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

HS 2016

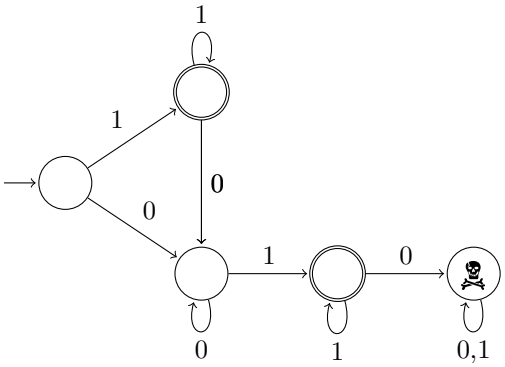
Prof. L. Vanbever & A. El-Hassany, & M. Apostolaki.
based on Prof. R. Wattenhoffer's material

# Discrete Event Systems

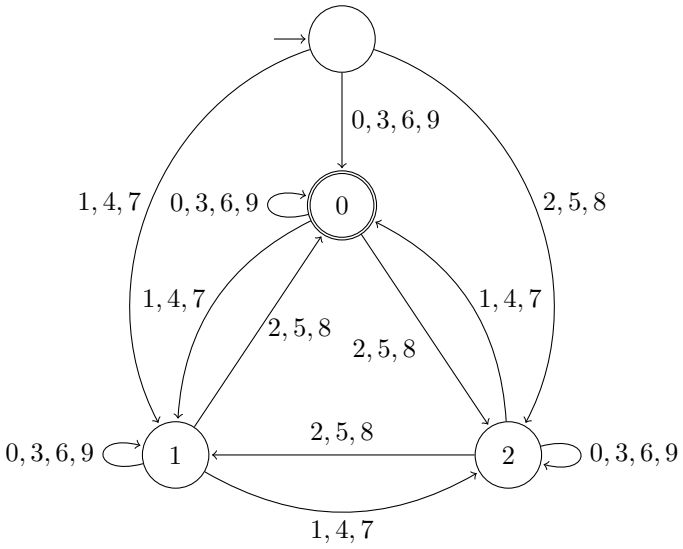
**Solution to Exercise Sheet 1**

## 1 Finite Automata

**a)** The following machine accepts all strings of the form 1*0*11*.

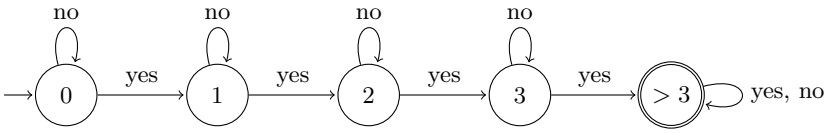



**b)** We use the fact that a number is divisible by three if and only if its cross sum (in German: *Quersumme*) is divisible by three. Our automaton has three states that denote the value of the cross sum computed so far modulo 3.

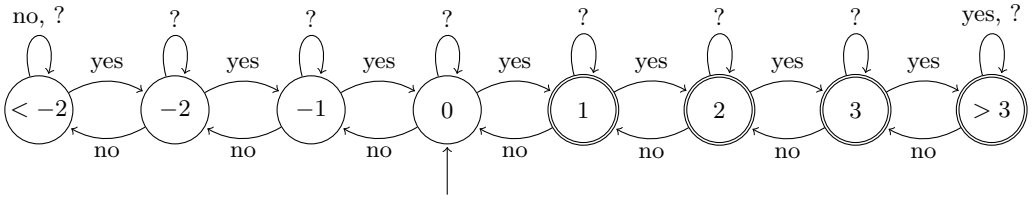


## 2 "Mais im Bundeshuus"

**a)** The automaton accepts if and only if at least four members of the Swiss Federal Council vote in favor of the proposition. The following machine remembers the number of approving votes.

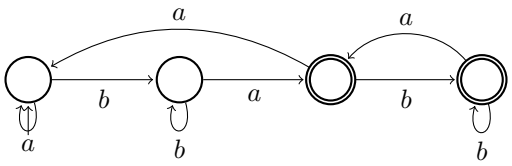



**b)** We consider the alphabet {yes, no, ?}, where "?" denotes abstention of voting.




## 3 Filter for an Input Stream [exam problem]

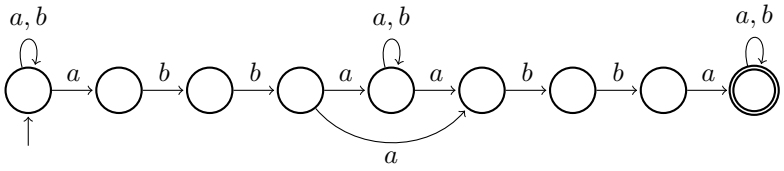
The following figure gives an example with four states.



