1. Ierarhia Chomsky: clasificare, definitii

Def: O gramatica G = (N, T, S, P) se numeste:

de tipul 1 -> dependenta de context (context sensitive), daca fiecare productie $u \rightarrow v$ a sa satisface conditia $|u| \leq |v|$, u contine cel putin un neterminal

de tipul 2 -> independenta de context (context free), daca fiecare productie $u \rightarrow v$ a sa satisface conditia $|u| = 1, v \neq \lambda$

de tipul 3 -> regulata, daca fiecare productie $u \rightarrow v$ a sa satisface conditia $|u| = 1, v \in T^* \cup T^*N, v \neq \lambda$

Orice gramatica e de tipul 0. Orice gramatica de tipul i este si de tipul i-1, unde i=3,2,1.

Familia limbajelor generate de gramatici de tipul i (i=0,1,2,3) se noteaza cu L_i (i=0,1,2,3). $L3 \subseteq L2 \subseteq L1 \subseteq L0$.

Def: Doua gramatici G_1 si G_2 sunt echivalente daca ele genereaza acelasi limbaj si sunt de acelasi tip.

Lema: Fie G=(N,T,S,P) o gramatica de tipul i (i=3,2,1). Exista o gramatica G_1 echivalenta cu G a.i. simbolul initial S_1 al lui G_1 sa nu apara in nici unul din cuvintele aflate in membrul al doilea al productiilor gramaticii G_1.

Def: O gramatica G=(N,T,S,P) se numeste <u>recursiva</u> daca pentru orice cuvant $w \in T^+$ exista un algoritm pentru a decide daca $w \in L(G)$ sau nu. (se noteaza $T^+ = T^* - \{\lambda\}$)

Teorema: Gramaticile dependente de context sunt recursive.

Demonstratie: Fie G = (N,T,S,P) o gramatica de tipul 1 si $w \in T^*$. Notam $n = |w|$ si definim recursiv multimile:

- $U_0 = \{S\}$
- $U_{m+1} = U_m \cup \{ v \mid v \in (N \cup T)^+, \exists u \in U_m \text{ a.i. } u \Rightarrow v \text{ si } |v| \leq n \}$

Parcurgem arborele pornind de la S pana cand nu se mai gasesc cuvinte derivate cu lungimea mai mica de n.

2. Eliminarea redenumirilor

Def: O productie de forma $X \rightarrow Y$, X si Y neterminale, se numeste <u>redenumire</u>.

Propozitie: Fie G=(N,T,S,P) o gramatica de tipul 2 sau 3, exista o gramatica G_1 echivalenta cu G si fara redenumiri.

Demonstratie: Fie G=(N,T,S,P) o gramatica de tipul 2 sau 3.

Fie $P1 = \{ A \rightarrow u \mid u \,!\!\in N, A \rightarrow u \in P \}$ si

$P2 = \{ A \rightarrow u \mid A \in N, \exists B \in N \text{ a.i. } A \overset{+}{\Rightarrow}_G B, B \rightarrow u \in P_1 \}$

Productiile din P_2 nu sunt redenumiri. Fie $P' = P \cup P_2$. Gramatica $G_1 = (N, T, S, P')$ este fara redenumiri si se arata usor ca este echivalenta cu G. $G_1 \sim G \iff L(G_1) = L(G)$

Exemplu: Fie G=({S,A,B}, {a,b,c}, S, P) unde P:

$S \rightarrow A \mid aS \mid a$
$A \rightarrow B \mid bA \mid b$
$B \rightarrow cB \mid c$

$S \rightarrow A$ si $A \rightarrow B$ sunt redenumiri. $\exists G_1$ echivalenta cu G care sa nu contina aceste productii.

Fiindca $A \rightarrow B$ inseamna ca $A \rightarrow cB$ si $A \rightarrow c$ trebuie sa adaugam aceste productii pentru a pastra echivalenta.

