WS 2014/15

Automaten und Formale Sprachen
Automata and Formal Languages

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2014WS/afs/

Wintersemester 2014/15

Chapter 0 Organizational Matters

- Lectures:
  - 4SWS Tue 08:30–10:00 (MI 00.13.009A)
    Fri 10:15–11:45 (MI 00.13.009A)
    Compulsory elective in area Theoretical Computer Science
    Module no. IN2041
- Exercises/Tutorial:
  - 2SWS Tutorial: Tue 12:00–13:30 (03.11.018)
  - Tutor: Moritz Fuchs
- Valuation:
  - 4V+2ZÜ, 8 ECTS points
- Office hours:
  - Fri 12:00–13:00 and by appointment

- Tutor sessions:
  - Moritz Fuchs, MI 03.09.037 (fuchsmo@in.tum.de)
    Office hours: Tue 14:00–16:00
- Secretariat:
  - Mrs. Lissner, MI 03.09.052 (lissner@in.tum.de)

- Problem sets and final exam:
  - problem sets are made available on Tuesdays on the course webpage
  - must be turned in a week later before class, if you want them marked
  - are discussed in the tutor session
  - probably 12 problem sets
- Exam:
  - final exam: Wednesday, February 11, 2015, 11:30–14:30, room MI HS3
  - the final exam is closed book, no auxiliary means are permitted except for one sheet of DIN-A4 paper, handwritten by yourself
  - to pass the final exam, it is necessary to obtain at least 40% of the point total

- Prerequisites:
  - Fundamentals of Algorithms and Data Structures (GAD)
  - Introduction to Theory of Computer Science (THEO)
- Supplementary courses:
  - Logics
  - Model Checking
  - Verification
  - ...
- Webpage:

  http://www14.in.tum.de/lehre/2014WS/afs/

1. Planned topics for the course

- Automata on finite words
  - Automata classes and conversions
    - Regular expressions, deterministic and nondetermistic automata
    - Conversion algorithms
  - Minimization and reduction
    - Minimizing DFAs
    - Reducing NFAs
  - Boolean operations and tests
    - Implementation on DFAs
    - Membership, complement, union, intersection, emptiness, universality, inclusion
    - Implementation on NFAs
  - Operations on relations
    - Projection, join, post, pre
  - Operations on finite universes: decision diagrams
  - Automata and logic
  - Applications: pattern-matching, verification, Presburger arithmetic

- Automata on infinite words
  - Automata classes and conversions
    - Omega-regular expressions
    - Büchi, Streett, Rabin, and Muller automata
  - Boolean operations
    - Union and intersection
    - Complement
  - Checking emptiness
  - Applications: verification using temporal logic

2. Literature

- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman:
  Introduction to Automata Theory, Languages and Computation,
  Addison-Wesley Longman, 3rd edition, 2006
- John Martin:
  Introduction to Languages and the Theory of Computation,
  McGraw-Hill, 2002
- Michael Sipser:
  Introduction to the Theory of Computation,
  International Edition, Thomson Course Technology:
  Australia-Canada-Mexico-Singapore-Spain-United Kingdom-United States, 2006
- Erich Grädel, Wolfgang Thomas, Thomas Wilke (eds.):
  Automata, logics, and infinite games: a guide to current research,
  LNCS **2500**, Springer-Verlag, 2002

Dominique Perrin, Jean-Eric Pin:
Infinite Words: Automata, Semigroups, Logic and Games,
Academic Press, 2004

Also see Javier Esparza's lecture notes from WS2012/13, onto which this incarnation of the course is also based (but which contain much more material).

Further relevant research papers will be made available during the course.

3. Notational conventions

We use standard notation and basic concepts, as detailed e.g., in the introductory course on

Discrete Structures, IN0015

http://wwwmayr.in.tum.de/lehre/2012WS/ds/index.html.en

4. Mathematical and Notational Basics

4.1 Sets

Example 1

$$A_1 = \{2, 4, 6, 8\};$$
$$A_2 = \{0, 2, 4, 6, \ldots\} = \{n \in \mathbb{N}_0; n \text{ even}\}$$

Notation:

$x \in A \Leftrightarrow A \ni x$	x element of A
$x \notin A$	x not element of A
$B \subseteq A$	B subset of A
$B \subsetneqq A$	B proper subset of A
$\emptyset$	empty set, as opposed to:
$\{\emptyset\}$	set with empty set as (only) element

Special Sets:

- $\mathbb{N} = \{1, 2, \ldots\}$
- $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$
- $\mathbb{Z} =$ set of the integers
- $\mathbb{Q} =$ set of the rational numbers
- $\mathbb{R} =$ set of the real numbers
- $\mathbb{C} =$ set of the complex numbers
- $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ residue classes for division by n
- $[n] = \{1, 2, \ldots, n\}$

Operations on Sets:

- $|A|$ cardinality of the set A
- $A \cup B$ set union
- $A \cap B$ set intersection
- $A \setminus B$ set difference
- $A \triangle B := (A \setminus B) \cup (B \setminus A)$ symmetric difference
- $A \times B := \{(a,b); a \in A, b \in B\}$ cartesian product
- $A \uplus B$ disjoint union; the elements are distinguished according to their origin
- $\bigcup_{i=0}^{n} A_i$ union of the sets $A_0, A_1, \ldots, A_n$
- $\bigcap_{i \in I} A_i$ intersection of the sets A_i mit $i \in I$
- $\mathsf{P}(M) := 2^M := \{N; N \subseteq M\}$ power set of the set M

Example 2

Für $M = \{a, b, c, d\}$ ist

$$\begin{aligned} P(M) \quad = \quad \{ \quad & \emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \\ & \{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}, \\ & \{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}, \\ & \{a,b,c,d\} \\ \} & \end{aligned}$$

Theorem 3

Let the cardinality of set M be n, $n \in \mathbb{N}$. Then $P(M)$ has 2^n elements!

Proof.

Let $M = \{a_1, \ldots, a_n\}$, $n \in \mathbb{N}$. To obtain a set $L \in P(M)$ (*i.e.* $L \subseteq M$), we have, for each $i \in [n]$, the (independent) choice to add a_i to L or not. This results in $2^{|[n]|} = 2^n$ different possibilities for L. □

Remarks:

1. The above theorem also holds for $n = 0$, *i.e.*, the empty set $M = \emptyset$.
2. The empty set is a subset of every set.
3. $P(\emptyset)$ has exactly $\emptyset$ as element.

4.2 Relations and Mappings

Let $A_1, A_2, \ldots, A_n$ be sets. A relation R over $A_1, \ldots, A_n$ is a subset

$$R \subseteq A_1 \times A_2 \times \ldots \times A_n = \bigtimes_{i=1}^{n} A_i$$

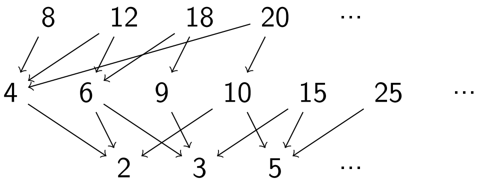
Other notation (infix notation) for $(a, b) \in R$: aRb.

Properties of relations ($R \subseteq A \times A$):

- reflexive: $(a, a) \in R \quad \forall a \in A$
- symmetric: $(a, b) \in R \Rightarrow (b, a) \in R \quad \forall a, b \in A$
- asymmetric: $(a, b) \in R \Rightarrow (b, a) \notin R \quad \forall a, b \in A$
- antisymmetric: $\big[(a, b) \in R \land (b, a) \in R\big] \Rightarrow a = b \quad \forall a, b \in A$
- transitive: $\big[(a, b) \in R \land (b, c) \in R\big] \Rightarrow (a, c) \in R \quad \forall a, b, c \in A$
- equivalence relation: reflexiv, symmetrisch und transitiv
- partial order (aka *partially ordered set*, *poset*): reflexive, antisymmetric and transitive

Example 4

Let $(a, b) \in R$ iff $a|b$, *i.e.*, "a divides b", $a, b \in \mathbb{N} \setminus \{1\}$.
The graphical representation of R without reflexive and transitive arcs is called Hasse diagram:



In the diagram, $a|b$ is denoted by an arc $b \longrightarrow a$.
The relation $|$ is a *partial order*.

Chapter I Automata Theory, an Algorithmic Approach

1. Automata as Data Structures

- Data structures allow us to represent sets of objects in a computer.
- Different data structures support different sets of operations (dictionary, stack, queue, priority queue, . . .):

Op. set	Operations	Data structures
Dictionary	insert, lookup, remove	Hash tables, arrays, search trees
Stack	push, pop	Linked list, array
Priority queue	insert_with_priority, extract_highest_priority	Heap, binomial heap, Fibonacci heap
Union-find	set union, find set	Linked lists, disjoint forests

Automata as Data Structures

In this course we look at automata as a data structure supporting

- the boolean operations of set theory (union, intersection, complement with respect to a given universe set)
- property checks (emptiness, universality, inclusion, equality)
- operations on relations (projections, joins, pre, post)

1.1 Algorithmic Operations on Sets and Relations

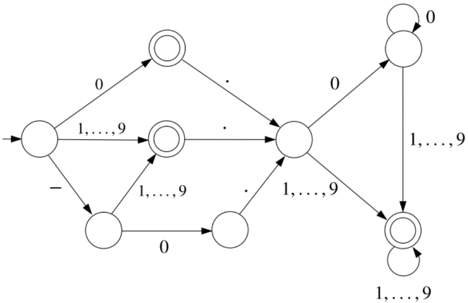
Member(x, X)	:	returns **true** if $x \in X$, **false** otherwise
Complement(X)	:	returns $U \setminus X$
Intersection(X, Y)	:	returns $X \cap Y$
Union(X, Y)	:	returns $X \cup Y$
Empty(X)	:	returns **true** if $X = \emptyset$, **false** otherwise
Universal(X)	:	returns **true** if $X = U$, **false** otherwise
Included(X, Y)	:	returns **true** if $X \subseteq Y$, **false** otherwise
Equal(X, Y)	:	returns **true** if $X = Y$, **false** otherwise
Projection$_1(R)$	:	returns the set $\pi_1(R) = \{x;\ (\exists x)[(x, y) \in R]\}$
Projection$_2(R)$	:	returns the set $\pi_2(R) = \{y;\ (\exists y)[(x, y) \in R]\}$
Join(R, S)	:	returns $R \circ S = \{(x, z);\ (\exists y)[(x, y) \in R \wedge (y, z) \in S]\}$
Post(X, R)	:	returns post$_R(X) = \{y \in U;\ (\exists x \in X)[(x, y) \in R]\}$
Pre(X, R)	:	returns pre$_R(X) = \{y \in U;\ (\exists x \in X)[(y, x) \in R]\}$

Basic Idea

- Elements of the universe can be encoded as words (strings over some alphabet)
- Sets can be encoded as languages (sets of words)
- Automata recognize languages

Example 5

A finite automaton for the strings encoding decimal numbers:



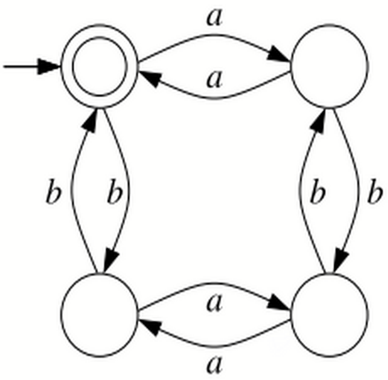
This is a first attempt! What can be corrected/improved?

1.2 Classes of Finite Automata

In the following, we show the definitions of

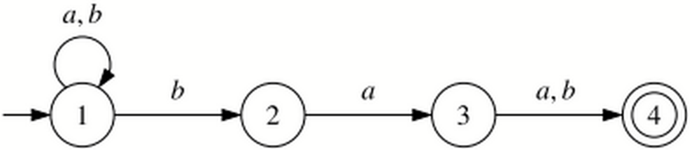
- deterministic finite automata (DFA)
- nondeterministic finite automata (NFA)
- nondeterministic finite automata with ϵ-transitions (NFA-ϵ)
- nondeterministic finite automata with regular-expression-transitions (NFA-reg)

Deterministic finite automata (DFA)



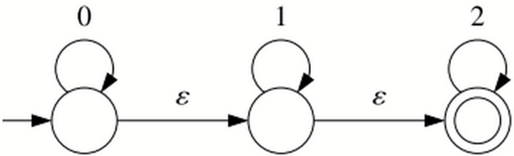
- Q *is a set of* states,
- Σ *is an alphabet,*
- $\delta \colon Q \times \Sigma \to Q$ *is a* transition function,
- $q_0 \in Q$ *is the* initial state, *and*
- $F \subseteq Q$ *is the set of* final states.

Nondeterministic finite automata (NFA)



- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ *is a* transition relation.

Nondeterministic finite automata with epsilon-transitions (NFA-e)



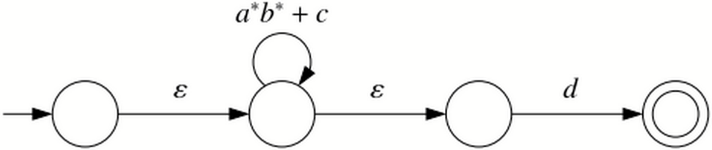
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ *is a* transition relation.

Regular expressions

$$r ::= \emptyset \mid \varepsilon \mid a \mid r_1 r_2 \mid r_1 + r_2 \mid r^* \qquad \textit{where } a \in \Sigma$$

- $L(\emptyset) = \emptyset$,
- $L(\varepsilon) = \{\varepsilon\}$,
- $L(a) = \{a\}$,
- $L(r_1 r_2) = L(r_1) \cdot L(r_2)$, $\qquad L_1 \cdot L_2 = \{w_1 w_2 \in \Sigma^* \mid w_1 \in L_1, w_2 \in L_2\}$
- $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
- $L(r^*) = L(r)^*$. $\qquad L^* = \bigcup_{i \geq 0} L^i$, where $L_0 = \{\varepsilon\}$ and $L_{i+1} = L^i \cdot L$

Nondeterministic finite automata with regular-expression transitions (NFA-reg)

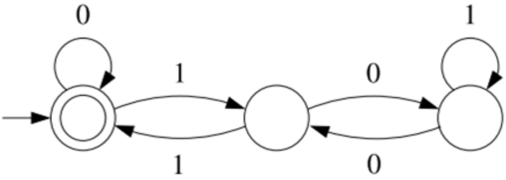


- $\delta\colon Q \times \mathcal{RE}(\Sigma) \to \mathcal{P}(Q)$ *is a relation such that* $\delta(q, r) = \emptyset$ *for all but a finite number of pairs* $(q, r) \in Q \times \mathcal{RE}(\Sigma)$.

1.3 Examples

Example 6

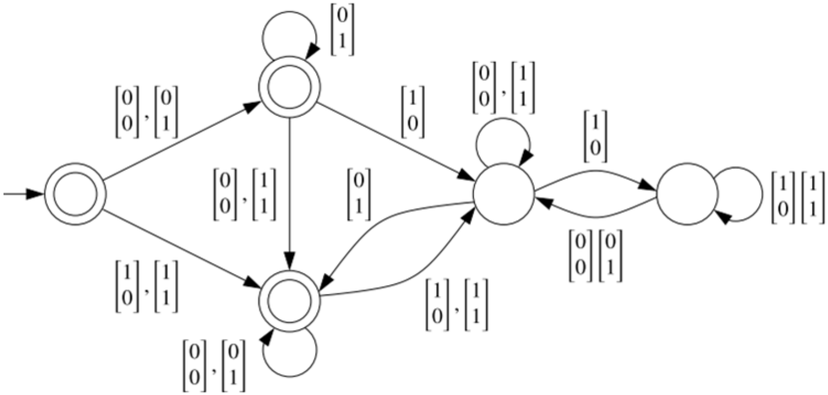
This is a DFA recognizing the multiples of 3, in binary notation:



The states, from left to right, correspond to the residue $\bmod 3$ of the binary number read so far. If this residue is r and the next digit being read is b, then the new residue is $2r + b \bmod 3$, as reflected by the arrows in the above diagram.

Example 7

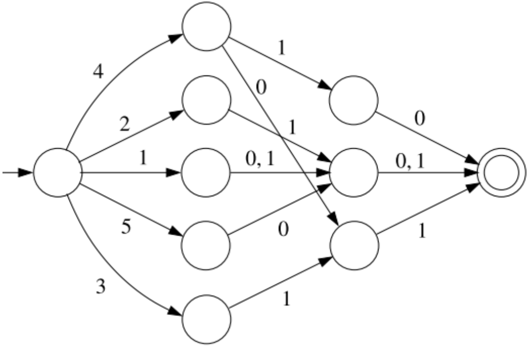
This is a DFA recognizing the nonnegative solutions of $2x - y \leq 2$ in binary (with least significant digit first):



Example 8

This is a DFA recognizing the (initial or intermediate) states of the program leading to termination. The inputs to the DFA are (in order) the number of the current line in the program, the value of the (binary) variable x, and the value of the (binary) variable y:

```
1  while x = 1 do
2     if y = 1 then
3        x ← 0
4     y ← 1 − x
5  end
```



Definition 9

Let $A = (Q, \Sigma, \delta, q_0, F$ be an automaton. A state $q \in Q$ is reachable from $q' \in Q$ if $q = q'$ or if there exists a run $q' \xrightarrow{a_1} \ldots \xrightarrow{a_n} q$ on some input $a_1 \ldots a_n \in \Sigma^*$. A is in normal form if every state is reachable from the initial state.

Unless we say otherwise, we always assume that automata are in normal form!

2. Conversion algorithms

2.1 NFA to DFA, power set construction

Theorem 10

Let L be the language accepted by some nondeterministic finite automaton. Then we can effectively construct a DFA M with

$$L = L(M) \ .$$

Proof.

Let $N = (Q, \Sigma, \delta, S, F)$ be an NFA.

Define

1. $M' := (Q', \Sigma, \delta', q'_0, F')$
2. $Q' := \mathcal{P}(Q)$ $\qquad (\mathcal{P}(Q) = 2^Q$ power set of $Q)$
3. $\delta'(Q'', a) := \bigcup_{q' \in Q''} \delta(q', a)$ for all $Q'' \in Q'$, $a \in \Sigma$
4. $q'_0 := S$
5. $F' := \{Q'' \subseteq Q;\ Q'' \cap F \neq \emptyset\}$

Thus

NFA N:	Q	Σ	δ	S	F
DFA M':	2^Q	Σ	δ'	S	F'

Proof (cont'd):

We have:

$$\begin{aligned} w \in L(N) &\Leftrightarrow \hat{\delta}(S, w) \cap F \neq \emptyset \\ &\Leftrightarrow \widehat{\delta'}(q'_0, w) \in F' \\ &\Leftrightarrow w \in L(M'). \end{aligned}$$

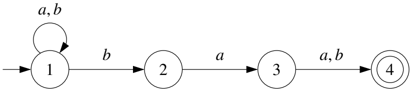
Here, $\hat{\delta}$ denotes the canonical extension of δ to words $w \in \Sigma^*$, and analogously $\widehat{\delta'}$. □

The corresponding algorithm for converting an NFA into a DFA is called subset construction, power set construction, or Myhill construction.

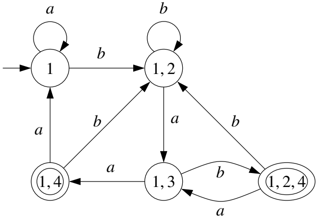
Remark: Of course, the algorithm should also put the NFA it constructs into normal form.

Example 11

NFA:

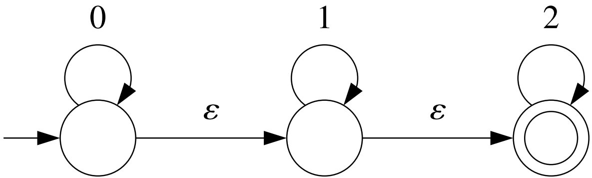


DFA:



2.2 NFA-e to DFA

Consider the NFA-ϵ

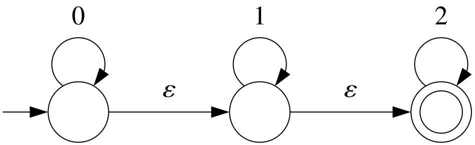


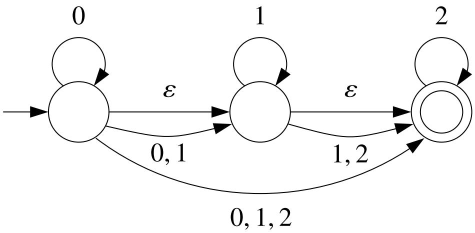
accepting $L(0^*1^*2^*)$.

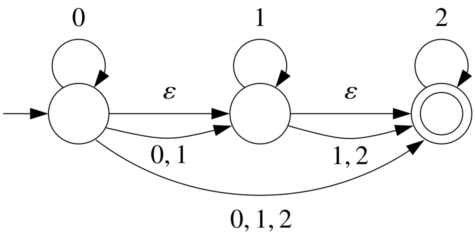
We perform the following algorithm NFA-ϵtoNFA:

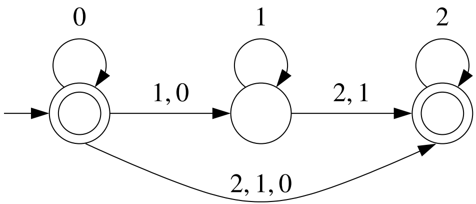
Input: NFA-ϵ $A = (Q, \Sigma, \delta, S, F)$
Output: NFA $B = (Q', \Sigma, \delta', q_0', F')$ with $L(A) = L(B)$
$Q_0' := S$; $Q' := S$; $\delta' := \emptyset$; $F' := F \cap S$
$\delta'' := \emptyset$; $W := \{(q, \alpha, q') \in \delta \mid q \in S\}$
while $W \neq \emptyset$ **do**
  pick (q_1, α, q_2) from W
  if $\alpha \neq \epsilon$ **then**
    add q_2 to Q'; add (q_1, α, q_2) to δ'; **if** $q_2 \in F$ **then** add q_2 to F' **fi**
    for all $q_3 \in \delta(q_2, \epsilon)$ **do if** $(q_1, \alpha, q_3) \notin \delta'$ **then** add (q_1, α, q_3) to W **fi**
    for all $a \in \Sigma, q_3 \in \delta(q_2, a)$ **do if** $(q_2, a, q_3) \notin \delta'$ **then** add (q_2, a, q_3) to W **fi**
  else co $\alpha = \epsilon$ **oc**
    add (q_1, α, q_2) to δ''; **if** $q_2 \in F$ **then** add q_1 to F' **fi**
    for all $\beta \in \Sigma \cup \{\epsilon\}, q_3 \in \delta(q_2, \beta)$ **do**
      if $(q_1, \beta, q_3) \notin \delta' \cup \delta''$ **then** add (q_1, β, q_3) to W **fi**
  fi
od

Example 12



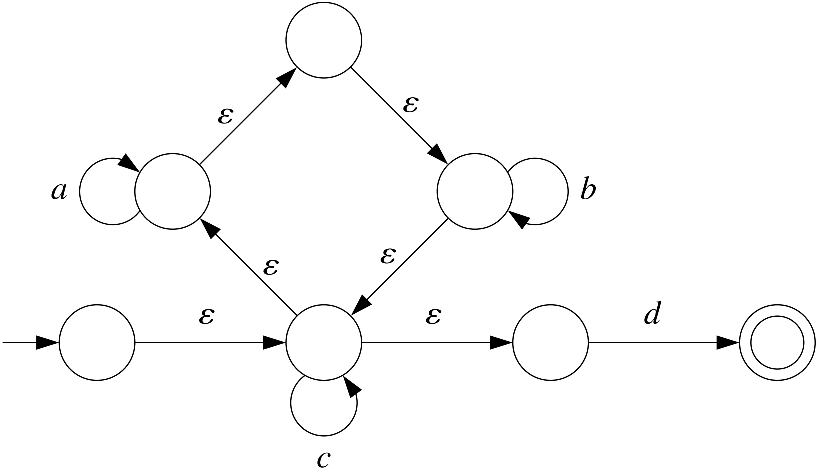



0
1
2
ε
ε
0, 1
1, 2
0, 1, 2


0
1
2
1, 0
2, 1
2, 1, 0

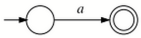
2.3 Regular expressions to NFA-ϵ

For the RE $(a^*b^* + c)^*d$, we intuitively construct the following NFA-ϵ:



Formally, we have the following rules:

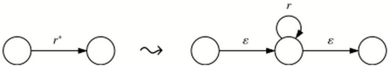
$$\begin{array}{rclrcl} \emptyset \cdot r & \rightsquigarrow & \emptyset & r \cdot \emptyset & \rightsquigarrow & \emptyset \\ r + \emptyset & \rightsquigarrow & r & \emptyset + r & \rightsquigarrow & r \\ \emptyset^* & \rightsquigarrow & \epsilon & & & \end{array}$$



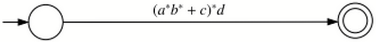
Automaton for the regular expression a, where $a \in \Sigma \cup \{\epsilon\}$

Rule for concatenation

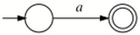
Rule for choice



Rule for Kleene iteration



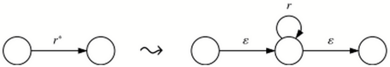
$$\begin{array}{rclrcl} \emptyset \cdot r & \rightsquigarrow & \emptyset & r \cdot \emptyset & \rightsquigarrow & \emptyset \\ r + \emptyset & \rightsquigarrow & r & \emptyset + r & \rightsquigarrow & r \\ \emptyset^* & \rightsquigarrow & \epsilon & & & \end{array}$$



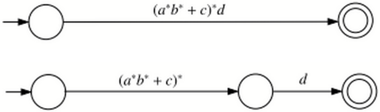
Automaton for the regular expression a, where $a \in \Sigma \cup \{\epsilon\}$

Rule for concatenation

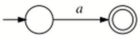
Rule for choice



Rule for Kleene iteration



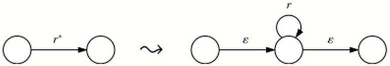
$$
\begin{array}{rclrcl}
\emptyset \cdot r & \rightsquigarrow & \emptyset & r \cdot \emptyset & \rightsquigarrow & \emptyset \\
r + \emptyset & \rightsquigarrow & r & \emptyset + r & \rightsquigarrow & r \\
\emptyset^* & \rightsquigarrow & \epsilon & & &
\end{array}
$$



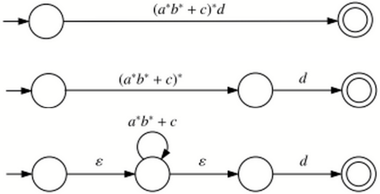
Automaton for the regular expression a, where $a \in \Sigma \cup \{\epsilon\}$

Rule for concatenation

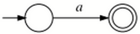
Rule for choice



Rule for Kleene iteration



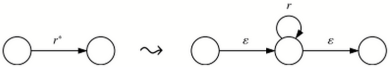
$$\emptyset \cdot r \rightsquigarrow \emptyset \qquad r \cdot \emptyset \rightsquigarrow \emptyset$$
$$r + \emptyset \rightsquigarrow r \qquad \emptyset + r \rightsquigarrow r$$
$$\emptyset^* \rightsquigarrow \epsilon$$



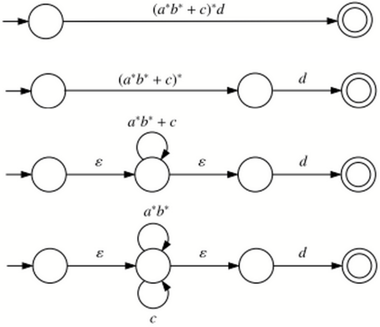
Automaton for the regular expression a, where $a \in \Sigma \cup \{\epsilon\}$

Rule for concatenation

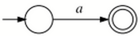
Rule for choice



Rule for Kleene iteration

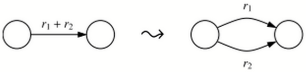


$$\emptyset \cdot r \rightsquigarrow \emptyset \qquad r \cdot \emptyset \rightsquigarrow \emptyset$$
$$r + \emptyset \rightsquigarrow r \qquad \emptyset + r \rightsquigarrow r$$
$$\emptyset^* \rightsquigarrow \epsilon$$

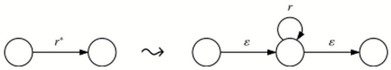


Automaton for the regular expression a, where $a \in \Sigma \cup \{\epsilon\}$

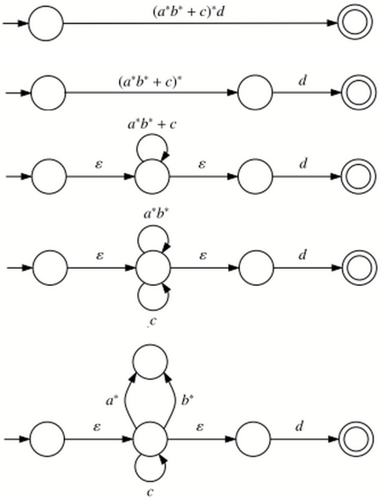
Rule for concatenation

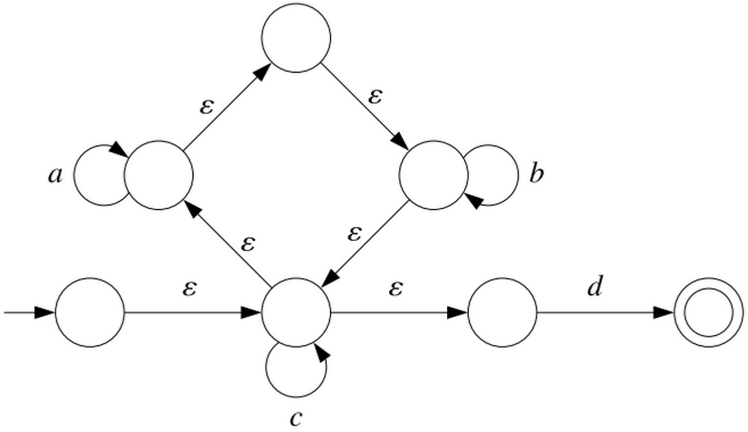


Rule for choice

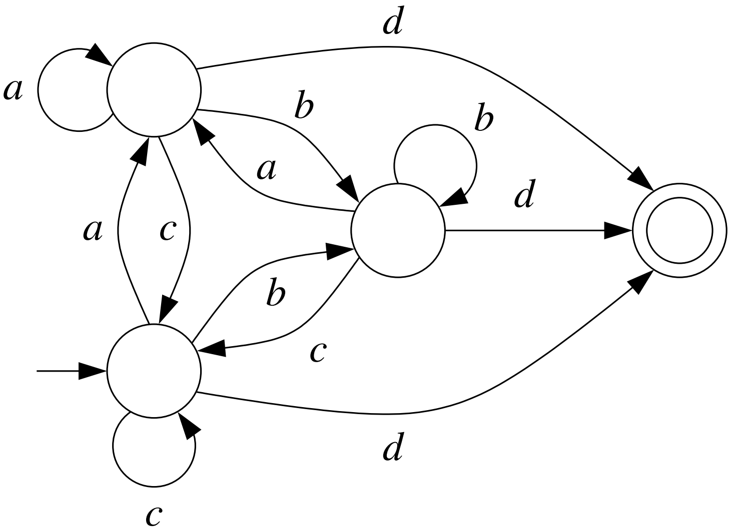


Rule for Kleene iteration



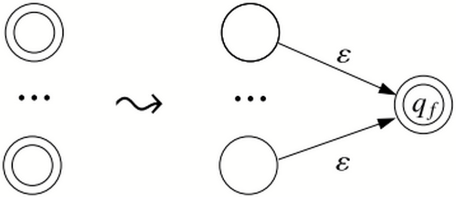

ε
ε
a
b
ε
ε
ε
ε
d
c

And finally, removing ϵ-transitions, we obtain:

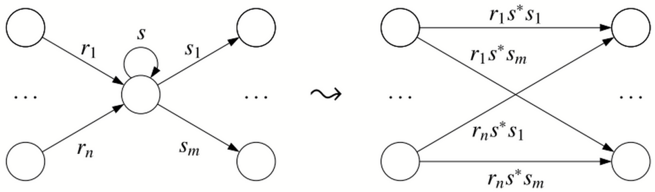


2.4 NFA-ϵ to regular expressions

Preprocessing:



Processing:

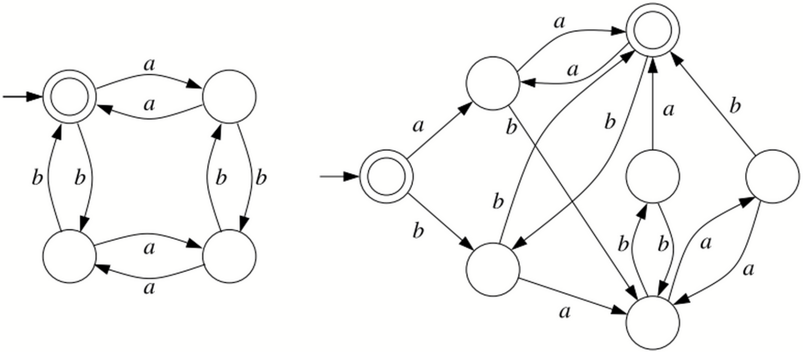


Postprocessing (if necessary):

3. Minimization and Reduction

In this section, we are going to look at the problem of constructing minimal size DFA's for a given regular language, or reducing the size of an NFA without changing the language it accepts.

Example 13



3.1 Residual

Definition 14

Let $L \subseteq \Sigma^*$ be a language, and $w \in \Sigma^*$ a word. The w-residual of L is the language

$$L^w := \{u \in \Sigma^*;\ wu \in L\}\,.$$

A language $L' \subseteq \Sigma^*$ is a residual of L if $L' = L^w$ for at least one $w \in \Sigma^*$.

We note that:

$$(L^w)^u = L^{wu}\,.$$

Relation between residuals and states:
Let A be a DFA and q a state of A.

Definition 15

The state-language $L_A(q)$ (or just $L(q)$) is the language recognized by A with q as initial state.

We remark:

- State-languages are residuals. For every state q of A, $L(q)$ is a residual of $L(A)$.
- Residuals are state-languages. For every residual R of $L(A)$, there is a state q such that $R = L(q)$.

Important consequence:

A regular language has finitely many residuals,

and, equivalently,

languages with infinitely many residuals are not regular.

