



ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Prof. Dr. Andreas Podelski
Matthias Heizmann
Christian Schilling

May 27th-28th, 2014

2. Lecture Computer Science Theory

Chapter I – Basic definitions

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

1.1.1 Relations (pp. 3-4)

The basic definition of an n -ary *relation* R is “set of n -tuples”. Typically $n = 2$, so R is a binary relation, i.e., $R \subseteq X \times Y$ for sets X, Y . There are many definitions on (binary) relations (see the script).

 hints for exercise 2 on sheet 4 


Chapter II – Finite automata and regular languages

§1 Finite automata (pp. 9-21)

2.1.1 Deterministic finite automata (continued)

In each state for each symbol in the alphabet there *must* be *exactly one* outgoing transition.

 solution to exercise 1 on sheet 2 

 (I.3)

2.1.2 Nondeterministic finite automata (NFAs)

Now we lift the above-mentioned limitation. Thus we get two additional cases when reading a symbol.

case 1 There is *more than one* outgoing transition in a state. Then the automaton nondeterministically chooses one transition to take.

case 2 There is *no* outgoing transition. Then the automaton stops and rejects.

- An alternative view is: the automaton is in several states at the same time. This can also be read as: The automaton is in a *set of states*.

A word is accepted by an NFA if and only if there is (at least) *some* sequence of transitions leading to a final state.

- An alternative view is: in each step the automaton *guesses* the next transition and, magically, it always guesses the right transition.

Even though in real life everything is deterministic, nondeterminism is an important concept in computer science. We will see it several times again in this course.

Every NFA can be converted to a DFA using the *powerset construction*.

- Hence NFAs do *not* recognize more languages than DFAs (conversion).
- Every DFA is also an NFA (simply without the nondeterminism).
- Altogether, DFAs and NFAs recognize the *same* languages.
- However, NFAs may be smaller (NFA: n states \Rightarrow DFA: up to 2^n states).

 solutions to exercises 2–3 on sheet 2 

2.1.3 ε -transitions

Once again we expand our automaton model and allow spontaneous transitions labeled by ε . The resulting ε -NFA can jump through ε -chains at any time. Every ε -NFA can be converted to an NFA:

- (a) For every state $q \in Q$ and every symbol $a \in \Sigma$ find all states q' which can be reached from q with a . Add transitions $q \xrightarrow{a} q'$.
- (b) Do one of the following alternative steps:

- (i) Make all states q final which can reach a final state via an ε -chain.
- (ii) Make q_0 final if it can reach a final state via an ε -chain.
- (c) Remove all ε -transitions.

What class of languages is accepted by ε -NFAs?

- NFAs do *not* recognize more languages than DFAs (conversion).
- Every NFA is also an ε -NFA (simply without the ε -transitions).
- Altogether, ε -NFAs and NFAs recognize the *same* languages.

 (1)

 solution to exercise 4 on sheet 2 

 solution to exercise 1 on sheet 3 

§2 Closure properties (pp. 21-23)

Let L_1, L_2 be finitely acceptable languages. Then there are finite automata $\mathcal{A}_1, \mathcal{A}_2$ with distinct states accepting them. This class is closed under

- union ($L_1 \cup L_2$): Construct the ε -NFA with a new initial state and ε -transitions to the two old initial states.
- complement ($\overline{L_1} = \Sigma^* \setminus L_1$): Final states become non-final states and vice versa. Note that this only works for DFAs.
- intersection ($L_1 \cap L_2$): Use the complement and union constructions together with the equivalence $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.
- difference ($L_1 \setminus L_2$): Use the equivalence $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$.
- concatenation ($L_1 \cdot L_2$): Make every final state q of \mathcal{A}_1 non-final and add ε -transitions from q to the initial state of \mathcal{A}_2 .
- iteration (L_1^*): Add a new initial and final state q_{init} and an ε -transition from q_{init} to the old initial state of \mathcal{A}_1 . Furthermore, make every final state q of \mathcal{A}_1 non-final and add ε -transitions from q to q_{init} .

 solution to exercise 5 on sheet 2 

§3 Regular expressions (pp. 24-26)

Syntax:

- \emptyset and ε are regular expressions over Σ .
- a is a regular expression over Σ for every $a \in \Sigma$.
- If re, re_1, re_2 are regular expressions over Σ , then $(re_1 + re_2)$, $(re_1 \cdot re_2)$, and re^* are also regular expressions over Σ .

