– CURSUL 9 –

ANALIZA SEMANTICĂ

Există construcții specifice limbajelor de programare care nu pot fi descrise cu ajutorul gramaticilor independente de context. Iată câteva exemple:

1. Fie limbajul $L_1 = \{w\#w | w \in \Sigma^*\}$, Σ alfabet finit, $|\Sigma| \geq 2$, $\# \notin \Sigma$. Acest limbaj abstractizează condiția ca un identificator-variabilă să fie declarat înainte de utilizare. Aici w reprezintă numele variabilei.
   Limbajul L_1 nu este un limbaj independent de context, deci această condiție nu poate fi formalizată cu o gramatică independentă de context.
2. Considerăm limbajul $L_2 = \{a^n b^m a^n c^m | n, m \geq 1\}$ care, de asemenea, nu este independent de context. L_2 abstractizează corespondența între declarația unei funcții cu parametri și utilizarea acesteia cu același număr și aceleași tipuri de parametri actuali ca cei formali (a^n reprezintă numele funcției, b^m reprezintă parametrii formali, iar c^m reprezintă paramatrii actuali ai funcției).

Din exemplele de mai sus desucem că prin analiza sintactică (la baza căreia avem gramatici independente de context/ translatoare stivă) nu se pot verifica condițiile din 1) și 2). Acestea vor fi rezolvate în etapa de analiză semantică.

Pentru un limbaj de programare, verificările semantice constau din:

- verificarea condiției ca un identificator să fie declarat înainte de utilizare;
- verificarea corespondenței între numărul și tipurile parametrilor actuali cu numărul și tipurile parametrilor formali;
- verificarea compatibilității tipurilor operanzilor într-o expresie etc.

Există și în aceste cazuri un model formal care descrie aceste condiții: gramaticile atributate (cu atribute).

GRAMATICI ATRIBUTATE

Definiție. O gramatică atributată (cu atribute) are o structură de forma $GA = (G, A, R, B)$, unde:

- $G = (N, \Sigma, S, P)$ este o gramatică independentă de context, numită gramatică de bază (care abstractizează sintaxa limbajului)

- $A = \bigcup_{X \in \Sigma \cup N} A(X)$, $A(X)$ fiind o mulțime de atribute asociate lui X. Dacă $a \in A(X)$, atunci vom folosi notația clasică $X.a$ sau $X \uparrow a$ pentru atributele sintetizate și $X \downarrow a$ pentru atributele moștenite.
- $R = \bigcup_{p \in P} R(p)$, $R(p)$ fiind o mulțime finită de reguli de atributare asociate producției p.
- $B = \bigcup_{q \in P} B(q)$, $B(q)$ fiind o mulțime finită de predicate (condiții semantice). În plus, dacă $A(X) \cap A(Y) \neq \emptyset \Longrightarrow X = Y$. Pentru fiecare apariție a lui X într-un arbore sintactic, se poate aplica cel mult o regulă pentru a calcula fiecare atribut $a \in A(X)$.

Definiție. Pentru fiecare producție $p: X_0 \to X_1 ... X_n \in P$ mulțimea atributelor definite de p este:

$$A(p) = \{X_i.a | 0 \leq i \leq n, \exists X_i.a \leftarrow f(...) \in R(p)\}$$

unde f este o funcție ale cărei argumente sunt alte atribute.

Atributul $X.a$ este numit sintetizat sau derivat dacă există $p: X \to \alpha \in P$ și $X.a \in A(p)$. Notăm acest atribut cu $X \uparrow a$, care sugerează modul cum este obținută valoarea sa.

Atributul $Y.b$ este numit moștenit dacă există $q: X \to \alpha Y \beta \in P$ și $Y.b \in A(q)$. Notăm acest atribut cu $Y \downarrow b$.

