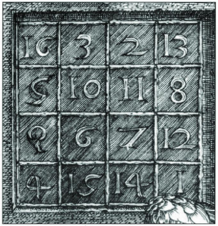
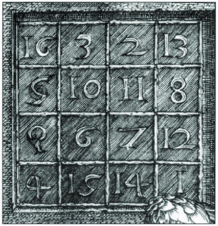
Logică și Structuri Discrete -LSD



Cursul 14 – Automate. Expresii regulate. Gramatici. Mașini Turing

dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro



Limbaje și Automate finite deterministe

Automate finite nedeterministe (NFA)

Expresii regulate

Gramatici

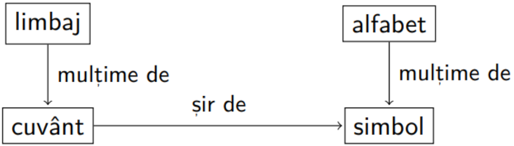
Mașini Turing

Ce e un limbaj

Alfabetul e o mulțime de *simboluri* (caractere)
$\{a, b, c\}$ sau $\{0, 1\}$ sau $\{0, 1, \ldots, 9\}$, ...

Cu simbolurile din alfabet putem forma *șiruri* (*cuvinte*, secvențe):
aba, 010010, 437, ...

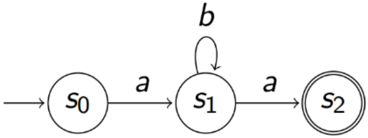
Un *limbaj* e o *mulțime de cuvinte* (șiruri)
ca orice mulțime, definită explicit: $\{a, ab, ac, abc\}$
sau după o regulă: șiruri de *a*, *b*, încep cu *a*, mai mulți *a* decât *b*

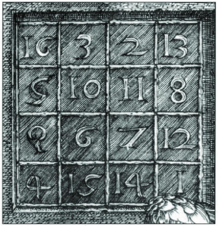


schemă: http://web.stanford.edu/class/cs103/lectures/14/Small14.pdf

Exemplu de automat determinist

automat care acceptă cuvinte cu oricâți de *b* (incl. 0) între doi *a*





Limbaje și Automate finite deterministe

Automate finite nedeterministe (NFA)

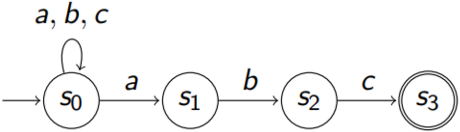
Expresii regulate

Gramatici

Mașini Turing

Automate finite nedeterministe (NFA): Exemplu (1)

Exemplu: toate șirurile de *a, b, c* care se *termină* în *abc*



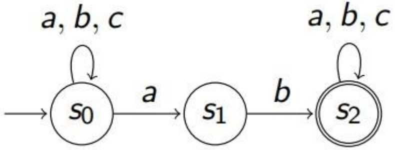
Din s_0, primind simbolul *a*, automatul poate
– rămâne în s_0
– trece în s_1
⇒ automatul poate urma *una din mai multe* căi

Un NFA acceptă dacă *există* o alegere ducând în stare acceptoare.

Dacă pentru un șir *...abc* alegem să trecem în s_1 la simbolul *a* (antepenultimul simbol), șirul va fi acceptat.

Automate finite nedeterministe (NFA): Exemplu (2)

Toate șirurile de *a, b, c* care *conțin* un subșir *ab*



Odată găsit *ab*, șirul e bun, oricum ar continua
tranzițiile din starea acceptoare trec tot în stare acceptoare

Avantajele NFA:

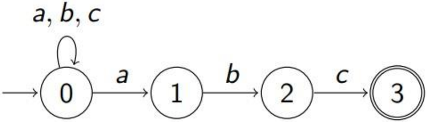
uneori se scrie mai ușor decât un automat determinist
(trebuie să descriem calea acceptoare, nu toate celelalte)

e util când *specificăm* un sistem: putem lăsa deschise mai multe posibilități, ne permite o alegere la implementare

Automate deterministe și nedeterministe

Orice automat nedeterminist are un automat determinist echivalent (acceptă aceleași șiruri). Prezentăm cum facem conversia.

Conversie NFA-DFA (exemplu)

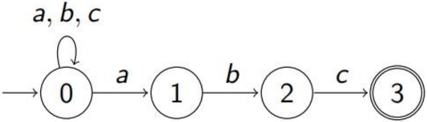


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}

Când obținem o nouă mulțime (roșu) adăugăm o linie la tabel.

Conversie NFA-DFA (exemplu)

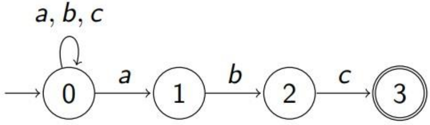


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}
{0, 1}	{0, 1}	{0, 2}	{0}

Când obținem o nouă mulțime (roșu) adăugăm o linie la tabel.

Conversie NFA-DFA (exemplu)

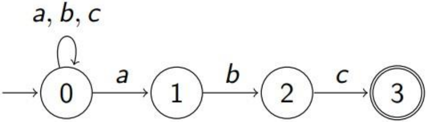


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}
{0, 1}	{0, 1}	{0, 2}	{0}
{0, 2}	{0, 1}	{0}	{0, 3}

Când obținem o nouă mulțime (roșu) adăugăm o linie la tabel.

Conversie NFA-DFA (exemplu)

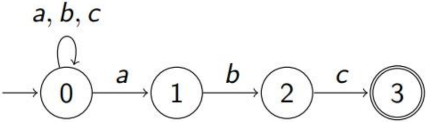


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}
{0, 1}	{0, 1}	{0, 2}	{0}
{0, 2}	{0, 1}	{0}	{0, 3}
{0, 3}	{0, 1}	{0}	{0}

Când obținem o nouă mulțime (roșu) adăugăm o linie la tabel.

Conversie NFA-DFA (exemplu)



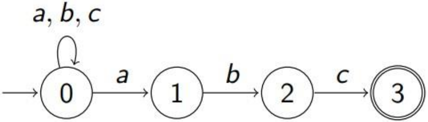
Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}
{0, 1}	{0, 1}	{0, 2}	{0}
{0, 2}	{0, 1}	{0}	{0, 3}
{0, 3}	{0, 1}	{0}	{0}

Când obținem o nouă mulțime (roșu) adăugăm o linie la tabel.

Fiecare mulțime obținută devine o stare în DFA-ul rezultat

Conversie NFA-DFA (exemplu)

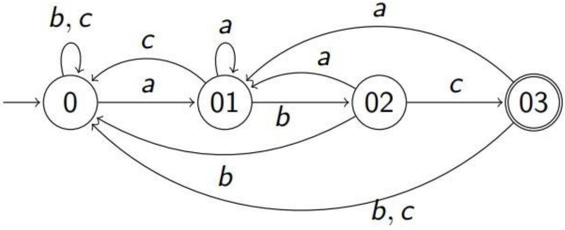


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}
{0, 1}	{0, 1}	{0, 2}	{0}
{0, 2}	{0, 1}	{0}	{0, 3}
{0, 3}	{0, 1}	{0}	{0}

Când obținem o nouă mulțime (roșu) adăugăm o linie la tabel.

Fiecare mulțime obținută devine o stare în DFA-ul rezultat



Stările acceptoare sunt cele care conțin o stare acceptoare din automatul inițial.

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

	a	d
{1}		

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

	a	d
{1}	{2, 4}	{5}

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}
{5}	{2, 4, 6, 8}	{1, 3, 7, 9}

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}
{5}	{2, 4, 6, 8}	{1, 3, 7, 9}
{1, 3, 5, 7}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

	a	d
$\{1\}$	$\{2, 4\}$	$\{5\}$
$\{2, 4\}$	$\{1, 3, 5, 7\}$	$\{2, 4, 6, 8\}$
$\{5\}$	$\{2, 4, 6, 8\}$	$\{1, 3, 7, 9\}$
$\{1, 3, 5, 7\}$	$\{2, 4, 6, 8\}$	$\{1, 3, 5, 7, 9\}$
$\{2, 4, 6, 8\}$	$\{1, 3, 5, 7, 9\}$	$\{2, 4, 6, 8\}$

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

	a	d
$\{1\}$	$\{2, 4\}$	$\{5\}$
$\{2, 4\}$	$\{1, 3, 5, 7\}$	$\{2, 4, 6, 8\}$
$\{5\}$	$\{2, 4, 6, 8\}$	$\{1, 3, 7, 9\}$
$\{1, 3, 5, 7\}$	$\{2, 4, 6, 8\}$	$\{1, 3, 5, 7, 9\}$
$\{2, 4, 6, 8\}$	$\{1, 3, 5, 7, 9\}$	$\{2, 4, 6, 8\}$
$\{1, 3, 7, 9\}$	$\{2, 4, 6, 8\}$	$\{5\}$

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

	a	d
$\{1\}$	$\{2, 4\}$	$\{5\}$
$\{2, 4\}$	$\{1, 3, 5, 7\}$	$\{2, 4, 6, 8\}$
$\{5\}$	$\{2, 4, 6, 8\}$	$\{1, 3, 7, 9\}$
$\{1, 3, 5, 7\}$	$\{2, 4, 6, 8\}$	$\{1, 3, 5, 7, 9\}$
$\{2, 4, 6, 8\}$	$\{1, 3, 5, 7, 9\}$	$\{2, 4, 6, 8\}$
$\{1, 3, 7, 9\}$	$\{2, 4, 6, 8\}$	$\{5\}$
$\{1, 3, 5, 7, 9\}$	$\{2, 4, 6, 8\}$	$\{1, 3, 5, 7, 9\}$

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

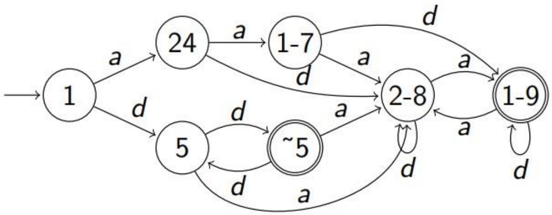
stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d : mută diagonal

	a	d
$\{1\}$	$\{2,4\}$	$\{5\}$
$\{2,4\}$	$\{1,3,5,7\}$	$\{2,4,6,8\}$
$\{5\}$	$\{2,4,6,8\}$	$\{1,3,7,9\}$
$\{1,3,5,7\}$	$\{2,4,6,8\}$	$\{1,3,5,7,9\}$
$\{2,4,6,8\}$	$\{1,3,5,7,9\}$	$\{2,4,6,8\}$
$\{1,3,7,9\}$	$\{2,4,6,8\}$	$\{5\}$
$\{1,3,5,7,9\}$	$\{2,4,6,8\}$	$\{1,3,5,7,9\}$

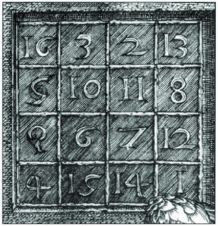


1-7 = $\{1, 3, 5, 7\}$

2-8 = $\{2, 4, 6, 8\}$

$\tilde{}5$ = $\{1, 3, 7, 9\}$

1-9 = $\{1, 3, 5, 7, 9\}$



Limbaje și Automate finite deterministe
Automate finite nedeterministe (NFA)
Expresii regulate
Gramatici
Mașini Turing

Putem exprima mai concis definiția unui limbaj?

Un limbaj = o mulțime de cuvinte peste un alfabet

Adesea ne interesează cuvinte cu structură simplă, “regulată”:
- un *întreg*: o secvență de cifre, eventual cu semn
- un *real*: parte întreagă + parte zecimală (una din ele opțională), exponent opțional
- un *identificator*: litere, cifre, _ începând cu literă sau _
- *nume de fișiere*: 01−*titlu*.mp3, 02−*alttitlu*.mp3, ...

Unele limbaje pot fi recunoscute eficient de *automate finite*
- dar scrierea automatului ia efort
- ⇒ se pot scrie mai simplu ca *expresii regulate*

Expresii regulate: definiție formală

O expresie regulată descrie un limbaj (regulat).