Gramatica echivalenta G_1=({S,A,B}, {a,b,c}, S, P_1) fara redenumiri va avea P_1:

$S \rightarrow aS \mid a \mid bA \mid b \mid cB \mid c$
$A \rightarrow bA \mid b \mid cB \mid c$
$B \rightarrow cB \mid c$

3. Gramatici regulate

Def: O gramatica G = (N, T, S, P) unde

- N este multimea neterminalelor, o multime nevida finita
- T este multimea terinalelor, o multime nevida finita
- $S \in N$ este simbolul de start
- $P = \{ u \rightarrow v \mid u,v \in (N \cup T)^*$ si u contine cel putin un neterminal$\}$ este multimea productiilor

este o gramatica regulata sau de tipul 3 daca fiecare productie $u \rightarrow v$ a sa satisface conditiile: $|u| = 1$, $v \in T^* \cup T^*N$, $v \neq \lambda$.

4. Forma normala (canonica) a unei gramatici regulate

Def Bris: O gramatica regulata este in forma normala daca productiile sale au forma

$A \to aB$ sau $A \to a$, unde $A,B \in N$ si $a \in T$.

Propozitie: Pentru orice gramatica regulata $G=(N,T,S,P)$ exista o gramatica $G_1 = (N_1, T, S, P_1)$ echivalenta cu ea si avand proprietatea ca fiecare producatie $u \to v \in P_1$ a sa satisface conditiile $u \in N_1$; $v \in T \cup TN_1$.

Aducerea la forma normala:

- Pentru fiecare productie de forma $A \to a_1a_2\ldots a_k$ cu $k>1$ si $a_i \in T$, adaugam in N_1 neterminalele: $A_1, A_2, \ldots, A_{k-1}$ si in P_1 productiile: $A \to a_1A_1$, $A_1 \to a_2A_2, \ldots, A_{k-1} \to a_k$
- Pentru fiecare productie de forma $A \to a_1a_2\ldots a_kB$ cu $k>1$, $a_i \in T$, $B \in N$, adaugam in N_1 neterminalele: $A_1, A_2, \ldots, A_{k-1}$ si in P_1 productiile: $A \to a_1A_1$, $A_1 \to a_2A_2, \ldots, A_{k-1} \to a_kB$

G_1 rezultat va fi in forma normala si va avea aceleasi terminale ca si G, neterminalele vor fi cele vechi la care le adaugam pe cele nou introduse, simbolul de start va fi acelasi, iar productiile vor fi cele pe care le-am introdus conform procedurilor de mai sus.

Exemplu: Fie $G=(\{S,A,B\}, \{a,b,c\}, S, \{S \to abc, S \to abA, \mathbf{A \to aB, B \to b}\})$

Productiile $A \to aB$ si $B \to b$ respecta regula si deci le adaugam in lista productiilor lui G_1.

Pentru productia $S \to abc$ adaugam neterminalele X si Y si productiile: $S \to aX$, $X \to bY$, $Y \to c$

Pentru productia $S \to abA$ adaugam neterminalul Z si productiile: $S \to aZ$, $Z \to bA$

$\Rightarrow G_1 = (\{S,A,B,X,Y,Z\}, \{a,b,c\}, S, \{ S \to aX, X \to bY, Y \to c, S \to aZ, Z \to bA, \mathbf{A \to aB, B \to b}\})$

5. Automate cu stari finite (definitii, limbaje acceptate)

Def: Se numeste Automat Finit Determinist (AFD) un quintuplu $(\Sigma, Q, q_0, \delta, F)$, unde:

- Σ este un alfabet numit alfabetul de intrare
- Q este o multime finita nevida numita multimea starilor interne
- $q_0 \in Q$ este starea initiala a automatului
- $\delta : Q \times \Sigma \to Q$ se numeste functia de tranzitie a automatului
  (egaliltatea $\delta(q,a) = r$ exprima trecerea automatului din starea *q* in starea *r* atunci cand citeste simbolul de intrare *a*)
- $F \subseteq Q$ este multimea starilor finale ale automatului