Canonical DFA for a regular language:

Definition 16

Let $L \subseteq \Sigma^*$ be a formal language. The canonical DFA for L is the DFA $C_L := (Q_L, \Sigma, \delta_L, q_{0L}, F_L)$ given by

- Q_L is the set of residuals of L, *i.e.*, $Q_L = \{L^w;\ w \in \Sigma^*\}$
- $\delta(K, a) = K^a$ for every $K \in Q_L$ and $a \in \Sigma$
- $q_{0L} = L$, and
- $F_L = \{K \in Q_L\ ; \epsilon \in K\}$

Theorem 17

The canoncial DFA for L recognizes L.

Proof.

Let $w \in \Sigma^*$. We show by induction on $|w|$ that $w \in L$ iff $w \in L(C_L)$.

$$
\begin{array}{lll}
 & \epsilon \in L & (w = \epsilon) \\
\iff & L \in F_L & (\text{definition of } F_L) \\
\iff & q_{0L} \in F_L & (q_{0L} = L) \\
\iff & \epsilon \in L(C_L) & (q_{0L} \text{ is the initial state of } C_L)
\end{array}
$$

$$
\begin{array}{lll}
 & aw' \in L & \\
\iff & w' \in L^a & (\text{definition of } L^a) \\
\iff & w' \in L(C_{L^a}) & (\text{induction hypothesis}) \\
\iff & aw' \in L(C_L) & (\delta_L(L, a) = L^a)
\end{array}
$$

□

Definition 18

Let $L \subseteq \Sigma^*$ be a formal language. Define the relation $\equiv_L \subseteq \Sigma^* \times \Sigma^*$ by

$$x \equiv_L y \Leftrightarrow (\forall z \in \Sigma^*)[xz \in L \Leftrightarrow yz \in L]$$

Lemma 19

$\equiv_L$ *is a right-invariant equivalence relation.*

Here right-invariant means:

$$x \equiv_L y \Rightarrow xu \equiv_L yu \text{ for all } u \ .$$

Proof.

Clear! □

Theorem 20 (Myhill-Nerode)

Let $L \subseteq \Sigma^*$. Then the following are equivalent:

1. *L is regular*
2. *$\equiv_L$ has finite* index *($=$ number of equivalence classes)*
3. *L is the union of some of the finitely many equivalence classes of $\equiv_L$.*

Proof.

(1)⇒(2):

Let $L = L(A)$ for some DFA $A = (Q, \Sigma, \delta, q_0, F)$.

Then we have

$$\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y) \quad \Rightarrow \quad x \equiv_L y \; .$$

Thus there are at most as many equivalence classes as A has states.

Proof.

(2)⇒(3):

Let $[x]$ be the equivalence class of x, $y \in [x]$ and $x \in L$.

Then, by the definition of $\equiv_L$, we have:

$$y \in L$$

Proof.

(3)⇒(1):

Define $A' = (Q', \Sigma, \delta', q_0', F')$ with

$$\begin{array}{rcll} Q' & := & \{[x];\ x \in \Sigma^*\} & \text{(}Q'\text{ finite!)} \\ q_0' & := & [\epsilon] & \\ \delta'([x], a) & := & [xa] \quad \forall x \in \Sigma^*, a \in \Sigma & \text{(consistent!)} \\ F' & := & \{[x];\ x \in L\} & \end{array}$$

Then:

$$L(A') = L$$

□

3.2 Construction of Minimal DFAs

Theorem 21

For a given regular language L, let A be the DFA constructed according to the Myhill-Nerode theorem. Then A has, among all DFAs for L, a minimal number of states.

Proof.

Let $A = (Q, \Sigma, \delta, q_0, F)$ mit $L(A) = L$. Then

$$x \equiv_A y :\Leftrightarrow \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$$

defines an equivalence relation which refines $\equiv_L$.
Thus: $|Q| = \text{index}(\equiv_A) \geq \text{index}(\equiv_L) =$ number of states of the Myhill-Nerode automaton. □

Algorithm for Constructing a Minimal DFA

Input: $A(Q, \Sigma, \delta, q_0, F)$ DFA $\quad (L = L(A))$

Output: equivalence relation on Q.

0. ensure that A is in normal form
1. mark all pairs $\{q_i, q_j\} \in Q^2$ with

$$q_i \in F \text{ and } q_j \notin F \text{ resp. } q_i \notin F \text{ and } q_j \in F \, .$$

2. **for** all unmarked pairs $\{q_i, q_j\} \in Q^2$, $q_i \neq q_j$ **do**

    if $(\exists a \in \Sigma)[\{\delta(q_i, a), \delta(q_j, a)\}$ is marked$]$ **then**

        mark $\{q_i, q_j\}$;

        for all $\{q, q'\}$ in $\{q_i, q_j\}$'s list **do**

            mark $\{q, q'\}$ and remove it from list;

            do this recursively for all pairs in the list of $\{q, q'\}$, and so on.

        od

    else

        for all $a \in \Sigma$ **do**

            if $\delta(q_i, a) \neq \delta(q_j, a)$ **then**

                enter $\{q_i, q_j\}$ into the list of $\{\delta(q_i, a), \delta(q_j, a)\}$

            fi

        od

    fi

od

3. Output: q equivalent to $q' \Leftrightarrow \{q, q'\}$ *not* marked.

Theorem 22

The above algorithm constructs a minimal DFA for $L(A)$.

Proof.

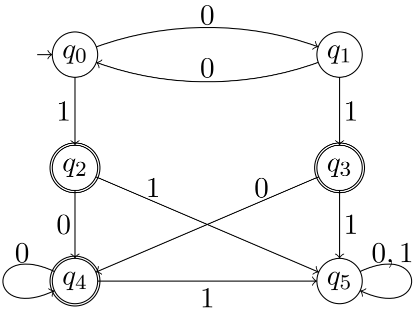
Let $A' = (Q', \Sigma', \delta', q_0', F')$ be the DFA constructed using the equivalence classes determined by the algorithm.
Obviously $L(A) = L(A')$.
We have: $\{q, q'\}$ becomes marked iff

$$(\exists\, w \in \Sigma^*)[\hat{\delta}(q, w) \in F \wedge \hat{\delta}(q', w) \notin F \text{ or vice versa}],$$

as can be seen by a simple induction on $|w|$.
Thus: The number of states of A' (*viz.*, $|Q'|$) equals the index of $\equiv_L$. □

Example 23

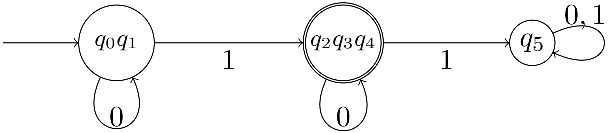
automaton A:



	q_0	q_1	q_2	q_3	q_4	q_5
q_0	/	/	/	/	/	/
q_1		/	/	/	/	/
q_2	×	×	/	/	/	/
q_3	×	×		/	/	/
q_4	×	×			/	/
q_5	×	×	×	×	×	/

automaton A':

$L(A') = 0^*10^*$



Theorem 24

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Then the running time for the above minimization algorithm is $O(|Q|^2|\Sigma|)$.

Proof.

For each $a \in \Sigma$, each position in the table is visited only a constant number of times. □

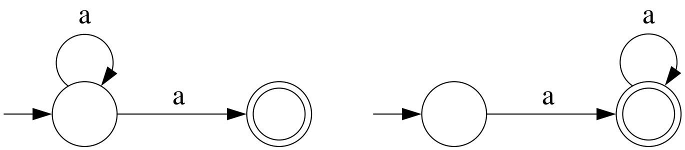
Remark:

The above minimization algorithm

- starts with a very coarse partition of the state set Q, containing $\equiv_L$
- splits a class of the partition whenever it has to
- does this as long as any further splitting might be possible
- finally forms the quotient automaton defined by the final partition of Q (which is a coarsening of $\equiv_A$)

3.3 Minimizing NFAs

We first observe that a minimal NFA need not be unique (unlike the situation for DFAs):



Minimal NFAs are hard to compute:

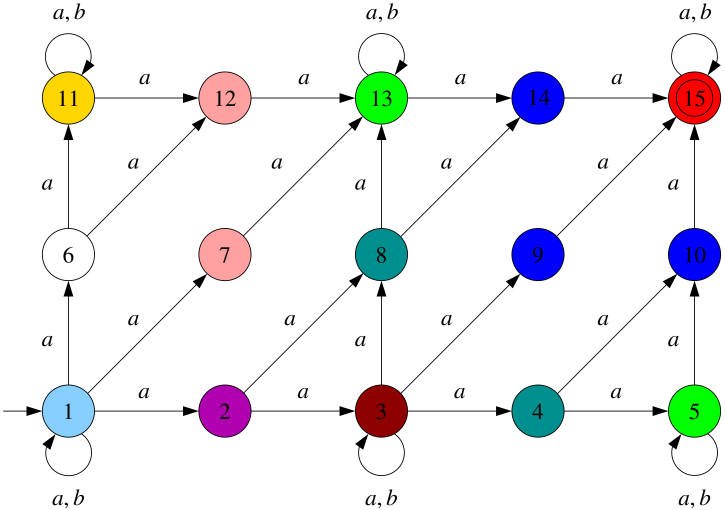
Theorem 25

The following decision problem is PSPACE-complete: given an NFA A and a number $k \geq 1$, is there an NFA with at most k states which is equivalent to A.

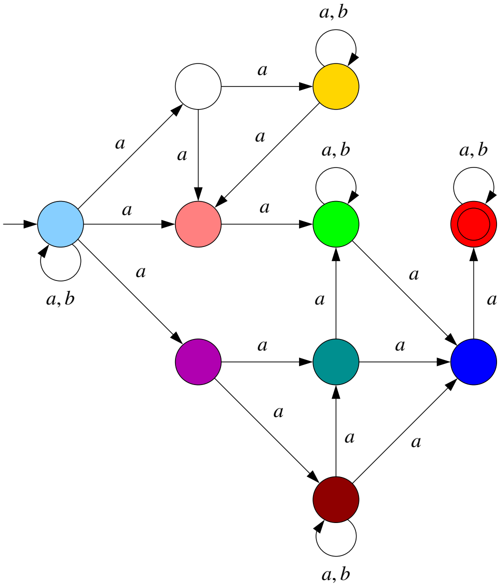
No proof.

However, quite often we can still compute a partition of the state set Q of a given NFA which leads to a reduction of the number of states.

Example 26



Constructing the quotient automaton, we obtain



What is a „suitable“ partition?

- The quotient w.r.t. the partition must recognize the same language as the original NFA.
- So, by the Lemma, we can take any partition that refines the language partition.
- A partition refines the language partition iff states in the same block recognize the same language (states in different blocks may not recognize different langauges, though!).
- Such partitions necessarily refine the partition $\{F, Q \setminus F\}$.

Computing a suitable partition

- Idea: use the same algorithm as for DFA, but with new notions of unstable block and block splitting.
- We must guarantee:

  after termination, states of a block recognize the same language

  or, equivalently

  after termination, states recognizing different languages belong to different blocks

Key observation:

If $L(q_1) \neq L(q_2)$ then either

- one of q_1, q_2 is final and the other non-final, or
- one of q_1, q_2, say q_1, has a transition $q_1 \xrightarrow{a} q_1'$ such that every a-transition $q_2 \xrightarrow{a} q_2'$ satisfies: $L(q_1') \neq L(q_2')$.

This suggests the following definition:

Definition: Let B, B' blocks of a partition P, and let $a \in \Sigma$.
The pair (a, B') splits B if there are states $q_1, q_2 \in B$ such that

$$\delta(q_1, a) \cap B' = \emptyset \quad \text{and} \quad \delta(q_2, a) \cap B' \neq \emptyset$$

The result of the split is the partition

$$Ref_P^{NFA}[B, a, B] = (P \setminus \{B\}) \cup \{B_0, B_1\}$$

where

$$B_0 = \{q \in B \mid \delta(q, a) \cap B' = \emptyset\}$$
$$B_1 = \{q \in B \mid \delta(q, a) \cap B' \neq \emptyset\}$$

A partition is unstable if there are B, a, B' such that (a, B') splits B , otherwise it is stable.

CSR(*A*)

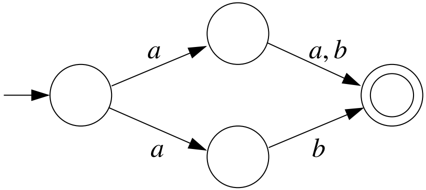
Input: NFA $A = (Q, \Sigma, \delta, q_0, F)$

Output: The partition *CSR*.

if $F = \emptyset$ or $Q \setminus F = \emptyset$ **then return** $\{Q\}$
else $P \leftarrow \{F, Q \setminus F\}$
while P is unstable **do**
    pick $B, B' \in P$ and $a \in \Sigma$ such that (a, B') splits B
    $P \leftarrow Ref_P^{NFA}[B, a, B']$
return P

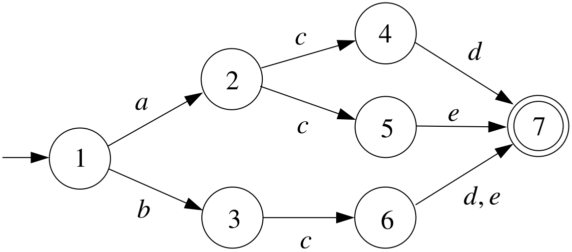
It is not hard to see that the construction given above results in an NFA which is equivalent to the original NFA.

However:
The result might not be minimal:



or

The result is finer than the language partition:



4. Implementing operations on sets using finite automata

4.1 Implementation using DFAs

Recall:

Member(x, X)	:	returns **true** if $x \in X$, **false** otherwise
Complement(X)	:	returns $U \setminus X$
Intersection(X, Y)	:	returns $X \cap Y$
Union(X, Y)	:	returns $X \cup Y$
Empty(X)	:	returns **true** if $X = \emptyset$, **false** otherwise
Universal(X)	:	returns **true** if $X = U$, **false** otherwise
Included(X, Y)	:	returns **true** if $X \subseteq Y$, **false** otherwise
Equal(X, Y)	:	returns **true** if $X = Y$, **false** otherwise

We assume that each object (input, automaton, etc.) is encoded by one word.

We observe:

Membership	:	trivial, linear for fixed automaton
		uniform word problem: low polynomial
Complement	:	trivial, swap final and non-final states
		linear (or even constant) time

Also consider these set operations:

$$\begin{array}{lcl}
\textbf{Intersection}(X,Y) & : & \text{returns } X \cap Y \\
\textbf{Union}(X,Y) & : & \text{returns } X \cup Y \\
\textbf{SetDifference}(X,Y) & : & \text{returns } X \setminus Y \\
\textbf{SymmetricSetDifference}(X,Y) & : & \text{returns } X \vartriangle Y \\
\textbf{Op}(X,Y,Z) & : & \text{returns } (X \cup Y) \setminus Z
\end{array}$$

The product construction or pairing for DFAs

Two DFAs run synchronously in parallel, an input word is accepted iff both automata accept it.

Theorem 27

Let $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be two DFAs. Then the product automaton or pairing $M = [M_1, M_2]$ of M_1 and M_2, defined by

$$M := (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$$

with $\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a))$ for all $q_1 \in Q_1, q_2 \in Q_2$ and $a \in \Sigma$, is a DFA recognizing $L(M_1) \cap L(M_2)$.

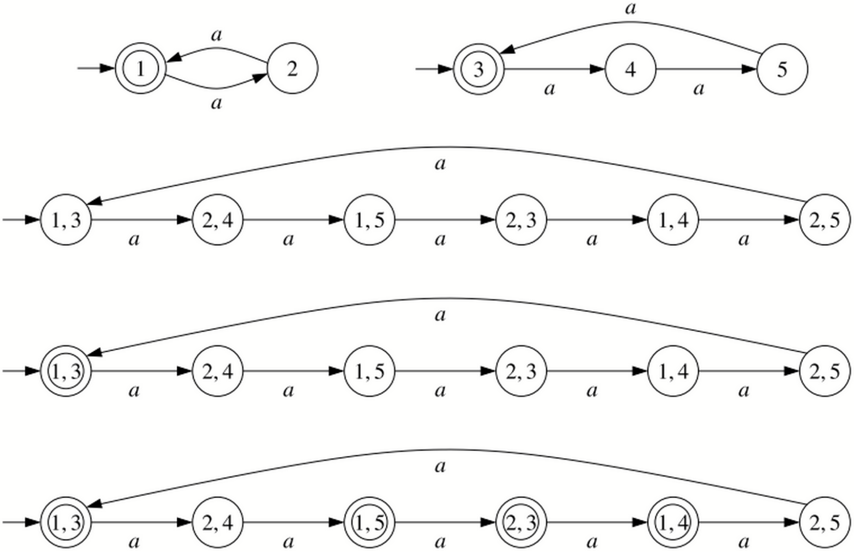
Proof.

Induction on $|w|$. We have:

$$
\begin{aligned}
w \in L(M) \quad &\Leftrightarrow \quad \hat{\delta}((s_1, s_2), w) \in F_1 \times F_2 \\
&\Leftrightarrow \quad (\hat{\delta}_1(s_1, w), \hat{\delta}_2(s_2, w)) \in F_1 \times F_2 \\
&\Leftrightarrow \quad \hat{\delta}_1(s_1, w) \in F_1 \wedge \hat{\delta}_2(s_2, w) \in F_2 \\
&\Leftrightarrow \quad w \in L(M_1) \wedge w \in L(M_2) \\
&\Leftrightarrow \quad w \in L(M_1) \cap L(M_2)\,.
\end{aligned}
$$

□

Question: Does the pairing construction (for intersection) also work for NFAs?



1
2
3
4
5
1, 3
2, 4
1, 5
2, 3
1, 4
2, 5
a

Definition 28

The reversal(mirror) of a word $w = a_1 \cdots a_n$ is

$$w^R := a_n \cdots a_1 \,.$$

The reversal of a language L is

$$L^R := \{w^R; w \in L\} \,.$$

Theorem 29

If L is a regular language, so is L^R.

Proof.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $L = L(M)$. We construct an ϵ-NFA $N = (Q \uplus \{q_0'\}, \Sigma, \delta', q_0', \{q_0\})$ as follows:

- we reverse all state transitions, *i.e.*, $\delta(q, a) = p$ iff $q \in \delta'(p)$;
- we create the new start state q_0' of N, with ϵ-transitions to all $f \in F$;
- q_0 becomes the (only) final state of N.

Following the state transitions of M on some arbitrary input $w \in \Sigma^*$ backwards, we easily see that

$$L(N) = L^R \, .$$

□

A generic algorithm

$$L_1 \widehat{\odot} L_2 \quad = \quad \{w \in \Sigma^* \mid (w \in L_1) \odot (w \in L_2)\}$$

Language operation	$b_1 \odot b_2$
Union	$b_1 \vee b_2$
Intersection	$b_1 \wedge b_2$
Set difference ($L_1 \setminus L_2$)	$b_1 \wedge \neg b_2$
Symmetric difference ($L_1 \setminus L_2 \cup L_2 \setminus L_1$)	$b_1 \Leftrightarrow \neg b_2$

$BinOp[\odot](A_1, A_2)$

Input: DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

Output: DFA $A = (Q, \Sigma, \delta, q_0, F)$ with $\mathcal{L}(A) = \mathcal{L}(A_1) \widehat{\odot} \mathcal{L}(A_2))$

$Q \leftarrow \emptyset$; $F \leftarrow \emptyset$

$q_0 \leftarrow [q_{01}, q_{02}]$

$W \leftarrow \{q_0\}$

while $W \neq \emptyset$ **do**

  pick $[q_1, q_2]$ **from** W

  add $[q_1, q_2]$ **to** Q

  if $(q_1 \in F_1) \odot (q_2 \in F_2)$ **then add** $[q_1, q_2]$ **to** F

  for all $a \in \Sigma$ **do**

    $q'_1 \leftarrow \delta_1(q_1, a)$; $q'_2 \leftarrow \delta_2(q_2, a)$

    if $[q'_1, q'_2] \notin Q$ **then add** $[q'_1, q'_2]$ **to** W

    add $([q_1, q_2], a, [q'_1, q'_2])$ **to** δ

return $(Q, \Sigma, \delta, q_0, F)$

Observation:

- The product automaton/pairing of two DFAs with n_1 resp. n_2 states has (in normal form) $O(n_1 \cdot n_2)$ states.
- Hence, for DFAs with n_1 resp. n_2 states and an alphabet Σ with k letters, the operations union, intersection, etc. can be carried out in $O(k \cdot n_1 \cdot n_2)$ time.

Language tests

Let A, A_1, and A_2 be DFAs, with $L = L(A)$, $L_1 = L(A_1)$, and $L_2 = L(A_2)$ the languages recognized by them, respectively. Note that we assume that all these automata are in normal form!
Then we have

- Emptiness: L is empty iff A has no final states.
- Universality: $L = \Sigma^*$ iff A has only final states.
- Inclusion: $L_1 \subseteq L_2$ iff $L_1 \setminus L_2 = \emptyset$.
- Equality: $L_1 = L_2$ iff $L_1 \triangle L_2 = \emptyset$.

InclDFA(A_1, A_2)

Input: DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

Output: **true** if $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, **false** otherwise

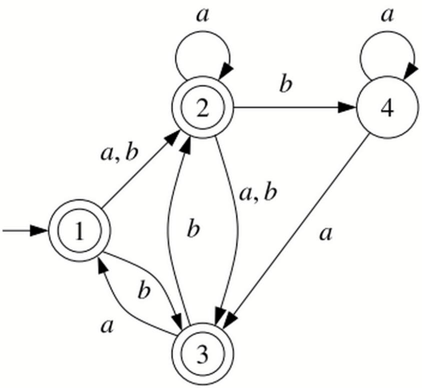
$Q \leftarrow \emptyset$;
$W \leftarrow \{[q_{01}, q_{02}]\}$
while $W \neq \emptyset$ **do**
    pick $[q_1, q_2]$ **from** W
    add $[q_1, q_2]$ **to** Q
    if $(q_1 \in F_1)$ **and** $(q_2 \notin F_2)$ **then return false**
    for all $a \in \Sigma$ **do**
        $q'_1 \leftarrow \delta_1(q_1, a)$; $q'_2 \leftarrow \delta_2(q_2, a)$
        if $[q'_1, q'_2] \notin Q$ **then add** $[q'_1, q'_2]$ **to** W
return true

4.2 Implementation using NFAs

Recall:

Member(x, X)	:	returns **true** if $x \in X$, **false** otherwise
Complement(X)	:	returns $U \setminus X$
Intersection(X, Y)	:	returns $X \cap Y$
Union(X, Y)	:	returns $X \cup Y$
Empty(X)	:	returns **true** if $X = \emptyset$, **false** otherwise
Universal(X)	:	returns **true** if $X = U$, **false** otherwise
Included(X, Y)	:	returns **true** if $X \subseteq Y$, **false** otherwise
Equal(X, Y)	:	returns **true** if $X = Y$, **false** otherwise

Membership



Prefix read	W
ϵ	$\{q_0\}$
a	$\{q_2\}$
aa	$\{q_2, q_3\}$
aaa	$\{q_1, q_2, q_3\}$
$aaab$	$\{q_2, q_3\}$
$aaabb$	$\{q_2, q_3, q_4\}$
$aaabba$	$\{q_1, q_2, q_3, q_4\}$

$Mem[A](w)$
Input: NFA $A = (Q, \Sigma, \delta, q_0, F)$, word $w \in \Sigma^*$,
Output: **true** if $w \in \mathcal{L}(A)$, **false** otherwise

$W \leftarrow \{q_0\}$;
while $w \neq \varepsilon$ **do**
  $U \leftarrow \emptyset$
  for all $q \in W$ **do**
    add $\delta(q, head(w))$ **to** U
  $W \leftarrow U$
  $w \leftarrow tail(w)$
return $(W \cap F \neq \emptyset)$

Complexity:
while loop executed |w| times
for loop executed at most |Q| times
each execution takes O(|Q|) time

Overall: O(|w||Q|^2) time

Complement:

- Swapping final and non-final states does not work.
- Solution: convert to DFA and then swap states.
- Problem: exponential blow-up of size of automaton!
  Hence try to avoid this whenever possible!
- However, in the worst case there is no better way: There are NFAs with n states such that any minimal NFA for their complement has $\Theta(2^n)$ states!

Union and intersection:

The product/pairing construction still works for union and intersection, with the same complexity, but (of course(!)) not for set difference or other non-monotonic operations.

There is a better construction for union (see a few slides down), but not for intersection.

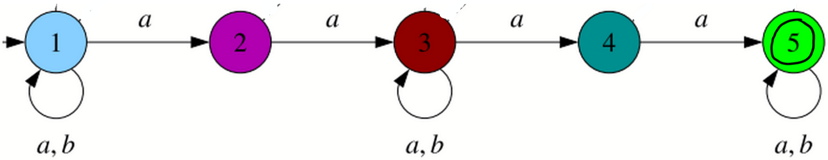
IntersNFA(A_1, A_2)

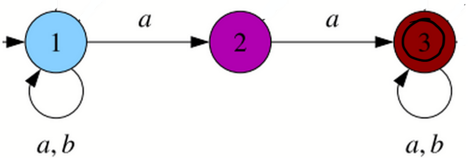
Input: NFA $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

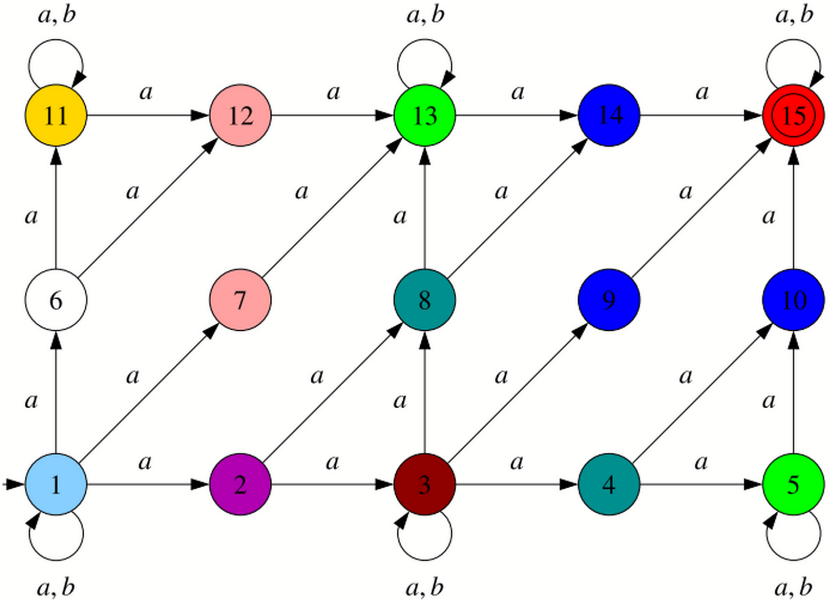
Output: NFA $A_1 \cap A_2 = (Q, \Sigma, \delta, q_0, F)$ with $\mathcal{L}(A_1 \cap A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$

$Q \leftarrow \emptyset$; $F \leftarrow \emptyset$
$q_0 \leftarrow [q_{01}, q_{02}]$
$W \leftarrow \{\, [q_{01}, q_{02}] \,\}$
while $W \neq \emptyset$ **do**
    pick $[q_1, q_2]$ **from** W
    add $[q_1, q_2]$ **to** Q
    if $q_1 \in F_1$ **and** $q_2 \in F_2)$ **then add** $[q_1, q_2]$ **to** F
    for all $a \in \Sigma$ **do**
        for all $q'_1 \in \delta_1(q_1, a), q'_2 \in \delta_2(q_2, a)$ **do**
        if $[q'_1, q'_2] \notin Q$ **then add** $[q'_1, q'_2]$ **to** W
        add $([q_1, q_2], a, [q'_1, q'_2])$ **to** δ
return $(Q, \Sigma, \delta, q_0, F)$

For the complexity, observe that in the worst case the algorithm must examine all pairs $[t_1, t_2]$ of transitions of $\delta_1 \times \delta_2$, but every pair is examined at most once. So the runtime is $\mathcal{O}(|\delta_1||\delta_2|)$.

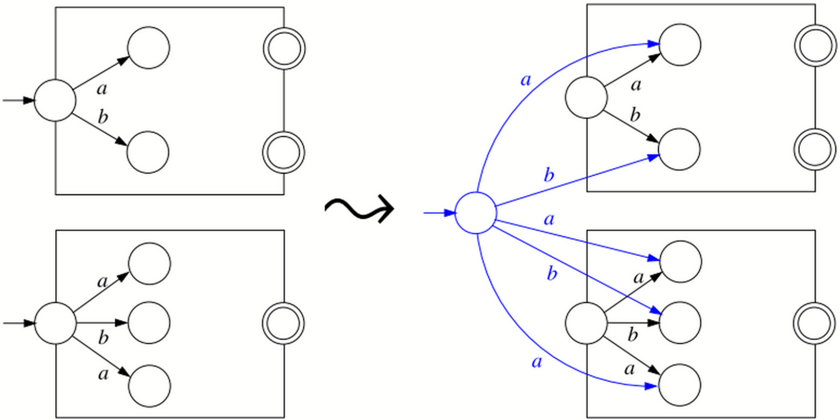

a
a
a
a
1
2
3
4
5
a, b
a, b
a, b


a
a
1
2
3
a, b
a, b



a, b
a, b
a, b
11
12
13
14
15
a
a
a
a
6
7
8
9
10
1
2
3
4
5
a, b
a, b
a, b

Union



UnionNFA(A_1, A_2)

Input: NFA $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

Output: NFA $A_1 \cup A_2$ with $L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$

$Q \leftarrow Q_1 \cup Q_2 \cup \{q_0\}$
$\delta \leftarrow \delta_1 \cup \delta_2$
$F \leftarrow F_1 \cup F_2$
for all $i = 1, 2$ **do**
    if $q_{0i} \in F_i$ **then add** q_0 **to** F
    for all $(q_{0i}, a, q) \in \delta_i$ **do**
        add (q_0, a, q) **to** δ
    if $\delta_i^{-1}(q_{0i}) = \emptyset$ **then**
        remove q_{0i} from Q
        for all $a \in \Sigma, q \in \delta_i(q_{0i}, a)$ **do**
            remove (q_{0i}, a, q) from δ_i
return $(Q, \Sigma, \delta, q_0, F)$

Observation:

Clearly, this type of pairing construction does not work for set difference:

SetDiff(A, A) should always produce an NFA recognizing the empty language, but the construction does not work this way!

Emptiness and universality

We observe that an NFA A (in normal form) recognizes the the empty language (*i.e.*, $L(A) = \emptyset$) iff every state of A is non-final.

However, we should also note that the statement

"An NFA is universal iff every state of it is final."

does not hold in general.

In fact, we have

Corollary 30

Emptiness (for DFAs and NFAs) is decidable in linear time.

And . . .

Theorem 31

The universality problem for NFAs is PSPACE-complete.

Proof.

We first show that the universality problem is in PSPACE. In fact, we show that it is in NPSPACE and apply Savitch's theorem.

Given an NFA $A = (Q, \Sigma, \delta, q_0, F)$ with $n = |Q|$ states, our algorithm guesses an input for $B =$ NFAtoDFA(A) leading from $\{q_0\}$ to a non-final state of B, *i.e.*, a set of states of A which are all non-final. If such a run exists, then there is one of length $\leq 2^n$. The algorithm does not store the whole run, only the current state of B, and hence it only needs space linear in n.

Proof (cont'd):

We prove PSPACE-hardness by reduction from the acceptance problem for linearly bounded automata (LBAs). An LBA N is a nondeterministic Turing machine that always halts and only uses the part of the tape containing the input. A configuration of N on an input of length k is encoded as a word of length k. A run of N on an input can be encoded as a word $c_0\#c_1 \ldots \#c_n$, where the c_i's are the encodings of the configurations.

Let Σ be the alphabet used to encode the run of the machine. Given an input x, N accepts if there exists a word w of Σ^* satisfying the following properties:

(a) w has the form $c_0\#c_1 \ldots \#c_n$, where the c_i's are configurations;

(b) c_0 is the initial configuration;

(c) c_n is an accepting configuration; and

(d) for every $0 \leq i \leq n-1$: c_{i+1} is a successor configuration of c_i according to the transition relation of N.

Proof (cont'd):

The reduction shows how to construct in polynomial time, given an LBA N and an input x, an NFA $A(N,x)$ accepting all the words of Σ^* that do not satisfy at least one of the conditions (a)-(d) above. We then have

- If N accepts x, then there is a word $w(N,x)$ encoding an accepting run of N on x, and so $L(A(N,x)) \subseteq \Sigma^* \setminus \{w(N,x)\}$.
- If N does not accept x, then no word encodes an accepting run of N on x, and so $L(A(N,x)) = \Sigma^*$.

Thus, N accepts x if and only if $L(A(N,x)) \neq \Sigma^*$, and we are done. □

Remarks:

1. Complement and then check for emptiness
   — exponential complexity
2. Possible improvements:
   — check for emptiness while complementing: on-the-fly-check
   — test for subsumption

A Subsumption Test

We observe that, while doing the conversion to and the universality check for a DFA, it might not be necessary to store all states.

Definition 32

Let A be a NFA, and let $B = \text{NFAtoDFA}(A)$. A state Q' of B is minimal if no other state Q'' of B satisfies $Q'' \subset Q'$.

Lemma 33

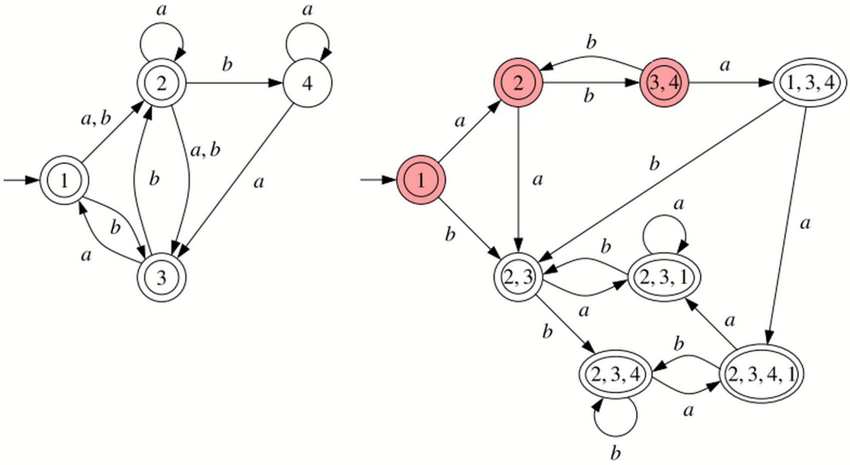
Let A be an NFA, and let $B = \textit{NFAtoDFA}(A)$. A is universal iff every minimal state of B is final.

Proof.