O expresie regulată peste un alfabet Σ e fie:

3 cazuri de bază:

- $\emptyset$ — limbajul vid
- ε — limbajul $\{\varepsilon\}$ (cu șirul vid)
- a — limbajul $\{a\}$ cu $a \in \Sigma$ (un cuvânt de o literă)

3 cazuri recursive: date e_1, e_2 expresii regulate, putem forma:

- $e_1 + e_2$ — *reuniunea* limbajelor
  în practică, notată adesea $e_1|e_2$ (*alternativă*, “sau”)
- $e_1 \cdot e_2$ — *concatenarea* limbajelor
- $e_1{}^*$ — *închiderea Kleene* a limbajului

Reguli de scriere și exemple

Omitem paranteze când sunt clare din relațiile de precedență cel mai prioritar: $*$, apoi concatenare și apoi reuniune $+$ punctul pentru concatenare se omite

În practică se mai folosesc abrevierile

$e?$ *pentru* $e + \varepsilon$ *(e, opțional)*

e^+ pentru $e^* \setminus \varepsilon$ (e, cel puțin o dată)

$(0 + 1)^*$ mulțimea tuturor șirurilor din 0 sau 1

$(0 + 1)^*0$ ca mai sus, încheiat cu 0 (numere pare în binar)

$1(0 + 1)^* + 0$ numere binare, fără zerouri inițiale inutile

Orice expresie regulată e recunoscută de un automat

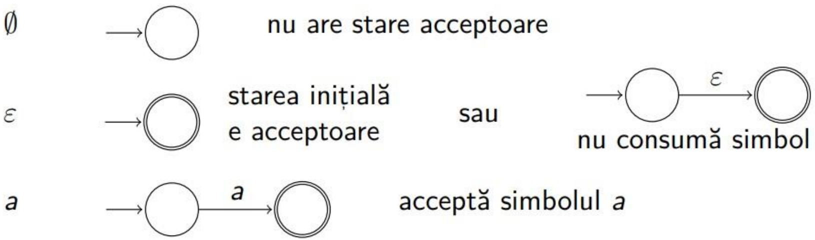
Construcție dată de Ken Thompson (creatorul UNIX, premiul Turing 1983)

Definim prin *inducție structurală*

cum traducem cele 3 *cazuri de bază* de expresie regulată

cum *combinăm* automatele în cele 3 *cazuri recursive*

⇒ descompunând, convertim *orice expresie regulată* în automat



în cele trei cazuri recursive, *combinăm* automatele limbajelor date

⇒ *automat finit nedeterminist* cu tranziții *ε* (nu consumă simbol)

Important Automate finite

Un *automat* finit determinist definește un *limbaj acceptat*.

Un astfel de limbaj se numește *limbaj regulat*.

El poate fi exprimat și printr-o *expresie regulată*.

Intersecția, reuniunea, și complementul limbajelor regulate produc *limbaje regulate*, la fel concatenarea și închiderea Kleene.

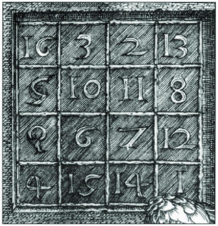
deci pot fi *recunoscute de automate finite*

Automatele finite *nedeterministe* se pot transforma în *deterministe*

deci recunosc tot limbaje regulate

dar numărul de stări poate crește exponențial

Automatele deterministe și nedeterministe și expresiile regulate au *aceeași putere expresivă* (descriu limbaje regulate).



Limbaje și Automate finite deterministe
Automate finite nedeterministe (NFA)
Expresii regulate
Gramatici
Mașini Turing

Limbaje formale, în general

Dorim să:

descriem un limbaj (cât mai simplu/clar/concis)

recunoaștem dacă un șir aparține unui limbaj,

generăm șiruri dintr-un limbaj

sau să *transformăm* astfel de șiruri

Limbaje care nu sunt regulate

Există limbaje foarte simple care nu sunt regulate:

$\{a^n b^n \mid n \geq 0\}$ paranteze echilibrate, ((()))

$\{ww \mid w \in \{a, b\}^*\}$ cuvânt, apoi repetat

$\{ww^R \mid w \in \{a, b\}^*\}$ cuvânt, apoi inversat (palindrom)

Automatele finite au *memorie finită*
număr finit de stări ⇒ nu pot *număra* mai mult de atât

Pentru primul caz, ar trebui să *numărăm* n de a, cu n oricât de mare

În cazul 2 și 3, ar trebui să memorăm cuvinte de lungime arbitrară ca să le comparăm ulterior.

Limbajele de programare trebuie descrise precis

Expresiile regulate nu ajung pentru a descrie limbaje (chiar uzuale).

Din standardul C:

(6.8.4) *selection-statement:*

if **(** *expression* **)** *statement*
if **(** *expression* **)** *statement* **else** *statement*
switch **(** *expression* **)** *statement*

(6.8.5) *iteration-statement:*

while **(** *expression* **)** *statement*
do *statement* **while** **(** *expression* **)** **;**
for **(** $expression_{opt}$ **;** $expression_{opt}$ **;** $expression_{opt}$ **)** *statement*
for **(** *declaration* $expression_{opt}$ **;** $expression_{opt}$ **)** *statement*

Sintaxa limbajelor de programare e descrisă prin *gramatici*.

Ce sunt gramaticile?

Gramaticile sunt un set de reguli care definesc modul în care cuvintele sunt combinate într-o limbă naturală sau într-un limbaj de programare.

Scopul gramaticilor este acela de a defini structura sintactică a limbajelor și de a permite mașinilor să proceseze și să înțeleagă limbajul respectiv.

Există mai multe tipuri de gramatici în limbaje, cum ar fi gramaticile regulate, gramaticile independente de context, gramaticile dependente de context și gramaticile nerestricționate.

Gramaticile sunt utilizate în diverse aplicatii, cum ar fi analiza și procesarea limbajului natural, compilarea și interpretarea limbajelor de programare și verificarea sintaxei în editoare de cod.

Gramatica limbajului natural

Exemplu: propoziții în limba engleză (mult simplificat)

A good student reads books.

S → NP VP	noun phrase + verb phrase
NP → subst	simplu: doar substantiv
NP → det NP	cu parte determinantă (art/adj)
VP → verb	predicat simplu: doar verb
VP → verb NP	verb cu complement

Am descris limbajul prin *reguli de producție* (de *rescriere*)

Simbolurile folosite în regulile de producție sunt:

neterminale: simboluri care apar în stânga → (sunt înlocuite)

terminale: simboluri care apar numai în dreapta →

O gramatică descrie un limbaj

Orice limbaj e descris prin *simbolurile* și *sintaxa* sa:
regulile după care simbolurile pot fi combinate corect.

O *gramatică* descrie cum se obțin șirurile unui limbaj
prin *reguli de producție* (*reguli de rescriere*) pornind
de la un *simbol de start*

O *derivare* a unui șir dintr-o gramatică e o *secvență de aplicări* a *regulilor de producție* care transformă simbolul de start în șirul dat.
indicăm la fiecare pas și simbolul transformat

O derivare ne arată că șirul aparține limbajului definit de gramatică.

S → NP VP → NP verb NP → NP verb noun
→ det NP verb noun → det det NP verb noun
→ det det noun verb noun → a good student reads books

Exemple de limbaje definite prin gramatici

șirurile de paranteze echilibrate:
orice paranteză deschisă (are o pereche închisă)
o paranteză se închide *după* închiderea celor deschise după ea

(1) $S \to \epsilon$ (notație pentru șirul vid)
(2) $S \to (S)S$

Derivare leftmost pt (())(): $S \xrightarrow{2} \underline{(S)S} \xrightarrow{2} (\underline{(S)S})S \xrightarrow{1} (()S)S \xrightarrow{1} (())S \xrightarrow{2} (())\underline{(S)S} \xrightarrow{1} (())()S \xrightarrow{1} (())()$

la fiecare pas am colorat neterminalul transformat, am indicat regula
Folosită 1 sau 2 și am subliniat în ce se transformă.

$\{ww^R \mid w \in \{a, b\}^*\}$ cuvânt+invers (palindrom, lungime pară)
$S \to \epsilon$
$S \to aSa$
$S \to bSb$

Gramatică formală

O gramatică formală G e formată din:

Σ: o mulțime de simboluri *terminale*
(din care se formează șirurile limbajului)

N: o mulțime de simboluri *neterminale*, $N \cap \Sigma = \emptyset$
(folosite doar în descrierea gramaticii, nu apar în limbaj)

P: o mulțime de *reguli de producție*, de forma
$(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$
un neterminal N, eventual într-un context (șir în stânga/dreapta) e rescris cu un șir de terminale și neterminale

$S \in N$: un *simbol de start*

Limbajul definit de G e format din toate șirurile de *terminale* care se pot obține din S printr-o derivare (aplicând oricâte reguli)

Gramatici recursive

O regulă de producție este *recursivă* dacă partea ei stângă (neterminalul ce va fi rescris + contextul său) apare și în partea ei dreaptă. Obs: o regulă recursivă se poate refolosi de oricâte ori.

Exemplu:

$S \rightarrow aSa$

$bA \rightarrow bbA$

O regulă de producție $A \rightarrow \beta$ este *indirect recursivă* dacă A poate fi derivat într-o formă ce îl conține pe A.

Exemplu:

$S \rightarrow aBa$

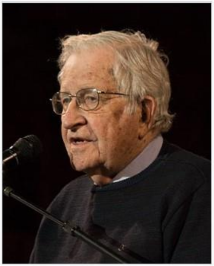
$B \rightarrow bSb$

O gramatică este *recursivă* dacă și numai dacă aceasta conține *cel puțin o regulă de producție recursivă sau indirect recursivă.*

Ierarhia Chomsky [după Noam Chomsky, 1956]

Ierarhia Chomsky este un set de clase formale de gramatică care generează limbaje formale .

Ierarhia acestor gramatici, numită și gramatica structurii frazelor , a fost descrisă de Noam Chomsky în 1956.



Noam Chomsky în 2017

Avram Noam Chomsky (Philadelphia , 7 decembrie 1928) este lingvist , academic , anarhist , om de știință cognitiv , teoretician al comunicării , activist politic și eseist SUA.

Profesor emerit de lingvistică la Institutul de Tehnologie din Massachusetts , este recunoscut ca fiind fondatorul gramaticii generativ-transformative , adesea menționat drept cea mai importantă contribuție la lingvistica teoretică a secolului al XX-lea .

Ierarhia Chomsky [după Noam Chomsky, 1956]

Notăm: neterminale A, B; terminale: a, b; șiruri arbitrare: α, β, γ

3) gramatici *regulate*: generează *limbaje regulate*

reguli de forma:

$A \to a$, $A \to \varepsilon$, $A \to aB$ (regulate la dreapta), SAU

$A \to a$, $A \to \varepsilon$, $A \to Ba$ (regulate la stânga), NU le combinăm

Limbajele regulate sunt recunoscute de automate finite

2) gramatici *independente de context*

reguli: $A \to \gamma$ stânga: neterminal; dreapta: șir arbitrar

Limbajele independente de context sunt acceptate de automatele cu stivă

1) gramatici *dependente de context*

reguli: $\alpha A\beta \to \alpha\gamma\beta$ A e rescris dacă apare între α și β $\gamma \neq \epsilon$ (nevid),

sau $S \to \epsilon$ doar dacă S nu apare în dreapta

Limbajele dependente de context pot fi recunoscute de o mașină Turing nedeterministă

0) gramatici *nerestricționate* (orice reguli de rescriere)

limbaje *recursiv enumerabile* (recunoscute de o mașină Turing)

Forma Backus-Naur (BNF)

dupa John Backus (dezvoltatorul limbajului FORTRAN)
și Peter Naur (ALGOL 60)   (fiecare: premiul *Turing*)

Notație frecvent folosită pentru gramatici independente de context

folosește ::= pentru definiție și | pentru alternativă

Neterminal ::= rescriere1 | rescriere2 | ... | rescriereN

uneori folosite cu extensii:

[*element-opțional*]
simbol * (steaua Kleene) pentru repetiție
simbol + (plus) pentru repetiție cel puțin odată
paranteze pentru gruparea elementelor

Exemple: instrucțiuni în C (simplificat)

Stmt ::= ExpStmt | IfStmt | WhileStmt | Block

ExpStmt ::= expr ;

IfStmt ::= ***if*** *(expr) Stmt* ***else*** *Stmt* | **if** (expr) Stmt

WhileStmt ::= ***while*** *(expr) Stmt Block*

::= { Stmt }*

Exemple: instrucțiuni în PYTHON

Program ::= (Statement)*

Statement ::= Assignment | FunctionDefinition | FunctionCall | IfStatement | While Statement | For Statement | Return Statement | Break Statement | Continue Statement | Pass Statement | Import Statement | Global Statement | Nonlocal Statement | ExpressionStatement | DelStatement

Assignment ::= Target "=" Expression

FunctionDefinition ::= "def" FunctionName "(" [ParameterList] ")" ["->" Type] ":" Suite

FunctionCall ::= FunctionName "(" [ArgumentList] ")"

Sursa: Documentația oficială a limbajului Python https://docs.python.org/3/

Exemple: instrucțiuni în PYTHON

IfStatement ::= "if" Expression ":" Suite ("elif" Expression ":" Suite)* ["else" ":" Suite]

Return Statement ::= "return" [Expression]

Break Statement ::= "break"

Continue Statement ::= "continue"

ExpressionList ::= (Expression ",")* Expression

Expression ::= Disjunction

Disjunction ::= Conjunction ("or" Conjunction)*

Conjunction ::= Comparison ("and" Comparison)*

Sursa: Documentația oficială a limbajului Python https://docs.python.org/3/

Exemple: instrucțiuni în PYTHON

Comparison ::= LambdaExpr | OrExpr (("==" | "!=" | "<" | "<=" | ">" | ">=" | "is" | "is not" | "in" | "not in") OrExpr)*

OrExpr ::= XorExpr ("|" XorExpr)*

XorExpr ::= AndExpr ("^" AndExpr)*

AndExpr ::= ShiftExpr ("&" ShiftExpr)*

ShiftExpr ::= ArithmeticExpr (("<<" | ">>") ArithmeticExpr)*

ArithmeticExpr ::= Term (("+" | "-") Term)*

Sursa: Documentația oficială a limbajului Python https://docs.python.org/3/

Expresii aritmetice

v1) E ::= num | E + E | E − E | E ∗ E | E / E | (E)

și aici avem *ambiguitate*:
nu e precizată precedența operatorilor

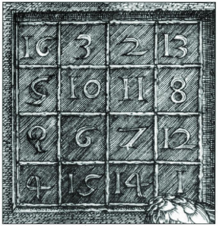
v2) Rescriem pe 3 nivele de *precedență*:

E ::= T | E + T | E − T

T ::= F | T ∗ F | T / F

F ::= num | (E)

Exemplu: $2 * (5 - 3)$



Limbaje și Automate finite deterministe
Automate finite nedeterministe (NFA)
Expresii regulate
Gramatici
Mașini Turing

Ce putem calcula?

