1.6 Alphabet, strings and languages

Automata theory $\sim$ basic models and methods of discrete signal processing

The concept of an automaton is a *mathematical abstraction*. An automaton may be realized e.g. as an electrical circuit, as a mechanical machine or (most usually) as a computer program.

In this course we concentrate on automata, whose

1. inputs are finite, discrete *strings*
2. outputs are of the form "accept"/"reject" ($\sim$ "input OK"/"input invalid")

Generalizations:

- infinite input strings ($\rightarrow$ "reactive" systems, Büchi automata)
- function automata ($\rightarrow$ Moore automata, Mealy automata, Turing function machines)

Basic concepts and notation

Alphabet (vocabulary): a finite nonempty set of *symbols*. Example

- *the binary alphabet* $\{0,1\}$;
- *the Latin alphabet* $\{A,B,\ldots,Z\}$.

string : an ordered sequence of symbols from an alphabet. Ex.:

- "01001", "0000": strings over the binary alphabet
- "TKTP", "XYZZY": strings over the Latin alphabet.
- *the empty string*. The empty string contains no symbols. It may be denoted by ε.

The *length* $|x|$ of a string x is the number of symbols in it. Ex.: $|01001| = 5$, $|TCS| = 3$, $|\varepsilon| = 0$.

The basic operation on strings is *catenation*, i.e., joining the strings. Catenation is sometimes denoted with ^. E.g.

- KALA^KUKKO = KALAKUKKO;
- if $x = 00$ and $y = 11$, then $xy = 0011$ and $yx = 1100$;
- for all x it holds that $x\varepsilon = \varepsilon x = x$;
- for all x, y, z it holds that $(xy)z = x(yz)$;
- for all x, y it holds that $|xy| = |x| + |y|$.

The set of all strings over an alphabet Σ is denoted by Σ^*. For example if $\Sigma = \{0,1\}$, then $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, \ldots\}$.

Any arbitrary subset $A \subseteq \Sigma^*$ is called *a formal language* over the alphabet Σ.

Automata and formal languages

Let M be an automaton whose inputs are strings over the alphabet Σ and the result is either "input accepted"/"input rejected". (Denoted with $1/0$.)

Let $M(x)$ denote the result of running M with the input x and let A_M be the set of inputs that M accepts, i.e.,

$$A_M = \{x \in \Sigma^* \mid M(x) = 1\}.$$

We say that M *recognizes* the language $A_M \subseteq \Sigma^*$.

An idea of automata theory: *the structure of M is reflected in the properties of the language A_M.*

Conversely: for a given desired I/O-mapping $f : \Sigma^* \to \{0,1\}$, by examining the language

$$A_f = \{x \in \Sigma^* \mid f(x) = 1\}$$

we may obtain some ideas as to what kind of an automaton is needed for computing f.

Established notation

In principle, notation may be freely chosen, but observing certain conventions make communication easier. The following notation is established:

Alphabet: Σ, Γ, ... (capital Greek letters). Ex. the binary alphabet $\Sigma = \{0,1\}$.

The size of an alphabet (the cardinality of a set): $|\Sigma|$.

Symbols: a, b, c, ... (small Latin letters from the beginning of the alphabet). Ex.: Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet; then $|\Sigma| = n$.

Strings: u, v, w, x, y, ... (small Latin letters near the end of the alphabet).

Catenation of strings: $x^\frown y$ or just xy.

Length of a string: $|x|$. *Examples:*

- $|abc| = 3$;
- let $x = a_1 \ldots a_m$, $y = b_1 \ldots b_n$; then $|xy| = m + n$.

The empty string: ε.

A string of the symbol a repeated n times: a^n. Examples:

- $a^n = \underbrace{aa\ldots a}_{n \text{ times}}$;
- $|a^i b^j c^k| = i + j + k$.

Repeating a string x a total of k times: x^k. Examples:

- $(ab)^2 = abab$;
- $|x^k| = k|x|$.

