ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Prof. Dr. Andreas Podelski
Matthias Heizmann
Christian Schilling

May 20th, 2014

1. Lecture
Computer Science Theory

Chapter I – Basic definitions

§1 Modelling in theoretical computer science (pp. 1-2)

What we will learn (excerpts):

- automata (models for different views of a computer)
- applications of formal languages (regular expressions, formal grammars)
- What problems can never be solved by computers?
- What problems are (probably) hard to solve in practice?

§2 Logic, sets, relations and functions (pp. 2-4)

1.2.1 Sets (pp. 2-3)

- Sets are written in curly braces (only exception: the empty set is also written $\emptyset$).
- One way to denote *finite sets* is to explicitly enumerate the elements:

$$\{\text{element}_1, \text{element}_2, \dots, \text{element}_n\}$$

- Sets are *unordered* and do *not contain duplicates.*
  They only exist in a mathematical world.
- Sets themselves can be elements of other sets. ✎(1(a))
- concepts and operations on sets S_1, S_2 and element e:
  (a) cardinality: $|S_1|$
  (b) (non-)elementship: $e \in S_1$ and $e \notin S_2$
  (c) subset: $S_1 \subseteq S_2$
  (d) union: $S_1 \cup S_2$

(e) intersection: $S_1 \cap S_2$

(f) difference: $S_1 \setminus S_2$ ✎(1(b))

- One of the most important *infinite* sets is $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.
  This way of writing infinite sets is unsatisfactory and error-prone.

  Another way to denote sets is by *set comprehension*. This way we can also write down *infinite sets*:
  $$\{\text{element} \mid \text{property}\}$$
  Read: *the set containing all elements for which the property holds.*

- We distinguish between unbounded finiteness and infinity.

- For every set S we can define its *powerset* $\mathbb{P}(S) = \{s \mid s \subseteq S\}$.

- For sets S_1, S_2 we define the Cartesian product $S_1 \times S_2$ which denotes the set containing all *pairs* where the first entry is from S_1 and the second entry is from S_2.

  Note: A *pair* is also called *tuple*. In general, we can define n-tuples with n entries. Tuples are written in normal braces and the order of the entries matters.

1.2.2 Relations (pp. 3-4)

We skip relations for this session.

§3 Alphabets, strings and languages (pp. 5-6)

We can model problems as *formal languages*, so we need to understand the concept.

- An *alphabet* Σ is a (finite) set of *symbols/letters/characters*.

- A *word/string* w is a finite sequence of symbols (possibly empty, ε).
  $|w|$ denotes the length of the word w, i.e., the number of symbols ($|\varepsilon| = 0$).

- Σ^* denotes the set of all words over Σ.
  Σ^+ denotes the set of all nonempty words over Σ (i.e., $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$, $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$).

- Two words can be *concatenated* in the obvious way.

- The terms *substring*, *prefix*, and *suffix* are defined in the obvious way.

- A *formal language* L is a set of words over an alphabet.
  Thus we can perform usual set-theoretic operations (see previous section). ✎(2)

- Two languages L_1, L_2 can also be *concatenated*. The result is a language L_3 with exactly all possible concatenations of words from L_1 and L_2 (order matters!).

- We define the *n-th power* of a language L, L^n, and the *Kleene closure*, L^*:
$$L^n = \begin{cases} \{\varepsilon\} & \text{if } n = 0 \\ L \cdot L^{n-1} & \text{if } n > 0 \end{cases} \qquad\qquad L^* = \bigcup_{i \in \mathbb{N}} L^i$$

✎ solutions to exercises 1–3 on sheet 1 ✎

Chapter II – Finite automata and regular languages

§1 Finite automata (pp. 9-14)

2.1.1 Deterministic finite automata

- We need a way to write down a (usually infinite) language in a formal way.
  Idea: We construct a machine that reads a word and tells whether the word is in the language (*acceptance*) or not (*rejection*).

- Simplest model: read the word from left to right and only use constant memory.

- This leads to the deterministic finite automaton (DFA). It reads a (finite) word and after each symbol it changes its internal state. When the word is fully read, it checks whether the current state is final (then the word is accepted) or not.

  An automaton $\mathcal{A}$ represents its language $L(\mathcal{A})$ "indirectly": We say it *decides* for every given $w \in \Sigma^*$ the question: $w \in L(\mathcal{A})$?

  An algorithm *decides* a yes/no-problem if for every possible input it gives the correct answer after a finite amount of time.

- Formally, we write $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ with
  - finite input alphabet Σ,
  - finite set of states Q,
  - transition function δ,
  - initial state $q_0 \in Q$, and
  - set of final states $F \subseteq Q$

- The next state *only* depends on the current state and the next symbol.

- In each state for each symbol in the alphabet there *must* be an outgoing transition.

✐(3)

✎ solution to exercise 4 on sheet 1 ✐

✎ hints for exercise sheet 3 ✐