Gramatici independente de context

Definitia 1: O gramatica G = (V, Σ, S, R) se numeste **gramatica indepenenta de context** daca toate regulile sale sunt de forma:

$A \to x$,

unde $A \in V$ si $x \in (V \cup \Sigma)^*$.

Definitia 2: Un limbaj generat de o gramatica independenta de context se numeste **limbaj independent de context**.

Observatii:
1. Orice limbaj regulat este i.d.c.
2. Limbajul $L = \{ a^n b^n \mid n \geq 0 \}$ (despre care am aratat ca nu este regulat) este i.d.c.. Intr-adevar, limbajul L este generat de gramatica i.d.c. G = ({S}, Σ, S, R) cu regulile:

$S \to aSb$
$S \to \lambda$

Definitia 3: O derivatie se numeste **derivatie la stanga** daca la fiecare pas al derivatiei variabila cea mai din stanga se inlocuieste.

O derivatie se numeste **derivatie la dreapta** daca la fiecare pas al derivatiei variabila cea mai din dreapta se inlocuieste.

Exemplu: Fie gramatica i.d.c. G = ({S, A, B}, {a, b}, S, R) ale carei reguli sunt:

$S \to AB$
$A \to aA$
$B \to Bbb$
$A \to a$
$B \to \lambda$

Este usor de observat ca limbajul generat de G este $L(G) = \{a^m b^{2n} \mid m \geq 1, n \geq 0\}$. Scriem in continuare doua derivatii care conduc la acelasi cuvant:

$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaB \Rightarrow aaBbb \Rightarrow aabb$ (derivare la stanga)

si, respectiv:

$S \Rightarrow AB \Rightarrow ABbb \Rightarrow Abb \Rightarrow aAbb \Rightarrow aabb$ (derivare la dreapta)

Se observa ca in cazul primei derivari de fiecare data variabila ce mai din stanga se inlocuieste, iar la a doua derivare variabila cea mai din dreapta se inlocuieste. Asadar, prima derivare este la stanga, iar a doua este la dreapta.

Arbore de derivare

O alta modalitate de a prezenta o gramatica i.d.c. este printr-un arbore de derivare.

Definitia 4: Fie G = (V, Σ, S, R) o gramatica indepenenta de context. Un arbore se numeste **arbore de derivare** al gramaticii G daca si numai daca indeplineste simultan regulile:

1. S este radacina arborelui
2. Fiecare frunza este etichetata cu λ sau un terminal a ∈ Σ
3. Fiecare nod intermediar (care nu este frunza) este etichetat cu un neterminal A ∈ V
4. Daca un varf este etichetat cu neterminalul A ∈ V si copii lui sunt a_1, a_2, é ., a_n, atunci gramatica G contine regula:

   $A \rightarrow a_1 a_2$é $.a_n$

5. Un nod care are ca si fiu pe λ nu mai poate avea alti copii.

Definitia 5: Un arbore se numeste **arbore partial de derivare** pentru gramatica G daca si numai daca indeplineste simultan regulile 1, 3, 4, si 5 de mai sus si regula:

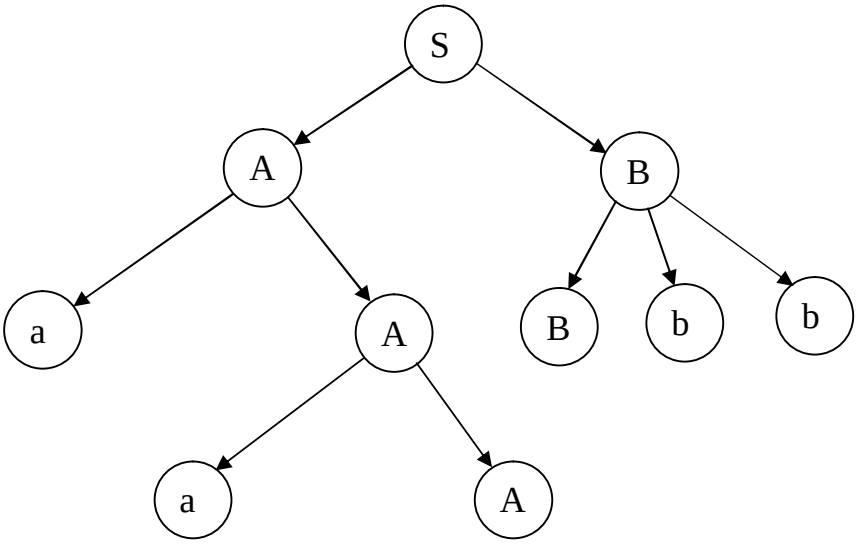
2'. Fiecare frunza este etichetata cu λ, cu un terminal a ∈ Σ sau cu un neterminal A ∈ V.

