Limbaje Regulate

Întrebarea centrală a teoriei calculabilităţii este

Ce este un calculator?

Pentru a da un răspuns uşor de înţeles la această întrebare vom descrie un calculator idealizat numit *model de calcul*. Ca orice model ştiinţific, şi modelul de calcul este precis doar în anumite privniţe şi imprecis în altele. Din acest motiv teoria automatelor prezintă mai multe modele de calcul, în funcţie de caracteristicele care ne interesează.

Vom începe cu **maşina cu stări finite** sau **automatul finit**.

1 Automate finite

Definiţia 1 *Un* **automat finit** *determinist (sau* **AFD***)este un 5-tuplu* $(Q, \Sigma, \delta, q_0, F)$ *unde*

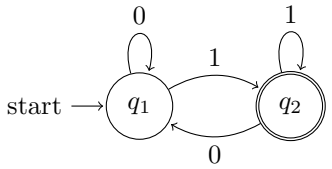
1. Q *este o mulţime finită ale cărei elemente se numesc* **stări**,
2. Σ *este o mulţime finită numită* **alfabet**,
3. $\delta : Q \times \Sigma \to Q$ *este* **funcţia de tranziţie**,
4. $q_0 \in Q$ *este* **starea de start** *(sau* **iniţială***) şi*
5. $F \subseteq Q$ *este* **mulţimea stărilor finale** *(sau* **de acceptare***)*

Exemplu de automat finit:

1. $M_1 = (Q, \Sigma, \delta, q_1, F)$, where $Q = \{q_1, q_2\}$, $\Sigma = \{0, 1\}$, $F = \{q_2\}$, and δ is described as

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

2. Reprezentare alternativă: Diagramă de stări. De exemplu



1.1 Calculabilitate

$A = (Q, \Sigma, \delta, q_0, F)$.

- Un *cuvânt* este un şir finit de elemente $w = a_1 a_2 \dots a_n$ cu $a_i \in \Sigma$ pentru $1 \leq i \leq n$. Numărul n se numeşte *lungimea* cuvântului w.
- Mulţimea tuturor cuvintelor cu elemente din Σ este Σ^*.
- ϵ este cuvântul cu lungimea 0.
- Funcţia de tranziţie δ se poate extinde să fie definită recursiv $\delta : Q \times \Sigma^* \to Q$ astfel:

$$\delta(q, w) = \begin{cases} q & \text{dacă } w = \epsilon, \\ \delta(\delta(q, a), w') & \text{dacă } w = aw'. \end{cases}$$

- Automatul finit A calculează acceptând cuvinte. Spunem că automatul finit A **acceptă** cuvântul w dacă $\delta(q_0, w) \in F$.
- Limbajul **recunoscut** (sau **acceptat**) de automatul A este $L(A) = \{w \mid \delta(q_0, w) \in F\}$.
- Spunem că un limbaj este **regulat** dacă este recunoscut de un automat finit determinist.

Observaţii:

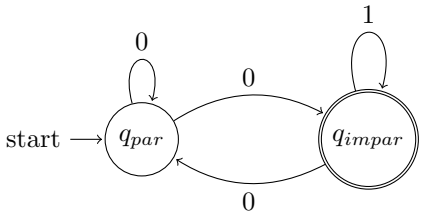
- Pentru a verifica dacă $w = a_1 a_2 \dots a_n$ este acceptabil, automatul A traversează cuvântul w de la stânga la dreapta calculând o secvenţă de n stări:

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

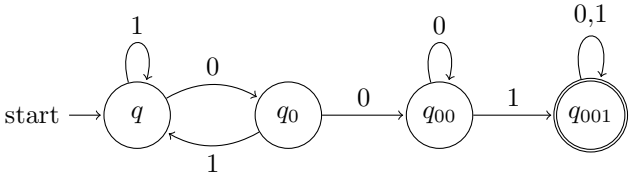
  Cuvântul w este acceptat de A dacă $q_n \in F$.

1.2 Designul automatelor finite

1. Un automat care recunoaşte toate cuvintele din $\{0,1\}^*$ cu număr impar de 1:



2. Un automat care recunoaşte toate cuvintele din $\{0,1\}^*$ care conţin 001:



1.3 Operaţii regulate

În general, un limbaj este o mulţime de cuvinte din Σ^*.

Definiţia 2 *Operaţiile regulate cu limbaje sunt:*

Reuniune: $A \cup B = \{w \mid w \in A \text{ sau } w \in B\}$.

Produs: $AB = \{vw \mid v \in A, w \in B\}$.

Închidere (Kleene): $A^* = \{w_1 w_2 \dots w_n \mid w_1, w_2, \dots, w_n \in A, n \geq 0\}$. *Observăm că*

$$A^* = \{\epsilon\} \cup A \cup AA \cup AAA \cup \dots = \bigcup_{n=0}^{\infty} A^n.$$

De exemplu, dacă $A = \{lup, miel\}$ şi $B = \{bun, rău\}$ atunci

- $A \cup B = \{lup, miel, bun, rău\}$,
- $AB = \{lupbun, luprău, mielbun, mielrău\}$,
- $A^* = \{\epsilon, lup, miel, luplup, lupmiel, miellup, mielmiel, lupluplup, \dots\}$

Exerciţiul 1 Să se arate că dacă A şi B sunt limbaje regulate atunci

1. $A \cup B$ este limbaj regulat.
2. AB este limbaj regulat.
3. A^* este limbaj regulat.

