------------------------------CURS 1-------------------------

Limbaje regulare:

- Gramatici regulare
- Automate finite
- Expresii regulare

= > Analiza sintactica

Limbaje independente de context:

- Gramatici independente de context, automate push-down
- Gramatici speciale: LL(k), LR(k)

= > Analiza sintactica

Gramatici de atribute:

= > Analiza semantica si Generare de cod intermediar.

Alfabet = o multime finita si nevida de elemente numite simboluri.

Secventa peste alfabet = o succesiune finita de simboluri din alfabet
Subsecventa = o succesiune de simboluri consecutive dintr-o secventa
Lungimea unei secvente = nr de simboluri din care este formata acea secventa(notatie |…|, ex. |abc| = 3)(secv vida = eps)

Prefix = o subsecventa care - fie este vida; - fie incepe cu primul simbol al secventei date
Sufix = o subsecventa care - fie este vida; - fie se termina cu ultimul simbol al secventei date.

Operatia *:
alfabet^* - inchiderea alfabetului
- multimea tuturor secventelor ce se pot obtine folosind secvente din alfabet

Limbaj = L - limbaj peste alfabet, daca L e inclus in alfabet^*
Cuvant = cuvant al unui limbaj - un element al limbajului

Gramatica independenta de context este un cvadruplu G = (N,SUMA, P,S)
N - este un alfabet de simboluti neterminale
SUMA - este un alfabet de simboluri terminale
N intersectat cu SUMA = multime vida
P - multimea regulilor de productie
S apartine lui N - simbolul de start

Reg. de productie sunt de forma A->alfa, Aapartine N, alfra apartine la (N reunit cu SUMA)*

Gramatica regulara:
Reg. de productie sunt de forma:
A -> aB
A -> b
A,B sunt neterminale si a, b terminale
Caz special S -> eps, poate apartine regulilor de productie. In acest caz, S nu apare in membrul drept al nici unei reguli de productie.

BNF(Backus Naur Form)

- Primitive(terminale)
- Variabile metalingvistice : intre paranteze unghiulare
- Conective metalingvistice: | , ::=

EBNF(Extended BNF)

- Primitive(terminale) - intre ghilimele
- Variab metalingvistice
- Conective metalingvistice: =, |, [optional], {de ori cate ori}
- Punctul este folosit ca sfarsit de regula

Analiza lexicala

- Identificatori
- Constate(literali)
- Cuvinte cheie(cuvinte rezervate)
- Operatori aritmetici, relationali,...

- Separatori

Cuvinte cheie - simboluri sintactice
Cuvinte rezervate = nu pot fi folosite ca identificatori

Date: fisier text
Rezultate: FIP, TS sau mesaj de eroare
TS - Tabela de simboluri (informatii despre identificatori si constante)
FIP - forma interna a programului (cod atom lexical, Pozitia in TS unde este cazul)
Analiza lexicala exemplu partial PAG 32

------------------------------CURS 2--------------------------

Automat finit este un ansamblu

$M = (Q, \Sigma, \delta, q_0, F)$:

Q – alfabetul starilor

Σ – alfabet de intrare

$\delta : Q x \Sigma \rightarrow \mathcal{P}(Q)$ functie de tranzitie

$q_0 \in Q$ - stare initialã

$F \subseteq Q$ multimea stãrilor finale

Cred ca a 3 a chestie e multimea tranzitiilor ????????

Limbaj acceptat de automat

$L(M) = \{w \mid w \in \Sigma^*, (q_0, w) \vdash^* (q_f, \varepsilon), q_f \in F\}$

Automate echivalente

M_1 echivalent cu M_2 daca: $L(M_1) = L(M_2)$

Automat finit determinist (AFD)

- Intr-o stare data si cu o intrare specificata, exista o singura tranzitie valida posibila
- Pentru fiecare combinatie de stare si intrare, exista o tranzitie unica.

Stare1 (Inițială) --[aeiou]--> Stare2 --[a-zA-Z]--> Stare3 (Finală)

Automat finit nedeterminist (AFN)

- Intr-o stare cu o intrare specificata, exista posibiltatea de a alege in mai multe tranzitii valide
- Nu este mereu clara tranzitia exacta pe care automatul o va face

Stare1 (Inițială) --[a]--> Stare2 --[b]--> Stare2 (Se poate repeta) --[epsilon]--> Stare3 (Finală)
(cumva ca are bucla, e nedeterminist)

Nu prea inteleg de la pag 48 ce I acolo

Echivalenta dintre AFD si AFN

Teorema:

$\forall\ M_1 - AFN \quad \exists\ M_2 - AFD \text{ echivalent}$

Nici la 50 nu inteleg mare lucru

AF - stari care nu contribuie la acceptarea unui cuvand:

- Stare neproductiva
- Stare inaccesibila
- Stare productiva
- Stare accesibila

stare productiva: $q \in Q$ a.i.

$$\exists\, w \in \Sigma^* \text{ si } q_f \in F \text{ a.i. } (q,w) \vdash^* (q_f,\varepsilon)$$

stare accesibila: $q \in Q$ a.i.

$$\exists\, w \in \Sigma^* \text{ a.i. } (q_0,w) \vdash^* (q,\varepsilon)$$

Def. O stare q este productiva daca exista un drum care duce de la starea q la o stare finala qf folosind simbolurile din w si apoi eps.
Def. O stare q este accesibila daca exista un drum care incepe de la starea initiala q0 si ajunge la starea q, folosing simbolurile din w.

TEOREME : oricare ar fi M1 -AF exista M2 - AF fara stari neproductive/inaccesibile ecxhivaleste, Constructie la PAG 55-56
Ceva metode de determinare a AFD echivalent pt un AFN dat, nu stiu daca trebe sa stiu astea la pag 57

Vrem minimizarea automatelor finite, sa avem automat determinist cu nr minim de stari, nu contine stari inaccesibile si neproductive, nu contine perechi de stari echivalente.

Minimizarea AFD:

- Automat cu nr minim de stari
- Stari diferentiate
- Stari k diferentiate
- Stari echivalente
- Stari k echivalente

Automatul redus:
Fie M1 - un automat finit oarecare

- Determinam AFD echivalent
- Eliminam starile inaccesibile si neproductive
- Determinam AFD echivalent complet definit

Fie $M = (Q, \Sigma, \delta, q_0, F)$ automatul rezultat.

- Determinam relatia $\equiv$ (stari echivalente, clase de echivalenta)
- Pe baza relatiei $\equiv$ determinam automatul:

  $M_\equiv = (Q/\equiv, \Sigma, \delta_\equiv, [q_0], F_\equiv)$

  $Q/\equiv$ - multimea claselor de echivalenta

  $\delta_\equiv([q],a) = [\delta(q,a)]$

  $F_\equiv = \{[q] \mid q \in F\}$

TEOREME:

1. Automatul redus are numar minim de stari dintre toate AFD echivalente
2. Oricare ar fi M1 - AF, exista M2 - automat redus echivalent

Stari diferentiate

q1 si q2 sunt stari diferentiate de x apartine sigma^*

daca $\exists\ q_f \in F$ a.i. $(q_1,x) \vdash^* (q_f,\varepsilon)$

si nu exista nici un $q \in F$ a.i. $(q_2,x) \vdash^* (q,\varepsilon)$

sau

daca $\exists\ q_f \in F$ a.i. $(q_2,x) \vdash^* (q_f,\varepsilon)$

si nu exista nici un $q \in F$ a.i. $(q_1,x) \vdash^* (q,\varepsilon)$

x(de mai sus) diferentiaza pe q1 si q2

Mai este o explicatie la PAG 65

La PAG 67 ceva proprietati ale relatie de k-echivalenta

--------------------------------CURS 3-----------------------------

Relatii de derivare ????????

- Derivare directa

  g⇒d dacaˇ și numai daca g=g1ag2 și d=g1bg2, unde (a→b)∈P

unde P reprezintă mulțimea producțiilor dintr-o gramatică.

- K-derivare

  (o succesiune de k derivãri directe)

- \+ derivare

  dacã $ k>0 a.i. cele 2 secvente sã fie într-o relatie de "k derivare"

- * derivare

  daca fie cele 2 secvente sunt egale, fie intre ele exista o relatie de +derivare

Tipuri de gramatici

Gramatica monotona: - $\forall a \to b \in P$: $|a| \le |b|$, unde $a,b \in (N \cup S)$

Această condiție specifică că lungimea șirului din partea stângă a regulii a este mai mică sau egală cu lungimea șirului din partea dreaptă a regulii b, pentru orice regulă $a \to b$ din gramatică.

- caz special: $S \to$ eps poate sa apartina lui P. In acest caz S nu apare în membrul drept al nici unei reguli de productie.

Gramatica dependenta de context:

- reguli de productie de forma: $aAb \to agb$, unde $A \in N, a,b,g \in (N \cup S)^*$, și g diferit de ε .
  Aceasta indică că o producție de tip GDC se aplică doar dacă A apare între două șiruri a și b, iar aceasta poate fi înlocuită cu g.
- caz special: $S \to$ eps poate sa apartina lui P. In acest caz S nu apare în membrul drept al nici unei reguli de productie.

Gramatica regulara:

- Reguli de productie de forma:

  $A \to aB$

  $A \to b$

  unde, $A,B \in N$ (nonterminale), iar $a,b \in \Sigma$ (terminale).
- caz special: $S \to$ eps poate apartine lui P In acest caz S nu apare în membrul drept al nici unei reguli de productie.