Valorile atributelor sintetizate asociate unui neterminal $X \in N$ se calculează pornind de la atributele asociate nodurilor subarborelui de rădăcină X. Atributele asociate unor terminali (tokeni) sunt întotdeauna sintetizate și reprezintă anumite valori intrinseci asociate acelor tokeni, cum ar fi valoarea unei constante, și de regulă sunt furnizate de analizorul lexical.

Atributele moștenite asociate lui $X \in N$ se calculează top-down, în funcție de atributele asociate nodurilor unui subarbore în care apare X ca descendent, direct sau indirect.

Exemple de atribute:

- pentru identificatori-variabilă: tip, adresă, nume
- pentro o constantă numerică: valoarea
- pentru o expresie: tipul său.

Definiție. Limbajul generat de gramatica atributată $GA = (G, A, R.B)$:

$L(GA) = \{w \in \Sigma^* | w \in L(G)$, pentru orice arbore de derivare T al lui w toate atributele pot fi evaluate și toate predicatele asociate sunt evaluate la $true\}$

Cu alte cuvinte, $L(GA)$ conține toate șirurile corecte din punct de vedere sintactic ($w \in L(G)$) și corecte din punct de vedere semantic.

Exemple.

1) $Exp \to Exp"+"Exp$
   $Exp \to Exp"OR"Exp$
   $Exp \to num$
   $Exp \to "TRUE"$
   $Exp \to "FALSE"$
   Se atributeză gramatica de mai sus astfel încât neterminalul Exp are asociat un atribut $type$ ce poate lua două valori, $type \in \{int, bool\}$ și astfel încât să existe compatibilitate între tipurile operanzilor.

$$\begin{aligned} Exp \uparrow type \to\ & Exp \uparrow type1"+"Exp \uparrow type2 \\ & [type1 == type2; type1 == int; type \leftarrow type1] \\ \to\ & Exp \uparrow type1"\text{OR}"Exp \uparrow type2 \\ & [type1 == type2; type1 == bool; type \leftarrow type1] \\ \to\ & num \uparrow val\ [type \leftarrow int] \\ \to\ & "\text{TRUE}"[type \leftarrow bool] \\ \to\ & "FALSE"\ [type \leftarrow bool] \end{aligned}$$

2) Se dă gramatica cu producțiile:
   $S \to N"."N$
   $N \to NB$
   $N \to B$
   $B \to 0$
   $B \to 1$

   Vom atributa S, N, B astfel încât S să aibă asociat un atribut sintetizat v a cărui valoare va reprezenta valoarea în zecimal a numărului în virgulă mobilă din baza 2 generat de S. Astfel:

$S \uparrow v$ $\quad v - valoarea\ in\ zecimal$
$N \downarrow f \uparrow v \uparrow l$ $\quad f - factorul\ de\ scala\ (departarea\ fata\ de\ punctul\ zecimal)$
$B \downarrow f \uparrow v$ $\quad l - lungimea\ sirului\ de\ biti\ generat\ de\ N$

$$S \uparrow v \to N \downarrow f1 \uparrow v1 \uparrow l1"."N \downarrow f2 \uparrow v2 \uparrow l2$$

$$[v \leftarrow v1 + v2; f1 \leftarrow 1; f2 \leftarrow 2^{-l2}]$$

$$N \downarrow f \uparrow v \uparrow l \to N \downarrow f1 \uparrow v1 \uparrow l1\ B \downarrow f2 \uparrow v2$$

