units
  of addition, multiplication, and string concatenation.
- $\Sigma \cdot \Sigma =$

Puzzles

- $\{\mathtt{A},\mathtt{B},\mathtt{C}\} \cdot \{\mathtt{1},\mathtt{2}\} = \{\mathtt{A1},\mathtt{A2},\mathtt{B1},\mathtt{B2},\mathtt{C1},\mathtt{C2}\}$
- $L \cdot \emptyset =$

  $\emptyset$
- $L \cdot \{\varepsilon\} =$

  L

  So $\{\varepsilon\}$ is the unit of language concatenation,
  just as $0, 1$ and ε are the units
  of addition, multiplication, and string concatenation.
- $\Sigma \cdot \Sigma =$

  The Σ-strings of length 2.

- Suppose L has p strings and K has q.
  What is the maximal number of strings in $L \cdot K$?

- Suppose L has p strings and K has q.
  What is the maximal number of strings in $L \cdot K$? $p \times q$

- Suppose L has p strings and K has q.
  What is the maximal number of strings in $L \cdot K$? $p \times q$
- Is it possible that $L \cdot K$ have fewer
  than $p \times q$ strings?

- Suppose L has p strings and K has q.
  What is the maximal number of strings in $L \cdot K$? $p \times q$

- Is it possible that $L \cdot K$ have fewer than $p \times q$ strings?

  $\{1, 11\} \cdot \{1, 11\} = \{1, 11, 111\}$

Associativity

$$\begin{aligned} L \cdot (K \cdot M) &= \{x \cdot (y \cdot z) \mid x \in L,\ y \in K,\ z \in M\} \\ &= \{(x \cdot y) \cdot z \mid x \in L,\ y \in K,\ z \in M\} \\ &= (L \cdot K) \cdot M \end{aligned}$$

Self-concatenation

- Notation: L^2 for $L \cdot L$.

Self-concatenation

- Notation: $\boldsymbol{L^2}$ for $\boldsymbol{L \cdot L}$.
- ***Can*** L^2 ***be the same as*** L***?***

Self-concatenation

- Notation: L^2 for $L \cdot L$.
- ***Can L^2 be the same as L?***

  Yes: E.g. the language E of strings of even length:

  $E \subseteq E \cdot E$ because $\varepsilon \in E$

  $E \cdot E \subseteq$ because the sum of even numbers is even.

Self-concatenation

- Notation: L^2 for $L \cdot L$.
- ***Can L^2 be the same as L?***

  Yes: E.g. the language E of strings of even length:

  $E \subseteq E \cdot E$ because $\varepsilon \in E$

  $E \cdot E \subseteq$ because the sum of even numbers is even.

  Also: $\emptyset$, $\{\varepsilon\}$

Repeated concatenation

- Define, for $n \geqslant 1$,

  $L^n \quad = \quad L \cdot \cdots \cdot L$ (L repeated n times)

Repeated concatenation

- Define, for $n \geqslant 1$,

  $L^n \quad = \quad L \cdot \dots \cdot L$ (L repeated n times)

- We have $L^n \cdot L^k = L^{n+k}$ for $n, k > 1$.

  To make this true for $k = 0$: $L^n \cdot L^0 = L^{n+0}$

  define L^0 to be

Repeated concatenation

- Define, for $n \geqslant 1$,

  $L^n \quad = \quad L \cdot \cdots \cdot L$ (L repeated n times)

- We have $L^n \cdot L^k = L^{n+k}$ for $n, k > 1$.

  To make this true for $k = 0$: $L^n \cdot L^0 = L^{n+0}$

  define L^0 to be $\{\varepsilon\}$,

  the neutral language for concatenation

Iterated concatenation

- We've generated the set Σ^* of all Σ-strings:
  - $\varepsilon \in \Sigma^*$
  - If $\sigma \in \Sigma$ and $w \in \Sigma^*$ then $\sigma w \in \Sigma^*$.

Iterated concatenation

- We've generated the set Σ^* of all Σ-strings:
  - $\varepsilon \in \Sigma^*$
  - If $\sigma \in \Sigma$ and $w \in \Sigma^*$ then $\sigma w \in \Sigma^*$.
- Generalizing this from Σ
  to a generic language L we get:
  - $\varepsilon \in L^*$
  - If $x \in L$ and $w \in L^*$ then $x \cdot w \in L^*$.