Gramatica independenta de context:

- Reguli de productie de forma:

  $A \to a$, unde $A \in N$ (nonterminal) și $a \in (N \cup \Sigma)^*$

Clasificarea Chomsky

- Gramatici de tip 0: nici o restrictie (suplimentara) referitoare la forma regulilor de productie
- Gramatici de tip 1:

  Dependente de context <=> gramatici monotone
- Gramatici de tip 2: gramatici independente de context
- Gramatici de tip 3: gramatici regulare

Limbaje regulare = limbaj generat de o gramatica regulara

Multimi regulare:

1. Φ este o m. reg. peste Σ
2. $\{\varepsilon\}$
3. $\{a\}$ daca: $a \in \Sigma$
4. $R \cup S$ daca R,S – multimi regulare peste Σ +
5. RS daca R,S – multimi regulare peste Σ
6. R^* daca R – multime regulara peste Σ
7. Orice alta multime regulara se obtine aplicand de un numar finit de ori reg. 1-6

Expresii regulare, sunt la fel ca mai sus, doar ca cu litere mici si fara multime, si la 4 e r+s.
Expresii regulare echivalente: multimile regulare reprezentate de acestea sunt egale.

Expresii regulare = secv. Obtinute prin concatenarea sumbolurilor din

$$\Sigma \cup \{\Phi, \varepsilon, +, *, (\ ,\ )\}$$

Multimile regulare asociate expresiilor regulare sunt limbaje regulare.

Deci: **Orice expresie regulara peste Σ, descrie un limbaj regular peste Σ.**

Proprietati de inchidere ale limbajelor regulare
TEOREMA: Daca L1,L2 sunt limbaje regulare peste alfabetul Σ atunci: L1 reunit cu L2, L1 intersectat cu L2, L1L2, L1*, complement(L1) sunt limbaje regulare peste alfabetul Σ

Lema de pompare pt limbaje regulare:
Această lema afirmă că pentru orice limbaj regular L, există un număr p, astfel încât orice șir w din limbajul
L și cu lungime cel puțin p poate fi descompus în trei părți, x,y,z astfel încât:
$w=xyz$, $|xy|\le p$, $|y|>0$, și $xy^iz \in L$ pentru orice $i \ge 0$

Adică, șirul y poate fi "pompat" de oricat de multe ori și șirul rezultat va fi tot în limbajul L.

- condiția $0<|y|$ adaugă o precizie suplimentară și face lema mai robustă în ceea ce privește demonstrația că un limbaj este regular.

Observatii:

- Lema da o conditie necesara, dar nu suficienta
- Daca un limbaj satisface conditiile lemei nu inseamna ca este regular
- Folosim negatia lemei de pompare pt a determina daca un limbaj nu este regular

------------------------------CURS 4--------------------------------

Automate finite cu eps-miscari

M = (Q, Σ, CV, q0,F): CV e functia de tranzitie

Ideea: putem avea si eps-tranzitii(automate cu eps-tranzitii)

TEOREMA: Pentru orice automat finit cu eps-miscari, exista un automat finit echivalent

Echivalenta dintre expresiile regulare si limbajele acceptate de AF

TEOREMA: Daca r este o expresie regulara, atunci exista un AF care accepta multimea secventelor reprezentate de aceasta expresie (multimea regulara). Si reciproc.

Echivalenta: - constructia automatului echivalent pentru fiecare dintre constructiile de mai sus

- constructia expresiei regulare ce descrie limbajul acceptat de un automat

Is niste exercitii la PAG 109

Proprietati: espresii regulare echivalente

(reuniune si concaten.)

$$r + s = s + r$$
$$(r+s)+t = r+(s+t)$$
$$(rs)t = r(st)$$
$$(r+s)t = rt + st$$
$$r(s+t) = rs + rt$$

(utilizarea lui Φ si ε)

$$\Phi + r = r + \Phi = r$$
$$\varepsilon r = r \varepsilon = r$$
$$\Phi r = r \Phi = \Phi$$

$$\Phi^* = \varepsilon$$
$$r^* + \varepsilon = \varepsilon + r^* = r^*$$
$$(\varepsilon + r)^* = r^*$$
$$(r^*)^* = r^*$$
$$(r^*s^*)^* = (r+s)^*$$