Since A and B recognize the same language, A is universal iff B is universal. So A is universal iff every state of B is final. But a state of B is final iff it contains some final state of A, and so every state of B is final iff every minimal state of B is final. □

UnivNFA(A)
Input: NFA $A = (Q, \Sigma, \delta, q_0, F)$
Output: **true** if $\mathcal{L}(A) = \Sigma^*$, **false** otherwise

```
𝒬 ← ∅;
𝒲 ← { {q0} }
while 𝒲 ≠ ∅ do
    pick Q′ from 𝒲
    if Q′ ∩ F = ∅ then return false
    add Q′ to 𝒬
    for all a ∈ Σ do
        if 𝒲 ∪ 𝒬 contains no Q″ ⊆ δ(Q′, a) then add δ(Q′, a) to 𝒲
return true
```



1
2
3
4
a
b
a, b
1
2
3, 4
1, 3, 4
2, 3
2, 3, 1
2, 3, 4
2, 3, 4, 1

Can this approach be correct?

After all, removing a non-minimal state, we might be preventing the addition of other minimal states in the future!?

Lemma 34

Let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA, and let $B =$ NFAtoDFA(A). After termination of UnivNFA(A), the set $\mathcal{Q}$ contains all minimal states of B.

Proof.

Assume the contrary.

Then B has a shortest path $Q_1 \to Q_2 \cdots Q_{n-1} \to Q_n$ such that, after termination,

- $Q_1 \in \mathcal{Q}$, $Q_n \notin \mathcal{Q}$
- Q_n is minimal

Since the path is shortest, $Q_2 \notin \mathcal{Q}$, and so when UnivNFA processes Q_1, it does not add Q_2. This can only be because UnivNFA already added some $Q_2' \subset Q_2$.

Proof (cont'd):

But then B has a path $Q'_2 \to Q'_3 \cdots Q'_{n-1} \to Q'_n$ with $Q'_n \subseteq Q_n$. Since Q_n is minimal, $Q'_n = Q_n$ and is minimal.

Thus, the path $Q'_2 \to \cdots \to Q'_n$ satisfies

- $Q'_2 \in \mathcal{Q}$, and
- Q'_n is minimal.

This contradicts our assumption that $Q_1 \to \cdots \to Q_n$ is as short as possible. $\square$

Inclusion and equality

Theorem 35

The inclusion problem for NFAs is PSPACE-complete.

Proof.

If, given tw o NFAs A_1 and A_2, we want to test whether $L(A_1) \subseteq L(A_2)$ or, equivalently, $L(A_1) \cap \overline{L(A_2)} = \emptyset$. The negation of the latter can easily be checked (using polynomial space) by guessing a word w (of length at most exponential in the size of A_1 and A_2) such that w is recognized by A_1 but not A_2.

PSPACE-hardness on the other hand follows since an NFA A is universal iff $L(A) = \Sigma^*$, *i.e.*, the universality problem reduces to the inclusion problem. □

- Algorithm: use $L_1 \subseteq L_2$ iff $L_1 \cap \overline{L_2} = \emptyset$

- Concatenate four algorithms:
  (1) determinize A_2,
  (2) complement the result,
  (3) intersect it with A_1, and
  (4) check for emptiness.

- State of (3): pair (q, Q) , where $q \in Q_1$ and $Q \subseteq Q_2$

- Easy optimizations:
  - store only the states of (3), not its transitions;
  - do not perform (1), then (2), then (3): instead, construct directly the states of (3);
  - check (4) while constructing (3);

Further optimization: subsumption test

Definition 36

Let A_1, A_2 be NFAs, and let $B_2 = \text{NFAtoDFA}(A_2)$. A state $[q_1, Q_2]$ of $[A_1, B_2]$ is minimal if no other state $[q_1', Q_2']$ satisfies $q_1' = q_1$ and $Q_2' \subset Q_2$.

Lemma 37

$LL(A_1) \subseteq L(A_2)$ iff every minimal state $[q_1, Q_2]$ of $[A_1, B_2]$ satisfying $q_1 \in F_1$ also satisfies $Q_2 \cap F_2 \neq \emptyset$.

Proof.

Since A_2 and B_2 recognize the same language, $L(A_1) \subseteq L(A_2)$ iff $L(A_1) \cap \overline{L(A_2)} = \emptyset$ iff $L(A_1) \cap \overline{L(B_2)} = \emptyset$ iff $[A_1, B_2]$ has a state $[q_1, Q_2]$ such that $q_1 \in F_1$ and $Q_2 \cap F_2 = \emptyset$. But $[A_1, B_2]$ has some state satisfying this condition iff it has some minimal state satisfying it. □

Algorithm `InclNFA`(A_1, A_2):
Input: `NFAs` $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
Output: **true** `if` $L(A_1) \subseteq L(A_2)$, **false** `otherwise`

$Q := \emptyset$
$W := \{ [q_{01}, \{q_{02}\}] \}$
while $W \neq \emptyset$ **do**
  `pick` $[q_1, Q_2]$ `from` W
  if $q_1 \in F_1$ **and** $Q_2 \cap F_2 = \emptyset$ **then return false fi**
  `add` $[q_1, Q_2]$ `to` Q
  for all $a \in \Sigma, q_1' \in \delta_1(q_1, a)$ **do**
    $Q_2' := \delta_2(Q_2, a)$
    if $W \cup Q$ `contains no` $[q_1'', Q_2'']$ `s.t.` $q_1'' = q_1'$ **and** $Q_2'' \subseteq Q_2'$ **then**
      `add` $[q_1', Q_2']$ `to` W
    fi
return true

- Complexity:
  - Let A_1, A_2 be NFAs with n_1, n_2 states over an alphabet with k letters.
  - Without the subsumption test:
    - The while-loop is executed at most $n_1 \cdot 2^{n_2}$ times.
    - The for-loop is executed at most $O(k \cdot n_1)$ times.
    - An execution of the for-loop takes $O(n_2^2)$ time.
    - Overall: $O(k \cdot n_1^2 \cdot n_2^2 \cdot 2^{n_2})$ time.
  - With the subsumption case the worst-case complexity is higher. Exercise: give an upper bound.

Important special case:

If A_1 is an NFA, but A_2 (already) is a DFA, then

- complementing A_2 is now trivial
- we obtain a running time $O(n_1^2 \cdot n_2)$

Remark: To check for equality, we just check inclusion in both directions. To obtain PSPACE-hardness for equality, just observe the universality problem as above.

5. Implementing operations on relations using finite automata

We discuss how to implement operations on relations over a (possibly infinite) universe U. Even though we will encode the elements of U as words, when implementing relations it is convenient to think of U as an abstract universe, and not as the set Σ^* of words over some alphabet Σ. The reason is that for some operations we encode an element of X not by one word, but by many, actually by infinitely many. In the case of operations on sets this is not necessary, and one can safely identify the object and its encoding as word.

We shall consider a number of operations on relations, some of which are closely related to operations on sets, which we have discussed above. For other types of operations:

Recall:

$\text{Projection}_1(R)$	:	returns the set $\pi_1(R) = \{x;\ (\exists x)[(x, y) \in R]\}$
$\text{Projection}_2(R)$	:	returns the set $\pi_2(R) = \{y;\ (\exists y)[(x, y) \in R]\}$
$\text{Join}(R, S)$	:	returns $R \circ S = \{(x, z);\ (\exists y)[(x, y) \in R \wedge (y, z) \in S]\}$
$\text{Post}(X, R)$	:	returns $\text{post}_R(X) = \{y \in U;\ (\exists x \in X)[(x, y) \in R]\}$
$\text{Pre}(X, R)$	:	returns $\text{pre}_R(X) = \{y \in U;\ (\exists x \in X)[(y, x) \in R]\}$

Encoding objects

- So far we have assumed for convenience:
  a) every word encodes one object.
  b) every object is encoded by exactly one word.

  We now analyze this in more detail.

- Example: objects → natural number encoding → *lsbf*
  $lsbf(5) = 101$ $lsbf(0) = \varepsilon$.
  Satisfies b), but not a).

- We argue that a) can be easily weakened to:
  a') the set of words encoding objects is a regular language.

- The *lsbf* encoding satisfies a'):
  set of encodings → $\{\varepsilon\} \cup \{w \in \Sigma^* \mid w \text{ ends with } 1\}$

Encoding pairs

- Extending the implementations to relations requires to encode pairs of objects.
- How should we encode a pair (n_1, n_2) of natural numbers?

- Consider the pair (n_1, n_2) .
- Assume n_1, n_2 encoded by w_1, w_2 in *lsbf* encoding
- Which should be the encoding of (n_1, n_2) ?
  - Cannot be $w_1 w_2$.
    Then same word encodes many pairs, violates b).

- First attempt: use a separator symbol &, and encode (n_1, n_2) by $w_1 \& w_2$.
  - Problem: not even the identity relation gives a regular language!

- Second attempt: encode (n_1, n_2) as a word over $\{0,1\} \times \{0,1\}$ (intuitively, the automaton reads w_1 and w_2 simultaneously).
  - Problem: what if w_1 and w_2 have different length?
  - Solution: fill the shortest one with 0s.
  - Satisfies b) and a'), but not (a):
    - The number k is encoded by all the words of $s_k 0^*$, where s_k is the *lsbf* encoding of k.
  - We call 0 the padding symbol or padding letter.

- So we assume:
  - The alphabet contains a padding letter #, different or not from the letters used to encode an object.
  - Each object x has a minimal encoding s_x .
  - The encodings of x are all the words of $s_x\#^*$.
  - A pair (x, y) of objects has a minimal encoding $s_{(x,y)}$.

  - The encodings of (x, y) are all the words of $s_{(x,y)}\#^*$.

- Question: if objects (pairs of objects) are encoded by multiple words, which is the set of objects (pairs) recognized by a DFA or NFA?

  (We can no longer say: an object is recognized if its encoding is accepted by the DFA or NFA!)

- Question: because of the new definition of "set of objects recognized by an automaton", do we have to change the implementation of the set operations?

Definition 38

Assume an encoding of the universe U over Σ^* has been fixed. Let A be an NFA.

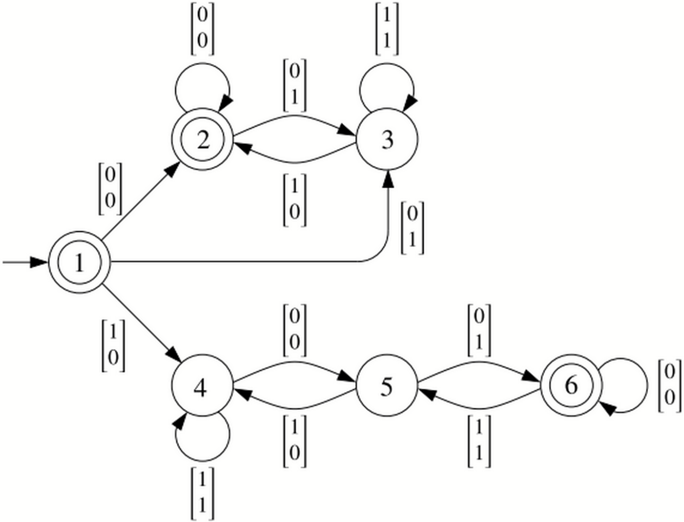
- A accepts $x \in U$ if it accepts all encodings of x.
- A rejects $x \in U$ if it accepts no encoding of x.
- A recognizes a set $X \subseteq U$ if

$$L(A) = \{w \in \Sigma^*;\ \ w \text{ encodes some element of } X\} \,.$$

A set is regular (with respect to the fixed encoding) if it is recognized by some NFA.

Notice that if A recognizes $X \subseteq U$ then, as one would expect, A accepts every $x \in X$ and rejects every $x \notin X$. Hence, with this definition, it may be the case that an NFA neither accepts nor rejects a given x. An NFA is well-formed if it recognizes some set of objects, and ill-formed otherwise.

Transducers



Definition 39

A transducer over Σ is an NFA over the alphabet $\Sigma \times \Sigma$.

Transducers are also called Mealy machines.

According to this definition, a transducer accepts sequences of pairs of letters, but it is convenient to look at it as a machine accepting pairs of words:

Definition 40

Let T be a transducer over Σ. Given words $w_1 = a_1 a_2 \ldots a_n$ and $w_2 = b_1 b_2 \ldots b_n$, we say that T accepts the pair (w_1, w_2) if it accepts the word $(a_1, b_1) \ldots (a_n, b_n) \in (\Sigma \times \Sigma)^*$.

Definition 41

Let T be a transducer.

- T accepts a pair $(x, y) \in X \times X$ if it accepts all encodings of (x, y).
- T rejects a pair $(x, y) \in X \times X$ if it accepts no encoding of (x, y).
- T recognizes a relation $R \subseteq X \times X$ if

$$L(T) = \{(w_x, w_y) \in (\Sigma \times \Sigma)^*;\ (w_x, w_y) \text{ encodes some pair of } R\} .$$

A relation is regular if it is recognized by some transducer.

Examples of regular relations on numbers (*lsbf* encoding):

- the identity relation $\{\ (n, n)\ ;\ n \in \mathbb{N}_0\}$
- the relation "is double of" $\{\ (n, 2n)\ ;\ n \in \mathbb{N}_0\}$

Example 42

The Collatz function is the function $f : \mathbb{N} \to \mathbb{N}$ defined as follows:

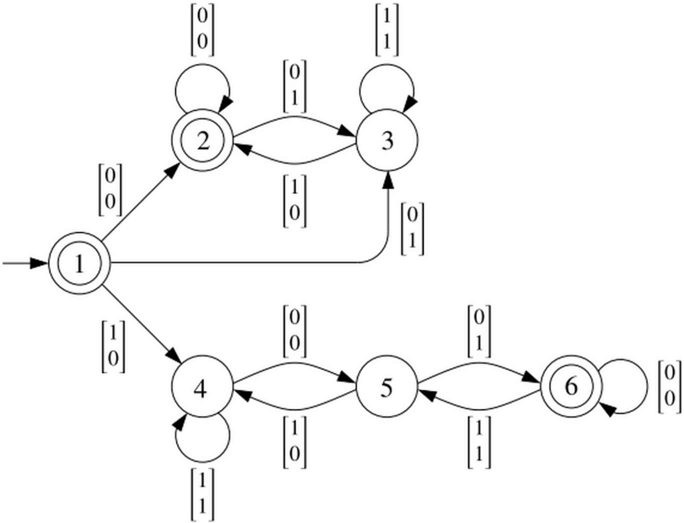
$$f(n) = \begin{cases} 3n + 1 & \text{if } n \text{ is odd} \\ n/2 & \text{if } n \text{ is even} \end{cases}$$

We next show a transducer that recognizes the relation $\{(n, f(n));\ n \in \mathbb{N}\}$ with *lsbf* encoding and with $\Sigma = \{0, 1\}$. The elements of $\Sigma \times \Sigma$ are represented as column vectors with two components. The transducer accepts for instance the pair $(7, 22)$ because it accepts the pairs $(111000^k, 011010^k)$, that is, it accepts

$$\begin{bmatrix}1\\0\end{bmatrix} \begin{bmatrix}1\\1\end{bmatrix} \begin{bmatrix}1\\1\end{bmatrix} \begin{bmatrix}0\\0\end{bmatrix} \begin{bmatrix}0\\1\end{bmatrix} \left(\begin{bmatrix}0\\0\end{bmatrix}\right)^k$$

for every $k \geq 0$.

Transducers



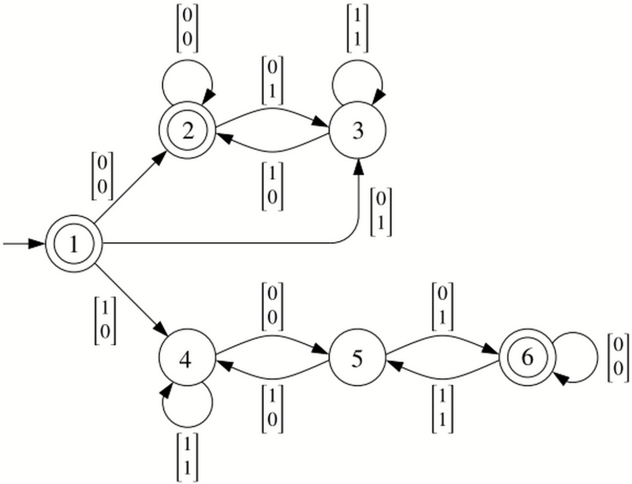
Determinism

- A transducer is deterministic if it is a DFA.
- Observe: if Σ has size n, then a state of a deterministic transducer with alphabet $\Sigma \times \Sigma$ has n^2 outgoing transitions.
- Warning! There is a different definition of determinism:
  - A letter $\begin{bmatrix} a \\ b \end{bmatrix}$ is interpreted as "output b on input a"
  - Deterministic transducer: only one move (and so only one output) for each input.

- Before implementing the new operations:
  - How do we check membership?
  - Can we compute union, intersection and complement of relations as for sets?

Implementing the operations

Projection



- Deleting the second component is not correct
  - Counterexample: $R = \{\ (4,1)\ \}$
  - $s_{(4,1)} = \begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}$
  - DFA for R:

Proj_1(T)

Input: transducer $T = (Q, \Sigma \times \Sigma, \delta, q_0, F)$

Output: NFA $A = (Q', \Sigma, \delta', q'_0, F')$ with $\mathcal{L}(A) = \pi_1(\mathcal{L}(T))$

$Q' \leftarrow Q$; $q'_0 \leftarrow q_0$; $F'' \leftarrow F$

$\delta' \leftarrow \emptyset$;

for all $(q, (a, b), q') \in \delta$ **do**

  add (q, a, q') **to** δ'

$F' \leftarrow PadClosure((Q', \Sigma, \delta', q'_0, F''), \#)$

PadClosure$(A, \#)$

Input: NFA $A = (\Sigma \times \Sigma, Q, \delta, q_0, F)$

Output: new set F' of final states

$W \leftarrow F$; $F' \leftarrow \emptyset$;

while $W \neq \emptyset$ **do**

  pick q **from** W

  add q **to** F'

  for all $(q', \#, q) \in \delta$ **do**

    if $q' \notin F'$ **then add** q' **to** W

return F'

- Problem: we may be accepting $s_x \#^k \#^*$ instead of $s_x \#^*$ and so according to the definition we are not acepting x !

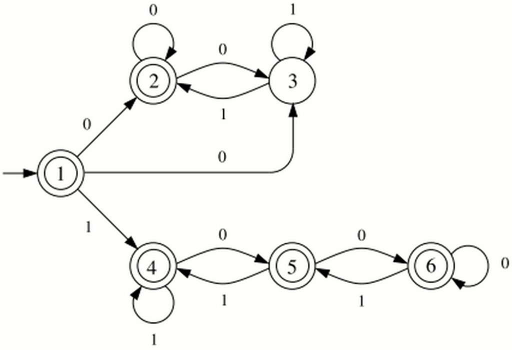
- Solution: if after eliminating the second components some non-final state goes with # ... # to a final state, we mark the state as final.

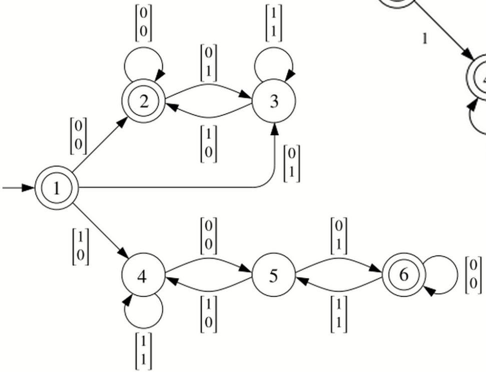
- Complexity: linear in the size of the transducer

- Observe: the result of a projection may be a NFA, even if the transducer is deterministic!!

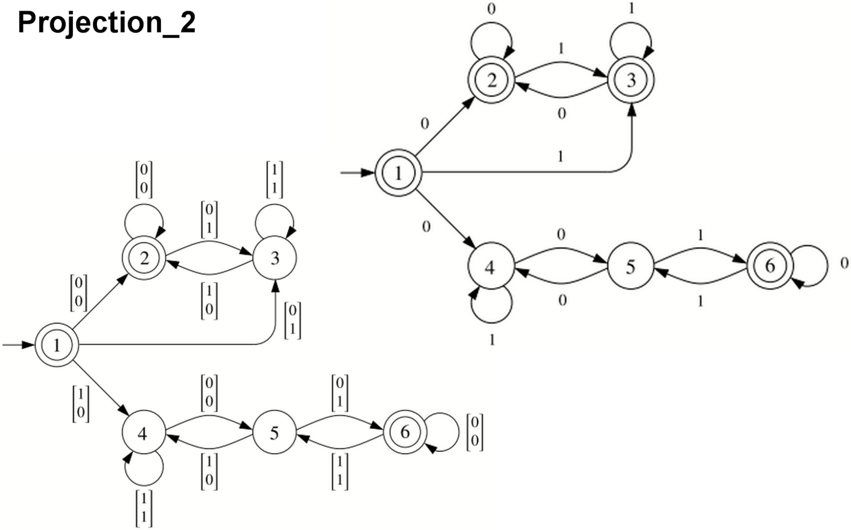
- This is the operation that prevents us from implementing all operations directly on DFAs.

Projection_1





Projection_2



Correctness proof

- Assume: transducer T recognizes a set of pairs
- Prove: the projection automaton A recognizes a set, and this set is the projection onto the first component of the set of pairs recognized by T.

a) A accepts either all encodings or no encoding of an object.
Assume A accepts at least one encoding w of an object x. We prove it accepts all.
If A accepts w, then T accepts $\begin{matrix} w \\ w' \end{matrix}$, for some w'. By assumption T accepts $\begin{matrix} w \\ w' \end{matrix} \begin{bmatrix} \# \\ \# \end{bmatrix}^*$, and so A accepts $w \, \#^*$.
Moreover, $w = s_x \#^k$ for some $k > 0$, and so, by padding closure, A also accepts $s_x \#^j$ for every $j < k$.

b) A only accepts words that are encodings of objects.
Follows easily from the fact that T satisfies the same property for pairs of objects.

Correctness proof

c) If A accepts an object x , then there is an object y such that T accepts (x, y) .

x accepted by A

$\Rightarrow$ s_x accepted by A (part a)

$\Rightarrow$ $\begin{matrix} s_x \\ w \end{matrix}$ accepted by T for some w

By assumption, T only accepts pairs of words encoding some pair of objects. So w encodes some object y. By assumption, T then accepts all encodings of (x, y). So T accepts (x, y) .

Correctness proof

d) If a pair of objects (x, y) is accepted by T, then x is accepted by A.

(x, y) accepted by T

$\Rightarrow$ $\begin{matrix} w_x \\ w_y \end{matrix}$ accepted by T for some encodings w_x, w_y of x and y

$\Rightarrow$ w_x accepted by A

$\Rightarrow$ x accepted by A (part a))

Remember:

The projection automaton of a deterministic transducer may be nondeterministic.

Joi

- Goal: given transducers T_1, T_2 recognizing relations R_1, R_2, construct a transducer $T_1 \circ T_2$ recog∘nizing the relation $R_1 \circ R_2$.

- First step: construct a transducer T that accepts $\begin{smallmatrix} w \\ v \end{smallmatrix}$ iff there is a "connecting" word u such that

  $\begin{smallmatrix} w \\ u \end{smallmatrix}$ is accepted by T_1 and $\begin{smallmatrix} u \\ v \end{smallmatrix}$ is accepted by T2.

We slightly modify the pairing construction.

Instead of:

$$\begin{bmatrix} q_{01} \\ q_{02} \end{bmatrix} \xrightarrow{a_1} \begin{bmatrix} q_{11} \\ q_{12} \end{bmatrix} \quad \text{iff} \quad \begin{matrix} q_{01} \xrightarrow{a_1} q_{11} \\ q_{02} \xrightarrow{a_1} q_{12} \end{matrix}$$

we now use

$$\begin{bmatrix} q_{01} \\ q_{02} \end{bmatrix} \xrightarrow{\begin{bmatrix} a_1 \\ b_1 \end{bmatrix}} \begin{bmatrix} q_{11} \\ q_{12} \end{bmatrix} \quad \text{iff} \quad \begin{matrix} q_{01} \xrightarrow{\begin{bmatrix} a_1 \\ c_1 \end{bmatrix}} q_{11} \\ q_{02} \xrightarrow{\begin{bmatrix} c_1 \\ b_1 \end{bmatrix}} q_{12} \end{matrix}$$

for some letter c1

The transducer T has a run

$$\begin{bmatrix} q_{01} \\ q_{02} \end{bmatrix} \xrightarrow{\begin{bmatrix} a_1 \\ b_1 \end{bmatrix}} \begin{bmatrix} q_{11} \\ q_{12} \end{bmatrix} \xrightarrow{\begin{bmatrix} a_2 \\ b_2 \end{bmatrix}} \begin{bmatrix} q_{21} \\ q_{22} \end{bmatrix} \cdots \begin{bmatrix} q_{(n-1)1} \\ q_{(n-1)2} \end{bmatrix} \xrightarrow{\begin{bmatrix} a_n \\ b_n \end{bmatrix}} \begin{bmatrix} q_{n1} \\ q_{n2} \end{bmatrix}$$

iff T_1 and T_2 have runs

$$q_{01} \xrightarrow{\begin{bmatrix} a_1 \\ c_1 \end{bmatrix}} q_{11} \xrightarrow{\begin{bmatrix} a_2 \\ c_2 \end{bmatrix}} q_{21} \cdots q_{(n-1)1} \xrightarrow{\begin{bmatrix} a_n \\ c_n \end{bmatrix}} q_{n1}$$

$$q_{02} \xrightarrow{\begin{bmatrix} c_1 \\ b_1 \end{bmatrix}} q_{12} \xrightarrow{\begin{bmatrix} c_2 \\ b_2 \end{bmatrix}} q_{22} \cdots q_{(n-1)2} \xrightarrow{\begin{bmatrix} c_n \\ b_n \end{bmatrix}} q_{n2}$$

- We have the same problem as before.
- Let $R_1 = \{ (2,4) \}$, $R_2 = \{ (4,2) \}$.
  Then $R_1 \circ R_2 = \{ (2,2) \}$.
- But the operation we have just defined does not yield the correct result.
- Solution: apply the padding closure again with padding symbol $\begin{bmatrix} \# \\ \# \end{bmatrix}$.

Join(T_1, T_2)

Input: transducers $T_1 = (Q_1, \Sigma \times \Sigma, \delta_1, q_{01}, F_1)$, $T_2 = (Q_2, \Sigma \times \Sigma, \delta_2, q_{02}, F_2)$

Output: transducer $T_1 \circ T_2 = (Q, \Sigma \times \Sigma, \delta, q_0, F)$

$Q, \delta, F' \leftarrow \emptyset;\ q_0 \leftarrow [q_{01}, q_{02}]$

$W \leftarrow \{[q_{01}, q_{02}]\}$

while $W \neq \emptyset$ **do**

    pick $[q_1, q_2]$ **from** W

    add $[q_1, q_2]$ **to** Q

    if $q_1 \in F_1$ and $q_2 \in F_2$ **then add** $[q_1, q_2]$ **to** F'

    for all $(q_1, (a, c), q_1') \in \delta_1, (q_2, (c, b), q_2') \in \delta_2$ **do**

        add $([q_1, q_2], (a, b), [q_1', q_2'])$ **to** δ

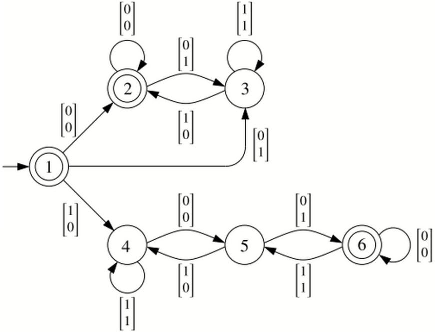
        if $[q_1', q_2'] \notin Q$ **then add** $[q_1', q_2']$ **to** W

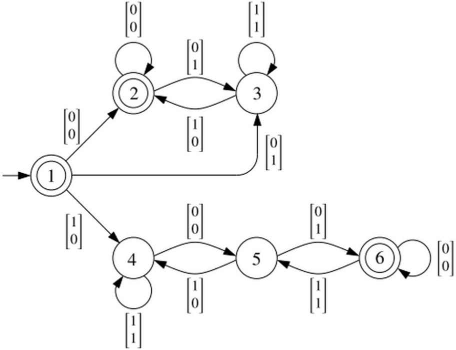
$F \leftarrow$ **PadClosure**$((Q, \Sigma \times \Sigma\delta, q_0, F'), (\#, \#))$

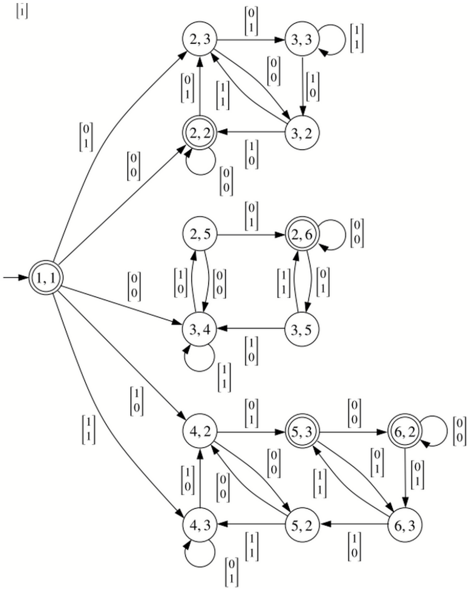
Complexity: similar to pairing

- Example:
  - Let f be the Collatz function.
  - Let $R_1 = R_2 = \{ (n, f(n)) \mid n \geq 0 \}$.
  - Then $R_1 \circ R_2 = \{ (n, f(f(n))) \mid n \geq 0 \}$.

$$f(f((n)) = \begin{cases} n/4 & \text{if } n \equiv 0 \, mod \, 4 \\ 3n/2 + 1 & \text{if } n \equiv 2 \, mod \, 4 \\ 3n/2 + 1/2 & \text{if } n \equiv 1 \, mod \, 4 \text{ or } n \equiv 3 \, mod \, 4 \end{cases}$$






1, 1
2, 3
3, 3
2, 2
3, 2
2, 5
2, 6
3, 4
3, 5
4, 2
5, 3
6, 2
4, 3
5, 2
6, 3

Pre and Post

- Goal (for post):

  given

  – an automaton A recognizing a set X, and

  – a transducer T recognizing a relation R

  construct an automaton B recognizing the set

  $$\{ y \mid \exists\, x \in X : (x, y) \in R \}$$

  We slightly modify the construction for join.

Instead of:

$$\begin{bmatrix} q_{01} \\ q_{02} \end{bmatrix} \xrightarrow{\begin{bmatrix} a_1 \\ b_1 \end{bmatrix}} \begin{bmatrix} q_{11} \\ q_{12} \end{bmatrix}$$

iff

$$q_{01} \xrightarrow{\begin{bmatrix} a_1 \\ c_1 \end{bmatrix}} q_{11}$$

$$q_{02} \xrightarrow{\begin{bmatrix} c_1 \\ b_1 \end{bmatrix}} q_{12}$$

for some letter c1

we now use

$$\begin{bmatrix} q_{01} \\ q_{02} \end{bmatrix} \xrightarrow{b_1} \begin{bmatrix} q_{11} \\ q_{12} \end{bmatrix}$$

iff

$$q_{01} \xrightarrow{a_1} q_{11}$$

$$q_{02} \xrightarrow{\begin{bmatrix} a_1 \\ b_1 \end{bmatrix}} q_{12}$$

for some letter a_1

From Join to Post

Join(T_1, T_2)

Input: transducers $T_1 = (Q_1, \Sigma \times \Sigma, \delta_1, q_{01}, F_1)$, $T_2 = (Q_2, \Sigma \times \Sigma, \delta_2, q_{02}, F_2)$

Output: transducer $T_1 \circ T_2 = (Q, \Sigma \times \Sigma, \delta, q_0, F)$

$Q, \delta, F' \leftarrow \emptyset$; $q_0 \leftarrow [q_{01}, q_{02}]$

$W \leftarrow \{[q_{01}, q_{02}]\}$

while $W \neq \emptyset$ **do**

  pick $[q_1, q_2]$ **from** W

  add $[q_1, q_2]$ **to** Q

  if $q_1 \in F_1$ and $q_2 \in F_2$ **then add** $[q_1, q_2]$ **to** F'

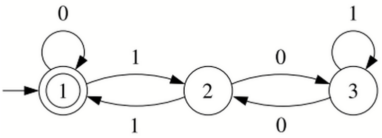
  for all $(q_1, (a, c), q_1') \in \delta_1, (q_2, (c, b), q_2') \in \delta_2$ **do**

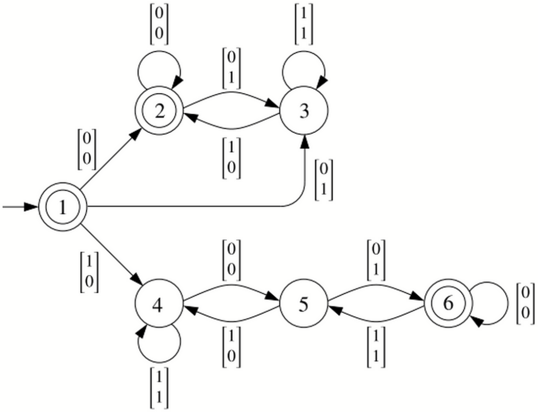
    add $([q_1, q_2], (a, b), [q_1', q_2'])$ **to** δ

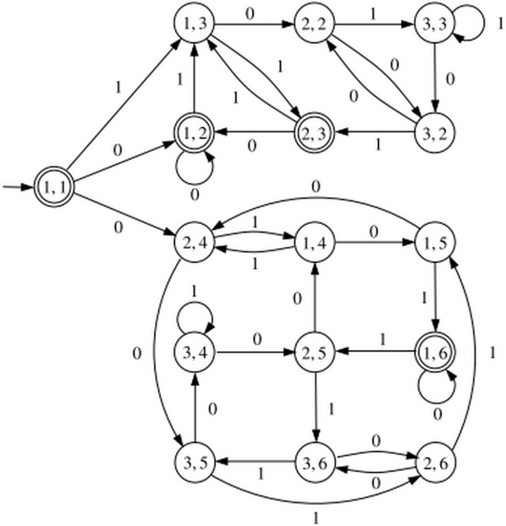
    if $[q_1', q_2'] \notin Q$ **then add** $[q_1', q_2']$ **to** W

$F \leftarrow$ **PadClosure**$((Q, \Sigma \times \Sigma\delta, q_0, F'), (\#, \#))$

Example: compute the set { f(n) | n multiple of 3 }






1, 3
2, 2
3, 3
1, 2
2, 3
3, 2
1, 1
2, 4
1, 4
1, 5
3, 4
2, 5
1, 6
3, 5
3, 6
2, 6
0
1

6. Some pattern matching

Given

- a word w (the text) of length n, and
- a regular expression p (the pattern) of length m,

determine the smallest number k' such that there is a subword $w_{k,k'}$ of w with

$$w_{k,k'} \in L(p) \; .$$

Remark: We here minimize the right end of the matching subword. To make a match unique, one could require *e.g.*, that its length is minimal (or maximal).