Idee de bază: Pentru a arăta că $A \circ B$ este un limbaj regulat atunci când A şi B sunt limbaje regulate şi $\circ$ o operaţie anume, se consideră date două automate finite M_A şi M_B astfel încât $L(M_A) = A$ şi $L(M_B) = B$. Trebuie construit un automat M cu ajutorul celor 2 automate existente astfel încât $L(M) = L(M_A) \circ L(M_B)$. De exemplu, pentru reuniune se presupun date automatele $M_A = (Q_1, \Sigma, \delta_1, q_1, F_1)$ şi $M_B = (Q_2, \Sigma, \delta_2, q_2, F_2)$ astfel încât $L(M_A) = A$ şi $L(M_B) = B$. Automatul $M = (Q, \Sigma, \delta, q_0, F)$ pentru $A \cup B$ poate fi construit să simuleze simultan comportamentul lui M_1 şi M_2 în timp de verifică dacă un cuvânt dat este acceptabil. În acest caz putem defini

- $q_0 := (q_1, q_2)$,
- $Q := Q_1 \times Q_2$,
- $F := (F_1 \times Q_2) \cup (Q_1 \times F_2)$ şi
- $\delta : Q \times \Sigma \to \Sigma$ cu $\delta((q, q'), a) := (\delta_1(q, a), \delta_2(q, a))$ pentru toţi $(q, q') \in Q$ şi $a \in \Sigma$.

Fie cuvântul $w = a_1 a_2 \dots a_n \in \Sigma^*$. Se observă uşor că avem $w \in L(M)$ dacă şi numai dacă $\delta((q_1, q_2), a_1 a_2 \dots a_n) \in F$ dacă şi numai dacă există

- $q_1^1, \dots, q_n^1 \in Q_1$ astfel încât $q_1^1 = \delta_1(q_1, a_1), \dots, q_n^1 = \delta_1(q_{n-1}^1, a_n)$, adică $\delta_1(q_1, w) = q_n^1$
- $q_1^2, \dots, q_n^2 \in Q_2$ astfel încât $q_2^1 = \delta_2(q_2, a_1), \dots, q_n^2 = \delta_2(q_{n-1}^2, a_n)$, adică $\delta_2(q_1, w) = q_n^2$

şi $(q_n^1, q_n^2) \in F$. Deoarece $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$, avem că $\delta(q_0, w) \in L(M)$ dacă şi numai dacă $\delta_1(q_1, w) \in F_1$ sau $\delta_2(q_2, w) \in F_2$, adică dacă $w \in L(M_A) \cup L(M_B) = A \cup B$. □

2 Nedeterminism

Nedeterminismul este un concept foarte important în teoria computaţională. Ideea principală este de a permite unui automat finit să aleagă care să fie starea $\delta(q, a)$ în care trece atunci când este într-o stare q şi citeşte un simbol $a \in \Sigma$. Formal, acest lucru înseamnă că în loc să avem δ o funcţie care determină starea următoare $\delta(q, a)$, un automat nedeterminist are δ o relaţie pentru care $\delta(q, a)$ este o mulţime de stări următoare plauzibile.

Definiţia formală a unui automat finit nedeterminist este

Definiţia 3 *Un* **automat finit nedeterminist** *(sau* **AFN***) este on 5-tuplu* $(Q, \Sigma, \delta, q_0, F)$ *unde*

1. Q *este o mulţime finită ale cărei elemente se numesc* **stările** *automatului,*
2. Σ *este o mulţime finită numită* **alfabetul** *de intrare al automatului,*
3. $\delta : Q \times \Sigma_\epsilon \to 2^Q$ *este* **funcţia de tranziţie**, *unde* $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$ *şi* 2^Q *este mulţimea submulţimilor lui* Q*,*
4. q_0 *este o stare specială numită* **starea de start** *(sau* **iniţială***) şi*
5. $F \subseteq Q$ *este* **mulţimea stărilor finale** *(sau* **de acceptare***).*

La fel ca automatele finite obişnuite (care sunt deterministe), şi automatele finite nedeterministe pot fi reprezentare cu diagrame de stări. De exemplu:

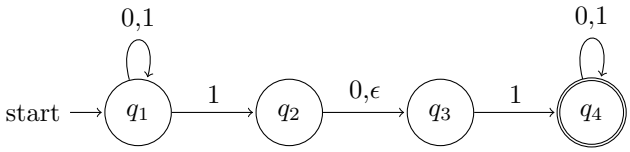


Figure 1: Diagramă de stări a unui automat finit nedeterminist N_1

Diagrama de stări a automatului N_1 este

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

2.1 Calculabilitate

Presupunem că $A = (Q, \Sigma, \delta, q_0, F)$ este un automat finit nedeterminist.