EXPRESII REGULARE

x	se *potriveste* cu caracterul ***x***
r*	zero sau mai multi ***r***
r+	unul sau mai multi ***r***
r?	zero sau un ***r*** (optional)
rs	concatenarea lui ***r*** cu ***s***
r|s	sau ***r*** sau ***s***
r{2,5}	doi, trei, patru sau cinci ***r***
r{2,}	doi sau mai multi r
r{4}	exact patru ***r***
	"character class"
[xyz]	se potriveste cu oricare dintre 'x', 'y', 'z'.
[abj-oZ]	se potriveste cu oricare dintre 'a', 'b', orice de la 'j' la 'o', sau 'Z'
[^A-Z]	*"negated character class"*
	se potriveste cu orice caracter cu exceptia celor specificate
.	orice caracter cu exceptia newline
{name}	se *potriveste* cu definitia "name"

LEX/FLEX PAG 118

--------------------------------CURS 5--------------------------------

Gramatici independente de context (GIC)

Derivari: - derivare de stanga: o derivare directa in care se inlocuieste cel mai din stanga neterminal

- Derivare de dreapta: o derivare directa in care se inlocuieste cel mmai din dreapta neterminal

Analiza sintactica

Analiza sintactica: pt cuvantul w, succesiunea de derivari directe:
$S \Rightarrow a_1 \Rightarrow a_2 \Rightarrow \ldots \Rightarrow a_n = w$
Analiza sintactica descendenta: dupa aceasta succesiune de derivari directe se obtine pornind de la S si terminand cu w
Analiza sintactica ascendenta: daca aceasta succesiune de derivari directe se obtin pornind de la w si terminand cu S

Arbore de derivare

Fie $G=(N,\Sigma,P,S)$ o gramatica independenta de context. Numim **arbore de derivare** sau **arbore de analiza sintactica** un arbore cu radacina, ordonat, cu urmatoarele proprietati:

1. Orice nod interior - o eticheta din N (adica un neterminal) ;
2. Orice nod frunza - o eticheta din $\Sigma \cup \{eps\}$ (adica un terminal) ;
3. Eticheta radacini este S (start-ul) ;
4. Daca un nod are eticheta A, iar nodurile succesoare acestuia, in ordine de la stanga la dreapta sunt etichetate cu $X_1,X_2,..,X_n$, atunci $A \rightarrow X_1X_2\ldots X_n$ trebuie sa fie o productie din P.

Frontiera(frontul): nodurile terminale, in ordine de la stanga la dreapta
Etichetele lor formeaza o secventa peste Σ^*
Obs: denumirea de frontiera se foloseste si pentru a denumi succesiunea etichetelor nodurilor terminale

TEOREMA: Fie G=(N,Σ,P,S) o gramatica independenta de context. Un cuvant w peste alfabetul Σ, deci din Σ(, apartine limbajului generat de G, adica w incluc in L(G) daca si numai daca w este frontul unui arbore de analiza sintactica.

O gramaticã G = (N, Σ, P, S) independentã de context este **ambiguã** dacã si numai dacã existã cel putin un cuvânt w care admite doi arbori de derivare distincti; în caz contrar gramatica este **neambiguã**.
<=> exista 2 analize sintactice care folosesc numai derivari de stanga, diferite
<=> exista 2 analize sintactice care folosesc numai derivari de dreapta, diferite

Gramatica G = (N, Σ, P, S) este **eps-independentã** daca:
a) dacã eps nu apartine lui L(G) atunci G nu are eps-productii
b) dacã eps apartine lui L(G) atunci avem o singurã productie S->eps iar celelalte productii nu-l contin în membrul drept pe S

TEOREMA: oricare ar fi G = (N, Σ, P, S), exista G' = (N', Σ', P', S') echivalenta, eps-independenta

Redenumire: reg de prod de forma A->B
Ciclu: o * derivare de forma A=>*B

Recursivitate:

- Reg de prod recursiva la stanga: **A -> Aa**
- Reg de prod recursiva la dreapta: **A -> aA**
- Reg de prod recursiva: **A -> aAb**
- **Gramatica recursiva la stanga:** are cel putin un neterminal recursiv la stanga
- **Gramatica recursiva la dreapta**: are cel putin un neterminal recursiv la dreapta

Forma normala Chomsky
O gramatica independenta de context este in **forma normala Chomsky**(FNC) daca orice regula de productie este una din formele:

a. A -> BC   A, B, C sunt neterminale
b. A -> a   A- neterminal, a - terminal

Si un caz special S -> eps, si S nu apare in membrul drept al nici unei reguli de prod.
TEOREMA: Oricare ar fi G=(N, S, P, S) o gramaticã independentã de context, întotdeauna existã o gramaticã în forma normalã Chomsky G', astfel încât L(G) = L(G').

Forma normala Greibach
O gramatica este un forma normala Greibach(FNG) daca P are productii numai de forma:
A -> aALFA, ALFA e neterminal, a - terminal, ALFA aprtine N*
+ caz special S -> eps
TEOREMA: Oricare ar fi G = (N, S, P, S) o gramaticã independentã de context, întotdeauna existã o gramaticã în forma normalã Greibach, astfel încât L(G)=L(G').

