CMSC 141
Introduction to Automata

Automata and Language Theory
23 August 2015

Elemar Teje
Computer Science Instructor
Department of Physical Sciences and Mathematics
University of the Philippines Visayas

Questions

Questions

What are the fundamental capabilities and limitations of computers?

Questions

What are the fundamental capabilities and limitations of computers?
What makes some problems computationally hard and others easy?

Introduction to Automata Theory

Questions

What are the fundamental capabilities and limitations of computers?
What makes some problems computationally hard and others easy?

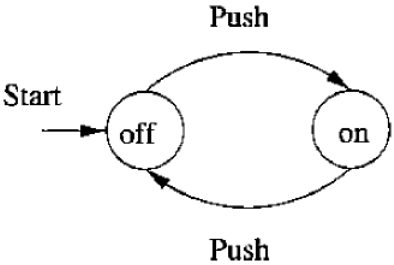
Automata Theory

Automata theory deals with the definitions and properties of mathematical models of computation.

Outline

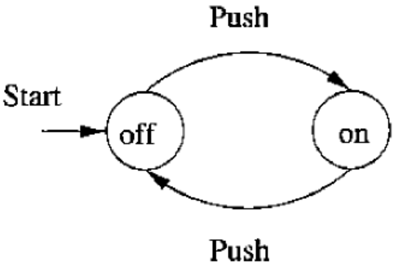
Introduction to Finite Automata

Example 1 On/Off Switch

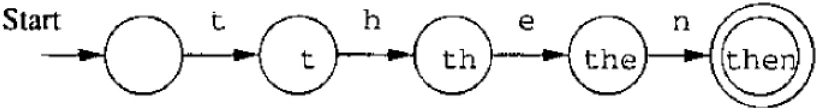


Introduction to Finite Automata

Example 1 On/Off Switch



Example 2 Recognition of 'then'



Outline

Structural Representations

Grammars

Grammars are useful models when designing software that process data with a recursive structure. The best-known is a "parser".

Structural Representations

Grammars

Grammars are useful models when designing software that process data with a recursive structure. The best-known is a "parser".

Regular Expressions

Regular Expressions also denote the structure of data, especially text strings.

Outline

Decidability

What can a computer do at all?

The problems that can be solved by computer are called *decidable*.

Structural Representations

Decidability

What can a computer do at all?
The problems that can be solved by computer are called *decidable*.

Intractability

What can a computer do efficiently?
The problems that can be solved by a computer using no more time than some *slowly growing function* (polynomial functions) of the size of the input are called *tractable*.

Outline

Formal Proofs

Deductive Proofs

Consists of a sequence of statements whose truth leads us from some initial statement called *hypothesis*, or the *given statement(s)*, to a *conclusion statement*.

Each step in the proof must follow, by some accepted logical principle, fromt either the given facts, or some of the previous statements in the deductive proof, or a combination of these.

Formal Proofs

Deductive Proofs

Consists of a sequence of statements whose truth leads us from some initial statement called *hypothesis*, or the *given statement(s)*, to a *conclusion statement*.

Each step in the proof must follow, by some accepted logical principle, fromt either the given facts, or some of the previous statements in the deductive proof, or a combination of these.

Theorem Forms

Formal Proofs

Deductive Proofs

Consists of a sequence of statements whose truth leads us from some initial statement called *hypothesis*, or the *given statement(s)*, to a *conclusion statement*.

Each step in the proof must follow, by some accepted logical principle, fromt either the given facts, or some of the previous statements in the deductive proof, or a combination of these.

Theorem Forms

- "If-Then"

Formal Proofs

Deductive Proofs

Consists of a sequence of statements whose truth leads us from some initial statement called *hypothesis*, or the *given statement(s)*, to a *conclusion statement*.

Each step in the proof must follow, by some accepted logical principle, fromt either the given facts, or some of the previous statements in the deductive proof, or a combination of these.

Theorem Forms

- "If-Then"
- If-And-Only-If Statements

Formal Proofs

Deductive Proofs

Consists of a sequence of statements whose truth leads us from some initial statement called *hypothesis*, or the *given statement(s)*, to a *conclusion statement*.

Each step in the proof must follow, by some accepted logical principle, fromt either the given facts, or some of the previous statements in the deductive proof, or a combination of these.

