Logică și structuri discrete

Gramatici

Marius Minea
marius@cs.upt.ro

http://cs.upt.ro/~marius/curs/lsd/

4 decembrie 2017

Recapitulare: Automate

Automatele pot descrie comportamentul unui sistem simplu.
din fiecare stare s, intrarea σ determină starea următoare s'

Un automat *recunoaște* un limbaj (face parte șirul din limbaj?)
ex. program cu comentarii încheiate corect

Traductoare: automate care produc ieșiri (în funcție de intrare)
putem *prelucra* limbaje (ex. elimina comentarii)

Automatele se pot reprezenta *eficient*:
tabel de tranziții:

st	a	b	c
0	1	0	0
1	1	2	0
2	1	0	3
3	1	0	0

$\Rightarrow$ putem determina eficient dacă un șir e acceptat
parcurgem șirul, la fiecare pas tabelul ne dă starea următoare

Recapitulare: Expresii regulate

Expresiile regulate reprezintă concis *limbaje regulate*, ca automatele

Sunt mai ușor de compus (din concatenare, alternativă, repetiție)

Putem să căutăm *tipare* în text:

identifică `[0-9]+` (șir nevid de cifre)

```
"images/d61_2016-07-06/logo-57x60.png" sizes="32x32"
```

sau o dimensiune (de imagine): `[0-9]+x[0-9]+`

```
"images/d61_2016-07-06/logo-57x60.png" sizes="32x32"
```

Pentru căutări, expresiile regulate se traduc în automate
(mai eficient de lucrat)

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 $\Rightarrow$ nu pot *număra* mai mult de atât

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

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

Exemplu: $a^n b^n$ nu e limbaj regulat

$a^n b^n$: *numărăm* câți a apar, verificăm că sunt la fel de mulți b.

Demonstrăm prin *reducere la absurd*.

Fie un automat determinist cu *n stări* care acceptă limbajul.
Fie șirul $a^n b^n$ (cu același n) și stările $s_0 \xrightarrow{a} s_1 \xrightarrow{a} \ldots \xrightarrow{a} s_n$.
mai departe, din s_n automatul acceptă b^n.

Din $n+1$ stări $s_0 \ldots s_n$, doar n pot fi diferite: există $i < j$ cu $s_i = s_j$
(șirul de stări are un *ciclu* pe parcurs)

Atunci $s_i \underbrace{\xrightarrow{a} \ldots \xrightarrow{a}}_{j-i \text{ simboluri } a} s_j$ e un ciclu ($s_j = s_i$). Îl repetăm încă o dată: mai adăugăm $j - i$ de a

Continuând apoi din s_j ajungem tot în s_n, dar am consumat a^{n+j-i}
$\Rightarrow$ automatul acceptă $a^{n+j-i} b^n$ (mai mulți a decât b), *contradicție*.

Lema de pompare pt. limbaje regulate (*pumping lemma*)

Exemplul anterior e un caz particular al unei proprietăți generale

Lema de pompare: În orice limbaj regulat
orice cuvânt suficient de lung conține un subșir ce poate fi repetat

$\exists p \in \mathbb{N}$ astfel ca orice cuvânt $w \in L$ cu $|w| \geq p$ (destul de lung) are forma $w = xyz$ cu

- $|y| \geq 1$ — se repetă un subșir nevid
- $|xy| \leq p$ — subșirul apare înainte de limita p
- $\forall k \geq 0 \; . \; xy^k z \in L$ — y poate fi repetat arbitrar între x și z

