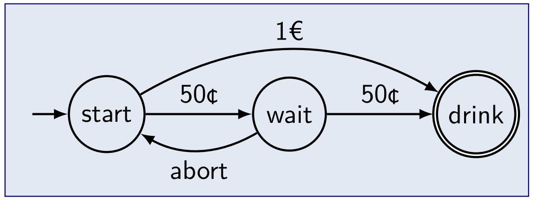
Finite State Machines and Regular Languages

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.jku.at

Motivation

- Behavior of a vending machine that delivers a drink:



- Infinitely many successful interaction sequences:

  1€
  50¢ 50¢
  50¢ abort 1€
  50¢ abort 50¢ abort 1€
  50¢ abort 50¢ abort 50¢ 50¢
  ...

- A finite description of these sequences:

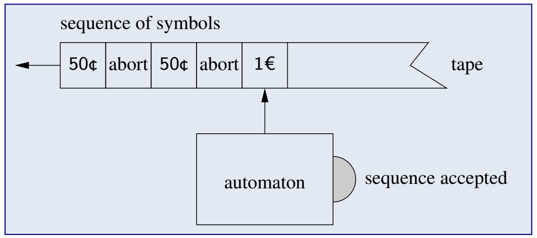
  $(50¢\ \text{abort})^*(1€ + 50¢\ 50¢)$

We will investigate automata and the associated interaction sequences.

1. Deterministic Automata

2. Nondeterministic Automata
3. Determinization of Automata
4. Minimization of Automata
5. Regular Languages
6. Regular Expressions to Automata
7. Automata to Regular Expressions
8. The Expressiveness of Regular Languages

Automaton Model



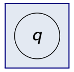
- Automaton is always in one of a finite set of states.
  - Automaton starts execution in a fixed start state.
- Input tape with a finite sequence of symbols (a word).
  - Tape is only read by the automaton.
- Execution proceeds in a sequence of state transitions.
  - Automata reads one symbol and moves tape head to next symbol.
  - The symbol read and the current state determine the next state.
- When the whole word is read, the automaton terminates.
  - The automaton signals whether it is in a final state.

If the automaton terminates in final state, the input word is *accepted*.

Deterministic Automata

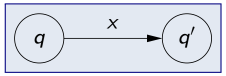
A deterministic finite-state machine (DFSM) $M = (Q, \Sigma, \delta, q_0, F)$:

- The state set Q, a finite set of states.

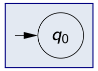


- An input alphabet Σ, a finite set of input symbols.

- The transition function $\delta : Q \times \Sigma \to Q$.
  - $\delta(q, x) = q'$ ... M reads in state q symbol x and goes to state q'.



- The start state $q_0 \in Q$.



- A set of final states (accepting states) $F \subseteq Q$.

Definition of an Automaton

$M = (Q, \Sigma, \delta, q_0, F)$
$Q = \{\ldots, q_0, \ldots\}$
$\Sigma = \{\ldots\}$
$F = \{\ldots\}$

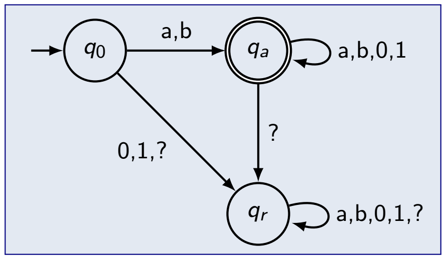
δ	$\ldots$	x	$\ldots$
$\vdots$			
q		$\delta(q, x)$	
$\vdots$			

The transition function δ is typically defined by a table.

Example

$M = (Q, \Sigma, \delta, q_0, F)$
$Q = \{q_0, q_a, q_r\}$
$\Sigma = \{\text{a}, \text{b}, 0, 1, ?\}$
$F = \{q_a\}$

δ	a	b	0	1	?
q_0	q_a	q_a	q_r	q_r	q_r
q_a	q_a	q_a	q_a	q_a	q_r
q_r	q_r	q_r	q_r	q_r	q_r



Accepts words of letters and digits starting with a letter.

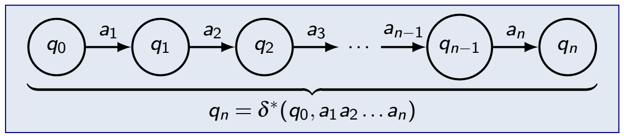
The Extended Transition Function

- The extended transition function $\delta^* : Q \times \Sigma^* \to Q$ of M:

$$\delta^*(q, \varepsilon) := q$$
$$\delta^*(q, wa) := \delta(\delta^*(q, w), a)$$

  - Σ^* is the set of all words over Σ.
  - $\varepsilon \in \Sigma^*$ is the empty word.
  - $a \in \Sigma$ is an input symbol, $w \in \Sigma^*$ a word.
- $q_n = \delta^*(q_0, a_1 a_2 \ldots a_n)$:

$q_1 = \delta(q_0, a_1)$
$q_2 = \delta(q_1, a_2)$
$\ldots$
$q_n = \delta(q_{n-1}, a_n)$



The generalization of the transition function δ to an input word.

The Language of an Automaton

The automata language $L(M) \subseteq \Sigma^*$ of M:

$$L(M) := \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

- The set of all words that drive M from its start state to a final state.

Word w is *accepted* by M, if $w \in L(M)$.

Example

```
e0, e1 ← true, true
while input stream is not empty do
    read input
    case input of
        0: e0 ← ¬e0
        1: e1 ← ¬e1
        default: return false
    end case
end while
return e0 ∧ e1
```

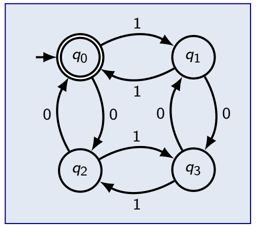
$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

$F = \{q_0\}$

δ	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

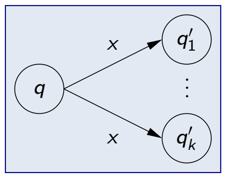


$L(M)$ is the set of bit strings with an even number of '0' and '1'.