Parcurgand un arbore de derivare in adancime, iar copiii de la stanga la dreapta si luand in considerare numai terminalele, se obtine o **propozitie** a limbajului L(G).

Parcurgand un arbore partial de derivare in adancime si luand in considerare numai frunzele, se obtine o asa numita forma **proprozitionala** a gramaticii G.

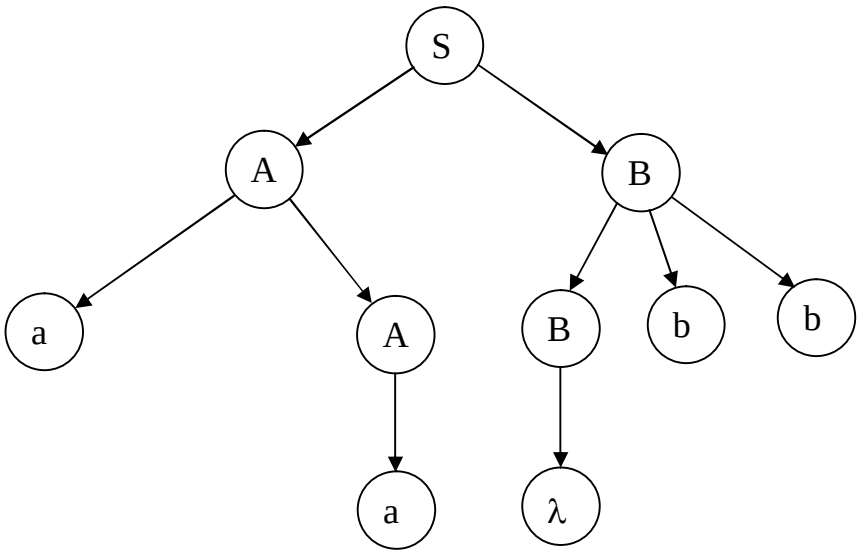
Un arbore de derivare da o descriere explicita si usoara a unei derivari.

Exemplu: Pentru gramatica din exemplul de mai sus arborele de mai jos este un arbore partial de derivare:



Arborele partial de derivare de mai sus are forma propozitionala: aaABbb.

Arborele de derivare de mai jos corespunde oricarei derivari din exemplul de mai sus:



Propozitia obtinuta in urma parcurgerii arborelui de derivare este: aabb.

Teorema 1: Fie G = (V, Σ, S, R) o gramatica independenta de context.

1. Pentru fiecare w ∈ Σ* generat de G exista un arbore derivare a carui parcurgere este w.
2. Parcurgerea unui arbore partial de derivare al lui G conduce la o forma propozitionala a gramaticii G.

Demonstratie:

1. Fiecare propozitie $w \in \Sigma^*$ se obtine in urma unei derivari care este echivalenta cu un arbore de derivare.
2. Evident.

Parsare top-down

Pentru o gramatica i.d.c. $G = (V, \Sigma, S, R)$ ne intereseaza sa decidem daca un string $w \in L(G)$ si, daca da, sa gasim un arbore de parsare. Pentru aceasta exista o metoda denumita **parsare top-down**.