Semantics:

- $L(\emptyset) = \emptyset$
- $L(\varepsilon) = \{\varepsilon\}$
- $L(a) = \{a\}$ for $a \in \Sigma$
- $L((re_1 + re_2)) = L(re_1) \cup L(re_2)$
- $L((re_1 \cdot re_2)) = L(re_1) \cdot L(re_2)$
- $L(re^*) = L(re)^*$

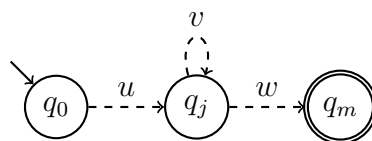
 solution to exercise 2 on sheet 3 

§4 Structural properties of regular languages (pp. 26-31)

2.4.1 Pumping lemma for regular languages

We have seen: if a language L is regular, then there is a DFA \mathcal{A} accepting it.

- L is finite if and only if there are no loops in \mathcal{A} (exception: sink state).
- Conversely, L is infinite if and only if there is at least one loop in \mathcal{A} .
- Loops are unbounded. Hence we can take *any* number of iterations.
- Thus, if \mathcal{A} recognizes a word $z \in \Sigma^*$ for which it goes once through a loop, it must also recognize a word z' with different loop iterations.



The pumping lemma for regular languages is then:

For every regular language $L \subseteq \Sigma^*$ there exists a number $n \in \mathbb{N}$, so that for all words $z \in L$ with $|z| \geq n$ there is a decomposition $z = uvw$ with $v \neq \varepsilon$ and $|uv| \leq n$ and for all $i \in \mathbb{N}$ it holds that $uw^i w \in L$.

Usually the pumping lemma is used to show that a language L is *not* regular. For this proof by contradiction it must be negated. If the following holds, then L is not regular.

For all numbers $n \in \mathbb{N}$ there exists a word $z \in L$ with $|z| \geq n$ and for all decompositions $z = uvw$ with $v \neq \varepsilon$ and $|uv| \leq n$ there exists an $i \in \mathbb{N}$ such that $uw^i w \notin L$.

 solution to exercise 3 on sheet 3 

 hints for exercise 1 on sheet 4 

2.4.2 Nerode relation

We skip this part in the interest of time.

Summary: For every regular language there is a DFA with *minimal number of states*. It is unique up to isomorphism (renaming of states).

§5 Decidability questions (pp. 32-34)

We skip this part in the interest of time.

Summary: All interesting problems are decidable for regular languages: *acceptance, emptiness, finiteness, equivalence, inclusion, intersection*

Chapter III – Context-free languages and push-down automata (pp. 37-69)

Nested bracket structures occur often in computer science. Respective languages are not regular, so we need another class – the context-free languages.

§1 Context-free grammars (pp. 38-43)

A *context-free grammar* is a 4-tuple $G = (N, T, P, S)$ where

- N is an alphabet of *non-terminal symbols* (capital letters),
- T is an alphabet of *terminal symbols* (small letters) with $N \cap T = \emptyset$,
- $S \in N$ is the *start symbol*, and
- $P \subseteq N \times (N \cup T)^*$ is a finite set of *productions* or *rules*.

 (2)

We define the *derivation relation* \vdash :

- $uAx \vdash uvx$ when there is a rule $A \rightarrow v$.
- \vdash^n denotes the application of \vdash n times .
- \vdash^* denotes the application of \vdash arbitrarily often.
- The language *generated* by G is $L(G) = \{w \in T^* \mid S \vdash^* w\}$.
- We call language L *context-free* if there is a grammar G with $L = L(G)$.

We have nondeterminism in two places:

- (a) the choice of the right-hand side of a production
- (b) the choice of the next non-terminal to replace

The latter choice can be eliminated in two ways:

- (a) Always replace the leftmost non-terminal (called *leftmost derivation*).
- (b) Consider a *derivation tree*.

If there is only one leftmost derivation or one derivation tree for each word $w \in L(G)$, we call G *unambiguous*, and *ambiguous* otherwise.

Note: It is *undecidable* whether a context-free grammar is ambiguous.

 hints for exercises 4–5 on sheet 4 