Регулярни изрази и автомати

Въведение в регулярните изрази и автомати

Всеки език, без значение дали е писмен или говорим, е съставен от думи, поради това трябва да имаме основни знания за тях. Очертава се единственият математически модел, който е в основата на всички говорни и езикови процеси: автомата. Регулярните изрази са други средства за обясняване и спецификация на думите и дърветата на решенията. Те са стандартно означаване на числа с условни знаци и за характеризиране на текстови последователности.

Регулярни изрази

Регулярните изрази (първо въведени от Клийни (1965г.)) са формула на специален език, която се използва за определяне на прости класове от низове.

Дефиниция

Регулярен израз е низ, представляващ шаблон, който се използва за:

- проверка дали някакъв низ отговаря на някакви условия
- търсене в низ
- извличане на части от низ
- замяна на подниз с друг

Основни шаблони на регулярни изрази

Търсенето при регулярните изрази изисква шаблон и множество от текстове, които се претърсват.

Регулярни изрази	Примерни съответствия
/еноти/	“Интерестни връзки към еноти и лемур”
/а/	“Мария беше спряна от Мона”
/каза Клара/	“Моят подарък, моля,” **каза Клара**
/песен/	“ нашата прекрасна песен”
/!/	“Вие сте крадец!”

Основни шаблони на регулярни изрази

- Регулярните изрази правят разлика между малки и големи букви. Можем да разрешим този проблем с използването на квадратни скоби [и].

Регулярен израз	Съответствия	Примерни шаблони
/ [еЕ]нот/	Енот или енот	“Енот”
/ [1234567890] /	Някакво число	“Числа от 7 до 5”

Основни шаблони на регулярни изрази

Квадратните скоби заедно със символа ^ могат да се използват за определяне на това, кой единичен знак не трябва да се съдържа.

Регулярен израз	Съответствия	Примерни шаблони
[^ А-Я]	Не главни букви	“Неговото име е Иван”
[е ^]	“е” или “^”	“Гладен ^”
[^о О]	“е” или “^”	“Гладен ^”
а^ б	Моделът “а^ б”	“Виж **а^ б** ”

Основни шаблони на регулярни изрази

Нямаме отговор на нашият въпрос дали енот или еноти? Не можем да използваме квадратните скоби, за това използваме питанката /?/, което означава “ знакът с предимство или нищо”.

Регулярен израз	Съответствия	Примерни шаблони
еноти?	енот или еноти	“**енот**”
ка?то	както или като	“**както**”

Основни шаблони на регулярни изрази

Разглеждаме езика на овцете, който се състои от низове:

баа!

бааа!

ббааа!

баааааа!

Друг оператор който ни позволява да казваме “няколко знака от а” е знакът * обикновено наричан Клийни*. Звездата означава “нула или много символа от предишния знак”. Изразът / [аб]* / означава “нула или много а-та или б-та”. Това ще съответства на низове аааа или абабаб или ббб.

Основни шаблони на регулярни изрази

Много важен специален знак е точката (/ . /) , който означава всеки отделен символ. Използва се често заедно със звездата на Клийни * и означава “който и да е низ от знаци”.

Котвите са специални знаци.Шаблонът / ^Аз / съответства на думата Аз ако е в началото на реда. Символът $ се използва за означаване на място в края на реда и /^Кучето\./$ съответства на ред в, който се съдържа фразата “Кучето”.

Разделяне, групиране и приоритет

Не можем да използваме квадратните скоби за търсене, имаме нужда от нов разделителен оператор, ще използваме символа | . Моделът / котка | куче / съответства на низа котка или на низа куче.

Кръглите скоби са полезни, когато използваме Клийни*. Имаме ред с колони Колона 1,Колона 2,Колона 3. Изразът / Колона [0-9]+ */ няма да съответства на модела от колони. С кръглите скоби можем да напишем изразът / (Колона [0-9]+ *)*/.

Памет

Търсим моделът ‘ the Xer they were, the Xer they will be’, където трябва двата X да бъдат в един и същи низ. Правим това като обграждаме първия X с кръгли скоби и поставяме втория X в числовият оператор \1, както следва: the(.*)er they were the \1er they will be.Това ще съответства на The bigger they were, the bigger they will be.Числовият оператор може да използва и други числа: \2 означава повторение на целият израз. Например the (.*)er they (.*), the\1er they \2 ще съответства на The bigger they were, the bigger they were.

Прост пример за регулярен израз

Трябва да се напише регулярен израз за намиране на английския пълен член “the” в различни случаи.

/the/- Прост (но неверен) пример;

/[tT]he/- по този начин се извикват и други думи, в които се съдържа пълният член “the” (напр. other или theology), затова се определя инстанция с граници на думата от двете страни, т.е. /\b[tT]he\b/;

/[^a-z][tT]he[^a-z]/- уточняват се инстанциите, в които няма букви от всяка страна на the;

/(^|[^a-z])[tT]he[^a-z]/.

По-сложен пример

Потребителят иска компютър с повече от 233 Mhz и 32MB дисково пространство за по-малко от 1000$. Първо ще потърсим изрази като 233Mhz или 32 MB, “Compaq” или “Mac”, 999.99$.

- регулярен израз за цените, т.е. за знак долар, последван от низ от цифри- /$[0-9]+/
- работа с дробта на доларите, като се добавя десетична запетая и две цифри след нея: /$[0-9]+\.[0-9][0-9]/.

По-сложен пример

Този пример позволява само $199.99, но не и $199. Трябва да направим центовете нездължителни, и да се уверим , че сме в границите на думата: /\b$[0-9]+(\.[0-9][0-9])?\b/.