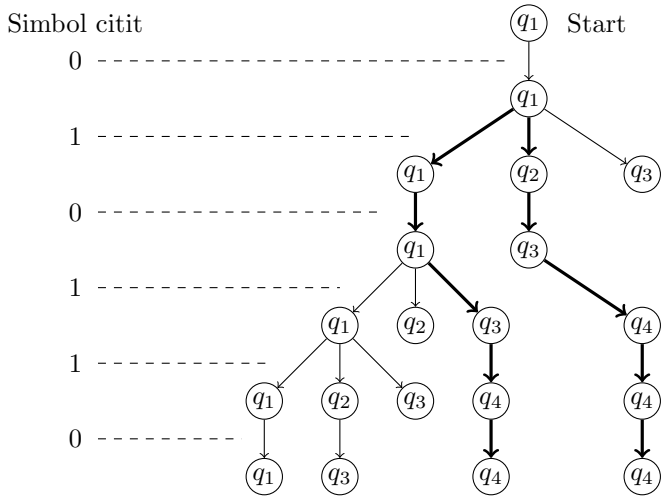
- Şi automatele finite nedeterministe calculează acceptând cuvinte.
- Spunem că automatul finit nedeterminist A **acceptă** un cuvânt $w \in \Sigma^*$ dacă putem scrie $w = y_1 y_2 \dots y_n$ cu $y_1, y_2, \dots, y_n \in \Sigma_\epsilon$ şi putem găsi stările $q_1, q_2, \dots, q_n \in Q$ astfel încât $q_1 \in \delta(q_0, y_1), q_2 \in \delta(q_1, y_2), \dots, q_n \in \delta(q_{n-1}, y_n)$ şi $q_n \in F$.

- Mulţimea cuvintelor acceptate de un AFN N este un limbaj, $L(N)$.

Nedeterminismul poate fi considerat o formă de calcul paralel în care procese sau thread-uri independente pot fi executate concurent. De exemplu, când un AFN se află în o stare q, citeşte simbolul a şi $\delta(q, a) = \{q_1, q_2, q_3\}$, automatul are de ales una din trei alternative. În astfel de situaţii, automatul nedeterminist crează procese independente, câte unul pentru fiecare alternativă. Aceste procese vor fi executate concurent.

Calculul nedeterminist poate fi vizualizat ca un arbore de stări posibile. Rădăcina arborelui este starea de start iar ramurile arborelui reprezintă calcule alternative. Fiii unui nod sunt stările alternative în care poate trece automatul când citeşte simbolul de intrare. Automatul acceptă cuvântul citit dacă cel puţin una din ramuri se termină cu o stare finală.

De exemplu, calculul efectuat de automatul nedeterminist N_1 din Figura 1 când citeşte cuvântul `010110` poate fi ilustrat astfel:



Ramurile îngroşate indică execuţii care se termină cu succes, într-o stare finală.

Utilitatea automatelor nedeterministe

Vom vedea mai târziu că orice automat nedeterminist poate fi convertit în un automat determinist care recunoaşte acelaşi limbaj. Două automate care recunosc acelaşi limbaj se numesc automate **echivalente**. De obicei, când vrem să construim un automat pentru un limbaj dat, este mult mai uşor să construim un automat nedeterminist decât unul determinist.

De obicei automatele nedeterministe pentru un limbaj dat au mult mai puţine stări decât un automat determinist echivalent. Deasemenea, automatele nedeterministe sunt uşor de înţeles cum funcţionează dacă examinăm diagrama lor de stări.

Exemplul 1 Presupumen că vrem să construim un automat finit care recunoaşte limbajul A de cuvinte de 0 şi 1 care conţin 1 pe a treia poziţie de la coadă. De exemplu, cuvântul `000100` este în acest limbaj, dar `0011` nu este. Automatul nedeteminist ilustrat în Figura 2 recunoaşte limbajul A. Intuitiv, automatul N_2 stă în starea q_1 până când ,,bănuieşte" că

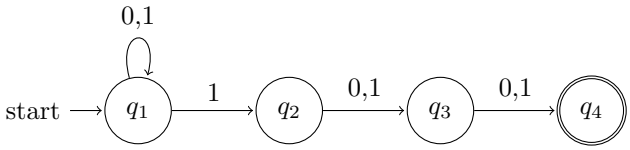
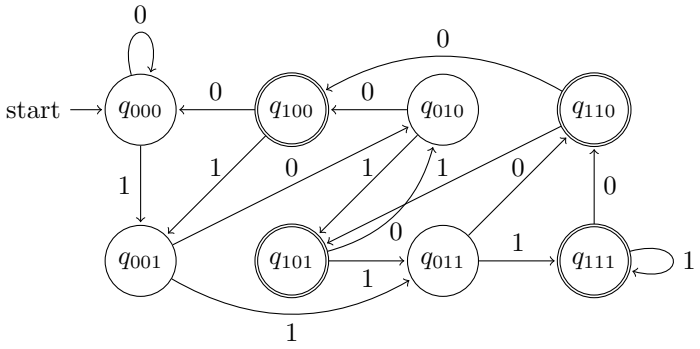


Figure 2: Automat finit nedeterminist N_2 pentru limbajul A

mai are 3 simboluri de citit. În acel moment, dacă simbolul de intrare este 1 automatul trece în starea q_2 şi apoi foloseşte stările q_3 şi q_4 pentru a ,,verifica" dacă bănuiala a fost corectă.

Dacă vrem să construim un automat finit determinist pentru limbajul A avem nevoie de cel puţin 8 stări. Automatul ilustrat mai jos este cel mai mic automat finit determinist care recunoaşte limbajul A.