DFA/NFA recunosc doar limbaje regulate

Într-un automat (DFA sau NFA) comportamentul e determinat complet de *stare* și *intrare*

Automatul "știe" doar starea în care se află:
  are *memorie finită*

$L=\{a^n b^n | n \in \mathbb{N}\}$ nu e un limbaj regulat
ar trebui să *numărăm* câți *a* apar, verificăm să fie la fel de mulți *b*.
fără nicio limitare

⇒ pt. a recunoaște limbajul
  avem nevoie de o structură cu *memorie nelimitată*

Conceptual, un calculator nu are nici el limită de memorie (desi în realitate aceasta este, desigur, finită).

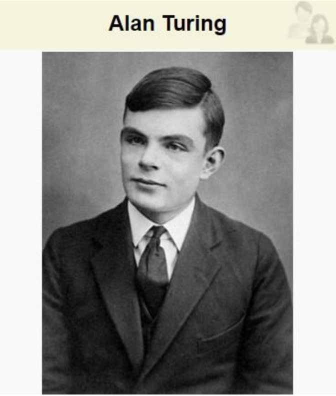
Mașina Turing = automat cu stări finite
+ memorie nelimitată

Mașina Turing

O mașină Turing este un model teoretic al unui calculator care poate efectua orice operație care poate fi formalizată într-un set de instrucțiuni.

Masinile Turing au fost dezvoltate de matematicianul Alan Turing în anii 1940 și au devenit un model fundamental pentru modul în care funcționează calculatoarele moderne.

Alan Turing



Alan Turing a fost un informatician, matematician, logician, criptanalist, filosof și maratonist britanic.

A fost o personalitate deosebit de influentă în dezvoltarea informaticii, aducând o formalizare a conceptelor de „algoritm” și „computație” cu mașina Turing, care poate fi considerată un model de calculator generic.

Turing este considerat a fi părintele informaticii și inteligenței artificiale teoretice.

Sursa: https://ro.wikipedia.org/wiki/Alan_Turing#

Alan Turing

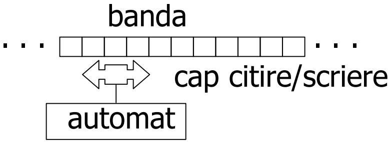
În timpul celui de al Doilea Război Mondial, Turing a lucrat pentru centrul de criptanaliză al Regatului Unit. A pus la punct mai multe tehnici de spargere a cifrurilor germane (printre care și ale mașinii Enigma). Rolul-cheie jucat de Turing în decodificarea mesajelor interceptate le-a permis Aliaților să-i învingă pe naziști în mai multe lupte importante; se estimează că activitatea echipei de la Bletchley Park a scurtat războiul în Europa cu doi până la patru ani.

În 1952, Turing a fost judecat pentru homosexualitate, pe când acest comportament sexual era încă incrimnat în Regatul Unit. A acceptat un tratament cu injecții de estrogen (castrare chimică(en)) drept alternativă la închisoare.

Turing a murit în 1954, cu 16 zile înainte de a împlini 42 de ani, în urma otrăvirii cu cianură.

Sursa: https://ro.wikipedia.org/wiki/Alan_Turing#

Mașina Turing



Mașina Turing e compusă din:

un *automat cu stări finite*

o *bandă* cu un număr infinit de *celule*

fiecare celulă a benzii conține un *simbol*

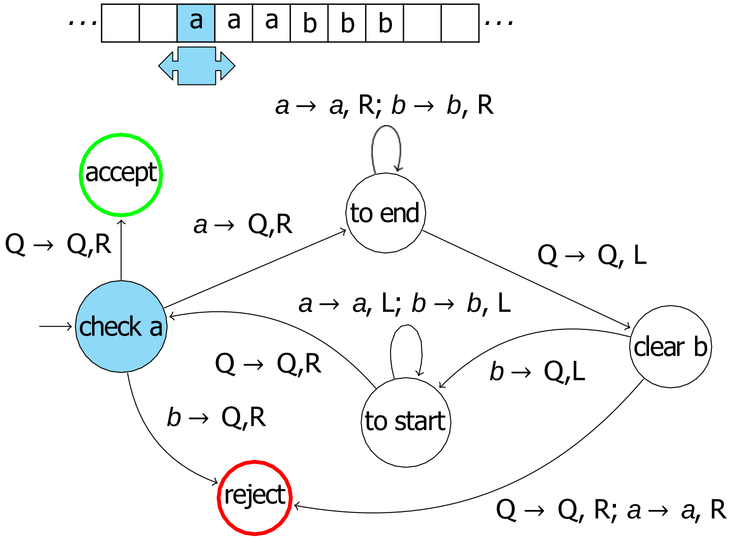
(banda poate fi infinită la unul/ambele capete, e echivalent)

un *cap* de citire/scriere al simbolurilor de pe bandă

(*controlat de automat*)

Automatul și conținutul benzii determină *împreună* comportamentul mașinii Turing.

Exemplu: mașina Turing pt. L=$\{a^n b^n | n \in \mathbb{N}\}$



Mașina Turing

Automatul și conținutul benzii determină comportamentul.

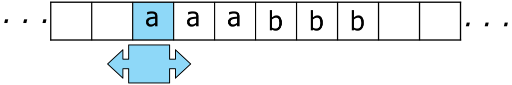
În funcție de

- *starea curentă* a automatului
- *simbolul citit* de pe bandă

Mașina efectuează următoarele acțiuni:

- trece în *starea următoare* (a automatului)
- *scrie* un (nou) *simbol* sub capul de citire/scriere
- *mută capul* de citire/scriere la stânga sau la dreapta

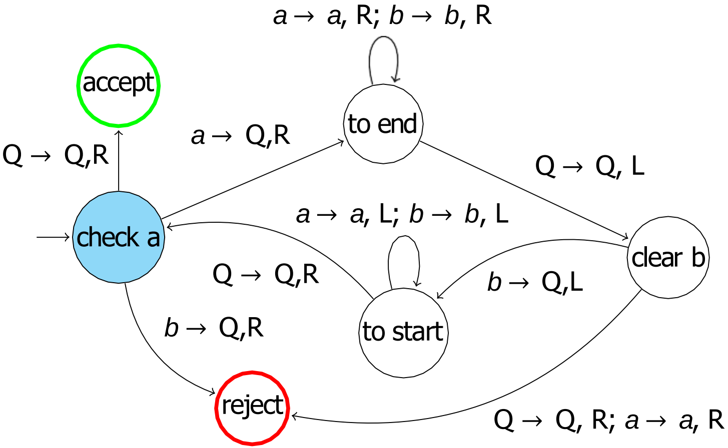
Mașina Turing



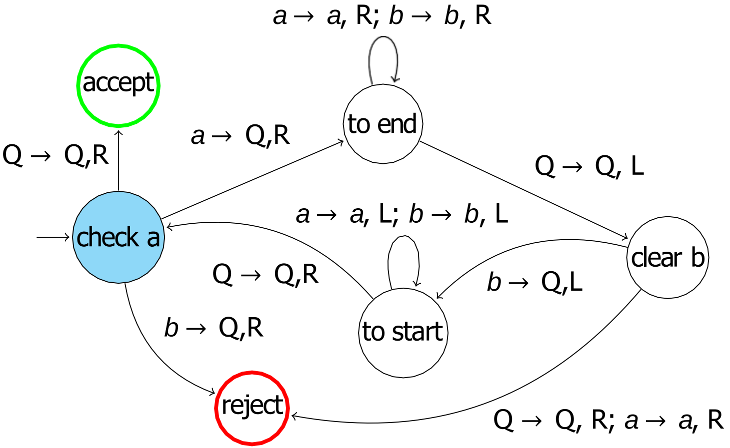
Inițial, banda conține un șir finit de *simboluri de intrare*,
capul de citire/scriere e pe *primul* simbol (cel mai din stânga);
restul celulelor conțin un *simbol special* (Q = *vid* sau *blank*)

La fiecare pas, este citit/scris *doar* simbolul aflat *imediat sub* capul de citire/scriere!

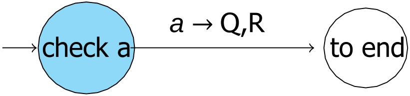
Ca orice automat, mașina Turing începe execuția din *starea inițială*.



Într-o mașină Turing, tranzițiile dintre stări au forma de mai jos
simbol citit → *simbol scris*, *direcție* mutare cap (L/R)



Într-o mașină Turing, tranzițiile dintre stări au forma de mai jos
simbol citit → *simbol scris*, *direcție* mutare cap (L/R)

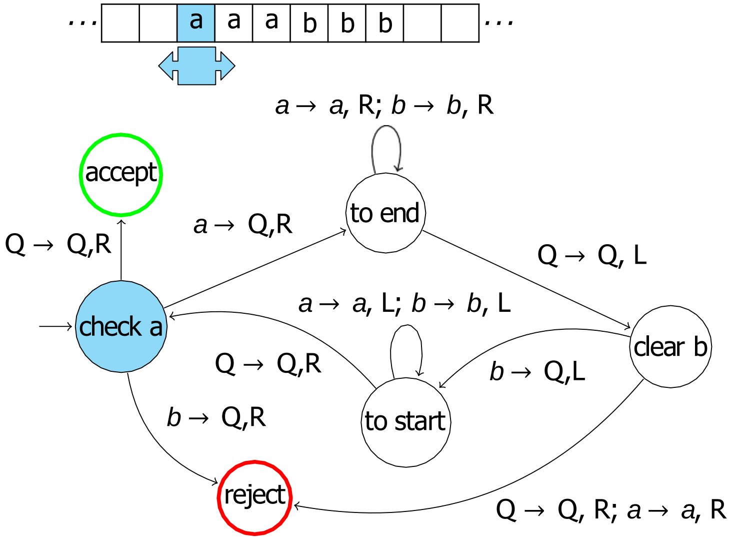


Tranziția *a* → Q, R: *dacă* se citește simbolul *a* de pe bandă, atunci se scrie Q și se mută capul de citire/scriere cu o celulă la *dreapta*

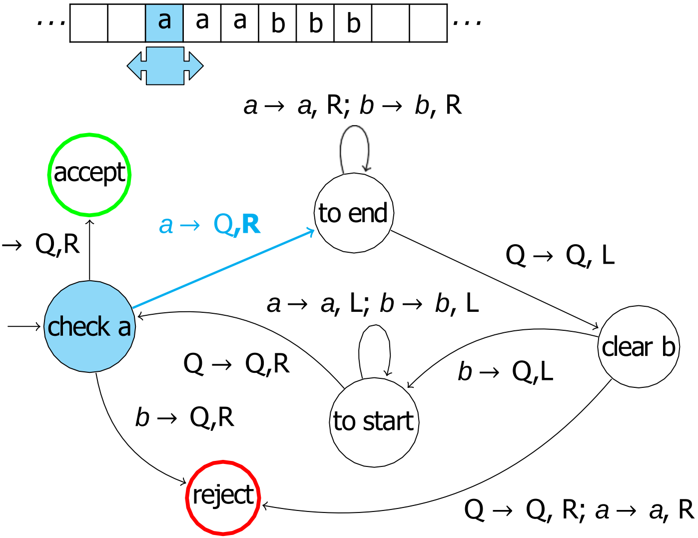
R: mută capul de citire/scriere cu o celulă la dreapta
L: mută capul de citire/scriere cu o celulă la stânga

Q: simbolul vid

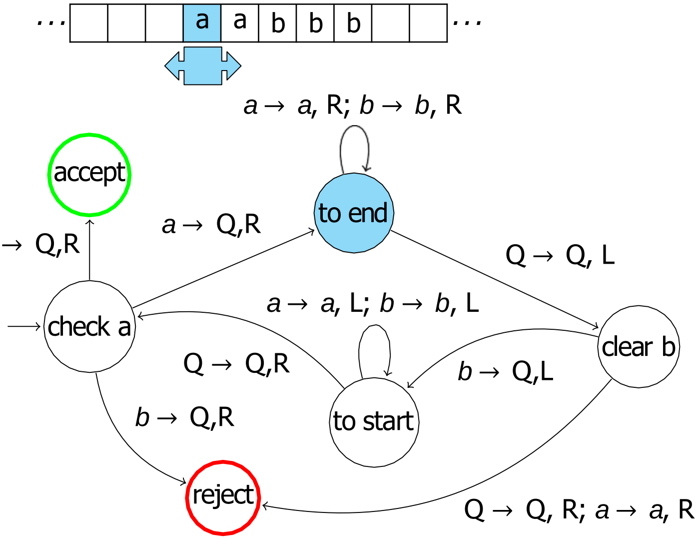
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