Няколко примера за определянето на скоростта на процесора и или за размера на паметта: /\b[0-9]+ *(Mhz|[Mm]egahertz)\b/; /\b[0-9]+ *(MB|[Mm]egabytes?)\b/.

/\b(Win|Win95|Win98|WinNT|Windows *(NT|95|98)?)\b/; /\b(Mac|Macintosh|Apple)\b/.

Оператори

предефинирани символни класове

\w символ от дума - буква, цифра или '_' , еквивалент: [a-zA-Z0-9_]

\W обратното на \w

\s празно пространство, еквивалент: [\ \t\r\n\f]

\S обратното на \s

\d цифра, еквивалент: [0-9]

\D обратното на \d

специални символи за количество

{n} предният символ или група се повтаря n пъти

{n,} предният символ или група се повтаря n или повече пъти

{,m} предният символ или група се повтаря m или по-малко пъти

{n,m}предният символ или група се повтаря между n и m пъти

*? , +? , ?? , {n}? , {n,}? , {,m}? , {n,m}?

Крайни автомати

Всеки регулярен израз може да бъде изпълнен като краен автомат с изключение на регулярните изрази, които използват свойствата на паметта. Крайните автомати могат да се използват и за много други неща. Езика на овцата се дефенира като низа:

baa!

baaa!

baaaa!

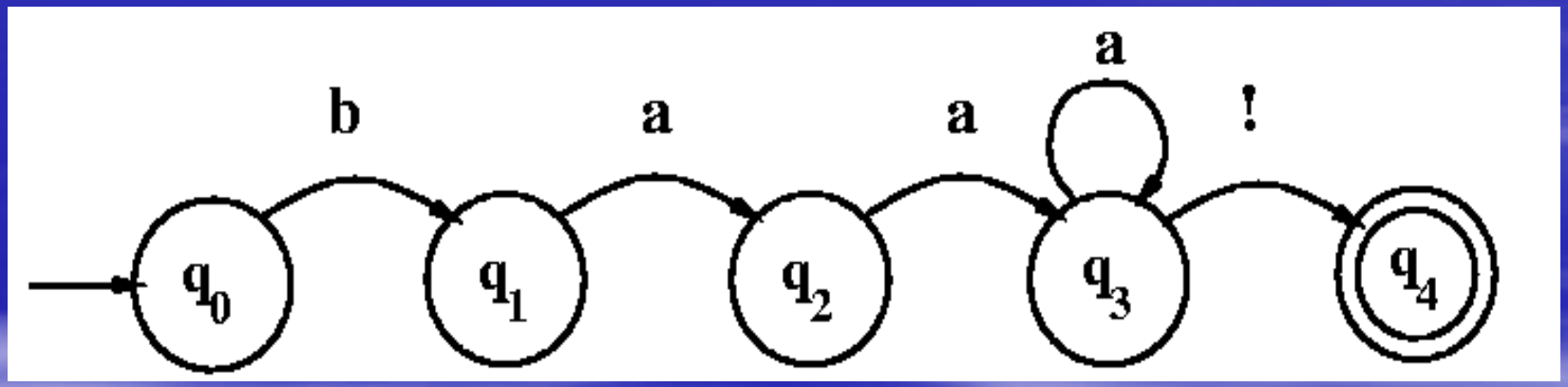
baaaaa!

baaaaaa!

Регулярният израз за този низ е baa+!.

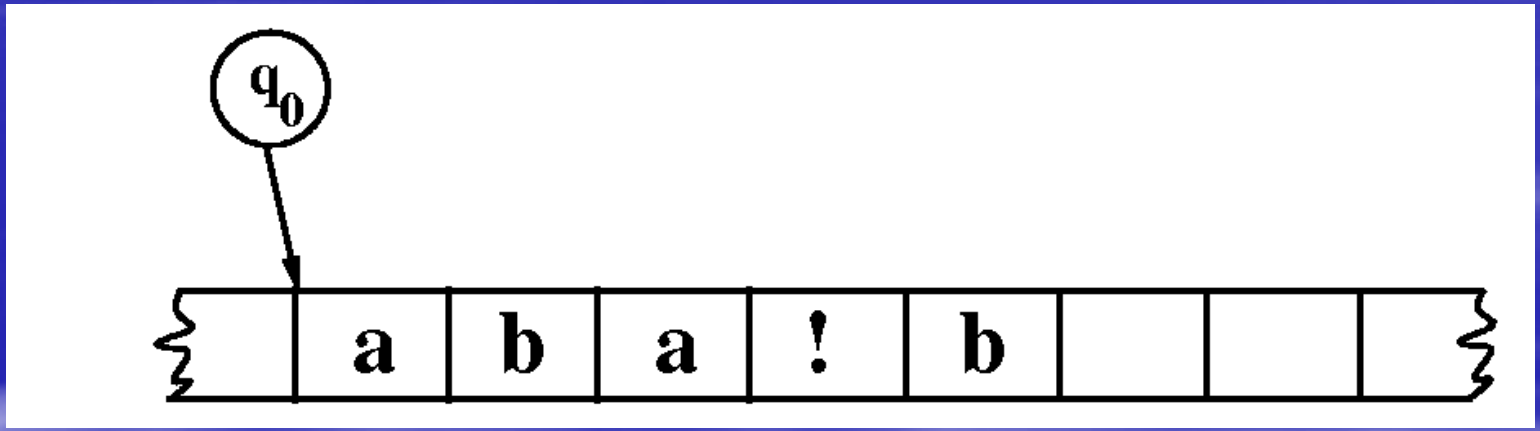
Крайни автомати

Фигура 1: Краен автомат за езика на овцата.