Exemplul 2 Automatul finit nedeterminist din Figura 3 recunoaşte limbajul A al cuvintelor de forma $a^m b^n c^p$ unde $m, n, p \in \mathbb{N}$. De exemplu, $aabbbbcc \in A$ dar $acba \notin A$.

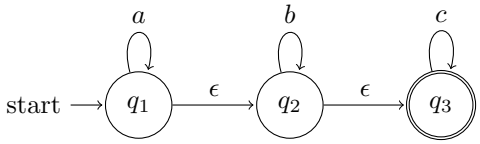


Figure 3: AFN N pentru cuvintele de forma $a^m b^n c^p$ unde $m, n, p \in \mathbb{N}$

2.2 Echivalenţa dintre AFD şi AFN

În general, spunem că două automate sunt **echivalente** dacă recunosc acelaş limbaj.

Vom demonstra că automatele finite deterministe şi cele nedeterministe recunosc aceleaşi limbaje. Pentru a dovedi acest lucru trebuie arătat că:

1. Pentru orice AFD M se poate construi un AFN N echivalent cu M. Această construcţie este imediată: dacă $M = (Q, \Sigma, \delta, q_0, F)$ este un AFD atunci putem defini AFN-ul $N = (Q, \Sigma, \delta', q_0, F)$ cu $\delta' : Q \times \Sigma_\epsilon \to 2^Q$ astfel: $\delta'(q, \epsilon) = \emptyset$ şi $\delta'(q, a) = \{\delta(q, a)\}$ pentru orice $q \in Q$ şi $a \in \Sigma$.

2. Pentru orice AFN N se poate construi un AFD M echivalent cu N.

Fie $N = (Q, \Sigma, \delta, q_0, F)$ un AFN cu n stări. Atunci Q are 2^n submulţimi. Fie $\mathcal{Q}'$ mulţimea tuturor submulţimilor lui Q. În continuare explicăm cum poate fi construit un AFD M echivalent cu N.

Pentru început considerăm cazul în care N nu conţine ϵ-tranziţii. În acest caz automatul finit determinist $M = (Q', \Sigma, \delta', \{q_0\}, F')$ poate fi definit astfel:

- $\delta' : \mathcal{Q} \times \Sigma \to \mathcal{Q}$, $\delta'(Q', a) = \bigcup_{q \in Q'} \delta(q, a)$ pentru toţi $Q' \in \mathcal{Q}$ şi $a \in \Sigma\}$,
- $F' = \{Q \in \mathcal{Q}' \mid Q \cap F \neq \emptyset\}$.

Este uşor de verificat că $L(M) = L(N)$, deci AFD-ul M este echivalent cu AFN-ul N.

Dacă N conţine ϵ-tranziţii atunci este util să definim operaţia $E(Q')$pentru fiecare submulţime Q' a lui Q astfel: $E(Q') := Q' \cup \{q \in Q \mid$ există o secvenţă $q' \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} q$ de 1 sau mai multe ϵ-tranziţii de la o stare $q' \in Q'$ la $q\}$. Operaţia $E(Q')$ se numeşte ϵ-închiderea lui Q'. Cu ajutorul acestei operaţii putem defini automatul finit determinist $M = (\mathcal{Q}', \Sigma, \delta', E(\{q_0\}), F')$ în felul următor:

- $\delta' : \mathcal{Q} \times \Sigma \to \mathcal{Q}$, $\delta'(Q', a) = E(R)$ unde $R = \bigcup_{q \in Q'} \delta(q, a)$ pentru toţi $Q' \in \mathcal{Q}$ şi $a \in \Sigma\}$,
- $F' = \{Q \in \mathcal{Q}' \mid Q \cap F \neq \emptyset\}$.

Şi în acest caz se poate demonstra că $L(M) = L(N)$, deci AFD-ul M este echivalent cu AFN-ul N.

2.3 Construcţia de automate pentru limbaje construite cu operaţii regulate

Dacă A, A_1 şi A_2 sunt limbaje regulate atunci $A_1 \cup A_2$, $A_1 A_2$ şi A^* sunt limbaje regulate.

Presupunem că avem AFN-uri pentru A, A_1 şi A_2. Figurile 4, 5 şi 6 ilustrează construcţia unui AFN pentru $A_1 \cup A_2$, $A_1 A_2$ şi A^* bazat pe automatele lui A, A_1 şi A_2.

3 Expresii regulate

În aritmetică putem folosi operaţii aritmetice pentru a construi expresii precum

$$(5 + 3) \times 4.$$

Această expresie are valoarea 32. Într-un mod asemănător putem folosi operaţiile regulate pentru a contrui expresii care descriu limbaje. Expresiile construite în acest fel se numesc **expresii regulate**. De exemplu, expresia

$$(0 \cup 1)0^*$$

este regulată şi are ca valoare limbajul cuvintelor care încep cu `0` sau `1` urmat de orice număr de `0`. Acest rezultat se obţine dacă se analizează părţile expresiei regulate. Mai întâi