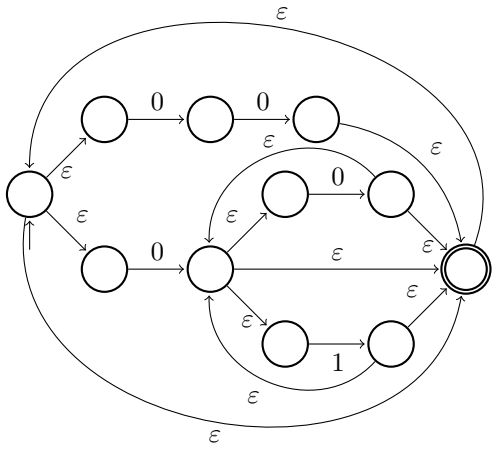

The main idea here is to use two different accepting states after having read $a$: one for zero $b$'s and one for more than zero $b$'s.

## 4 Nondeterministic Finite Automata

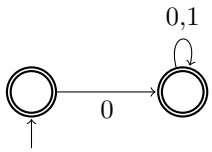
**a)** The following automaton also accepts strings containing $a\,b\,b\,a\,b\,b\,a$ as a substring.



**b)** The following automaton is obtained using the transformations presented in the lecture.

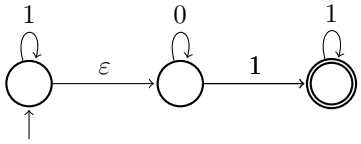



This automaton, however, is not minimal. If you take a closer look at the regular expression $(00 \cup (0(0 \cup 1)^*))^*$, you will see that it can be simplified to $(0(0 \cup 1)^*)^*$ and this in turn to $0(0 \cup 1)^*$. A minimal automaton for this regular expression is given by




**Note:** The absence of a transition with label 1 in the left state means that if a 1 is read, the automaton goes into a non-accepting state and stays there for the rest of the execution. NFAs may be under-determined in this notion, but FAs must provide a transition for every symbol of the alphabet and potentially a crash state to "capture" non-accepting executions.

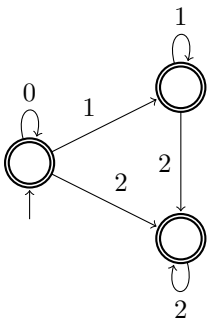
**c)** Three states are enough if we can use NFAs. With FAs, we needed at least four states.



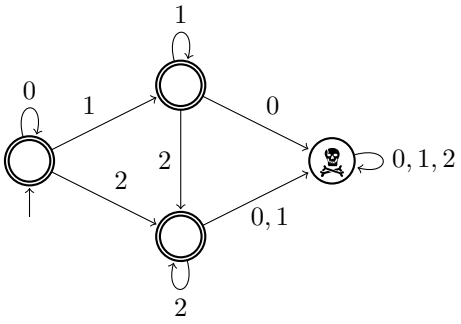

**d)** A deterministic machine for which every state is an accepting state, accepts *every* string of the corresponding alphabet. However, this does not hold for a nondeterministic automaton, namely if it is under-determined.

# 5 De-randomization

**a)** The automaton accepts strings adhering to the regular expression $0^*1^*2^*$. Without $\varepsilon$-transitions we get the following automaton.

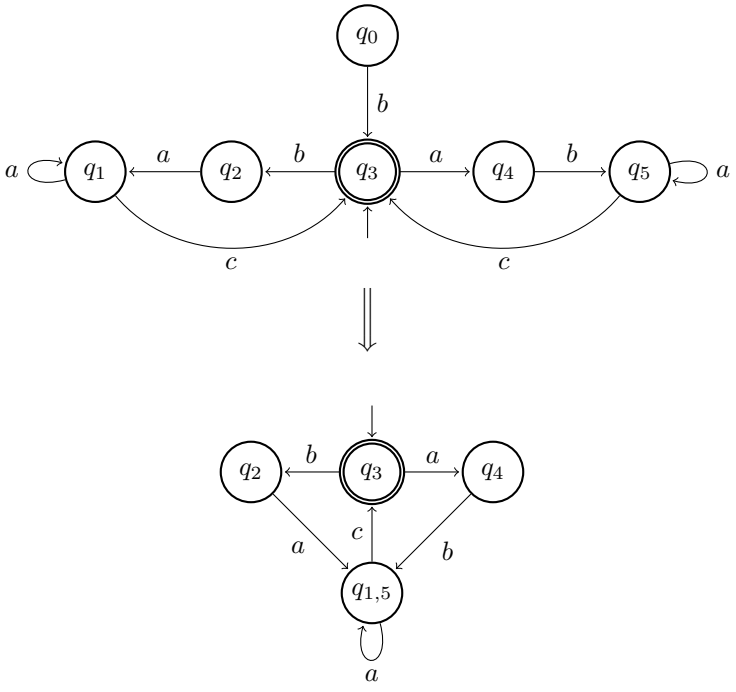



**b)** The deterministic automaton can be found by applying the power set construction presented in the lecture followed by the state minimization algorithm. However, it is obvious that the automaton shown below does the job.




# 6 States Minimization

State $q_0$ can be omitted as it is not reachable. Moreover, states $q_1$ and $q_5$ can be merged, as there is no input sequence which will show a difference between these two states. A regular expression of the language is $((ba \cup ab)a^*c)^*$.



# 7 "Regular" Operations in UNIX

In UNIX, the special symbol "$" stands for the end of a line. We have:

```
egrep '(password|passwort)(a|e|i|o|u|A|E|I|O|U)*$' <file>
```