Parsarea top-down porneste cu regulile de forma:

$S \rightarrow x.$

Daca exista $x \in (V \cup \Sigma)^*$ care sa se potriveasca cu w, atunci se inlocuieste in x variabila A cea mai din stanga cu y daca exista regula de forma:

$A \rightarrow y.$

Adica avem derivatia:

$S => x = x_1Ax_2 => x_1yx_2 \ (x_1 \in \Sigma^*).$

Daca yx_2 nu este format numai din terminale, atunci se inlocuieste in yx_2 variabila B cea mai din stanga cu partea dreapta z a unei reguli de forma:

$B \rightarrow z.$

Se continua inlocuirea cea mai stanga variabila pana ce se obtine w sau pana se observa ca derivatia nu conduce la w.

Exemplu:

Fie gramatica i.d.c.:

$S \rightarrow SS|aSb|bSa|\lambda.$

Pentru string-ul w = aabb avem in prima iteratie a algoritmului:

$S => SS$
$S => aSb$
$S => bSa$
$S => \lambda$

Este evident ca ultimele doua derivari nu pot conduce la w. Pentru primele obtinem in a doua iteratie a algoritmului (inlocuind variabila S din stanga cu partea dreapta a regulilor gramaticii):

S => SS => SSS
S => SS => aSbS
S => SS => bSaS
S => SS => S

Si pentru a doua:

S => aSb => aSSb
S => aSb => aaSbb
S => aSb => abSab
S => aSb => ab

Derivarile 3, 7 si 8 se elimina evident pentru ca nu pot conduce la w = aabb. Derivarea 4 se poate elimina si ea pentru ca am ajuns la o forma propozitionala prin care am mai trecut.

In iteratia urmatoare a algoritmului se obtine w in derivarea 6:

S => aSb => aaSbb => aabb = w.

Este evident ca o implementare top-down directa (exhaustiva) conduce la un algoritm exponential ineficient. De aceea inainte de aplicarea algoritmului se aplica optimizari gramaticii.

Teorema 2: Daca o gramatica nu contine reguli de forma:

$A \rightarrow \lambda$

si

$A \rightarrow B,$

unde A, B sunt variabile, atunci algoritmul de cautare exhaustiva se termina in numar finit de pasi si decide daca exista sau nu o derivatie care sa conduca la orice string dat w.

Demonstratie: La fiecare iteratie a algoritmului fiecare forma propozitionala construita de o derivatie creste in lungime cu cel putin o unitate sau o variabila este inlocuita cu o combinatie de terminale. Cresterea in lungime nu are rost sa o efectuam decat pana fiecare forma propozitionala ajunge de lungimea lui w. Asadar, avem cel mult |w|-1

cresteri in lungime a fiecarei forme propozitionale si cel mult |w| inlocuiri de variabile cu terminale.

Algoritmul se incheiei dupa cel mult 2|w|-1 iteratii (q.e.d.).

Daca aplicam algoritmul top-down in forma exhaustiva obtinem cel mult:

$$\sum_{k=1}^{2|w|-1} |R|^k = \frac{|R|^{2|w|}-1}{|R|-1}$$

forme propozitionale, unde |R| este numarul de reguli ale gramaticii. Asadar, complexitatea algoritmului top-down in implementare exhaustiva are complexitatea $O(|R|^{2|w|})$.

Exista insa algoritm in complexitate $O(|w|^3)$ care parseaza orice $w \in L(G)$.

Definitie 1: O gramatica i.d.c. G = (V, Σ, S, R) se numeste **gramatica simpla** sau **s-gramatica**, daca toate regulile ei sunt de forma:

$A \rightarrow ax,$

unde $A \in V$, $a \in \Sigma$, $x \in V^*$ si nu exista $A \in V$, $a \in \Sigma$, $x, y \in V^*$ astfel incat regulile:

$A \rightarrow ax,$
$A \rightarrow ay$

sa fie amandoua in R.