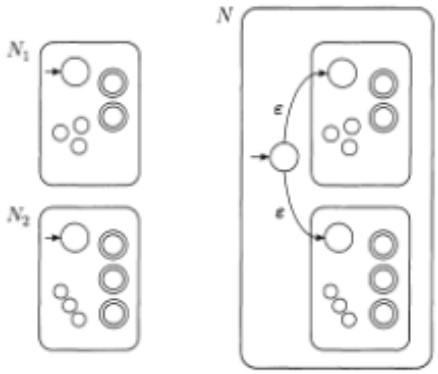


Figure 4: Construcţia unui automat care recunoaşte limbajul $A_1 \cup A_2$

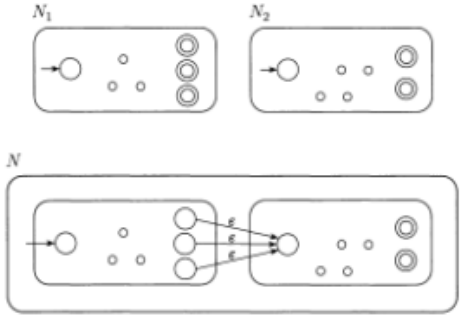


Figure 5: Construcţia unui automat care recunoaşte limbajul $A_1 A_2$

simbolurile `0` şi `1` sunt o prescurtare pentru limbajele $\{0\}$ şi $\{1\}$. Deci $(0 \cup 1)$ înseamnă $(\{0\} \cup \{1\})$. Valoarea acestei subexpresii este limbajul $\{0, 1\}$. Partea `0`* înseamnă $\{0\}^*$, iar valoarea ei este limbajul şirurilor finite de 0. Concatenarea celor două subexpresii ataşează şiruri din cele două părţi pentru a obţine valoarea întregii expresii.

Expresiile regulate au un rol important în aplicaţiile informaticii care operează cu text. Adeseori, utilizatorii doresc să caute cuvine care satisfac anumite şabloane. Programe utilitare precum AWK şi GREP din Unix, limbaje moderne de programare precum PERL şi numeroase editoare de texte oferă mecanisme de descriere a şabloanelor de căutare cu expresii regulate.

Definiţia 4 Spunem că R este **expresie regulată** dacă R este de forma

1. a pentru un simbol a dintr-un alfabet Σ,
2. ϵ,
3. $\emptyset$,
4. $(R_1 \cup R_2)$ unde R_1, R_2 sunt expresii regulate,

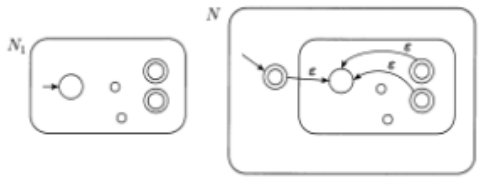


Figure 6: Construcţia unui automat care recunoaşte limbajul A^*

5. R_1R_2 unde R_1, R_2 sunt expresii regulate,
6. R_1^* unde R_1 este o expresie regulată.

Expresiile regulate ϵ şi $\emptyset$ nu trebuie confundate. Expresia ϵ reprezintă limbajul $\{\epsilon\}$ format doar din cuvântul vid, în timp ce $\emptyset$ reprezintă limbajul care nu conţine nici un cuvânt.

Parantezele din expresii regulate pot fi omise şi în acest caz evaluarea lor se face considerând următoarea ordine de precendenţă a operatorilor: stea (*), concatenare şi apoi reuniune. Notaţia R^+ este folosită ca abreviere pentru $R\,R^*$, deci $R^* \cup \epsilon = R^+$. Deasemenea, R^k este folosită ca prescurtare pentru concatenarea repetată $\underbrace{R\,R\,\ldots\,R}_{k \text{ ori}}$. Când vrem să facem distincţie între o expresie regulată şi limbajul care este valoarea ei,vom scrie $L(R)$ pentru limbajul descris de expresia regulată R.

Exemplul 3 În exemplele următoare considerăm că alfabetul Σ este $\{0, 1\}$.

1. $0^*10* = \{w \mid w$ conţine un singur $1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w$ conţine cel puţin un $1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w$ conţine subşirul $001\}$.
4. $\Sigma^*(01^+)^* = \{w \mid$ orice 0 în w este urmat de cel puţin un $1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w$ este un cuvânt cu lungime pară$\}$.
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid w$ este un cuvânt cu lungime multiplu de $3\}$.
7. $01 \cup 10 = \{01, 10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w$ începe şi se termină cu acelaş simbol$\}$.
9. $(0 \cup \epsilon)1^* = 01^* \cup 1^*$.
10. $(0 \cup \epsilon)(1 \cup \epsilon) = \{\epsilon, 0, 1, 01\}$.
11. $1^*\emptyset = \emptyset$. Concatenarea mulţimii vide la orice limbaj produce mulţimea vidă.
12. $\emptyset^* = \{\epsilon\}$.

3.1 Echivalenţa cu automatele finite

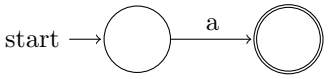
Expresiile regulate şi automatele finite sunt echivalente din punct de vedere al puterii de descriere, deoarece:

1. orice limbaj acceptat de un automat finit este valoarea unei expresii regulate, şi
2. orice limbaj care este valoarea unei expresii regulate este acceptat de un automat finit.

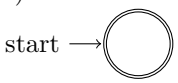
În continuare vom demonstra aceste două afirmaţii.

1. Dacă un limbaj este descris de o expresie regulată atunci limbajul este acceptat de către un automat finit. Să presupunem că limbajul este descris de o expresie regulată R. Vom a arăta cum putem converti R într-un AFN. Distingem 6 cazuri.