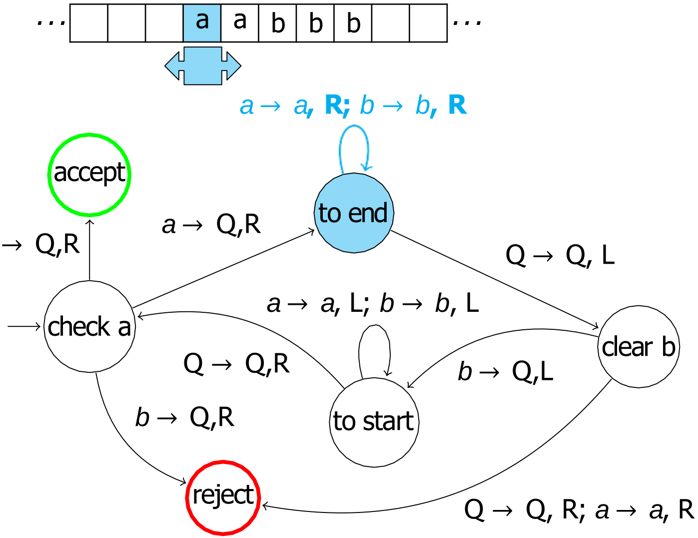
Exemplu: *aaabbb* ∈ L={$a^n b^n | n \in \mathbb{N}$}



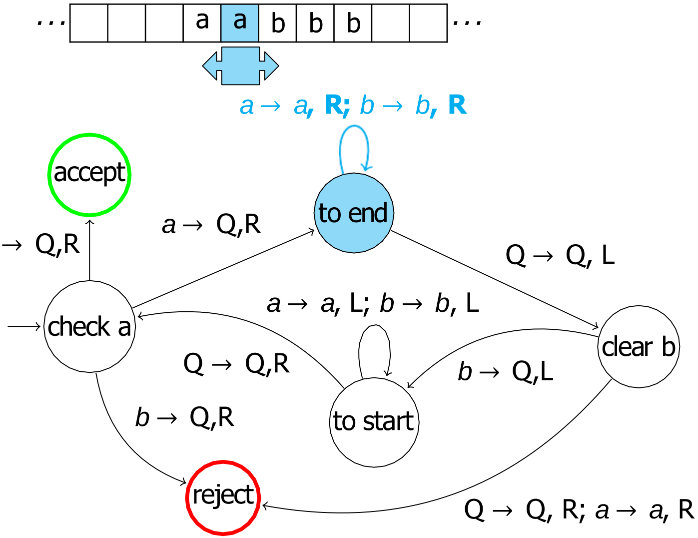
Exemplu: *aaabbb* ∈ L={$a^n b^n$|$n \in \mathbb{N}$}



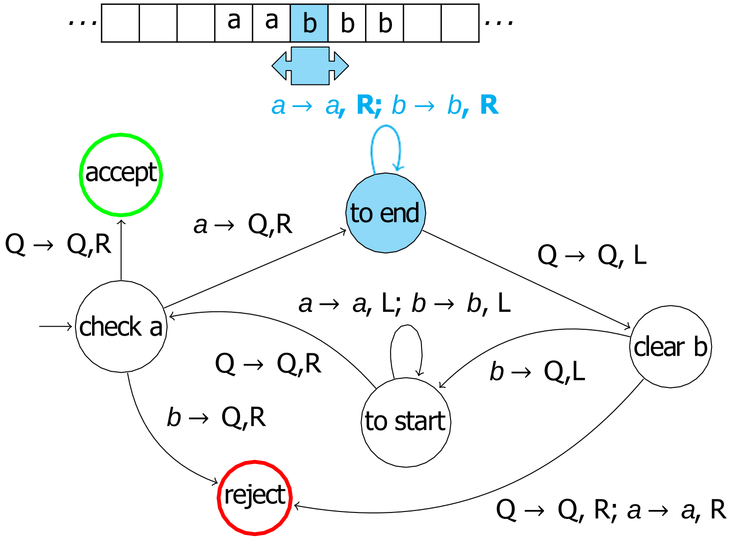
Exemplu: *aaabbb* ∈ L={$a^n b^n | n \in \mathbb{N}$}



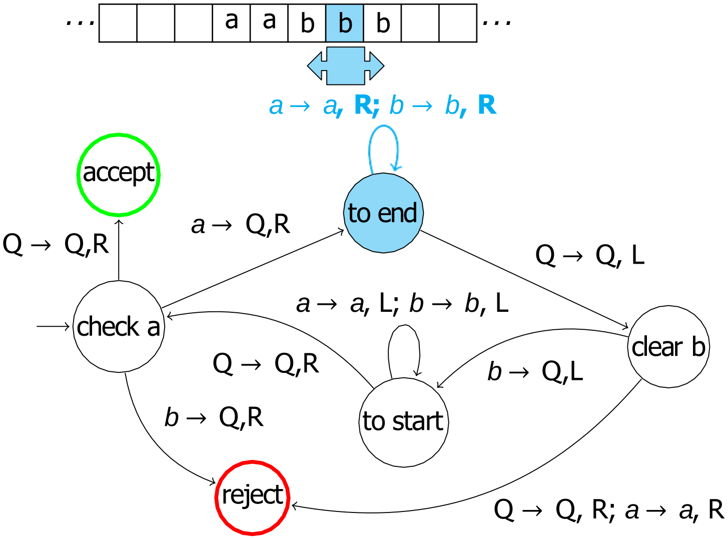
Exemplu: *aaabbb* ∈ L={$a^n b^n$|n ∈ ℕ}



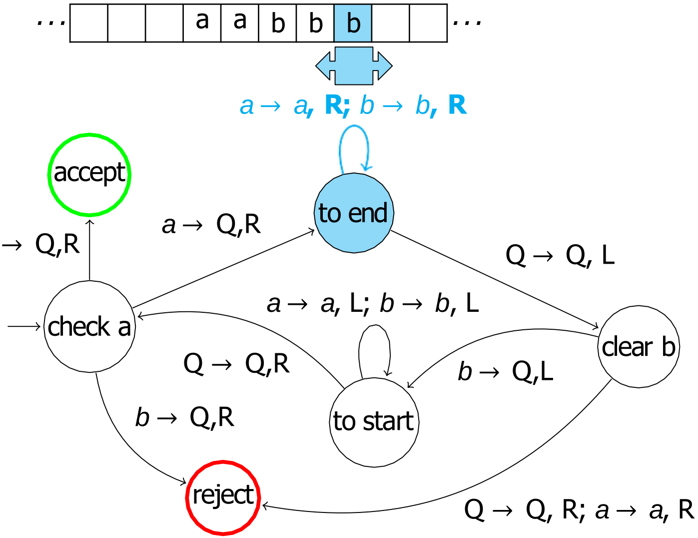
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



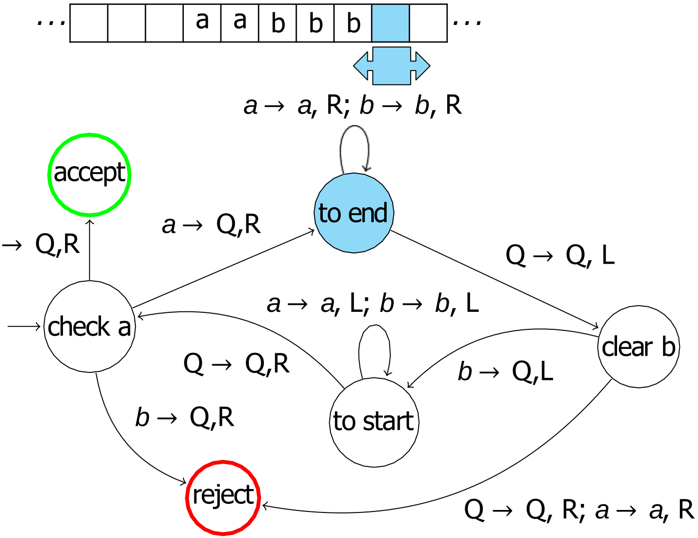
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



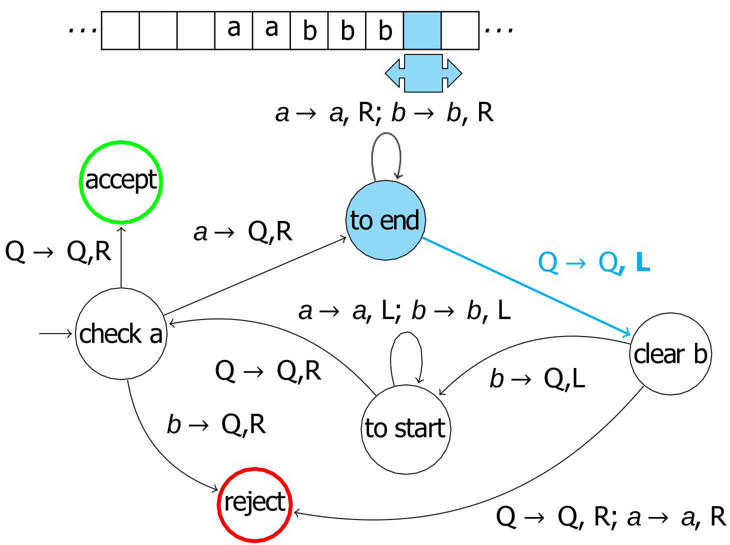
Exemplu: *aaabbb* ∈ L={$a^n b^n$|n ∈ ℕ}



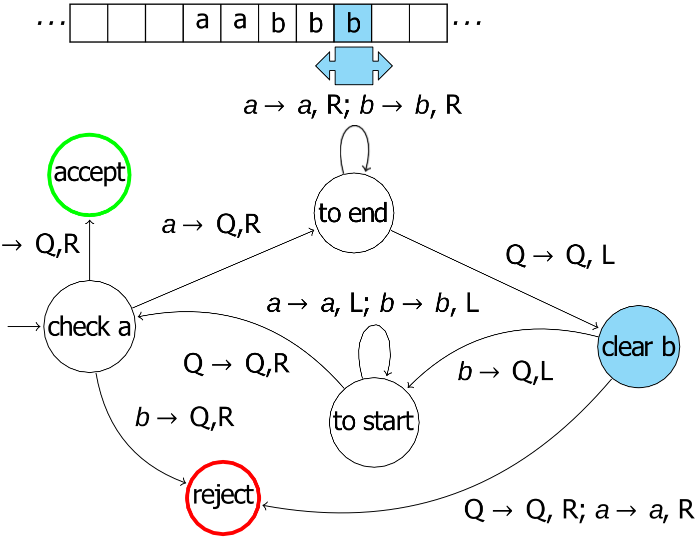
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



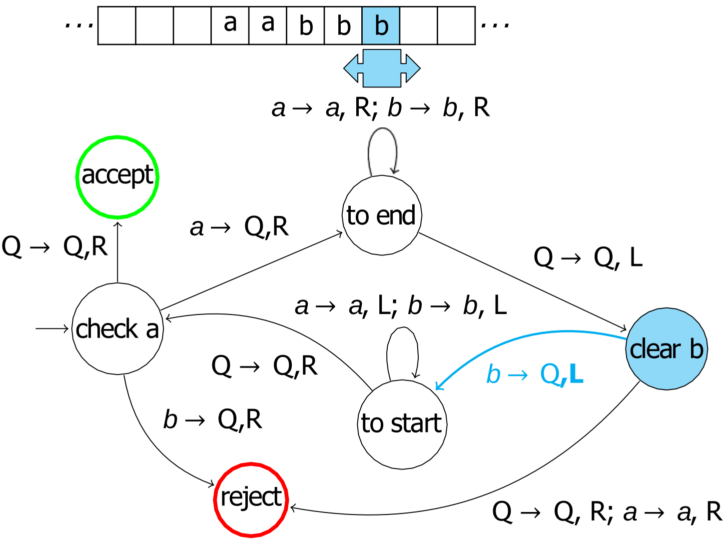
Exemplu: *aaabbb* ∈ L={$a^n b^n$|*n* ∈ ℕ}



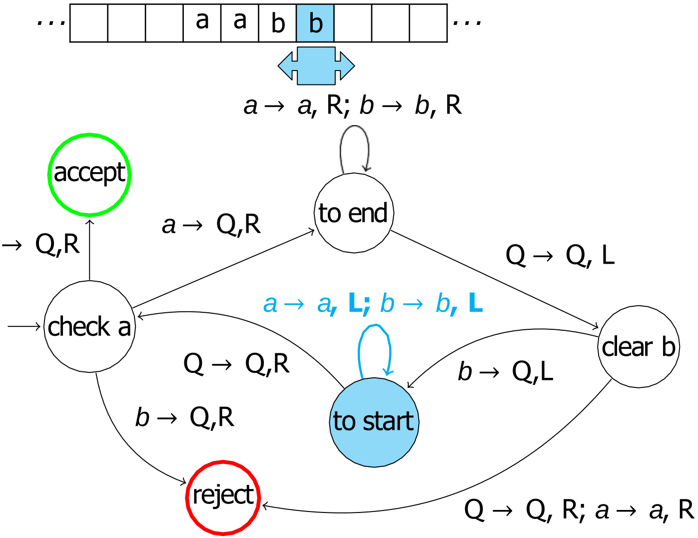
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