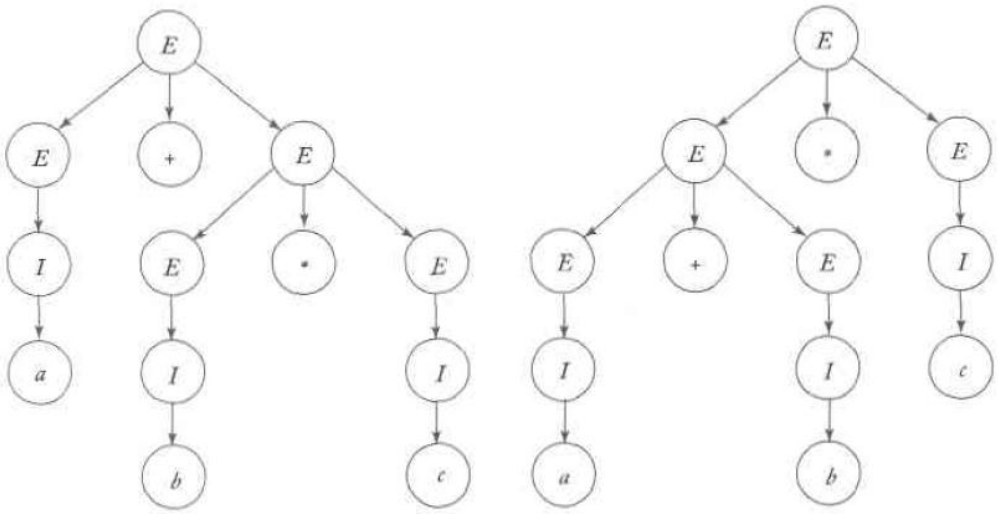
Este evident ca parsarea oricarei propozitii $w \in L(G)$ se poate face intr-o gramatica simpla in complexitate O(|w|). Este ideal atunci sa ne apropiem cat mai mult posibil de o s-gramatica inainte de a incepe parsarea.

Ambiguitatea gramaticilor i.d.c.

Definitie 2: O gramatica i.d.c. G se numeste **ambigua** daca exista mai mult de un arbore de derivare pentru anumite propozitii $w \in L(G)$.

Atunci cand definim o gramatica trebuie sa eliminam ambiguitatile pentru ca fiecare propozitie sa fie interpretata unic.

De exemplu, pentru expresia a+b*c putem construi doi arbori de derivare:



Ambiguitatea de mai sus este rezolvata pentru limbajele de programare prin stabilirea precedentei de aplicare a opertorilor, ceea ce face ca numai primul arbore de derivare sa fie valid.

Definitie 3: Un limbaj i.d.c. se numeste **ambiguu** daca orice gramatica ce il genereaza este ambigua.

Simplificarea gramaticilor i.d.c.

Teorema 3: Fie o gramatica i.d.c. G = (V, Σ, S, R) care contine urmatoarea regula:

$A \rightarrow x_1Bx_2$.

Consideram regulile din R care au pe B ca termen in partea stanga:

$B \rightarrow y_1|y_2|é\ .|y_n$.

Plecand de la G construim gramatica G' = (V, Σ, S, R') prin eliminarea regulii

$A \rightarrow x_1Bx_2$

si prin adaugarea regulilor:

$A \rightarrow x_1y_1x_2|x_1y_2x_2|é\ .|x_1y_nx_2$.

Gramaticile G si G' sunt echivalente, adica L(G) = L(G').

Demonstratie: Daca o derivare a lui w $\in$ L(G) in G nu contine regula eliminata in constructia lui G', atunci aceeasi derivare a lui w se obtine si in G'.

Daca pentru derivarea unei propozitii w $\in$ L(G) in G se foloseste regula ce se doreste a fi eliminata, atunci avem (in G):

(1) $S =>^* z_1Az_2 => z_1x_1Bx_2z_2 => z_1x_1y_kx_2z_2 =>^* w$ ($k \in \{1, 2, é\ ., n\}$).