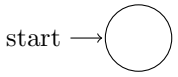
1. Dacă $R = a$ pentru un $a \in \Sigma$ atunci $L(R) = \{a\} = L(N)$ unde N este AFN-ul cu diagrama de stări



2. $R = \epsilon$. Atunci $L(R) = \{\epsilon\} = L(N)$ unde N este AFN-ul cu diagrama de stări



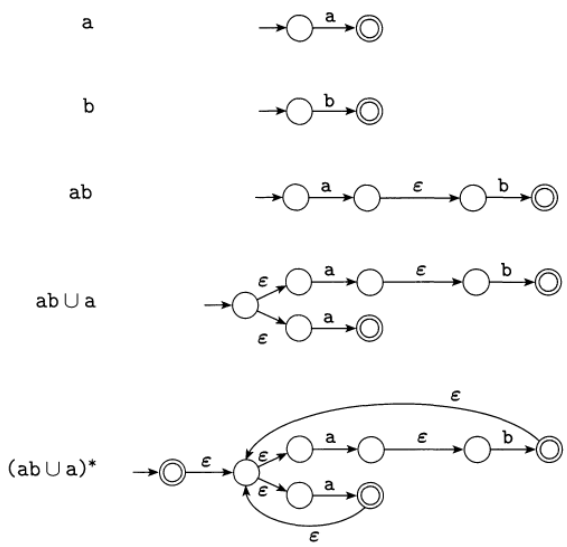
3. $R = \emptyset$. Atunci $L(R) = \emptyset = L(N)$ unde N este AFN-ul cu diagrama de stări



4. $R = R_1 \cup R_2$.
5. $R = R_1 R_2$
6. $R = R_1^*$.

În ultimele 3 cazuri folosim construcţiile ilustrate în figurile 4, 5 şi 6 care au fost folosite pentru a demonstra că limbajele regulate sunt închise în raport cu operaţiile regulate. Altfel spus, construim un AFN pentru R din AFN-urile pentru R_1 şi R_2 (sau doar R_1 în cazul 6).

Exemplul 4 Expresia regulată $(ab \cup a)^*$ poate fi convertită în un AFN efectuând secvenţa de cinci paşi ilustrată mai jos.



Exemplul 5 Expresia regulată $(a \cup b)^* aba$ poate fi convertită în un AFN efectuând secvenţa de şase paşi ilustrată în Figura 7.

2. Dacă un limbaj este recunoscut de un automat finit atunci limbajul este descris de o expresie regulată. Vom descrie o procedură de conversie a automatelor finite deterministe în expresii regulate echivalente. Procedura de construcţie este formată din 2 părţi. Mai întâi arătăm cum putem converti un AFD într-un **automat finit nedeterminist generalizat** (sau AFNG pe scurt), iar apoi arătăm cum putem converti un AFNG într-o expresie regulată.

Automatele finite nedeterministe generalizate sunt la fel ca automatele finite nedeterministe cu singura diferenţă cu tranziţiile pot fi etichetate cu orice expresii regulate în loc să fie etichetate doar cu simboluri din $\Sigma \cup \{\epsilon\}$. Un AFNG poate citi blocuri de simboluri (sau cuvinte) de intrare, pe când un AFN poate citi doar un singur simbol deodată. Un AFNG îşi schimbă starea de-a lungul unui arc de tranziţie dacă citeşte un cuvânt de simboluri de intrare care se află în limbajul descris de expresia regulată a arcului de tranziţie. Un AFNG este nedeterminist şi deci poate procesa un cuvânt de intrare în mai multe feluri. AFNG-ul acceptă cuvântul de intrare dacă cuvântul pe care-l procesează poate cauza AFNG-ul să treacă în o stare de ieşire atunci când termină de citit.

Pentru a simplifica lucrurile, putem presupune ca toate AFNG-urile sunt în o formă specială care satisface condiţiile următoare:

- Starea de pornire are arcuri spre fiecare altă stare dar nu are arcuri care să vină de la altă stare.
- Există o singură stare finală în care vin arcuri de la toate celelalte stări dar din care nu pleacă nici un arc. Cu excepţia stărilor iniţială şi finală, există un arc între orice două stări şi deasemenea de la orice stare la ea însăşi.

