CM0081 Automata and Formal Languages
§ 1.5 The Central Concepts of Automata Theory

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2024-1

Preliminaries

Conventions

- The number and page numbers assigned to chapters, examples, exercises, figures, quotes, sections and theorems on these slides correspond to the numbers assigned in the textbook [Hopcroft, Motwani and Ullman 2007].
- The natural numbers include the zero, that is, $\mathbb{N} = \{0, 1, 2, \dots\}$.
- The power set of a set A, that is, the set of its subsets, is denoted by $\mathcal{P}\,A$.

Alphabets and Strings

Definition

An **alphabet** is a finite, non-empty set of symbols.

Alphabets and Strings

Definition

An **alphabet** is a finite, non-empty set of symbols.

Examples

$$\Sigma_1 = \{0, 1\},$$
$$\Sigma_2 = \{a, b, \ldots, z\},$$
$$\Sigma_3 = \{\, x \mid x \text{ is a Unicode codepoint} \,\}.$$

Alphabets and Strings

Definition

A **string** (or **word** or **event**) is a finite sequence of symbols of an alphabet.

Alphabets and Strings

Definition

A **string** (or **word** or **event**) is a finite sequence of symbols of an alphabet.

Definition

The **empty string**, denoted by ε, is the string with zero occurrences of symbols.

Observation

The empty string may be chosen from any alphabet.

Alphabets and Strings

Definition

A **string** (or **word** or **event**) is a finite sequence of symbols of an alphabet.

Definition

The **empty string**, denoted by ε, is the string with zero occurrences of symbols.

Observation

The empty string may be chosen from any alphabet.

Conventions

Alphabets: $\Sigma, \Gamma, \ldots$
Symbols: $a, b, c, \ldots$
Strings: $w, x, y, z, \ldots$

All Strings over an Alphabet

Definition

Let Σ be an alphabet. The **set of all the strings over Σ** (including the empty string), denoted Σ^*, can be inductively defined by the following clauses:

i) Basis step: $\varepsilon \in \Sigma^*$,

ii) Inductive step: If $x \in \Sigma^*$ and $a \in \Sigma$ then $xa \in \Sigma^*$.

Or, equivalently, by using the following inference rules:

$$\frac{}{\varepsilon \in \Sigma^*} \qquad\qquad \frac{x \in \Sigma^* \qquad a \in \Sigma}{xa \in \Sigma^*}$$

Operations on Alphabets

Definition

Let a be a symbol on an alphabet Σ. The **powers of $\mathbf{a}$**, denoted a^n, with $n \geq 0$, is the string formed by n repetitions of the symbol a (see, e.g. [Kozen 2012]). This operation is recursively defined by:

$$
\begin{gathered}
(-)^{(-)} : \Sigma \times \mathbb{N} \to \Sigma^* \\
a^0 = \varepsilon, \\
a^{n+1} = a^n a.
\end{gathered}
$$

Operations on Strings

Definition

Let Σ be an alphabet. The **length** of a string x on Σ, denoted $|x|$ is the number of symbols in x. This function is recursively defined by

$$
\begin{aligned}
|-| &: \Sigma^* \to \mathbb{N} \\
|\varepsilon| &= 0, \\
|xa| &= |x| + 1.
\end{aligned}
$$

Operations on Strings

Definition

Let Σ be an alphabet. The **concatenation** of strings is recursively defined by

$$(-) \cdot (-) : \Sigma^* \times \Sigma^* \to \Sigma^*$$
$$x \cdot \varepsilon = x,$$
$$x \cdot ya = (x \cdot y)a.$$

Operations on Strings

Definition

Let Σ be an alphabet. The **concatenation** of strings is recursively defined by

$$(-) \cdot (-) : \Sigma^* \times \Sigma^* \to \Sigma^*$$
$$x \cdot \varepsilon = x,$$
$$x \cdot ya = (x \cdot y)a.$$

That is, let $x = a_1 a_2 \dots a_n$ and $y = b_1 b_2 \dots b_n$ two strings, then

$$x \cdot y = a_1 a_2 \dots a_m b_1 b_2 \dots b_n.$$

Operations on Strings

Definition

Let Σ be an alphabet. The **concatenation** of strings is recursively defined by

$$(-) \cdot (-) : \Sigma^* \times \Sigma^* \to \Sigma^*$$
$$x \cdot \varepsilon = x,$$
$$x \cdot ya = (x \cdot y)a.$$

That is, let $x = a_1 a_2 \dots a_n$ and $y = b_1 b_2 \dots b_n$ two strings, then

$$x \cdot y = a_1 a_2 \dots a_m b_1 b_2 \dots b_n.$$

Notation

We remove the dot in the concatenation, that is, $xy := x \cdot y$.

Operations on Strings

Some properties of concatenation

Let x, y and z be strings.

(i) Concatenation is associative, that is, $x(yz) = (xy)z$.

(ii) The empty empty word is the unit for concatenation, that is, $x\varepsilon = \varepsilon x = x$.

