Introduction to Logic and Computer Science
Foundation of Computer Science
Logic in Computer Science

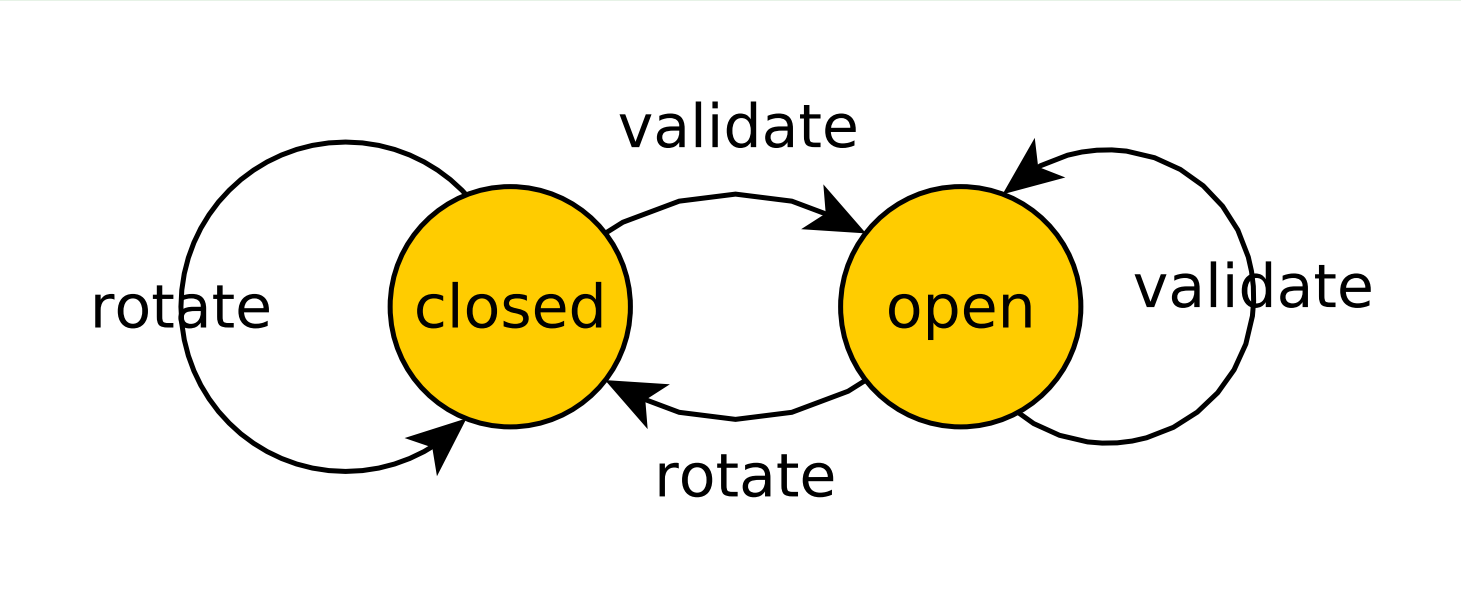
Lecture #11: finite automaton

Tamás Mihálydeák, László Aszalós

This work was supported by the construction EFOP-3.4.3-16-2016-00021.
The project was supported by the European Union, co-financed by the European Social Fund.