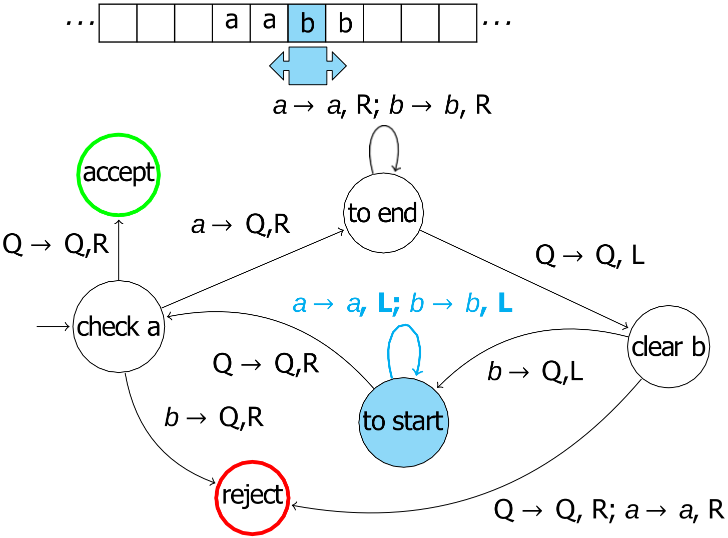
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



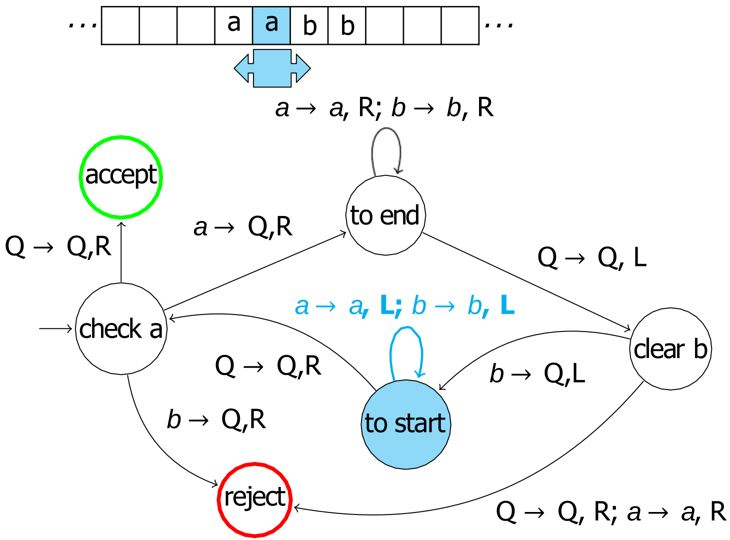
Exemplu: *aaabbb* ∈ L={$a^n b^n$|n ∈ ℕ}



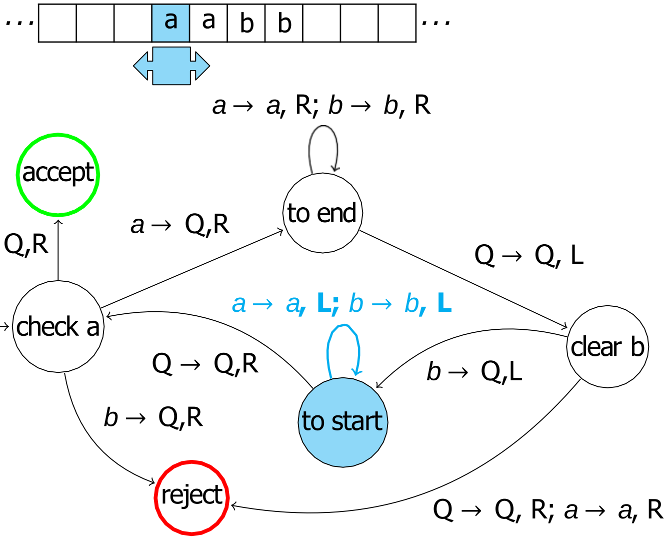
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



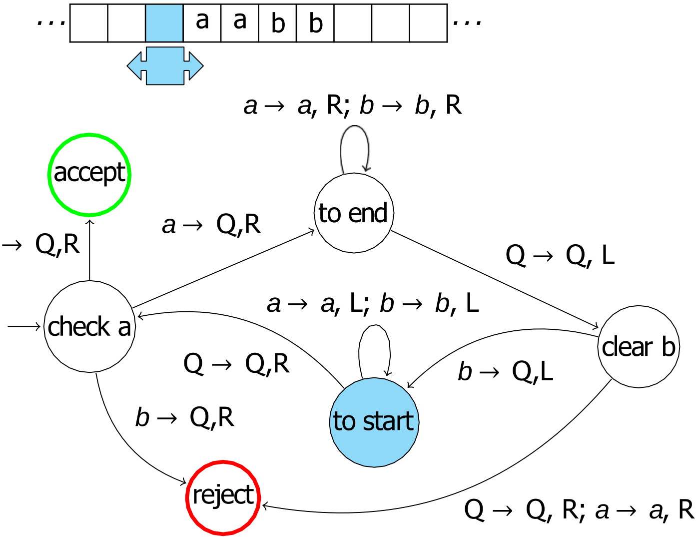
Exemplu: *aaabbb* ∈ L={$a^n b^n$|n ∈ ℕ}



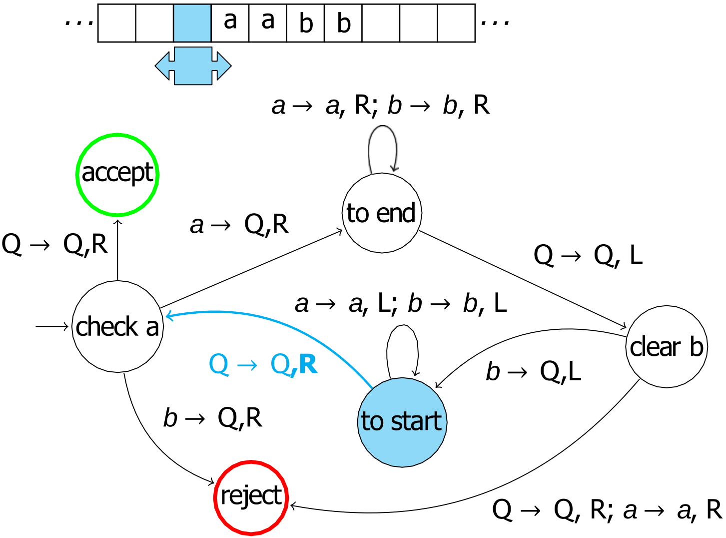
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



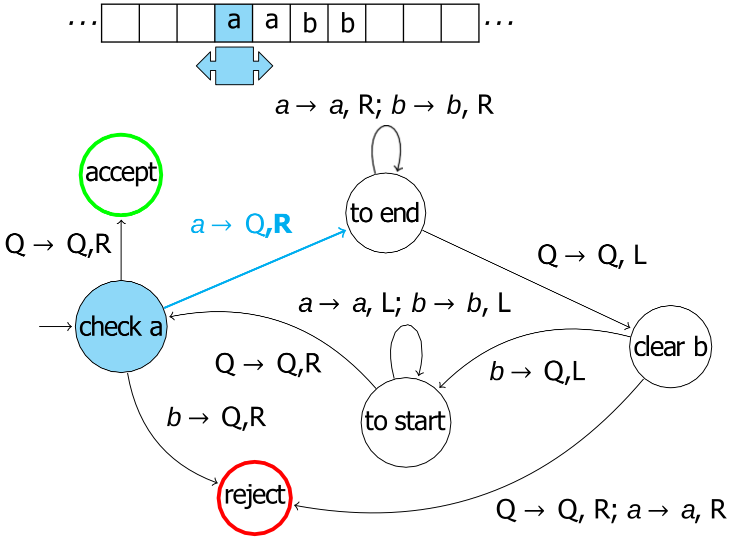
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



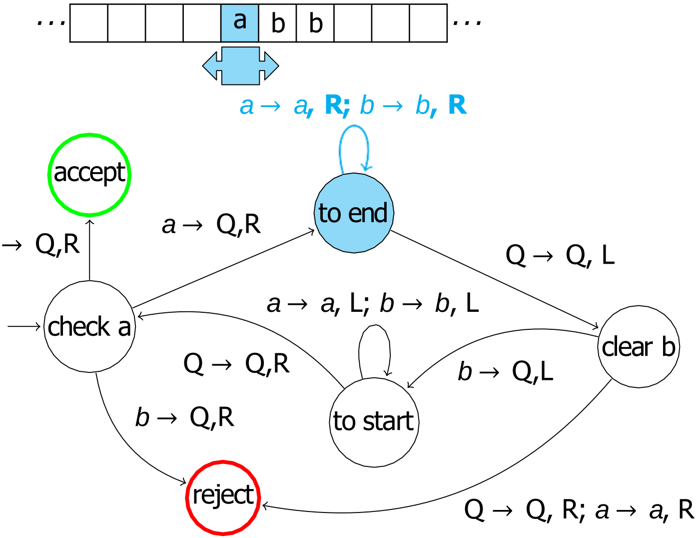
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



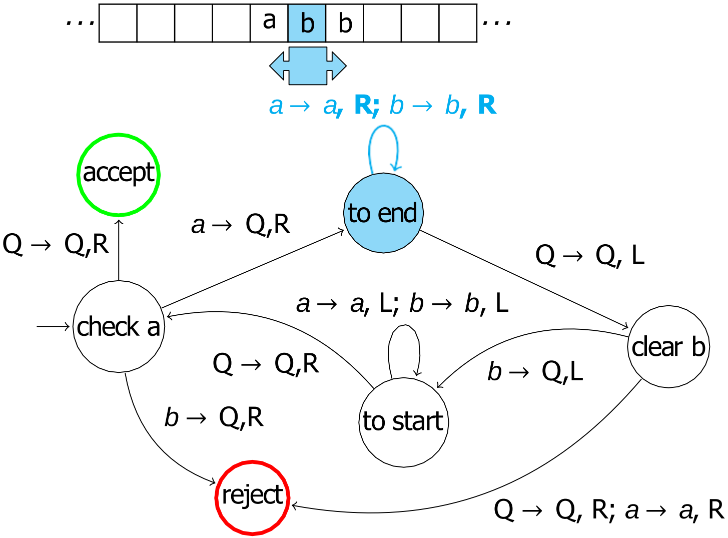
Exemplu: *aaabbb* ∈ L={$a^n b^n$|n ∈ ℕ}



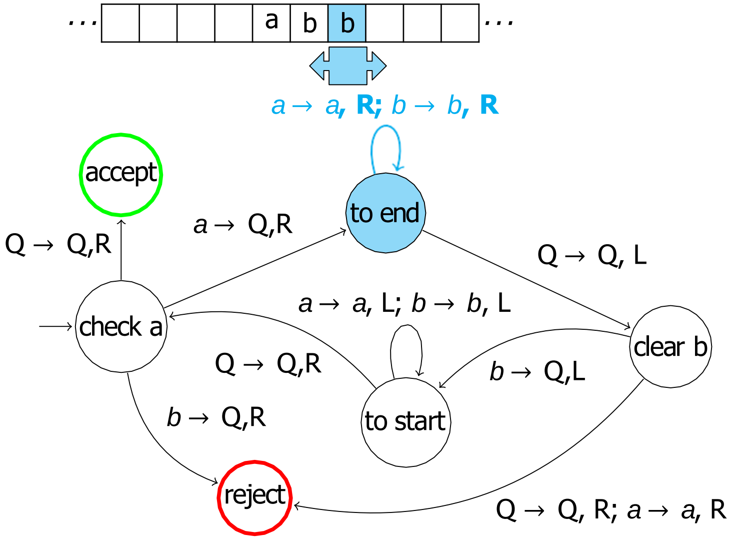
Exemplu: *aaabbb* ∈ L={$a^n b^n | n \in \mathbb{N}$}



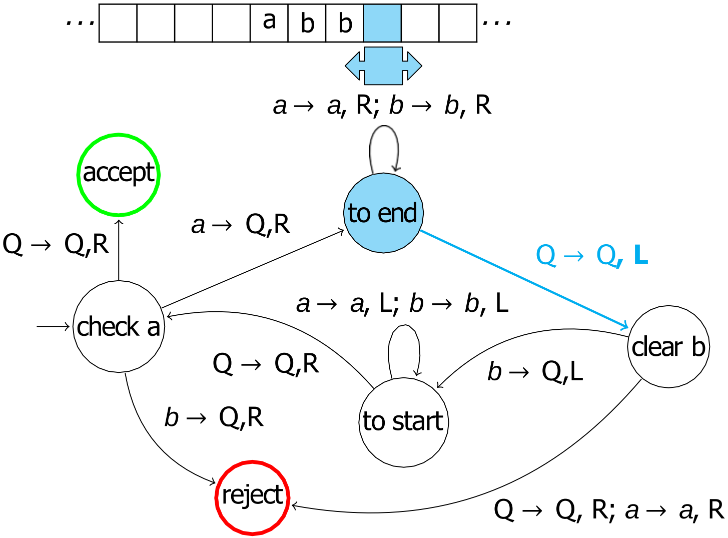
Exemplu: *aaabbb* ∈ L={$a^n b^n$|$n \in \mathbb{N}$}