Nondeterministic Automata

A nondeterministic finite-state machine (NFSM) $M = (Q, \Sigma, \delta, S, F)$:

- The state set Q, a finite set of states.
- An input alphabet Σ, a finite set of input symbols.
- The transition function $\delta : Q \times \Sigma \to P(Q)$.
  - $P(Q)$ ...the set of all subsets (the powerset) of Q.
  - $\delta(q, x) = \{q'_1, \ldots, q'_k\}$ ... M reads in state q symbol x and goes to one of the states $q'_1, \ldots, q'_k$.



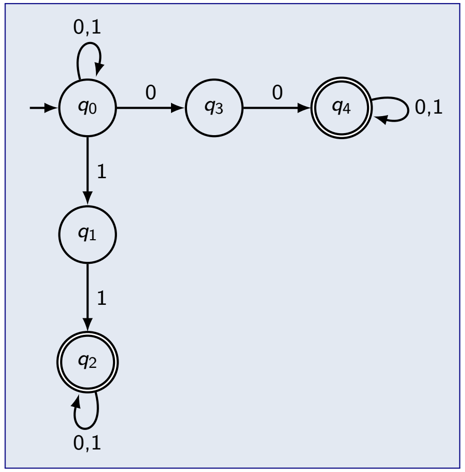
- The start state $q_0 \in Q$.
- The set of start states $S \subseteq Q$.
- A set of final states (accepting states) $F \subseteq Q$.

A DFSM is essentially just a special case of a NFSM.

Example

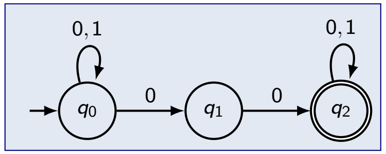
$M = (Q, \Sigma, \delta, S, F)$
$Q = \{q_0, q_1, q_2, q_3, q_4\}$
$\Sigma = \{0, 1\}$
$S = \{q_0\}$
$F = \{q_2, q_4\}$

δ	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$



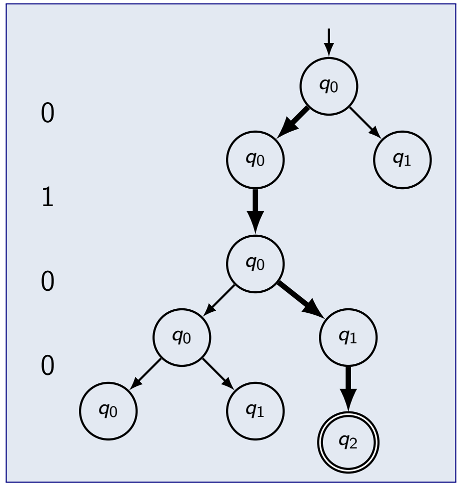
Accepts bit strings that contain ’00’ or ’11’.

Interpretation of Nondeterminism



- Automaton splits itself into multiple copies that investigate all paths in parallel.
- Input is accepted, if at least one copy reaches final state.

A certain form of parallel search.



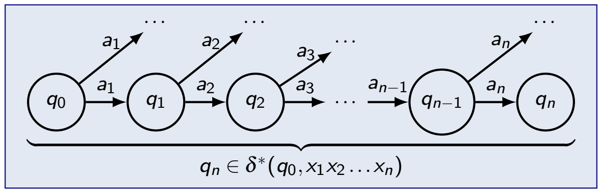
The Language of a Nondeterministic Automaton

- The extended transition function $\delta^* : Q \times \Sigma^* \to Q$ of M:

$$\delta^*(q, \varepsilon) := \{q\}$$
$$\delta^*(q, wa) := \{q'' \mid \exists q' \in \delta^*(q, w) : q'' \in \delta(q', a)\}$$

- $q_n \in \delta^*(q_0, a_1 a_2 \ldots a_n)$:

$q_1 \in \delta(q_0, a_1)$
$q_2 \in \delta(q_1, a_2)$
$\ldots$
$q_n \in \delta(q_{n-1}, a_n)$

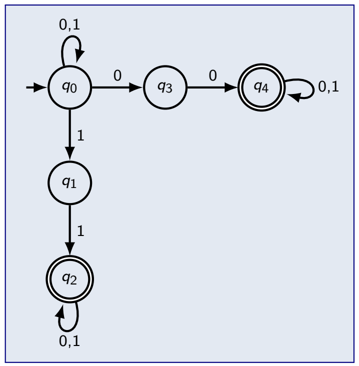


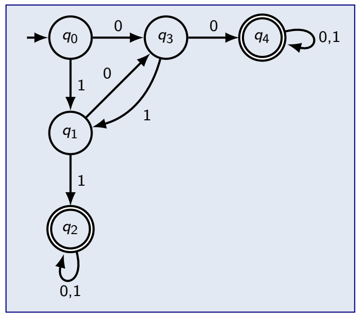
- The automata language $L(M) \subseteq \Sigma^*$ of M:

$$L(M) := \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

Example

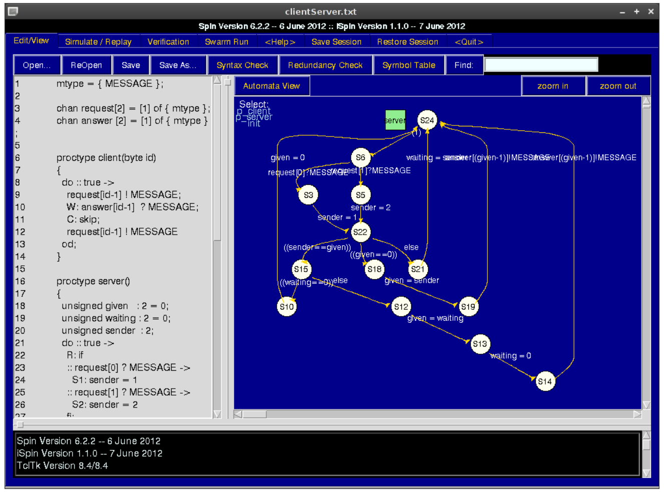
A NFSM may be easier to construct than a DFSM.





