Finite State Automata and Regular Languages

Prof. Licia Sbattella

aa 2007-08

Translated and adapted by L. Breveglieri

FINITE STATE AUTOMATA AND THE RECOGNITION OF REGULAR LANG.

PROBLEM: recognizing whether a string belongs to a given language, before elaborating it.

In order to describe a string recognition procedure, abstract machines called AUTOMATA are used.

In the following there will be shown and discussed:

1. Finite state automata and their relationship with regular languages.
2. Non-deterministic and deterministic behaviour of the recognizer.
3. Construction of the minimum automaton.
4. Composition of recognizers by means of the various operations that preserve the structure of the language family (closure).
5. How to exploit finite state automata to analyze languages.

RECOGNITION ALGORITHMS

DOMAIN: a set of strings over the alphabet Σ.
IMAGE: the answer *yes* or *no* (recognition is a decision problem).

Applying the recognition algorithm α to the string *x* is indicated as α (*x*).
The string *x* is *recognized* (*accepted*) or *unrecognized* (*rejected*)
by α if α (*x*) $=$ yes or α (*x*) $=$ no, respectively.

The language L(α) is defined as aside: $\boxed{L(\alpha) = \left\{ x \in \Sigma^* \mid \alpha(x) = yes \right\}}$

If the language L is semidecidable ($=$ recursively denumerable, but not recursive), it may happen that for some uncorrect string *x* the algorithm α will not terminate, that is α (*x*) is undefined.

Here only decidable languages will be considered, having a recognition algorithm that terminates for every string, whether accepted or rejected.

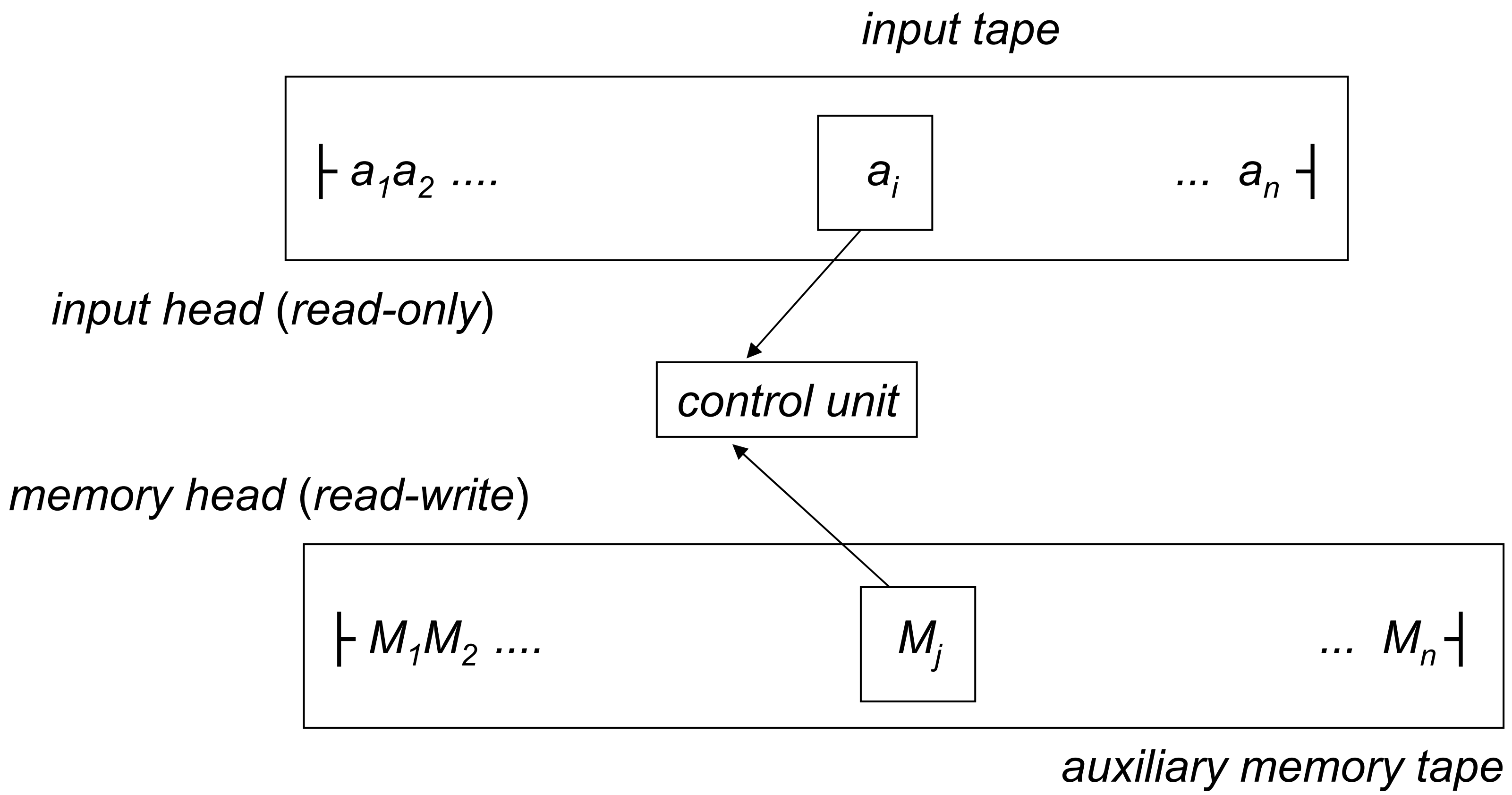
Almost all the problems of some interest here, have solutions with a relatively low time complexity degree (= the number of the computation steps of the recognition algorithm), that is a linear or at worst a polynomial complexity, with respect to the size of the problem (in most cases, the length of the string.)