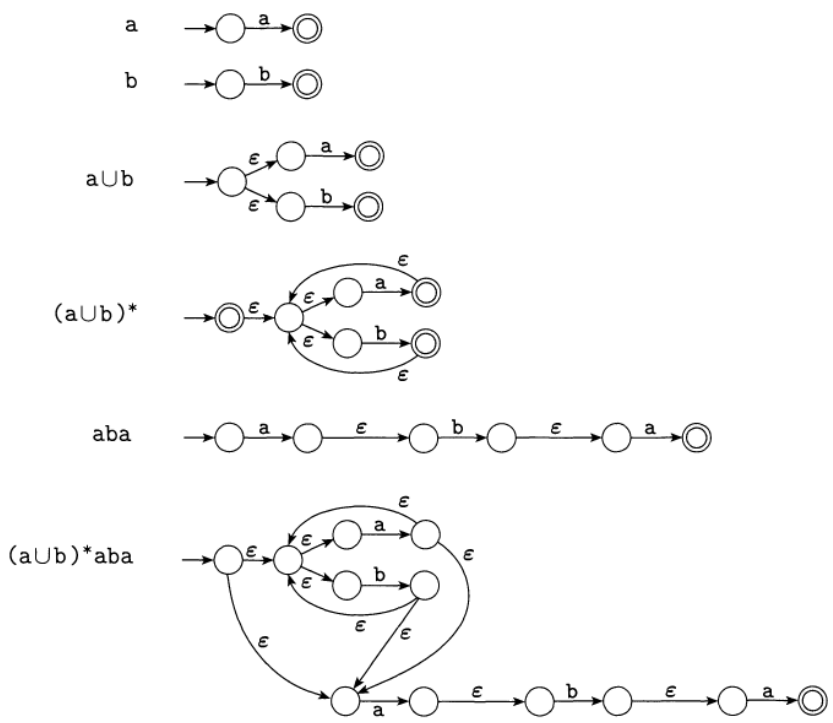


Figure 7: Construcţia unui automat finit nedeterminist pentru $(a \cup b)^*aba$

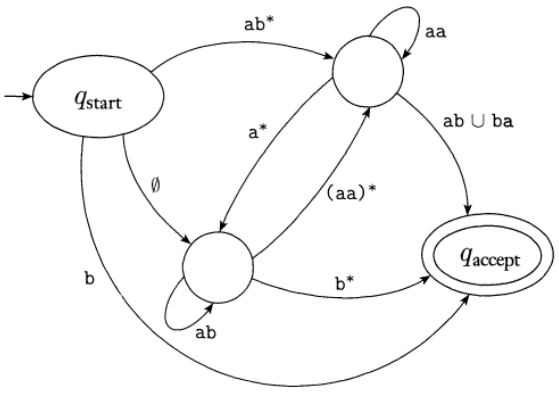


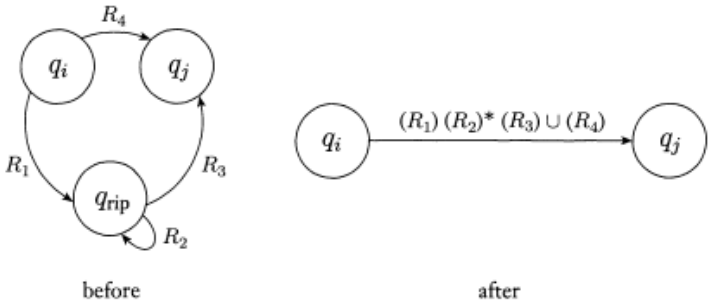
Figure 8: Un automat finit nedeterminist generalizat (AFNG)

Orice AFD poate fi convertit în un AFNG în formă specială:

- Se adaugă o stare iniţială nouă împreună cu un arc cu eticheta ϵ de la ea la starea iniţială veche.
- Se adaugă o stare finală nouă împreună cu arcuri cu eticheta ϵ de la toate stările finale vechi la starea finală nouă.
- Dacă există mai multe arcuri de la o stare la alta, le înlocuim cu un singur arc a cărui etichetă este reuniunea etichetelor arcurilor înlocuite.
- Dacă nu există un arc de la o stare la alta, adăugăm un arc cu eticheta $\emptyset$.

În continuare explicăm cum putem converti un AFNG cu formă specială în o expresie regulată. Fie k numărul de stări al AFNG-ului. Deoarece starea de start şi cea finală trebuie să fie diferite, avem $k \geq 2$. Dacă $k > 2$, construim un AFNG echivalent cu $k - 1$ stări. În momentul în care ajungem la $k = 2$, avem un AFNG cu un singur arc care pleacă de la starea de start la starea finală. Eticheta acestui arc este expresia regulată echivalentă.

Pasul crucial este construirea unui AFNG echivalent cu $k - 1$ stări atunci când $k > 2$. Acest process se face selectând o stare oarecare diferită de cea iniţială sau finală, eliminând-o şi apoi corectând toate arcurile de tranziţie ale AFNG-ului rămas încât să recunoască acelaş limbaj ca AFNG-ul iniţial. Presupunând că starea care se elimină este q_{rip}, arcul dintre stările q_i şi q_j se modifică astfel:



4 Limitări ale automatelor finite. Limbaje neregulate

Automatele finite deterministe au o putere limitată. Este important să ne dăm seama ce nu pot face automatele finite deterministe.

Pentru început vom lua în considerare limbajul $L = \{0^n 1^n \mid b \in \mathbb{N}\}$. Dacă încercăm să găsim un AFD pentru L, ne vom da seama că automatul trebuie să reţină câţi 0 au fost citiţi până la un moment dat. Deoarece numărul de 0 nu este limitat, automatul ar trebui să ţină evidenţa unui număr nelimitat de posibilităţi. Dar acest lucru este imposibil deoarece numărul de stări al unui AFD este finit.

În continuare vom descrie Lema de Pompare, un rezultat teoretic care ne ajută să definim o metodă de demonstrare a faptului că unele limbaje sunt neregulate.

4.1 Lema de Pompare

Acest rezultat teoretic afirmă că toate limbajele regulate au o anumită proprietate. Prin urmare, dacă putem demonstra că un limbaj oarecare nu are acea proprietate, putem concluziona că acel limbaj nu este regulat. Proprietatea se referă la faptul că toate şirurile din limbaj pot fi ,,pompate" dacă au lungimea mai mare decât o valoare anume numită **lungime de pompare**. Altfel spus, fiecare cuvânt suficient de lung conţine o secţiune care poate fi repetată ori de câte ori, iar cuvântul obţinut rămâne în limbaj.