Крайни автомати

Фигура 2



Крайни автомати

Може да представим автомата и с таблица на състоянията. Както в означенията на графа, таблицата на състоянията представя стартово състояние, допустимите състояния и преходите, завършващи със съответните им символи.

	Input		
State	b	a	!
0	1	Ø	Ø
1	Ø	2	Ø
2	Ø	3	Ø
3	Ø	3	4
4:	Ø	Ø	Ø

Крайни автомати

По формално краен автомат се дефинира чрез следните пет параметъра:

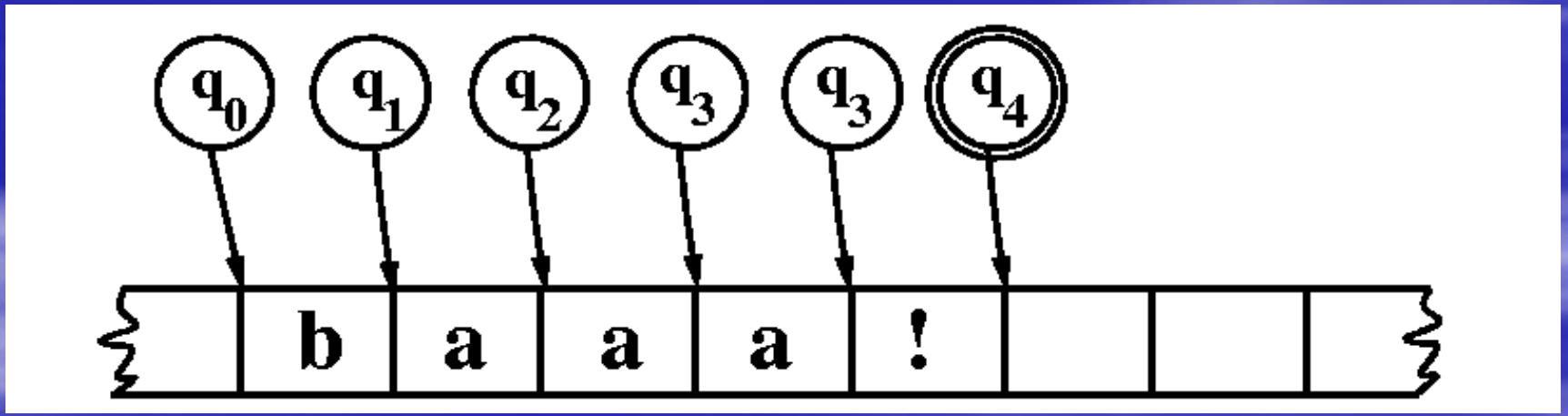
- Q: множество от N състояния q0, q1, …, qN
- ∑: множество от входове (символи на азбуката)
- q0: началното състояние
- F: набор от крайни състояния, F е подмножество на Q
- δ(q,i): преходна функция или преходна матрица между състояния. Даването на състояния qeQ и вход символа ieΣ, δ(q,i) връща ново състояние q'eQ. δ е така свързано с Q x ∑ и Q;

Крайни автомати

function D-RECOGNIZE(*tape, machine*) **return** accept or rejest
index ← Beginning of tape
current-state ← Initial state of machine
loop
if End of input has been reached **then**
if current-state is an accept state **then**
return accept
else
return reject
elsif *transition-table[current-state, tape[index]]* is empty **then**
return reject
else
current-state ← *transition-table[current-state, tape[index]]*
index ← *index+1*
end

Крайни автомати

Фигура 3 очертава изпълнението на алгоритъма за крайният автомат за езика на овцете даващ прост вход baa!