In the theory and practice of formal languages, one computation step is a single atomic operation of the abstract recognition machine (the automaton), which can manipulate only one symbol at a time. Therefore, it is customary to present the recognition algorithm by means of an automaton of some kind, whether a recognizer or a transtider machine, mainly for the following reasons:

1. To outline the correspondence between the various families of languages and the respective generative devices (that is, the grammars).
2. To skip any unnecessary and premature reference to the effective implementation of the algorithm in some programming language.

Once the recognizer automaton has been designed, it is relatively easy to write the corresponding program (in C or similar programming languages).

GENERAL MODEL OF A RECOGNIZER AUTOMATON:



THE AUTOMATON ANALYZES THE INPUT STRING and executes a series of moves. Each move depends on the symbols currently pointed by the heads and also on the current state. The move may have the following effects:

1. Shift the input head of on position leftwards or rightwards.
2. Replace the current memory symbol by a new one and shift the memory head of one position leftwards or righwards.
3. Change the current state.

Some of the above listed operation may be absent.

ONE-WAY AUTOMATON: the input head can be shifted only rightwards. In the following only this case is considered. It corresponds to scanning the input string left to right only once.

NO AUXILIARY MEMORY: this is the celebrated FINITE STATE AUTOMATON. It is the machine that recognizes regular languages.

AUXILIARY MEMORY structured and handled as a PUSHDOWN STACK. This is the as much known machine that recognizes free languages, A CONFIGURATION (instantaneous) is the set of the three components that determine the behaviour of the automaton:

- the still unread part of the input tape
- the contents of the memory tape and the position of the memory head
- the current state of the control unit

INITIAL CONFIGURATION: the input head is positioned on the symbol immediately following the start marker, the control unit is in a specific state (initial state), and the memory tape contains a special initial symbol only. The configuration of the automaton changes through a series of transitions, each driven by a move. The whole series is the computation of the automaton.

DETERMINISTIC BEHAVIOUR – in every instantaneous configutation, at most one move is possible (or none). Otherwise (two or more moves), the automaton is said to be non-deterministic (or indeterministic).

FINAL CONFIGURATION: the control unit is a special state qualified as final and the input head is positioned on the end-marker of the string to be recognized (sometimes the final configuration is characterized by a condition for the memory tape: to be empty or to contain only one special final symbol).

A SOURCE STRING x IS ACCEPTED BY THE AUTOMATON if it starts from the initial input configuration $\vdash x \dashv$, executes a series of transitions (moves) and reaches a final configuration. If the automaton is non-deterministic, it could reach the same final configuration in two or more different sequences of transitions, or even reach two or more different final configurations.

THE COMPUTATION terminates either because the automaton has reached a final configuration or because the automaton can not execute any more transition step (due to the fact that in the current instantaneous configuration, there is not any possible move). In the former case the input string is accepted, in the latter one it is rejected.

THE SET OF ALL THE STRINGS ACCEPTED BY THE AUTOMATON constitutes the language RECOGNIZED (or ACCEPTED, or DEFINED) BY THE AUTOMATON.

Two automata that accept the same language are said to be EQUIVALENT.
CAUTION: two equivalent automata may well be modeled differently, nor need they have necessarly the same computational complexity.

REGULAR LANGUAGES, which are recognized by finite state automata, are a subfamily of the languages recognizable in real time by a TURING MACHINE. FREE LANGUAGES are a subfamily of the languages recognized by a TURING MACHINE that have a polynomial time complexity.

FINITE STATE AUTOMATA (or simply FINITE AUTOMATA)

Many applications of information science and engineering make use of finite state automata: digital design, theory of control, communication protocols, the study of system realiability and security, etc.

IN THE FOLLOWING:
- state-transition graphs
- finite automata as recognizers of regular languages defined by uni-linear grammars
- deterministic and non-deterministic behaviour
- how to transform a non-det. automaton into a det. one
- language subfamilies characterized by a local recognition test

STATE-TRANSITION GRAPH

A finite state automaton consists of the following elements:

- the input tape (containing the input string $x \in \Sigma^*$)
- the control unit and its finite memory, containing the state table
- the input head (initially positioned at the start-marker of *x*), which at every move is shifted rightwards as far as the end-marker of *x* is reached, or an error is encountered

After reading an input character, the automaton updates the current state of the control unit.

After scanning the string *x*, the automaton recognizes *x* or not depending on the current state.

STATE-TRANSITION GRAPH (continued)

STATE-TRANSITION GRAPH: is a directed graph that represents the automaton, and consists of the following elements:

NODES: represent the states of the control unit
ARCS: represent the moves of the automaton

Each arc is labeled by an input symbol, and represents the move that is enabled when the current state matches the outgoing state of the arc and the current input symbol matches the label of the arc.

The state-transition graph has a unique INITIAL STATE (if it had two or more, it would be non-deterministic), but may well have no, one or more FINAL STATES (if it has none, it recognizes the empty set).

The graph can be represented under the form of an INCIDENCE MATRIX. Each entry of the matrix is indexed by the current state and the input symbol, and contains the next state. The incidence matrix is often calle STATE TABLE.