Entrace gate

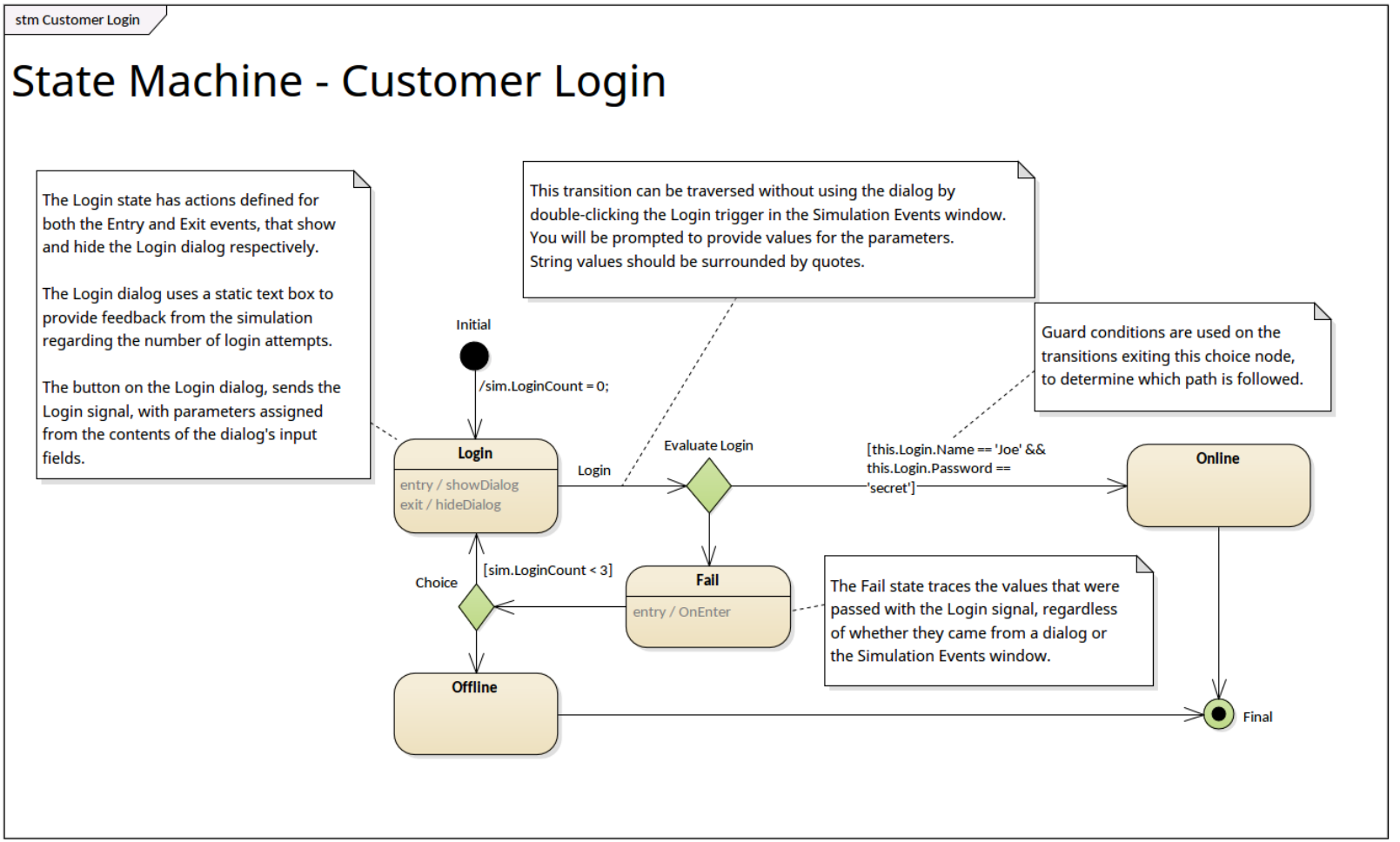
Example - automaton



If one validates his ticket at the gate, it opens and he can then pass through it; which in turn closes the gate. With an open gate, we would validate the ticket in vain, and we could try to rotate the closed gate in vain, the gate status would not change.

UML state diagram

Example - software planning



Finite automaton

- The model in the previous example is the finite automaton.
- This is the *simplest* automaton:
  - it is only able to read,
  - it has no memory
- As here we will only use the finite automaton, we omit the word *finite*

Finite automaton

Definition

A finite automaton is an ordered 5-tuple: $A = \langle Q, \Sigma, \delta, q_0, F \rangle$, where

- Q: the finite set (internal) states,
- Σ: the finite (external) alphabet (the alphabet of the string to be analysed),
- $\delta : Q \times \Sigma \to Q$: transition function,
- $q_0 \in Q$: initial state
- $F \subseteq Q$: the set of accepting state(s)