Официални езици

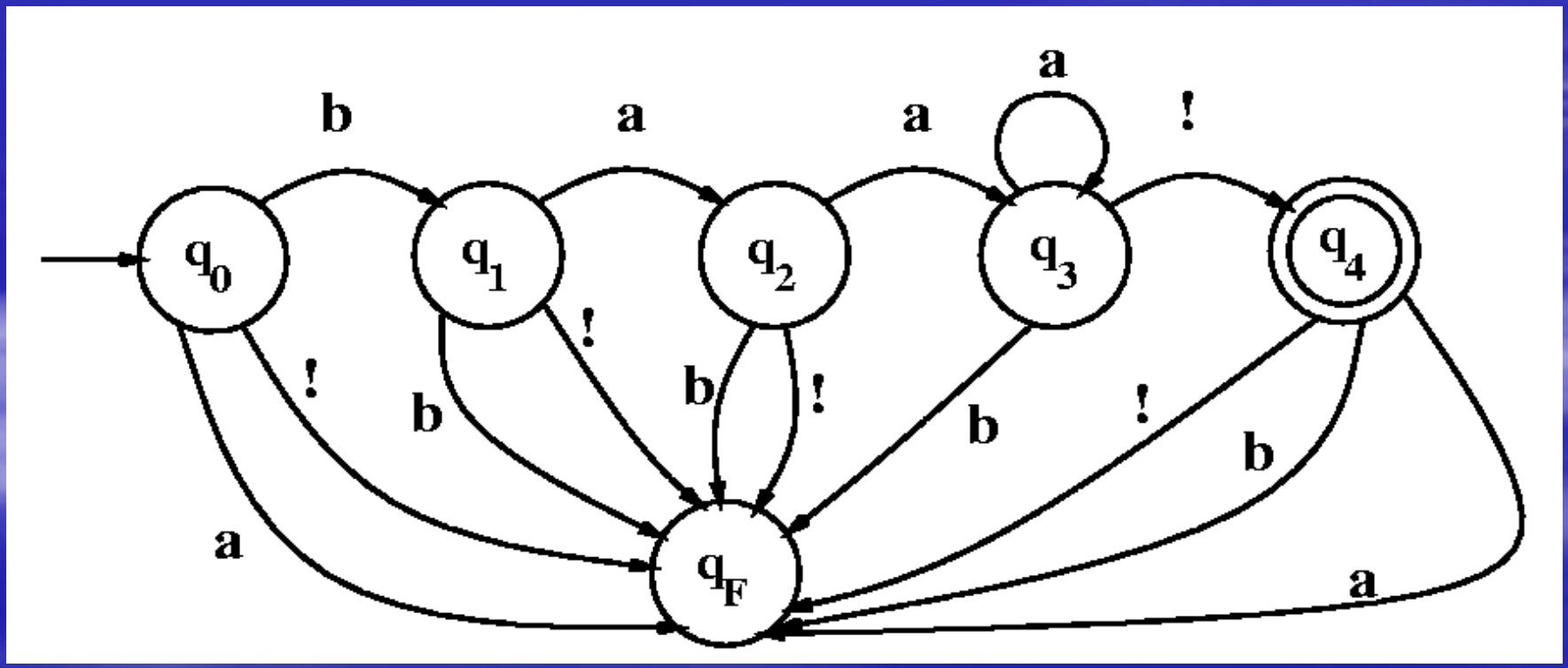
Набор от низове, всеки низ е съчетание от символи от азбуката. Използва се за моделиране на част от естествения език, като част от фонетиката, морфологията или синтаксиса. Също може да моделира различни състояния. В лингвистиката, терминът генеративна граматика се използва за означаване на граматиката на официалния език.

Азбуката за езика на овцата е зададена: ∑ = a,b,!. Давайки модел m (като точен краен автомат), ние можем да използваме L(m), за да означим официалния език дефиниран чрез m, който е набора от безкрайности:

L(m) = {baa!, baaa!, baaaa!, baaaaa!, baaaaaa! …}

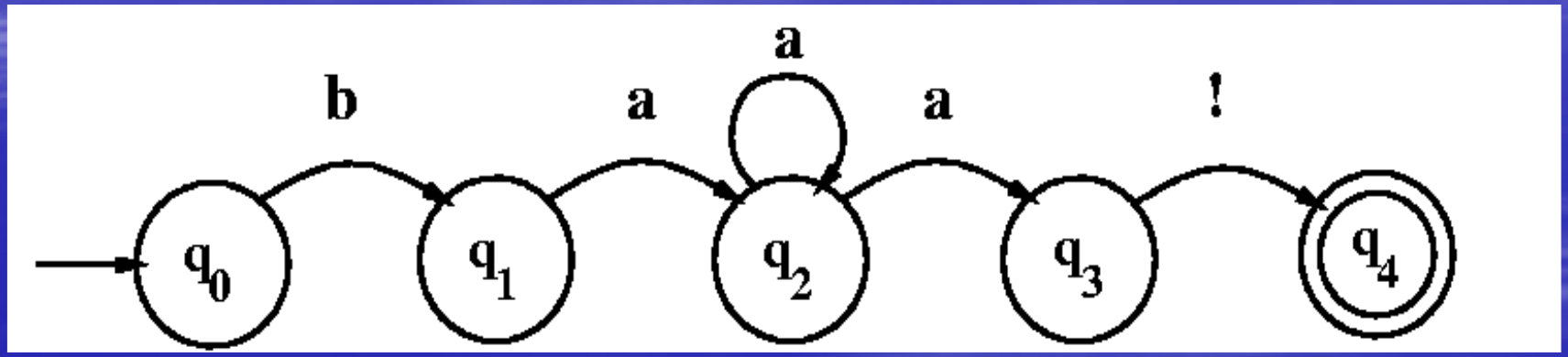
Официални езици

Фигура 4: Добавяне на пропадащи състояния към КА за езика на овцата от Фигура 1.

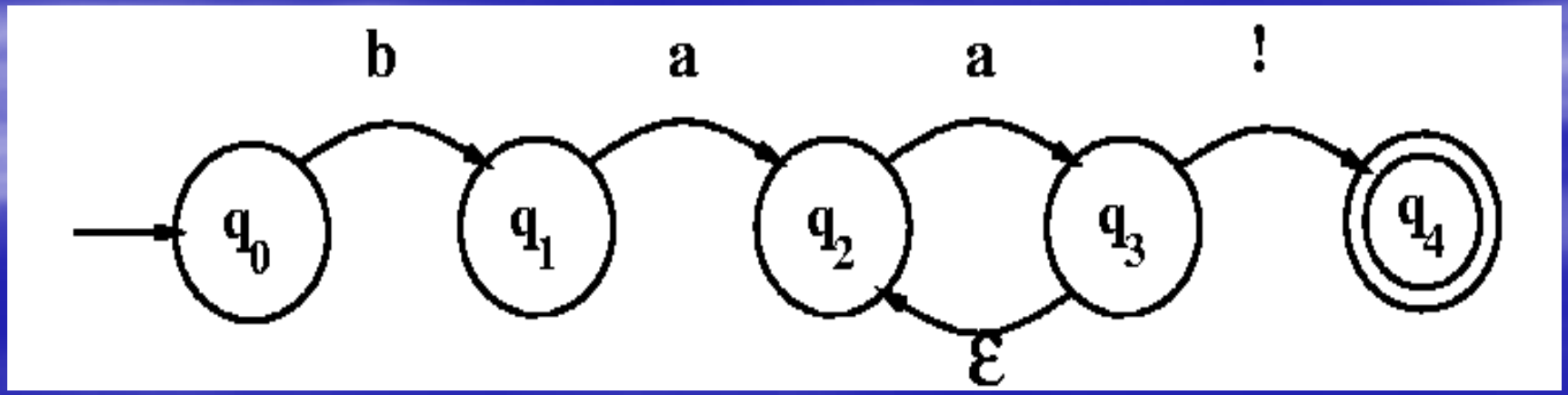


Недетерминирани крайни автомати

Автомати, които могат да взимат решения, наричаме НДКА.