Derivarea de mai sus se poate rescrie in G' astfel:

(2) $S =>^* z_1Az_2 => z_1x_1y_kx_2z_2 =>^* w$

folosind regula introdusa in G':

$A \rightarrow x_1y_kx_2$.

Reciproc, daca avem derivarea (2) in G', atunci se obtine derivarea (1) in G pentru orice w $\in$ L(G') (q.e.d.).

Exemplu: Pentru gramatica:

$S \rightarrow aaA|abBb|AA$
$B \rightarrow b|aAa,$

eliminand regula:

$S \rightarrow abBb$

se obtine gramatica echivalenta:

$S \rightarrow aaA|\ abbb|abaAab|AA$
$B \rightarrow b|aAa.$

Se observa ca a doua regula este inutila, ceea ce face ca si variabila B sa fie inutila in G'.

Eliminarea regulilor inutile

In gramatica G data prin regulile:

$S \rightarrow aSb|\lambda|A$
$A \rightarrow aA$

se observa ca a treia si ultima regula sunt inutile deoarece nu conduc la o derivare a niciunei propozitii $w \in L(G)$. Mai mult, si variabila A este inutila in G.

Definitie 4: O variabila $A \in V$ din gramatica i.d.c. $G = (V, \Sigma, S, R)$ se numeste **utila** daca exista $x, y \in (V \cup T)^*$ si $w \in L(G)$ pentru care avem derivarea:

$S =>^* xAy =>^* w.$

In exemplul de mai sus variabila S este utila, A nu este.

Pornind de la o gramatica $G = (V, \Sigma, S, R)$ urmatorul algoritm construieste o gramatica $G' = (V' \subseteq V, \Sigma, S, R' \subseteq R)$ echivalenta in care toate variabilele si toate regulile sunt utile:

Pasul 1. $V' := \varnothing$;
Pasul 2. **Repeta**
Adaugat := **false**;
Pentru fiecare $A \in V$ **executa**
Daca exista in R regula de forma
$A \rightarrow x$, unde $x \in (V' \cup \Sigma)^*$
atunci
$V' := V' \cup \{A\}$;
Adaugat := **true**;
sfarsit daca;
sfarsit pentru;
pana cand not Adaugat;
Pasul 3. R' contine regulile din R care sunt definite folosind simboluri din $V' \cup \Sigma$.

Teorema 4: L(G) = L(G'), unde G' este gramatica ce a fost construita cu algoritmul de mai sus.

Demonstratie: Orice derivare in G de forma:

$S =>^* w$, unde $w \in L(G)$

este formata numai cu reguli din R'. Asadar, cele doua gramatici sunt echivalente (q.e.d.).

Eliminarea λ-regulilor

Definitie 5: O regula de forma:

$A \rightarrow \lambda$

se numeste **λ-regula**.

O variabila A pentru care exista derivarea:

$A =>^* \lambda$

se numeste variabila **λ-derivabila**.

Urmatorul algoritm determina multimea V_λ formata cu variabilele λ-derivabile:

Pasul 1: $V_\lambda := \varnothing$;
Pasul 2: **Pentru** fiecare regula de forma $A \rightarrow \lambda$ **executa**
$V_\lambda := V_\lambda \cup \{A\}$;
sfarsit pentru;
Pasul 3: **Repeta**
Adaugat := **false**;
Pentru fiecare $A \in V$ **executa**
Daca exista in R regula de forma
$A \rightarrow x$, unde $x \in (V_\lambda)^*$
atunci
$V_\lambda := V_\lambda \cup \{A\}$;
Adaugat := **true**;
sfarsit daca;
sfarsit pentru;
pana cand not Adaugat;

Daca o gramatica G = (V, Σ, S, R) nu accepta propozitia λ ($\lambda \notin L(G)$), atunci gramatica G' = (V, Σ, S, R') construita de algoritmul mai jos este echivalenta cu G si nu contine λ-reguli:

Pasul 1: Se construieste V_λ cu algoritmul de mai sus;
Pasul 2: R' se initializeaza cu regulile R, mai putin cele de forma $A \rightarrow \lambda$.
Pasul 3: Pentru fiecare regula in care in partea dreapta apar $k > 0$ variabile din V_λ, se adauga 2^k-1 reguli in R' distincte doua cate doua in care cel putin o variabila din V_λ se elimina (inlocuindu-se cu λ).