EXAMPLE – decimal constants in numerical form

$$\begin{tikzcd}
 & & q_2 \arrow[d, "\bullet"] & & \\
{} \arrow[r] & q_0 \arrow[ur, "0"] \arrow[dr, "\Delta"'] & q_3 \arrow[r, "0 \text{ U } \Delta"] & q_4 \arrow[loop below, "0 \text{ U } \Delta"] \arrow[r] & {} \\
 & & q_1 \arrow[u, "\bullet"'] \arrow[loop below, "0 \text{ U } \Delta"] & &
\end{tikzcd}$$

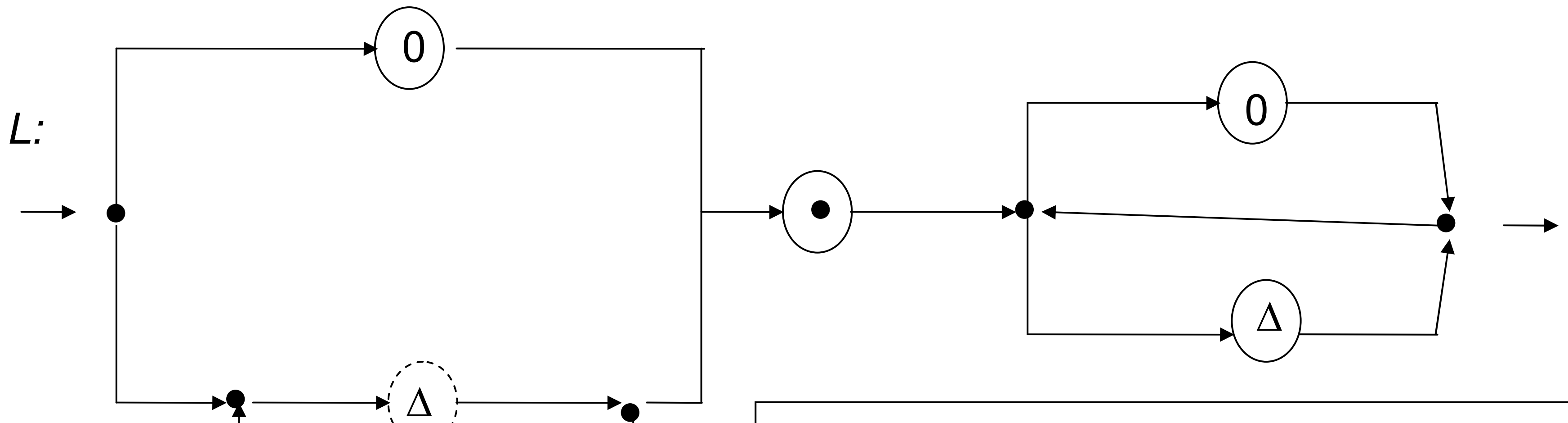
current	current input character				
state	0	1	...	9	•
$\rightarrow q_0$	q_2	q_1	...	q_1	–
q_1	q_1	q_1	...	q_1	q_3
q_2	–	–	...	–	q_3
q_3	q_4	q_4	...	q_4	–
$q_4 \rightarrow$	q_4	q_4	...	q_4	–

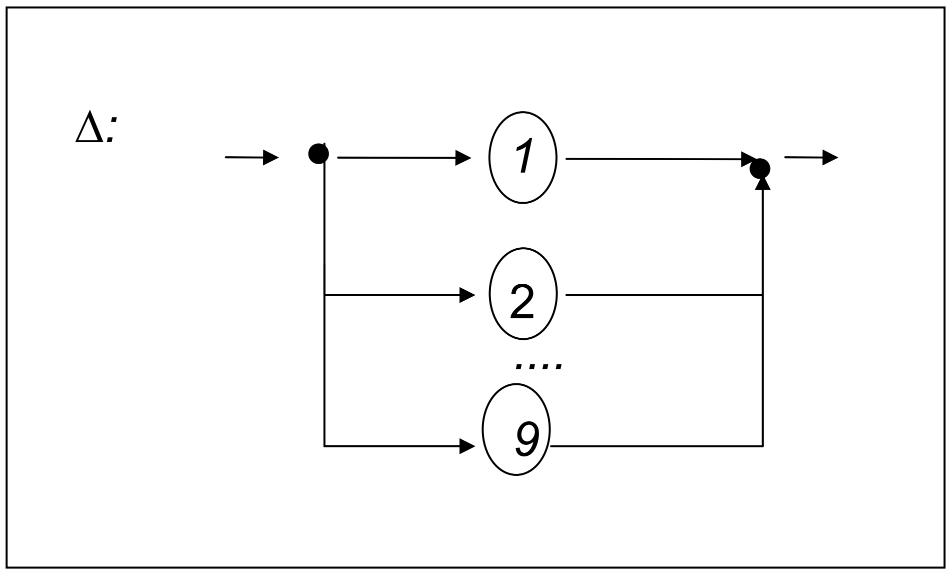
$$\Sigma = \Delta \bigcup \{0, \square\}, \text{ where } \Delta = \{1,2,3,4,5,6,7,8,9\}$$

$$L = (0 \bigcup \Delta(0 \bigcup \Delta)^* \bullet (0 \bigcup \Delta)^+$$

NOTE: in the drawing, topologically identical arcs are grouped into one (from q_0 to q_1 there shluld be 9 arcs ....).

Here follow the syntax diagrams of the automaton (similar to those that can be used to represent EBNF productions), which are the DUAL FORM of the state-transition graph.