Simplificarea GIC

- Simbol neproductiv - A(neterminam) este neproductiv daca nu exista nicio derivare de forma

  A =>* x, x - termial*
- Simbol inaccesibil - X apartine N U Σ, este un simbol inaccesibil daca nu exista nici o * derivare:

  S =>* aXb, a,b e ori terminal ori neterminal
- Simbol neutilizabil - un simbol este neutilizabil daca el este fie inaccesibil, fie neproductiv.

Obs: Fie G= (N, S, P, S) o gramatica independenta de context:
• Fie un simbol neterminal A al gramaticii G. Daca nu exista o regula de productie A -> a in P atunci A este neproductiv
• Fie un simbol terminal a al gramaticii G. Daca nu exista o regula de productie de forma B -> alfa a beta in P atunci a este inaccesibil

Eliminarea eps-productiilor

1. Construim multimea N_ε care are ca elemente acele neterminale care prin derivare conduc la ε adică :

- $N_\varepsilon = \{A \mid A \in N, A \Rightarrow^* \varepsilon\}$

  alg. ≈ determinarea simb. productive

2. Determinam noile reguli de productie

- astfel incat productiile de forma $A \rightarrow \varepsilon$ se elimina
- dar, daca $\varepsilon \in L(G)$, atunci $\exists\ S \rightarrow \varepsilon$ si S nu apare în membrul drept al nici unei productii

Determinam noile reguli de productie PAG 155

Eliminarea redenumirilor

PP. G – ε-independenta (daca nu , luam gr.echiv. ε-ind.)

Pentru fiecare **A∈ N**

se elimina redenumirile de forma $A \rightarrow B$ ($\forall B \in N$)

- construieste multimile $N_A = \{B \mid A \Rightarrow^* B\}$;

  (≈ det. simb. accesibile)

- determinam noile reguli de productie

Determinam noile reguli de productie - algoritm PAG 160

Gramatica proprie - este o gramatica fara simb. neutilizabile, eps-independenta, fara cicluri.

VEZI ASTA CUM O FACI

!!!!!!!!!!!Eliminarea reg. prod. recursive la stanga PAG 166

Lema de pompare pentru limbaje independente de context

Fie L un limbaj independent de context. Exista atunci o constanta p dependenta numai de L astfel ca daca z apartine la L si lungimea lui z e mai mare sau egal cu p, atunci avem descompunerea z = uvwxy cu proprietatile:

a. Lungimea lui vx >= 1
b. Lungimea lui vwx <= p
c. uv^iwx^iy apartine L, oricare ar fi I apartine lui N

TEOREMA: Dacã L1 si L2 sunt limbaje independente de context atunci: L1U L2 , L1L2 , L1 * sunt limbaje independente de context.
L1∩L2, compl(L1) - nu sunt neaparat l.i.c

IS CEVA EXERCITII LA PAG 177

---------------------------------CURS 6------------------------------

Automate Push Down (APD)
PAG 181

Un automat push down este un ansamblu de:

$M = (Q, \Sigma, \Gamma, \delta, q_o, Z_o, F)$, unde:

Q	alfabetul starilor;
Σ	alfabetul de intrare;
Γ	alfabetul memoriei stivă; ;
$q_o \in Q$	stare iniţială;
$Z_o \in \Gamma$	simbolul de start al memoriei stivă;
$F \subseteq Q$	mulţimea stărilor finale;

$\delta : Q x(\Sigma \cup \{\varepsilon\}) x\Gamma \rightarrow \mathcal{P}(Q x\Gamma^*)$ funcţia de tranziţie

Functiile de tranzitii is tuple:
(q,a,x): unde q -stare curenta, a - inputul care se proceseaza, x - top of stack
-> {(qn,alfa), ...}: qn - stare noua, alfa - top of stack care se inlocuieste

Configuratie

(q,x,a) apartine Q x S* x alfabetyl de sus, spanzuratoarea

- Automatul se gaseste in starea q, pe banda de intrare urmeaza sa se citeasca(accepte) secventa x, iar in memoria stiva aven secventa a.
- Configuratie initiala: (q0,w,Z0)

Secventa acceptata de automat:

- Dupa criteriul tivei vide: Leps
- Dupa criteriul starii finale: Lf

TEOREME:

Fie automatul push-down M. Exista intotdeauna un automat push-down M' astfel incat
Leps(M') = Lf(M) si reciproc.
Oricare ar fi G - o gramatica independenta de context, exista un automat push-down M astfel incat Leps(M) = L(G) si reciproc.