NFA-based solution

PatternMatchingNFA(t, p)
Input: text $t = a_1 \dots a_n \in \Sigma^+$, pattern $p \in \Sigma^*$
Output: the first occurrence of p in t, or $\perp$ if no such occurrence exists.

1 $A \leftarrow \mathit{RegtoNFA}(\Sigma^* p)$
2 $S \leftarrow \{q_0\}$
3 **for all** $k = 0$ to $n - 1$ **do**
4   **if** $S \cap F \neq \emptyset$ **then return** k
5     $S \leftarrow \delta(S, a_{k+1})$
6 **return** $\perp$

- Line 1 takes $O(m^3)$ time, output has $O(m)$ states
- Loop is executed at most n times
- One iteration takes $O(s^2)$ time , where s is the number of states of A
- Since $s = O(m)$, the total runtime is $O(m^3 + nm^2)$, and $O(nm^2)$ for $m \leq n$.

DFA-based solution

PatternMatchingDFA(t, p)
Input: text $t = a_1 \ldots a_n \in \Sigma^+$, pattern p
Output: the first occurrence of p in t, or $\perp$ if no such occurrence exists.

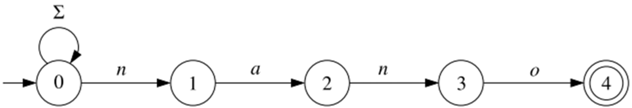
```
1  A ← NFAtoDFA(RegtoNFA(Σ* p))
2  q ← q0
3  for all k = 0 to n − 1 do
4      if q ∈ F then return k
5      q ← δ(q, a_{k+1})
6  return ⊥
```

- Line 1 takes $2^{O(m)}$ time
- Loop is executed at most n times
- One iteration takes constant time
- Total runtime is $O(n) + 2^{O(m)}$

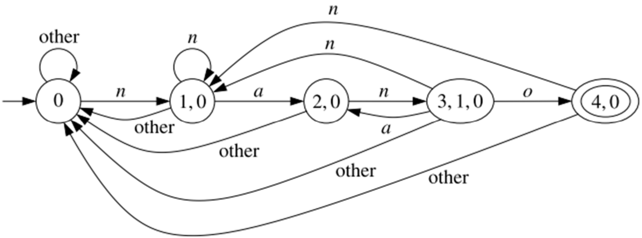
The word case

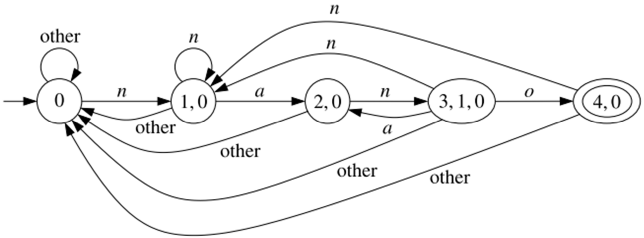
- The pattern p is a word of length m
- Naive algorithm: move a window of size m along the word one letter at a time, and compare with p after each step. Runtime: $O(nm)$
- We give an algorithm with $O(n+m)$ runtime for any alphabet of size $0 \leq |\Sigma| \leq n$.
- First we explore in detail the shape of the DFA for $\Sigma^* p$.

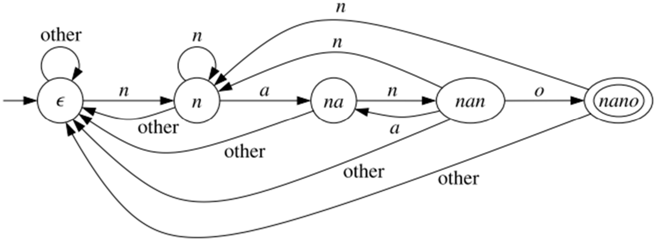
Obvious NFA for $\Sigma^* p$ and $p = nano$



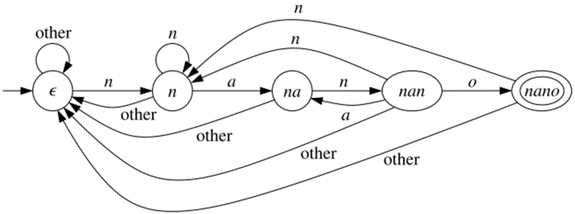
Result of applying NFAtoDFA




other
n
n
n
0
n
1, 0
a
2, 0
n
3, 1, 0
o
4, 0
other
a
other
other
other

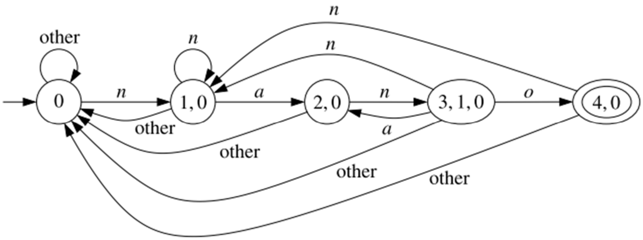

other
n
n
n
ϵ
n
n
a
na
n
nan
o
nano
other
a
other
other
other

Intuition



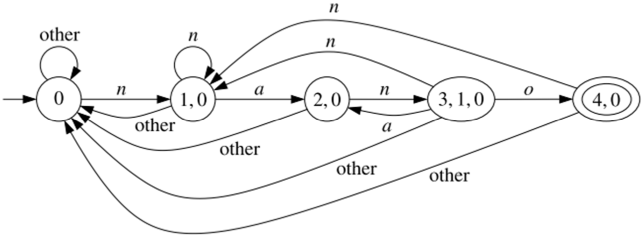
- Transitions of the „spine“ correspond to hits: the next letter is the one that „makes progress“ towards nano
- Other transitions correspond to misses, i.e., „wrong letters“ and „throw the automaton back“

Observations



- For every state $i = 0, 1, \ldots, 4$ of the NFA there is exactly one state S of the DFA such that i is the largest state of S.
- For every state S of the DFA, with the exception of $S = \{0\}$, the result of removing the largest state is again a state of the DFA.

Observations



- For every state $i = 0,1,\ldots,4$ of the NFA there is exactly one state S of the DFA such that i is the largest state of S.
- For every state S of the DFA, with the exception of $S = \{0\}$, the result of removing the largest state is again a state of the DFA.
- Do these properties hold for every pattern p?

Heads and tails, hits and misses

- Head of S, denoted $h(S)$: largest state of S
- Tail of S, denoted $t(S)$: rest of the state
- Example: $h(\{3,1,0\}) = 3$, $t(\{3,1,0\}) = \{1,0\}$

- Given a state S, the letter leading to the next state in the „spine“ is the (unique) hit letter for S
- All other letters are miss letters for S
- Example: hit for {3,1,0} is *o*, whereas *n* or *a* are misses

- Fund. Prop: Let S_k be the k-th state picked from the worklist during the execution of *NFAtoDFA(A_p)*.
  (1) $h(S_k) = k$,
  (2) If $k > 0$, then $t(S_k) = S_l$ for some $l < k$

Proof Idea:

- (1) and (2) hold for $S_0 = \{0\}$.
- For S_k we look at $\delta(S_k, a)$ for each a, where δ transition relation of A_p .
- By i.h. we have $S_k = \{k\} \cup S_l$ for some $l < k$
- We distinguish two cases: *a* is a hit for S_k, and *a* is a miss for S_k .

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

Hit:

$$\begin{array}{ccc} \{k\} & \cup & S_l \\ \downarrow a & & \downarrow a \\ \{k+1\} & \cup & \delta(S_l, a) \end{array}$$

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

$$\{k\} \quad \cup \quad S_l$$

Hit: a ↓ a ↓

$$\{k+1\} \cup \delta(S_l, a)$$

Added to the worklist earlier, and so some $S_{l'}$

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

Hit:

$$
\begin{array}{ccc}
\{k\} & \cup & S_l \\
\downarrow a & & \downarrow a \\
\{k+1\} & \cup & \delta(S_l, a) \\
= & & = \\
\{k+1\} & \cup & S_{l'}
\end{array}
$$

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

$$\begin{array}{ccc} \{k\} & \cup & S_l \\ \downarrow a & & \downarrow a \\ \emptyset & \cup & \delta(S_l, a) \end{array}$$

Miss:

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

$$\{k\} \quad \cup \quad S_l$$

Miss: $\quad a \downarrow \qquad\qquad a \downarrow$

$$\emptyset \quad \cup \quad \delta(S_l, a)$$

$$=$$

$$S_{l'}$$

Consequences

Prop: The result of applying *NFAtoDFA*(A_p), where A_p is the obvious NFA for $\Sigma^* p$, yields a minimal DFA with m states and $|\Sigma|m$ transitions.

Proof: All states of the DFA accept different languages.

So: concatenating *NFAtoDFA and PatternMatchingDFA* yields a $O(n + |\Sigma|m)$ algorithm.

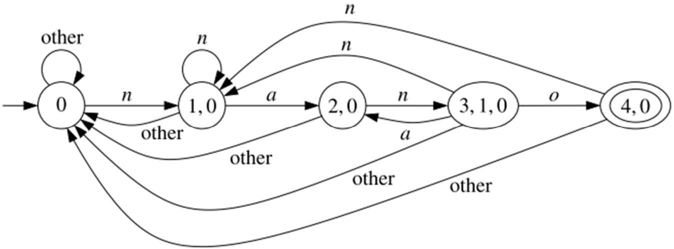
- Good enough for constant alphabet
- Not good enough for $|\Sigma| = O(n)$

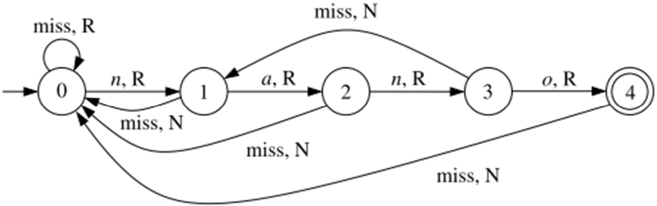
Lazy DFAs

- We introduce a new data structure: lazy DFAs. We construct a lazy DFA for $\Sigma^* p$ with m states and $2m$ transitions.
- Lazy DFAs: automata that read the input from a tape by means of a reading head that can move one cell to the right or stay put
- DFA=Lazy DFA whose head never stays put

Lazy DFA for $\Sigma^* p$

- By the fundamental property, the DFA B_p for $\Sigma^* p$ behaves from state S_k as follows:
  - If a is a hit, then $\delta_B(S_k, a) = S_{k+1}$, i.e., the DFA moves to the next state in the spine.
  - If a is a miss, then $\delta_B(S_k, a) = \delta_B(t(S_k), a)$, i.e., the DFA moves to the same state it would move to if it were in state $t(S_k)$.
- When a is a miss for S_k, the lazy automaton moves to state $t(S_k)$ without advancing the head. In other words, it „delegates“ doing the move to $t(S_k)$
- So the lazyDFA behaves the same for all misses.


other
n
n
n
0
n
1, 0
a
2, 0
n
3, 1, 0
o
4, 0
other
a
other
other
other


miss, R
miss, N
0
n, R
1
a, R
2
n, R
3
o, R
4
miss, N
miss, N
miss, N

- Formally,
  - $\delta_C(S_k, a) = (S_{k+1}, R)$ if a is a hit
  - $\delta_C(S_k, a) = (t(S_k), N)$ if a is a miss
- So the lazy DFA has $m + 1$ states and $2m$ transitions, and can be constructed in $O(m)$ space.

- Running the lazy DFA on the text takes $O(n + m)$ time:
  - For every text letter we have a sequence of „stay put“ steps followed by a „right“ step. Call it a macrostep.
  - Let S_{j_i} be the state after the i-th macrostep. The number of steps of the i-th macrostep is at most $j_{i-1} - j_i + 2$.

  So the total number of steps is at most

$$\sum_{i=1}^{n} (j_{i-1} - j_i + 2) = j_0 - j_n + 2n \leq m + 2n$$

Computing $Miss$

- For the $O(m+n)$ bound it remains to show that the lazy DFA can be constructed in $O(m)$ time.
- Let $Miss(k)$ be the head of the state reached from S_k by a miss.
- It is easy to compute each of $Miss(0), \ldots, Miss(m)$ in $O(m)$ time, leading to a $O(n+m^2)$ time algorithm.
- Already good enough for almost all purposes. But, can we compute all of $Miss(0), \ldots, Miss(m)$ together in time $O(m)$? Looks impossible!
- It isn‘t though ...

$$miss(S_i) = \begin{cases} S_0 & \text{if } i = 0 \text{ or } i = 1 \\ \delta_B(miss(S_{i-1}), b_i) & \text{if } i > 1 \end{cases}$$

$$\delta_B(S_j, b) = \begin{cases} S_{j+1} & \text{if } b = b_{j+1} \text{ (hit)} \\ S_0 & \text{if } b \neq b_{j+1} \text{ (miss) and } j = 0 \\ \delta_B(miss(S_j), b) & \text{if } b \neq b_{j+1} \text{ (miss) and } j \neq 0 \end{cases}$$

Miss(p)
Input: word pattern $p = b_1 \cdots b_m$.
Output: heads of targets of miss transitions.

```
Miss(0) ← 0; Miss(1) ← 0
for i ← 2, . . . , m do
    Miss(i) ← DeltaB( Miss(i − 1) , b_i )
```

DeltaB(j, b)
Input: number $j \in \{0, \ldots, m\}$, letter b.
Output: head of the state $\delta_B(S_j, b)$.

```
while b ≠ b_{j+1} and j ≠ 0 do  j ← Miss(j)
if b = b_{j+1} then return j + 1
else return 0
```

```
Miss(p)
Input: word pattern p = b_1 ··· b_m.
Output: heads of targets of miss transitions.
Miss(0) ← 0; Miss(1) ← 0
for i ← 2, . . . , m do
    Miss(i) ← DeltaB( Miss(i − 1) , b_i )
```

```
DeltaB(j, b)
Input: number j ∈ {0, . . . , m}, letter b.
Output: head of the state δ_B(S_j, b).
while b ≠ b_{j+1} and j ≠ 0 do  j ← Miss(j)
if b = b_{j+1} then return j + 1
else return 0
```

- All calls to *DeltaB* lead together to $O(m)$ iterations of the while loop.
- The call $DeltaB(Miss(i-1), b\_i)$ executes at most $Miss(i-1) - (Miss(i) - 1)$ iterations.

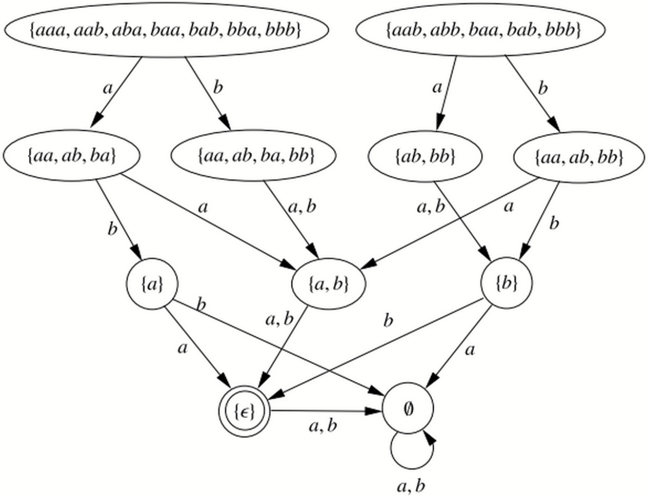
- Total number of iterations:

$$\sum_{i=2}^{m} (Miss(i-1) - Miss(i) + 1)$$
$$\leq Miss(1) - Miss(m) + m$$
$$\leq m$$

7. Finite Universes

- When the universe is finite (*e.g.*, the interval $[0, 2^{32} - 1]$), all objects can be encoded by words of the same length.
- A language L has length $n \geq 0$ if
  - $L = \emptyset$ and $n = 0$, or
  - $L \neq \emptyset$ and every word of L has length n.
- L is a fixed-length language if it has length n for some $n \geq 0$.
- Observe:
  - Fixed-length languages contain finitely many words.
  - $\emptyset$ and $\{\varepsilon\}$ are the only two languages of length 0.

The Master Automaton



- The master automaton over Σ is the tuple $\mathrm{M} = (Q_M, \Sigma, \delta_M, F_M)$, where
  - Q_M is the set of all fixed-length languages;
  - $\delta_M\colon Q_M \times \Sigma \rightarrow Q_M$ is given by $\delta_M(L, a) = L^a$;
  - F_M is the set $\{\, \{\varepsilon\} \,\}$.

- Prop: The language recognized from state L of the master automaton is L.

  Proof: By induction on the length n of L.

  $n = 0$. Then either $L = \emptyset$ or $L = \{\varepsilon\}$, and result follows by inspection.

  $n > 0$. Then $\delta_M(L, a) = L^a$ for every $a \in \Sigma$, and L^a has smaller length than L. By induction hypothesis the state L^a recognizes the language L^a, and so the state L recognizes the language L.

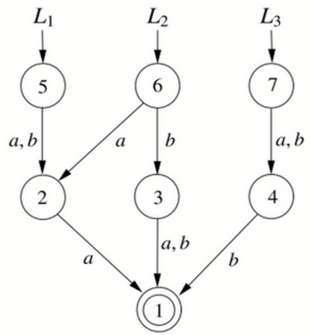
- We denote the „fragment“ of the master automaton reachable from state L by A_L :
  - Initial state is L.
  - States and transitions are those reachable from L.

- Prop: A_L is the minimal DFA recognizing L.

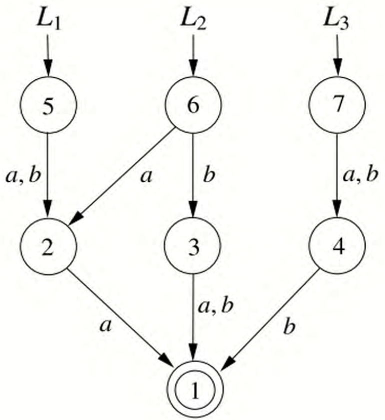
  Proof: By definition, all states of A_L are reachable from its initial state. Since every state of the master automaton recognizes its „own“ language, distinct states of A_L recognize distinct languages.

Data structure for fixed-length languages

- The structure representing the set of languages $\mathcal{L} = \{L_1, \ldots, L_m\}$ is the fragment of the master automaton containing states $L_1, \ldots, L_m$ and their descendants.
- It is a multi-DFA , i.e., a DFA with multiple initial states.

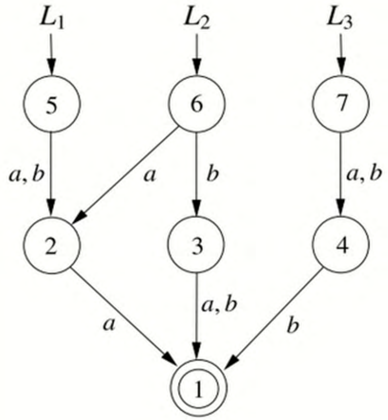


In order to manipulate multi-DFAs we represent them as a *table of nodes*. Assume $\Sigma = \{a_1, \ldots, a_m\}$. A *node* is a pair $\langle q, s \rangle$, where q is a *state identifier* and $s = (q_1, \ldots, q_m)$ is the *successor tuple* of the node. The multi-DFA is represented by a table containing a node for each state, but the state corresponding to the empty language[1].



Ident.	a-succ	b-succ
2	1	0
3	1	1
4	0	1
5	2	2
6	2	3
7	4	4

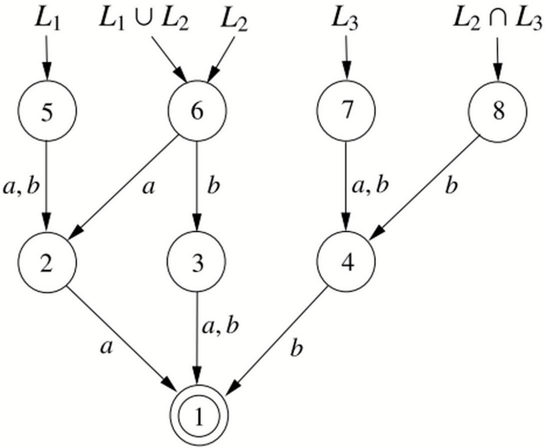
- We represent multi-DFAs as tables of nodes .
- A node is a pair $\langle q, s \rangle$ where
  - q is a state identifier, and
  - $s = (q_1, \dots, q_m)$ is a successor tuple.
- The table for a multi-DFA contains a node for each state but the state for the empty language.



Ident.	a-succ	b-succ
2	1	0
3	1	1
4	0	1
5	2	2
6	2	3
7	4	4

- The procedure $make[T](s)$
  - returns the state identifier of the node of table T having s as successor tuple, if such a node exists;
  - otherwise it adds a new node $\langle q, s \rangle$ to T, where q is a fresh identifier , and returns q.
- $make[T](s)$ assumes that T contains a node for every identifier in s.

Implementing union and intersection



- We give a recursive algorithm $inter[T](q_1, q_2)$:
  - Input: state identifiers q_1, q_2 from table T.
  - Output: identifier of the state recognizing $L(q_1) \cap L(q_2)$ in the multi-DFA for T.
  - Side-effect: if the identifier is not in T, then the algorithm adds new nodes to T, i.e., after termination the table T may have been extended.
- The algorithm follows immediately from the following properties

(1) if $L_1 = \emptyset$, then $L_1 \cap L_2 = \emptyset$;

(2) if $L_2 = \emptyset$, then $L_1 \cap L_2 = \emptyset$;

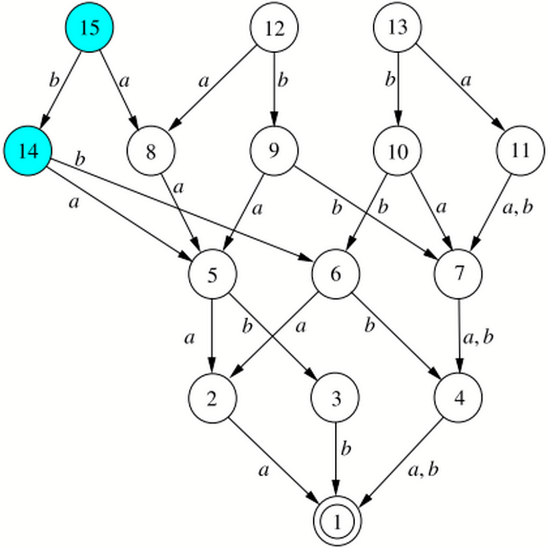
(3) If $L_1 \neq \emptyset$ and $L_2 \neq \emptyset$, then $(L_1 \cap L_2)^a = L_1^a \cap L_2^a$ for every $a \in \Sigma$.

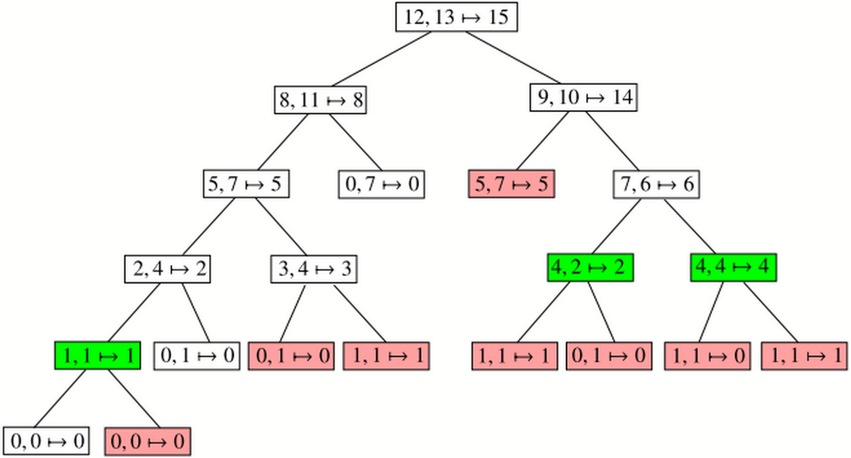
$inter[T](q_1, q_2)$

Input: table T, states q_1, q_2 of T

Output: state recognizing $\mathcal{L}(q_1) \cap \mathcal{L}(q_2)$

if $G(q_1, q_2)$ is not empty **then return** $G(q_1, q_2)$
if $q_1 = q_\emptyset \vee q_2 = q_\emptyset$ **then return** $q_\emptyset$
if $q_1 \neq q_\emptyset \wedge q_2 \neq q_\emptyset$ **then**
    for all $i = 1, \ldots, m$ **do** $r_i \leftarrow inter[T](q_1^{a_i}, q_2^{a_i})$
    $G(q_1, q_2) \leftarrow \mathtt{make}[T](r_1, \ldots, r_m)$
    return $G(q_1, q_2)$


15
12
13
b
a
a
b
b
a
14
8
9
10
11
b
a
a
a
b
b
a
a, b
5
6
7
a
b
a
b
a, b
2
3
4
a
b
a, b
1



12, 13 ↦ 15
8, 11 ↦ 8
9, 10 ↦ 14
5, 7 ↦ 5
0, 7 ↦ 0
5, 7 ↦ 5
7, 6 ↦ 6
2, 4 ↦ 2
3, 4 ↦ 3
4, 2 ↦ 2
4, 4 ↦ 4
1, 1 ↦ 1
0, 1 ↦ 0
0, 1 ↦ 0
1, 1 ↦ 1
1, 1 ↦ 1
0, 1 ↦ 0
1, 1 ↦ 0
1, 1 ↦ 1
0, 0 ↦ 0
0, 0 ↦ 0

Fixed-length complement

In principle ill-defined, because the complement of a fixed-length language is not fixed-length.

We implement the fixed-length complement instead.

Can't we just swap the states for the empty language and the language containing the empty word?

Yes and no ...

Fixed-length complement

Equations:

- if $L = \emptyset$, then $\overline{L} = \Sigma^n$, where n is the length of L;
- if $L = \{\epsilon\}$, then $\overline{L} = \emptyset$; and
- if $\emptyset \neq L \neq \{\epsilon\}$, then $\left(\overline{L}\right)^a = \overline{L^a}$.

  (Observe that $w \in \left(\overline{L}\right)^a$ iff $aw \notin L$ iff $w \notin L^a$ iff $w \in \overline{L^a}$.)

$comp[T, n](q)$

Input: table T, length n, state q of T of length n

Output: state recognizing the fixed-length complement of $L(q)$

if $G(q)$ is not empty **then return** $G(q)$
if $n = 0$ **and** $q = q_\emptyset$ **then return** q_ϵ
else if $n = 0$ **and** $q = q_\epsilon$ **then return** $q_\emptyset$
else $/ * \; n \geq 1 \; * /$
  for all $i = 1, \ldots, m$ **do** $r_i \leftarrow comp[T, n-1](q^{a_i})$
  $G(q) \leftarrow \mathtt{make}[T](r_1, \ldots, r_m)$
  return $G(q)$

Emptiness

empty[T](q)
Input: table T, state q of T
Output: **true** if $\mathcal{L}(q) = \emptyset$, **false** otherwise

1 **return** $q = q_\emptyset$

Universality

- if $L = \emptyset$, then L is not universal;
- if $L = \{\epsilon\}$, then L is universal;
- if $\emptyset \neq L \neq \{\epsilon\}$, then L is universal iff L^a is universal for every $a \in \Sigma$.

$univ[T](q)$

Input: table T, state q of T

Output: **true** if $\mathcal{L}(q)$ is fixed-length universal,
false otherwise

```
if G(q) is not empty then return G(q)
if q = q_∅ then return false
else if q = q_ε then return true
else  /* q ≠ q_∅ and q ≠ q_ε */
    for all i = 1, ..., m do r_i ← comp[T](q^{a_i})
    G(q) ← and(univ[T](r_1), ..., univ[T](r_m))
    return G(q)
```

Inclusion and Equality

Inclusion. Given two languages $L_1, L_2 \subseteq \Sigma^n$, in order to check $L_1 \subseteq L_2$ we compute $L_1 \cap L_2$ and check whether it is equal to L_1 using the equality check shown next. The complexity is dominated by the complexity of computing the intersection.

$eq[T](q_1, q_2)$
Input: table T, states q_1, q_2 of T
Output: **true** if $\mathcal{L}(q_1) = \mathcal{L}(q_2)$, **false** otherwise
1 **return** $q_1 = q_2$

$eq[T_1, T_2](q_1, q_2)$

Input: tables T_1, T_2, states q_1 of T_1, q_2 of T_2

Output: **true** if $\mathcal{L}(q_1) = \mathcal{L}(q_2)$, **false** otherwise

if $G(q_1, q_2)$ is not empty **then return** $G(q_1, q_2)$
if $q_1 = q_{\emptyset 1}$ and $q_2 = q_{\emptyset 2}$ **then** $G(q_1, q_2) \leftarrow \texttt{true}$
else if $q_1 = q_{\emptyset 1}$ and $q_2 \neq q_{\emptyset 2}$ **then** $G(q_1, q_2) \leftarrow \texttt{false}$
else if $q_1 \neq q_{\emptyset 1}$ and $q_2 = q_{\emptyset 2}$ **then** $G(q_1, q_2) \leftarrow \texttt{false}$
else $/ * q_1 \neq q_{\emptyset 1}$ and $q_2 \neq q_{\emptyset 2} * /$
    $G(q_1, q_2) \leftarrow \textbf{and}(\mathsf{eq}(q_1^{a_1}, q_2^{a_1}), \ldots, \mathsf{eq}(q_1^{a_m}, q_2^{a_m}))$
return $G(q_1, q_2)$

What if the starting point is an NFA?

- Given: NFA A accepting a fixed-length language and containing no cycles.

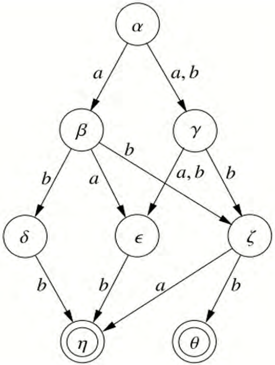
  Goal: simultaneously determinize and minimize A
- Each state of A accepts a fixed-length language.
- We give an algorithm *state*(S):
  - Input: a subset S of states of A accepting languages of the same length.
  - Output: the state of the master automaton accepting $\bigcup_{q \in S} L(q)$.
- Goal is achieved by calling *state*($\{q_0\}$)

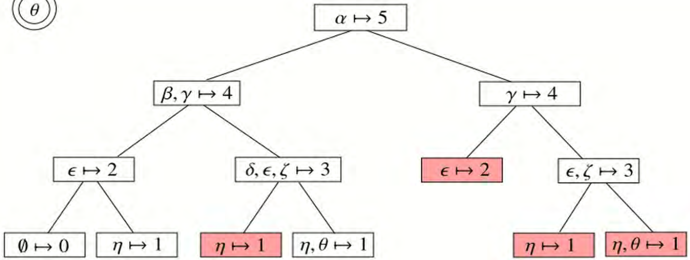
Equations:

- if $S = \emptyset$ then $\mathcal{L}(S) = \emptyset$;
- if $S \cap F \neq \emptyset$ then $\mathcal{L}(S) = \{\epsilon\}$
- if $S \neq \emptyset$ and $S \cap F = \emptyset$, then $\mathcal{L}(S) = \bigcup_{i=1}^{n} a_i \cdot \mathcal{L}(S_i)$, where $S_i = \delta(S, a_i)$.

state$[A](S)$
Input: NFA $A = (Q, \Sigma, \delta, q_0, F)$, set $S \subseteq Q$
Output: master state recognizing $\mathcal{L}(S)$

if $G(S)$ is not empty **then return** $G(S)$
else if $S = \emptyset$ **then return** $q_\emptyset$
else if $S \cap F \neq \emptyset$ **then return** q_ϵ
else $/* \; S \neq \emptyset$ and $S \cap F = \emptyset \; */$
  for all $i = 1, \ldots, m$ **do** $S_i \leftarrow \delta(S, a_i)$
  $G(S) \leftarrow make(state[A](S_1), \ldots, state[A](S_m));$
  return $G(S)$


α
a
a, b
β
γ
b
b
a
a, b
b
δ
ϵ
ζ
b
b
a
b
η
θ


α ↦ 5
β, γ ↦ 4
γ ↦ 4
ϵ ↦ 2
δ, ϵ, ζ ↦ 3
ϵ ↦ 2
ϵ, ζ ↦ 3
∅ ↦ 0
η ↦ 1
η ↦ 1
η, θ ↦ 1
η ↦ 1
η, θ ↦ 1

Operations on relations

Definition 6.10 *A word relation* $R \subseteq \Sigma^* \times \Sigma^*$ *has* length $n \geq 0$ *if it is empty and* $n = 0$, *or if it is nonempty and for all pairs* (w_1, w_2) *of* R *the words* w_1 *and* w_2 *have length* n. *If* R *has length* n *for some* $n \geq 0$, *then we say that* R *is a* fixed-length *word relation, or that* R has fixed-length.

Definition 6.12 *The* master transducer *over the alphabet* Σ *is the tuple* $MT = (Q_M, \Sigma \times \Sigma, \delta_M, F_M)$, *where*

- Q_M *is is the set of all fixed-length relations;*
- $\delta_M : Q_M \times (\Sigma \times \Sigma) \to Q_M$ *is given by* $\delta_M(R, [a, b]) = R^{[a,b]}$ *for every* $q \in Q_M$ *and* $a, b \in \Sigma$;
- $F_M = \{(\varepsilon, \varepsilon)\}$.

With T_R as the "fragment" of MT with R as root we get:

Proposition 6.13 *For every fixed-length word relation* R, *the transducer* T_R *is the minimal deterministic transducer recognizing* R.

Storing minimal transducers

Like minimal DFA, minimal deterministic transducers are represented as tables of nodes. However, a remark is in order: since a state of a deterministic transducer has $|\Sigma|^2$ successors, one for each letter of $\Sigma \times \Sigma$, a row of the table has $|\Sigma|^2$ entries, too large when the table is only sparsely filled. Sparse transducers over $\Sigma \times \Sigma$ are better encoded as NFAs over Σ by introducing auxiliary states: a transition $q \xrightarrow{[a,b]} q'$ of the transducer is "simulated" by two transitions $q \xrightarrow{a} r \xrightarrow{b} q'$, where r is an auxiliary state with exactly one input and one output transition.