Teorema 5: L(G) = L(G'), unde G' este gramatica ce a fost construita cu algoritmul de mai sus.

Demonstratie: Fie w ∈ L(G). Inseamna ca exista in G o derivare de forma:

S =>* w.

Daca in derivarea de mai sus avem A =>* λ, atunci:

(1) S =>* uAv =>* uv =>* w.

Derivarea de mai sus este echivalenta in G' cu:

(2) S =>* uv =>* w.

Reciproc, daca avem in G' o derivare de tipul (2) pentru w ∈ L(G), ea corespunde in G unei derivari de forma (1) (q.e.d.).

Exemplu: Sa se elimine λ-regulile pentru gramatica de mai jos:

$S \rightarrow ABaC$
$A \rightarrow BC,$
$B \rightarrow b|\lambda,$
$C \rightarrow D|\lambda,$
$D \rightarrow d.$

Se observa ca V_λ = {A, B, C} (B, C intra direct in V_λ, deoarece exista regulile B → λ si C → λ, iar A intra in V_λ deoarece exista regula A → BC). Facand toate substitutiile posibile in partea dreapta a regulilor 1 si 2, obtinem gramatica echivalenta fara λ-reguli:

$S \rightarrow ABaC\,|BaC|\,AaC\,|ABa|\,aC\,|Aa|\,Ba|a$
$A \rightarrow B\,|C|\,BC,$
$B \rightarrow b,$
$C \rightarrow D,$
$D \rightarrow d.$

Eliminarea regulilor unitare

Definitie 6: O regula de forma:

$A \rightarrow B$

se numeste **regula unitara**, unde $A, B \in V$.

Este evident ca o regula de forma $A \rightarrow A$ n-are nici un efect in generarea propozitiilor, ele putand fi eliminate. Asadar, prezinta interes numai eliminarea regulilor de forma $A \rightarrow B$, unde $A \neq B$.

Presupunem ca avem o gramatica $G = (V, \Sigma, S, R)$ fara λ-reguli pentru care vrem sa eliminam regulile unitare. Daca exista λ-reguli in G, ele pot fi eliminate (asa cum am vazut mai sus).

Algoritmul de constructie a gramaticii $G' = (V, \Sigma, S, R')$ echivalente cu G fara reguli unitate are urmatorii pasi:

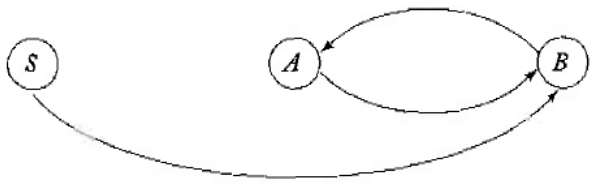
1. R' o initializam cu regulile din R care nu sunt unitare.
2. **Pentru** fiecare pereche de variabile $A \neq B$ pentru care $A =>^* B$ **executa**
   Pentru fiecare regula de forma $B \rightarrow x$ din R' **executa**
   Adauga in R' regula $A \rightarrow x$;
   sfarsit pentru;
   sfarsit pentru.

Pentru determinarea perechilor de variabile $A \neq B$ pentru care $A =>^* B$ se constrieste graful de dependente $G_d = (V, U)$, unde $U = \{(A, B) \mid A, B \in V, \tilde{n}A \rightarrow B\grave{o} \in R\}$. Este evident ca $A =>^* B$ $(A \neq B)$ daca si numai daca exista un drum de la A la B in G_d. In consecinta, determinarea perechilor de variabile $A \neq B$ pentru care $A =>^* B$ se face aplicand un algoritm in timp liniar de parcurgere a grafului G_d din fiecare variabila.