Theorem Forms

- "If-Then"
- If-And-Only-If Statements
- Not If-Then Statements

Outline

Other Forms of Proof

Proofs about Sets

Other Forms of Proof

Proofs about Sets

- Equivalence about Sets

Other Forms of Proof

Proofs about Sets

- Equivalence about Sets
- The Contrapositive

Proofs about Sets

- Equivalence about Sets
- The Contrapositive

Proofs by

Other Forms of Proof

Proofs about Sets

- Equivalence about Sets
- The Contrapositive

Proofs by

- Contradiction

Other Forms of Proof

Proofs about Sets

- Equivalence about Sets
- The Contrapositive

Proofs by

- Contradiction
- Counterexample

Other Forms of Proof

Proofs about Sets

- Equivalence about Sets
- The Contrapositive

Proofs by

- Contradiction
- Counterexample

Inductive Proofs

Other Forms of Proof

Proofs about Sets

- Equivalence about Sets
- The Contrapositive

Proofs by

- Contradiction
- Counterexample

Inductive Proofs

- Integer Inductions

Other Forms of Proof

Proofs about Sets

- Equivalence about Sets
- The Contrapositive

Proofs by

- Contradiction
- Counterexample

Inductive Proofs

- Integer Inductions
- Structural Inductions

Other Forms of Proof

Proofs about Sets

- Equivalence about Sets
- The Contrapositive

Proofs by

- Contradiction
- Counterexample

Inductive Proofs

- Integer Inductions
- Structural Inductions
- Mutual Inductions

Outline

Alphabets

An *alphabet* is a finite, nonempty set of symbols. Conventionally, we use the symbol Σ for an alphabet. Common alphabets include:

Alphabets

An *alphabet* is a finite, nonempty set of symbols. Conventionally, we use the symbol Σ for an alphabet. Common alphabets include:

- $\Sigma = \{0, 1\}$, the binary alphabet

Alphabets

An *alphabet* is a finite, nonempty set of symbols. Conventionally, we use the symbol Σ for an alphabet. Common alphabets include:

- $\Sigma = \{0, 1\}$, the binary alphabet
- $\Sigma = \{a, b, \ldots, z\}$, the set of all lowercase letters

Alphabets

An *alphabet* is a finite, nonempty set of symbols. Conventionally, we use the symbol Σ for an alphabet. Common alphabets include:

- $\Sigma = \{0, 1\}$, the binary alphabet
- $\Sigma = \{a, b, \ldots, z\}$, the set of all lowercase letters
- The set of all ASCII characters, or the set of all printable ASCII characters.

Outline

Strings

A *string* (or sometimes *word*) is a finite sequence of symbols chosen from some alphabet. For example 10101 is a string from the binary alphabet $\Sigma = \{0, 1\}$. Same as the string 111.

Strings

A *string* (or sometimes *word*) is a finite sequence of symbols chosen from some alphabet. For example 10101 is a string from the binary alphabet $\Sigma = \{0, 1\}$. Same as the string 111.

Empty String

The *empty string* is the string with zero occurrences of symbols. This string, denoted by ϵ, is a string that may be chosen from any alphabet whatsoever.

The Central Concepts of Automata Theory

Strings

A *string* (or sometimes *word*) is a finite sequence of symbols chosen from some alphabet. For example 10101 is a string from the binary alphabet $\Sigma = \{0, 1\}$. Same as the string 111.

Empty String

The *empty string* is the string with zero occurrences of symbols. This string, denoted by ϵ, is a string that may be chosen from any alphabet whatsoever.

Length of a String

Strings are often classified by their *length*, that is, the number of positions for symbols in the string. For instance, 10101 has length 5. The standard notation for the length of a string w is $|w|$. For example, $|110| = 3$ and $|\epsilon| = 0$.

Powers of an Alphabet

If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using an exponential notation. We define Σ^k to be the set of strings of length k, each of whose symbols is in Σ.

Powers of an Alphabet

If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using an exponential notation. We define Σ^k to be the set of strings of length k, each of whose symbols is in Σ.

Type Convention for Symbols and Strings

Commonly, we shall use lower-case letters at the beginning of the alphabet (or digits) to denote symbols, and lower-case letters near the end of the alphabet, typically w, x, y, and z, to denote strings.

Powers of an Alphabet

If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using an exponential notation. We define Σ^k to be the set of strings of length k, each of whose symbols is in Σ.

Type Convention for Symbols and Strings

Commonly, we shall use lower-case letters at the beginning of the alphabet (or digits) to denote symbols, and lower-case letters near the end of the alphabet, typically w, x, y, and z, to denote strings.