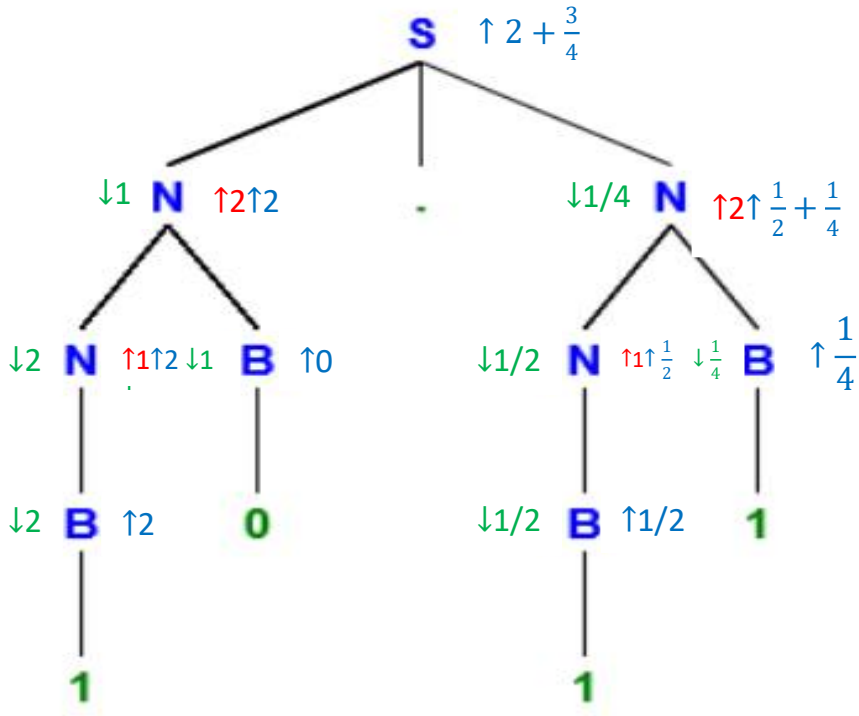
$$[f1 \leftarrow 2 * f; f2 \leftarrow f; v \leftarrow v1 + v2; l \leftarrow l1 + 1]$$

$$B \downarrow f1 \uparrow v1\ [f1 \leftarrow f; v \leftarrow v1; l \leftarrow 1]$$

$B \downarrow f \uparrow v \rightarrow 0\ [v \leftarrow 0]$

$$\rightarrow 1\ [v \leftarrow f]$$

Ordinea de evaluare a atributelor: l bottom-up, apoi f top-down, apoi v bottom-up



3) Pentru gramatica de la punctul 2), o variantă mai simplă de atributare este cea în care folosim doar două atribute sintetizate, v, l cu aceeași semnificație ca la punctul 2), doar că diferă modul de calcul pentru v:

$S \uparrow v$ $\quad v - valoarea\ in\ zecimal$

$N \uparrow v \uparrow l$ $\quad l - lungimea\ sirului\ de\ biti\ generat\ de\ N$

$B \uparrow v$

$S \uparrow v \rightarrow N \uparrow v1 \uparrow l1"."N \uparrow v2 \uparrow l2$

$$[v \leftarrow v1 + v2 \times 2^{-l2}]$$

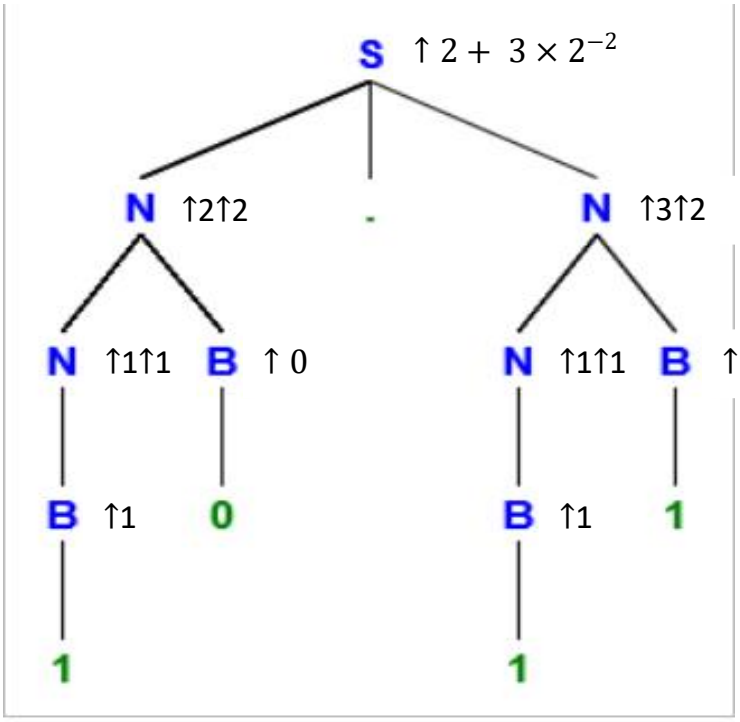
$N \uparrow v \uparrow l \rightarrow N \uparrow v1 \uparrow l1\ B \uparrow v2$