Def: Se extinde δ la $\overline{\delta} : Q \times \Sigma^* \to Q$ care este definita astfel:

- $\overline{\delta}(q, \lambda) = \lambda$
- $\overline{\delta}(q, wx) = \delta(\overline{\delta}(q, w), x)$, pentru orice $w \in \Sigma^*$, orice $x \in \Sigma$ si orice $q \in Q$

Def: Fie $A = (\Sigma, Q, q_0, \delta, F)$ un AFD. **Limbajul acceptat** de automatul A este: $L(A) = \{ w \mid w \in \Sigma^*, \delta(q_0,w) \in F \}$

Lema: Fie A un AFD. Fiind data starea $\delta(q,w) = s$, cu $w \neq \lambda$, $w = w_1w_2\ldots w_n$, $w_i \in \Sigma$, $i = 1,2,\ldots n$, exista starile $q_1, q_2, \ldots q_{n+1}$ a.i. $q_1 = q$, $q_{n+1} = s$, $q_{i+1} = \delta(q_i,w_i)$, $i = 1, 2, \ldots, n$.

Lema: Fie A un AFD. Fiind date starile $q_1, q_2, \ldots q_{n+1}$ a.i. $q_{i+1} = \delta(q_i,w_i)$, $i = 1, 2, \ldots, n$, atunci $q_{n+1} = \delta(q_1,w)$, cu $w = w_1w_2\ldots w_n$, $w_i \in \Sigma$, $i = 1,2,\ldots n$.

Def: Un Automat Finit Nedeterminist (AFN) este este un quintuplu $(\Sigma, Q, Q_0, \delta, F)$, unde:

- Σ este un alfabet numit alfabetul de intrare
- Q este o multime finita nevida numita multimea starilor interne
- $Q_0 \subseteq Q$ este o multime nevida, numita multimea starilor initiale ale automatului
- $\delta : Q \times \Sigma \to 2^Q$ se numeste functia de tranzitie a automatului
- $F \subseteq Q$ este multimea starilor finale ale automatului

Def: Fie $A = (\Sigma, Q, Q_0, \delta, F)$ un AFN. **Limbajul acceptat** de A este format din toate cuvintele $w = w_1w_2\ldots w_n$ ($w_i \in \Sigma$, $i = 1, 2, \ldots, n$) pentru care exista $q_1, q_2, \ldots, q_{n+1}$ cu $q_1 \in Q_0$, $q_{n+1} \in F$ si $q_{i+1} \in \delta(q_i;w_i)$, $i=1,2,\ldots,n$, adica

$L(A) = \{ w \mid w \in \Sigma^*, \delta(q_0, w) \cap F \neq \emptyset \}$

Este evident ca orice AFD poate fi privit ca un AFN, prin urmare avem urmatoarele teoreme:

Teorema: Limbajul reprezentabil intr-un AFD este reprezentabil intr-un AFN.

Teorema: Limbajul reprezentabil intr-un AFN este reprezentabil intr-un AFD.

Demonstratie: Fie L un limbaj reprezentat in AFN-ul A = ($\sum$, Q, Q_0, δ, F). Consideram AFD A_1 = ($\sum$, 2^Q, Q_0, δ_1, F_1), unde functia de tranzitie este definita astfel:

- $\delta_1(P, a) = \emptyset$, daca $P = \emptyset$
- $\delta_1(P, a) = \bigcup_{q \in P} \delta(q, a)$, daca $P \neq \emptyset$,

iar multimea starilor finale este: $F_1 = \{ S \mid S \subseteq Q, S \cap F \neq \emptyset \}$.