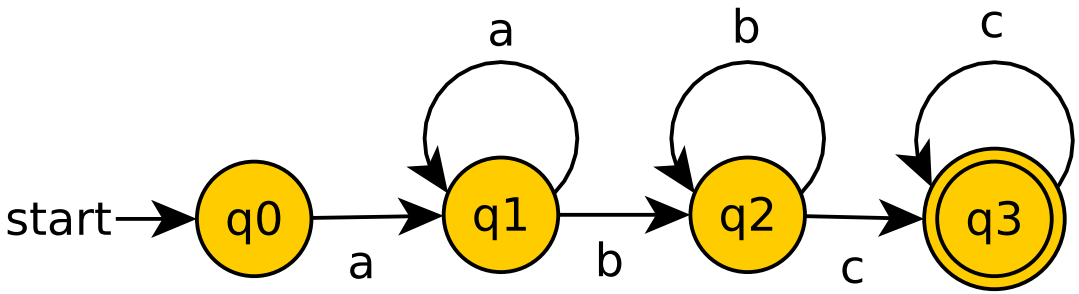
Remark

- A transition function determines the new state of the automaton after reading a character.
- An automaton accepts a sequence of characters (a string), if
  - it reads the whole string and
  - after reading the last character it is in accepting state.

Examples

$A = \langle Q, \Sigma, \delta, q_0, F \rangle$, where

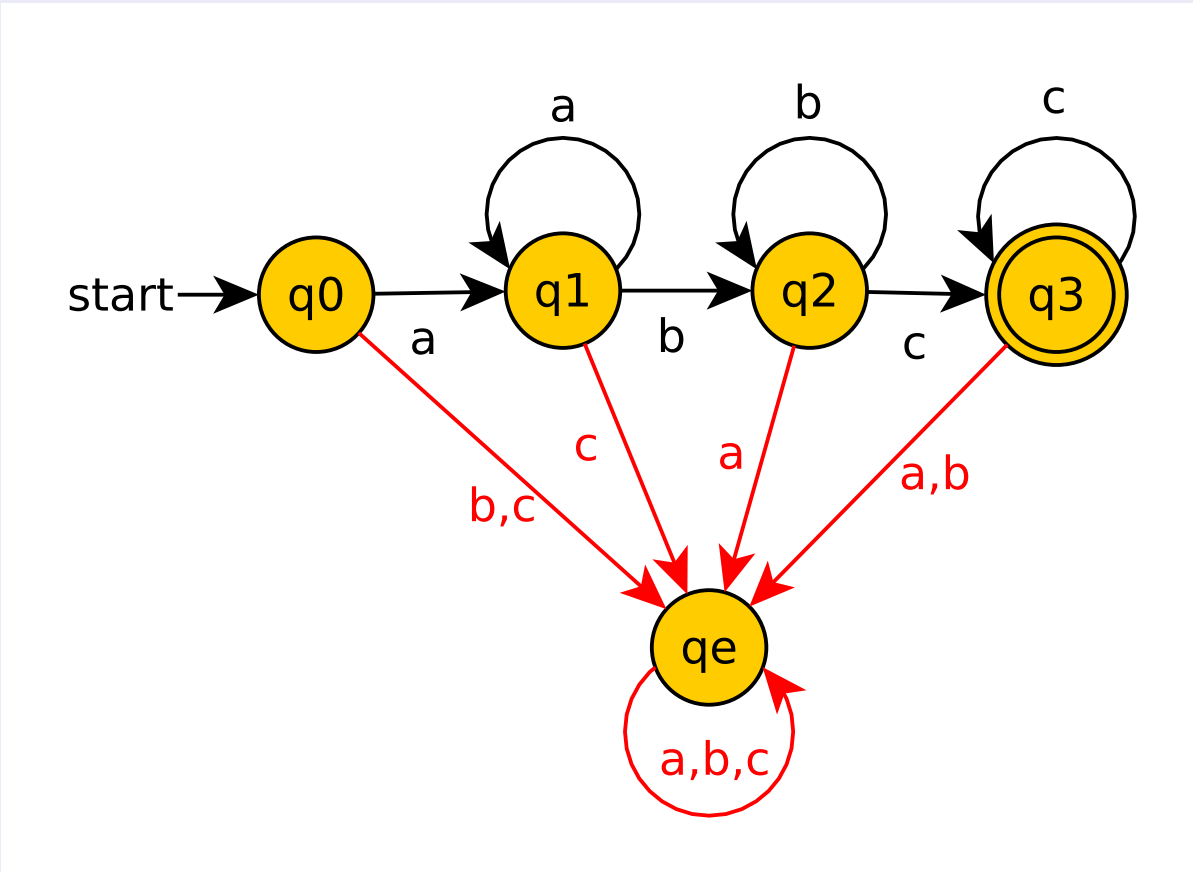
- $Q = \{q_0, q_1, q_2, q_3, q_e\}$ is the finite set of (internal) states;
- $\Sigma = \{a, b, c\}$ is the finite (external) alphabet;
- $\delta(q_0, a) = q_1$, $\delta(q_1, a) = q_1$, $\delta(q_1, b) = q_2$,
- $\delta(q_2, b) = q_2$, $\delta(q_2, c) = q_3$, $\delta(q_3, c) = q_3$
- and the value of δ is q_e otherwise (the transition function);
- $q_0 \in Q$ is the initial state;
- $F = \{q_3\}$ i.e. q_3 is the accepting state.