$$[v \leftarrow 2v1 + v2; l \leftarrow l1 + 1]$$

$\rightarrow B \uparrow v1 \quad [v \leftarrow v1; l \leftarrow 1]$

$B \uparrow v \quad \rightarrow 0\ [v \leftarrow 0]$

$\rightarrow 1\ [v \leftarrow 1]$



Algoritm pentru depistarea circularității in gramaticile atributate

Fie gramatica atributată $GA = (G, Atr, R, B)$, unde $G = (N, \Sigma, S, P)$ este gramatica independentă de context de bază. Fie producția $p: X_0 \rightarrow X_1 ... X_n$. Notăm cu $D_p = (V, E)$ graful de dependență asociat lui p, definit prin $V = \{a \in Atr(X_i) | 0 \leq i \leq n\}$, $(a, b) \in E$ dacă și numai dacă există o regulă semantică $b \leftarrow f(..., a, ...)$ asociată lui p. Pentru un arbore de derivare T din G vom nota cu D_T graful orientat obținut prin combinarea grafurilor D_q, unde q este o producție aplicată în T.

Spunem că GA este bine definită dacă pentru orice arbore de derivare T asociat unei derivări din G, D_T, este aciclic.

$Dependency - test(GA)$
{

$for\ (X \in N \cup \Sigma)$
$\mathcal{F}(X) \leftarrow \{graful\ ca\ are\ ca\ varfuri\ atributele\ lui\ X\ si\ nicio\ muchie\}$;
$do\{$
$change \leftarrow false$;
$for\ (fiecare\ p\colon A \rightarrow X_1 \ldots X_n \in P)\{$
$for(G_1 \in \mathcal{F}(X_1), \ldots, G_n \in \mathcal{F}(X_n))\{$
$D \leftarrow D_p$;
$for\ (fiecare\ b \rightarrow c\ din\ G_j, 1 \leq j \leq n)$
$adauga\ muchia\ b \rightarrow c\ in\ graful\ D$;
$if(D\ are\ un\ ciclu)\ error(\ )$;
$else\ \{$
$G \leftarrow graf\ nou\ cu\ noduri\ atributele\ lui\ A\ si\ nicio\ muchie$;
$for(fiecare\ b, c \in Atr(A))$
$if\ (exista\ drum\ in\ D\ de\ la\ b\ la\ c)$
$adauga\ muchia\ b \rightarrow c\ la\ G$;
$if\ (G \notin \mathcal{F}(A))$
$\{\ adauga\ G\ la\ \mathcal{F}(A)$;
$change \leftarrow true;\}$
} // end_else
} // end_for
} // end_for
$\}while(change == true)$ // end_do
} // end_Dependency_test

Gramatici $S-$ atributate

Definiție. O gramatică atributată care are toate atributele sintetizate se numește gramatică $S-$atributată.

Teoremă. Orice gramatică atributată este echivalentă cu o gramatică $S-$atributată.