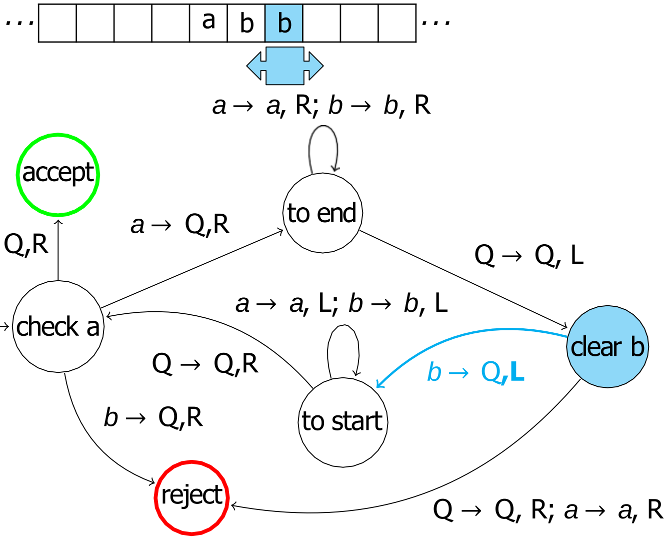
Exemplu: *aaabbb* ∈ L={$a^n b^n$ | $n \in \mathbb{N}$}



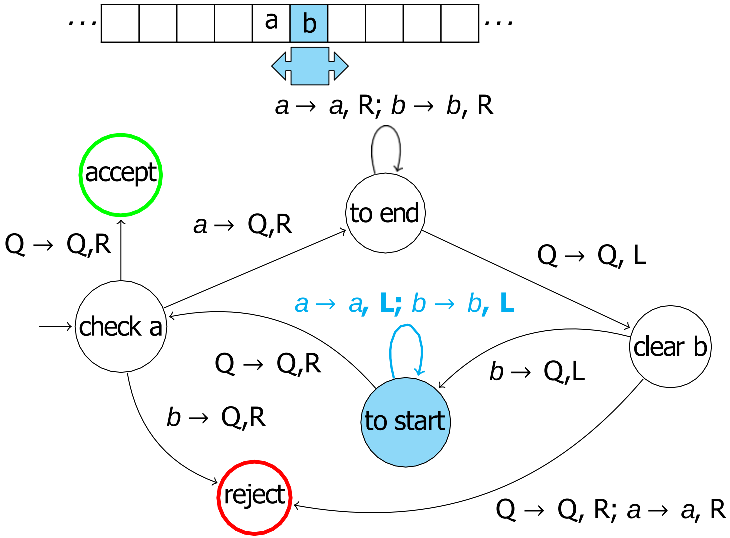
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



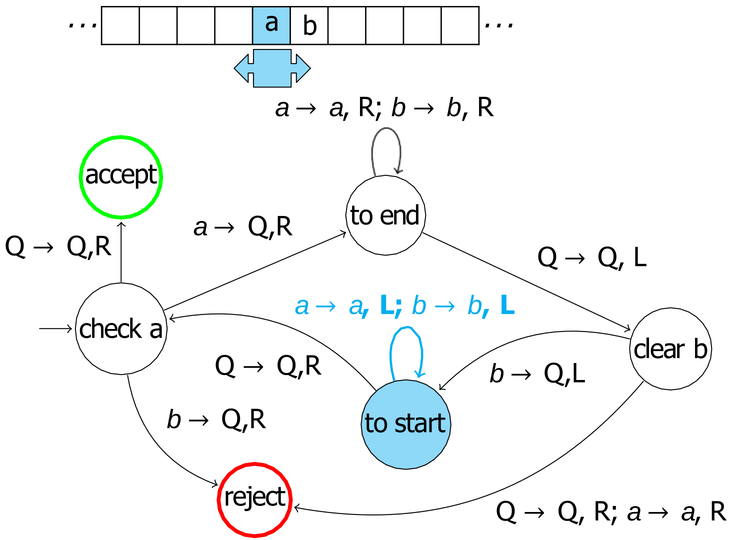
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



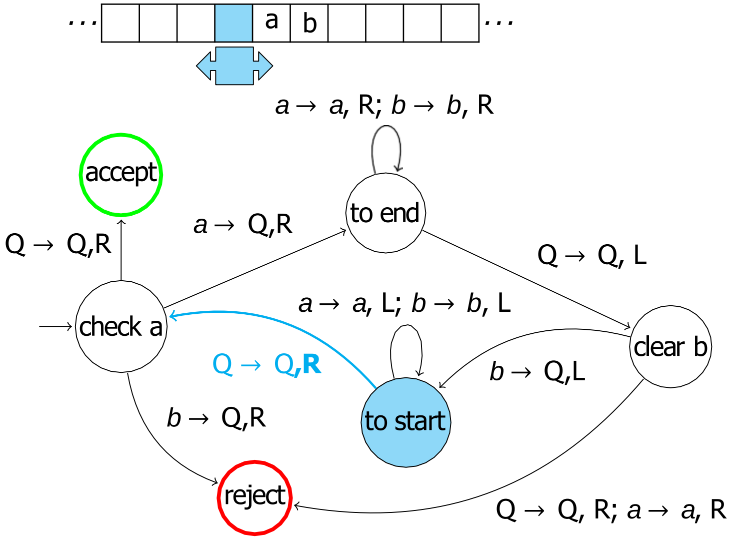
Exemplu: *aaabbb* ∈ L={$a^n b^n | n \in \mathbb{N}$}



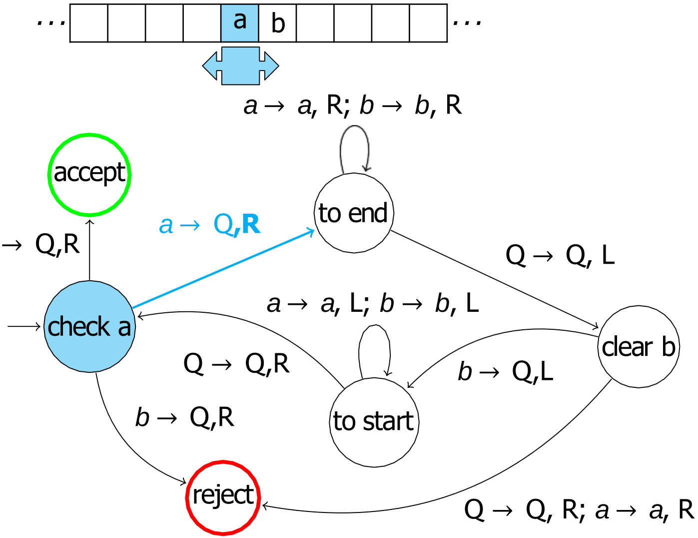
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



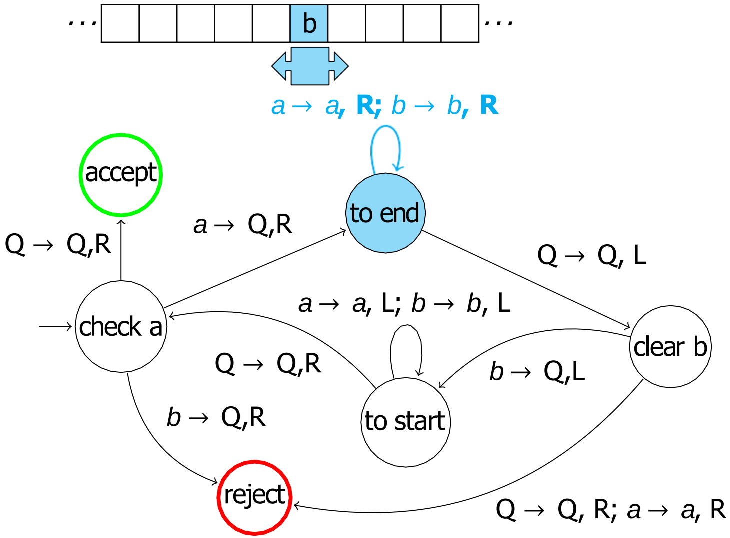
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



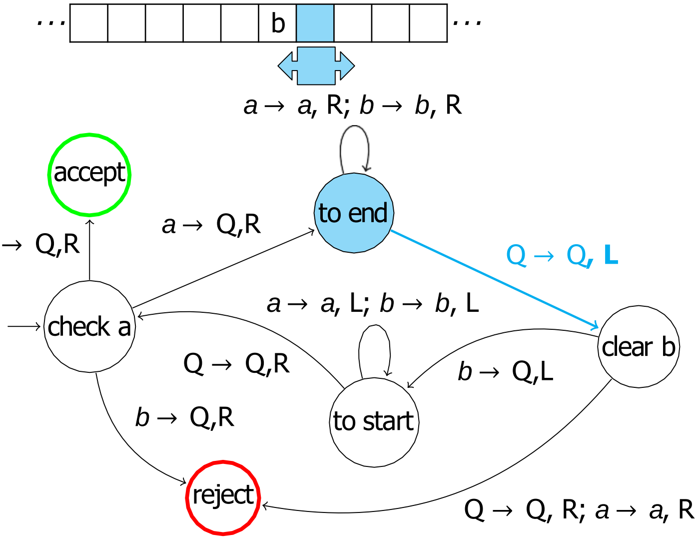
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



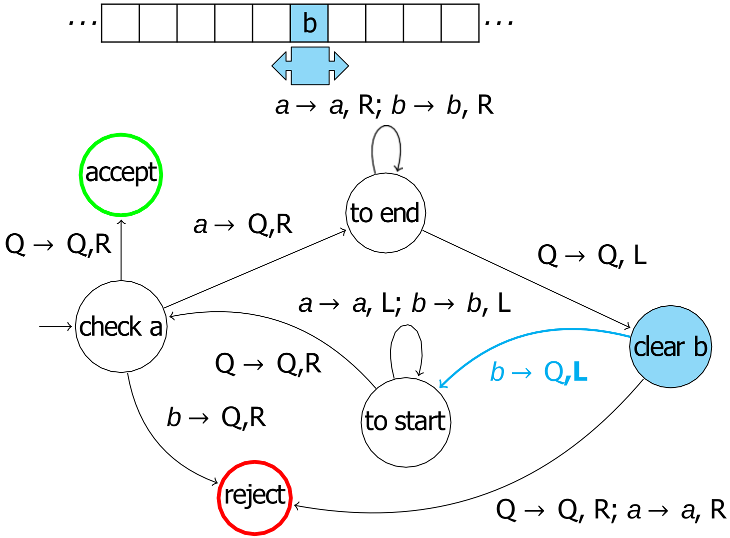
Exemplu: *aaabbb* ∈ L=$\{a^n b^n | n \in \mathbb{N}\}$



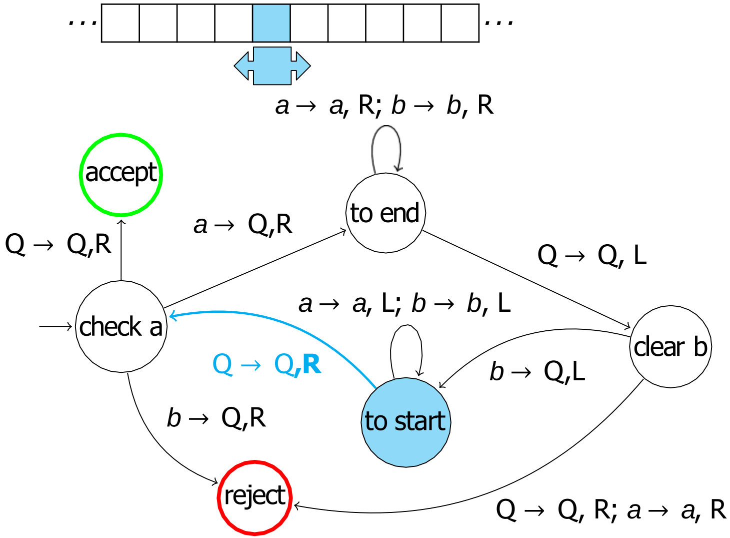
Exemplu: *aaabbb* ∈ L={$a^n b^n$|n ∈ ℕ}



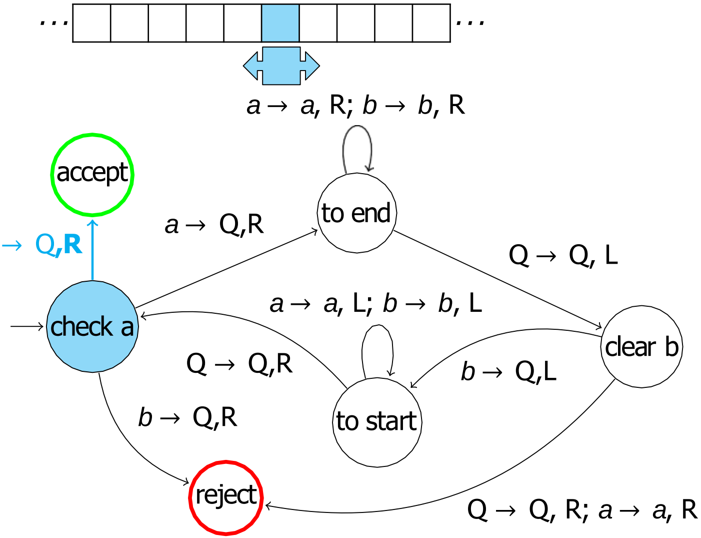
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