Iterated concatenation

- We've generated the set Σ^* of all Σ-strings:
  - $\varepsilon \in \Sigma^*$
  - If $\sigma \in \Sigma$ and $w \in \Sigma^*$ then $\sigma w \in \Sigma^*$.
- Generalizing this from Σ
  to a generic language L we get:
  - $\varepsilon \in L^*$
  - If $x \in L$ and $w \in L^*$ then $x \cdot w \in L^*$.
- So

$$\begin{aligned} L^* &= \{w_1 \cdot \cdots \cdot w_k \mid k \geqslant 0,\ w_i \in L\} \\ &= \cup_{k \geqslant 0}\, L^k \end{aligned}$$

Iterated concatenation

- We've generated the set Σ^* of all Σ-strings:
  - $\varepsilon \in \Sigma^*$
  - If $\sigma \in \Sigma$ and $w \in \Sigma^*$ then $\sigma w \in \Sigma^*$.
- Generalizing this from Σ
  to a generic language L we get:
  - $\varepsilon \in L^*$
  - If $x \in L$ and $w \in L^*$ then $x \cdot w \in L^*$.
- So

$$L^* = \{w_1 \cdot \cdots \cdot w_k \mid k \geqslant 0,\ w_i \in L\}$$
$$= \cup_{k \geqslant 0} L^k$$

- This is the ***(Kleene) star*** of L.
  $x \in L^*$ iff $x = \varepsilon$ or $x = w_1 \cdot w_2 \cdots w_n$ for some $w_i \in L$.

Some properties of star

- L^* is the smallest language containing L and ε and closed under concatenation.

Some properties of star

- L^* is the smallest language containing L and ε and closed under concatenation.
- $L^* \cdot L^* \subseteq L^*$:
  If $u = x_1 \cdots x_p$ and $v = y_1 \cdots y_m$, where $x_i, y_j \in L$,
  then $u \cdot v = x_1 \cdots x_p \cdot y_1 \cdots y_m \in L^*$
  (concatenation is associative!)

Some properties of star

- L^* is the smallest language containing L and ε and closed under concatenation.
- $L^* \cdot L^* \subseteq L^*$:
  If $u = x_1 \cdots x_p$ and $v = y_1 \cdots y_m$, where $x_i, y_j \in L$,
  then $u \cdot v = x_1 \cdots x_p \cdot y_1 \cdots y_m \in L^*$
  (concatenation is associative!)
- A proof by induction on $x \in L^*$ that for all $y \in L^*$ we have $x \cdot y \in L^*$:

Some properties of star

- L^* is the smallest language containing L and ε and closed under concatenation.
- $L^* \cdot L^* \subseteq L^*$:

  If $u = x_1 \cdots x_p$ and $v = y_1 \cdots y_m$, where $x_i, y_j \in L$,
  then $u \cdot v = x_1 \cdots x_p \cdot y_1 \cdots y_m \in L^*$
  (concatenation is associative!)
- A proof by induction on $x \in L^*$ that for all $y \in L^*$ we have $x \cdot y \in L^*$:
  - Basis: If $y \in L^*$ then $x \cdot y = \varepsilon \cdot y = y \in L^*$.

Some properties of star

- L^* is the smallest language containing L and ε and closed under concatenation.
- $L^* \cdot L^* \subseteq L^*$:
  If $u = x_1 \cdots x_p$ and $v = y_1 \cdots y_m$, where $x_i, y_j \in L$,
  then $u \cdot v = x_1 \cdots x_p \cdot y_1 \cdots y_m \in L^*$
  (concatenation is associative!)
- A proof by induction on $x \in L^*$ that for all $y \in L^*$ we have $x \cdot y \in L^*$:
  - Basis: If $y \in L^*$ then $x \cdot y = \varepsilon \cdot y = y \in L^*$.
  - Step: Assume $x \cdot y \in L^*$ for all $y \in L^*$.
    Then for $v \in L$ $(v \cdot x) \cdot y = v \cdot (x \cdot y)$ which is in L^* by definitio
    L^*.