DETERMINISTIC FINITE STATE AUTOMATON

A DETERMINISTIC FINITE AUTOMATON M consists of five elements:

1. Q, the *set of states* (finite and not empty)
2. Σ, the *input alphabet* (or terminal alphabet)
3. the *transition function* δ: $(Q \times \Sigma) \rightarrow Q$
4. $q_0 \in Q$, the *initial state*
5. $F \subseteq Q$, the *set of final states* (may be empty)

The transition function encodes the moves of the automaton: $\boxed{\delta(q_i, a) = q_j}$

$$\boxed{\begin{array}{l} x = ab \quad \delta(q_0, a) = q_1 \quad \delta(q_1, b) = q_2 \\ \delta(q_0, ab) = q_2 \\ \\ \forall q \in Q: \quad \delta(q, \varepsilon) = q \end{array}}$$

When M is in the current state q_i and reads the input symbol *a*, it switches the current state to q_j.

If δ (q_i, *a*) is undefined, the automaton stops (enters an error state and rejects the input string).

the domain now is $(Q \times \Sigma^*)$ and the trans. funct. is now

$\forall$ char. $a \in \Sigma$, $\forall$ string $y \in \Sigma^*$:

$\delta(q, ya) = \delta(\delta(q, y), a)$

EXECUTION OF A COMPUTATION (= sequence of TRANSITIONS):

$$\boxed{\delta(q, y) = q'}$$

The transition function δ is extended by posing as aside if, and only if, there exists a path from state *q* to state *q*', labeled by the input string *y*. If the automaton moves through such a path, it recognizes the string *y*.
The automaton executes a sequence of transitions, that is a computation.

RECOGNITION OF A STRING:
A string *x* is recognized (or accepted) if and only if when the automaton moves through a path labeled by *x*, it starts from the initial state and ends at one of the final states:

$$\boxed{\delta(q_0, x) \in F}$$

The empty string is accepted if and only if the initial state is final as well.

THE LANGUAGE RECOGNIZED BY THE AUTOMATON M:

$$\boxed{L(M) = \left\{ x \in \Sigma^* \mid x \quad \text{is recognized by} \quad \mathrm{M} \right\}}$$

The family of the languages recognized by finite stata automata is named family of *finite state languages.*

Two finite automata are said to be equivalent if they recognize the same language.

The time complexity of finite state automata is optimal: the string *x* is accepted or rejected in real-time, that is in a number of steps equal to the length of the string itself. Since it takes exactly as many steps to scan the string from left to right, recognition time complexity could not be lower than this.

EXAMPLE – decimal constants in numercal form (continued)

The automaton M is defined as follows:

$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

$$\Sigma = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, \square\}$$

$$q_0 = q_0$$

$$F = \{q_4\}$$

examples of transitions:

$$\delta(q_0, 3\square1) = \delta(\delta(q_0, 3\square), 1) = \delta(\delta(\delta(q_0, 3), \square), 1) =$$

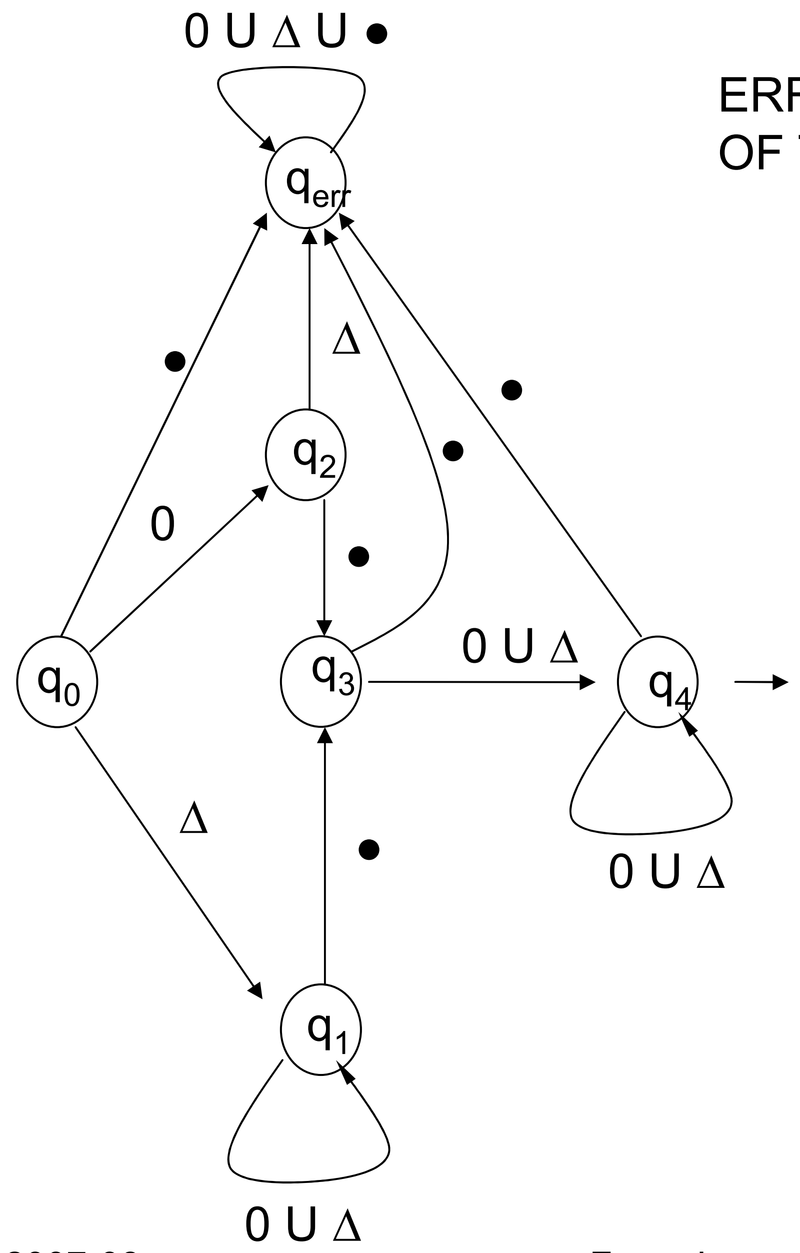
$$= \delta(\delta(q_1, \square), 1) = \delta(q_3, 1) = q_4$$