The language of both automata is the set of all bit strings that contain ’00’ or ’11’, but this is much easier to see in the NFSM.

Application of Nondeterministic Automata



Modeling and verification of concurrent systems.

Determinization of Automata

Every language accepted by some DFSM is also accepted by some NFSM, but does also the converse hold?

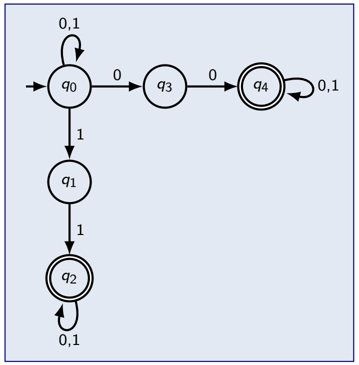
Theorem (Subset Construction): Let $M = (Q, \Sigma, \delta, S, F)$ be a NFSM. Then $L(M') = L(M)$ for the DFSM $M' = (Q', \Sigma, \delta', q'_0, F')$ defined as follows:

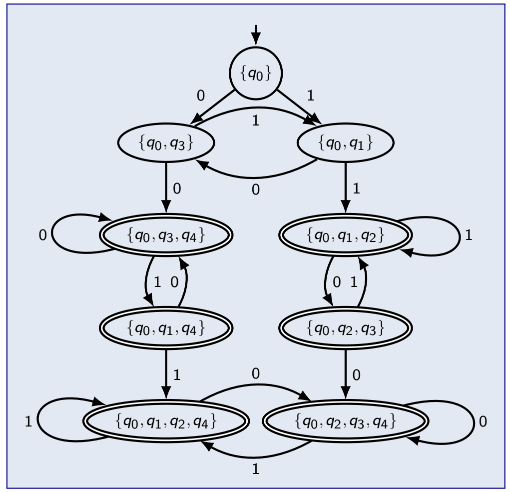
$$Q' = P(Q)$$
$$\delta'(q', a) = \bigcup_{q \in q'} \delta(q, a)$$
$$q'_0 = S$$
$$F' = \{q' \in Q' \mid q' \cap F \neq \emptyset\}$$

- States of M' are sets of states of M.
- Successor of state q' of M' is the set of successors in M of all states in q'.
- The start state of M' is the set of all start states of M.
- End states of M' are all those states that contain end states of M.

NFSMs and DFSMs accept the same set of languages.

Example





The DFSM accepts the same language but is not necessarily minimal.

Correctness Proof

Proof that $L(M) = L(M')$, i.e., $w \in L(M) \Leftrightarrow w \in L(M')$.

$\Rightarrow$ Assume $w = a_1 a_2 \dots a_n \in L(M)$.

- Then there exists a sequence of states $q_0, q_1, q_2, \dots, q_n$ with $q_0 \in S$ and $q_n \in F$ and $q_1 \in \delta(q_0, a_1), q_2 \in \delta(q_1, a_2), \dots, q_n \in \delta(q_{n-1}, a_n)$.
- Take the sequence of state sets $Q_0, Q_1, Q_2, \dots, Q_n$ with $Q_0 = S$, $Q_1 = \delta'(Q_0, a_1), Q_2 = \delta'(Q_1, a_2), \dots, Q_n = \delta'(Q_{n-1}, a_n)$.
- We know $q_0 \in S = Q_0$; according to the definition of δ', we thus have $q_1 \in \delta(q_0, a_1) \subseteq \delta'(Q_0, a_1) = Q_1$; we thus have $q_2 \in \delta(q_1, a_2) \subseteq \delta'(Q_1, a_2) = Q_2$; ...; we thus have $q_n \in \delta(q_{n-1}, a_n) \subseteq \delta'(Q_{n-1}, a_n) = Q_n$.
- Since $q_n \in Q_n$ and $q_n \in F$, we have $Q_n \cap F \neq \emptyset$ and thus $w \in L(M')$.

$\Leftarrow$ Analogous (see lecture notes).

Minimization of Deterministic Automata

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFSM.

- Binary relation $\sim_k$ on Q:

$$q_1 \sim_0 q_2 :\Leftrightarrow (q_1 \in F \Leftrightarrow q_2 \in F)$$
$$q_1 \sim_{k+1} q_2 :\Leftrightarrow \forall a \in \Sigma : \delta(q_1, a) \sim_k \delta(q_2, a)$$

  - $q_1 \sim_k q_2$: starting with both states, the same words of length k are accepted.

- Bisimulation relation $\sim$:

$$q_1 \sim q_2 \Leftrightarrow \forall k \in \mathbb{N} : q_1 \sim_k q_2$$

  - $q_1 \sim q_2$: starting with both states, the same words are accepted.

If $q_1 \sim q_2$, then q_1 and q_2 are *state equivalent*.

Minimization of Deterministic Automata

function PARTITION$(Q, \Sigma, \delta, q_0, F)$
  $P \leftarrow \{F, Q \backslash F\}$
  repeat
    $S \leftarrow P$
    $P \leftarrow \emptyset$
    for $p \in S$ **do**
      $P \leftarrow P \cup \{[s]^S_p \mid s \in p\}$
    end for
  until $P = S$
  return P
end function

function MINIMIZE$(Q, \Sigma, \delta, q_0, F)$
  $Q \leftarrow \{q \in Q \mid \exists w \in \Sigma^* : \delta^*(q_0, w) = q\}$
  $Q' \leftarrow$ PARTITION$(Q, \Sigma, \delta, q_0, F)$
  for $q' \in Q', a \in \Sigma$ **do**
    set $\delta'(q', a)$ to that partition q'' of Q'
      where $\forall q \in q' : \delta(q, a) \in q''$
  end for
  let q'_0 be that partition of Q' where $q_0 \in q'_0$
  $F' \leftarrow \{q \in Q' : q \cap F \neq \emptyset\}$
  return $(Q', \Sigma, \delta', q'_0, F')$
end function

State partition $[s]^S_p := \{\ t \in p \mid \forall a \in \Sigma, q \in S : \delta(t, a) \in q \Leftrightarrow \delta(s, a) \in q\}$

- S a set of state sets, p a state set in S.
- All states in p that lead for every transition to the same set in S as state s.