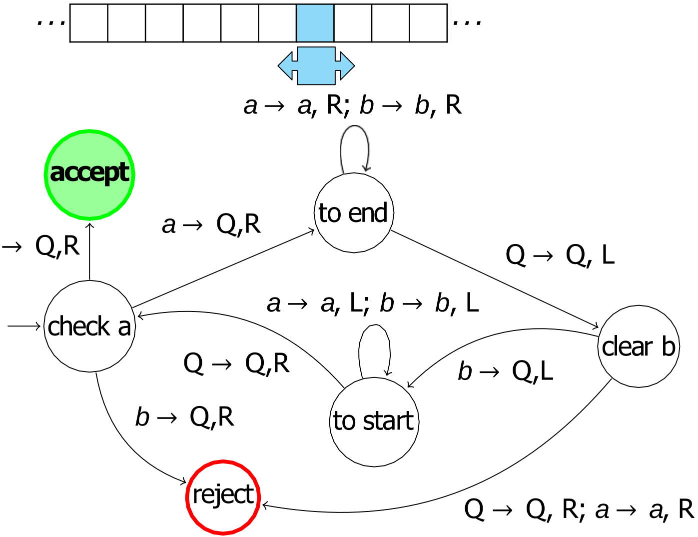
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



Exemplu: *aaabbb* ∈ L={$a^n b^n$|n ∈ ℕ}



Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



Important:

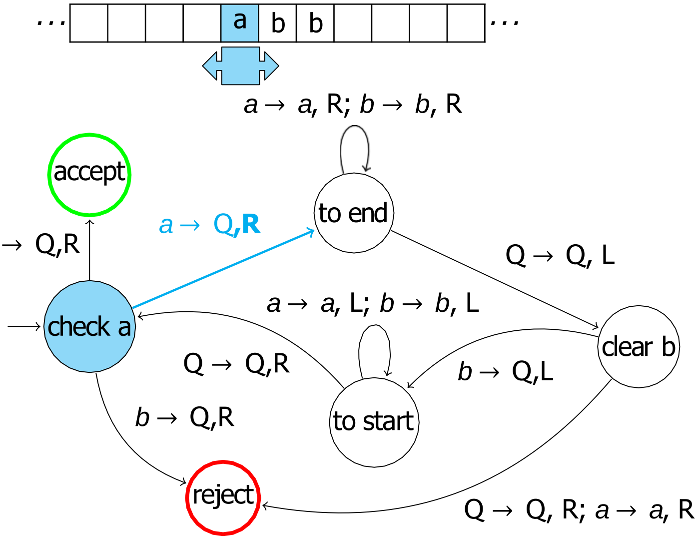
Spre deosebire de un automat DFA/NFA, o mașină Turing **nu** *se oprește* la terminarea șirului de intrare!

Execuția continuă până când se ajunge într-una din *stările finale*:

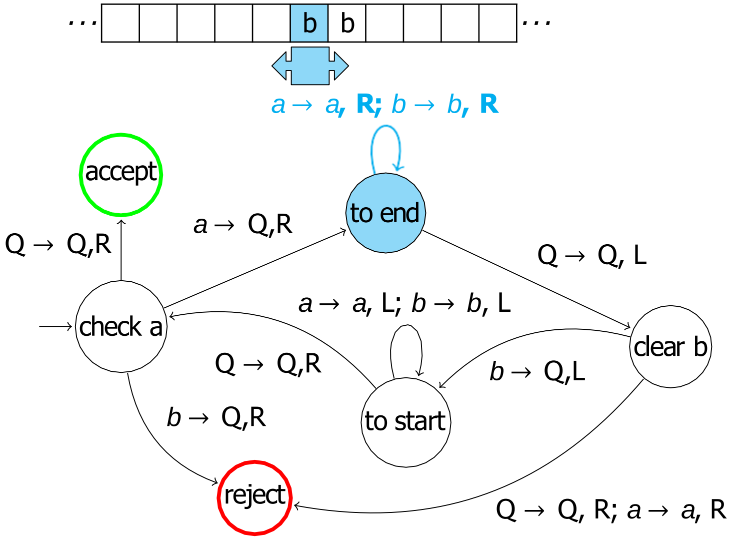
de **acceptare**: șirul este acceptat, face parte din limbaj

de **rejectare**: șirul este respins, nu face parte din limbaj

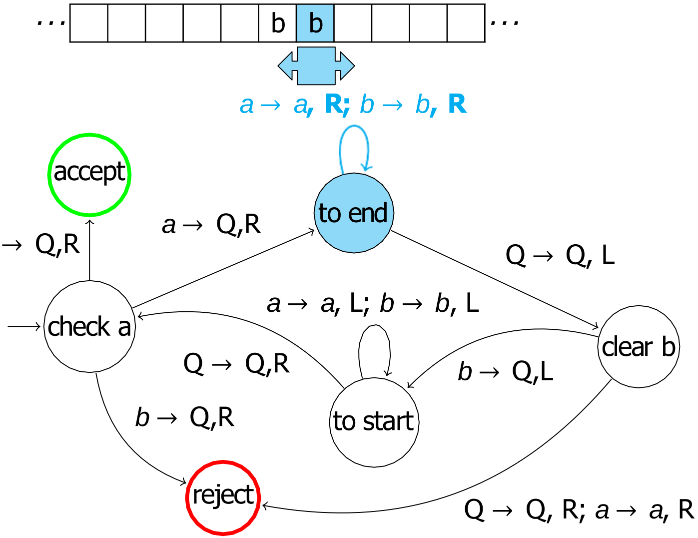
Exemplu: $abb \notin L = \{a^n b^n | n \in \mathbb{N}\}$



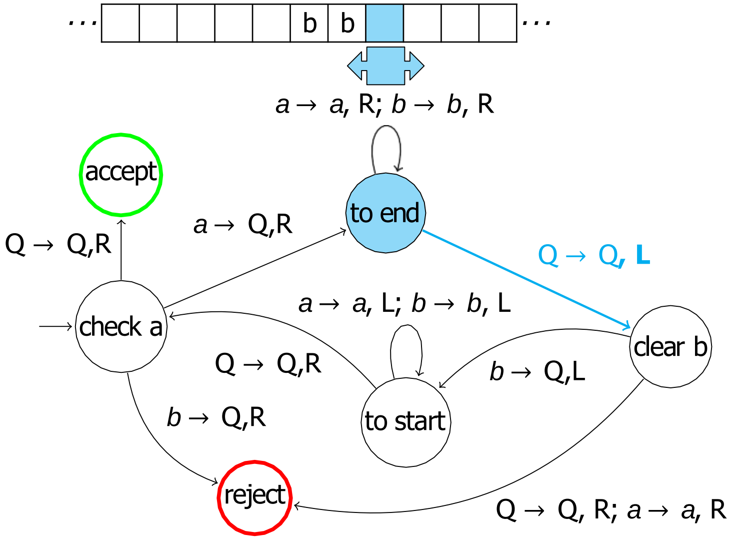
Exemplu: $abb \notin L = \{a^n b^n \mid n \in \mathbb{N}\}$



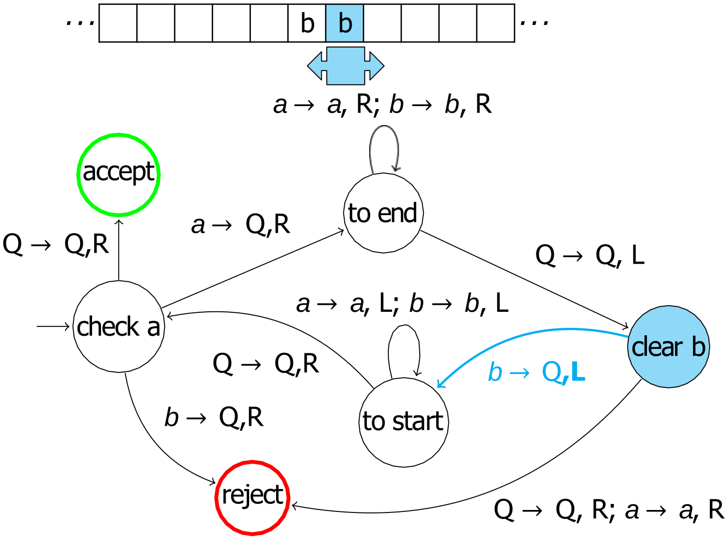
Exemplu: $abb \notin L = \{a^n b^n | n \in \mathbb{N}\}$



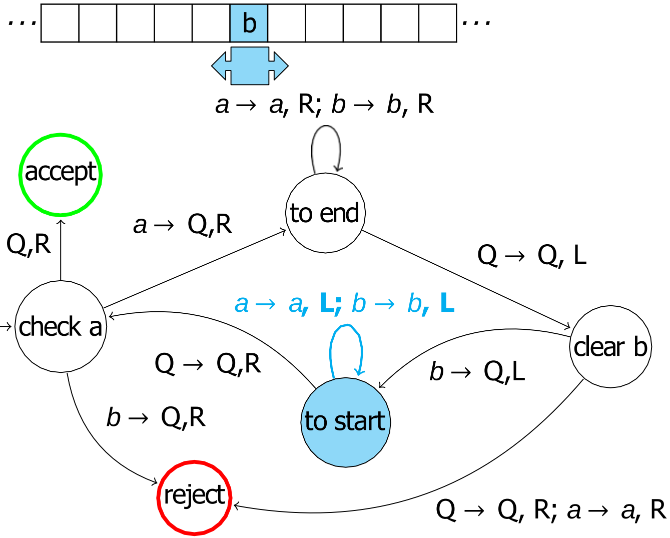
Exemplu: $abb \notin L=\{a^n b^n | n \in \mathbb{N}\}$



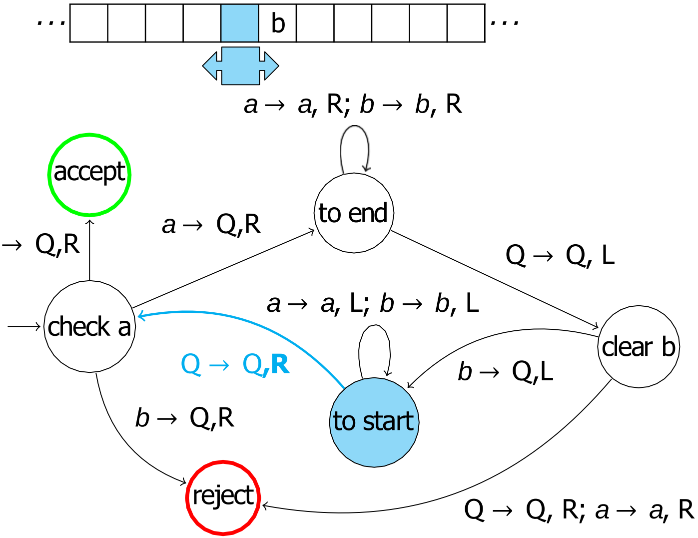
Exemplu: $abb \notin L = \{a^n b^n | n \in \mathbb{N}\}$



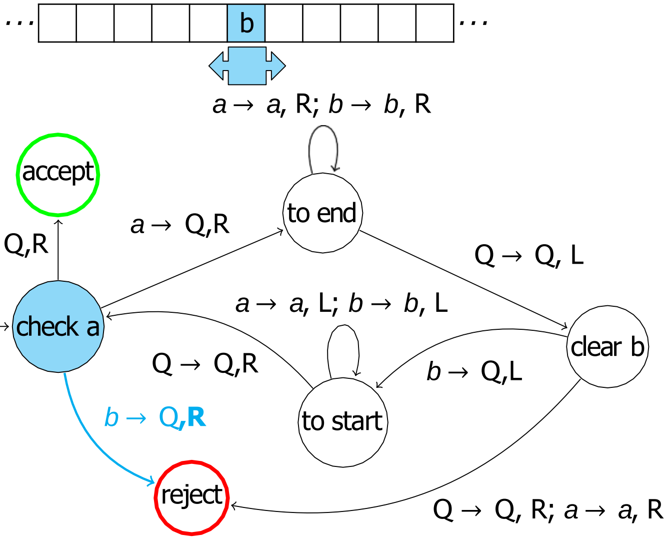
Exemplu: $abb \notin L = \{a^n b^n | n \in \mathbb{N}\}$