G.I.C => APD echivalent

Fie F - gram independenta de context. Cine este M - APD astfel incat L(G) = Leps(M)?
M=({q}, Σ, N U Σ, sigma, q, S, semn ciudat)

1. Daca (A-> a) apartine lui P, atunci (q,alfa) apartine sigma(q,eps,A)
2. Sigma(q,a,a) = {{q,eps}} oricare ar fi a apartine alfabetului de intrare
3. Sigma(..,..,..) = semn ciudat in alte cazuri

Determinism:

$$\mathbf{M = (Q, \Sigma, \Gamma, \delta, q_o, Z_o, F)}$$

este ***determinist*** ddacă:

$$\forall \mathbf{Z} \in \Gamma, \forall \mathbf{q} \in \mathbf{Q}\ , \forall \mathbf{a} \in \Sigma$$

$$1)\ |\delta(q,\varepsilon,Z)| = 0 \quad \text{si} \quad |\delta(q,a,Z)| <= 1$$

$$2)\ |\delta(q,\varepsilon,Z)| = 1 \quad \text{si} \quad |\delta(q,a,Z)| = 0$$

In caz contrar, automatul nu este determinist.

Multimea limbajelor acceptate de APD nedeterministe e strict mai larga decat multimea limbajelor acceptate de APD deterministe.

---------------------------------CURS 7--------------------------------

Analizatorul descendent cu reveniri

Configuratie: (s,I,alfa,beta)

S - starea automatului

- q: stare normala
- r: stare de revenire
- t: stare de terminare
- e: stare de eroare

I - pozitia(urmatoare) in secventa de intrare

Alfa - stiva de lucru(de istorie): istoria regulilor de productie aplicate.

Beta - stiva de intrare: partea inca neprelucrata

Configuratie initiala: (q,1,eps, S)

Tranzitii:

- expandare: $(q,i,\alpha,A\beta) \vdash (q, i, \alpha A_1, \gamma_1\beta)$
- avans: $(q,i,\alpha,a_i\beta) \vdash (q, i+1, \alpha a_i, \beta)$
- insucces de moment: $(q,i,\alpha,a\beta) \vdash (r,i,\alpha,a\beta)$, $a<>a_i$
  $(q,i,\alpha,\varepsilon) \vdash (r,i,\alpha,\varepsilon)$, $i \neq n+1$
- succes: $(q,n+1,\alpha,\varepsilon) \vdash (t,n+1,\alpha,\varepsilon)$
- revenire: $(r,i,\alpha a,\beta) \vdash (r,i-1,\alpha,a\beta)$
- alta incercare: $(r,i,\alpha A_j,\gamma_j\beta) \vdash \dots$
  daca $\exists\ A_{j+1}\to\gamma_{j+1}$ $\vdash (q,i,\alpha A_{j+1},\gamma_{j+1}\beta)$
  altfel daca $i=1, A=S$ $\vdash (e,i,\alpha,A\beta)$
  $\alpha=\varepsilon$, $\beta=\varepsilon$
  altfel $\vdash (r,i,\alpha,A\beta)$

Observatie: Se numeroteaza regulile de productie cu acelasi membru stang.

Striva de lucru contine informatiile referitoare la regulile de productie aplicate.

--------------------------------CURS 8--------------------------------

Analiza sintactica descendenta.
Gramatici si analiza LL(k)

Firstk si Followk n am prea folosit

Gramatici LL(k)

- Analizatoare LL(k)
- Analiza sintactica descendenta
- Secv. De intrare este citita de la stanga la dreapta
- Se folosesc derivari de stanga

PAG 222

Gramatici LL(1)
TEOREMA: G - este de tip LL(1) ddaca

$\forall A \in N$ A - neterminal

$A \rightarrow \alpha_1 | \alpha_2 | \alpha_3 | \dots | \alpha_m$ sunt diferite alternative ale productiilor neterminalului A

- $FIRST_1(\alpha_i) \cap FIRST_1(\alpha_j) = \Phi$, i<>j
- daca $\exists$ i a.i. $\alpha_i \overset{*}{\Rightarrow} \varepsilon$ atunci: $\varepsilon \in FIRST_1(\alpha_i)$

$FIRST_1(\alpha_j) \cap FOLLOW_1(A) = \Phi$, i<>j

Analiza sintactica LL(1)

- Se construieste tabelul de analiza LL(1)

$M(X,a) =$

- (α,i) daca $X \rightarrow \alpha \in P$, $a \in FIRST_1(\alpha)$
  $X \rightarrow \alpha$ – a i-a regula de productie
- (α,i) daca $\varepsilon \in FIRST_1(\alpha)$, $a \in FOLLOW_1(X)$
  $X \rightarrow \alpha$ – a i-a regula de productie
- pop $X = a$
- acc $X = \$, a = \$$
- err in toate celelalte cazuri

- Analizorul sintactic LL(1)

Automat: (α,β,Π)