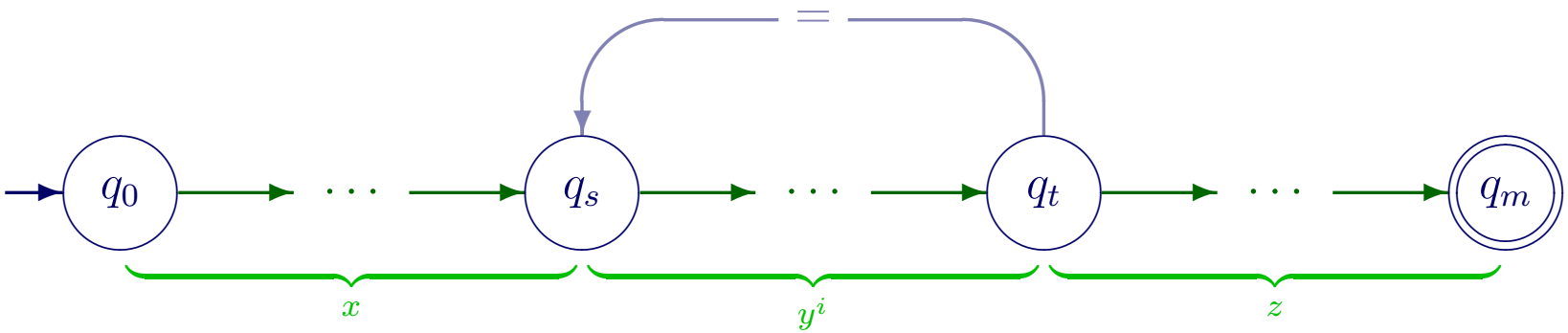
Demonstrația lemei de pompare

Fie un limbaj regulat și automatul care-l recunoaște.

Alegem *lungimea de pompare* p = numărul de stări din automat.
Fie șirul $a_1 a_2 ... a_m$ cu $m \geq p$, și stările parcurse de automat:

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots \xrightarrow{a_m} q_m$$

Avem $m + 1 > p$ stări, deci cel puțin o stare se repetă:
$q_s = q_t$ cu $s, t \leq m$



https://en.wikipedia.org/wiki/Pumping_lemma_for_regular_languages

Parcurgând șirul $y = a_{s+1} ... a_t$ din q_s ajungem înapoi în $q_t = q_s$
deci putem repeta șirul y, și $xy^i z$ e în limbajul acceptat, q.e.d.

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*.

Gramatica limbajului natural

Exemplu: propoziții în limba engleză (mult simplificat)

A good student reads books.

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

Am descris limbajul prin *reguli de producție* (de *rescriere*)
neterminale: simboluri care apar în stânga $\to$ (sunt înlocuite):
terminale: simboluri care apar numai în dreapta $\to$

O gramatică descrie un limbaj

Orice limbaj e descris prin *simbolurile* și *sintaxa* sa:
regulile prin care simboluri 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*

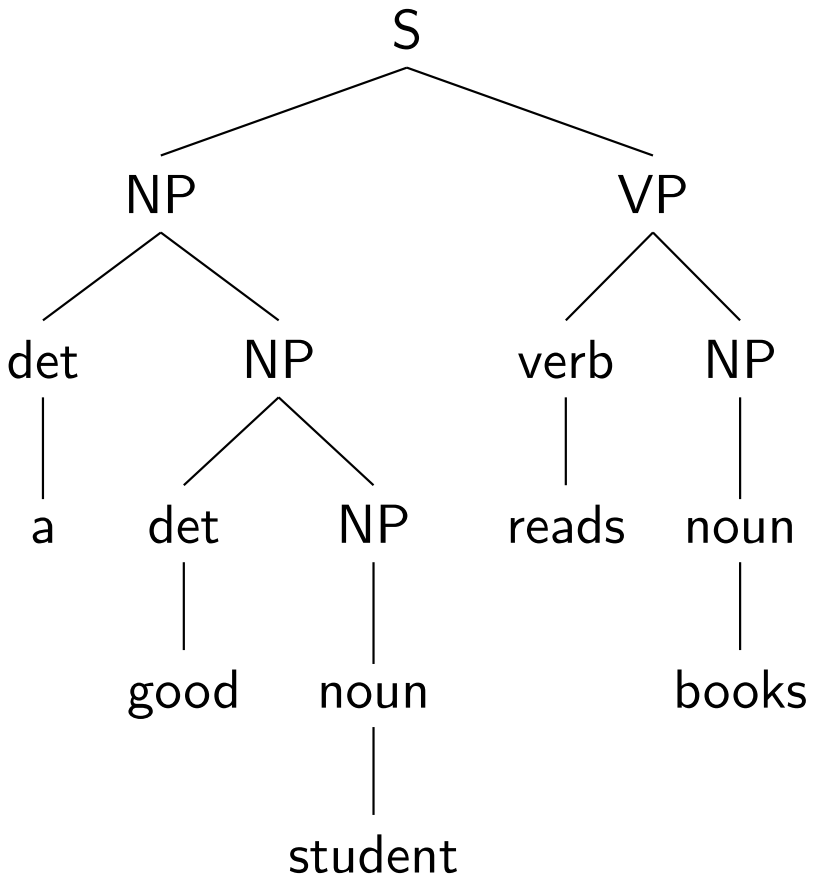
O *derivare* a unui șir dintr-o gramatică e o secvență de aplicări a *regulilor de producție* care transformă simbolul de start în șirul dat.
indicăm la fiecare pas și simbolul transformat

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

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

Arborele de derivare

Arborele de derivare e o reprezentare *ierarhică* a unei derivări, scriind partea dreaptă a fiecărei reguli sub partea stângă:



A good student reads books.