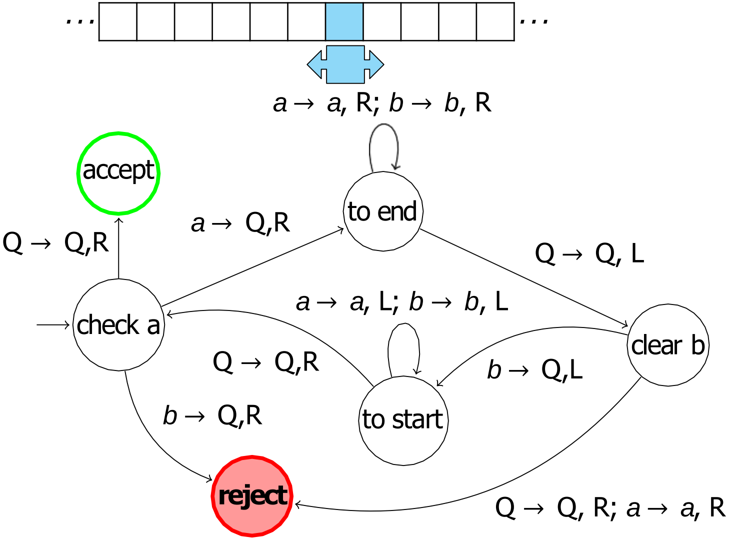
Exemplu: $abb \notin L = \{a^n b^n | n \in \mathbb{N}\}$



Exemplu: $abb \notin L = \{a^n b^n \mid n \in \mathbb{N}\}$

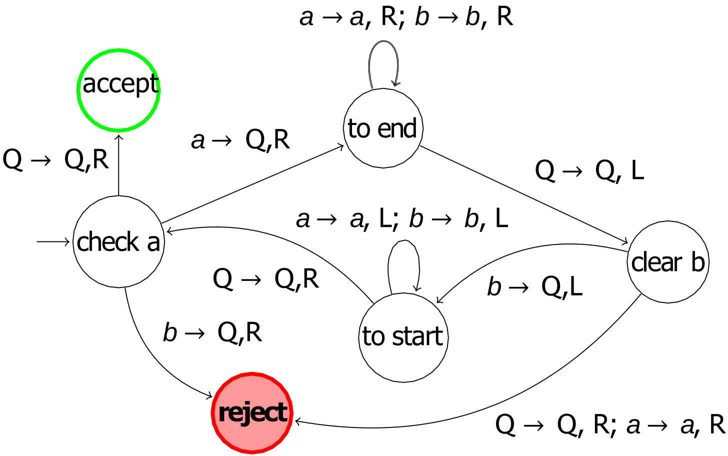


Exemplu: $abb \notin L = \{a^n b^n \mid n \in \mathbb{N}\}$



Obs.: mașinile Turing sunt deterministe!

Pentru fiecare combinație de stare non-finală q și simbol de bandă γ, există o *unică* tranziție $\delta(q, \gamma)$
(tranzițiile lipsă, dacă există, duc implicit în *starea de rejectare*)



Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)

Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)

Σ: mulțimea finită a *simbolurilor de intrare* (din șirul inițial)

Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)

Σ: mulțimea finită a *simbolurilor de intrare* (din șirul inițial)

Γ: mulțimea simbolurilor de pe bandă (care pot fi scrise);
important: $\Sigma \subset \Gamma$ (Γ are cel puțin un simbol în plușQ)

Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)

Σ: mulțimea finită a *simbolurilor de intrare* (din șirul inițial)

Γ: mulțimea simbolurilor de pe bandă (care pot fi scrise);
important: $\Sigma \subset \Gamma$ (Γ are cel puțin un simbol în plușQ)

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
funcția de tranziție:
dă starea următoare,
simbolul cu care e înlocuit cel curent
și mutarea la stânga sau dreapta
(în unele versiuni, echivalente, capul poate să rămână pe loc)

Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)

Σ: mulțimea finită a *simbolurilor de intrare* (din șirul inițial)

Γ: mulțimea simbolurilor de pe bandă (care pot fi scrise);
important: $\Sigma \subset \Gamma$ (Γ are cel puț in un simbol în plușQ)

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
funcția de tranziție:
dă starea următoare,
simbolul cu care e înlocuit cel curenț
și mutarea la stânga sau dreapta
(în unele versiuni, echivalente, capul poate să rămână pe loc)

$q_0 \in Q$: starea inițială a automatului de control

Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)

Σ: mulțimea finită a *simbolurilor de intrare* (din șirul inițial)

Γ: mulțimea simbolurilor de pe bandă (care pot fi scrise);
important: $\Sigma \subset \Gamma$ (Γ are cel puț in un simbol în plușQ)

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
funcția de tranziție:
dă starea următoare,
simbolul cu care e înlocuit cel curenț
și mutarea la stânga sau dreapta
(în unele versiuni, echivalente, capul poate să rămână pe loc)

$q_0 \in Q$: starea inițială a automatului de control

Q $\in \Gamma \setminus \Sigma$: simbolul vid (blanc) (Q $\notin \Sigma$)
toate celulele cu excepția unui număr finit sunt inițial vide

Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)

Σ: mulțimea finită a *simbolurilor de intrare* (din șirul inițial)

Γ: mulțimea simbolurilor de pe bandă (care pot fi scrise);
  important: $\Sigma \subset \Gamma$ (Γ are cel puțin un simbol în plușQ)

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
  funcția de tranziție:
    dă starea următoare,
    simbolul cu care e înlocuit cel curent
    și mutarea la stânga sau dreapta
  (în unele versiuni, echivalente, capul poate să rămână pe loc)

$q_0 \in Q$: starea inițială a automatului de control

Q $\in \Gamma \setminus \Sigma$: simbolul vid (blanc) (Q $\notin \Sigma$)
  toate celulele cu excepția unui număr finit sunt inițial vide

$F \subseteq Q$: mulțimea stărilor finale, automatul se oprește (halt)

Important:

Spre deosebire de un automat DFA/NFA, o mașină Turing **nu se oprește** la terminarea șirului de intrare!

Important:

Spre deosebire de un automat DFA/NFA, o mașină Turing **nu se oprește** la terminarea șirului de intrare!

Execuția continuă până când se ajunge într-una din **stările finale**:

de **acceptare**: șirul este acceptațface parte din limbaj

de **rejectare**: șirul este respinșnu face parte din limbaj

Important:

Spre deosebire de un automat DFA/NFA, o mașină Turing **nu se oprește** la terminarea șirului de intrare!

Execuția continuă până când se ajunge într-una din **stările finale**:
de **acceptare**: șirul este acceptat,face parte din limbaj
de **rejectare**: șirul este respins,nu face parte din limbaj

Există situații în care nu se ajunge **niciodată** într-o stare finală!

Pentru anumite limbaje și anumite șiruri de intrare, mașina Turing poate intra într-un **ciclu infinit,** *fără să accepte sau să respingă vreodată* șirul de intrare primit!

Mașini Turing de tip subrutină

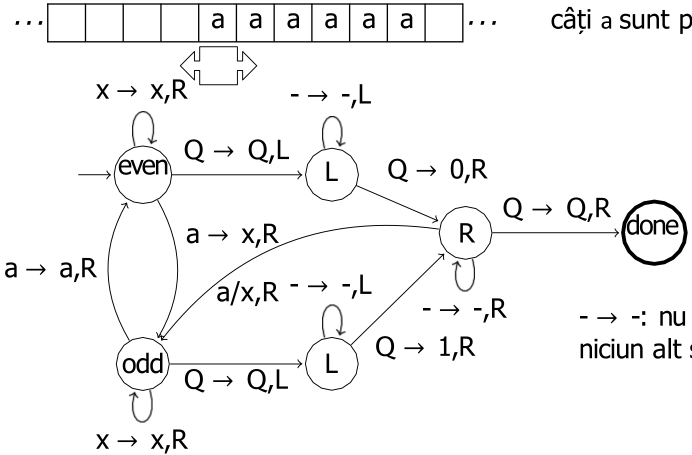
O mașină Turing este *de tip subrutină* dacă, în loc să accepte sau să respingă un șir de intrare, *efectuează anumite transformări* asupra acestuia

după care intră într-o stare finală (marcată *done* sau *halt*).

Astfel, pe bandă rezultă un nou șir, care poate fi prelucrat mai departe sau acceptat/respins de o altă mașină Turing.

Mașinile Turing subrutină pot fi folosite pentru a *compune* mașini Turing mai *complexe* din mașini Turing mai *simple*.

Exemplu: numără simboluri și scrie numărul în binar



câți a sunt pe bandă?

- → -: nu modifică niciun alt simbol

obține fiecare bit din nr. de a

⊢: schimbă a cu x din 2 în 2

⊢: scrie 0 sau 1 după paritate

repetă până ∄ a: *done*

QQQQaaaaaaQ ⊢ QQQQxaxaxaQ

⊢ QQQ0xaxaxaQ ⊢ QQQ0xxxaxxQ

⊢ QQ10xxxaxxQ ⊢ QQ10xxxxxxQ

⊢ Q110xxxxxxQ ⊢ Q110xxxxxxQ

done

Cât de puternică e o mașină Turing?

Ce putem calcula cu ajutorul ei?

Conceptual, un calculator ideal e un calculator cu memorie nelimitată (RAM, HDD/SSD, etc.).

Un calculator ideal *poate simula* o mașină Turing (poate face tot ce face și aceasta)

O mașină Turing *poate simula* un calculator ideal (poate face aceleași lucruri: aritmetica, cicluri, decizii, variabile, etc.) !

Practic poate descrie *orice calcul* (implementabil prin program)

Metodă de calcul efectivă

O metodă de calcul efectivă este un *sistem computațional* cu următoarele proprietăți:

- calculul consistă dintr-o *serie de pași*
- există *reguli fixe* conform cărora un pas e urmat de un altul
- orice calcul care produce un rezultat va ajunge la acesta într-un *număr finit de pași*
- orice calcul care ajunge la un rezultat ajunge la un rezultat *corect*

Calculabilitate – Teza Church-Turing

Ce se poate *calcula*, și cum putem defini această noțiune ?

Teza Church-Turing (o afirmație despre noțiunea de *calculabilitate*)

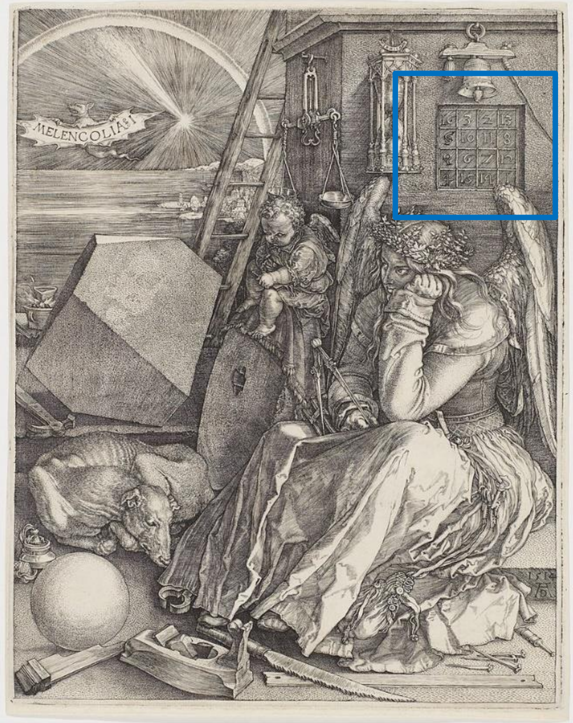
Orice metodă de calcul efectivă este *echivalentă cu*, sau *mai slabă* decât, o mașină Turing.

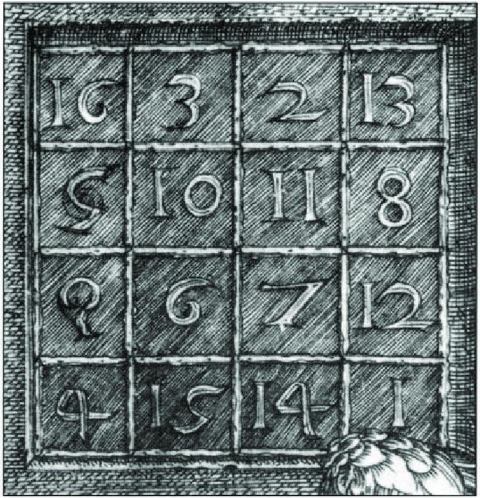
Următoarele modele de calcul sunt echivalente:

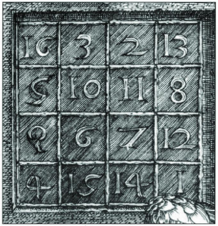
- lambda-calculul
- mașina Turing
- funcțiile recursive

Gravura Melancolia
1514
Albrecht Dürer
- renascentist german

Gravura Melancolia
1514
Albrecht Dürer
- renascentist german




16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1



Vă mulțumesc!

Bibliografie

Conținutul cursului se bazează pe materialele de anii trecuți de la cursul de LSD, predat de conf. dr. ing.Marius Minea și ș.l. dr. ing. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)