- banda de intrare: α (stiva de intrare)
- stiva β (stiva de lucru)
- banda de iesire Π => sirul regulilor de productie

config. initiala: $(w\$, S\$, \varepsilon)$

config. finala: $(\$, \$, \Pi)$

tranzitii

- push $(\mathbf{a}x\$, \mathbf{A}\beta, \Pi) \vdash (ax\$, \alpha\beta, \Pi\mathbf{i})$ dc.: $M(A,a) = (\alpha,i)$
- pop $(\mathbf{a}x\$, \mathbf{a}\beta, \Pi) \vdash (x\$, \beta, \Pi)$
- acc $(\$, \$, \Pi) \vdash$ acc
- err in celelalte cazuri

Se adauga \$ la orice sfarsit de cuvant

Obs: O gramatica este de tip LL(1) ddaca tabelul de analiza nu contine conflicte (nu exista mai mult de o valoare intr-o celula din tabel)

----------------------------CURS 9-10-------------------------------

Gramatici LR(k)

- Analizatoare LR(k)
- Analiza sintactica ascendenta
- Secv de intrare este citita de la stanga la dreapta
- Se folosesc derivari de dreapta

Metoda: deplasare - reducere(shift-reduce)

O gramatica $G = (N, \Sigma, P, S)$
este de tip LR(k) pentru k>=0
ddaca din:

- $S' \overset{*}{=>}_{dr} \alpha A w =>_{dr} \alpha \beta w$
- $S' \overset{*}{=>}_{dr} \gamma B x =>_{dr} \alpha \beta y$
- $FIRST_k(w) = FIRST_k(y)$

rezulta ca:

- $A = B$
- $x = y$
- $\alpha = \gamma$

Pasi in analiza LR(k):

- Gramatica imbogatita (adaugam S' -> S)
- Constructia colectiei canonice (facem desenul ala cu multe ramuri)
- Constructia tabelului de analiza
- Analiza -> automat

La multimea cuvintelor de analizat se adauga la sfarsit $(marcator de sfarsit de cuvant)

Colectia canonica LR(k)

C = {Ii - elemente de analiza pentru un prefix viabil }

- In I0 avem un prim element de analiza
- Am cel putin un element in Ij => adaug altele: functia Closure
- Am o multime Ij => construiesc multimile goto(Ii,X)

Ii corespunde unei stari a automatului

E - multimea elementelor de analiza

LR(0)

- Se adauga S' nou simbol de start
- In I0 avem [S' -> .S] ...

Functia Closure

Dacă avem un element de analiză de forma [[A→a.Bb], unde A→a.Bb este un element de producție cu un punct (simbolul punctat), atunci Closure adaugă toate elementele de producție care pot începe cu B în acest context.

Functia Goto

Funcția Goto în contextul LR(0) este o operație care generează o nouă configurație de analiză dintr-o configurație existentă, mutând punctul (simbolul punctat) peste un simbol X.

Tabel de analiza LR(0)

$T(I_i$, actiune) =

- s (shift, deplasare)

  daca: $[A \to \alpha.\beta] \in I_i$, $\beta <> \varepsilon$

  si: $T(I_i, X) = I_j$, daca $I_j = goto(I_i, X)$
- L (reducere cu r.p. nr. L)

  daca $[A \to \alpha.] \in I_i$

  $A \to \alpha \in P$: regula de prod. cu numarul L

  si: $T(I_i, X)$ nu se completeaza
- acc daca: $[S' \to S.] \in I_i$

Toate celelalte cazuri se considera eroare .

Automatul LR(0)

- Configuratie (stiva_de_lucru), banda_de_intrare, banda_de_iesire)
- Pe stiva: prefixe viabile, stari ale analizorului
- Configuratie initiala: ($0, w$, eps)
- Configuratie finala: ($0SIacc, $, banda_de_iesire)

Tranzitii:

deplasare:

$(\$ \gamma s_k , a_i..a_n\$, \Pi) \vdash (\$\gamma s_k a_i s_m , a_{i+1}\ldots a_n\$, \Pi)$

daca: $T(s_k,\text{actiune}) = s$ si $T(s_k,a_i) = s_m$

reducere:

$(\$ \gamma s_{p-1}X_p s_p\ldots X_k s_k, a_i..a_n\$,\Pi) \vdash (\$\gamma s_{p-1}A\ s_m, a_i\ldots a_n\$, \boldsymbol{L}\Pi)$

daca: $T(s_k,\text{action}) = \boldsymbol{L}$

si: $A \rightarrow X_p\ldots X_k$ – r.p. cu nr. $\boldsymbol{L}$

$T(s_{p-1}, \text{actiune}) = s$

$T(s_{p-1}, A) = s_m$

acceptare: $(\$\ 0S\ s_{acc} , \$, \Pi) \vdash$ acc.

eroare: *orice alta situatie*

O gramatica este LR(0) daca si numai daca tabelul de analiza nu contine conflicte.