$S \rightarrow NP\ VP$

$NP \rightarrow det\ NP$ $VP \rightarrow verb\ NP$

$NP \rightarrow det\ NP$ $NP \rightarrow noun$

$NP \rightarrow noun$

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) $S \to (S)S$

O posibilă derivare: $S \xrightarrow{2} \underline{(S)S} \xrightarrow{2} (\underline{(S)S})S \xrightarrow{1} (()S)S \xrightarrow{1} (())S \xrightarrow{2} (())\underline{(S)S} \xrightarrow{1} (())()S \xrightarrow{1} (())()$

la fiecare pas, am colorat neterminalul transformat, am indicat regula folosită $\xrightarrow{1}$ sau $\xrightarrow{2}$ 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
$(\Sigma \cup N)^* N (\Sigma \cup N)^* \to (\Sigma \cup N)^*$
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)

Ierarhia Chomsky [după Noam Chomsky]

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
Conversie în automate: câte o stare pentru fiecare neterminal,
$A \to aB$ devine Ⓐ $\xrightarrow{a}$ Ⓑ, $A \to a$ devine Ⓐ $\xrightarrow{a}$ ⓞ (accept)

2) gramatici *independente de context*
reguli: $A \to \gamma$ stânga: neterminal; dreapta: șir arbitrar

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

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* }

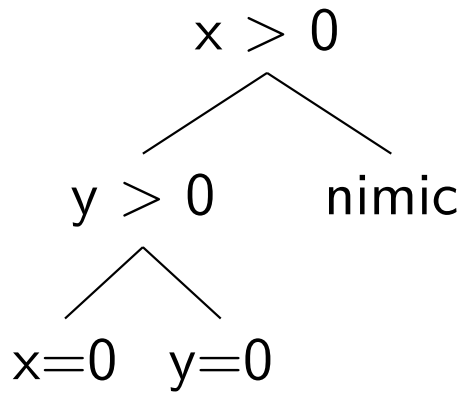
Problema: cu care `if` se potrivește `else` ?

```
if (x > 0) if (y > 0) x = 0; else y = 0;
```

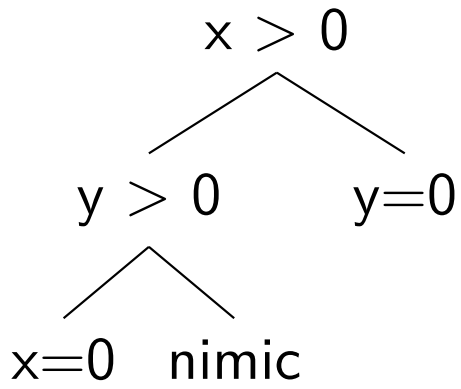
Gramatici ambigue

O gramatică e *ambiguă* dacă există șiruri cu mai mulți *arbori de derivare* (arbori sintactici)

```
if (x > 0) if (y > 0) x = 0; else y = 0;
```



interpretarea corectă



interpretare incorectă (trebuie eliminată)

Exemplu de dezambiguare

Pentru a dezambigua gramatica, trebuie rescrisă: distingem între
- un `if` *echilibrat*, care are `else`
- un `if` *neechilibrat*, fără `else`

Cum `else` e asociat cu cel mai apropiat `if`, ramura `then` e *întotdeauna echilibrată* (definim echilibrate restul de instrucțiuni).

