C14

miercuri, 24 ianuarie 2024 19:41

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 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 $e_1 \mid e_2$ (*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)

$(0 + 1)^*$ multimea tuturor sirurilor din 0 sau 1

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

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

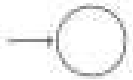
$(0 + 1)^*0$ ca mai sus, încheiat cu 0 (numere pare în binar)

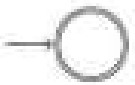
$1(0 + 1)^* + 0$ numere binare, fără zerouri inițiale inutile

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

$\emptyset$  nu are stare acceptoare

ε  starea inițială e acceptoare

sau 

nu consumă simbol

a 

acceptă simbolul a

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 formale, în general

Dorim să:

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

numai 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?

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)

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

$S \to NP\ VP \to NP\ verb\ NP \to NP\ verb\ noun$
$\to det\ NP\ verb\ noun \to det\ det\ NP\ verb\ noun$
$\to det\ det\ noun\ verb\ noun \to a\ good\ student\ reads\ books$

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

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:
$S \rightarrow aSa$

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]

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

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

reguli de forma:

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

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

Limbajele regulate sunt recunoscute de automate finite

2) gramatici *independente de context*

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

Limbajele independente de context sunt acceptate de automatele cu stivă

1) gramatici *dependente de context*

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

sau $S \rightarrow \varepsilon$ doar dacă S nu apare în dreapta

1) gramatici *dependente de context*
reguli: $\alpha A\beta \rightarrow \alpha\gamma\beta$ A e rescris dacă apare între α și β $\gamma \neq \epsilon$ (nevid),
sau $S \rightarrow \epsilon$ 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)

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

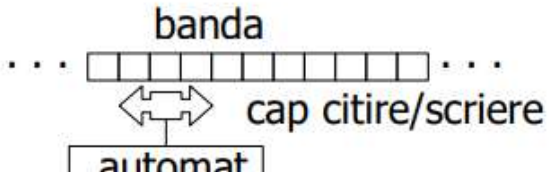
$\Rightarrow$ pt. a recunoaște limbajul
avem nevoie de o structură cu *memorie nelimitată*

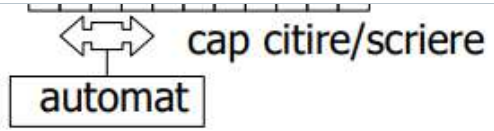
Conceptual, un calculator nu are nici el limită de memorie
(desi în realitate aceasta este, desigur, finită).

Mașina Turing = automat cu stări finite + memorie nelimitată

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.





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.

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

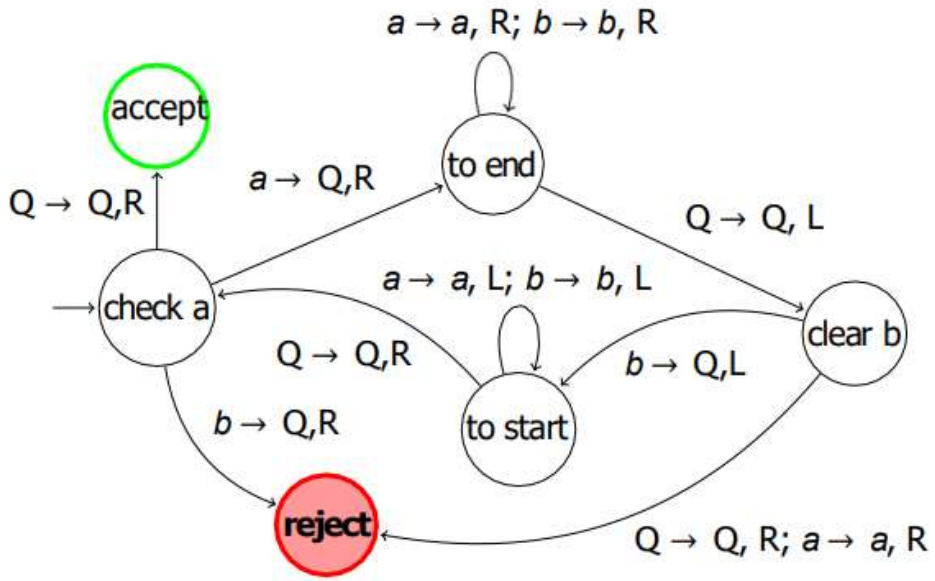
Execuția continuă până când se ajunge într-una din *stările finale*:
de **acceptare**: șirul este acceptat, face parte din limbaj
de **rejectare**: șirul este respins, nu face parte din limbaj

Obs.: mașinile Turing sunt deterministe!

Pentru fiecare combinație de stare non-finală q și simbol de bandă

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:

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 plus Q)

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
funcția de tranziție:

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
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)

$q_0 \in Q$: 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!

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

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

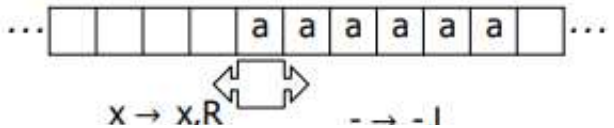
O mașină Turing este *de tip subrutină* dacă, în loc să accepte sau să respingă un șir de intrare, *efectuează anumite transformări* asupra acestuia

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

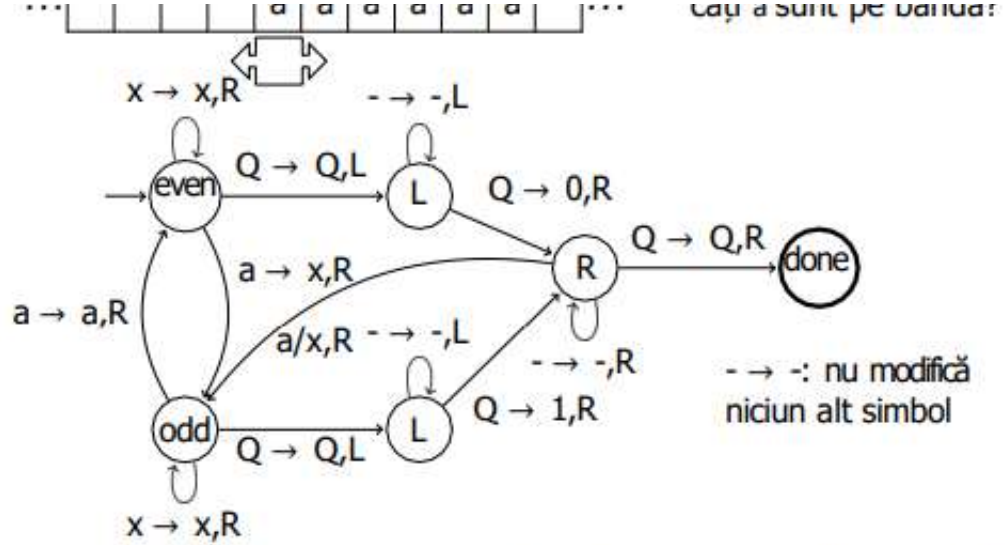
Astfel, pe bandă rezultă un nou șir, care poate fi prelucrat mai departe sau acceptat/respins de o altă mașină Turing.

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

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

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