ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Networked Systems Group (NSG)

HS 2018

Prof. L. Vanbever & A. El-Hassany, & M. Apostolaki.
based on Prof. R. Wattenhofer's material

Discrete Event Systems

Exercise Sheet 5

1 Context Free or Not?

For the following languages, determine whether they are context free or not. Prove your claims!

a) $L_1 = \{w\#x\#y\#z \mid w, x, y, z \in \{a, b\}^* \text{ and } |w| = |z|, |x| = |y|\}$

b) $L_2 = \{w\#x\#y\#z \mid w, x, y, z \in \{a, b\}^* \text{ and } |w| = |y|, |x| = |z|\}$

Remark: Languages L_1 is the same as in Exercice Sheet 4.

2 Counter Automaton

A push-down automaton is basically a finite automaton augmented by a stack. Consider a finite automaton that (instead of a stack) has an additional *counter* C, i.e., a register that can hold a single integer of arbitrary size. Initially, $C = 0$. We call such an automaton a *Counter Automaton* M. M can only increment or decrement the counter, and test it for 0. Since theoretically, all possible data can be coded into one single integer, a counter automaton has unbounded memory. Further, let $\mathcal{L}_{count}$ be the set of languages recognized by counter automata.

a) Let $\mathcal{L}_{reg}$ be the set of regular languages. Prove that $\mathcal{L}_{reg} \subseteq \mathcal{L}_{count}$.

b) Prove that the opposite is not true, that is, $\mathcal{L}_{count} \nsubseteq \mathcal{L}_{reg}$. Do so by giving a language which is in $\mathcal{L}_{count}$, but not in $\mathcal{L}_{reg}$. Characterize (with words) the kind of languages a counter automaton can recognize, but a finite automaton cannot.

c) Which automaton is stronger? A counter automaton or a push-down automaton? Explain your decision.

3 An Unsolvable Problem

It's the first day of your internship at the software firm Bug Inc., and your boss calls you to his office in order to explain your task for the next three months. He says that many clients complain that the programs of Bug Inc. often contain faulty loops that never terminate. In order to prevent such errors in future, you are asked to implement a program that may check whether a given program will halt on all possible inputs or not.

a) Try to find a proof that convinces your boss that this is not possible for general programs.

Hint: The proof works by contradiction. Assume a procedure `halt(P:Program):boolean` that takes a program `P` and decides whether `P` halts on all possible inputs or not. Now construct a program `X` that terminates if `halt(X)` is false and loops endlessly if `halt(X)` is true, which yields the desired contradiction.

b) Your boss still disagrees and proposes the following method: `halt(Y)` simply simulates the execution of program `Y`. If the program terminates it returns true, and if it loops it returns false. Where is the problem of this approach?

c) Your boss is finally convinced but argues that your proof is a very special case that hardly reflects reality. Are there assumptions under which it is always possible to check whether a program halts or not?