The set of all strings over the alphabet Σ Σ^*. Ex..:

- $\{a,b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$.

Induction on strings

In automata theory, constructions are often carried out by "induction on the length of the string." This means that a function is first defined for the empty string ε (or occasionally for strings of length one). Then it is assumed that the function is defined for all strings u of a given length and this is used to define the function for strings $w = ua$ that are one symbol longer.

Example. Let Σ be an arbitrary alphabet. The *reversal* w^R of the string $w \in \Sigma^*$ is defined inductively by:

1. $\varepsilon^R = \varepsilon$;
2. if $w = ua$, $u \in \Sigma^*$, $a \in \Sigma$, then $w^R = a\hat{}u^R$.

The inductive (recursive) definition can be used for calculations, for example:

$$\begin{array}{rclcl} (011)^R & = & 1\hat{}(01)^R & = & 1\hat{}(1\hat{}0^R) \\ & = & 11\hat{}(0\hat{}\varepsilon^R) & = & 110\hat{}\varepsilon^R \\ & = & 110\hat{}\varepsilon & = & 110. \end{array}$$

but it is more importantly used for proving properties of constructions by induction. Example:

Claim. Let Σ be an alphabet. For all $x, y \in \Sigma^*$ it holds that $(xy)^R = y^R x^R$.

Proof. Induction over the length of y.

1. The induction base $y = \varepsilon$: $(x\varepsilon)^R = x^R = \varepsilon^R x^R$.
2. The induction step: Let y be of the form $y = ua$, $u \in \Sigma^*$, $a \in \Sigma$. Assume that the claim holds for x, u. Then we have:

$$\begin{array}{rcll} (xy)^R & = & (xua)^R & \\ & = & a\hat{}(xu)^R & \text{[definition of } R\text{]} \\ & = & a\hat{}(u^R x^R) & \text{[induction assumption]} \\ & = & (a\hat{}u^R)x^R & \text{[associativity of } \hat{}\text{]} \\ & = & (ua)^R x^R & \text{[definition of } R\text{]} \\ & = & y^R x^R. \square & \end{array}$$

1.7 Countable and uncountable sets

Definition 1.10 A set X is *countably infinite* if there is a bijection $f : \mathbb{N} \to X$. An infinite set that is not countably infinite is *uncountable*. A set is *countable*, if it is finite or countably infinite.

Intuitively speaking X is countable, if its elements can be ordered and indexed with natural numbers:

$$X = \{x_0, x_1, x_2, \ldots, x_{n-1}\},$$

if X is a finite set with n elements and

$$X = \{x_0, x_1, x_2, \ldots\},$$

if X is countably infinite.

All subsets of a countable set are also countable, but uncountable sets have both countable and uncountable subsets. Thus uncountable sets are in a certain sense "larger" than countable sets.

Theorem 1.11 The set of strings Σ^* over any alphabet Σ is countably infinite.

Proof. Form the bijection $f : \mathbb{N} \to \Sigma^*$ as follows. Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$. Define an "alphabetical order" on the symbols in Σ; let the order be $a_1 < a_2 < \cdots < a_n$.

The strings in Σ^* may now be listed in the *canonical* order with regard to the alphabetical order as follows: (engl. canonical t. lexicographic order) seuraavasti:

- first strings of length 0 are listed (= ε), then strings of length 1 (= $a_1, a_2, \ldots, a_n$), then strings of length 2, etc.;
- strings of the same length are ordered alphabetically among themselves.