Defincient automaton

Remark

According to our definition the function δ is total and not partial. In order to make the figure simpler the missing arrows point to a corrupted state.



Notation

Internal states are represented by circles, the transition function is denoted by arrows and the characters of the internal alphabet. The initial state is the state with an unlabelled arrow pointing at it. The final state(s) are the double walled circles.

Accepting automata

Definition

$A = \langle Q, \Sigma, \delta, q_0, F \rangle$ is an automaton, and $w = a_1 a_2 \dots a_n \in \Sigma^\star$ is a word. The automaton accepts the word w if

$$\delta(\dots(\delta(\delta(q_0, a_1), a_2)\dots), a_n) \in F.$$

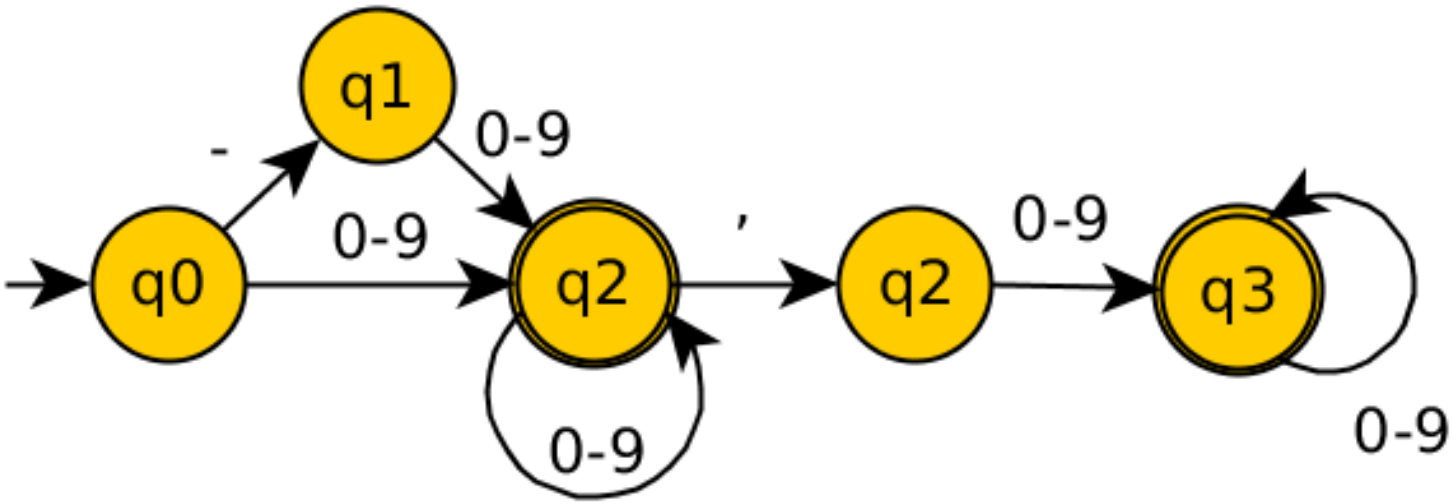
Definition

Let $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ be an automaton. If $L = \{w \in \Sigma^\star | A \text{ accepts } w\}$, then A accepts (recognises) the language L. We denote it as: $L = L_A$.