Computing joins

Equations:

- $\emptyset \circ R = R \circ \emptyset = \emptyset;$
- $\{(\varepsilon, \varepsilon)\} \circ \{(\varepsilon, \varepsilon)\} = \{(\varepsilon, \varepsilon)\};$
- $R_1 \circ R_2 = \bigcup_{a,b,c \in \Sigma} [a, b] \cdot \left(R_1^{[a,c]} \circ R_2^{[c,b]}\right).$

Input: transducer table T, states q_1, q_2 of T

Output: state recognizing $\mathcal{L}(q_1) \circ \mathcal{L}(q_2)$

$join[T](q_1, q_2)$

  if $G(q_1, q_2)$ is not empty **then return** $G(q_1, q_2)$

  if $q_1 = q_\emptyset$ **or** $q_2 = q_\emptyset$ **then return** $q_\emptyset$

  else if $q_1 = q_\epsilon$ **and** $q_2 = q_\epsilon$ **then return** q_ϵ

  else $/ * \; q_\emptyset \neq q_1 \neq q_\epsilon, q_\emptyset \neq q_2 \neq q_\epsilon \; * /$

    for all $(a_i, a_j) \in \Sigma \times \Sigma$ **do**

      $q_{a_i,a_j} \leftarrow union[T]\left(join\left(q_1^{[a_i,a_1]}, q_2^{[a_1,a_j]}\right), \ldots, join\left(q_1^{[a_i,a_m]}, q_2^{[a_m,a_j]}\right)\right)$

    $G(q_1, q_2) = make(q_{a_1,a_1}, \ldots, q_{a_1,a_m}, \ldots, q_{a_m,a_m})$

    return $G(q_1, q_2)$

Pre and Post

Pre and Post can be reduced to intersection and projection. Define:

$$emb(L) = \{[v_1, v_2] \in (\Sigma \times \Sigma)^n \mid v_2 \in L\}$$

$$pre_S(L) = \{w_1 \in \Sigma^n \mid \exists [v_1, v_2] \in S : v_1 = w_1 \text{ and } v_2 \in L\}$$

Then we have:

$$pre_S(L) = proj_1(S \cap emb(L))$$

We use this to derive equations.

Equations:

$$\textit{if } S = \emptyset \textit{ or } L = \emptyset, \textit{ then } pre_S(L) = \emptyset;$$

$$\textit{if } S \neq \emptyset \neq L \textit{ then } pre_S(L) = \bigcup_{a,b \in \Sigma} a \cdot pre_{S[a,b]}(L^b),$$

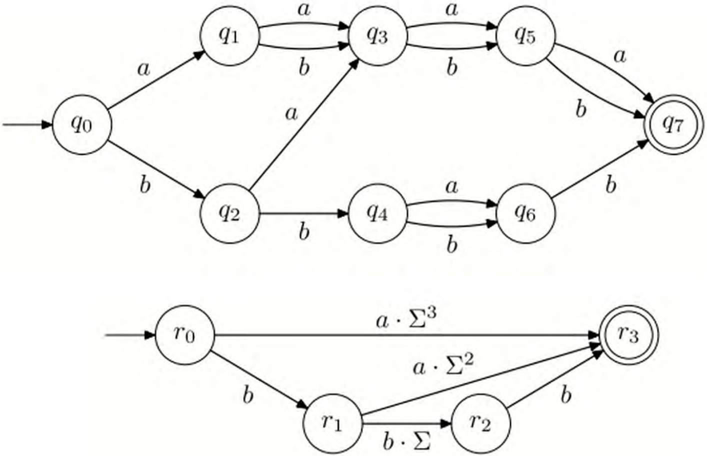
$$\textit{where } S^{[a,b]} = \{w \in (\Sigma \times \Sigma)^* \mid [a,b]w \in S\}.$$

$$
\begin{aligned}
(pre_S(L))^a &= (proj_1(S \cap emb(L)))^a \\
&= \left(proj_1 \left(\bigcup_{b \in \Sigma} [a,b] \cdot (S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \left(\bigcup_{b \in \Sigma} proj_1 \left([a,b] \cdot (S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \left(\bigcup_{b \in \Sigma} a \cdot proj_1 \left((S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \bigcup_{b \in \Sigma} proj_1 \left((S \cap emb(L))^{[a,b]} \right) \\
&= \bigcup_{b \in \Sigma} proj_1 \left(S^{[a,b]} \cap emb \left(L^b \right) \right) \\
&= \bigcup_{b \in \Sigma} pre_{S^{[a,b]}}(L^b)
\end{aligned}
$$

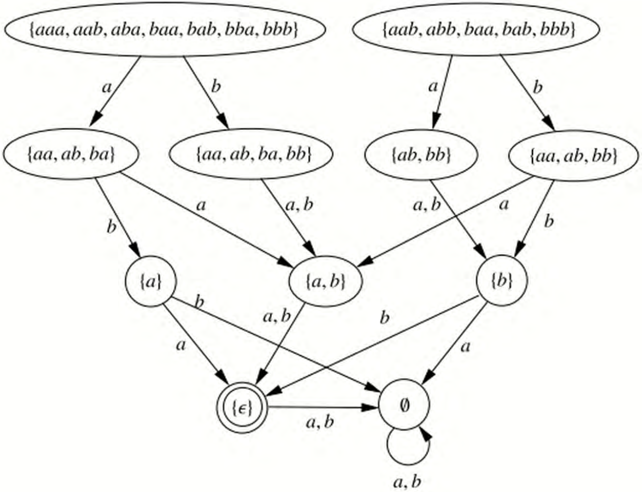
Input: transducer table TT, table T, state r of TT, state q of T
Output: state of T recognizing $pre_{\mathcal{L}(r)}(\mathcal{L}(q))$

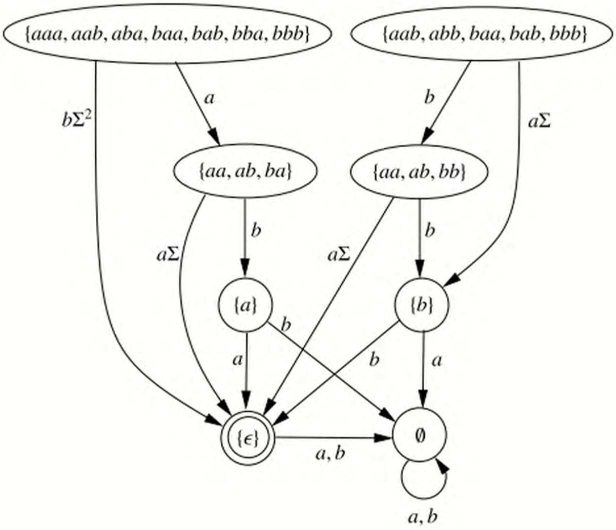
$pre[TT, T](r, q)$
  if $G(r, q)$ is not empty **then return** $G(r, q)$
  if $r = r_\emptyset$ **or** $q = q_\emptyset$ **then return** $q_\emptyset$
  else if $r = r_\epsilon$ **and** $q = q_\epsilon$ **then return** q_ϵ
  else
    for all $a_i \in \Sigma$ **do**
      $q_{a_i} \leftarrow union\left(pre[TT, T]\left(q^{[a_i, a_1]}, r^{a_1}\right), \ldots, pre[TT, T]\left(q^{[a_i, a_m]}, r^{a_m}\right)\right)$
    $G(r, q) \leftarrow make(q_{a_1}, \ldots, q_{a_m})$;
    return $G(r, q)$

Binary Decision



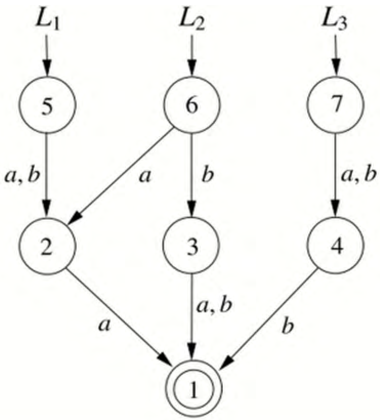
The master z-automaton

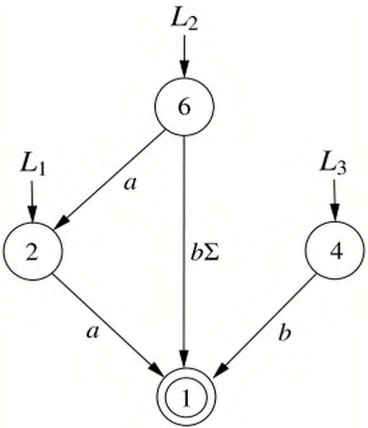




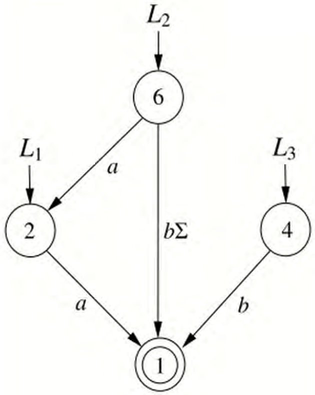
$\{aaa, aab, aba, baa, bab, bba, bbb\}$
$\{aab, abb, baa, bab, bbb\}$
$b\Sigma^2$
a
b
$a\Sigma$
$\{aa, ab, ba\}$
$\{aa, ab, bb\}$
$a\Sigma$
b
$a\Sigma$
b
$\{a\}$
$\{b\}$
b
a
b
a
$\{\epsilon\}$
$\emptyset$
a, b
a, b

Length: 2

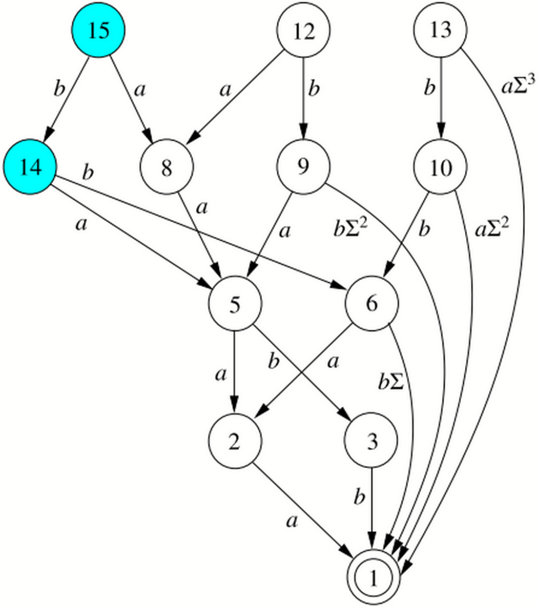


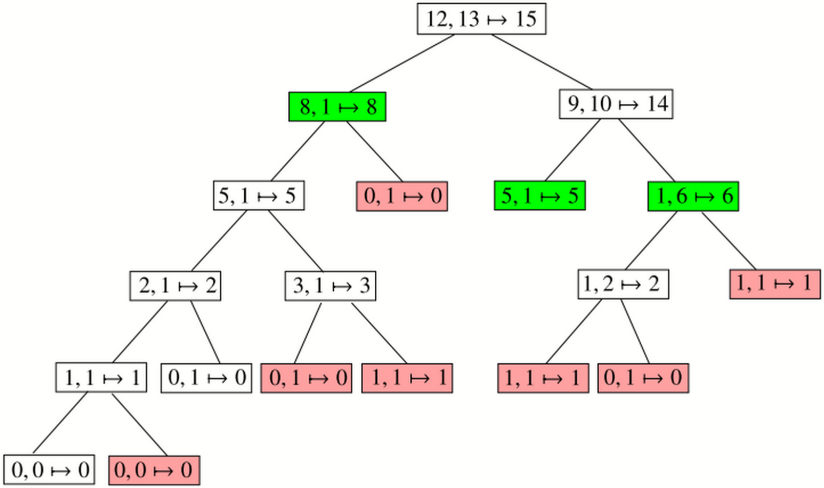


Data structure for z-automata



Ident.	Length	a-succ	b-succ
1	0	0	0
2	1	1	0
4	1	0	1
6	2	2	1


15
12
13
b
a
a
b
b
$a\Sigma^3$
14
8
9
10
b
a
a
a
$b\Sigma^2$
b
$a\Sigma^2$
5
6
a
b
a
$b\Sigma$
2
3
a
b
1



12, 13 ↦ 15
8, 1 ↦ 8
9, 10 ↦ 14
5, 1 ↦ 5
0, 1 ↦ 0
5, 1 ↦ 5
1, 6 ↦ 6
2, 1 ↦ 2
3, 1 ↦ 3
1, 2 ↦ 2
1, 1 ↦ 1
1, 1 ↦ 1
0, 1 ↦ 0
0, 1 ↦ 0
1, 1 ↦ 1
1, 1 ↦ 1
0, 1 ↦ 0
0, 0 ↦ 0
0, 0 ↦ 0

8. Verification

- We use languages to describe the implementation and specification of a system.
- We reduce the verification problem to language inclusion between implementation and specification.

```
1  while x = 1 do
2      if y = 1 then
3          x ← 0
4      y ← 1 − x
5  end
```

- Configuration: triple $[l, n_x, n_y]$ where
  - l is the current value of the program counter, and
  - n_x, n_y are the current values of x, y

  Examples: [0,1,1], [5,0,1]

- Initial configuration: configuration with $l = 1$

- Potential execution: finite or infinite sequence of configurations

  Examples: [0,1,1][4,1,0]

  [2,1,0][5,1,0]

  [1,1,0][2,1,0][4,1,0][1,1,0]

```
while x = 1 do
    if y = 1 then
        x ← 0
    y ← 1 − x
end
```

- Execution: potential execution starting at an initial configuration, and where configurations are followed by their „legal successors“ according to the program semantics.

  Examples: [1,1,1][2,1,1][3,1,1][4,0,1][1,0,1][5,0,1]
  [1,1,0][2,1,0][4,1,0][1,1,0]

- Full execution: execution that cannot be extended (either infinite or ending at a configuration without successors)

Verification as a language problem

- Implementation: set E of executions
- Specification:
  - subset P of the potential executions that satisfy a property , or
  - subset V of the potential executions that violate a property
- Implementation satisfies specification if :
  - $E \subseteq P$, or
  - $E \cap V = \emptyset$.
- If E and P regular: inclusion checkable with automata
- If E and V regular: disjointness checkable with automata

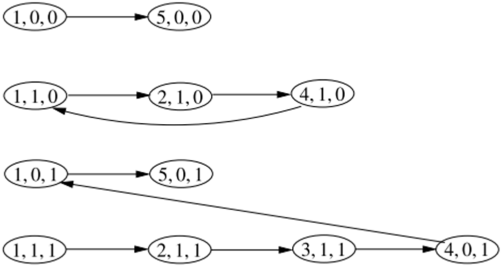
Verification as a language problem

- Implementation: set E of executions
- Specification:
  - subset P of the potential executions that satisfy a property , or
  - subset V of the potential executions that violate a property
- Implementation satisfies specification if :
  - $E \subseteq P$, or
  - $E \cap V = \emptyset$.
- If E and P regular: inclusion checkable with automata
- If E and V regular: disjointness checkable with automata

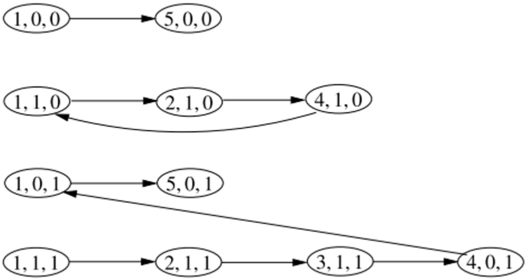
- How often is the case that E, P, V are regular?

System NFA

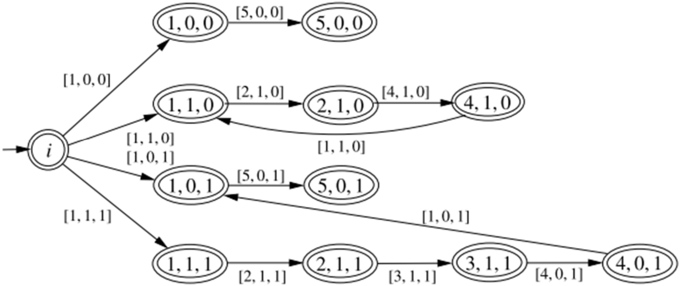
```
while x = 1 do
    if y = 1 then
        x ← 0
    y ← 1 − x
end
```



System NFA



System NFA



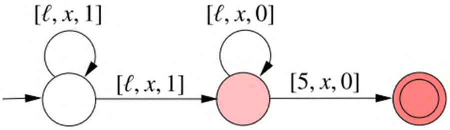
1, 0, 0
[5, 0, 0]
5, 0, 0
[1, 0, 0]
1, 1, 0
[2, 1, 0]
2, 1, 0
[4, 1, 0]
4, 1, 0
i
[1, 1, 0]
[1, 0, 1]
[1, 1, 0]
1, 0, 1
[5, 0, 1]
5, 0, 1
[1, 1, 1]
[1, 0, 1]
1, 1, 1
[2, 1, 1]
2, 1, 1
[3, 1, 1]
3, 1, 1
[4, 0, 1]
4, 0, 1

Property NFA

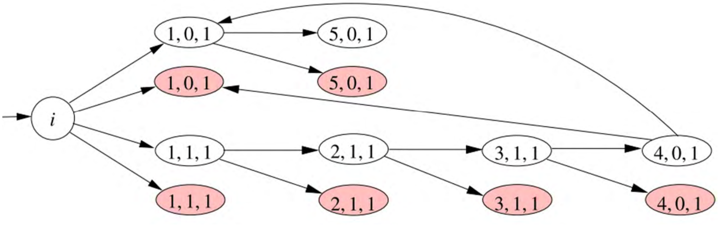
- Is there a full execution such that
  - initially $y = 1$,
  - finally $y = 0$, and
  - y never increases?
- Set of potential executions for this property:

$$[l, x, 1][l, x, 1]^* \ [l, x, 0]^* \ [5, x, 0]$$

- Automaton for this set:



Intersection of the system and property NFAs



- Automaton is empty, and so no execution satisfies the property

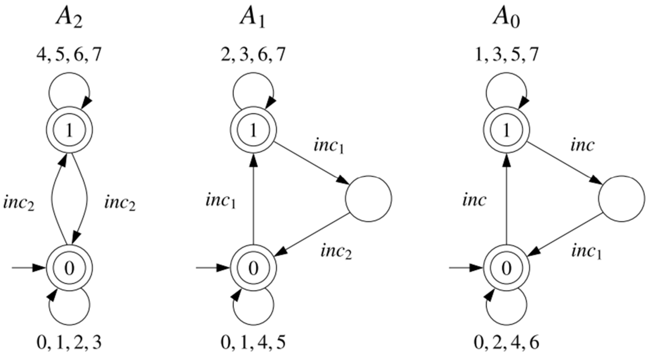
Another property

- Is the assignment $y \leftarrow x - 1$ redundant?
- Potential executions that use the assignment:

$$[l, x, y]^*([4, x, 0][1, x, 1] + [4, x, 1][1, x, 0])\ [l, x, y]^*$$

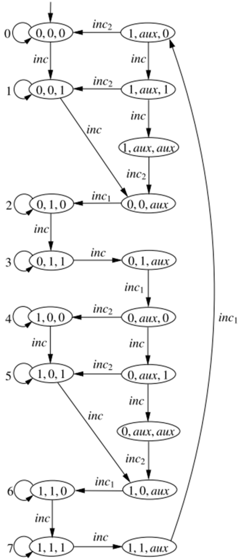
- Therefore: assignment redundant iff none of these potential executions is a real execution of the program.

Networks of automata



- Tuple $\mathcal{A} = \langle A_1, \dots, A_n \rangle$ of NFAs .
- Each NFA has its own alphabet Σ_i of actions
- Alphabets usually not disjoint!
- A_i participates in action a if $a \in \Sigma_i$.
- A configuration is a tuple $\langle q_1, \dots, q_n \rangle$ of states, one for each automaton of the network.
- $\langle q_1, \dots, q_n \rangle$ enables a if every participant in a is in a state from which an a-transition is possible.
- Enabled actions can occur, and their occurrence simultaneously changes the states of their participants. Non-participants stay idle and don‘t change their states.

Configuration graph of the network



AsyncProduct$(A_1, \ldots, A_n)$

Input: a network of automata $\mathcal{A} = A_1, \ldots A_n$, where
$A_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_1), \ldots, A_n = (Q_n, \Sigma_n, \delta_n, q_{0n}, Q_n)$

Output: the asynchronous product $A_1 \otimes \cdots \otimes A_n = (Q, \Sigma, \delta, q_0, F)$

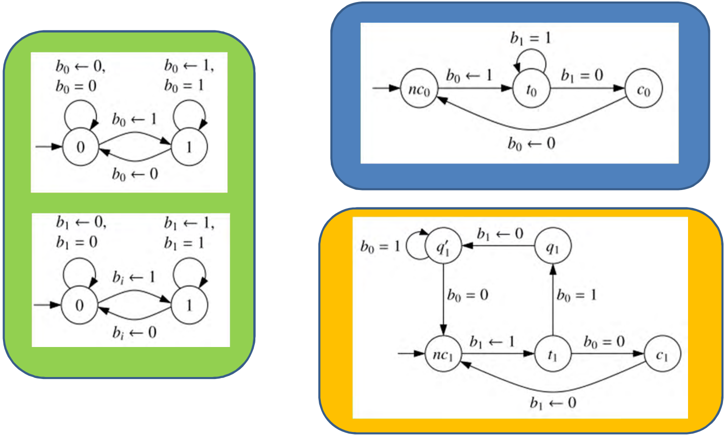
$Q, \delta, F \leftarrow \emptyset$
$q_0 \leftarrow [q_{01}, \ldots, q_{0n}]$
$W \leftarrow \{[q_{01}, \ldots, q_{0n}]\}$
while $W \neq \emptyset$ **do**
  pick $[q_1, \ldots, q_n]$ **from** W
  add $[q_1, \ldots, q_n]$ **to** Q
  add $[q_1, \ldots, q_n]$ **to** F
  for all $a \in \Sigma_1 \cup \ldots \cup \Sigma_n$ **do**
    for all $i \in [1..n]$ **do**
      if $a \in \Sigma_i$ **then** $Q'_i \leftarrow \delta_i(q_i, a)$ **else** $Q'_i = \{q_i\}$
    for all $[q'_1, \ldots, q'_n] \in Q'_1 \times \ldots \times Q'_n$ **do**
      if $[q'_1, \ldots, q'_n] \notin Q$ **then add** $[q'_1, \ldots, q'_n]$ **to** W
      add $([q_1, \ldots, q_n], a, [q'_1, \ldots, q'_n])$ **to** δ
return $(Q, \Sigma, \delta, q_0, F)$

Concurrent programs as networks of automata: Lamport's 1-bit algorithm (JACM86)

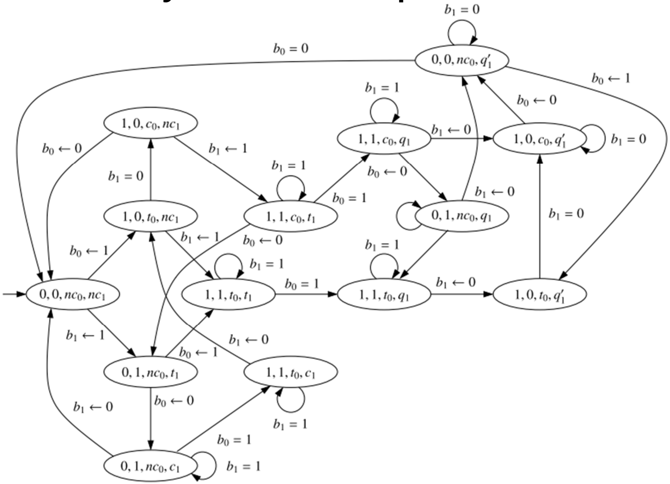
Shared variables: b[1], ..., b[n] ∈ {0,1}, initially 0
Process i ∈ {1, ...,n}

```
repeat forever
      noncritical section
   T: b[i]:=1
      for j ∈ {1, ...,i-1}
         if b[j]=1 then  b[i]:=0
                         await ¬b[j]
                         goto T
      for j ∈ {i+1, ...,N}  await ¬b[j]
      critical section
      b[i]:=0
```

Network for the two-process case



Asynchronous product



Checking properties of the algorithm

- Deadlock freedom: every configuration has at least one successor.
- Mutual exclusion: no configuration of the form $[b_0, b_1, c_0, c_1]$ is reachable
- Bounded overtaking (for process 0): after process 0 signals interest in accessing the critical section, process 1 can enter the critical section at most one before process 0 enters.
  - Let NC_i, T_i, C_i be the configurations in which process i is non-critical, trying, or critical
  - Set of potential executions violating the property:

$$\Sigma^* \; T_0 \; (\Sigma \setminus C_0)^* \; C_1 \; (\Sigma \setminus C_0)^* \; NC_1 \; (\Sigma \setminus C_0)^* \; C_1 \; \Sigma^*$$

CheckViol$(A_1, \ldots, A_n, V)$

Input: a network $\langle A_1, \ldots A_n \rangle$, where $A_i = (Q_i, \Sigma_i, \delta_i, q_{0i}, Q_i)$;
an NFA $V = (Q_V, \Sigma_1 \cup \ldots \cup \Sigma_n, \delta_V, q_{0v}, F_v)$.

Output: **true** if $A_1 \otimes \cdots \otimes A_n \otimes V$ is nonempty, **false** otherwise.

$Q \leftarrow \emptyset$; $q_0 \leftarrow [q_{01}, \ldots, q_{0n}, q_{0v}]$
$W \leftarrow \{q_0\}$
while $W \neq \emptyset$ **do**
    pick $[q_1, \ldots, q_n, q]$ **from** W
    add $[q_1, \ldots, q_n, q]$ **to** Q
    for all $a \in \Sigma_1 \cup \ldots \cup \Sigma_n$ **do**
        for all $i \in [1..n]$ **do**
            if $a \in \Sigma_i$ **then** $Q'_i \leftarrow \delta_i(q_i, a)$ **else** $Q'_i = \{q_i\}$
        $Q' \leftarrow \delta_V(q, a)$
        for all $[q'_1, \ldots, q'_n, q'] \in Q'_1 \times \ldots \times Q'_n \times Q'$ **do**
            if $\bigwedge_{i=1}^{n} q'_i \in F_i$ **and** $q \in F_v$ **then return true**
            if $[q'_1, \ldots, q'_n, q'] \notin Q$ **then add** $[q'_1, \ldots, q'_n, q']$ **to** W
return false

The state-explosion problem

- In sequential programs, the number of reachable configurations grows exponentially in the number of variables.
- Proposition: The following problem is PSPACE-complete.
  - Given: a boolean program π (program with only boolean variables), and a NFA A_V recognizing a set of potential executions
  - Decide: Is $E_\pi \cap L(A_V)$ empty?

The state-explosion problem

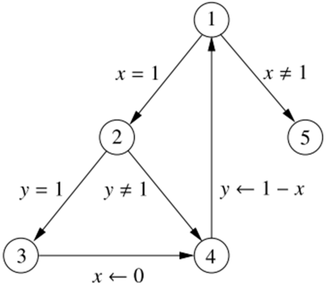
- In concurrent programs, the number of reachable configurations also grows exponentially in the number of components.
- Proposition: The following problem is PSPACE-complete.
  - Given: a network of automata $\mathcal{A} = \langle A_1, \ldots, A_n \rangle$ and a NFA A_V recognizing a set of potential executions of $\mathcal{A}$
  - Decide: Is $L(A_1 \otimes \cdots \otimes A_n \otimes A_V) = \emptyset$?

Symbolic exploration

- A technique to palliate the state-explosion problem
- Configurations can be encoded as words.
- The set of reachable configurations of a program can be encoded as a language.
- We use automata to compactly store the set of reachable configurations.

Flowgraphs

while $x = 1$ **do**
  if $y = 1$ **then**
    $x \leftarrow 0$
  $y \leftarrow 1 - x$
end



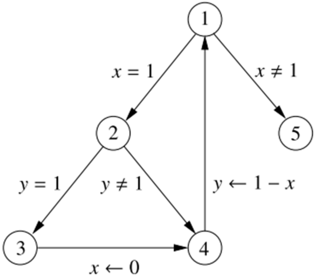
Step relations

- Let l, l' be two control points of a flowgraph.
- The step relation $S_{l,l'}$ contains all pairs

$$([l, x_0, y_0], [l', x_0', y_0'])$$

of configurations such that :

if at point l the current values of x, y are x_0, y_0,
then the program can take a step,
after which the new control point is l', and the new values of x, y are x_0', y_0'.



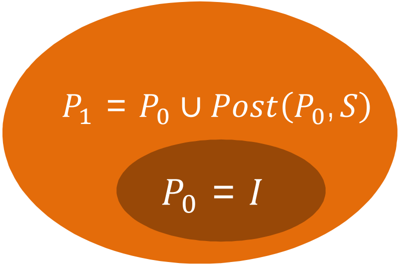
$$S_{4,1} = \{ \, (\, [4, x_0, y_0], [1, x_0, 1 - x_0] \,) \mid x_0, y_0 \in \{0,1\} \, \}$$

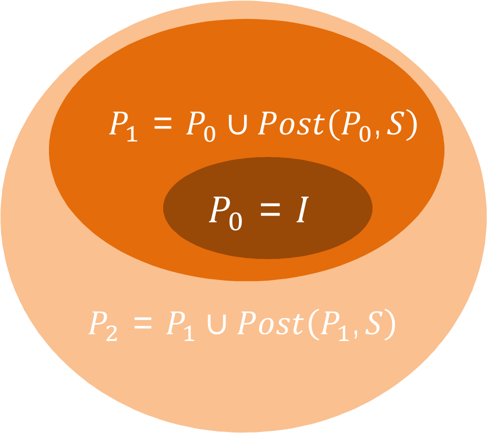
- The global step relation S is the union of the step relations $S_{l,l'}$ for all pairs l, l' of control points.