Sequence of partitionings $P_0, P_1, \dots, P_n$ of Q such that P_k consists of those partitions whose elements are related by $\sim_k$ and $\sim_n = \sim$.

Example

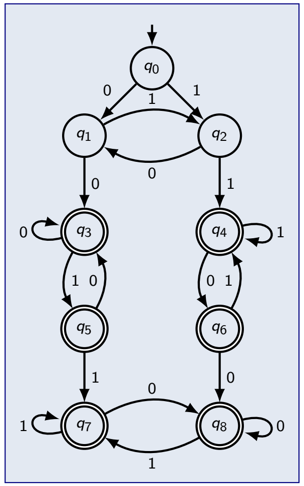
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$

- $P_0 := \{p_0, p_3\}$

  $p_0 := \{q_0, q_1, q_2\} = Q \backslash F$
  $p_3 := \{q_3, q_4, q_5, q_6, q_7, q_8\} = F$

  - All transitions from p_3 lead to p_3.
    - p_3 need not be partitioned further.
  - p_0 has to be partitioned further.
    - $\delta(q_0, 0) = q_1 \in p_0$,
      $\delta(q_1, 0) = q_3 \in p_3$.
    - $\delta(q_0, 1) = q_2 \in p_0$,
      $\delta(q_2, 1) = q_4 \in p_3$.
    - $\delta(q_1, 0) = q_3 \in p_3$,
      $\delta(q_2, 0) = q_2 \in q_0$.

    q_0, q_1, q_2 must be separated.



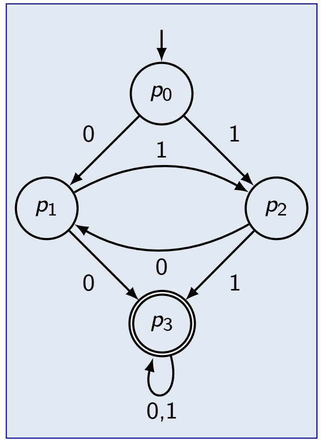
Example

- $P_1 := \{p_0, p_1, p_2, p_3\}$

  $p_0 := \{q_0\}$, $p_1 := \{q_1\}$, $p_2 := \{q_2\}$,
  $p_3 := \{q_3, \ldots, q_8\}$

- $P_n := P_1$.
  - No further split is possible.

Minimal DFSM whose language is the set of all bit strings that contain '00' or '11'.



Regular Expressions

- The set of regular expressions $Reg(\Sigma)$ over $\Sigma = \{a_1, \ldots, a_n\}$:
  - $\emptyset \in Reg(\Sigma)$ and $\varepsilon \in Reg(\Sigma)$.
  - $a_1 \in Reg(\Sigma), \ldots, a_n \in Reg(\Sigma)$.
  - If $r_1 \in Reg(\Sigma)$, then $(r_1 \cdot r_2) \in Reg(\Sigma)$ and $(r_1 + r_2) \in Reg(\Sigma)$.
  - If $r \in Reg(\Sigma)$, then $(r^*) \in Reg(\Sigma)$.

  $r ::= \emptyset \mid \varepsilon \mid a_1 \mid \ldots \mid a_n \mid (r \cdot r) \mid (r + r) \mid (r^*)$

- Syntactic Conventions:
  - * binds stronger than $\cdot$ which binds stronger than $+$.
  - $\cdot$ is often omitted.

$$(a + (b \cdot (c^*))) \equiv$$
$$a + b \cdot c^* \equiv$$
$$a + bc^*$$

Regular expressions denote languages (sets of words).

The Shell Command grep

```
NAME
       grep, egrep, fgrep, rgrep - print lines matching a pattern

SYNOPSIS
       grep [options] PATTERN [FILE...]
...
REGULAR EXPRESSIONS
       A  regular  expression  is  a  pattern that describes a set of strings.
       ...
       The  fundamental building blocks are the regular expressions that match
       a single character.  Most characters, including all letters and digits,
       are  regular expressions that match themselves.
       ...
       A regular expression may be followed by the repetition operator *;
       the preceding item will be matched zero or more times.
       ...
       Two regular expressions may  be  concatenated;  the  resulting  regular
       expression  matches  any  string formed by concatenating two substrings
       that respectively match the concatenated subexpressions.
       ...
       Two regular expressions may be joined by  the  infix  operator  |;  the
       resulting  expression matches any string matching either subexpression.
```


Regular Languages

- Regular expression language $L(r) \subseteq \Sigma^*$:
  - $L(\emptyset) := \emptyset$.
  - $L(\varepsilon) := \{\varepsilon\}$.
  - $L(a) := \{a\}$.
  - $L(r_1 \cdot r_2) := L(r_1) \circ L(r_2)$.
  - $L(r_1 + r_2) := L(r_1) \cup L(r_2)$.
  - $L(r^*) := L(r)^*$.

  Concatenation: $L_1 \circ L_2 := \{w_1 \cdot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$

  Finite Closure: $L^* := \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots$

$$L^0 := \{\varepsilon\}$$
$$L^{i+1} := L \circ L^i$$

- Syntactic Simplification: $r+s+t$, $r \cdot s \cdot t$

$$L((r+s)+t) = L(r+(s+t))$$
$$L((r \cdot s) \cdot t) = L((r \cdot s) \cdot t)$$

A language L is *regular*, if there is a regular expression r with $L(r) = L$.