(iii) Concatenation is not commutative, that is, $xy \neq yx$.

Operations on Strings

Example

Let Σ be an alphabet and let x and y be strings over Σ. Prove that

$$|xy| = |x| + |y|.$$

Operations on Strings

Proof

By structural induction on y.

- Basis step ($y = \varepsilon$):

$$\begin{aligned} |x\varepsilon| &= |x| && \text{(def. of concatenation)} \\ &= |x| + |\varepsilon| && \text{(def. of length)} \end{aligned}$$

- Induction step ($y = wa$):

$$\begin{aligned} |x(wa)| &= |(xw)a| && \text{(def. of concatenation)} \\ &= |xw| + 1 && \text{(def. of length)} \\ &= (|x| + |w|) + 1 && \text{(IH)} \\ &= |x| + (|w| + 1) && \text{(arithmetic)} \\ &= |x| + |wa| && \text{(def. of length)} \end{aligned}$$

■

Operations on Strings

Proof

By structural induction on y (or by mathematical induction on $|y|$).

- Basis step ($y = \varepsilon$) (or $|y| = 0$, then $y = \varepsilon$):

$$\begin{aligned} |x\varepsilon| &= |x| && \text{(def. of concatenation)} \\ &= |x| + |\varepsilon| && \text{(def. of length)} \end{aligned}$$

- Induction step ($y = wa$) (or $|y| = n + 1$, then $y = wa$ where $|w| = n$):

$$\begin{aligned} |x(wa)| &= |(xw)a| && \text{(def. of concatenation)} \\ &= |xw| + 1 && \text{(def. of length)} \\ &= (|x| + |w|) + 1 && \text{(IH)} \\ &= |x| + (|w| + 1) && \text{(arithmetic)} \\ &= |x| + |wa| && \text{(def. of length)} \end{aligned}$$

■

Operations on Strings

Strings, length and concatenation in Haskell

```
data List   a = Nil | Cons a (List a)
data RList  a = Lin | Snoc (RList a) a
```

Operations on Strings

Strings, length and concatenation in Haskell

```haskell
data List   a = Nil | Cons a (List a)
data RList  a = Lin | Snoc (RList a) a
```

```haskell
lengthR :: RList a -> Int
lengthR Lin           = 0                -- Eq. 1
lengthR (Snoc xs x)   = lengthR xs + 1   -- Eq. 2
```

Operations on Strings

Strings, length and concatenation in Haskell

```haskell
data List   a = Nil | Cons a (List a)
data RList  a = Lin | Snoc (RList a) a
```

```haskell
lengthR :: RList a -> Int
lengthR Lin           = 0               -- Eq. 1
lengthR (Snoc xs x)   = lengthR xs + 1  -- Eq. 2
```

```haskell
(+++) :: RList a -> RList a -> RList a
(+++) xs Lin           = xs                   -- Eq. 1
(+++) xs (Snoc ys y)   = Snoc (xs +++ ys) y   -- Eq. 2
```

Operations on Strings

Example

Prove that `lengthR (xs +++ ys)` =`lengthR xs + lengthR ys`.

Operations on Strings

Example

Prove that `lengthR (xs +++ ys)` = `lengthR xs + lengthR ys`.

Proof by structural recursion on ys

- Basis step (ys is `Lin`):

```
lengthR (xs +++ Lin)
 = lengthR xs                        (Eq. 1 of (+++))
 = lengthR xs +++ lengthR Lin        (Eq. 1 of lengthR)
```

Operations on Strings

Proof by structural recursion on |ys| (continuation)

- Induction step (ys is `Snoc ys' y'`):

```
lengthR (xs +++ (Snoc ys' y'))
 = lengthR (Snoc (xs +++ ys') y'))        (Eq. 2 of (+++))
 = lengthR (xs +++ ys') + 1               (Eq. 2 of lengthR)
 = (lengthR xs + lengthR ys') + 1         (IH)
 = lengthR xs + (lengthR ys' + 1)         (arithmetic)
 = lengthR xs + lengthR (Snoc ys' y)      (Eq. 2 of lengthR)
```

∎

Operations on Strings

Definition

Let x be a string on an alphabet Σ. The **powers of $\mathbf{x}$**, denoted x^n, with $n \geq 0$, is recursively defined by

$$
\begin{aligned}
(-)^{(-)} &: \Sigma^* \times \mathbb{N} \to \Sigma^* \\
x^0 &= \varepsilon, \\
x^{n+1} &= x^n \cdot x.
\end{aligned}
$$

Operations Alphabets

Definition

The $\mathbf{n}$-**power** of an alphabet Σ, denoted Σ^n, is the set of strings of length n over Σ.