Diferente:

- AFD are o singura stare initiala q_0 , AFN are o multime nevida de stari initiale Q_0

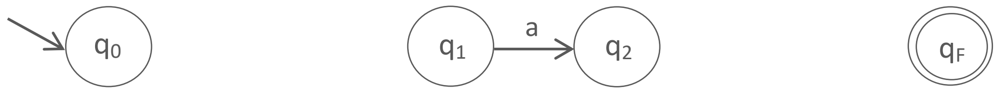
- functia δ:
  - $\delta_{AFN}: Q \times \sum \rightarrow 2^Q$

    AFN accepta tranzitii cu λ, adica poate exista $\delta(q, \lambda)=q_1$

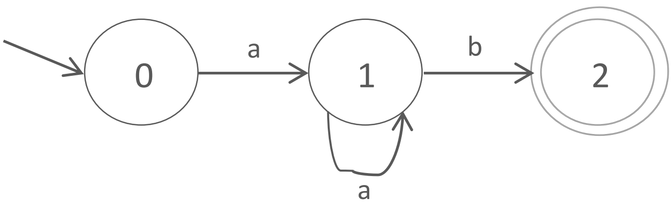
    AFN accepta mai multe tranzitii dintr-o stare q cu un caracter a, adica poate exista $\delta(q,a)=q1$ si $\delta(q,a)=q2$
  - $\delta_{AFD}: Q \times \sum \rightarrow Q$

    AFD accepta o singura tranzitie dintr-o stare q cu un caracter a, adica $|\delta AFD (q,a)|=1$ oricare $q \in Q$ si oricare $a \in \sum$

Notatii: Stare initiala | Tranzitia $\delta(q_1, a) = q_2$ | Stare finala



Exemplu: Se da limbajul *L*(A)={ $a^k b \mid k > 0$ }. Sa se scrie AFD A. //a^k inseamna a scris de k ori: $a^1 = a$, $a^3 = aaa$



A = ($\sum$, Q, Q_0, δ, F) unde:

$\sum = \{a, b\}$,

Q = {0,1,2}, $q_0 = 0$, F = {2}

δ(0, a) = 1,

δ(1, a) = 1,

δ(1, b) = 2

6. Stari accesibile ale unui AFD

Def: Multimea starilor accesibile ale unui AFD A = ($\sum$, Q, q_0, δ, F) este multimea

$Q_a = \{ q \mid q \in Q, \exists w \in \sum^* \text{ a.i. } \delta(q_0,w) = q \}$

Cu alte cuvinte, starile accesibile ale unui automat sunt acele stari in care se poate ajunge pornind din starea initiala si primind la intrare un cuvant oarecare w.

Starile accesibile pot fi calculate cu urmatorul algoritm:

- $U_0 = \{q_0\}$
- $U_{m+1} = U_m \cup \{ q \mid q \in Q, \exists a \in \sum \text{ si } \exists s \in U_m, \text{ a.i. } \delta(s, a) = q \}$

Cel mai mic $i \in N$ pentru care $U_i = U_{i+1}$ ne permite determinarea starilor accesibile: $Q_a = U_i$.

Acelasi algoritm dar scris de Dragulici:

p1. $U_0 = \{q_0\}$; i := 0

p2. $U_{i+1} = U_i \cup \{ \delta(q,a) \mid q \in U_i, a \in \sum\}$

p3. if $U_{i+1} = U_i$ then $Q_a = U_i$; stop;

else i = i+1; go to p2.

7. Echivalenta dintre automate cu stari finite si gramatici regulate

Pentru o gramatica regulata G exista un automat A astfel ca *L*(A) = *L*(G):

G=(N, T, S,P)	A=(∑, Q, q_0, δ, F)
T	∑ = T
N	Q = N ∪ {q_F}, F = {q_F}
S	q_0 = S
P: S⟶aA	δ(q_0, a) = q_A
P: A⟶b	δ(q_A, b) = q_F
daca S⟶λ	F = {q_F} ∪ S

i) data o gramatica regulata, sa se construiasca un AFN echivalent (se accepta si un AFD)