Examples

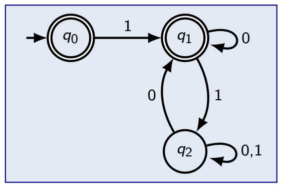
- Language of identifiers

  $(a+b)(a+b+0+1)^*$
- Language of bit strings containing '00' or '11'

  $(0+1)^*(00+11)(0+1)^*$
- Language of vending machine

  (50¢ abort)*(1€ + 50¢ 50¢)
- Regular language

  $\varepsilon + 1(0+1(0+1)^*0)^*$



Is the language of every automaton regular? Is every regular language the language of some automaton?

Regular Expressions and Automata

1. For every regular expression r over Σ, there exists an automaton M with input alphabet Σ such that $L(M) = L(r)$.

   Proof by construction of automaton M from arbitrary regular expression r.
2. For every automaton M with input alphabet Σ, there exists a regular expression r over Σ such that $L(r) = L(M)$.

   Proof by construction of regular expression r from arbitrary automaton M.

Automata and regular expressions describe the same sets of languages.

Generation of Lexical Analyzers

Various tools for the generation of lexical analyzers (lexers, scanners).

- Input: a regular expression.

```
IDENT: LETTER (LETTER | DIGIT)* ;
```

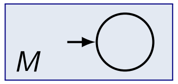
- Output: an automaton (implemented by a program).

```
public final void mIDENT(...) throws ... {
  ...
  mLETTER();
  _loop271: do {
    switch ( LA(1)) {
      case 'a':  ... case 'z': { mLETTER(); break; }
      case '0':  ... case '9': { mDIGIT(); break; }
      default: { break _loop271; }
    }
  } while (true);
  ...
}
```

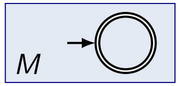

Base Cases

We construct from r a NFSM M' with a single start state and arbitrarily many accepting states (one of which may be the start state).