Example 1

Note that $\Sigma^0 = \{\epsilon\}$, regardless of what alphabet Σ is.
If $\Sigma = \{0, 1\}$, then $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$,
$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
Confusion with Σ and Σ^1.

Powers of an Alphabet

The set of all strings over an alphabet Σ is conventionally denoted by Σ^* (The $*$ symbol is called the **Kleene star**, and is named after the mathematician and logician Stephen Cole Kleene). For instance, $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. Put another way,

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

The set of nonempty strings from alphabet Σ is denoted by Σ^+. Thus,

Powers of an Alphabet

The set of all strings over an alphabet Σ is conventionally denoted by Σ^* (The $*$ symbol is called the **Kleene star**, and is named after the mathematician and logician Stephen Cole Kleene). For instance, $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. Put another way,

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

The set of nonempty strings from alphabet Σ is denoted by Σ^+. Thus,

- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$

The Central Concepts of Automata Theory

Powers of an Alphabet

The set of all strings over an alphabet Σ is conventionally denoted by Σ^* (The $*$ symbol is called the **Kleene star**, and is named after the mathematician and logician Stephen Cole Kleene). For instance, $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. Put another way,

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

The set of nonempty strings from alphabet Σ is denoted by Σ^+. Thus,

- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$
- $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$.

Concatenation of Strings

Let x and y be strings. Then xy denotes the *concatenation* of x and y, that is, the string formed by making a copy of x following it by a copy of y.

Concatenation of Strings

Let x and y be strings. Then xy denotes the *concatenation* of x and y, that is, the string formed by making a copy of x following it by a copy of y.

Example 2

Let $x = 10101$ and $y = 110$, then $xy = 10101110$ and $yx = 11010101$. For any string w, the equations $\epsilon w = w\epsilon = w$ hold. That is, ϵ is the *identity for concatenation*.

Outline

Languages

A *Language* is a set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet. If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a *language over* Σ.

Languages

A *Language* is a set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet. If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a *language over* Σ.

Example 3

The Central Concepts of Automata Theory

Languages

A *Language* is a set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet. If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a *language over* Σ.

Example 3

- The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$: $\{\epsilon, 01, 0011, 000111, \dots\}$.

Languages

A *Language* is a set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet. If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a *language over* Σ.

Example 3

- The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$: $\{\epsilon, 01, 0011, 000111, \dots\}$.
- The set of strings of 0's and 1's with an equal number of each: $\{\epsilon, 01, 10, 0011, 0101, 1001, \dots\}$.

The Central Concepts of Automata Theory

Languages

A *Language* is a set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet. If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a *language over* Σ.

Example 3

- The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$: $\{\epsilon, 01, 0011, 000111, \ldots\}$.
- The set of strings of 0's and 1's with an equal number of each: $\{\epsilon, 01, 10, 0011, 0101, 1001, \ldots\}$.
- The set of binary numbers whose value is a prime: $\{10, 11, 101, 111, 1011, \ldots\}$

Languages

A *Language* is a set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet. If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a *language over* Σ.

Example 3

- The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$: $\{\epsilon, 01, 0011, 000111, \ldots\}$.
- The set of strings of 0's and 1's with an equal number of each: $\{\epsilon, 01, 10, 0011, 0101, 1001, \ldots\}$.
- The set of binary numbers whose value is a prime: $\{10, 11, 101, 111, 1011, \ldots\}$
- Σ^* is a language for any alphabet Σ.

The Central Concepts of Automata Theory

Languages

A *Language* is a set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet. If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a *language over* Σ.

Example 3

- The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$: $\{\epsilon, 01, 0011, 000111, \ldots\}$.
- The set of strings of 0's and 1's with an equal number of each: $\{\epsilon, 01, 10, 0011, 0101, 1001, \ldots\}$.
- The set of binary numbers whose value is a prime: $\{10, 11, 101, 111, 1011, \ldots\}$
- Σ^* is a language for any alphabet Σ.
- $\emptyset$, the empty language, is a language over any alphabet.

Example 3

- The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$: $\{\epsilon, 01, 0011, 000111, \ldots\}$.
- The set of strings of 0's and 1's with an equal number of each: $\{\epsilon, 01, 10, 0011, 0101, 1001, \ldots\}$.
- The set of binary numbers whose value is a prime: $\{10, 11, 101, 111, 1011, \ldots\}$
- Σ^* is a language for any alphabet Σ.
- $\emptyset$, the empty language, is a language over any alphabet.
- $\{\epsilon\}$, the language consisting of only the empty string, is also a language over any alphabet. Note: $\emptyset \neq \{\epsilon\}$.