Фигури 5 и 6 показжат НДКА за езика на овцата.



Използване на НДКА за приемане на низове

Ако искаме да разберем дали низ е инстанция на даден езика или не, и използваме НДКА за разпознаването, трябва да следваме грешна дъга и да я отстраним, когато я открием. Съществуват три стандартни решения на този проблем:

- Връщане назад
- Гледане във вътрешността
- Паралелизъм

Връщане назад

Таблица на преходите

	Input			
State	b	a	!	ϵ
0	1	$\emptyset$	$\emptyset$	$\emptyset$
1	$\emptyset$	2	$\emptyset$	$\emptyset$
2	$\emptyset$	2,3	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$	4	$\emptyset$
4:	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Този подход предлага правенето на избор да става, по начин, който ни позволява да се връщаме назад до първоначалното състояние при грешен избор, като се запомнят всички алтернативни състояния и се запази достатъчна информация тях.

Връщане назад

Следжащата фигура показва алгоритъм за използване на НДКА за разпознаване на вход низ. Функцията ND-RECOGNIZE използва променлива agenda, за да запази информация за всички установени непроучвани избора, създадени по време на изпълнението на функцията. Променливата, която търси в текущото състояние current-search-state представя различни избори, които могат да бъдат използвани.

Връщане назад

Функцията ND-RECOGNIZE започва чрез създаване на първоначално състояние на търсене и го поставя в променливата – agenda. Следва функцията NEXT, която се извиква, за да възстанови отделна точка в списака agenda и да я посочи на променливата в текущото състояние - current-search-state.

Връщане назад

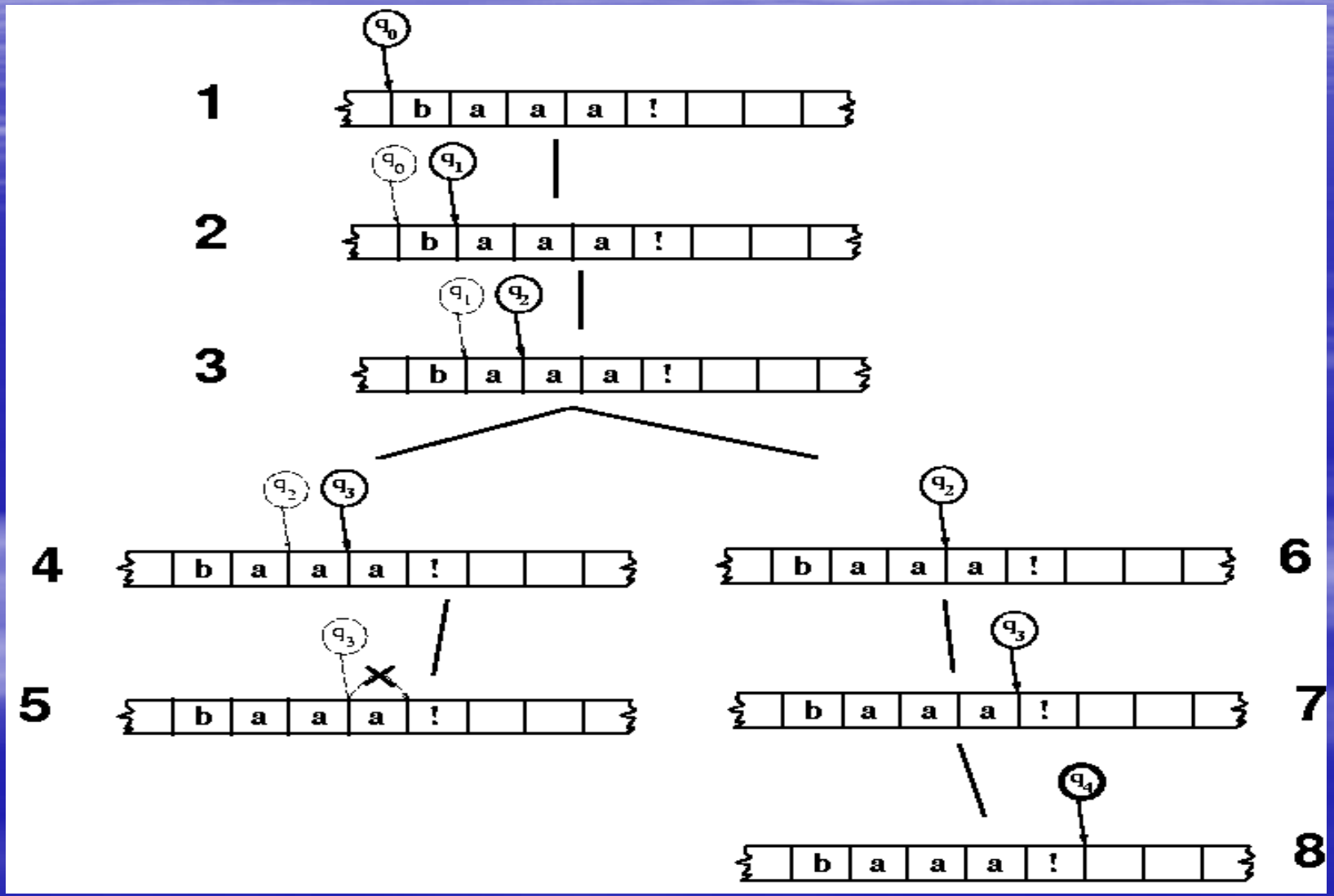
Първата задача ND-RECOGNIZE на главната примка е да определи дали всички съдържания на лентата са разпознати успешно. Това става чрез повикване на приемащо състояние ACCEPT-STATE?. Чрез извикването на функцията за създаване на нови състояния GENERATE-NEW-STATES, се генерират набор от възможни следващи стъпки, при неуспех.