The bijection f is thus:

$$
\begin{array}{rcl}
0 & \mapsto & \varepsilon \\
1 & \mapsto & a_1 \\
2 & \mapsto & a_2 \\
\vdots & & \vdots \\
n & \mapsto & a_n \\
n+1 & \mapsto & a_1 a_1 \\
n+2 & \mapsto & a_1 a_2 \\
\vdots & & \vdots \\
2n & \mapsto & a_1 a_n
\end{array}
\qquad
\begin{array}{rcl}
2n+1 & \mapsto & a_2 a_1 \\
\vdots & & \vdots \\
3n & \mapsto & a_2 a_n \\
\vdots & & \vdots \\
n^2+n & \mapsto & a_n a_n \\
n^2+n+1 & \mapsto & a_1 a_1 a_1 \\
n^2+n+2 & \mapsto & a_1 a_1 a_2 \\
\vdots & & \vdots
\end{array}
$$

□

In fact all programs in any programming language are strings on some alphabet (for example, over the ASCII character set for the language C). According to Theore 1.11 the set of strings over any alphabet is countably infinite, so also for any programming language the set of possible programs is also countable.

Next it is shown that the set of all formal languages is uncountable. Thus there are "more" formal languages than possible programs, and therefore *no programming language can be used to design automatons to recognize all formal languages*. (In other words there are I/O-mappings that cannot be realized on a computer.)

Theorem 1.12 The set of formal languages over any alphabet Σ is uncountable.

Proof (Cantor's diagonalization argument). Let the family of all formal languages over Σ be denoted by $\mathcal{P}(\Sigma^*) = \mathcal{A}$. Assume that it would be possible to enumerate all formal languages over Σ:

$$\mathcal{A} = \{A_0, A_1, A_2, \ldots\}.$$

Let the strings in Σ^* in canonical order be $x_0, x_1, x_2, \ldots$. Use this numbering to define the formal language $\tilde{A}$:

$$\tilde{A} = \{x_i \in \Sigma^* \mid x_i \notin A_i\}.$$

Since $\tilde{A} \in \mathcal{A}$ and the numbering of $\mathcal{A}$ was assumed to cover all languages in $\mathcal{A}$, we should have $\tilde{A} = A_k$ for some $k \in \mathbb{N}$. But then by definition of $\tilde{A}$ we have

$$x_k \in \tilde{A} \Leftrightarrow x_k \notin A_k = \tilde{A}.$$

The contradiction shows that our assumption is incorrect and no such enumeration of all formal languages over Σ can exist. □

$\tilde{A}$

↘	A_0	A_1	A_2	A_3	$\cdots$
x_0	1 /0	0	0	1	$\cdots$
x_1	0	0 /1	0	0	$\cdots$
x_2	1	1	0 /1	1	$\cdots$
x_3	0	0	0	1 /0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Pictorially the idea of the proof can be presented as follows. Form the "incidence matrix" of the languages $A_0, A_1, A_2, \ldots$ and the strings $x_0, x_1, x_2, \ldots$ so that column j of row i has value 1 if $x_i \in A_j$ and 0 otherwise. Then the language $\tilde{A}$ differs from every language A_k on the "diagonal" of the matrix:

1.8 *Excursion: Turing's Halting Problem

By Theorems 1.11 ja 1.12 the are formal languages (I/O-mappings), that cannot be realized by e.g. C-programs. Could a *concrete example* be found?

The most famous example is Turing's Halting Problem (Alan Turing, 1936). With C programs the result can be given the following form:

Claim. There is no C function `halt(p,x)`, that would receive as input the text of an arbitrary C function `p` and the input `x` to be given to `p` and returns 1, if the execution of `p` halts with input `x`, and 0 if the execution of `p` with input `x` loops endlesly.

Proof. Assume for contradiction that one could design such a function `halt`. Use `halt` to define another function `confuse`.

Let `c` stand for the program text of `confuse` and observe the computation of `confuse` with its own description `c` as input.

```
void confuse(char *p){
   int halt(char *p, char *x){
   ... /* The body of function halt. */
   }
   if (halt(p,p) == 1) while (1);
}
```

We obtain a contradiction:

`confuse(c)` halts

⇔

`halt(c,c) == 1`

⇔

`confuse(c)` does not halt.

The contradiction implies that such a function `halt` cannot exist. □

There are *lots* of similar *undecidable problems*. We will return to this towards the end of the course.