Some examples

- $\{0\}^* =$

Some examples

- $\{0\}^* = \{\varepsilon, 0, 00, 000, \ldots, 0^n, \ldots\}$

Some examples

- $\{0\}^* = \{\varepsilon, 0, 00, 000, \ldots, 0^n, \ldots\}$
- $\{\varepsilon\}^* =$

Some examples

- $\{0\}^* = \{\varepsilon, 0, 00, 000, \ldots, 0^n, \ldots\}$
- $\{\varepsilon\}^* = \{\varepsilon\}$

Some examples

- $\{0\}^* = \{\varepsilon, 0, 00, 000, \ldots, 0^n, \ldots\}$
- $\{\varepsilon\}^* = \{\varepsilon\}$
- $\emptyset^* =$

Some examples

- $\{0\}^* = \{\varepsilon, 0, 00, 000, \ldots, 0^n, \ldots\}$
- $\{\varepsilon\}^* = \{\varepsilon\}$
- $\emptyset^* = \{\varepsilon\}$

Some examples

- $\{0\}^* = \{\varepsilon, 0, 00, 000, \ldots, 0^n, \ldots\}$
- $\{\varepsilon\}^* = \{\varepsilon\}$
- $\emptyset^* = \{\varepsilon\}$
- $\{00\}^* =$

Some examples

- $\{0\}^* = \{\varepsilon, 0, 00, 000, \ldots, 0^n, \ldots\}$
- $\{\varepsilon\}^* = \{\varepsilon\}$
- $\emptyset^* = \{\varepsilon\}$
- $\{00\}^* = \{0^{2n} \mid n \geqslant 0\}$

LANGUAGE RESIDUES

Acceptable complements

- Given $L \subseteq \Sigma^*$ and $w \in \Sigma^*$
  consider the strings in L of the form $w \cdot x$,
  i.e. that start with w .
- For example if L consists of English words
  and w is **`con`** then we look at words
  that start with **`con`**, such as
  `consider`, **`contrary`**, **`condense`**, and **`con`** itself.

Acceptable complements

- Given $L \subseteq \Sigma^*$ and $w \in \Sigma^*$
  consider the strings in L of the form $w \cdot x$,
  i.e. that start with w.
- For example if L consists of English words
  and w is **`con`** then we look at words
  that start with **`con`**, such as
  `consider`, **`contrary`**, **`condense`**, and **`con`** itself.
- Here are the remainders (the x in $w \cdot x$)) of the above:
  `sider`, **`rary`**, **`dense`**, and ε.
  I.e. the strings that complement **`con`** to an English word.

Acceptable complements

- Given $L \subseteq \Sigma^*$ and $w \in \Sigma^*$
  consider the strings in L of the form $w \cdot x$,
  i.e. that start with w .
- For example if L consists of English words
  and w is **`con`** then we look at words
  that start with **`con`**, such as
  `consider`, **`contrary`**, **`condense`**, and **`con`** itself.
- Here are the remainders (the x in $w \cdot x$)) of the above:
  `sider`, **`rary`**, **`dense`**, and ε.
  I.e. the strings that complement **`con`** to an English word.
- The resulting language is the ***residue*** of the English language over **`co`**
  `con` itself is the ***trunk*** of that residue.

Acceptable complements

- Given $L \subseteq \Sigma^*$ and $w \in \Sigma^*$
  consider the strings in L of the form $w \cdot x$,
  i.e. that start with w .
- For example if L consists of English words
  and w is **`con`** then we look at words
  that start with **`con`**, such as
  `consider`, **`contrary`**, **`condense`**, and **`con`** itself.
- Here are the remainders (the x in $w \cdot x$)) of the above:
  `sider`, **`rary`**, **`dense`**, and ε.
  I.e. the strings that complement **`con`** to an English word.
- The resulting language is the ***residue*** of the English language over **`co`**
  `con` itself is the ***trunk*** of that residue.
- In general, given $L \subseteq \Sigma^*$ and $w \in \Sigma^*$
  the ***residue of*** L ***over*** w is the language $L/w = \{x \mid w \cdot x \in L\}$

Examples of residues

- Take $L =$ English words.

  $L/$**`invent`** contains the strings **`or`**, **`ion`**, **`ive`**, **`ed`** and **`ing`**
  since **`inventor`**, **`invention`**, **`inventive`** and **`invented`** are words.

Examples of residues

- Take $L =$ English words.

  $L/$**`invent`** contains the strings **`or`**, **`ion`**, **`ive`**, **`ed`** and **`ing`**
  since **`inventor`**, **`invention`**, **`inventive`** and **`invented`** are words.

- ϵ is also in $L/$**`invent`** since **`invent`** is a word.

Examples of residues

- Take $L =$ English words.