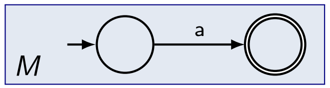
- **Case $\mathbf{r} = \emptyset$:**



- **Case $\mathbf{r} = \varepsilon$:**

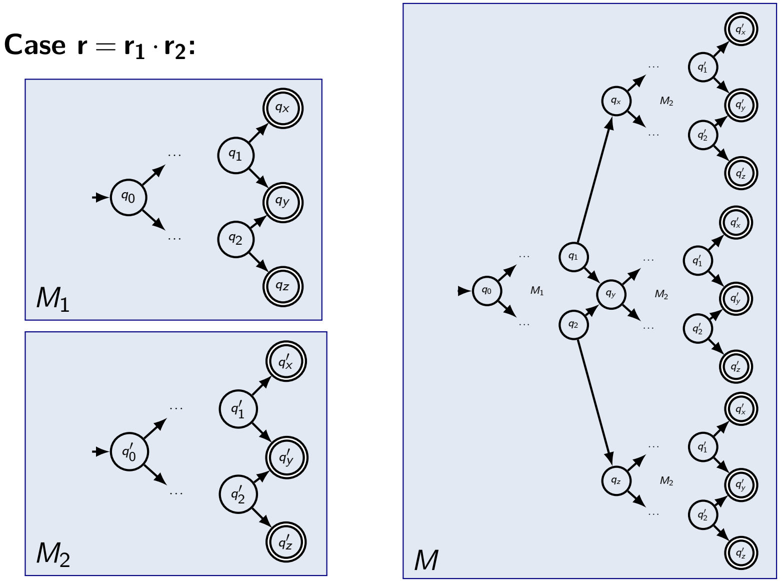


- **Case $\mathbf{r} = \mathbf{a}$:**



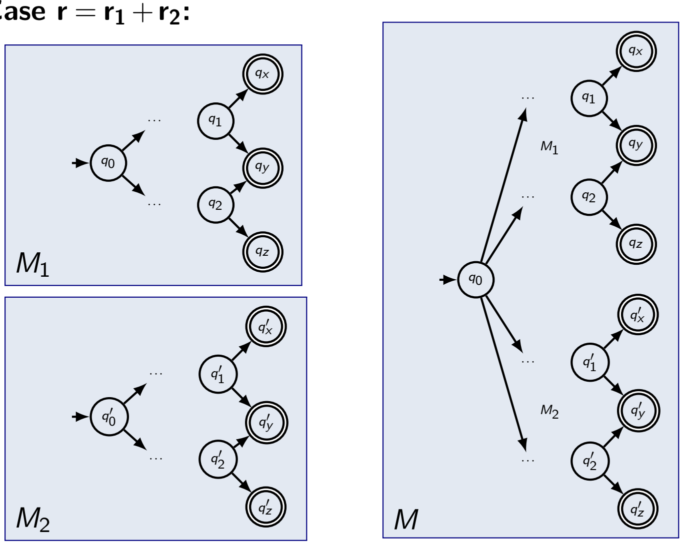
Concatenation

- **Case $\mathbf{r} = \mathbf{r_1} \cdot \mathbf{r_2}$:**



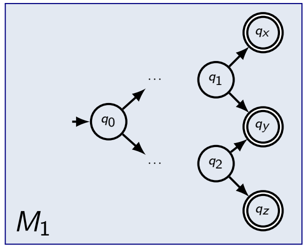
Union

- **Case $\mathbf{r = r_1 + r_2}$:**

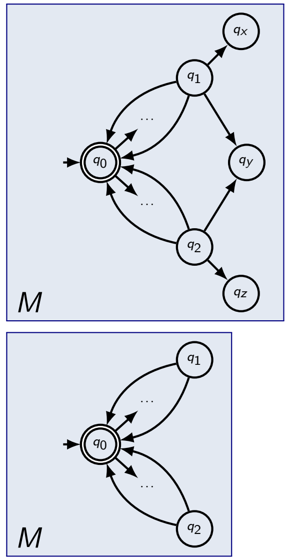


Finite Closure

- **Case $\mathbf{r = r_1^*}$:**

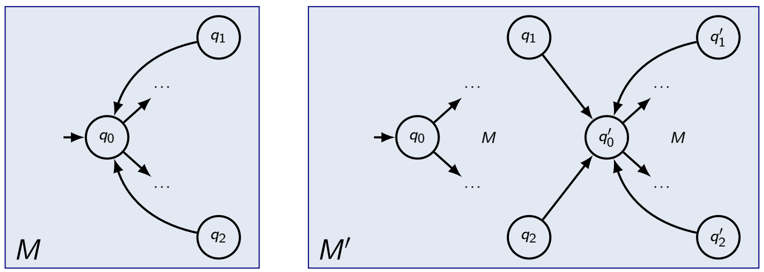


- We may remove q_x, q_y, q_z, if they do not lead to acceptance.
- M cannot (yet) serve as M_2 in case $r_1 \cdot r_2$ or as M_1, M_2 in case $r_1 + r_2$ due to the transitions back to q_0.



Removal of Back Transitions

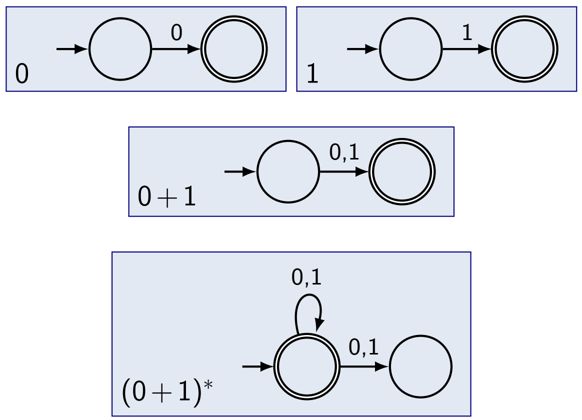
We can construct another automaton without transitions back to q_0.



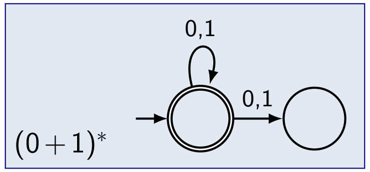
M and M' accept the same language.

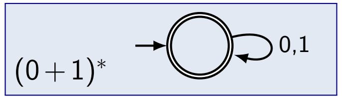
Example

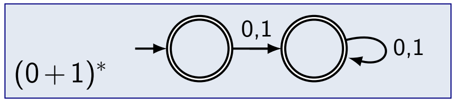
We construct an automaton for $(0+1)^*(00+11)(0+1)^*$.



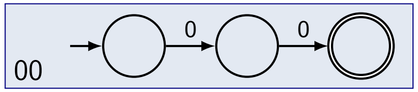
Example (Contd)

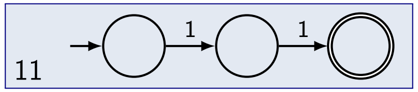


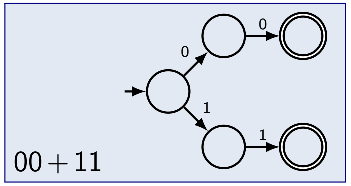


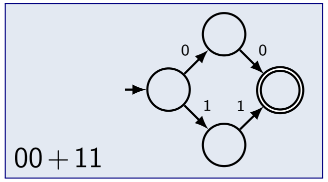


Example (Contd)

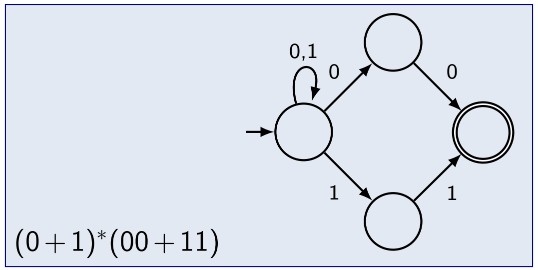


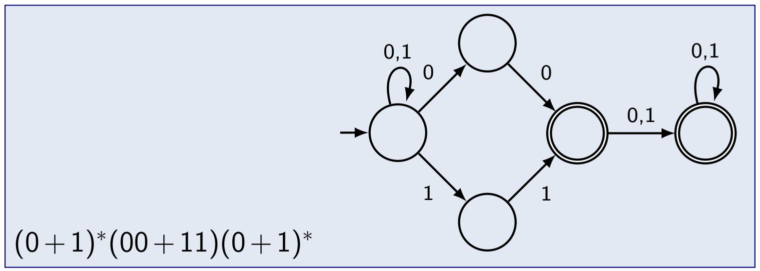




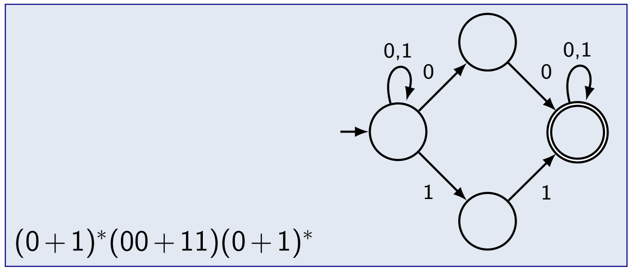


Example (Contd)





Example (Contd)



Simplification after every step yields smaller automata.

Automata to Regular Expressions

DFSM $M = (Q, \sigma, \delta, q_0, F)$ to regular expression r with $L(r) = L(M)$.

- Let $R_{q,p}$ be the set of words that drive M from q to p:

$$R_{q,p} := \{w \in \Sigma^* \mid \delta^*(q,w) = p\}$$

- $L(M)$ is the set of words that drive M from q_0 to some end state:

$$L(M) = R_{q_0,p_1} \cup \ldots \cup R_{q_0,p_n}$$

  where $F = \{p_1, \ldots, p_n\}$.
- Assume we can construct regular expression $r_{q,p}$ such that

$$L(r_{q,p}) = R_{q,p}$$

  (for arbitrary q, p).
- Then we can construct r:

$$r := r_{q_0,p_1} + \ldots + r_{q_0,p_n}$$

It remains to show how to define $r_{q,p}$.