$q_4 \in F$ string $3\square1$ is accepted

rejected:

$\delta(q_0, 3\square) = q_3$ is not final $3\square \notin L$

$\delta(q_0, 02) = \delta(\delta(q_0, 0), 2) = \delta(q_2, 2)$ is undefined $02 \notin L$



ERROR STATE AND NATURAL COMPLETION OF THE AUTOMATON

$\boxed{q_{err} = \text{error state}}$

It is always possible to complete the automaton by adding the error state, without changing the accepted language.

$\forall$ state $q \in Q$ $\quad \forall$ char $a \in \Sigma$

if $\delta(q,a)$ is undefined

set $\delta(q,a) = q_{err}$

$\forall$ char $a \in \Sigma$ set $\delta(q_{err},a) = q_{err}$

AUTOMATON IN REDUCED FORM

An automaton may contain unuseful parts (i.e., states), which do not give any contribution to the recognition process, and which usually can be eliminated.

A state *q* is said to be REACHABLE FROM THE STATE *p* if there exists a computation that moves the automaton from the current state *p* to the current state *q*.

A state *q* is said to be REACHABLE if *q* it reachable from the initial state.

A state *q* said to be is DEFINED if a final state is reachable from state *q*.

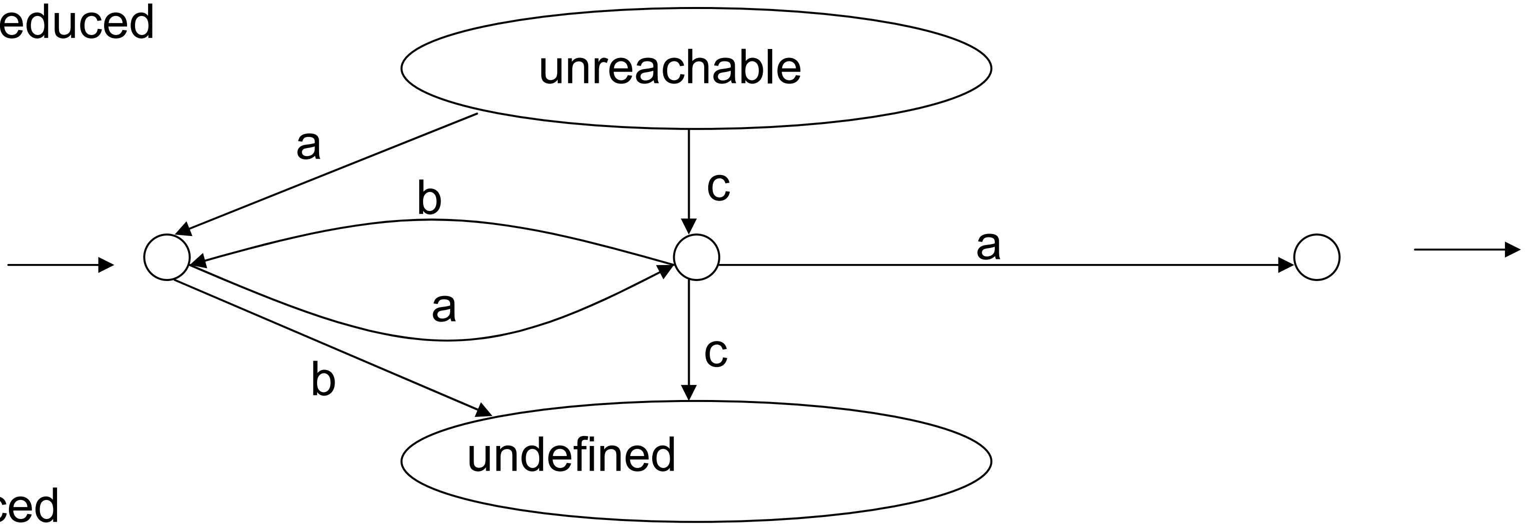
A state *q* is said to be USEFUL (or TRIM) if *q* is both reachable and defined (that is, if *q* is placed on a path connecting the initial state to a final state).

An automaton A is said to be in REDUCED FORM (or in TRIM FORM) if every state of A is useful.