function ND-RECOGNIZE(tape, machine) **returns** accept or reject
agenda ←{(Innitial state of machine, beginning of tape)}
current-search-state ←NEXT(*agenda*)
loop
ifACCEPT-STATE? (*current-search-state*) returns true **then**
return accept
else
agenda← agendaȕGENERATE-NEW-STATES(*current-search-state*)
if *agenda* is empty **then**
return reject
else
current-search-state ←NEXT(*agenda*)
end
function GENERATE-NEW-STATE(*current-state*) **returns** a set of search-states
current-node← the node the current search-state is in
index ← the point on the tape the current search-state is looking at
return a list of search states from transition table as follows:
(*transition-table*[*current-node, e*], *index*)
ȕ
(*transition-table*[*current-node, tape* [*index*]], *index+1*)
function ACCEPT-STATE?(*state*) **returns** true or false
current-state ← the node search-state is in
index ← the point on the tape search-state is looking at
if *index* is at the end of the tape **and** *current-node* is an accept state of
machine **then**
return true
else
return false

Разпознаването като търсене

Важно е да се разбере защо ND-RECOGNIZE връща стойност от отхвърлянето само, когато текущата променлива е празна. За разлика от D-RECOGNIZE той не връща охвърляне, когато достигне края на лентата в неприемливо машинно състояние. Това е, защото при недетерминирани случаи, такива блокове отбелязват неуспех.

Фигура 7 илюстрира процеса на ND-RECOGNIZE като се опитва да се справи с входа baaa!.



Разпознаването като търсене

D-RECOGNIZE извършва задачата за разпознаване на низове в регулярните езици, чрез използване на способ за систематично проучване на възможните пътища през машината.

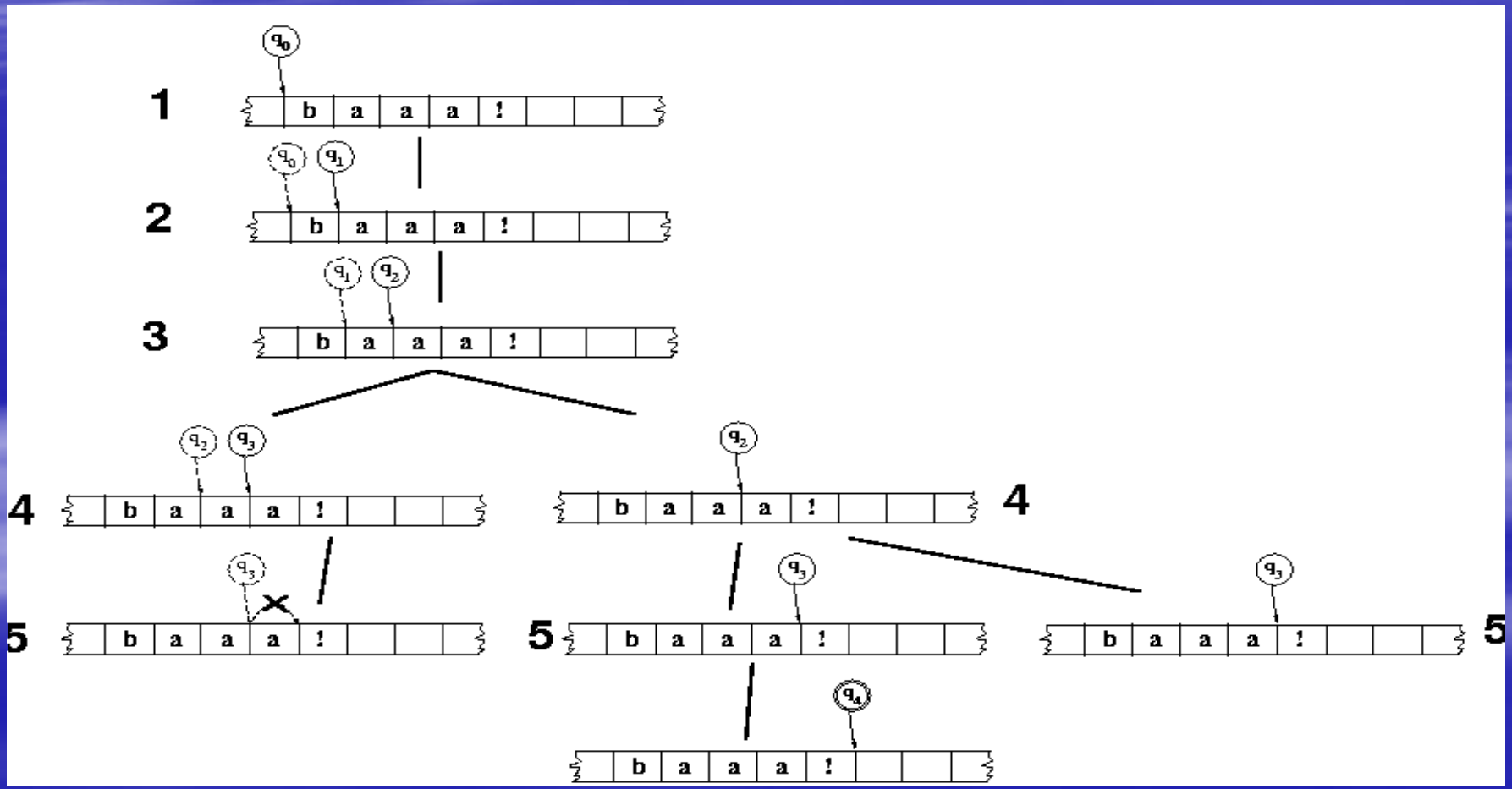
Алгоритмите като ND-RECOGNIZE, които си служат със системни търсения за решения се наричат търсене в пространството на състоянията.