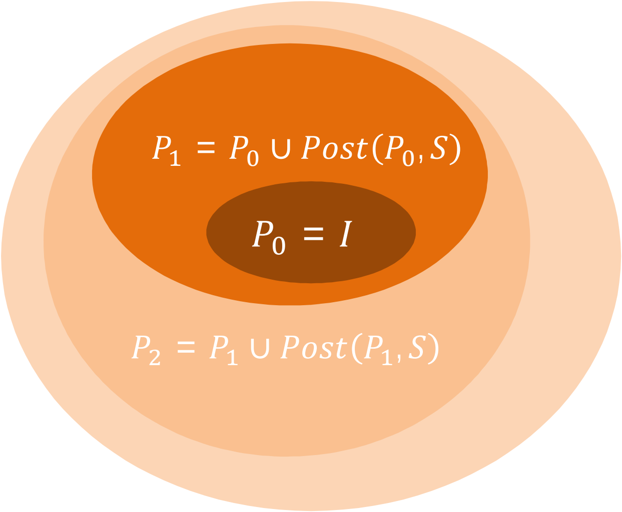
Computing reachable configurations

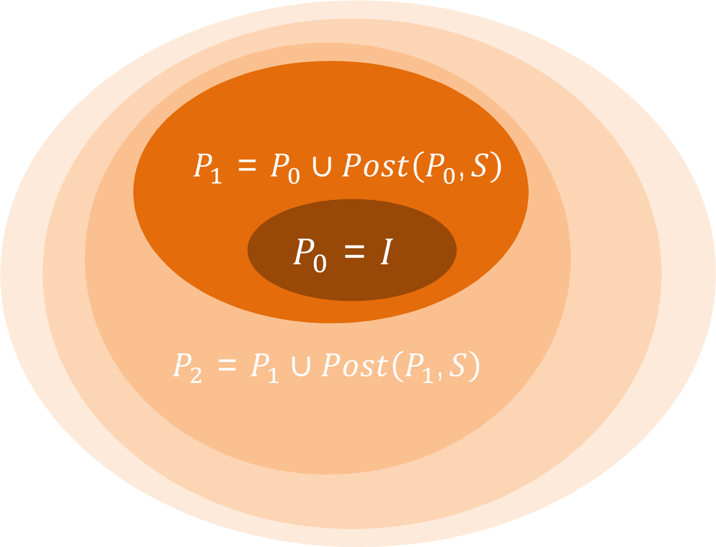
- Start with the set of initial configurations.
- Iteratively: add the set of successors of the current set of configurations until a fixed point is reached.

$$P_0 = I$$


$P_1 = P_0 \cup Post(P_0, S)$
$P_0 = I$


$P_1 = P_0 \cup Post(P_0, S)$
$P_0 = I$
$P_2 = P_1 \cup Post(P_1, S)$


$P_1 = P_0 \cup Post(P_0, S)$
$P_0 = I$
$P_2 = P_1 \cup Post(P_1, S)$



$P_1 = P_0 \cup Post(P_0, S)$
$P_0 = I$
$P_2 = P_1 \cup Post(P_1, S)$

$Reach(I, R)$

Input: set I of initial configurations; relation R

Output: set of configurations reachable form I

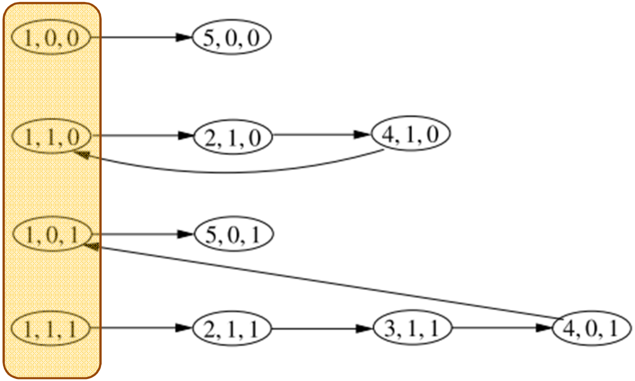
$OldP \leftarrow \emptyset$; $P \leftarrow I$

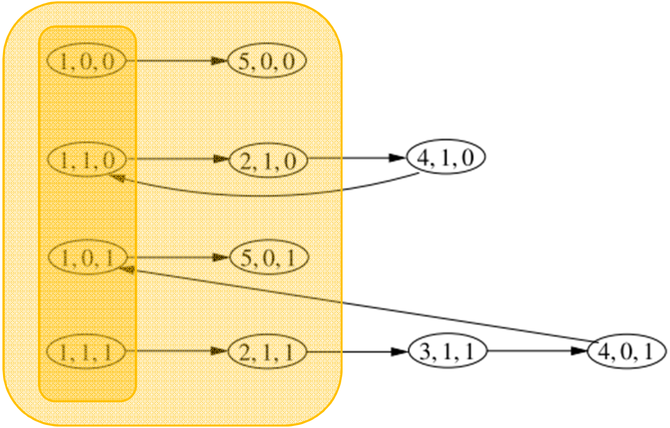
while $P \neq OldP$ **do**

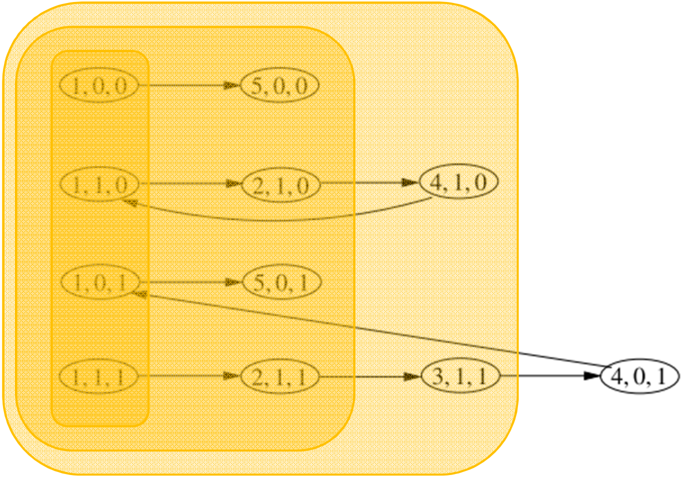
    $OldP \leftarrow P$

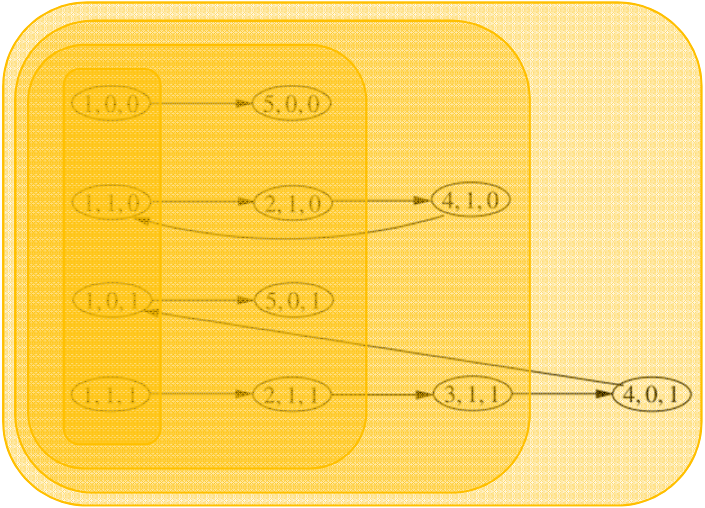
    $P \leftarrow$ **Union**$(P,$ **Post**$(P, S))$

return P


1, 0, 0
5, 0, 0
1, 1, 0
2, 1, 0
4, 1, 0
1, 0, 1
5, 0, 1
1, 1, 1
2, 1, 1
3, 1, 1
4, 0, 1

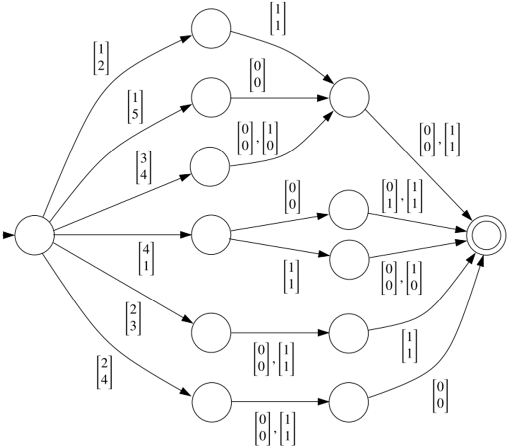

1, 0, 0
5, 0, 0
1, 1, 0
2, 1, 0
4, 1, 0
1, 0, 1
5, 0, 1
1, 1, 1
2, 1, 1
3, 1, 1
4, 0, 1


1, 0, 0
5, 0, 0
1, 1, 0
2, 1, 0
4, 1, 0
1, 0, 1
5, 0, 1
1, 1, 1
2, 1, 1
3, 1, 1
4, 0, 1


1, 0, 0
5, 0, 0
1, 1, 0
2, 1, 0
4, 1, 0
1, 0, 1
5, 0, 1
1, 1, 1
2, 1, 1
3, 1, 1
4, 0, 1

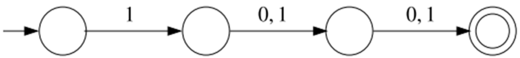
Example: Transducer for the global step relation

```
while x = 1 do
    if y = 1 then
        x ← 0
    y ← 1 − x
end
```

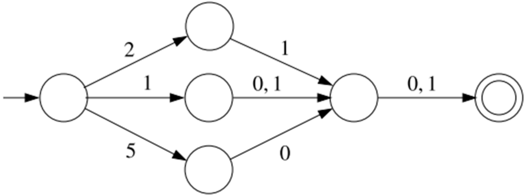


Example: DFAs generated by Reach

- Initial configurations

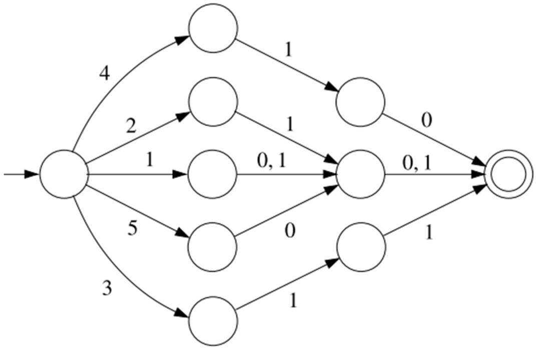


- Configurations reachable in at most 1 step



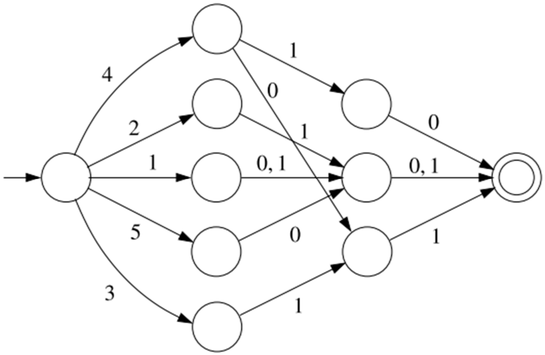
Example: DFAs generated by Reach

- Configurations reachable in at most 2 steps



Example: DFAs generated by Reach

- Configurations reachable in at most 3 steps



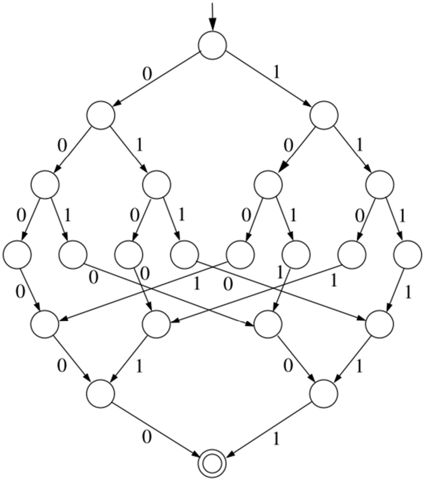
Variable orders

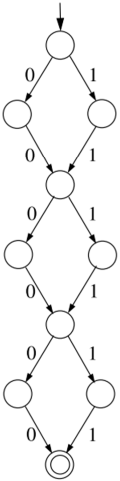
- Consider the set Y of tuples $[x_1, \ldots, x_{2k}]$ of booleans such that
$$x_1 = x_{k+1}, x_2 = x_{k+2}, \ldots, x_k = x_{2k}$$
- A tuple $[x_1, \ldots, x_{2k}]$ can be encoded by the word $x_1 x_2 \ldots x_{2k-1} x_{2k}$ but also by the word $x_1 x_{k+1} \ldots x_k x_{2k}$.
- For $k = 3$, the encodings of Y are then, respectively

$$\{000000, 001001, 010010, 011011, 100100, 101101, 110110, 111111\}$$

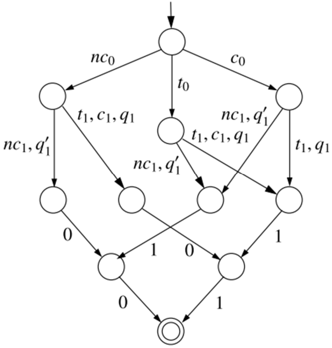
$$\{000000, 000011, 001100, 001111, 110000, 110011, 111100, 111111\}$$

- The minimal DFAs for these languages have very different sizes!

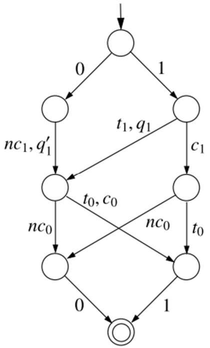

0 1
0 1 0 1
0 1 0 1 0 1 0 1
0 0 0 1 0 1 1 1
0 1 0 1
0 1


0 1
0 1
0 1
0 1
0 1
0 1

Another example: Lamport's algorithm

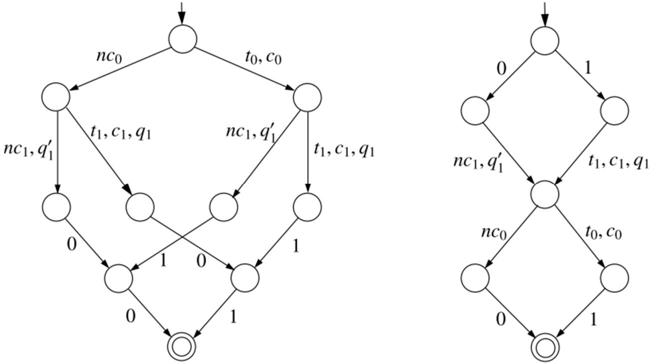


$\langle v_0, v_1, s_0, s_1 \rangle$ encoded by $s_0 s_1 v_0 v_1$



$\langle v_0, v_1, s_0, s_1 \rangle$ encoded by $v_1 s_1 s_0 v_0$

Larger sets can yield smaller DFAs!



- DFAs after adding the configuration $\langle c_0, c_1, 1{,}1 \rangle$ to the set

- When encoding configurations, good variable orders can lead to much smaller automata.
- Unfortunately, the problem of finding an optimal encoding for a language represented by a DFA is NP-complete.

9. Automata and Monadic Second-Order Logic

Logics on words

- Regular expressions give operational descriptions of regular languages.
- Often the natural description of a language is declarative:
  - **even number of a's and even number of b's** vs.
    $\big(aa + bb + (ab + ba)(aa + bb)^*(ba + ab)\big)^*$
  - **words not containing 'hello'**
- Goal: find a declarative language able to express all the regular languages, and only the regular languages.

Logics on words

- Idea: use a logic that has an interpretation on words
- A formula expresses a property that each word may satisfy or not, like
  - **the word contains only a's**
  - **the word has even length**
  - **between every occurrence of an a and a b there is an occurrence of a c**
- Every formula (indirectly) defines a language: the language of all the words over the given fixed alphabet that satisfy it.

First-order logic on words

- Atomic formulas: for each letter a we introduce the formula $Q_a(x)$, with intuitive meaning: **the letter at position x is an a.**

First-order logic on words: Syntax

- Formulas constructed out of atomic formulas by means of standard “logic machinery”:
  - Alphabet $\Sigma = \{a, b, \dots\}$ and position variables $V = \{x, y, \dots\}$
  - $Q_a(x)$ is a formula for every $a \in \Sigma$ and $x \in V$.
  - $x < y$ is a formula for every $x, y \in V$
  - If $\varphi, \varphi_1, \varphi_2$ are formulas then so are $\neg\varphi$ and $\varphi_1 \vee \varphi_2$
  - If φ is a formula then so is $\exists x\ \varphi$ for every $x \in V$

Abbreviations

$$\varphi_1 \wedge \varphi_2 := \neg(\neg\varphi_1 \vee \neg\varphi_2)$$

$$\varphi_1 \rightarrow \varphi_2 := \neg\varphi_1 \vee \varphi_2$$

$$\forall x\, \varphi := \neg\exists\, x \neg\varphi$$

$$\begin{aligned}
\text{first}(x) &:= \\
\text{last}(x) &:= \\
y = x + 1 &:= \\
y = x + 2 &:= \\
y = x + (k+1) &:=
\end{aligned}$$

Examples (without semantics yet)

- "The last letter is a b and before it there are only a's."

$$\exists x\, Q_b(x) \wedge \forall x\, (\mathrm{last}(x) \rightarrow Q_b(x) \;\wedge\; \neg\mathrm{last}(x) \rightarrow Q_a(x))$$

- "Every a is immediately followed by a b."

$$\forall x\, (Q_a(x) \rightarrow \exists y\, (y = x + 1 \wedge Q_b(y)))$$

- "Every a is immediately followed by a b, unless it is the last letter."

$$\forall x\, (Q_a(x) \rightarrow \forall y\, (y = x + 1 \rightarrow Q_b(y)))$$

- "Between every a and every later b there is a c."

$$\forall x \forall y\, (Q_a(x) \wedge Q_b(y) \wedge x < y \rightarrow \exists z\, (x < z \wedge z < y \wedge Q_c(z)))$$

First-order logic on words: Semantics

- Formulas are interpreted on pairs $(w, \mathcal{J})$ called interpretations, where
  - w is a word, and
  - $\mathcal{J}$ assigns positions to the free variables of the formula (and maybe to others too—who cares)
- It does not make sense to say a formula is true or false: it can only be true or false for a given interpretation.
- If the formula has no free variables (if it is a sentence), then for each word it is either true or false.

- Satisfaction relation:

$$
\begin{array}{lllll}
(w,\mathfrak{J}) & \models & Q_a(x) & \textit{iff} & w[\mathfrak{J}(x)] = a \\
(w,\mathfrak{J}) & \models & x < y & \textit{iff} & \mathfrak{J}(x) < \mathfrak{J}(y) \\
(w,\mathfrak{J}) & \models & \neg\varphi & \textit{iff} & (w,\mathfrak{J}) \not\models \varphi \\
(w,\mathfrak{J}) & \models & \varphi 1 \vee \varphi_2 & \textit{iff} & (w,\mathfrak{J}) \models \varphi_1 \textit{ or } (w,\mathfrak{J}) \models \varphi_2 \\
(w,\mathfrak{J}) & \models & \exists x\, \varphi & \textit{iff} & |w| \geq 1 \textit{ and some } i \in \{1,\ldots,|w|\} \textit{ satisfies } (w,\mathfrak{J}[i/x]) \models \varphi
\end{array}
$$

- More logic jargon:
  - A formula is valid if it is true for all its interpretations
  - A formula is satisfiable if is is true for at least one of its interpretations

The empty word ...

- ... is as usual a pain in the eh, neck.
- It satisfies all universally quantified formulas, and no existentially quantified formula.

Can we only express regular languages? Can we express all regular languages?

- The language $L(\varphi)$ of a sentence φ is the set of words that satisfy φ.
- A language L is expressible in first-order logic or FO-definable if some sentence φ satisfies $L(\varphi) = L$.
- Proposition: a language over a one-letter alphabet is expressible in first-order logic iff it is finite or co-finite (its complement is finite).
- Consequence: we can only express regular languages, but not all, not even the language of words of even length.

Proof sketch

1. If L is finite, then it is FO-definable

2. If L is co-finite, then it is FO-definable.

Proof sketch

3. If L is FO-definable (over a one-letter alphabet), then it is finite or co-finite.
   1) We define a new logic QF (quantifier-free fragment)
   2) We show that a language is QF-definable iff it is finite or co-finite
   3) We show that a language is QF-definable iff it FO-definable.

1) The logic QF

- $x < k$ $x > k$
  $x < y + k$ $x > y + k$
  $k < last$ $k > last$
  are formulas for every variable x, y and every $k \geq 0$.
- If f_1, f_2 are formulas, then so are $f_1 \vee f_2$ and $f_1 \wedge f_2$

2) L is QF-definable iff it is finite or co-finite

($\rightarrow$) Let f be a sentence of QF.

Then f is an and-or combination of formulas $k < last$ and $k > last$.

- $L(k < last) = \{k+1, k+2, \ldots\}$ is co-finite (we identify words and numbers)
- $L(k > last) = \{0,1,\ldots,k\}$ is finite
- $L(f_1 \vee f_2) = L(f_1) \cup L(f_2)$ and so if $L(f)$ and $L(g)$ finite or co-finite the L is finite or co-finite.
- $L(f_1 \wedge f_2) = L(f_1) \cap L(f_2)$ and so if $L(f)$ and $L(g)$ finite or co-finite the L is finite or co-finite.

2) L is QF-definable iff it is finite or co-finite

(←) If $L = \{k_1, \ldots, k_n\}$ is finite, then

$$(k_1 - 1 < last \wedge last < k_1 + 1) \vee \cdots \vee$$
$$(k_n - 1 < last \wedge last < k_n + 1)$$

expresses L.

If L is co-finite, then its complement is finite, and so expressed by some formula. We show that for every f some formula $neg(f)$ expresses $\overline{L(f)}$

- $neg(k < last) = (k - 1 < last \wedge last < k + 1)$
  $\vee\ last < k$
- $neg(f_1 \vee f_2) = neg(f_1) \wedge neg(f_2)$
- $neg(f_1 \wedge f_2) = neg(f_1) \vee neg(f_2)$

3) Every first-order formula φ has an equivalent QF-formula $QF(\varphi)$

- $QF(x < y) = x < y + 0$
- $QF(\neg\varphi) = neg(QF(\varphi))$
- $QF(\varphi_1 \lor \varphi_2) = QF(\varphi_1) \lor QF(\varphi_2)$
- $QF(\varphi_1 \land \varphi_2) = QF(\varphi_1) \land QF(\varphi_2)$
- $QF(\exists x\, \varphi) = QF(\exists x\, QF(\varphi))$
  - If $QF(\varphi)$ disjunction, apply $\exists x\, (\varphi_1 \lor \ldots \lor \varphi_n) = \exists x\, \varphi_1 \lor \ldots \lor \exists x\, \varphi_n$
  - If $QF(\varphi)$ conjunction (or atomic formula), see example in the next slide.

- Consider the formula

$$\exists x \quad x < y + 3 \quad \land$$
$$z < x + 4 \quad \land$$
$$z < y + 2 \quad \land$$
$$y < x + 1$$

- The equivalent QF-formula is

$$z < y + 8 \;\land\; y < y + 5 \;\land\; z < y + 2$$

Monadic second-order logic

- First-order variables: interpreted on positions
- Monadic second-order variables: interpreted on sets of positions.
  - Diadic second-order variables: interpreted on relations over positions
  - Monadic third-order variables: interpreted on sets of sets of positions
  - New atomic formulas: $x \in X$

Expressing „even length"

- Express

  There is a set X of positions such that

  – X contains exactly the even positions, and

  – the last position belongs to X.

- Express

  X contains exactly the even positions

  as

A position is in X iff it is second position or the second successor of another position of X

Syntax and semantics of MSO

- New set $\{X, Y, Z, \dots\}$ of second-order variables
- New syntax: $x \in X$ and $\exists x \ \varphi$
- New semantics:
  - Interpretations now also assign sets of positions to the free second-order variables.
  - Satisfaction defined as expected.

Expressing $c^*(ab)^*d^*$

- Express:

There is a block X of consecutive positions such that

  - **before X there are only c's;**
  - **after X there are only b's;**
  - **a's and b's alternate in X;**
  - **the first letter in X is an a, and the last is a b.**

- Then we can take the formula

$$\exists X\ (Cons(X)\ \wedge Boc(X) \wedge Aod(X) \wedge \mathrm{Alt}(X) \wedge Fa(X) \wedge Lb(X)\ )$$

- X is a block of consecutive positions

- Before X there are only c's

- In X a's and b's alternate

Every regular language is expressible in MSO logic

- Goal: given an arbitrary regular language L, construct an MSO sentence φ such having $L = L(\varphi)$.
- We use: if L is regular, then there is a DFA A recognizing L.
- Idea: construct a formula expressing

  the run of A on this word is accepting

- Fix a regular language L.
- Fix a DFA A with states $q_0, \dots, q_n$ recognizing L.
- Fix a word $w = a_1 a_2 \dots a_m$.
- Let P_q be the set of positions i such that after reading $a_1 a_2 \dots a_i$ the automaton A is in state q.
- We have:

  A accepts w iff $m \in P_q$ for some final state q.

- Assume we can construct a formula

$$Visits(X_0, \ldots, X_n)$$

which is true for $(w, \mathcal{J})$ iff

$$\mathcal{J}(X_0) = P_{q_0}, \ldots, \mathcal{J}(X_n) = P_{q_n}$$

- Then $(w, \mathcal{J})$ satisfies the formula

$$\psi_A := \exists X_0 \ldots \exists X_n \; \text{Visits}(X_0, \ldots X_n) \wedge \exists x \left(\text{last}(x) \wedge \bigvee_{q_i \in F} x \in X_i \right)$$

iff w has a last letter and $w \in L$, and we easily get a formula expressing L .

- To construct $Visits(X_0, \ldots, X_n)$ we observe that the sets P_q are the unique sets satisfying
  a) $1 \in P_{\delta(q_0, a_1)}$ i.e., after reading the first letter the DFA is in state $\delta(q_0, a_1)$.
  b) The sets P_q build a partition of the set of positions, i.e., the DFA is always in exactly one state.
  c) If $i \in P_q$ and $\delta(q, a_{i+1}) = q'$ then $i + 1 \in P_{q'}$, i.e., the sets „match“ δ.
- We give formulas for a) , b), and c)

- Formula for a)

$$\text{Init}(X_0,\ldots,X_n) = \exists x \left(\text{first}(x) \wedge \left(\bigvee_{a\in\Sigma} (Q_a(x) \wedge x \in X_{i_a}) \right) \right)$$

- Formula for b)

$$\text{Partition}(X_0,\ldots,X_n) = \forall x \left(\bigvee_{i=0}^{n} x \in X_i \;\wedge \bigwedge_{\substack{i,j=0 \\ i\neq j}}^{n} (x \in X_i \rightarrow x \notin X_j) \right)$$

- Formula for c)

$$\text{Respect}(X_0, \ldots, X_n) =$$

$$\forall x \forall y \left(y = x + 1 \rightarrow \bigvee_{\substack{a \in \Sigma \\ i, j \in \{0, \ldots, n\} \\ \delta(q_i, a) = q_j}} (x \in X_i \land Q_a(x) \land y \in X_j) \right)$$

- Together:

$$\begin{aligned}\text{Visits}(X_0, \ldots X_n) := {} & \text{Init}(X_0, \ldots, X_n) \land \\ & \text{Partition}(X_0, \ldots, X_n) \land \\ & \text{Respect}(X_0, \ldots, X_n)\end{aligned}$$

Every language expressible in MSO logic is regular

- Recall: an interpretation of a formula is a pair $(w, \mathcal{I})$ consisting of a word w and assignments $\mathcal{I}$ to the free first and second order variables (and perhaps to others).

$$\left(aab\ ,\ \begin{array}{l} x \mapsto 1 \\ y \mapsto 3 \\ X \mapsto \{2,3\} \\ Y \mapsto \{1,2\} \end{array} \right) \qquad \left(ba\ ,\ \begin{array}{l} x \mapsto 2 \\ y \mapsto 1 \\ X \mapsto \emptyset \\ Y \mapsto \{1\} \end{array} \right)$$

- We encode interpretations as words.

$$\left(aab\,,\ \begin{array}{l} x \mapsto 1 \\ y \mapsto 3 \\ X \mapsto \{2,3\} \\ Y \mapsto \{1,2\} \end{array} \right) \qquad \left(ba\,,\ \begin{array}{l} x \mapsto 2 \\ y \mapsto 1 \\ X \mapsto \emptyset \\ Y \mapsto \{1\} \end{array} \right)$$

$$\begin{array}{cccc} & a & a & b \\ x & 1 & 0 & 0 \\ y & 0 & 0 & 1 \\ X & 0 & 1 & 1 \\ Y & 1 & 1 & 0 \end{array} \qquad \begin{array}{ccc} & b & a \\ x & 0 & 1 \\ y & 1 & 0 \\ X & 0 & 0 \\ Y & 1 & 0 \end{array}$$

- Given a formula with n free variables, we encode an interpretation $(w, \mathcal{I})$ as a word $enc(w, \mathcal{I})$ over the alphabet $\Sigma \times \{0,1\}^n$.
- The language of the formula φ , denoted by $L(\varphi)$, is given by

$$L(\varphi)=\{enc(w, \mathcal{I}) \mid (w, \mathcal{I}) \models \varphi\}$$

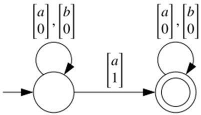
- We prove by induction on the structure of φ that $L(\varphi)$ is regular (and explicitely construct an automaton for it).

Case $\varphi = Q_a(x)$

- $\varphi = Q_a(x)$. Then $free(\varphi) = x$, and the interpretations of φ are encoded as words over $\Sigma \times \{0,1\}$. The language $L(\varphi)$ is given by

$$L(\varphi) = \left\{ \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \cdots \begin{bmatrix} a_k \\ b_k \end{bmatrix} \middle| \begin{array}{l} k \geq 0, \\ a_i \in \Sigma \text{ and } b_i \in \{0,1\} \text{ for every } i \in \{1,\ldots,k\}, \text{ and} \\ b_i = 1 \text{ for exactly one index } i \in \{1,\ldots,k\} \text{ such that } a_i = a \end{array} \right\}$$

and is recognized by

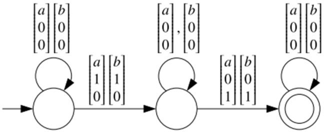


Case $\varphi = x < y$

- $\varphi = x < y$. Then $free(\varphi) = \{x, y\}$, and the interpretations of ϕ are encoded as words over $\Sigma \times \{0,1\}^2$. The language $L(\varphi)$ is given by

$$L(\varphi) = \left\{ \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \cdots \begin{bmatrix} a_k \\ b_k \\ c_k \end{bmatrix} \;\middle|\; \begin{array}{l} k \geq 0, \\ a_i \in \Sigma \text{ and } b_i, c_i \in \{0,1\} \text{ for every } i \in \{1, \ldots, k\}, \\ b_i = 1 \text{ for exactly one index } i \in \{1, \ldots, k\}, \\ c_j = 1 \text{ for exactly one index } j \in \{1, \ldots, k\}, \text{ and} \\ i < j \end{array} \right\}$$

and is recognized by

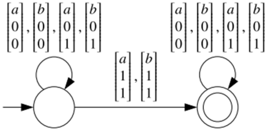


Case $\varphi = x \in X$

- $\varphi = x \in X$. Then $free(\varphi) = \{x, X\}$, and interpretations are encoded as words over $\Sigma \times \{0,1\}^2$. The language $L(\varphi)$ is given by

$$L(\varphi) = \left\{ \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \cdots \begin{bmatrix} a_k \\ b_k \\ c_k \end{bmatrix} \;\middle|\; \begin{array}{l} k \geq 0, \\ a_i \in \Sigma \text{ and } b_i, c_i \in \{0,1\} \text{ for every } i \in \{1, \ldots, k\}, \\ b_i = 1 \text{ for exactly one index } i \in \{1, \ldots, k\}, \text{ and} \\ \text{for every } i \in \{1, \ldots, k\}, \text{ if } b_i = 1 \text{ then } c_i = 1 \end{array} \right\}$$

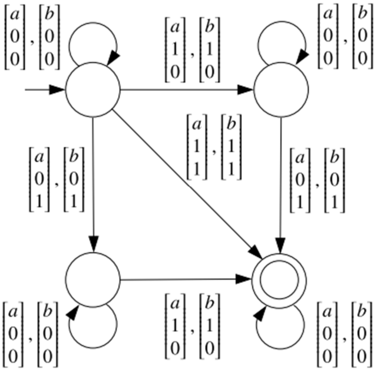
  and is recognized by



Case $\varphi = \neg\psi$

- Then $free(\varphi) = free(\psi)$. By i.h. $L(\psi)$ is regular.
- $L(\varphi)$ is equal to $\overline{L(\psi)}$ minus the words that do not encode any implementation („the garbage“).
- Equivalently, $L(\varphi)$ is equal to the intersection of $\overline{L(\psi)}$ and the encodings of all interpretations of ψ.
- We show that the set of these encodings is regular.
  - Condition for encoding: Let x be a free first-oder variable of ψ . The projection of an encoding onto x must belong to 0^*10^* (because it represents one position).
  - So we just need an automaton for the words satisfying this condition for every free first-order variable.

Example: $free(\varphi) = \{x, y\}$

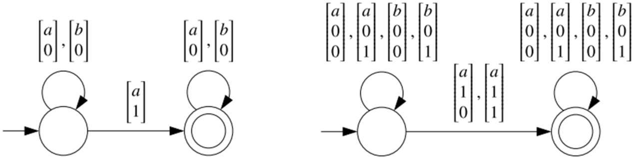


Case $\varphi = \varphi_1 \vee \varphi_2$

- Then $free(\varphi) = free(\varphi_1) \cup free(\varphi_2)$. By i.h. $L(\varphi_1)$ and $L(\varphi_2)$ are regular.
- If $free(\varphi_1) = free(\varphi_2)$ then $L(\varphi) = L(\varphi_1) \cup L(\varphi_2)$ and so $L(\varphi)$ is regular.
- If $free(\varphi_1) \neq free(\varphi_2)$ then we extend $L(\varphi_1)$ to a language L_1 encoding all interpretations of $free(\varphi_1) \cup free(\varphi_2)$ whose projection onto $free(\varphi_1)$ belongs to $L(\varphi_1)$. Similarly we extend $L(\varphi_2)$ to L_2. We have
  - L_1 and L_2 are regular.
  - $L(\varphi) = L_1 \cup L_2$.

Example: $\varphi = Q_a(x) \vee Q\_b(y)$

- L_1 contains the encodings of all interpretations $(w, \{x \mapsto n_1, y \mapsto n_2\})$ such that the encoding of $(w, \{x \mapsto n_1\})$ belongs to $L(Q_a(x))$.
- Automata for $L(Q_a(x))$ and L_1:

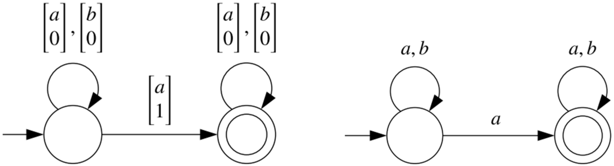


Cases $\varphi = \exists x\ \psi$ and $\varphi = \exists X\ \psi$

- Then $free(\varphi) = free(\psi) \setminus \{x\}$ or $free(\varphi) = free(\psi) \setminus \{X\}$
- By i.h. $L(\psi)$ is regular.
- $L(\varphi)$ is the result of projecting $L(\psi)$ onto the components for $free(\psi) \setminus \{x\}$ or $free(\psi) \setminus \{X\}$.

Example: $\varphi = Q_a(x)$

- Automata for $Q_a(x)$ and $\exists x\, Q_a(x)$



The mega-example

- We compute an automaton for

  $\exists x\,(\mathrm{last}(x) \wedge Q_b(x)) \;\wedge\; \forall x\,(\neg\mathrm{last}(x) \rightarrow Q_a(x))$

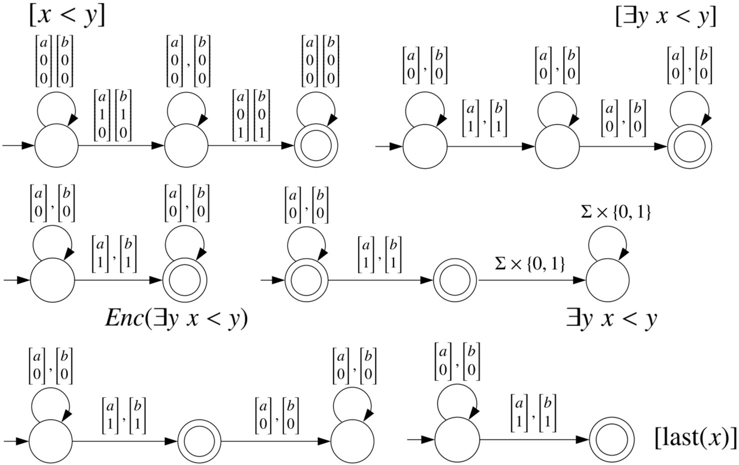
- First we rewrite φ into

  $\exists x\,(\mathrm{last}(x) \wedge Q_b(x)) \;\wedge\; \neg\exists x\,(\neg\mathrm{last}(x) \wedge \neg Q_a(x))$

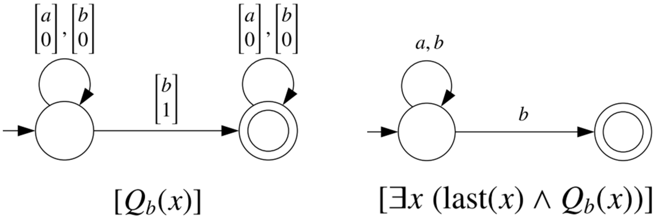
- In the next slides we
  1. compute a DFA for $last(x)$
  2. compute DFAs for $\exists x\ (last(x) \wedge Q_b(x))$ and $\neg\exists x\ (\neg last(x) \wedge \neg Q_a(x))$
  3. compute a DFA for the complete formula.

- We denote the DFA for a formula ψ by $[\psi]$.