Examples

Given $\Sigma = \{0, 1\}$ then

$$\Sigma^0 = \{\varepsilon\},$$
$$\Sigma^1 = \{0, 1\},$$
$$\Sigma^2 = \{00, 01, 10, 11\},$$
$$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}.$$

Operations Alphabets

Example

Let Σ be an alphabet. Then

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

Languages

Definition

If Σ is an alphabet and $L \subseteq \Sigma^*$ then L is a **language** over Σ.

Languages

Definition

If Σ is an alphabet and $L \subseteq \Sigma^*$ then L is a **language** over Σ.

Examples

- $\emptyset$ and Σ^* are languages over any alphabet

Languages

Definition

If Σ is an alphabet and $L \subseteq \Sigma^*$ then L is a **language** over Σ.

Examples

- $\emptyset$ and Σ^* are languages over any alphabet
- The set of string of 0's and 1's with equal number of each

$$\{\varepsilon, 01, 10, 0011, 0110, 1001, 1100, \dots\}$$

Languages

Definition

If Σ is an alphabet and $L \subseteq \Sigma^*$ then L is a **language** over Σ.

Examples

- $\emptyset$ and Σ^* are languages over any alphabet
- The set of string of 0's and 1's with equal number of each

$$\{\varepsilon, 01, 10, 0011, 0110, 1001, 1100, \ldots\}$$

- $\{\, 0^n 1^n \mid n \geq 1 \,\}$

Languages

Definition

If Σ is an alphabet and $L \subseteq \Sigma^*$ then L is a **language** over Σ.

Examples

- $\emptyset$ and Σ^* are languages over any alphabet
- The set of string of 0's and 1's with equal number of each

$$\{\varepsilon, 01, 10, 0011, 0110, 1001, 1100, \dots\}$$

- $\{\, 0^n 1^n \mid n \geq 1 \,\}$
- $\{\varepsilon\} \neq \emptyset$

Languages

Definition

If Σ is an alphabet and $L \subseteq \Sigma^*$ then L is a **language** over Σ.

Examples

- $\emptyset$ and Σ^* are languages over any alphabet
- The set of string of 0's and 1's with equal number of each

$$\{\varepsilon, 01, 10, 0011, 0110, 1001, 1100, \dots\}$$

- $\{\, 0^n 1^n \mid n \geq 1 \,\}$
- $\{\varepsilon\} \neq \emptyset$
- The set of binary numbers whose value is a prime

Languages

Definition

If Σ is an alphabet and $L \subseteq \Sigma^*$ then L is a **language** over Σ.

Examples

- $\emptyset$ and Σ^* are languages over any alphabet
- The set of string of 0's and 1's with equal number of each

$$\{\varepsilon, 01, 10, 0011, 0110, 1001, 1100, \dots\}$$

- $\{ 0^n 1^n \mid n \geq 1 \}$
- $\{\varepsilon\} \neq \emptyset$
- The set of binary numbers whose value is a prime
- The set of legal C programs

Languages

Question

Is the set of legal English words a language?

Decision Problems

Definition

Let a set A the domain of a problem. A **decision problem** on A is a function (see, e.g. [Kozen 2012])

$$f : A \to \{0, 1\}.$$

Decision Problems

Definition

Let $L \subseteq \Sigma^*$ be a language and let $w \in \Sigma^*$ be string. The **decision problem for $\mathbf{L}$** is to decide whether or not $w \in L$.

Decision Problems

Definition

Let $L \subseteq \Sigma^*$ be a language and let $w \in \Sigma^*$ be string. The **decision problem for $\mathbf{L}$** is to decide whether or not $w \in L$.

Some questions

(i) Is it a problem or a decision problem?

Decision Problems

Definition

Let $L \subseteq \Sigma^*$ be a language and let $w \in \Sigma^*$ be string. The **decision problem for $\mathbf{L}$** is to decide whether or not $w \in L$.

Some questions

(i) Is it a problem or a decision problem?

(ii) Is it a language or a problem?

Decision Problems

Definition

Let $L \subseteq \Sigma^*$ be a language and let $w \in \Sigma^*$ be string. The **decision problem for $\mathbf{L}$** is to decide whether or not $w \in L$.

Some questions

(i) Is it a problem or a decision problem?

(ii) Is it a language or a problem?

(iii) Is the problem decidable or undecidable?

Decision Problems

Definition

Let $L \subseteq \Sigma^*$ be a language and let $w \in \Sigma^*$ be string. The **decision problem for $\mathbf{L}$** is to decide whether or not $w \in L$.

Some questions

(i) Is it a problem or a decision problem?

(ii) Is it a language or a problem?

(iii) Is the problem decidable or undecidable?

(iv) Is the problem tractable or intractable?

References

- Hopcroft, J. E., Motwani, R. and Ullman, J. D. [1979] (2007). Introduction to Automata Theory, Languages, and Computation. 3rd ed. Pearson Education (cit. on p. 2).
- Kozen, D. C. [1997] (2012). Automata and Computability. Third printing. Undergraduate Texts in Computer Science. Springer. DOI: 10.1007/978-1-4612-1844-9 (cit. on pp. 9, 35).