Algoritm general de evaluare a atributelor într-o gramatică $S-$atributată

```
postorder (n: node)
{
  for (fiecare fiu m al lui n, de la stanga la dreapta)
       postorder(m);
  evalueaza toate atributele asociate lui n;
}
```

Evaluarea atributelor sintetizate intr-un parser de tip LR

- Fiecărui simbol X din stiva de lucru i se asociază valorile tuturor atributelor sintetizate (folosind, de exemplu, o structură de tip $union$).
- Atunci când se efectuează o reducere pentru producția $p: X_0 \to X_1 ... X_n$, valorile sintetizate asociate neterminalului din membrul stâng, X_0, se vor calcula folosind aserțiunile (regulile de atributare) din $R(p)$, care sunt funcții ce depind de atributele asociate simbolurilor $X_1, ..., X_n$ ale căror valori sunt pe stivă.
- După ce se face reducerea, se introduc pe stivă valorile asociate lui X_0.
- În $bison$, valorile atributelor sintetizate asociate lui $X_0, X_1, ..., X_n$ sunt referite cu $$, respectiv $1, ..., $n.

Gramatici $L-$ atributate

Definiție. Spunem că gramatica atributată $GA = (G, A, R, B)$ (unde $G = (N, \Sigma, S, P)$ este gramatică independentă de context de bază) este $L-$atributată dacă pentru orice producție $X_0 \to X_1 ... X_n \in P$ atributele moștenite asociate lui X_i, $1 \leq i \leq n$, depind de:

- atributele moștenite și/sau sintetizate asociate lui $X_1, ..., X_{i-1}$
- atributele moștenite asociate lui X_0.

Algoritm general de evaluare a atributelor unei gramatici $L-$atributate

```
dfvisit (n:node)
{
  for (fiecare fiu m al lui n, de la stanga la dreapta)
        {
         evalueaza atributele mostenite ale lui m;
         dfvisit(m);
        }
  evalueaza atributele sintetizate ale lui n;
}
```

Implementarea gramaticilor L-atributate într-un parser recursiv descendent

Fie gramatica atributată $GA = (G, A, R, B)$, unde $G = (N, \Sigma, S, P)$ este gramatica independentă de context de bază, de tip $LL(1)$.

Pentru fiecare neterminal $A \in N$ se construiește o funcție cu numele A care ca parametri transmiși prin valoare atributele moștenite și ca parametri transmiși prin referință atributele sintetizate.

Pe baza token-ului curent și a mulțimilor $First(\alpha.Follow(A)), A \to \alpha \in P$, se decide ce producție a lui A este aplicată (a se vedea algoritmul recursiv descendent pentru gramatici $LL(1)$ din cursul 4).

Codul asociat producției constă din:

- pentru x terminal (token) care are asociat atributul $\uparrow v$ a cărui valoare este furnizată de către analizorul lexical (*lexer* sau *scanner*), se salvează valoarea atributului într-o variabilă locală, după care se avansează la următorul token (se apelează scanner-ul).
- Pentru un neterminal B se apelează funcția corespendentă.
- Pentru o funcție de evaluare a atributelor, se introduce asignarea corespunzătoare.
- Pentru un predicat, se introduce instrucțiune if cu alternativa $error()$.
- Atributele sintetizate care nu apar în lista de parametri ai lui A sunt implementate ca variabile locale.

Implementarea gramaticilor *L*-atributate într-un parser de tip *LR*

În acest caz sunt necesare unele condiții suplimentare referitoare la producțiile gramaticii astfel încât atunci când se efectuează reducerile, atributele moștenite să poată fi calculate corect.

Gramatica se modifică în felul următor: pentru o producție în care există atribute moștenite, se introduc neterminali noi de forma $M \to \lambda$ (care nu introduc nimic nou din punct de vedere sintactic) exact înaintea unui neterminal A care are asociat un atribut moștenit, care sunt asociați în mod unic cu producția în care apar. Rolul acestor neterminali este de a furniza modul corect de calcul al atributului moștenit.

În $bison$, acești neterminali noi sunt acțiuni interioare, nu apar în mod explicit.