Automata to Regular Expressions (Contd)

We define for $0 \leq j \leq |Q|$ the following set $R^j_{q,p}$ of words:

- $R^0_{q,p}$: those words of length zero or one that drive M from q to p.

$$R^0_{q,p} := \begin{cases} \{a_1, \ldots, a_n\}, & \text{if } q \neq p \\ \{a_1, \ldots, a_n, \varepsilon\}, & \text{if } q = p \end{cases}$$

  - $a_1, \ldots, a_n \in \Sigma$: those symbols that drive M from q to p:
    $\delta(q, a_i) = p$ for $1 \leq i \leq n$
- $R^{j+1}_{q,p}$: those words that drive M from q to p through states in Q_{j+1}:

$$R^{j+1}_{q,p} := \{w \in R_{q,p} \mid \forall 1 \leq k \leq |w| : \delta^*(q, w \downarrow k) \in Q_{j+1}\}$$

  - Q_{j+1}: the subset of the first $j+1$ symbols in Q:
    $Q_{j+1} = \{q_0, \ldots, q_j\}$
  - $w \downarrow k$: the prefix of w with length k.

Automata to Regular Expressions (Contd)

- Assume we can construct regular expression $r^j_{q,p}$ with

$$L(r^j_{q,p}) = R^j_{q,p}$$

- We can then define regular expression $r_{q,p}$ as

$$r_{q,p} := r^{|Q|}_{q,p}$$

  - We know: $R_{q,p} = R^{|Q|}_{q,p}$.

It remains to show how to define $r^j_{q,p}$.

Automata to Regular Expressions (Contd)

We define $r^j_{q,p}$ by induction on j:

$$r^0_{q,p} := \begin{cases} \emptyset, & \text{if } q \neq p \wedge n = 0 \\ a_1 + \ldots + a_n, & \text{if } q \neq p \wedge n \geq 1 \\ a_1 + \ldots + a_n + \varepsilon, & \text{if } q = p \end{cases}$$

$$r^{j+1}_{q,p} := r^j_{q,p} + r^j_{q,q_j} \cdot (r^j_{q_j,q_j})^* \cdot r^j_{q_j,p}$$

- We have to show $L(r^0_{q,p}) = R^0_{q,p}$.
  - Follows from definition.
- We have to show $L(r^{j+1}_{q,p}) = R^{j+1}_{q,p}$.
  - Core of the proof.

It remains to show $L(r^{j+1}_{q,p}) = R^{j+1}_{q,p}$.

Automata to Regular Expressions (Contd)

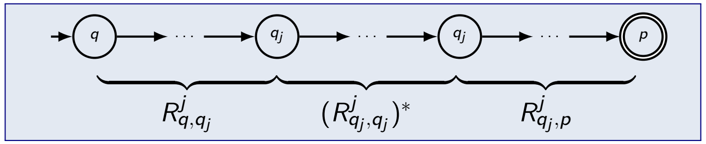
- By the definition of $r^{j+1}_{q,p}$, it suffices to show

$$R^j_{q,p} \cup R^j_{q,q_j} \circ (R^j_{q_j,q_j})^* \circ R^j_{q_j,p} = R^{j+1}_{q,p}$$

  - We show for arbitrary word w

$$w \in R^j_{q,p} \cup R^j_{q,q_j} \circ (R^j_{q_j,q_j})^* \circ R^j_{q_j,p} \Leftrightarrow w \in R^{j+1}_{q,p}$$

- If w drives M from state p to state q via states in Q_{j+1},
  - it either drives M from p to q only via states in Q_j,
  - or we have an occurrence of state $q_j \in Q_{j+1} \backslash Q_j$ along the path:



- In second case, w consists of
  - prefix that drives M from q to first occurrence of q_j via states in Q_j,
  - part that drives M repeatedly from one q_j to the next via states in Q_j
  - suffix that drives M from last occurrence of q_j to p via states in Q_j.

Alternative Construction

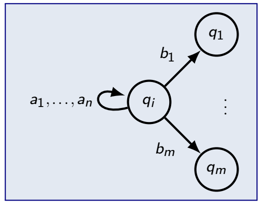
- Arden's Lemma: Let L, U, V be regular languages with $\varepsilon \notin U$. Then

$$L = U \circ L \cup V \Leftrightarrow L = U^* \circ V$$

  - We can solve regular expression equation $l = u \cdot l + v$ as $l = u^* \cdot v$.

Core of a (simpler) construction of a regular expression from a NFSM.

Alternative Construction (Contd)



- For every state q_i construct an equation:

$$X_i = (a_1 + \ldots + a_n) \cdot X_i + b_1 \cdot X_1 + \ldots + b_m \cdot X_m$$

  - If q_i is accepting: $X_i = (a_1 + \ldots + a_n) \cdot X_i + b_1 \cdot X_1 + \ldots + b_m \cdot X_m + \varepsilon$
- In resulting equation system, solve equation for some X_i:

$$X_i = (a_1 + \ldots + a_n)^* \cdot (b_1 \cdot X_1 + \ldots + b_m \cdot X_m)$$

  - If q_i is accepting: $X_i = (a_1 + \ldots + a_n)^* \cdot (b_1 \cdot X_1 + \ldots + b_m \cdot X_m + \varepsilon)$
- Substitute the result, simplify, repeat with another equation.
  - Each substitution removes one variable from the system.

Solution for X_0 is the regular expression for the language of the automaton.

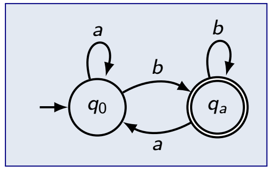
Alternative Construction (Contd)

Some language-preserving regular expression transformations:

$$a \cdot \varepsilon = a$$
$$\varepsilon \cdot a = a$$
$$a \cdot (b + c) = a \cdot b + a \cdot c$$
$$(a + b) \cdot c = a \cdot c + b \cdot c$$

After every step, simplify the result to get an equation to which Arden's lemma can be applied.