  $L/\texttt{invent}$ contains the strings $\texttt{or}, \texttt{ion}, \texttt{ive}, \texttt{ed}$ and $\texttt{ing}$
  since $\texttt{inventor}, \texttt{invention}, \texttt{inventive}$ and $\texttt{invented}$ are words.
- ϵ is also in $L/\texttt{invent}$ since $\texttt{invent}$ is a word.
- The residue $L/\texttt{ad}$ contains the strings $\texttt{vance}, \texttt{apt}, \texttt{opt}, \texttt{d},$ and ϵ.

Examples of residues

- Take $L =$ English words.

  $L/\texttt{invent}$ contains the strings $\texttt{or}, \texttt{ion}, \texttt{ive}, \texttt{ed}$ and $\texttt{ing}$
  since $\texttt{inventor}, \texttt{invention}, \texttt{inventive}$ and $\texttt{invented}$ are words.

- ϵ is also in $L/\texttt{invent}$ since $\texttt{invent}$ is a word.

- The residue $L/\texttt{ad}$ contains the strings $\texttt{vance}, \texttt{apt}, \texttt{opt}, \texttt{d},$ and ϵ.

- Take $L = \{\texttt{ab}\}$, a singleton language.
  We have $L/\varepsilon = \{\texttt{ab}\}, L/\texttt{a} = \{\texttt{b}\}$, and $L/\texttt{ab} = \varepsilon$.

  For any other string w , $L/w = \emptyset$.

Examples of residues

- Take $L =$ English words.

  $L/\texttt{invent}$ contains the strings $\texttt{or}, \texttt{ion}, \texttt{ive}, \texttt{ed}$ and $\texttt{ing}$
  since $\texttt{inventor}, \texttt{invention}, \texttt{inventive}$ and $\texttt{invented}$ are words.

- ϵ is also in $L/\texttt{invent}$ since $\texttt{invent}$ is a word.

- The residue $L/\texttt{ad}$ contains the strings $\texttt{vance}, \texttt{apt}, \texttt{opt}, \texttt{d},$ and ϵ.

- Take $L = \{\texttt{ab}\}$, a singleton language.
  We have $L/\varepsilon = \{\texttt{ab}\}, L/\texttt{a} = \{\texttt{b}\}$, and $L/\texttt{ab} = \varepsilon$.

  For any other string w , $L/w = \emptyset$.

- For any language L we have $L/\varepsilon = L$:

  $w \in L$ iff $\varepsilon \in L/w$.

Another example

- $L = \{0,\ 00,\ 010\}$

$$
\begin{aligned}
L/\varepsilon &= L \\
L/0 &= \{\varepsilon,\ 1, 0\} \\
L/00 &= \{\varepsilon\} \\
L/01 &= \{0\} \\
L/010 &= \{\varepsilon\} \\
L/w &= \emptyset \qquad \text{for any other } w
\end{aligned}
$$

- $L/00 = L/010$, so there are five distinct residues.

The regular languages

- The ***basic Σ-languages*** are generated from the finite Σ-languages and Σ^* by the clauses
  - the set operations of union, intersection, and difference; and
  - the language operations of concatenation, plus and star.

The regular languages

- The ***basic Σ-languages*** are generated from the finite Σ-languages and Σ^* by the clauses
  - the set operations of union, intersection, and difference; and
  - the language operations of concatenation, plus and star.
- That is:
  - Σ^* and the Finite languages are basic.
  - If L, M are basic then so are $L \cap M$, $L \cup M$ and $L - M$.
  - If L is basic then so are L^+ and L^*.

Regular languages

- The ***regular*** Σ-languages are generated:
  - The finite Σ-languages are regular.
  - The union, intersection, and difference of regular languages are regular.
  - The concatenation and star of regular languages are regular.

Regular languages

- The ***regular*** Σ-languages are generated:

  - ▸ The finite Σ-languages are regular.
  - ▸ The union, intersection, and difference of regular languages are re lar.
  - ▸ The concatenation and star of regular languages are regular.

- Rephrased:
  - The finite languages are regular.
  - If L, M are regular then so are $L \cap M$, $L \cup M$ and $L - M$.
  - If L is regular then so is L^* .

Strictly-regular languages

- A formally narrower definition:
  - $\emptyset, \{\varepsilon\}$ and $\{\sigma\}$ (for every $\sigma \in \Sigma$) are strictly-regular.
  - If L, M are regular then so is $L \cup M$.
  - If L, M are regular then so are $L \cdot M$ and L^*.
- Every strictly-regular language is regular.
- We shall prove the coverse later.

Regular expressions

- ***Regular expressions (RegExp's)*** over Σ are notations for strictly-reg languages:

  each expression is a ***road-map***, i.e. recipe, notation,
  for the strictly-regular definition of a language.