Theorem

Finite automata accept regular languages, i.e. if L is a regular language, then there is an automaton A such that $L = L_A$

Example



This automaton accepts the strings `5`; `-92`; `754`; `-15,7` and `79,642`.
It does not accept the strings `--4`; `2-7`; `-,5` and `12,`
The recognised language is the set of well-formed natural numbers and decimals.

Deterministic automaton in Python

```
dfa = {0:{'a':0, 'b':1},
       1:{'a':2, 'b':0},
       2:{'a':1, 'b':2}}

def accepts(transitions,initial,accepting,s):
    state = initial
    for c in s:
        state = transitions[state][c]
    return state in accepting

>>> accepts(dfa,0,{0},'babbbab')
True
>>> accepts(dfa,0,{0},'babbbabb')
False
```

Problem

Remark

- Within the framework defined so far, there are some very difficult tasks to solve, such as *Consider the language of reversing the words of a language recognised by an automaton. Could it be recognised by an automaton?* This question and more can be answered very easily if we use a more general concept of automaton.
- So far, each step and its outcome has been clearly identifiable. Non-deterministic execution at several points allows a choice for the next possible steps. The more general automaton is called a non-deterministic automaton.
- This kind of automaton differs from the conventional automaton only in respect of the transition function: we can go in several directions, or it is possible to make a state transition without reading the input.

Definition

A non-deterministic automaton $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ is an ordered 5-tuple, where

- Q the finite set (internal) states,
- Σ the finite (external) alphabet,
- $\delta : Q \times \Sigma_\varepsilon \to 2^Q$ transition function,
- $q_0 \in Q$ initial state, and
- $F \subseteq Q$ the set of accepting states.

Remark

In the definition

- 2^Q denotes the set of subsets of Q (its power-set)
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, as it can changes its state without any input.

Words accepted by a non-deterministic automaton

Definition

$A = \langle Q, \Sigma, \delta, q_0, F \rangle$ is a non-deterministic automaton and
$w = a_1 a_2 \ldots a_n \in \Sigma^\star$ is a word. A accepts the word w if there exist

- a number $m \geq n$
- a sequence of states $r_0, r_1, \ldots r_m$ ($r_i \in Q$), and
- a sequence of *letters* $b_0, b_1, \ldots b_m$ ($b_i \in \Sigma \cup \{\varepsilon\}$) such that

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, y_{i+1})$ $0 \leq i < m$, and
3. $r_m \in F$

Non-deterministic automaton

Remark

We can define the language accepted by a non-deterministic automaton in a similar way.

Theorem

For any non-deterministic automaton A there exists (in a constructive way) a deterministic automaton A' such that $L_A = L_{A'}$

Non-deterministic automaton in Python (just for experts)

Source: https://swizec.com/blog/strangest-line-of-python-you-have-ever-seen/swizec/3012

```
from optparse import OptionParser
from collections import defaultdict
def run(inputWord):
    a, states = open("a.txt") ,defaultdict(list)
    state, final= a.readline().split()[1:],a.readline().split()[1:]
    [states[(i.split()[0], i.split()[1])].append(i.split()[3]) for i in a]
    for letter in inputWord:
        state = [st for s in state for st in states[(s,letter)]]
    return any(i in state for i in final)

print [(b,"YES") if run(b) else (b,"NO") for b in OptionParser().parse_args()[1]]

# a.txt:
# init: q0
# final: q2 q3
# q0 a -> q1
# q0 b -> q3
# q1 a -> q1
# q1 b -> q2
# q2 a -> q2
# q2 b -> q2
```