Example



$$X_0 = a \cdot X_0 + b \cdot X_a$$
$$X_a = b \cdot X_a + a \cdot X_0 + \varepsilon$$

$$X_a = b^* \cdot (a \cdot X_0 + \varepsilon)$$
$$X_0 = a \cdot X_0 + b \cdot b^* \cdot (a \cdot X_0 + \varepsilon)$$
$$= a \cdot X_0 + b \cdot b^* \cdot a \cdot X_0 + b \cdot b^* \cdot \varepsilon$$
$$= (a + b \cdot b^* \cdot a) \cdot X_0 + b \cdot b^*$$
$$X_0 = (a + b \cdot b^* \cdot a)^* \cdot b \cdot b^*$$

Regular expression $(a + b \cdot b^* \cdot a)^* \cdot b \cdot b^*$.

Closure Properties of Regular Languages

Let L, L_1, L_2 be regular. Then also the following languages are also regular:

1. the complement $\overline{L} = \{x \in \Sigma^* \mid x \notin L\}$;
   - Proof: construction of complement automaton.
2. the union $L_1 \cup L_2 = \{x \in \Sigma^* \mid x \in L_1 \lor x \in L_2\}$;
   - Proof: construction of regular expression $r_1 + r_2$.
3. the intersection $L_1 \cap L_2 = \{x \in \Sigma^* \mid x \in L_1 \land x \in L_2\}$;
   - Proof: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.
4. the concatenation $L_1 \circ L_2$;
   - Proof: construction of regular expression $r_1 \cdot r_2$.
5. the finite closure L^*.
   - Proof: construction of regular expression r^*.

Regular languages can be composed in quite a flexible way.

The Pumping Lemma

Let L be a regular language.

- Pumping Lemma: there exists a natural number n
  - the pumping length of L
- such that every word $w \in L$ with $|w| \geq n$ can be decomposed into three substrings x, y, z, i.e.

$$w = xyz$$

  with $|y| \geq 1$ and $|xy| \leq n$,
- such that also the word with an arbitrarily number of repetitions of the middle part is in the language:

$$xy^k z \in L$$

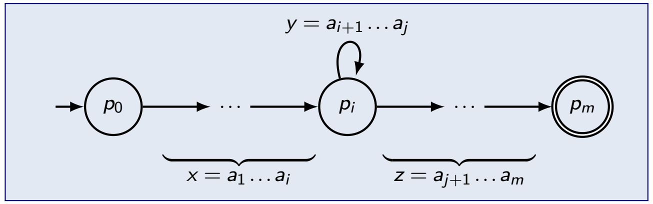
  (for every $k \geq 0$).

Every sufficiently long word of a regular language can be "pumped" to an arbitrarily long word of the language.

Proof of the Pumping Lemma

Regular language L, DFSM M with $L = L(M)$, number n of states of M.

- Let $w = a_1 a_2 \ldots a_m \in L$ with $m \geq n$.
  - Let $p_0, p_1, \ldots, p_m$ be the states that M passes when accepting w.
- Since $m \geq n$, $p_i = p_j$ for some i, j:



- We can define

$$x := a_1 \ldots a_i$$
$$y := a_{i+1} \ldots a_j$$
$$z := a_{j+1} \ldots a_m$$

  such that $w = xyz$ and, for every k, also $xy^k z \in L$.

Example

The Pumping Lemma can be used to show that a language is *not* regular.

- Assume $L = \{0^i 1^i \mid i \in \mathbb{N}\} = \{\varepsilon, 01, 0011, 000111, \ldots\}$ is regular.
  - Let n be the pumping length of L.
- Take word $w := 0^n 1^n \in L$ with $|w| \geq n$. Then $w = xyz$ with

$$xyz = 0^n 1^n, |y| \geq 1, |xy| \leq n$$

- We thus know $x = 0^{n_1}$, $y = 0^{n_2}$, $z = 0^{n_3} 1^{n_4}$ such that

$$n_1 + n_2 + n_3 = n_4, n_2 \geq 1$$

- By the Pumping Lemma, we know $xy^2 z \in L$, which implies

$$n_1 + 2n_2 + n_3 = n_4$$

- But this contradicts

$$n_1 + 2n_2 + n_3 = (n_1 + n_2 + n_3) + n_2 = n_4 + n_2 \geq n_4 + 1 > n_4$$

Thus L is not regular.

Example

The Pumping Lemma can be used to show that a language is *not* regular.

- Assume $L = \{0^{i^2} \mid i \in \mathbb{N}\} = \{\varepsilon, 0, 0000, 000000000, \ldots\}$ is regular.
  - Let n be the pumping length of L.
- Take word $w := 0^{n^2} \in L$ with $|w| \geq n$. Then $w = xyz$ with

$$xyz = 0^{n^2}, |y| \geq 1, |xy| \leq n$$

- By the Pumping Lemma, we know $xy^2 z \in L$.
  - $|xy^2 z| = |xyz| + |y| = n^2 + |y|$ is a square number.
- But we know

$$n^2 < n^2 + 1 \leq n^2 + |y| \leq n^2 + n < n^2 + 2n + 1 = (n+1)^2$$

- But this contradicts $n^2 + |y|$ is a square number.

Thus L is not regular.

Example

Regular languages are too weak to capture general arithmetic.

- But some languages defined by arithmetic are regular.
  - $L := \{0^i \mid i \text{ is even}\} = \{\varepsilon, 00, 0000, 000000, \ldots\}$
  - $L = L((00)^*)$
- Finite languages are always regular.
  - $L = \{0^{i^2} \mid i \in \mathbb{N} \wedge i \leq 3\} = \{\varepsilon, 0, 0000, 000000000\}$
  - $L = L(\varepsilon + 0 + 0000 + 000000000)$

More powerful models are needed to capture general computations.