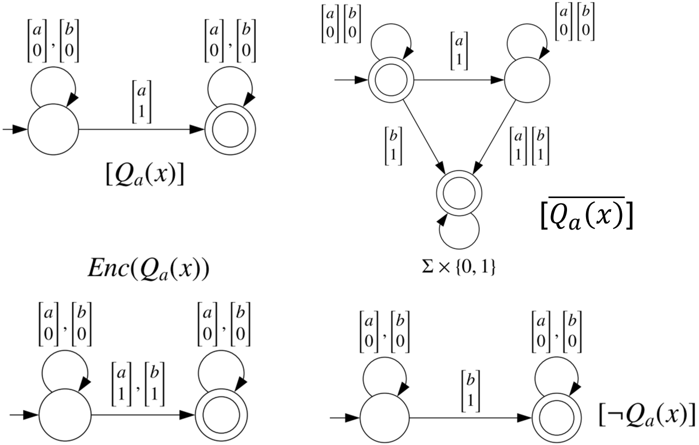
$[last(x)]$



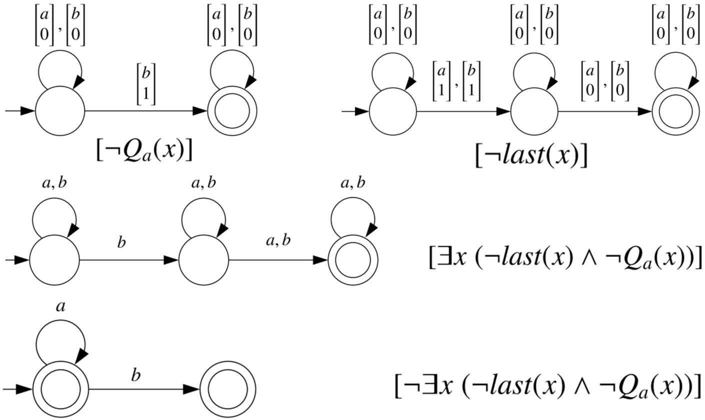
$[\exists x\,(last(x) \wedge Q_b(x))]$



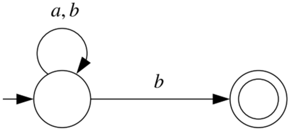
$[\neg Q_a(x)]$



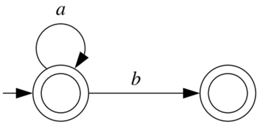
$[\neg\exists x\, (\neg last(x) \wedge \neg Q_a(x))]$



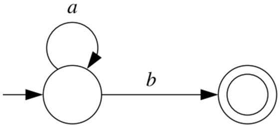
$$[\exists x\,\big(last(x) \wedge Q_b(x)\big)$$
$$\wedge \neg\exists x\,\big(\neg last(x) \wedge \neg Q_a(x)\big)]$$



$[\exists x\,(\mathrm{last}(x) \wedge Q_b(x))]$



$[\neg\exists x\,(\neg last(x) \wedge \neg Q_a(x))]$



$[\exists x\,(\mathrm{last}(x) \wedge Q_b(x)) \;\wedge\; \neg\exists x\,(\neg\mathrm{last}(x) \wedge \neg Q_a(x))]$

10. Presburger Arithmetic

Presburger Arithmetic is the first-order theory over the natural numbers ($\mathbb{N}_0$) with addition ($+$) as relation. It is convenient to also allow the constants 0 and 1 and the relations $\leq$ and $<$, with the canonical interpretation.

PA is named in honor of Mojżesz Presburger (1904–1943?):

- born in Warsaw
- died in Holocaust (1943?)
- student of Alfred Tarski
- MA-thesis: About the completeness of a certain system of integer arithmetic in which addition is the only operation (1930)

Again we are interested in which arithmetical problems can be solved using automata!

Syntax of PA

- Symbols:
  - variables x, y, z …
  - constants 0, 1
  - arithmetic symbols +, =<
  - logical symbols or, not, Exists
  - parenthesis

- Terms:
  - a variable is a term
  - 0 and 1 are terms
  - if t and u are terms, then t + u is a term

- Atomic formulas: t =< u , where t and u are terms

Syntax of PA

- Formulas:
  - every atomic formula is a formula;
  - if φ_1, φ_2 are formulas, then so are $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, and $\exists x \varphi_1$.

- Free and bound variables:
  - a variable is bound if it is in the scope of an existential quantifier, otherwise it is free.

- A formula without free variables is called a sentence

Abbreviations

Conjunction, implication, bi-implication, universal quantification

$$n = \underbrace{1 + 1 + \ldots + 1}_{n \text{ times}} \qquad nx = \underbrace{x + x + \ldots + x}_{n \text{ times}}$$

$$\begin{aligned} t \geq t' &= t' \leq t \\ t = t' &= t \leq t' \land t \geq t' \\ t < t' &= t \leq t' \land \neg(t = t') \\ t > t' &= t' < t \end{aligned}$$

Semantics (intuition)

- The semantics of a sentence is "true" or "false"

- The semantics of a formula with free variables (x_1, ..., x_k) is the set containing all tuples (n_1, ..., n_k) of natural numbers that "satisfy the formula"

Semantics (more formally)

- An interpretation of a formula F is any function that assigns a natural number to every variable appearing in f (and perhaps also to others).

  Given an interpretation I, a variable x, and a number n, we denote by I[n/x] the interpretation that assigns to x the number n , and to all other variables the same value as I.

Semantics (more formally)

- We define when an interpretation satisfies a formula F.

$$\begin{array}{lll} \mathfrak{I} \models t \leq u & \text{iff} & \mathfrak{I}(t) \leq \mathfrak{I}(u) \\ \mathfrak{I} \models \neg\varphi_1 & \text{iff} & \mathfrak{I} \not\models \varphi_1 \\ \mathfrak{I} \models \varphi_1 \vee \varphi_2 & \text{iff} & \mathfrak{I} \models \varphi_1 \text{ or } \mathfrak{I} \models \varphi_2 \\ \mathfrak{I} \models \exists x\, \varphi & \text{iff} & \text{there exists } n \geq 0 \text{ such that } I[n/x] \models \varphi \end{array}$$

- Lemma: Let F be a formula, and let I1, I2 be two interpretations of F. If I1 and I2 assign the same values to all FREE variables of F, then either they both satisfy F or none of them satisfies F.

- Consequence: if F is a sentence, either all interpretations satisfy F, or none of them satisfies F.

Semantics (more formally)

- We say a sentence is true if it is satisfied by all interpretations.

- We say a sentence is false if it is not satisfied by any interpretation.

- A model or solution of a formula F is the projection of any interpretation that satisfies F onto the free variables of F.

- The set of models or solutions of F is also called the solution space of F, and denoted by Sol(F).

Language of a formula

we encode natural numbers as strings over $\{0, 1\}$ using the least-significant-bit-first encoding *lsbf*. If we have free variables $x_1, \ldots, x_k$, the elements of the solution space are encoded as a word over $\{0, 1\}^k$. For instance, the word

$$\begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} \quad \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

is an encoding of the solution $(3, 10, 0)$. The language of a formula is then defined to be

$$\mathcal{L}(\varphi) = \{lsbf(s) \mid s \in Sol(\varphi)\}$$

Constructing an NFA for the solution space

Given a formula F, we construct an NFA Aut(F) such that L(Aut(F)) = L(F).

We can take:

- Aut(not F) = CompNFA(Aut(F))
- Aut(F or G) = UnionNFA(Aut(F), Aut(G))
- Aut(Exists x F) = Projection_x(Aut (F))

So it remains to define Aut(F) for an atomic formula F.

All atomic formulas equivalent (same solutions) to atomic formulas of the form

$$\varphi = \quad a_1 x_1 + \ldots + a_n x_n \leq b = \quad a \cdot x \leq b$$

where the a_i and b can be arbitrary integers (possibly negative).

Consider a candidate solution

$$\begin{array}{cccccc} & \zeta_0 & \zeta_1 & \cdots & \zeta_m \\ \begin{matrix} c_1 \\ c_2 \\ \cdots \\ c_n \end{matrix} & \begin{bmatrix} \zeta_{10} \\ \zeta_{20} \\ \cdots \\ \zeta_{n0} \end{bmatrix} & \begin{bmatrix} \zeta_{11} \\ \zeta_{21} \\ \cdots \\ \zeta_{n1} \end{bmatrix} & \begin{matrix} \cdots \\ \cdots \\ \cdots \\ \cdots \end{matrix} & \begin{bmatrix} \zeta_{1m} \\ \zeta_{2m} \\ \cdots \\ \zeta_{nm} \end{bmatrix} \end{array}$$

For every $j \leq m$, let $c^j \in \mathbb{N}^n$ denote the tuple of numbers encoded by the prefix $\zeta_0 \ldots \zeta_{j-1}$. For instance, for the encoding $\zeta_0\zeta_1\zeta_2$ of the tuple $(0, 4, 7, 3)$ given by

$$\begin{array}{c} \\ 0 \\ 4 \\ 7 \\ 3 \end{array} \quad \begin{array}{c} \zeta_0 \\ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \end{array} \begin{array}{c} \zeta_1 \\ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \end{array} \begin{array}{c} \zeta_2 \\ \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \end{array} \qquad \text{we get} \qquad \begin{array}{c} \\ 0 \\ 0 \\ 3 \\ 3 \end{array} \quad \begin{array}{c} \zeta_0 \\ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \end{array} \begin{array}{c} \zeta_1 \\ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \end{array}$$

and so $c^2 = (0, 0, 3, 3)$. Define further $c^0 = (0, 0, 0, 0)$; i.e., before reading anything all components of the tuple are 0.

We construct a DFA for the solution space of φ. The idea is that after reading a prefix $\zeta_0 \ldots \zeta_{j-1}$ the automaton should be in the state

$$\left\lfloor \frac{1}{2^j} \left(b - a \cdot c^j \right) \right\rfloor \tag{10.1}$$

Initially we have $c^0 = (0, \ldots, 0)$, and so the initial state is the number $\frac{1}{2^0}(b - a \cdot c^0) = b$. For the transitions, assume that before and after reading the letter ζ_j the automaton is in the states q and q', respectively. Then we have

$$q = \left\lfloor \frac{1}{2^j}\left(b - a \cdot c^j\right) \right\rfloor \qquad \text{and} \qquad q' = \left\lfloor \frac{1}{2^{j+1}}\left(b - a \cdot c^{j+1}\right) \right\rfloor$$

From the definition of c^j we get:

$$c^{j+1} = c^j + 2^j \zeta_j$$

Inserting this in the expression for q', and comparing with q, we obtain the following relation between q and q':

$$q' = \left\lfloor \frac{1}{2}(q - a \cdot \zeta_j) \right\rfloor$$

So for every state q and every letter $\zeta \in \{0, 1\}^n$ we take $\delta(q, \zeta) := \frac{1}{2}(q - a \cdot \zeta)$.

PAtoDFA(φ)

Input: PA formula $\varphi = a \cdot x \leq b$

Output: DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(A) = \mathcal{L}(\varphi)$

$q_0 \leftarrow s_b$

$W \leftarrow \{s_b\}$

while $W \neq \emptyset$ **do**

  pick s_k **from** W

  add s_k **to** Q

  if $k \geq 0$ **then add** s_k **to** F

  for all $\zeta \in \{0,1\}^n$ **do**

    $j \leftarrow \left\lfloor \frac{1}{2}(k - a \cdot \zeta) \right\rfloor$

    if $s_j \notin Q$ **then add** s_j **to** W

    add (s_k, ζ, s_j) **to** δ

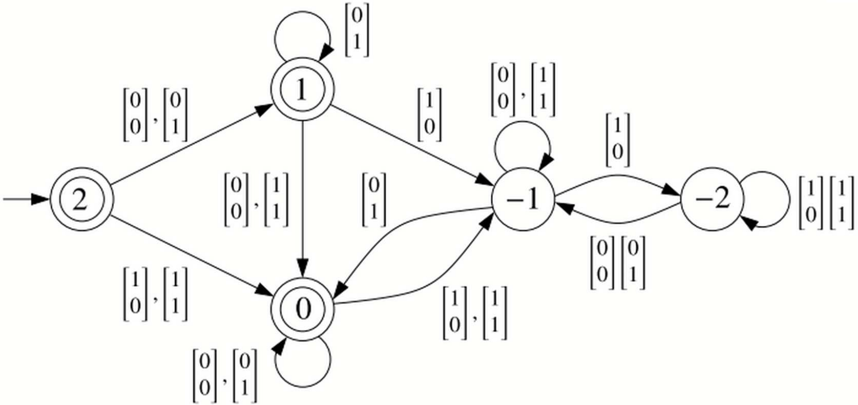


Figure 10.1: DFAs for the formula $2x - y \leq 2$.

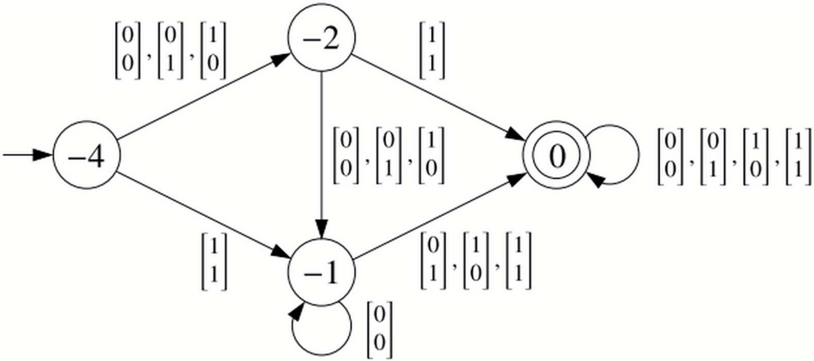


Figure 10.2: DFAs for the formula $x + y \geq 4$.

Lemma 10.3 *Let $\varphi = a \cdot x \leq b$ and $s = \sum_{i=1}^{k} |a_i|$. All states s_j added to the worklist during the execution of PAtoDFA(φ) satisfy*

$$-|b| - s \leq j \leq |b| + s.$$

Proof: The property holds for s_b, the first state added to the worklist. We show that if all the states added to the worklist so far satisfy the property, then so does the next one.

Let s_j be this next state. Then there exists a state s_k in the worklist and $\zeta \in \{0, 1\}^n$ such that $j = \lfloor \frac{1}{2}(k - a \cdot \zeta) \rfloor$. Since by assumption s_k satisfies the property we have

$$-|b| - s \leq k \leq |b| + s$$

and so

$$\left\lfloor \frac{-|b| - s - a \cdot \zeta}{2} \right\rfloor \leq j \leq \left\lfloor \frac{|b| + s - a \cdot \zeta}{2} \right\rfloor \tag{10.2}$$

Now we observe

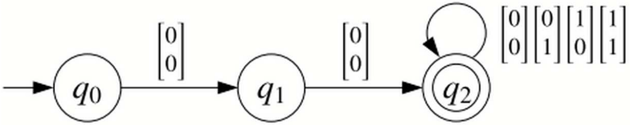
$$\begin{array}{rcccl} -|b|-s & \leq & \dfrac{-|b|-2s}{2} & \leq & \left\lfloor \dfrac{-|b|-s-a\cdot\zeta}{2} \right\rfloor \\ \left\lfloor \dfrac{|b|+s-a\cdot\zeta}{2} \right\rfloor & \leq & \dfrac{|b|+2s}{2} & \leq & |b|+s \end{array}$$

which together with 10.2 yields

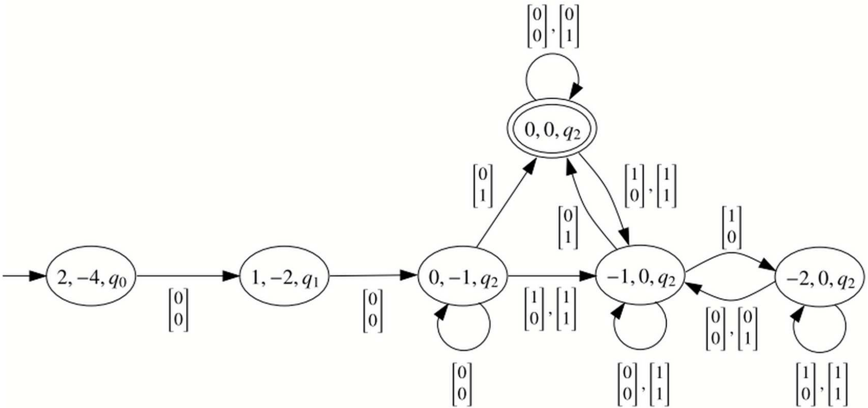
$$-|b|-s \leq j \leq |b|+s$$

and we are done. □

$$\exists z\, x = 4z \;\wedge\; \exists w\, y = 4w \;\wedge\; 2x - y \leq 2 \;\wedge\; x + y \geq 4$$



DFA for the formula $\exists z x = 4z \;\wedge\; \exists w\, y = 4w$.



2, −4, q0
1, −2, q1
0, −1, q2
0, 0, q2
−1, 0, q2
−2, 0, q2

Interlude

11. Semilinear Sets

See, e.g.,

Kracht, M.:
A new proof of a theorem by Ginsburg and Spanier.
Manuscript, Dept. of Linguistics, UCLA (2002)

Fischer, Michael J. and Michael O. Rabin:
Super-exponential complexity of Presburger Arithmetic.
SIAM-AMS Proceedings, vol. 7, AMS (1974)

Chapter II ω-Automata

1. ω-Automata and ω-Languages

- ω-automata accept (or reject) words of infinite length
- ω-languages consisting of infinite words appear:
  - in verification, as encodings of non-terminating executions of a program
  - in arithmetic, as encodings of sets of real numbers

ω-Languages

- An ω-word is an infinite sequence of letters.
- The set of all ω-words is denoted by Σ^ω.
- An ω-language is a set of ω-words, i.e., a subset of Σ^ω.
- A language L_1 can be concatenated with an ω-language L_2 to yield the ω-language $L_1 L_2$, but two ω-languages cannot be concatenated.
- The ω-iteration of a language $L \subseteq \Sigma^\star$, denoted by L^ω, is an ω-language.
- Observe: $\emptyset^\omega = \emptyset$.

ω-Regular Expressions

- ω-regular expressions have syntax

$$s ::= r^{\omega} \mid r s_1 \mid s_1 + s_2$$

  where r is an (ordinary) regular expression.

- The ω-language $L_{\omega}(s)$ of an ω-regular expression s is inductively defined by

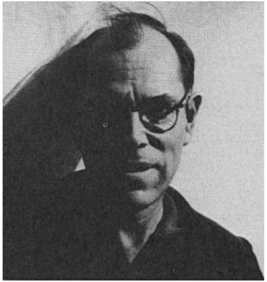
$$L_{\omega}(r^{\omega}) = \left(L(r)\right)^{\omega} \quad L_{\omega}(r s_1) = L(r) L_{\omega}(s_1)$$

$$L_{\omega}(s_1 + s_2) = L_{\omega}(s_1) \cup L_{\omega}(s_2)$$

- A language is ω-regular if it is the language of some ω-regular expression .

Büchi Automata

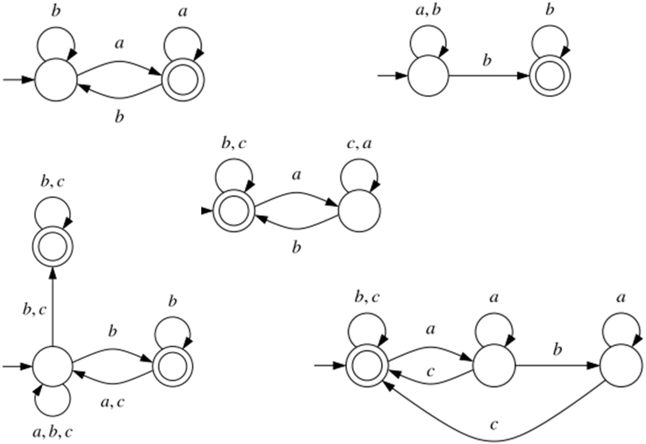
- Invented by J.R. Büchi, swiss logician.



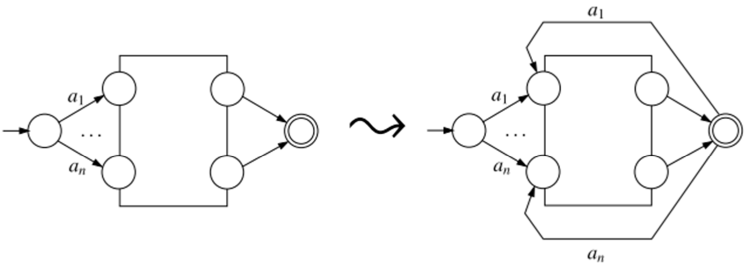
Büchi Automata

- Same syntax as DFAs and NFAs, but different acceptance condition.
- A run of a Büchi automaton on an ω-word is an infinite sequence of states and transitions.
- A run is accepting if it visits the set of final states infinitely often.
  - Final states renamed to accepting states.
- A DBA or NBA A accepts an ω-word if it has an accepting run on it; the ω-language $L_\omega(A)$ of A is the set of ω-words it accepts.

Some examples

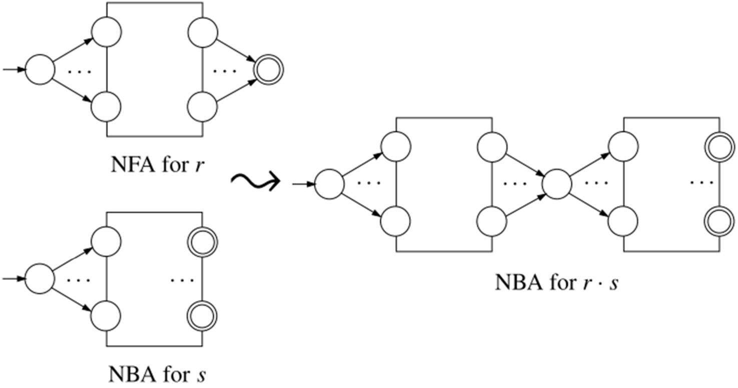


From ω-Regular Expressions to NBAs

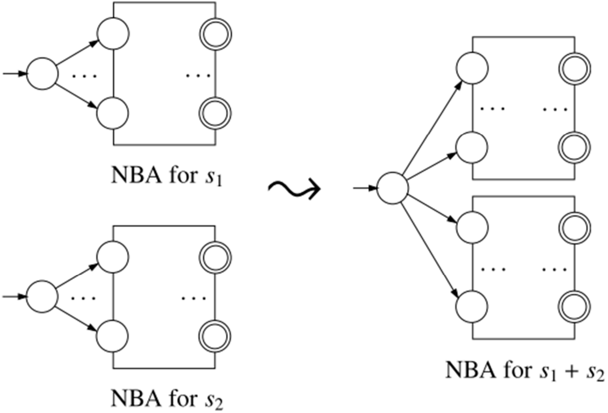


NFA for r $\rightsquigarrow$ NBA for r^ω

From ω-Regular Expressions to NBAs



From ω-Regular Expressions to NBAs

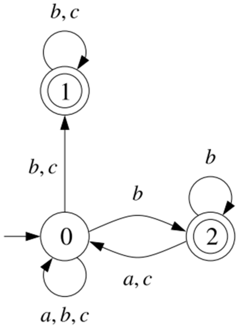


From NBAs to ω-Regular Expressions

- Lemma: Let A be a NFA, and let q, q' be states of A . The language $L_q^{q'}$ of words with runs leading from q to q' and visiting q' exactly once is regular.
- Let $r_q^{q'}$ denote a regular expression for $L_q^{q'}$.

From NBAs to ω-Regular Expressions

- Example:



$$
\begin{aligned}
r_0^1 &= (a+b+c)^*(b+c) \\
r_0^2 &= (a+b+c)^* b \\
r_1^1 &= (b+c)^* \\
r_2^2 &= b + (a+c)(a+b+c)^* b
\end{aligned}
$$

From NBAs to ω-Regular Expressions

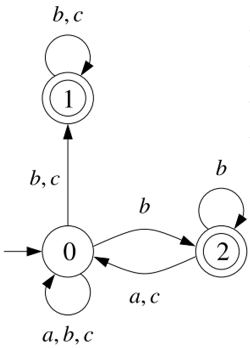
- Given a NBA A , we look at it as a NFA, and compute regular expressions $r_q^{q'}$.
- We show:

$$L_\omega(A) = L\Big(\sum_{q \in F} r_{q_0}^{q} \left(r_q^{q}\right)^{\omega} \Big)$$

  – An ω-word belongs to $L_\omega(A)$ iff it is accepted by a run that starts at q_0 and visits some accepting state q infinitely often.

From NBAs to ω-Regular Expressions

- Example:



$$\begin{aligned} r_0^1 &= (a+b+c)^*(b+c) \\ r_0^2 &= (a+b+c)^*b \\ r_1^1 &= (b+c)^* \\ r_2^2 &= b+(a+c)(a+b+c)^*b \end{aligned}$$

$$L_\omega(A) = r_0^1\,(r_1^1)^\omega + r_0^2\,(r_2^2)^\omega$$

DBAs are less expressive than NBAs

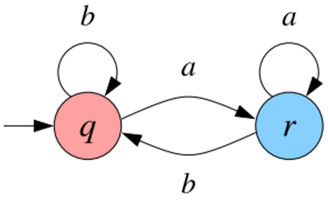
- Prop.: The ω-language $(a+b)^* b^\omega$ is not recognized by any DBA.
- Proof: By contradiction. Assume some DBA recognizes $(a+b)^* b^\omega$.
  - DBA accepts b^ω → DFA accepts b^{n_0}

    DBA accepts $b^{n_0} a\, b^\omega$ → DFA accepts $b^{n_0} a\, b^{n_1}$

    DBA accepts $b^{n_0} a\, b^{n_1}\, a b^\omega$ → DFA accepts $b^{n_0} a\, b^{n_1} a\, b^{n_2}$ etc.
  - By determinism, the DBA accepts $b^{n_0} a\, b^{n_1} a\, b^{n_2} \ldots a\, b^{n_i} \ldots$, which does not belong to $(a+b)^* b^\omega$.

Generalized Büchi Automata

- Same power as Büchi automata, but more adequate for some constructions.
- Several sets of accepting states.
- A run is accepting if it visits each set of accepting states infinitely often.



$$\mathcal{F} = \{\ \{q\}, \{r\}\ \}$$

From NGAs to NBAs

- Important fact:

All the sets $F_1, \ldots, F_n$ are visited infinitely often

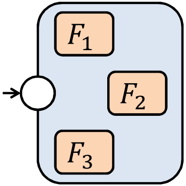
is equivalent to

F_1 is eventually visited

and

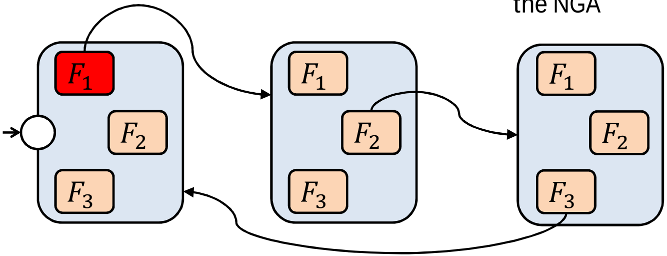
every visit to F_i is eventually followed by a visit to $F_{i \oplus 1}$

From NGAs to NBAs



NGA with 3 sets of accepting states

Equivalent NBA with 3 copies of the NGA

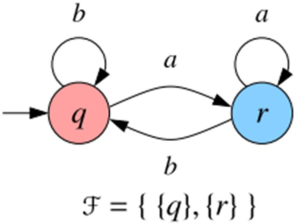


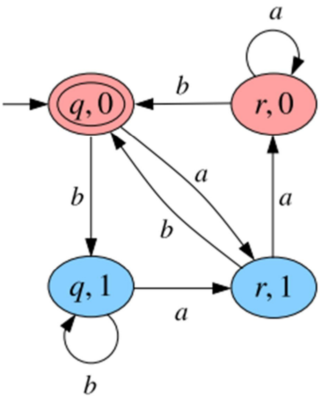
NGAtoNBA(A)

Input: NGA $A = (Q, \Sigma, q_0, \delta, \mathcal{F})$, where $\mathcal{F} = \{F_1, \ldots, F_m\}$

Output: NBA $A' = (Q', \Sigma, \delta', q'_0, F')$

```
Q′, δ′, F′ ← ∅; q′₀ ← [q₀, 0]
W ← {[q₀, 0]}
while W ≠ ∅ do
   pick [q, i] from W
   add [q, i] to Q′
   if q ∈ F₀ and i = 0  then add  [q, i]  to F′
   for all a ∈ Σ, q′ ∈ δ(q, a) do
      if q ∉ Fᵢ then
         if [q′, i] ∉ Q′ then add  [q′, i]  to W
         add ([q, i], a, [q′, i]) to δ′
      else /* q ∈ Fᵢ */
         if [q′, i ⊕ 1] ∉ Q′ then add  [q′, i ⊕ 1]  to W
         add ([q, i], a, [q′, i ⊕ 1]) to δ′
return (Q′, Σ, δ′, q′₀, F′)
```


b
a
a
q
r
b
$\mathcal{F} = \{ \{q\}, \{r\} \}$


a
b
q, 0
r, 0
a
b
a
b
q, 1
r, 1
a
b

DGAs have the same expressive power as DBAs, and so are not equivalent to NGAs.

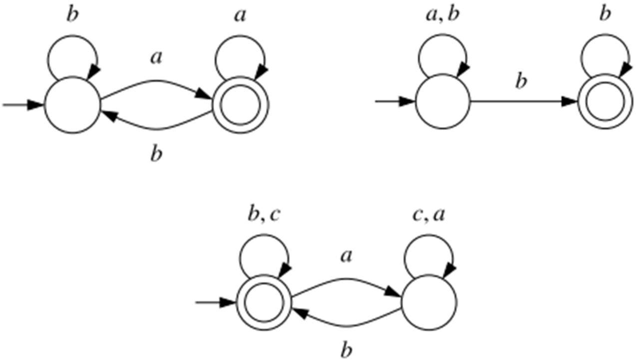
- Question: Are there other classes of omega-automata with
  - the same expressive power as NBAs or NGAs, and
  - with equivalent deterministic and nondeterministic versions?

We are only willing to change the acceptance condition!

Co-Büchi automata

- A nondeterministic co-Büchi automaton (NCA) is syntactically identical to a NBA, but a run is accepting iff it only visits accepting states finitely often.

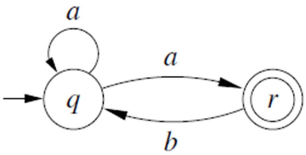
Which are the languages?

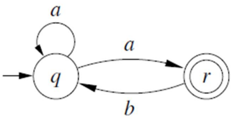


Determinizing co-Büchi automata

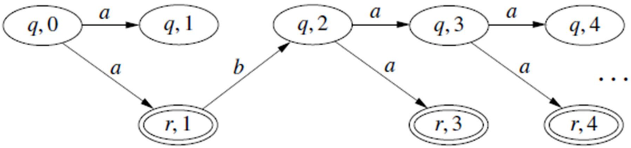
- Given a NCA A we construct a DCA B such that $L(A) = L(B)$.
- We proceed in three steps:
  - We assign to every ω-word w a directed acyclic graph $dag(w)$ that ``contains´´ all runs of A on w.
  - We prove that w is accepted by A iff $dag(w)$ is infinite but contains only finitely many breakpoints.
  - We construct a DCA B that accepts an ω-word w iff $dag(w)$ is infinite and contains finitely many breakpoints.

- Running example:

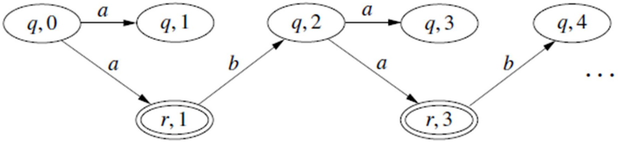




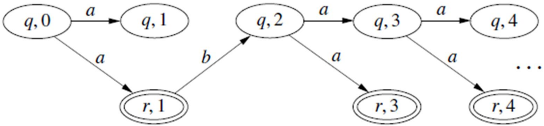
$dag(aba^{\omega})$

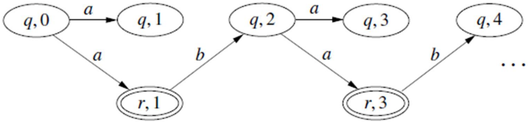


$dag((ab)^{\omega})$

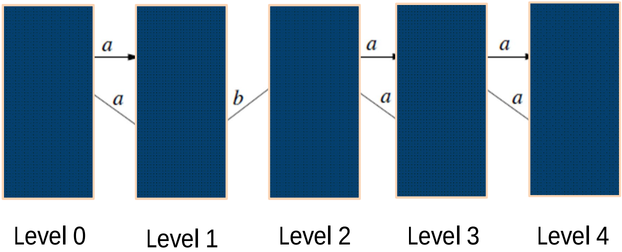


- A accepts w iff some infinite path of $dag(w)$ only visits accepting states finitely often





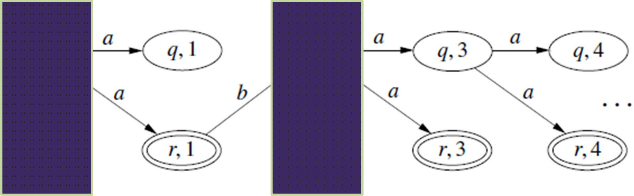
Levels of a dag

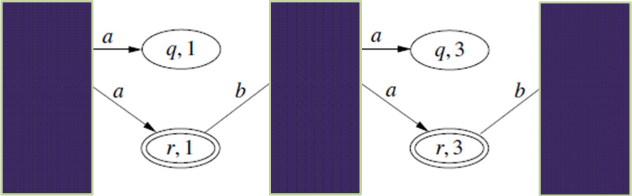


Breakpoints of a dag

- We defined inductively the set of levels that are breakpoints:
  - Level 0 is always a breakpoint
  - If level l is a breakpoint, then the next level l' such that every path between l and l' visits an accepting state is also a breakpoint.

Only two breakpoints





Infinitely many breakpoints

- Lemma: A accepts w iff $dag(w)$ is infinite and has only finitely many breakpoints.

Proof:

If A accepts w, then A has at least one run on w, and so $dag(w)$ is infinite. Moreover, the run visits accepting states only finitely often, and so after it stops visiting accepting states there are no further breakpoints.

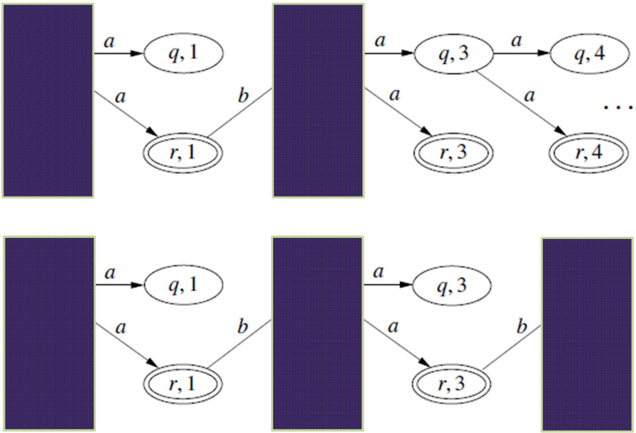
If $dag(w)$ is infinite, then it has an infinite path, and so A has at least one run on w. Since $dag(w)$ has finitely many breakpoints, then every infinite path visits accepting states only finitely often.

