Prof. Javier Esparza
Philipp Czerner

Technical University of Munich
Chair for Foundations of Software Reliability

Automata and Formal Languages

Winter Term 2023/24 – Exercise Sheet 8

Exercise 8.1.

(a) Give a recursive algorithm for the following operation:

INPUT: States p and q of the master automaton.
OUTPUT: State r of the master automaton such that $L(r) = L(p) \cdot L(q)$.

Observe that the languages $L(p)$ and $L(q)$ can have different lengths. Try to reduce the problem for p, q to the problem for p^a, q.

(b) Give a recursive algorithm for the following operation:

INPUT: A state q of the master automaton.
OUTPUT: State r of the master automaton such that $L(r) = L(q)^R$

where R is the reverse operator.

(c) A *coding* over an alphabet Σ is a function $h\colon \Sigma \mapsto \Sigma$. A coding h can naturally be extended to a function over words, i.e., $h(\varepsilon) = \varepsilon$ and $h(w) = h(w_1)h(w_2)\cdots h(w_n)$ for every $w \in \Sigma^n$. Give an algorithm for the following operation:

INPUT: A state q of the master automaton and a coding h.
OUTPUT: State r of the master automaton such that $L(r) = \{h(w) : w \in L(q)\}$.

Can you make your algorithm more efficient when h is a permutation?

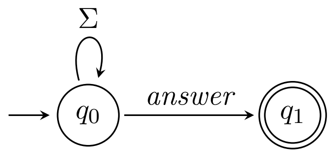
Exercise 8.2.

Let $\Sigma = \{\mathit{request}, \mathit{answer}, \mathit{working}, \mathit{idle}\}$.

(1) Build a regular expression and an automaton recognizing all words with the property P_1: for every occurrence of *request* there is a later occurrence of *answer*.

(2) Build an automaton recognizing all words with the property P_2: there is an occurrence of *answer* before which only *working* and *request* occur.

(3) Using automata theoretic constructions, prove that all words accepted by the automaton A below satisfy P_1, and give a regular expression for all words accepted by the automaton A that violate P_2.



Exercise 8.3.

Suppose there are n processes being executed concurrently. Each process has a critical section and a non critical section. At any time, at most one process should be in its critical section. In order to respect this mutual exclusion property, the processes communicate through a channel c. Channel c is a queue that can store up to m messages.

A process can send a message x to the channel with the instruction $c\,!\,x$. A process can also consume the first message of the channel with the instruction $c\,?\,x$. If the channel is full when executing $c\,!\,x$, then the process blocks and waits until it can send x. When a process executes $c\,?\,x$, it blocks and waits until the first message of the channel becomes x.

Consider the following algorithm. Process i declares its intention of entering its critical section by sending i to the channel, and then enters it when the first message of the channel becomes i:

```
process(i):
    while true do
        c ! i
        c ? i
        /* critical section */
        /* non critical section */
```

(a) Sketch an automaton that models a channel of size $m > 0$ where messages are drawn from some finite alphabet Σ.

(b) Model the above algorithm, with $n = 2$ and $m = 1$, as a network of automata. There should be three automata: one for the channel, one for `process`(0) and one for `process`(1).

(c) Construct the asynchronous product of the network obtained in (b).

(d) Use the automaton obtained in (c) to show that the above algorithm violates mutual exclusion, i.e. the two processes can be in their critical sections at the same time.

(e) Design an algorithm that makes use of a channel to achieve mutual exclusion for two processes ($n = 2$). You may choose m as you wish.

(f) Model your algorithm from (e) as a network of automata.

(g) Construct the asynchronous product of the network obtained in (f).

(h) Use the automaton obtained in (g) to show that your algorithm achieves mutual exclusion.