Exemplu: Fie o gramatica G=(N, T, S, P), unde N = {S,A}

T = {a,b}

P = { S⟶aA, A⟶aA, A⟶b }

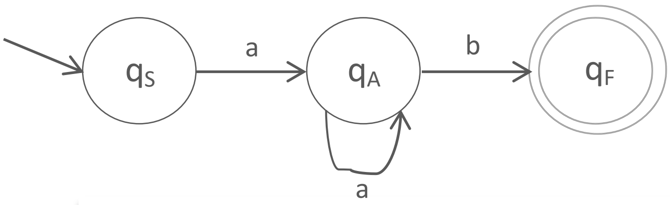
=> AFD echivalent A = (∑, Q, q_0, δ, F) unde ∑ = T = {a,b}

Q = N ∪ {q_F} = {q_S,q_A,q_F} //am folosit notatia S = q_S si A = q_A

F = {q_F}

q_0 = q_S

δ : δ(q_S, a) = q_A, δ(q_A, a) = q_A, δ(q_A, b) = q_F



AF echivalent cu o gramatică regulată

- Fie gramatica regulată la dreapta G = <N, Σ, S, P>.
- Să se construiască un AF = <Q, Σ, δ, q0, F> care acceptă exact limbajul generat de G.

- Notăm stările automatului cu neterminalele gramaticii, plus un neterminal nou, inexistent în gramatică, adică, Q = N ∪ {A}, A ∉N și fie q0 = S.
- Construim δ astfel:
- a) dacă B → aC ∈ P atunci C ∈ δ(B, a);
- b) dacă B → a ∈ P atunci A ∈ δ(B, a);
- Considerăm δ(A, a) = ∅, a ∈Σ
- F = {A}, dacă S → λ ∉ P și F = {S, A}, dacă S → λ ∈ P

Exemplu

Gramatica regulată echivalentă cu un AF

- Fie AF = <Q, Σ, δ, q0, F> un AFD.
- Definim gramatica regulată G = <N, Σ, q0, P> având producțiile deduse astfel:
- q1 → aq2 ∈ P dacă δ(q1, a)=q2;
- q1 → a ∈ P dacă δ(q1, a)=q2, q2 ∈ F;
- Constatăm că L(G) = L(AF).
- **$L_{AF} \subseteq L^3$**

8. Aducerea la Forma normala Chomsky (FNC) a unei gramatici de tip 2

Def Bris: O gramatica este in forma normala Chomsky daca productiile sale au forma

$A \rightarrow BC$ sau $A \rightarrow a$, unde $A,B,C \in N$ si $a \in T$.

Exemplu: Fie $G=(\{S,Z\}, \{a,c,z\}, S, \{S \rightarrow aaZccc, \mathbf{Z \rightarrow z}\})$
Productia $Z \rightarrow z$ respecta regula si deci o adaugam in lista productiilor lui G_1.
Pentru fiecare terminal ramas se creeaza un neterminal si o productie: $A_1 \rightarrow a$, $C_1 \rightarrow c$
Pentru productia $S \rightarrow aaZccc$ se creeaza neterminale si productii pentru alipirea terminalelor:

- pentru aa vom avea $A_2 \rightarrow A_1A_1$
- pentru cc vom avea $C_2 \rightarrow C_1C_1$
- pentru ccc vom avea $C_3 \rightarrow C_1C_2$ (sau $C_3 \rightarrow C_2C_1$)

iar apoi se creeaza ce neterminale si productii mai sunt necesare pentru a forma o productie echivalenta:

- pentru aaZ vom avea $Z_1 \rightarrow A_2Z$
- pentru aaZccc vom avea $S \rightarrow Z_1C_3$

=> $G_1 = (\{S, Z, A_1, A_2, C_1, C_2, C_3, Z_1\}, \{a, c, z\}, S, \{ S \rightarrow Z_1C_3, Z_1 \rightarrow A_2Z, A_2 \rightarrow A_1A_1, A_1 \rightarrow a, C_3 \rightarrow C_1C_2, C_2 \rightarrow C_1C_1, C_1 \rightarrow c, Z \rightarrow z\})$