Разпознаването като търсене

Последователността, по която НДКА избира следващото състояние, което да изследва се определя от стратегията на търсене. Търсенето първо в дълбочина или стратегията Last In First Out съответства на текущата променлива като стек. За да се изследва текущата променлива (agenda) се определя стратегията й за търсене: търсенето първо в широчина или стратегията First In First Out съответсва на текущата променлива като опашка.

Разпознаването като търсене

Краен автомат, показващ метода търсене първо в дълбочина.



Свързване на детерминираните и недетерминираните автомати

Съществува прост алгоритъм за превръщане на НДКА в еквивалентен КА, въпреки че броят на състоянията в този еквивалентен детерминиран автомат е много по-голям.

Основният подход на доказателството е изграден начина,по който НДКА прави синтактичен разбор на входната си информация. Може да се види при Люис и Пападимитроу (1981г.) или Хопкрофт и Улман (1979г.).

Резултатният ДКА може да има толкова състояния, колкото има отделения набор от състояния в първичния НДКА.

Регулярни езици и крайни автомати

Класът на езиците, които са формулирани чрез регулярни изрази е точно същият като класът на езиците, които се определят от КА. Класът на регулярните езици (или регулярни набори) обикновено е следния:

- 0 е регулярен език;
- всяко a є ∑ Ư е, където {a} е регулярен език;
- Ако L1 и L2 са регулярни езици, то:
  - L1 • L2 = {xy | x є L1, y є L2 }, конкатенацията на L1 и L2
  - L1 Ư L2, обединението или дизюнкцията на L1 и L2
  - L*1, приближението на Клийни на L1

Регулярни езици и крайни автомати

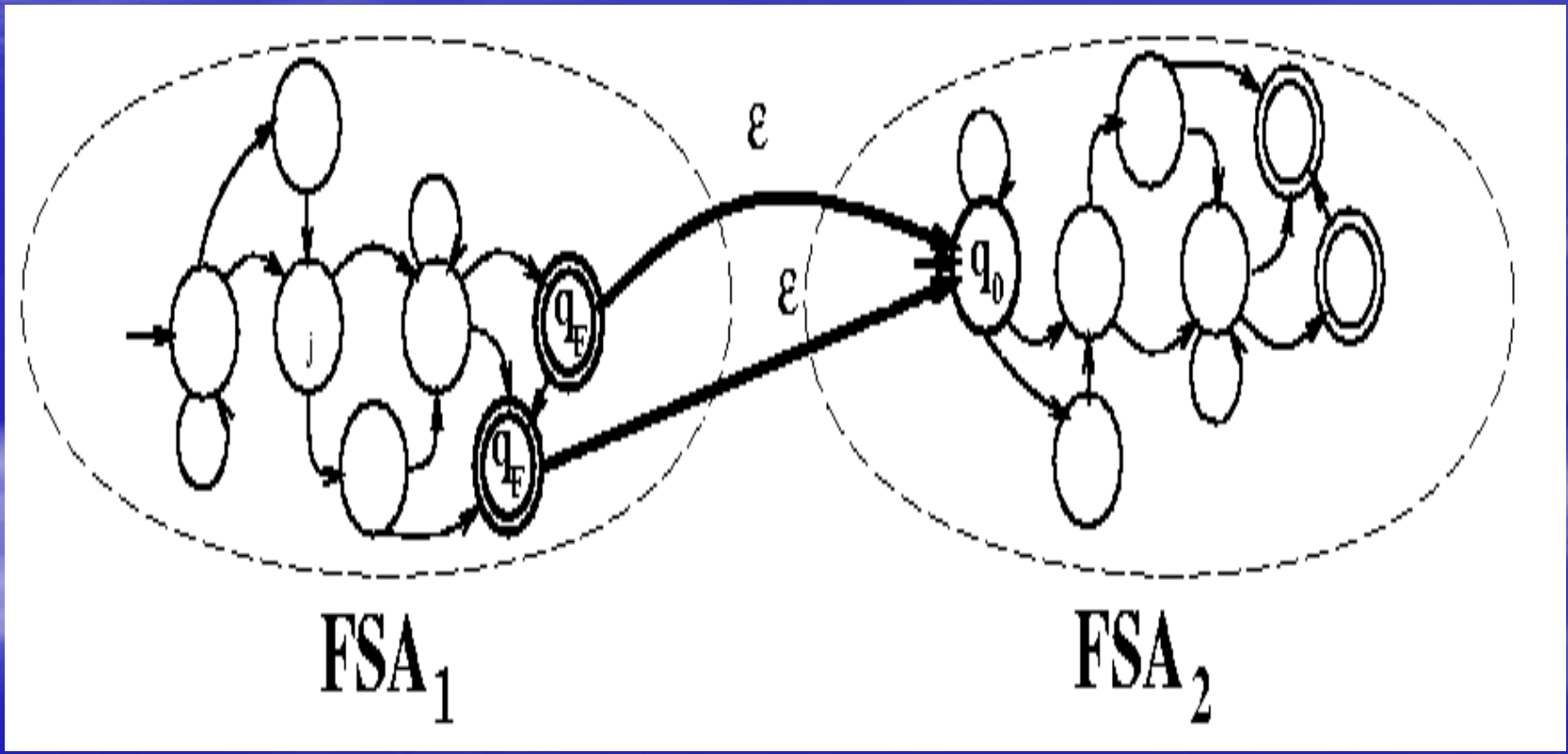
Регулярните езици са също затворени от следващите операции:

- Сечение
- Разлика
- Допълване
- Обръщение

Доказателство, че регулярните изрази са еквивалентни на КА може да се открие при Хопкрофт и Улман (1979г.),се състои от две части и има индуктивен подход.

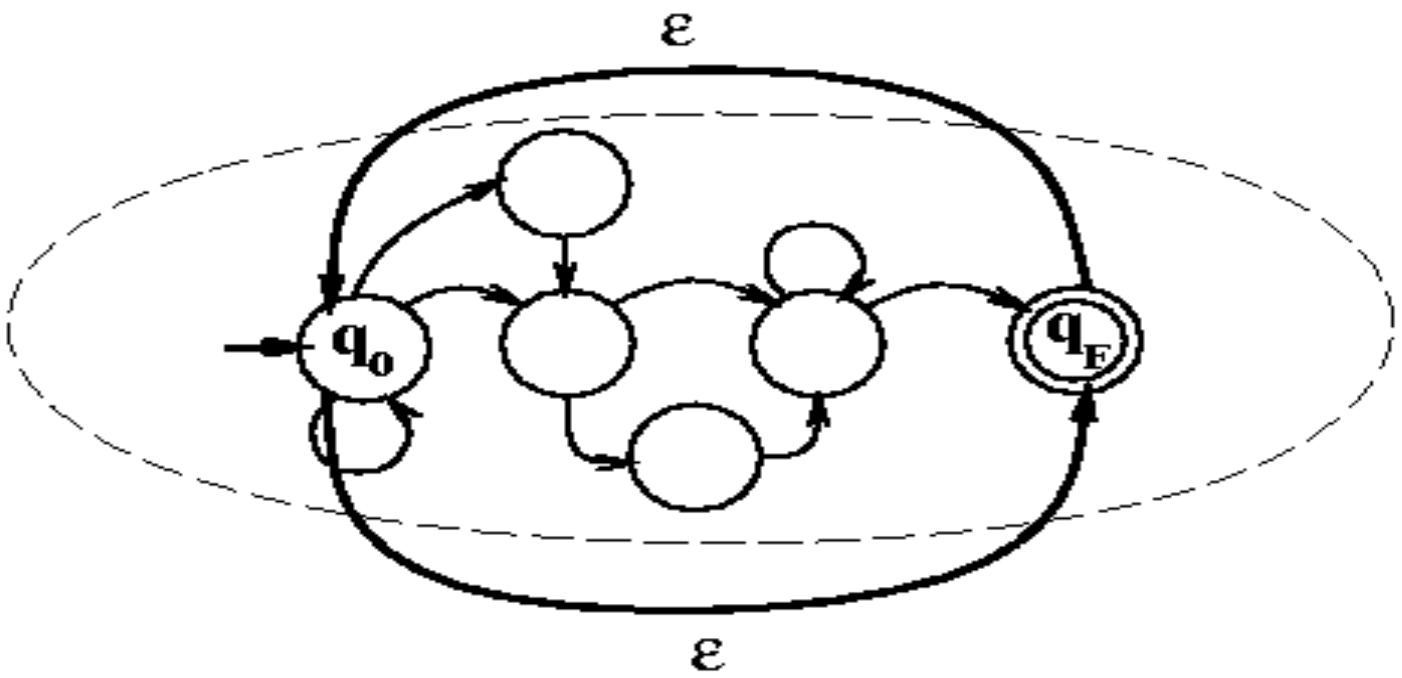
Регулярни езици и крайни автомати

Конкатенация



Регулярни езици и крайни автомати

Приближение



Библиографични и исторически бележки

Крайният автомат възниква през 1950г. В началото модела на Тюринг (1936г.) за компилация на алгоритми се разглежда като основа на съвременната компютърна наука.

Втората парадигма е работата на Шанън за информационната теория.

Третата основа е Мак Калъч-Питс (Мак Калъч и Питс, 1943г.) опростен модел като част от “компютърен елемент”, който меже да се опише в термините на логиката на задачата.

Библиографични и исторически бележки

На базата на Мак Калъч-Питс метода, Клийни (1951г.) и (1956г.) дефинира крайните автомати и регулярните изрази и доказва тяхната еднозначност. НДКА са въведени от Робин и Скот (1959г.), които също доказват тяхната еднозначност за детерминирането им.

Съществуват много глобални успехи внасящи основната математическа теория за автомати; някои личности, които се занимават с това: Хопкрофт и Улман (1979г.) и Люис и Пападимитроу (1981г.).