Regular expressions

- ***Regular expressions (RegExp's)*** over Σ are notations for strictly-reg
  languages:
  each expression is a ***road-map***, i.e. recipe, notation,
  for the strictly-regular definition of a language.
- $\mathcal{L}(\alpha)$ is the language denoted by a RegExp α.

Regular expressions

- ***Regular expressions (RegExp's)*** over Σ are notations for strictly-reg
  languages:
  each expression is a ***road-map***, i.e. recipe, notation,
  for the strictly-regular definition of a language.
- $\mathcal{L}(\alpha)$ is the language denoted by a RegExp α.
- For the initial strictly-regular languages we use the following names:
  - $\emptyset$ is denoted by $\boldsymbol{\emptyset}$,
  - $\{\varepsilon\}$ by ε, and
  - for each $\sigma \in \Sigma$ the singleton language $\{\sigma\}$ is denoted by σ.

Regular expressions

- ***Regular expressions (RegExp's)*** over Σ are notations for strictly-reg
  languages:
  each expression is a ***road-map***, i.e. recipe, notation,
  for the strictly-regular definition of a language.
- $\mathcal{L}(\alpha)$ is the language denoted by a RegExp α.
- For the initial strictly-regular languages we use the following names:
  - $\emptyset$ is denoted by $\boldsymbol{\emptyset}$,
  - $\{\varepsilon\}$ by ε, and
  - for each $\sigma \in \Sigma$ the singleton language $\{\sigma\}$ is denoted by σ.
- Suppose language L is denoted by α and K by β. Then
  - $L \cup K$ is denoted by $(\alpha) \mathbf{U} (\alpha)$,
  - $L \cdot K$ by $(\alpha) \bullet (\beta)$, and
  - L^* by $(\alpha)^\star$.

Examples

- $\{ab\}$ is denoted by $(\mathtt{a}) \bullet (\mathtt{b})$.

Examples

- $\{ab\}$ is denoted by `(a)` $\bullet$ `(b)`.
- $\{a, b\}$ is denoted by `(a)` **U** `(b)`.

Examples

- $\{ab\}$ is denoted by $(\mathtt{a}) \bullet (\mathtt{b})$.
- $\{a, b\}$ is denoted by $(\mathtt{a}) \cup (\mathtt{b})$.
- $\{\varepsilon, ab\}$ is denoted by $(\varepsilon) \cup ((\mathtt{a}) \bullet (\mathtt{b}))$.

Examples

- $\{ab\}$ is denoted by $(\mathtt{a}) \bullet (\mathtt{b})$.
- $\{a, b\}$ is denoted by $(\mathtt{a}) \cup (\mathtt{b})$.
- $\{\varepsilon, ab\}$ is denoted by $(\varepsilon) \cup ((\mathtt{a}) \bullet (\mathtt{b}))$.
- $\{a, b\}^*$ is denoted by $((\mathtt{a}) \cup (\mathtt{b}))^*))$.

Abbreviation conventions

- The operators used in RegExp are given a decreasing binding priority omit parentheses and bullets where unambiguous.

Abbreviation conventions

- The operators used in RegExp are given a decreasing binding priority omit parentheses and bullets where unambiguous.
- Examples:

Abbreviation conventions

- The operators used in RegExp are given a decreasing binding priority omit parentheses and bullets where unambiguous.
- Examples:
  - $\{a\}^{\star}\mathtt{b}$ for $((\mathtt{a})^{\star})\bullet(\mathtt{b})$.

Abbreviation conventions

- The operators used in RegExp are given a decreasing binding priority omit parentheses and bullets where unambiguous.
- Examples:
  - $\{a\}^{\star}$b for ((a)*)•(b) .
  - a•b**U**c for ((a)•(b))**U**(c) .

Abbreviation conventions

- The operators used in RegExp are given a decreasing binding priority omit parentheses and bullets where unambiguous.
- Examples:
  - $\{a\}^{\star}\mathtt{b}$ for $((\mathtt{a})^{\star}) \bullet (\mathtt{b})$.
  - $\mathtt{a} \bullet \mathtt{b} \mathbf{U} \mathtt{c}$ for $((\mathtt{a}) \bullet (\mathtt{b})) \mathbf{U} (\mathtt{c})$.
  - $(\mathtt{a}^{\star}\mathtt{b})^{\star}$ for $((\mathtt{a})^{\star}) \bullet ((\mathtt{b})^{\star})$.