9. Algoritmul CYK (pentru gramatici de tipul 2)

i) Folosing algoritumului CYK dem. ca un cuvant este sau nu generat de o gramatica independenta de context
program de rezolvare : https://www.xarg.org/tools/cyk-algorithm

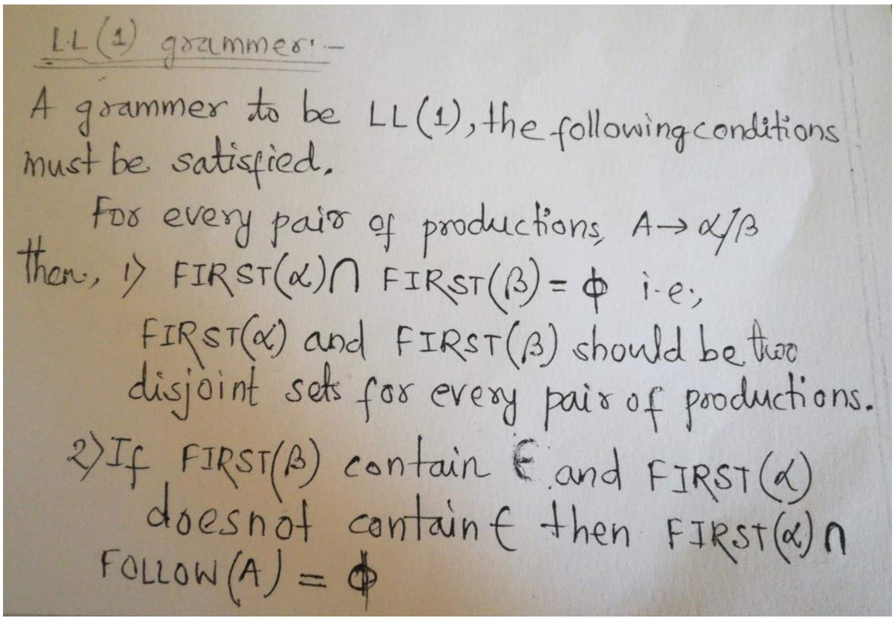
ii) sa se genereze derivare unui cuvant cu ajutorul algoritmului CYK
Exemplu: Fie o gramatica cu productiile: $\{S \rightarrow AB, A \rightarrow DC, B \rightarrow b, C \rightarrow c, D \rightarrow d \}$
Derivarea cuvantului dcb este: $\mathbf{S} \rightarrow^1 \mathbf{A}B \rightarrow^2 \mathbf{D}CB \rightarrow^5 d\mathbf{C}B \rightarrow^4 dc\mathbf{B} \rightarrow^3 dcb$

10. Gramatici de tip LL(1) (pentru gramatici de tipul 2)

Explicatii: part1: https://www.youtube.com/watch?v=I8hOEeSmY5E
part2: https://www.youtube.com/watch?v=BFFkYhb6Hmc

A->a|b. You can check if a grammar of LL(1) or not by using following two expressions:

if (first (a) ∩ first (b) != null) => not LL(1)
if (first (a) ∩ first (b) == null)
if (λ ϵ first (a) OR follow (A) ∩ first (b) == null) => LL(1).


LL(1) grammer:-
A grammer to be LL(1), the following conditions must be satisfied.
For every pair of productions $A \rightarrow \alpha / \beta$
then, 1) $FIRST(\alpha) \cap FIRST(\beta) = \phi$ i.e.,
$FIRST(\alpha)$ and $FIRST(\beta)$ should be two disjoint sets for every pair of productions.
2) If $FIRST(\beta)$ contain ϵ and $FIRST(\alpha)$ doesnot contain ϵ then $FIRST(\alpha) \cap FOLLOW(A) = \phi$

alte surse de inspiratie pentru teorie in general : https://staff.fmi.uvt.ro/~mircea.dragan/docs/Limbaje2.pdf