Analiza sintactica SLR = Simple LR

Elemende de analiza SLR:

[A -> a.b, u] ; u = FOLLOW1(A) |u|=1

Analiza sintactica SLR

- Constructia tabelului de analiza SLR
- Actiunea de reducere depinde de predictia u

= > reducerea va avea o coloana pentru fiecare a (terminal)

Tabelul : linii: elementele colectiei canonice

Coloane: terminale, neterminale si $

Celula: s(stare), r(nr. Regulii de productie, acc)

O gramatica este SLR daca si numai daca tabelul de analiza nu contine conflicte.

LR(1) e la fel

Colectia canonica:

elem. initial

$$[S' \to .S\ ,\ \$]$$

Closure

$$[A \to \alpha.B\beta,\ a] \Rightarrow [B \to .\gamma,\ b] \in Closure([A \to \alpha.B\beta,\ a])$$

$$B \to \gamma \qquad \forall\ b \in FIRST_1(\beta a)$$

goto

$$goto(I,X) =$$

$$Closure(\{[A \to \alpha X.\beta, a] \mid [A \to \alpha.X\beta, a] \in I\})$$

Construirea tabelului de analiza LR(1):

$[A \to \alpha.X\beta, b] \in I_i$: $goto(I_i, X) = I_j$ <= functia *goto*

$action(I_i, X) = sj$

$[A \to \alpha.\ ,\ a] \in I_i$ $action(I_i, a) = rL$

L – nr. reg. de productie: $A \to \alpha$

$A <> S'$

$[S' \to S.,\ \$] \in I_i$ $action(I_i, \$) = acc$

Analizator LR(1) - similar cu LR(0), SLR

Analizator sintactic LALR

- [A -> alfa.beta, a] (ce I subliniat e nucleu)
- Colectia canonica LR(1)
- Fuzioneaza elemente de analiza cu nuclee identice si care nu creeaza conflicte
- Predictia: reuniunea predictiilor
- Tabelul LALR & analiza : similar LR(1)

LR(1-uri)

Conflict:

$[\, A \rightarrow \alpha_1.a\alpha_2 \,,\, u \,]$ deplasare-reducere
$[\, B \rightarrow \beta_1. \,,\, a \,]$

$[\, A \rightarrow \alpha_1. \,,\, a \,]$ reducere-reducere
$[\, B \rightarrow \beta_1. \,,\, a \,]$

EXEMPLU ANALIZA LR(1) PAG 270

-------------------------------CURS 11-------------------------------

Gramatica de atribute

G - gramatica independenta de context

- Fiecarui simbol al gramaticii I se asociaza 0 sau mai multe atribute : multime finita de atribute
- Fiecarei reguli de productie I se asociaza o multime finita de espresii ale atributelor asociate simbolurilor regulii de productie

=> reguli de evaluare a atributelor

Atribute:

Asocierile atribut - valoare sunt definite numai peste o "analiza sintactica" un arbore de derivare

Evaluator de atribute:

- Calculeaza valori & propaga valorile calculate
- Traverseaza arborele de derivare
- Strategie de traversare a arborelui si propagare a valorilor

Dacă un atribut b depinde de un alt atribut c, atunci regula semantică pentru calculul atributului b trebuie să fie evaluată după regula semantică care îl produce pe c.
=> Graful de dependenta

Metode de evaluare

- Metode bazate pe arborele de derivare:
- Metode bazate pe reguli
- Metode bazate pe o ordine pre-fixata

Fie regulile de evaluate a atributelor asociate urmatoarei reguli de productie:
A -> X1…Xk

- Atribut sintetizat: un atribut al lui A;
  regula de evaluare atribuie valoare atributului lui A
- Atribut mostenit: atribut al lio Xi;
  Regula de evaluare atribuie valoare atributului lui Xi
  El depinde de valorile parintilor si fratilor

Gramatica S-atributata

Def: exista doar atribute sintetizate si acestea depind de valorile atributelor copiilor
Evaluatea atributelor: - parcurgere 'in sus' a arboreluii de analiza sintactica
-> analizator sintactic descendent

Gramatica L-atributata

Def: pentru orice regula regula de productie A -> X1 X2 … Xn

- Un atribut mostenit a lui Xi depinde de atributele mostenite ale lui A si de atribute ale lui X1, X2, Xi-1
- Orice atribut sintetizat al lui A nu depinde de alte atribute sintetizate ale lui A

Gramatica de atribute(GA)

- Gramatica independenta de context
- Atribute + expresii ale atributelor

------------------------------CURS 12------------------------------

Translator finit PAG 292
Translator push-down PAG 296
Masini Turing PAG 303

RECAPITULARE PAG 310