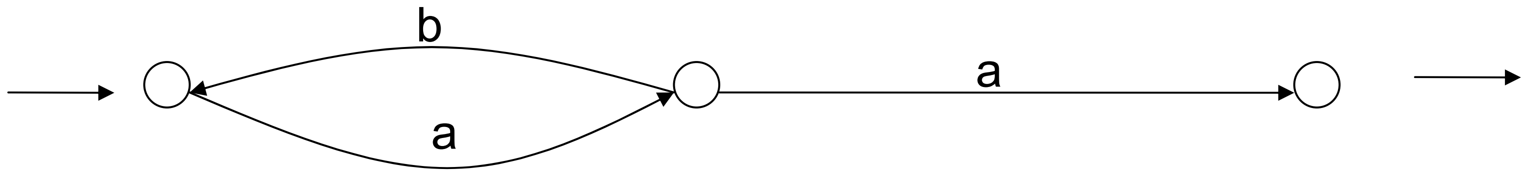
PROPERTY – Every finite state automaton has an equivalent reduced form. To reduce an automaton: first identify all the states that are not useful, then strip them off the automaton, along with all the incoming and outgoing arcs.

EXAMPLE – elimination of the unuseful states

not reduced



reduced



MINIMUM AUTOMATON

PROPERTY – For every finite state language, there exists one and only one deterministic finite state recognizer that has the smallest possible number of states (and is called the *minimum automaton*).

UNDISTINGUISHABLE STATE – A state *p* is UNDISTINGUISHABLE FROM A STATE *q*, if and only if for every input string *x* either both the next states δ (*p* ,x) and δ (*q*, *x*) are final, or neither one is; equivalently if one starts from *p* and *q*, scans the string *x* but does not reach a final state in either case.

Two undistinguishable states can be MERGED and thus the number of states of the automaton can be reduced, without changing the recognized language.

UNDISTINGUISHABILITY is a reflexive, symmetric and transitive binary relation, and therefore is an *equivalence* relation.

p is DISTINGUISHABLE from *q* if
1. *p* is final and *q* is not (or viceversa)
2. δ (*p*, *a*) is distinguishable from δ (*q*, *a*)

EXAMPLE

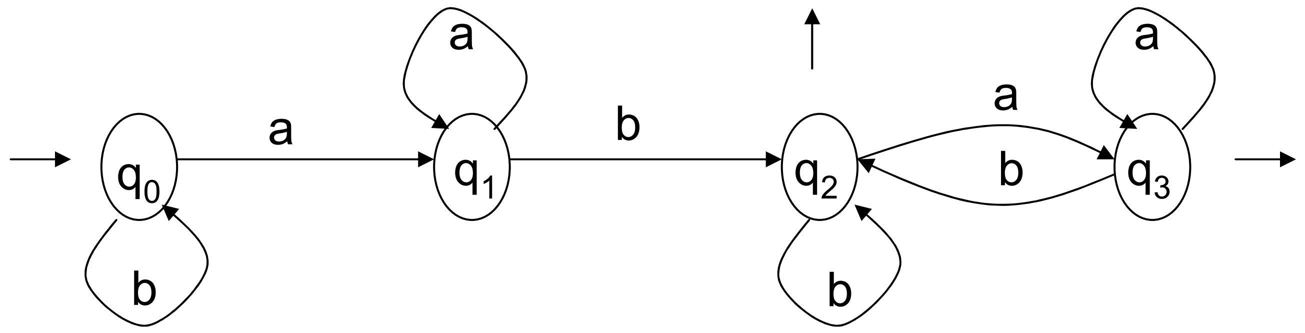


Table of indistinguishability:

q0				
q1	(1,1)(0,2)			
q2	X	X		
q3	X	X	(3,3)(2,2)	
	q0	q1	q2	q3

Undistinguishable state pairs: q_2 and q_3

q0				
q1	X			
q2	X	X		
q3	X	X		
	q0	q1	q2	q3

Equivalence classes of the undistinguishability relation: $[q_0]$, $[q_1]$, $[q_2, q_3]$.

REDUCTION OF THE NUMBER OF STATES (MINIMIZATION)

The classes of equivalence of the undistinguishability relation of the original automaton M are the states of the minimum automaton M’, equivalent to M. To define the transition function of M’ it suffices to state that there is an arc from the class [..., p_r, ...] to the class [..., q_s, ...], as shown aside:

$$\boxed{[\dots, p_r, \dots] \xrightarrow{b} [\dots, q_s, \dots]}$$

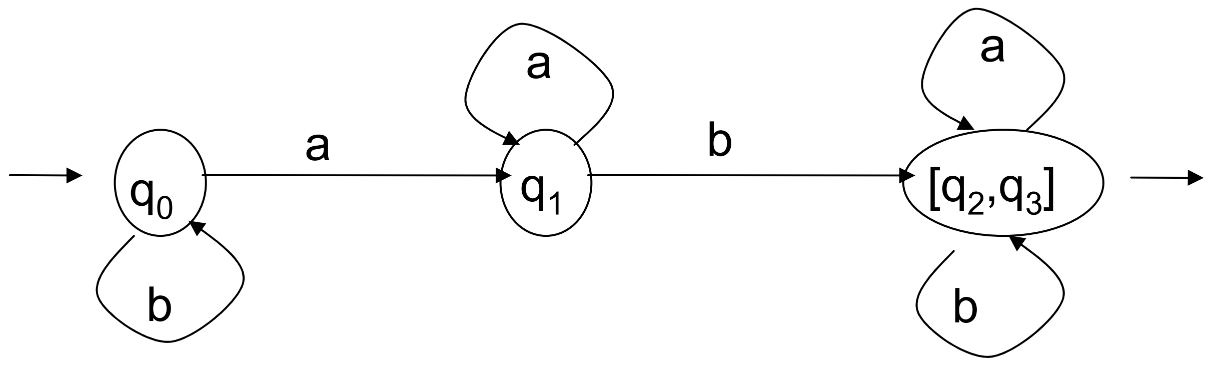
if and only if in M there is an arc as shown aside:

$$\boxed{p_r \xrightarrow{b} q_s}$$

that is, if there is an arc between two states belonging to the two classes.
CAUTION: the same arc of M’ may originate from two or more arcs of M.