Lema de Pompare. Dacă L este un limbaj regulat atunci există un număr p numit **lungime de pompare** astfel încât, dacă $s \in A$ are lungimea $|s| \geq p$ atunci s poate fi despărţit în 3 părţi $s = xyz$ pentru care au loc următoarele:

1. $xy^i z \in A$ pentru toţi $i \geq 0$.
2. $|y| > 0$.
3. $|xy| \leq p$.

Cum poate fi demonstrată Lema de Pompare?

Dacă L este un limbaj regulat, atunci L este recunoscut de un AFD $M = (Q, \Sigma, \delta, q_0, F)$. Alegem $p =$ numărul de stări din Q şi demonstrăm că orice cuvânt cu lungime $|s| \geq p$ poate fi despărţit în trei părţi $s = xyz$ pentru care au loc afirmaţiile din Lemă.

Distingem 2 cazuri. Primul caz este cel în care L nu conţine cuvinte de lungime p. În acest caz Lema de Pompare este evident adevărată deoarece toate cuvintele cu lungime mai mare sau egală cu p satisfac orice condiţii fiindcă astfel de cuvinte nu există. Al doilea caz este cel mai interesant. Dacă $w = a_1a_2 \dots a_n \in L$ are lungimea $n \geq p$, atunci M trece prin următoarea secvenţă de stări când citeşte cuvântul w:

$$q_0 \stackrel{a_1}{\to} q_1 \stackrel{a_2}{\to} q_2 \stackrel{a_3}{\to} \dots \stackrel{a_k}{\to} q_k \stackrel{a_{k+1}}{\to} \dots \stackrel{a_j}{\to} q_j \stackrel{a_{j+1}}{\to} \dots \stackrel{a_n}{\to} q_n$$

cu $q_n \in F$. Deoarece M are n stări, trebuie să existe două stări egale în secvenţa $q_0, q_1, \dots, q_n$. Presupunem că $q_i = q_j$ unde $0 \leq i < j \leq n$. În acest caz alegem $x = a_1 \dots a_k$, $y = a_{k+1} \dots a_j$ şi $z = a_{j+1} \dots a_n$. Se observă că toate cuvintele de forma xy^iz sunt acceptate de M deoarece:

- M trece din starea q_0 în q_i când citeşte cuvântul $x = a_1 \dots a_i$,
- M trece de i ori din starea q_k în q_k când citeşte cuvântul y^i unde $y = a_{k+1} \dots a_j$, şi
- M trece din starea q_j în starea finală q_n când citeşte cuvântul $z = a_{j+1} \dots a_n$.

Exemplul 6 Să se arate că limbajul $L = \{0^n1^n \mid n \geq 0\}$ nu este regulat.

Demonstraţie. Dacă L ar fi regulat atunci ar exista $p > 0$ astfel încât orice cuvânt 0^n1^n cu lungimea $2n \geq p$ se poate descompune în $0^n1^n = xyz$ cu $|y| > 0$, $|xy| < p$ şi $xy^iz \in L$ pentru toţi $i \geq 0$. Dacă $i = 0$ atunci $xz \in L$, deci $xz = 0^m1^m$ pentru $m \geq 0$. Dacă $i = 1$ atunci $xyz = 0^{m+r}1^{m+r} \in L$ pentru un număr $r > 0$ deoarece $|y| > 0$. Acest lucru este posibil doar dacă $x = 0^m, y = 0^r1^r$ şi $z = 0^m$. Însă în acest caz $zx^2z = 0^{m+r}1^n0^r1^{m+r} \notin L$, ceea ce contrazice Lema de Pompare. Deci L nu poate fi limbaj regulat.

Exerciţii

1. Inversul oricărui cuvânt $w = a_1a_2 \dots a_n$ este cuvântul $w^R = a_n \dots a_2a_1$.
2. Să se demonstreze că dacă A şi B sunt limbaje regulate de cuvinte cu alfabetul Σ, atunci şi limbajele $A \cap B$, $\Sigma^* - A$ şi $A - B$ sunt regulate.
3. Să se demonstreze că următoarele limbaje nu sunt regulate. Pentru demonstrare se poate folosi Lema de Pompare şi faptul că reuniunea, intersecţia şi complementul limbajelor regulate sunt regulate.
   (a) $\{0^n1^m0^n \mid m, n \geq 0\}$.
   (b) $\{0^m1^n \mid m \neq n\}$.
   (c) $\{w \in \{0,1\}^* \mid w$ nu este palindrom$\}$. Dacă A este un limbaj atunci inversul lui A este limbajul $A^R = \{w^R \mid w \in A\}$. Să se arate că dacă A este limbaj regulat atunci şi A^R este limbaj regulat.
   (d) $\{wtw \mid w, t \in \{0,1\}^+\}$.
4. Fie $B_n = \{a^k \mid k$ este un multiplu de $n\}$. Să se arate că pentru orice $n \geq 1$, limbajul B_n este regulat.

Bibliografie

1. M. Sipser. Introduction to the Theory of Computation. Second Edition. Thomson Course Technology. 2006.