Stmt ::= BalancedStmt | UnBalancedIf

BalancedStmt ::= ExpStmt | WhileStmt | Block | BalancedIf

ExpStmt ::= expr ;

WhileStmt ::= `while` (expr) Stmt

Block ::= { Stmt* }

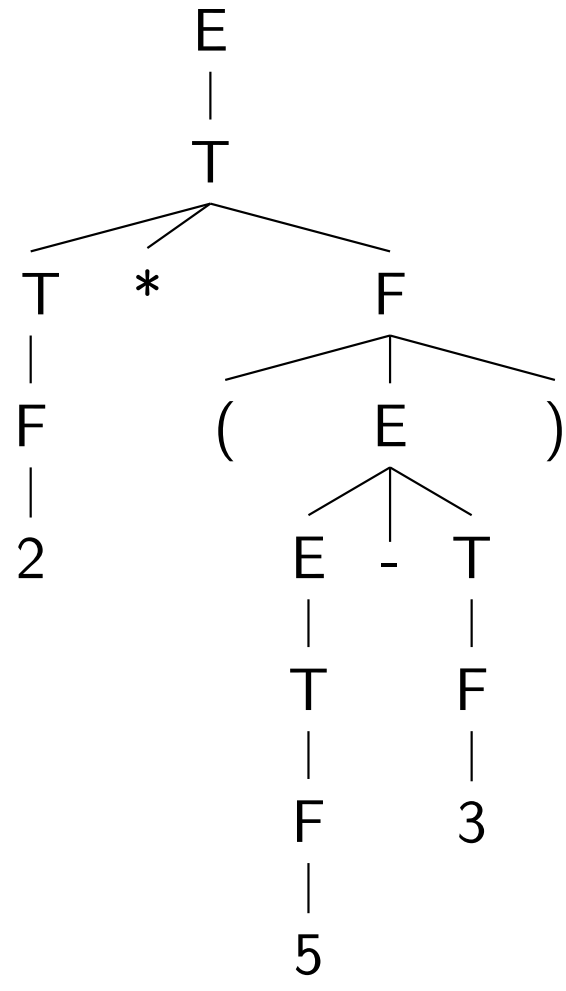
BalancedIf ::= `if` (expr) BalancedStmt `else` Stmt

UnBalancedIf ::= `if` (expr) Stmt

Expresii aritmetice (1)

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)`

Expresii aritmetice (2)

În această scriere, E apare primul în stânga producțiilor lui E
recursivitate *la stânga* $\Rightarrow$ nu putem implementa direct
(nu știm când să oprim apelul recursiv)

E ::= T | E + T | E - T

T ::= F | T * F | T / F

F ::= num | (E)

v3) Eliminăm *recursivitatea la stânga* $\Rightarrow$ putem scrie direct cod

E ::= T RestE

RestE ::= ϵ | + T RestE | - T RestE

T ::= F RestT

RestT ::= ϵ | * F RestT | / F RestT

F ::= num | (E)

Expresii prefix și postfix (1)

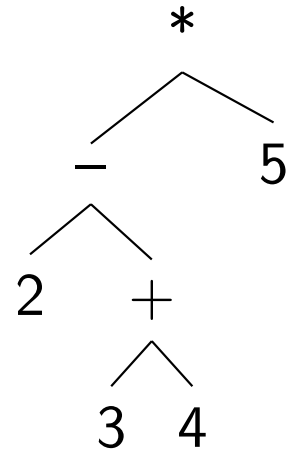
nu necesită paranteze, subexpresiile reies din structură

Expresii *prefix*: operatorul *înaintea* operanzilor
implicit: la fel și în *sub*expresii

E ::= num | Op E E

Op ::= + | - | * | /

$$\underbrace{* \underbrace{- \; 2 \; \underbrace{+ \; 3 \; 4}} \; 5}_{} \quad = \quad \texttt{(2 - (3 + 4)) * 5}$$



Putem scrie direct cod pornind de la această gramatică!

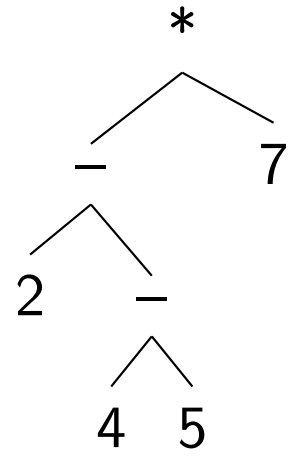
Expresii prefix și postfix (2)

Expresii *postfix*: operatorul *după* operanzi, la fel în subexpresii

E ::= num | E E Op

Op ::= + | - | * | /

$\underbrace{2\ \underbrace{4\ 5\ -}\ -}\ 7\ *$ = (2 - (4 - 5)) * 7



Putem elimina recursivitatea la stânga:

E ::= num RestE

RestE ::= ϵ | E Op RestE

Scrierile se pot obține prin traversarea arborelui expresiei:

- în *preordine*: întâi operatorul, apoi subexpresiile (în același fel)
- în *postordine*: întâi subexpresiile (în același fel), apoi operatorul