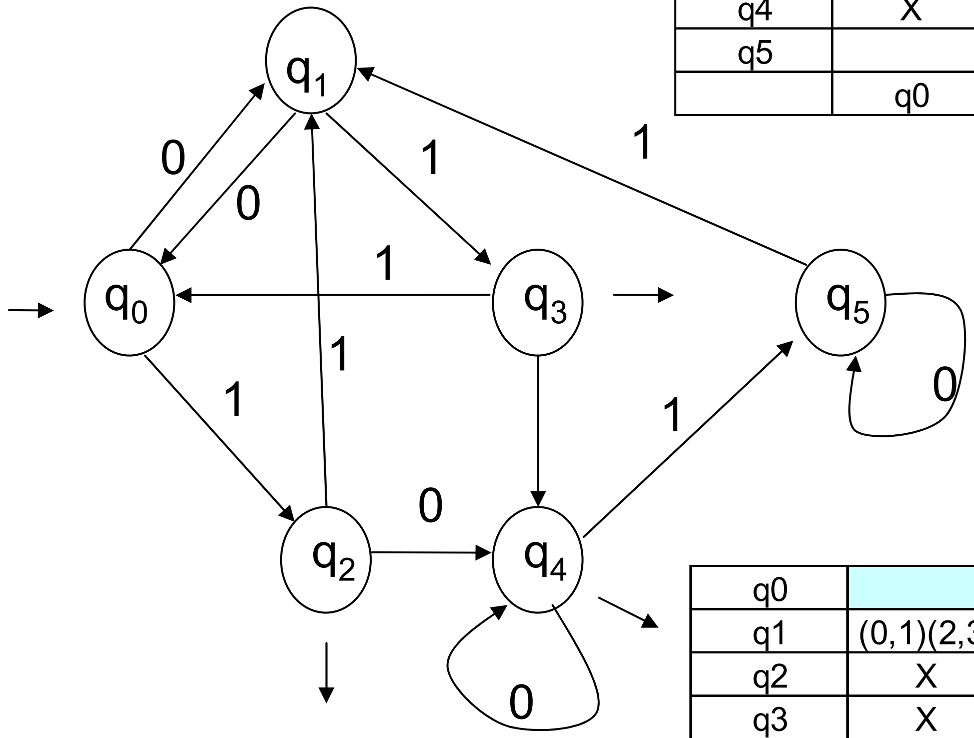
EXAMPLE (continued)

Minimum automaton of the previous example:

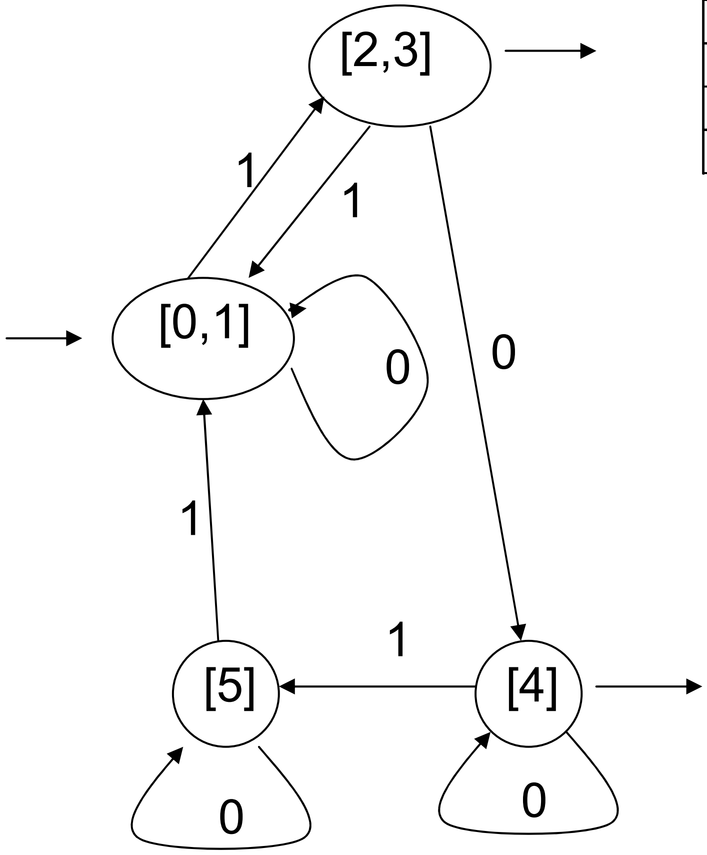


EXAMPLE

q0						
q1						
q2	X	X				
q3	X	X				
q4	X	X				
q5			X	X	X	
	q0	q1	q2	q3	q4	q5



q0						
q1	(0,1)(2,3)					
q2	X	X				
q3	X	X	(4,4)(0,1)			
q4	X	X	(4,4)(1,5)	(4,4)(0,5)		
q5	(1,5)(2,1)	(0,5)(3,1)	X	X	X	
	q0	q1	q2	q3	q4	q5



q0						
q1						
q2	X	X				
q3	X	X				
q4	X	X	X	X		
q5	X	X	X	X	X	
	q0	q1	q2	q3	q4	q5

Equivalence classes of M:
$[q_0, q_1], [q_2, q_3], [q_4], [q_5]$

How does the minimization algorithm behave when the transition function δ is not total ? Assume of modifying the automaton M by removing transition $\delta (q_3, a) = q_3$. To do so, redefine $\delta (q_3, a) = q_{err}$. The states q_2 and q_3 are thus distinguishable, because $\delta (q_2, a) = q_3$ and $\delta (q_3, a) = q_{err}$, and q_3 is distinguishable from q_{err}. Therefore M is minimum.