Exemplu: Sa se elimine regulile unitare ale gramaticii:

$S \rightarrow Aa|B,$
$B \rightarrow A|bb,$
$A \rightarrow a\,|bc|\,B$

Graful G_d este:



Evident, parcurgand graful de mai sus obtinem: S =>* B, S =>* A, A =>* B si B =>* A.

Aplicand algoritmul de mai sus obtinem gramatica echivalenta, dar fara reguli unitare:

$S \rightarrow a\,|bc|\,bb|\,Aa,$
$A \rightarrow a\,|bb|\,bc,$
$B \rightarrow a\,|bb|\,bc.$

Teorema 6: L(G) = L(G'), unde G' este gramatica ce a fost construita cu algoritmul de mai sus (eliminand regulile unitare).

Demonstratie: Fie w $\in$ L(G). Daca in derivarea propozitiei w se folosess reguli unitare astfelo incat A =>* B din R:

S =>* uAv =>* uBv => uxv =>* w,

unde x $\notin$ V si in derivarea A =>* B s-au folosit numai reguli unitare. De asemenea, in G avem B $\rightarrow$ x. In fiecare dintre aceste situatii in locul derivarii A =>* B putem considera (conform algoritmului) derivarea A => x corespunzatoare regulii A $\rightarrow$ x din R'. Asadar, avem:

S =>* uAv => uxv =>* w.

Dupa inlocuirea tuturor situatiilor de forma celor de mai sus se obtine o derivare a lui w cu reguli din R', ceea ce inseamna ca w $\in$ L(G').

Consideram acum w $\in$ L(G'). Fiecare regula de forma A $\rightarrow$ x folosita in derivarea lui w in G', daca nu este din R, inseamna ca a fost introdusa in R' in situatia in care A =>* B si B $\rightarrow$ x (in R). Inlocuind in derivarea lui w in G' situatiile A => x cu A =>* B => x, obtinem o derivare a lui w in L(G). Asadar, w $\in$ L(G) (q.e.d.).

In concluzie, simplificarea unei gramatici se face in urmatorii pasi:

1. Se elimina λ-regulile
2. Se elimina regulile unitare
3. Se elimina regulile inutile (si variabilele inutile).

Forma normala Chomsky

Definitie 7: O gramatica i.d.c. se spune ca este in **forma normala Chomsky**, daca este formata numai cu reguli de forma:

$$A \to BC \qquad (A, B, C \in V)$$

si reguli de forma:

$$A \to a \qquad (A \in V, a \in \Sigma).$$

Teorema 7: Pentru fiecare gramatica i.d.c. $G = (V, \Sigma, S, R)$ care nu recunoaste cuvantul vid exista o gramatica echivalenta G' in forma normala Chomsky.

Demonstratie: Presupunem fara a reduce generalitatea ca gramatica G nu are λ-reguli si nici reguli unitare. Constructia gramaticii echivalente in forma normala Chomsky se face in doi pasi:

Pasul 1: Pornind de la G se construieste gramatica $G_1 = (V_1, \Sigma, S, R_1)$. Initial consideram $V_1 = V$ si $R_1 = \varnothing$.

Consideram fiecare regula din R:

$$A \to x_1x_2\acute{e}\ .x_n,$$

unde $x_1, x_2, \acute{e}\ ., x_n \in V \cup \Sigma$.

Daca n = 1, inseamna (din presupunerea initiala) ca $x_1 \in \Sigma$. In aceasta situatie regula:

(1) $A \to x_1$

se adauga in R'.