Outline

Problems

A *problem* is a question of deciding whether a given string is a member of some particular language. More precisely, if Σ is an alphabet, and L is a language over Σ, then the problem L_p is: Given a string w in Σ^*, decide whether or not w is in L.

Problems

A *problem* is a question of deciding whether a given string is a member of some particular language. More precisely, if Σ is an alphabet, and L is a language over Σ, then the problem L_p is: Given a string w in Σ^*, decide whether or not w is in L.

Set Formers as a Way to Define Languages

Problems

A *problem* is a question of deciding whether a given string is a member of some particular language. More precisely, if Σ is an alphabet, and L is a language over Σ, then the problem L_p is: Given a string w in Σ^*, decide whether or not w is in L.

Set Formers as a Way to Define Languages

1. $\{w|$ something about $w\}$

Problems

A *problem* is a question of deciding whether a given string is a member of some particular language. More precisely, if Σ is an alphabet, and L is a language over Σ, then the problem L_p is: Given a string w in Σ^*, decide whether or not w is in L.

Set Formers as a Way to Define Languages

1. $\{w|$ something about $w\}$
   - $\{w|w$ consists of an equal number of 0's and 1's$\}$.

Problems

A *problem* is a question of deciding whether a given string is a member of some particular language. More precisely, if Σ is an alphabet, and L is a language over Σ, then the problem L_p is: Given a string w in Σ^*, decide whether or not w is in L.

Set Formers as a Way to Define Languages

1. $\{w|$ something about $w\}$
   - $\{w|w$ consists of an equal number of 0's and 1's$\}$.
   - $\{w|w$ is a binary integer that is prime$\}$

The Central Concepts of Automata Theory

Problems

A *problem* is a question of deciding whether a given string is a member of some particular language. More precisely, if Σ is an alphabet, and L is a language over Σ, then the problem L_p is: Given a string w in Σ^*, decide whether or not w is in L.

Set Formers as a Way to Define Languages

1. $\{w|$ something about $w\}$
   - $\{w|w$ consists of an equal number of 0's and 1's$\}$.
   - $\{w|w$ is a binary integer that is prime$\}$
   - $\{w|w$ is a syntactically correct C program$\}$

The Central Concepts of Automata Theory

Problems

A *problem* is a question of deciding whether a given string is a member of some particular language. More precisely, if Σ is an alphabet, and L is a language over Σ, then the problem L_p is: Given a string w in Σ^*, decide whether or not w is in L.

Set Formers as a Way to Define Languages

1. $\{w|$ something about $w\}$
    - $\{w|w$ consists of an equal number of 0's and 1's$\}$.
    - $\{w|w$ is a binary integer that is prime$\}$
    - $\{w|w$ is a syntactically correct C program$\}$
2. {parameters | parameter condition(s)}

The Central Concepts of Automata Theory

Problems

A *problem* is a question of deciding whether a given string is a member of some particular language. More precisely, if Σ is an alphabet, and L is a language over Σ, then the problem L_p is: Given a string w in Σ^*, decide whether or not w is in L.

Set Formers as a Way to Define Languages

1. $\{w|$ something about $w\}$
   - $\{w|w$ consists of an equal number of 0's and 1's$\}$.
   - $\{w|w$ is a binary integer that is prime$\}$
   - $\{w|w$ is a syntactically correct C program$\}$
2. $\{$parameters $|$ parameter condition(s)$\}$
   - $\{0^n 1^n | n \geq 1\}$.

The Central Concepts of Automata Theory

Problems

A *problem* is a question of deciding whether a given string is a member of some particular language. More precisely, if Σ is an alphabet, and L is a language over Σ, then the problem L_p is: Given a string w in Σ^*, decide whether or not w is in L.

Set Formers as a Way to Define Languages

1. $\{w|$ something about $w\}$
   - $\{w|w$ consists of an equal number of 0's and 1's$\}$.
   - $\{w|w$ is a binary integer that is prime$\}$
   - $\{w|w$ is a syntactically correct C program$\}$
2. $\{$parameters $|$ parameter condition(s)$\}$
   - $\{0^n 1^n | n \geq 1\}$.
   - $\{0^i 1^j | 0 \leq i \leq j\}$.

End of Lesson!
Next Lesson: Finite Automata