ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Prof. Dr. Andreas Podelski
Matthias Heizmann
Christian Schilling

June 18th & 24th, 2014

4. Lecture
Computer Science Theory

✎ solutions to exercises on sheet 4 ✐

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

§3 Pushdown automata (pp. 48-57)

It turns out that pushdown automata (PDAs) and context-free grammars (CFGs) are equal in power, i.e., a language accepted by some PDA is generated by some CFG and vice versa.
As usual in this course, this can be proven by giving an algorithm which, given a PDA (CFG), translates into a CFG (PDA) with the same language. Hence it can be done by a computer. We do one direction – the other one is much more complicated.

3.6 Theorem (CFG to PDA)
We want to construct a pushdown automaton $\mathcal{A}$ for a CFG $G = (N, T, P, S)$ whose language accepted with the empty stack is the same as the language generated by G, i.e., $L_\varepsilon(\mathcal{A}) = L(G)$.

Construction idea: We simulate the leftmost derivation in G. The stack is used to keep track of the word derived by the grammar so far. There are two cases:

(a) The topmost symbol is a terminal symbol, say, a. Then the PDA reads a from the input string and removes it from the stack.

Formally, we introduce transitions $(q, a) \xrightarrow{a} (q, \varepsilon)$ for each $a \in T$.

(b) The topmost symbol is a non-terminal symbol, say, A. Then the PDA reads nothing (ε-transition) from the input string and replaces A with the right-hand side u of some rule $A \to u$.

Formally, we introduce transitions $(q, A) \xrightarrow{\varepsilon} (q, u)$ for each $A \in N$ and $(A, u) \in P$.

An interesting fact: We do not need more than one state. Altogether, the components of our PDA $\mathcal{A} = (\Sigma, Q, \Gamma, \to, q, Z_0, F)$ are defined as follows:

$$\Sigma = T, Q = \{q\}, \Gamma = N \cup T, Z_0 = S, F = \emptyset, \text{ and } \to \text{ as described above}$$

We skip the formal proof that the PDA is equivalent to the CFG. ✎(1)

§4 Closure properties (pp. 58-59)

If we have two context-free languages L_1, L_2, then we can construct context-free grammars $G_1 = (N_1, T, P_1, S_1), G_2 = (N_2, T, P_2, S_2)$ for them. We may assume without loss of generality that $N_1 \cap N_2 = \emptyset$.
Context-free languages are closed under the following operations:

- union ($L_1 \cup L_2$): new grammar with additional rule $S \to S_1 \mid S_2$
- concatenation ($L_1 \cdot L_2$): new grammar with additional rule $S \to S_1 S_2$
- iteration (L_1^*): new grammar with additional rule $S \to \varepsilon \mid S_1 S$
- intersection with regular languages (skipped)

However, context-free languages are not closed under the following operations:

- intersection ($L_1 \cap L_2$): take $L_1 = \{a^m b^m c^n\}$ and $L_2 = \{a^m b^n c^n\}$, then $L_1 \cap L_2 = \{a^n b^n c^n\}$, which is not context-free (not shown)
- complement ($\overline{L_1}$): follows from $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

§5 Transformation in normal forms (pp. 60-62)

We skip this part in the interest of time.

Summary: There exist certain normal forms for context-free grammars. They are useful for applying certain algorithms.

§6 Deterministic context-free languages (pp. 63-67)

We skip this part in the interest of time.

Summary: We can define deterministic PDAs. They are weaker than nondeterministic PDAs.

§7 Questions of decidability (pp. 68-69)

We skip this part in the interest of time.

Summary: The membership, emptiness, and finiteness problems are decidable for context-free languages.
The intersection, equivalence, and inclusion problems as well as the problem whether a context-free grammar is unambiguous are undecidable.