Constructing the DCA

- If we could tell if a level is a breakpoint by looking at it, we could take the set of breakpoints as states of the DCA.
- However, we also need some information about its ``history´´.
- Solution: add that information to the level!
- States: pairs $[P, O]$ where:
  - P is the set of states of a level, and
  - $O \subseteq P$ is the set of states ``that owe a visit to the accepting states‘‘. Formally: $q \in O$ if q is the

Constructing the DCA

- States: pairs $[P, O]$ where:
  - P is the set of states of a level, and
  - $O \subseteq P$ is the set of states ``that owe a visit to the accepting states''.
- Formally: $q \in O$ if q is the endpoint of a path starting at the last breakpoint that has not yet visited any accepting state.



a
q, 1
a
r, 1
b
a
q, 3
a
q, 4
a
a
...
r, 3
r, 4
a
q, 1
a
r, 1
b
a
q, 3
a
r, 3
b

Constructing the DCA

- States: pairs $[P, O]$
- Initial state: pair $[\{q_0\}, \emptyset]$ if $q_0 \in F$, and $[\{q_0\}, \{q_0\}]$ otherwise.
- Transitions: $\delta([P, Q], a) = [P', O']$ where $P' = \delta(P, a)$, and
  - $O' = \delta(O, a) \setminus F$ if $O \neq \emptyset$ (automaton updates set of owing states)
  - $O' = \delta(P, a) \setminus F$ if $O = \emptyset$ (automaton starts search for next breakpoint)
- Accepting states: pairs $[P, \emptyset]$ (no owing states)

NCAtoDCA(A)

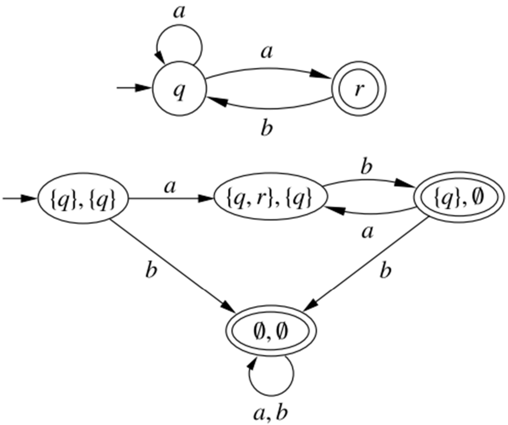
Input: NCA $A = (Q, \Sigma, \delta, q_0, F)$

Output: DCA $B = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q}_0, \tilde{F})$ with $L_\omega(A) = \overline{B}$

$\tilde{Q}, \tilde{\delta}, \tilde{F} \leftarrow \emptyset$; **if** $q_0 \in F$ **then** $\tilde{q}_0 \leftarrow [q_0, \emptyset]$ **else** $\tilde{q}_0 \leftarrow [\{q_0\}, \{q_0\}]$
$W \leftarrow \{ \tilde{q}_0 \}$
while $W \neq \emptyset$ **do**
    pick $[P, O]$ **from** W; **add** $[P, O]$ **to** $\tilde{Q}$
    if $P = \emptyset$ **then add** $[P, O]$ **to** $\tilde{F}$
    for all $a \in \Sigma$ **do**
        $P' = \delta(P, a)$
        if $O \neq \emptyset$ **then** $O' \leftarrow \delta(O, a) \setminus F$ **else** $O' \leftarrow \delta(P, a) \setminus F$
        add $([P, O], a, [P', O'])$ **to** $\tilde{\delta}$
        if $[P', O'] \notin \tilde{Q}$ **then add** $[P', Q']$ **to** W

- Complexity: at most 3^n states

Running example



Recall ...

- Question: Are there other classes of omega-automata with
  - the same expressive power as NBAs or NGAs, and
  - with equivalent deterministic and nondeterministic versions?

Are co-Büchi automata a positive answer?

Unfortunately no ...

- Lemma: No DCA recognizes the language $(b^*a)^\omega$.

Proof: Assume the contrary. Then the same automaton seen as a DBA recognizes the complement $(a + b)^*b^\omega$. Contradiction.

So the quest goes on ...

Muller automata

- A nondeterministic Muller automaton (NMA) has a collection $\{F_0, F_1, \ldots, F_{m-1}\}$ of sets of accepting states.
- A run is accepting if the set of states it visits infinitely often is equal to one of the sets in the collection.

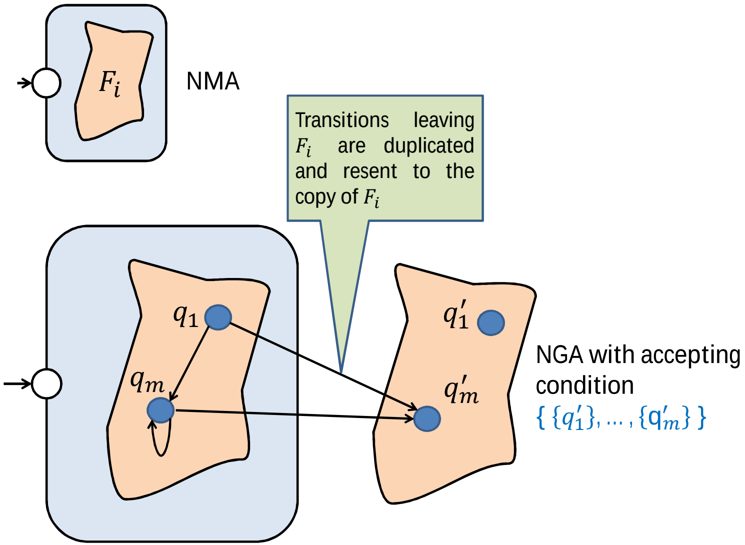
From Büchi to Muller automata

- Let A be a NBA with set F of accepting states.
- A set of states of A is good if it contains some state of F.
- Let G be the set of all good sets of A.
- Let A' be "the same automaton" as A, but with Muller condition G.
- Let ρ be an arbitrary run of A and A'. We have

	ρ is accepting in A
iff	$\inf(\rho)$ contains some state of F
iff	$\inf(\rho)$ is a good set of A
iff	ρ is accepting in A'

From Muller to Büchi automata

- Let A be a NMA with condition $\{F_0, F_1, \ldots, F_{m-1}\}$.
- Let $A_0, \ldots, A_{m-1}$ be NMAs with the same structure as A but Muller conditions $\{F_0\}, \{F_1\}, \ldots, \{F_{m-1}\}$ respectively.
- We have: $L(A) = L(A_0) \cup \ldots \cup L(A_{m-1})$
- We proceed in two steps:
  1. we construct for each NMA A_i an NGA $A_i{'}$ such that $L(A_i) = L(A_i')$
  2. we construct an NGA A' such that
     $$L(A') = L(A'_0) \cup \ldots \cup L(A'_{m-1})$$

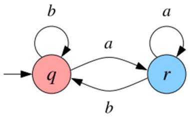

F_i
NMA
Transitions leaving F_i are duplicated and resent to the copy of F_i
q_1
q_m
q'_1
q'_m
NGA with accepting condition
$\{\, \{q'_1\}, \ldots, \{q'_m\} \,\}$

NMA1toNGA(A)

Input: NMA $A = (Q, \Sigma, q_0, \delta, \{F\}$

Output: NGA $A = (Q', \Sigma, q'_0, \delta', \mathcal{F}')$

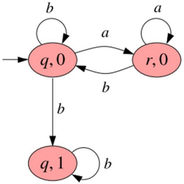
```
Q', δ', 𝓕' ← ∅
q'_0 ← [q_0, 0]
W ← {[q_0, 0]}
while W ≠ ∅ do
   pick [q, i] from W; add [q, i] to Q'
   if q ∈ F and i = 1 then add {[q, 1]} to 𝓕'
   for all a ∈ Σ, q' ∈ δ(q, a) do
      if i = 0 then
         add ([q, 0], a, [q', 0]) to δ'
         if [q', 0] ∉ Q' then add [q', 0] to W
         if q' ∈ F then
            add ([q, 0], a, [q', 1]) to δ'
            if [q', 1] ∉ Q' then add [q', 1] to W
      else /* i = 1 */
         if q' ∈ F then
            add ([q, 1], a, [q', 1]) to δ'
            if [q', 1] ∉ Q' then add [q', 1] to W
return (Q', Σ, q'_0, δ', 𝓕')
```



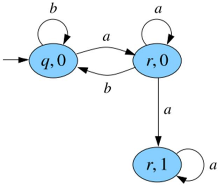
$\mathcal{F} = \{F_0, F_1\}$

$F_0 = \{q\}$

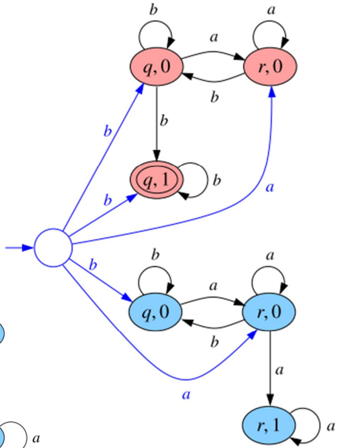
$F_1 = \{r\}$



$\mathcal{F}'_0 = \{ [q, 1] \}$



$\mathcal{F}'_1 = \{ [r, 1] \}$

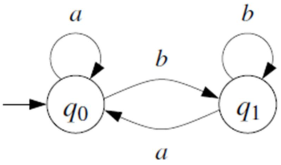


Equivalence of NMAs and DMAs

- Theorem (Safra): Any NBA with n states can be effectively transformed into a DMA of size $n^{O(n)}$.

  Proof: Omitted.

- DMA for $(a + b)^* b^\omega$:



with accepting condition $\{\{q_1\}\}$

- Question: Are there other classes of omega-automata with
  - the same expressive power as NBAs or NGAs, and
  - with equivalent deterministic and nondeterministic versions?
- Answer: Yes, Muller automata

Is the quest over?

- Recall the translation NBA $\Rightarrow$ NMA
- The NMA has the same structure as the NBA; its accepting condition are all the good sets of states.
- The translation has exponential complexity.

New question: Is there a class of ω-automata with

– the same expressive power as NBAs,

– equivalent deterministic and nondeterministic versions, and

– polynomial conversions to and from Büchi automata?

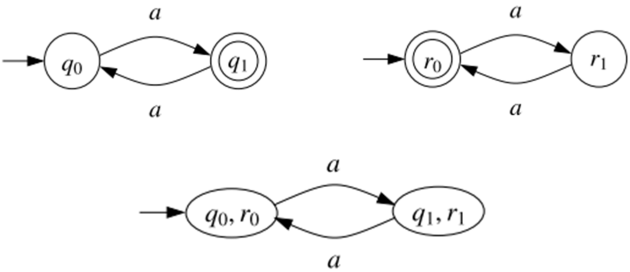
Rabin automata

- The acceptance condition is a set of pairs $\{ \langle F_0, G_0 \rangle, \dots, \langle F_{m-1}, G_{m-1} \rangle \}$
- A run ρ is accepting if there is a pair $\langle F_i, G_i \rangle$ such that ρ visits the set F_i infinitely often and the set G_i finitely often.
- Translations NBA ⇒ NRA and NRA ⇒ NBA are left as an exercise.
- Theorem (Safra): Any NBA with n states can be effectively transformed into a DRA with $n^{O(n)}$ states and $O(n)$ accepting pairs.

2. Implementing Boolean Operations for Büchi Automata

Intersection of NBAs

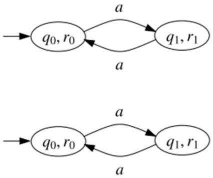
- The algorithm for NFAs does not work ...



Solution

Apply the same idea as in the conversion NGA ⇒ NBA

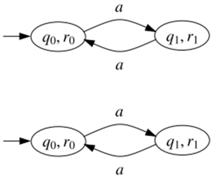
1. Take two copies of the pairing $[A_1, A_2]$.



Solution

Apply the same idea as in the conversion NGA ⇒ NBA

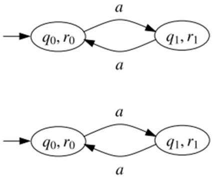
1. Take two copies of the pairing $[A_1, A_2]$.
2. Redirect transitions of the first copy leaving F_1 to the second copy.



Solution

Apply the same idea as in the conversion NGA $\Rightarrow$ NBA

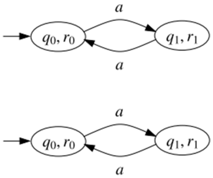
1. Take two copies of the pairing $[A_1, A_2]$.
2. Redirect transitions of the first copy leaving F_1 to the second copy.
3. Redirect transitions of the second copy leaving F_2 to the second copy.



Solution

Apply the same idea as in the conversion NGA ⇒ NBA

1. Take two copies of the pairing $[A_1, A_2]$.
2. Redirect transitions of the first copy leaving F_1 to the second copy.
3. Redirect transitions of the second copy leaving F_2 to the second copy.
4. Set F to the set F_1 in the first copy.



IntersNBA(A_1, A_2)

Input: NBAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

Output: NBA $A_1 \cap_\omega A_2 = (Q, \Sigma, \delta, q_0, F)$ with $L_\omega(A_1 \cap_\omega A_2) = L_\omega(A_1) \cap L_\omega(A_2)$

$Q, \delta, F \leftarrow \emptyset$
$q_0 \leftarrow [q_{01}, q_{02}, 1]$
$W \leftarrow \{ [q_{01}, q_{02}, 1] \}$
while $W \neq \emptyset$ **do**
    pick $[q_1, q_2, i]$ **from** W
    add $[q_1, q_2, i]$ **to** Q'
    if $q_1 \in F_1$ **and** $i = 1$ **then add** $[q_1, q_2, 1]$ **to** F'
    for all $a \in \Sigma$ **do**
        for all $q_1' \in \delta_1(q_1, a), q_2' \in \delta(q_2, a)$ **do**
            if $i = 1$ **and** $q_1 \notin F_1$ **then**
                add $([q_1, q_2, 1], a, [q_1', q_2', 1])$ **to** δ
                if $[q_1', q_2', 1] \notin Q'$ **then add** $[q_1', q_2', 1]$ **to** W
            if $i = 1$ **and** $q_1 \in F_1$ **then**
                add $([q_1, q_2, 1], a, [q_1', q_2', 2])$ **to** δ
                if $[q_1', q_2', 2] \notin Q'$ **then add** $[q_1', q_2', 2]$ **to** W
            if $i = 2$ **and** $q_2 \notin F_2$ **then**
                add $([q_1, q_2, 2], a, [q_1', q_2', 2])$ **to** δ
                if $[q_1', q_2', 2] \notin Q'$ **then add** $[q_1', q_2', 2]$ **to** W
            if $i = 2$ **and** $q_2 \in F_2$ **then**
                add $([q_1, q_2, 2], a, [q_1', q_2', 1])$ **to** δ
                if $[q_1', q_2', 1] \notin Q'$ **then add** $[q_1', q_2', 1]$ **to** W
return $(Q, \Sigma, \delta, q_0, F)$

Special cases/improvements

- If all states of at least one of A_1 and A_2 are accepting, the algorithm for NFAs works.
- Intersection of NBAs $A_1, A_2, \ldots, A_k$
  - Do NOT apply the algorithm for two NBAs $(k-1)$ times.
  - Proceed instead as in the translation

    NGA ⇒ NBA: take k copies of $[A_1, A_2, \ldots, A_k]$

    ($kn_1 \ldots n_k$ states instead of $2^k n_1 \ldots n_k$)

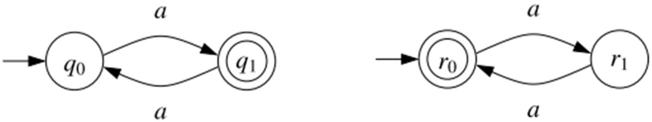
Complement

- Main result proved by Büchi: NBAs are closed under complement.
- Many later improvements in recent years.
- Construction radically different from the one for NFAs.

Problems

- The powerset construction does not work.

- Exchanging final and non-final states in DBAs also fails.

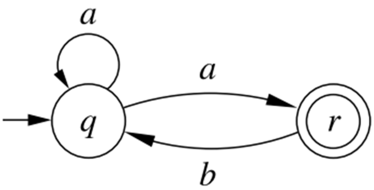


Solution

- Extend the idea used to determinize co-Büchi automata with a new component.
- Recall: a NBA accepts a word w iff some path of $dag(w)$ visits final states infinitely often.
- Goal: given NBA A, construct NBA $\bar{A}$ such that:

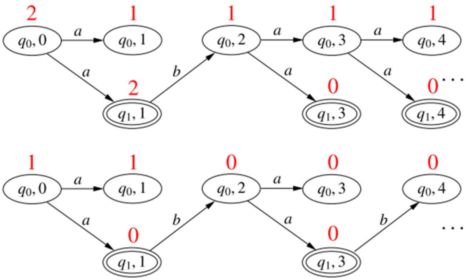
A rejects w
iff
no path of $dag(w)$ visits accepting states of A i.o.
iff
some run of $\bar{A}$ visits accepting states of $\bar{A}$ i.o.
iff
$\bar{A}$ accepts w

Running example



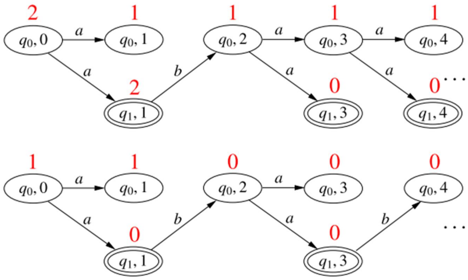
Rankings

- Mappings that associate to every node of $dag(w)$ a rank (a natural number) such that
  - ranks never increase along a path, and
  - ranks of accepting nodes are even.



Odd rankings

- A ranking is odd if every infinite path of $dag(w)$ visits nodes of odd rank i.o.



Prop.: no path of $dag(w)$ visits accepting states of A i.o.
iff
$dag(w)$ has an odd ranking

Proof: Ranks along infinite paths eventually reach a **stable rank**.

($\leftarrow$): The stable rank of every path is odd. Since accepting nodes have even rank, no path visits accepting nodes i.o.

($\rightarrow$): We construct a ranking satisfying the conditions.

Give each accepting node $\langle q, l \rangle$ rank $2k$, where k is the maximal number of accepting nodes in a path starting at $\langle q, l \rangle$.

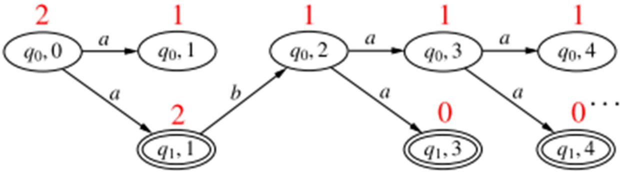
Give a non-accepting node $\langle q, l \rangle$ rank $2k + 1$, where 2k is the maximal even rank among its descendants.

- Goal:

A rejects w
iff
$dag(w)$ has an odd ranking
iff
some run of $\bar{A}$ visits accepting states of $\bar{A}$ i.o.
iff
$\bar{A}$ accepts w

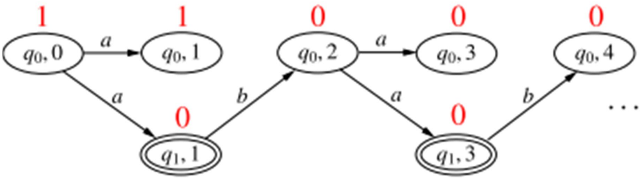
- Idea: design $\bar{A}$ so that
  - its runs on w are the rankings of $dag(w)$, and
  - its acceptings runs on w are the odd rankings of $dag(w)$.

Representing rankings



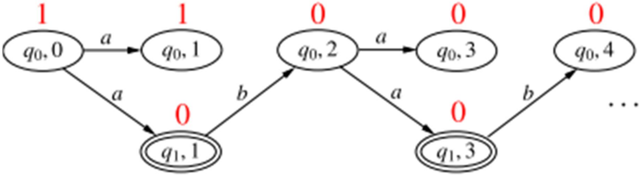
$$\begin{bmatrix}2\\ \bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\ 2\end{bmatrix} \xrightarrow{b} \begin{bmatrix}1\\ \bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\ 0\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\ 0\end{bmatrix} \cdots$$

Representing rankings



$$\begin{bmatrix}1\\ \bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\ 0\end{bmatrix} \xrightarrow{b} \begin{bmatrix}0\\ \bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}0\\ 0\end{bmatrix} \xrightarrow{b} \begin{bmatrix}0\\ \bot\end{bmatrix} \dots$$

Representing rankings



$$\begin{bmatrix}1\\\bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\0\end{bmatrix} \xrightarrow{b} \begin{bmatrix}0\\\bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}0\\0\end{bmatrix} \xrightarrow{b} \begin{bmatrix}0\\\bot\end{bmatrix} \cdots$$

- We can determine if $\begin{bmatrix}n_1\\n_2\end{bmatrix} \xrightarrow{l} \begin{bmatrix}n_1'\\n_2'\end{bmatrix}$ may appear in a ranking by just looking at n_1, n_2, n_1', n_2' and l : ranks should not increase.

First draft for $\bar{A}$

- For a two-state A (more states analogous):
  - States: all $\begin{bmatrix} n_1 \\ n_2 \end{bmatrix}$ where accepting states get even rank
  - Initial states: all states of the form $\begin{bmatrix} n_1 \\ \bot \end{bmatrix}$
  - Transitions: all $\begin{bmatrix} n_1 \\ n_2 \end{bmatrix} \xrightarrow{a} \begin{bmatrix} n_1' \\ n_2' \end{bmatrix}$ s.t . ranks don´t increase
- The runs of the automaton on a word w correspond to all the rankings of $dag(w)$.
- Observe: $\bar{A}$ is a NBA even if A is a DBA, because there are many rankings for the same word.

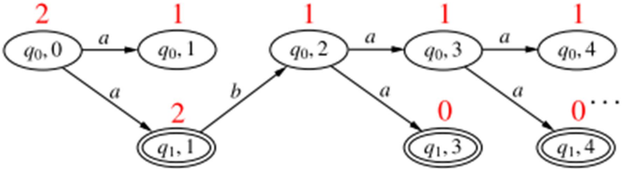
Problems to solve

- How to choose the accepting states?
  - They should be chosen so that a run is accepted iff its corresponding ranking is odd.
- Potentially infinitely many states (because rankings can contain arbitrarily large numbers)

Solving the first problem

- We use owing states and breakpoints again:
  - A breakpoint of a ranking is now a level of the ranking such that no state of the level owes a visit to a node of odd rank.
  - We have again: a ranking is odd iff it has infinitely many breakpoints.
  - We enrich the state with a set of owing states, and choose the accepting states as those in which the set is empty.

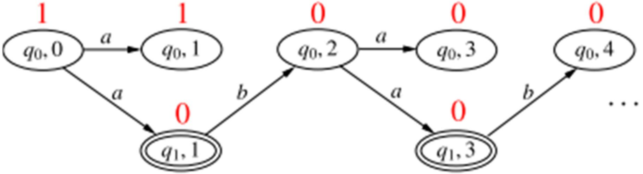
Owing states



$$\begin{bmatrix}2\\ \bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\ 2\end{bmatrix} \xrightarrow{b} \begin{bmatrix}1\\ \bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\ 0\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\ 0\end{bmatrix} \cdots$$

$\{q_0\}$ $\{q_1\}$ $\emptyset$ $\{q_1\}$ $\emptyset$

Owing rankings



$$\begin{bmatrix}1\\ \bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\ 0\end{bmatrix} \xrightarrow{b} \begin{bmatrix}0\\ \bot\end{bmatrix} \xrightarrow{a} \begin{bmatrix}0\\ 0\end{bmatrix} \xrightarrow{b} \begin{bmatrix}0\\ \bot\end{bmatrix} \cdots$$

$\emptyset$ $\{q_1\}$ $\{q_0\}$ $\{q_0, q_1\}$ $\{q_0\}$

Second draft for $\bar{A}$

- For a two-state A (the case of more states is analogous):
  - States: all pairs $\begin{bmatrix} n_1 \\ n_2 \end{bmatrix}, O$ wher accepting states get even rank, and O is set of owing states (of even rank)
  - Initial states: all $\begin{bmatrix} n_1 \\ \bot \end{bmatrix}, \{q_0\}$ where n_1 even if q_0 accepting.
  - Transitions: all $\begin{bmatrix} n_1 \\ n_2 \end{bmatrix}, O \xrightarrow{a} \begin{bmatrix} n_1' \\ n_2' \end{bmatrix}, O'$ s.t. ranks don't increase and owing states are correctly updated
  - Final states: all states $\begin{bmatrix} n_1 \\ n_2 \end{bmatrix}, \emptyset$

- The runs of $\bar{A}$ on a word w correspond to all the rankings of $dag(w)$.
- The accepting runs of $\bar{A}$ on a word w correspond to all the odd rankings of $dag(w)$.
- Therefore: $L(\bar{A}) = \overline{L(A)}$

Solving the second problem

Proposition: If w is rejected by A, then $dag(w)$ has an odd ranking in which ranks are taken from the range $[0,2n]$, where n is the number of states of A. Further, the initial node gets rank $2n$.

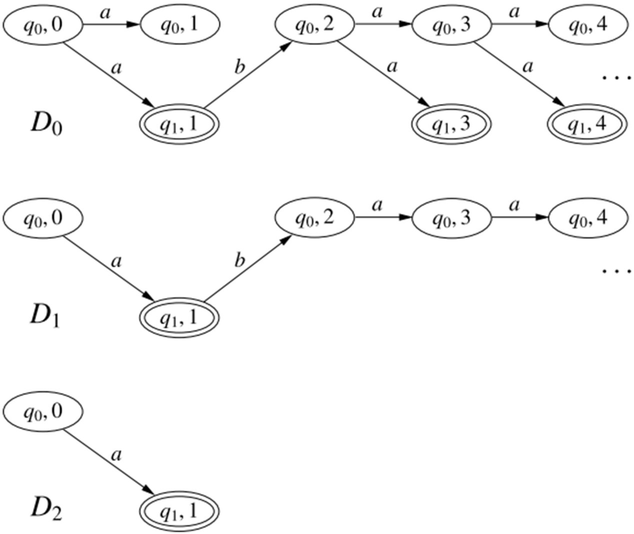
Proof: We construct such a ranking as follows:

- we proceed in $n + 1$ rounds (from round 0 to round n), each round with two steps $k.0$ and $k.1$ with the exception of round n which only has $n.0$
- each step removes a set of nodes together with all its descendants.
- the nodes removed at step $i.j$ get rank $2i + j$
- the rank of the initial node is increased to $2n$ if necessary (preserves the properties of rankings).

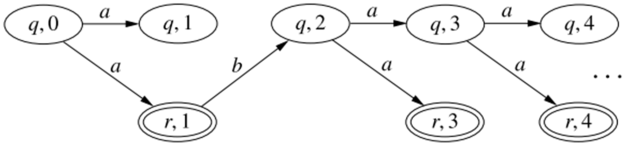
The steps

- Step $i.0$: remove all nodes having only finitely many successors.
- Step $i.1$: remove nodes that are non-accepting and have no accepting descendants

- This immediately guarantees :
  1. Ranks along a path cannot increase.
  2. Accepting states get even ranks, because they can only be removed at step $i.0$
- It remains to prove: no nodes left after $n + 1$ rounds .


q0, 0
a
q0, 1
a
b
q0, 2
a
q0, 3
a
q0, 4
a
a
. . .
D0
q1, 1
q1, 3
q1, 4
q0, 0
q0, 2
a
q0, 3
a
q0, 4
a
b
. . .
D1
q1, 1
q0, 0
a
D2
q1, 1

- To prove: no nodes left after n rounds .
- Each level of a dag has a width



- We define the width of a dag as the largest level width that appears infinitely often.
- Each round decreases the width of the dag by at least 1.
- Since the intial width is at most n after at most n rounds the width is 0, and then step $n.0$ removes all nodes.

Final $\bar{A}$

- For a two-state A (the case of more states is analogous):
  - States: all pairs $\begin{bmatrix} n_1 \\ n_2 \end{bmatrix}, O$ where O set of owing states and accepting states get even rank
  - Initial state: all $\begin{bmatrix} 2n \\ \bot \end{bmatrix}, \{q_0\}$
  - Transitions: all $\begin{bmatrix} n_1 \\ n_2 \end{bmatrix}, O \xrightarrow{a} \begin{bmatrix} n_1' \\ n_2' \end{bmatrix}, O'$ s.t. ranks don‘t increase and owing states are correctly updated
  - Final states: all states $\begin{bmatrix} n_1 \\ n_2 \end{bmatrix}, \emptyset$

An example

- We construct the complements of

  $A_1 = (\{q\}, \{a\}, \delta, \{q\}, \{q\})$ with $\delta(q, a) = \{q\}$

  $A_2 = (\{q\}, \{a\}, \delta, \{q\}, \emptyset)$ with $\delta(q, a) = \{q\}$

- States of A_1:

  $\langle 0, \emptyset\rangle, \langle 2, \emptyset\rangle, \langle 0, \{q\}\rangle, \langle 2, \{q\}\rangle$

- States of A_2:

  $\langle 0, \emptyset\rangle, \langle 1, \emptyset\rangle, \langle 2, \emptyset\rangle, \langle 0, \{q\}\rangle, \langle 2, \{q\}\rangle$

- Initial state of A_1 and A_2: $\langle 2, \{q\}\rangle$

An example

- Transitions of A_1:
  $\langle 2, \{q\}\rangle \xrightarrow{a} \langle 2, \{q\}\rangle$, $\langle 2, \{q\}\rangle \xrightarrow{a} \langle 0, \emptyset\rangle$, $\langle 0, \{q\}\rangle \xrightarrow{a} \langle 0, \{q\}\rangle$
- Transitions of A_2:
  $\langle 2, \{q\}\rangle \xrightarrow{a} \langle 2, \{q\}\rangle$, $\langle 2, \{q\}\rangle \xrightarrow{a} \langle 1, \emptyset\rangle$, $\langle 2, \{q\}\rangle \xrightarrow{a} \langle 0, \emptyset\rangle$,
  $\langle 1, \emptyset\rangle \xrightarrow{a} \langle 1, \emptyset\rangle$, $\langle 1, \emptyset\rangle \xrightarrow{a} \langle 0, \{q\}\rangle$,
  $\langle 0, \{q\}\rangle \xrightarrow{a} \langle 0, \{q\}\rangle$
- Final states of A_1: $\langle 0, \emptyset\rangle, \langle 2, \emptyset\rangle$ (unreachable)
- Final states of A_2: $\langle 0, \emptyset\rangle, \langle 1, \emptyset\rangle, \langle 2, \emptyset\rangle$ (only $\langle 1, \emptyset\rangle$ is reachable)

CompNBA(A)
Input: NBA $A = (Q, \Sigma, \delta, q_0, F)$
Output: NBA $\overline{A} = (\overline{Q}, \Sigma, \overline{\delta}, \overline{q}_0, \overline{F})$ with $L_\omega(\overline{A}) = \overline{L_\omega(A)}$

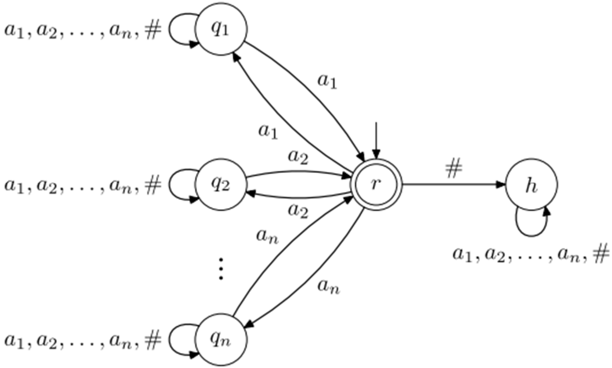
$\overline{Q}, \overline{\delta}, \overline{F} \leftarrow \emptyset$
$\overline{q}_0 \leftarrow [lr_0, \{q_0\}]$
$W \leftarrow \{\ [lr_0, \{q_0\}]\ \}$
while $W \neq \emptyset$ **do**
    pick $[lr, P]$ **from** W; **add** $[lr, P]$ **to** $\overline{Q}$
    if $P = \emptyset$ **then add** $[lr, P]$ **to** $\overline{F}$
    for all $a \in \Sigma, lr' \in \mathfrak{R}$ such that $lr \overset{a}{\mapsto} lr'$ **do**
        if $P \neq \emptyset$ **then** $P' \leftarrow \{q \in \delta(P, a) \mid lr'(q) \text{ is even}\}$
        else $P' \leftarrow \{q \in Q \mid lr'(q) \text{ is even}\}$
        add $([lr, P], a, [lr', P'])$ **to** $\overline{\delta}$
        if $[lr', P'] \notin \overline{Q}$ **then add** $[lr', P']$ **to** W
return $(\overline{Q}, \Sigma, \overline{\delta}, \overline{q}_0, \overline{F})$

Complexity

- A state consists of a level of a ranking and a set of owing states.
- A level assigns to each state a number f $[0,2n]$ or the symbol $\bot$.
- So the complement NBA has at most $(2n+2)^n \cdot 2^n \in n^{O(n)} = 2^{O(n \log n)}$ states.
- Compare with 2^n for the NFA case.
- We show that the $\log n$ factor is unavoidable.

- We define a family $\{L_n\}_{n\geq 1}$ of ω-languages s.t.
  - L_n is accepted by a NBA with $n + 2$ states.
  - Every NBA accepting $\overline{L_n}$ has at least $n! \in 2^{\Theta(n \log n)}$ states.
- The alphabet of L_n is $\Sigma_n = \{1,2, \ldots, n, \#\}$.
- Assign to a word $w \in \Sigma_n$ a graph $G(w)$ as follows:
  - Vertices: the numbers $1,2, \ldots, n$.
  - Edges: there is an edge $i \rightarrow j$ iff w contains infinitely many occurrences of ij.
- Define: $w \in L_n$ iff $G(w)$ has a cycle.

- L_n is accepted by a NBA with $n + 2$ states.



Every NBA accepting $\overline{L_n}$ has at least $n! \in 2^{\Theta(n \log n)}$ states.

- Let τ denote a permutation of $1,2,\ldots,n$.
- We have:
  a) For every τ, the word $(\tau\,\#)^\omega$ belongs to $\overline{L_n}$ (i.e., its graph contains no cycle).
  b) For every two distinct τ_1, τ_2, every word containing inf. many occurrences of τ_1 and inf. many occurrences of τ_2 belongs to L_n.

Every NBA accepting $\overline{L_n}$ has at least $n! \in 2^{\Theta(n \log n)}$ states.

- Assume A recognizes $\overline{L_n}$ and let τ_1, τ_2 distinct. By (a), A has runs ρ_1, ρ_2 accepting $(\tau\_1\ \#)^\omega$, $(\tau_2\ \#)^\omega$. The sets of accepting states visited i.o. by ρ_1, ρ_2 are disjoint.
  - Otherwise we can ``interleave‘‘ ρ_1, ρ_2 to yield an acepting run for a word with inf. Many occurrences of τ_1, τ_2 , contradicting (b).
- So A has at least one accepting state for each permutation, and so at least $n!$ States.