

Automata and Formal Languages – Homework 7

Due 10.12.2012.

Exercise 7.1

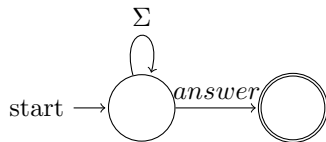
Let $\Sigma = \{request, answer, working, idle\}$.

- (a) Build an automaton recognizing all words with the property P_1 : after every *request* there is *answer* later on (not necessarily immediately).

Does it guarantee that every *request* has its own *answer*? More precisely, let us denote $w = w_1w_2 \cdots w_n$ and assume that there are k *requests*. Let us define $f : \{1, \dots, k\} \rightarrow \{1, \dots, m\}$ such that $w_{f(i)}$ is the i th *request* in w . Provided w satisfies P_1 , is there always an injective function $g : \{1, \dots, k\} \rightarrow \{1, \dots, m\}$ satisfying $w_{g(i)} = answer$ and $f(i) < g(i)$ for all $i \in \{1, \dots, k\}$?

If words were infinite and there were infinitely many *requests*, would P_1 guarantee that every *request* has its own *answer*? More precisely, let us denote $w = w_1w_2 \cdots$ and assume that there are infinitely many *requests*. Let us define $f : \mathbb{N} \rightarrow \mathbb{N}$ such that $w_{f(i)}$ is the i th *request* in w . Provided w satisfies P_1 , is there always an injective function $g : \mathbb{N} \rightarrow \mathbb{N}$ satisfying $w_{g(i)} = answer$ and $f(i) < g(i)$ for all $i \in \{1, \dots, k\}$?

- (b) Build an automaton recognizing all words with the property P_2 : there is an *answer* and before that there are only *workings* and *requests*.
- (c) Let \mathcal{A} be the following automaton



Using the intersection construction, prove that all accepting runs of \mathcal{A} satisfy P_1 and find all accepting runs violating P_2 .

Exercise 7.2

This exercise focuses on modelling and verification of mutual exclusion protocols. Let us consider having two agents, one having his internal variable *id* set to 0, the other has her variable *id* set to 1. They both run the following mutex program:

```
while(true)
  enter(id)
  critical-command
  leave(id)
loop-arbitrarily-many-times
  non-critical-command
```

The definitions of procedures `enter(int)` and `leave(int)` as well as global variables used and their initial values are specified below.

```
(a) int turn:=0
    proc enter(int i){
        while(turn=1-i) do
            skip
    }
    proc leave(int i){
        turn:=1-i
    }
}
```

Design an asynchronous network of automata capturing this behaviour.

Furthermore, build an automaton recognizing all runs reaching a configuration with both agents in the critical section. Using the intersection algorithm, prove that there are no such runs of this system, i.e. it is a *mutex* algorithm.

Do all infinite runs satisfy that if a process wants to enter the critical section then it eventually enters it?

```
(b) bool flag[0]:=false
    bool flag[1]:=false
    proc enter(int i){
        flag[i]:=true
        while(flag[1-i]) do
            skip
    }
    proc leave(int i){
        flag[i]:=false
    }
}
```

Design an asynchronous network of automata capturing this behaviour.

Can a deadlock occur?

(c) Peterson's algorithm combines both approaches:

```
int turn:=0
bool flag[0]:=false
bool flag[1]:=false
proc enter(int i){
    turn:=1-i
    flag[i]:=true
    while(flag[1-i] & turn=1-i)
        skip
}
proc leave(int i){
    flag[i]:=false
}
}
```

Can a deadlock occur?

What kind of starving can occur?