Daca $n \geq 2$, atunci pentru fiecare $x_i = a \in \Sigma$ ($i \in \{1, 2, \acute{e}\ ., n\}$) introducem in V_1 variabila X_a (daca nu este deja introdusa in V_1), iar in R' regula $X_a \to a$. De asemenea, in R' introducem regula:

(2) $A \to B_1B_2\acute{e}\ .B_n,$

unde $B_i = x_i$, daca $x_i \in V$ si $B_i = X_a$, daca $x_i = a \in \Sigma$, pentru orice $i \in \{1, 2, é\ ., n\}$.

Gramatica G_1 contine numai reguli de forma (1) si/sau (2). Este evident ca $L(G) = L(G_1)$.

Pasul 2: Pornind de la gramatica G_1 se construieste gramatica $G_2 = (V_2, \Sigma, S, R_2)$. Pornim cu $V_2 = V_1$ si $R_2 = \varnothing$.

In R_2 introducem regulile din R_1 de forma $A \to a$ si regulile de forma $A \to BC$ (in care in partea dreapta apar exact doua variabile din V_1).

Pentru fiecare regula din R_1 de forma:

$A \to B_1B_2é\ .B_n$,

cu $n > 2$ in V_2 introducem varibilele $Y_1, Y_2, é\ ., Y_{n-2}$, iar in R_2 introducem regulile:

$A \to B_1Y_1$
$Y_1 \to B_2Y_2$
é .
$Y_{n-3} \to B_{n-2}Y_{n-2}$
$Y_{n-2} \to B_{n-1}B_n$

Este evident ca $L(G) = L(G_1) = L(G_2)$ si G_2 este in forma normala Chomsky (q.e.d.).

Exemplu: Sa se gaseasca gramatica in forma normala Chomsky echivalenta cu gramatica i.d.c. G data prin regulile:

$S \to ABa$
$A \to aab$
$B \to Ac$

Gramatica G_1 echivalenta cu G are regulile:

$S \to ABX_a$
$A \to X_aX_aX_b$
$B \to AX_c$
$X_a \to a$
$X_b \to b$
$X_c \to c$,

iar $V_1 = \{S, A, B, X_a, X_b, X_c\}$.

Gramatica in forma normala Chomsky echivalenta cu G este:

$S \to AY_1$
$Y_1 \to BX_a$
$A \to X_aY_2$
$Y_2 \to X_aX_b$
$B \to AX_c$
$X_a \to a$
$X_b \to b$
$X_c \to c,$

iar $V_2 = \{S, A, B, X_a, X_b, X_c, Y_1, Y_2\}$.

Forma normala Greibach

Definitie 8: O gramatica i.d.c. este in **forma normala Greibach**, daca ea are numai reguli de forma:

$A \to ax,$

unde $A \in V$, $a \in \Sigma$ si $x \in V^*$.

Teorema 8: Pentru fiecare gramatica i.d.c. $G = (V, \Sigma, S, R)$ care nu recunoaste cuvantul vid exista o gramatica echivalenta G' in forma normala Greibach.

Exemplu: Sa se gaseasca gramatica in forma normala Greibach echivalenta cu gramatica i.d.c. G data prin regulile:

$S \to AB$
$A \to aAb$
$B \to BB$
$B \to Aa$
$A \to a$
$B \to b$

Intr-un prim pas facem prin substitutii ca partea dreapta a regulilor sa inceapa cu terminale:

$S \rightarrow aAbB|aB$
$A \rightarrow aAb$
$B \rightarrow AaB|bB$ <=> $B \rightarrow aAbaB|aaB|bB$
$B \rightarrow aAba|aa$
$A \rightarrow a$
$B \rightarrow b$

In al doilea pas procedam ca in primul pas al transformarii in forma normala Chomski:

$S \rightarrow aAX_bB|aB$
$A \rightarrow aAX_b$
$B \rightarrow aAX_bX_aB|aX_aB|bB$
$B \rightarrow aAX_bX_a|aX_a$
$X_a \rightarrow a$
$X_b \rightarrow b$
$A \rightarrow a$
$B \rightarrow b$