Exemple

1) Se consideră gramatica atributată

$S \to a\ A \uparrow s\ C \downarrow s \uparrow t$
$S \to b\ A \uparrow s\ BC \downarrow s \uparrow t$
$C \downarrow f \uparrow t \to\ \ c[t \leftarrow g(f)]$

Neterminalul C moștenește atributul sintetizat de A. Atunci când facem reducerea $C \to c$, valoarea $C \downarrow s$ este fie în $val[top-1]$, fie in $val[top-2]$, unde val este un vector în care păstrăm valorile atributelor sintetizate asociate simbolurilor de pe stiva parserului $LR(1)$, iar top este indicele simbolului din vârful stivei.

Pentru a rezolva această ambiguitate, este adăugat înaintea lui C un neterminal nou M, asociat în mod unic cu a doua producție a gramaticii, care va moșteni atributul sintetizat de A.

$S \to a\ A \uparrow s\ C \downarrow s \uparrow t$
$S \to b\ A \uparrow s\ BM \downarrow s \uparrow v\ C \downarrow v \uparrow t$
$C \downarrow s \uparrow t \to c\ [t \leftarrow g(s)]$
$M \downarrow s \uparrow t \to [t \leftarrow s]$

Atunci când se va face reducerea $M \to \lambda$, deoarece acest neterminal apare doar în producția $S \to bABMC$, atributul moștenit al lui M își va extrage valoarea de la atributul sintetizat aflat pe stivă cu 2 poziții înaintea lui. Astfel, atunci când se va face reducerea $C \to c$, atributul moștenit asociat lui C va avea ca valoare atributul sintetizat al simbolului ce apare înaintea sa pe stivă, indiferent dacă C apare în producția $C \to aAC$ sau în $C \to bABMC$.

2) Fie producția

$S \to aA \uparrow sC \downarrow m \quad [m \leftarrow h(s)]$.

În acest caz introducem înaintea lui C marcatorul N, asociat unic cu producția $S \to aANC$, și producția $N \to \lambda$, astfel ca producțiile atributate să fie:

$S \to aA \uparrow sN \downarrow s_1 \uparrow v\ C \downarrow m \quad [s_1 \leftarrow h(s), m \leftarrow v]$

$N \downarrow s_1 \uparrow v \to [v \leftarrow s_1]$

Analiza sintactică și evaluarea atributelor moștenite într-un parser $LR(1)$

Input: gramatica atributată $GA = (G, A, R, B)$, unde $G = (N, \Sigma, S, P)$ este gramatica de bază de tip $LL(1)$.

Output. Parser $LR(1)$ care calculează valorile tuturor atributelor în stiva asociată.

- Presupunem că fiecare neterminal X are asociat cel mult un atribut moștenit, m_X, și cel mult unul sintetizat, s_X.
- În cele ce urmează vom considera producția $A \to X_1 ... X_n$ care va fi înlocuită cu și producțiile $A \to M_1 X_1 ... M_n X_n$, $M_j \to \lambda$, unde $M_1, ..., M_n$ sunt neterminali noi asociați în mod unic cu producția $A \to X_1 ... X_n$.