It is possible to prove that the minimum deterministic automaton is unique. This property does not hold for the minimum non-deterministic automaton; which usually is not unique (but in some special cases).

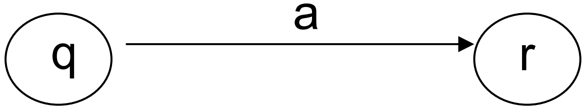
The uniqueness of the minumum automaton offers a way to check whether two deterministic finite state automata are equivalent. First minimize the number of states of both and obtain the corresponding minimum automata; then check whether the two minimum state-transition graphs are topologically indentical (that is, if nodes and arcs can be exactly overlapped). The two original automata are equivalent if and only of the two minimum corresponding one are topologically identical.
CAUTION: there exist different algorithms as well.
CAUTION: it does not work for non-det. automata (non-unique min. form !)

FROM THE AUTOMATON TO THE GRAMMAR

FINITE STATE AUTOMATA AND UNI-LINEAR GRAMMARS
(or type 3 Chomsky grammars) deal with the same language family.

Procedure to construct a finite automaton corresponding to a right linear grammar.
First it is necessary to modify the definition of finite automaton, in order to include explicitly the notion of inteterminism.
The states of the automaton Q are the non-terminal symbols of the grammar G, and the initial state is the axiom.

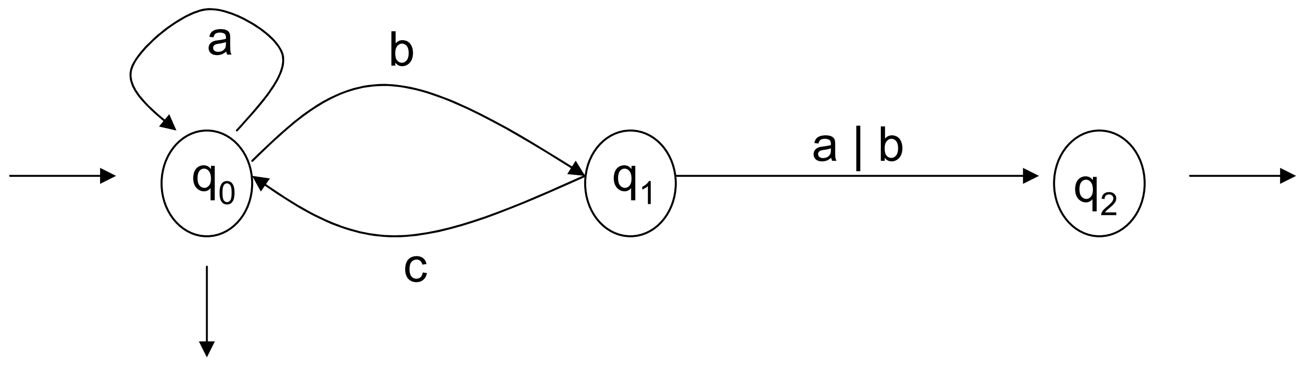
$$\boxed{\begin{array}{l} q \to ar \\ \text{if } q \in F \\ q \to ar \quad q \to \varepsilon \end{array}}$$



There is a one-to-one (or bijective) correspondence between the computations of the automaton and the derivations of the grammar, as shown aside:

$$\boxed{q_0 \overset{+}{\Rightarrow} x}$$

EXAMPLE



$$q_0 \to aq_0 \mid bq_1 \mid \varepsilon$$
$$q_1 \to cq_0 \mid aq_2 \mid bq_2$$
$$q_2 \to \varepsilon$$

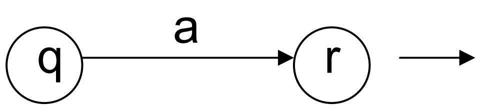
recognition of abc

$$q_0 \Rightarrow bq_1 \Rightarrow bcq_0 \Rightarrow bcaq_0 \Rightarrow bca\varepsilon$$

Non-nullable form of the grammar:

In this version of the unilinear grammar, a move into a final state is mapped onto two production rules, as below:

$$Null = \{q_0, q_2\}$$
$$q_0 \to aq_0 \mid bq_1 \mid a \mid \varepsilon$$
$$q_1 \to cq_0 \mid aq_2 \mid bq_2 \mid a \mid b \mid c$$



$$q \to ar \mid a$$

Bibliography

– S. Crespi Reghizzi, *Linguaggi Formali e Compilazione*, Pitagora Editrice Bologna, 2006

– Hopcroft, Ullman, *Formal Languages and their Relation to Automata*, Addison Wesley, 1969

– A. Salomaa – *Formal Languages*, Academic Press, 1973

– D. Mandrioli, C. Ghezzi – *Theoretical Foundations of Computer Science*, John Wiley & Sons, 1987

– L. Breveglieri, S. Crespi Reghizzi, *Linguaggi Formali e Compilatori: Temi d’Esame Risolti,* web site (eng + ita)