- Asociem pe stivă fiecărui simbol (terminal sau neterminal) valoarea atributului sintetizat (dacă există).
- Atributul sintetizat s_{X_j}, dacă există, apare în stiva parserului LR asociat lui X_j, iar cel moștenit, m_{X_j}, dacă există, va apărea pe stivă asociat cu M_j. Producția atributată va fi $M_j \downarrow m_{M_j} \uparrow s_{M_j} \rightarrow [s_{M_j} \leftarrow f(m_{M_j})]$. Dacă atributul m_A există, $A \neq S$, atunci valoarea sa se găsește pe stivă, fiind asociată cu simbolul dinaintea lui M_1. Dacă $A = S$ are asociat un atribut moștenit, atunci valoarea acestuia trebuie plasată pe stivă inițial, înaintea începerii procesului de parsare, asociat eventual cu simbolul inițial al stivei (să ne reamintim că inițial pe stivă este plasat simbolul 0, corespunzător stării inițiale a AFD care recunoaște mulțimea prefixelor viabile, urmat de perechi $simbol, stare$, astfel că întotdeauna în vârful stivei apare o stare).
- Pentru reducerea numărului de marcatori: dacă X_j nu are asociat niciun atribut moștenit, atunci nu se introduce M_j. De asemeni, dacă m_{X_1} există și este egal cu m_A, atunci putem omite M_1.
- Atunci cînd se efectuează reducerea $M_j \rightarrow \lambda$, știm cu ce producție, unică, este asociat M_j, deci știm unde găsim valorile care intervin în calculul (funcția) valorii atributului asociat lui M_j, s_{M_j}. Deoarece gramatica este L-atributată, valoarea lui s_{M_j} este calculată în funcție de atributele sintetizate și/sau moștenite care apar înaintea lui M_j în producția asociată, deci le regăsim pe stivă.
- Pentru reducerea $A \rightarrow M_1X_1 ... M_nX_n$, vom calcula doar atributul sintetizat s_A, în funcție de atributele asociate lui $M_1, X_1, ..., M_n, X_n$ care sunt pe stivă.
- Atributul m_A este deja plasat pe stivă, după reducere chiar înaintea lui A.
- În Bison, în locul neterminalilor noi $M_1, ..., M_n$ se adaugă acțiunile semantice interioare $\left[s_{M_j} \leftarrow f\left(m_{M_j}\right)\right], 1 \leq j \leq n$.

Propoziție. Dacă gramatica de bază $G = (N, \Sigma, S, P)$ a gramaticii atributate $GA = (G, A, R, B)$ este de tip $LL(1)$, atunci prin introducerea marcatorilor ca în algoritmul explicat mai sus se obține o gramatică $LR(1)$.

Marcatorii sunt neterminali din care se derivă doar λ și care apar o singură dată în corpul unei producții unice. Dacă gramatica de bază este $LL(1)$, putem să determinăm dacă șirul w din intrare este derivat din S într-o derivare care începe cu $S \rightarrow \alpha$ prin analiza primului simbol al lui w (sau a următorului, dacă $w = \lambda$). Astfel, dacă parsăm bottom-up pe w, atunci atunci faptul că un prefix al lui w poate

fi redus la α și apoi la S este cunoscut îndată ce începutul lui w apare pe stivă. În particular, dacă inserăm marcatori oriunde în α, stările LR vor încorpora faptul că acel marcator trebuie să fie acolo, și se va face reducerea lui λ la marcatorul respectiv în momentul potrivit.

Exemplu de gramatică $LR(1)$ pentru care introducerea marcatorilor conduce la o gramatică care nu este $LR(1)$

Fie gramatica atributată cu producțiile:

$S \to L \downarrow c \quad [c \leftarrow 0]$

$L \downarrow c \to L \downarrow c_1 a \quad [c_1 \leftarrow c + 1]$

$L \downarrow c \to \lambda \quad [print\ c]$

Introducem marcatorul M, astfel că gramatica de mai sus devine:

$S \to L \downarrow c \quad [c \leftarrow 0]$

$L \downarrow c \to M \downarrow m \uparrow s\ L \downarrow s\ a \quad [m \leftarrow c + 1]$

$L \downarrow c \to \lambda \quad [print\ c]$

$M \downarrow m \uparrow s \ \to \lambda \quad [s \leftarrow m]$

Gramatica $S \to L,\ \ L \to La|\lambda$ este $LR(1)$, dar gramatica $S \to L, L \to MLa|\lambda, M \to \lambda$ nu este $LR(1)$.

Un exemplu mai complex de atributare a gramaticii sintactice pentru un mini-limbaj like-Pascal

$Z \to$ "PROGRAM" ID ";" "VAR" VH "BEGIN" B "END" "."
$V \to ID$ ":" T ";" V // declararea variabilelor
$\to \lambda$
$T \to$ "INTEGER"
$\to$ "BOOLEAN"
$H \to$ "FUNCTION" ID ":" T "VAR" VH "BEGIN" B "END" ";" H
$\to \lambda$
$B \to ID$ " := " E ";" B // corpul programului/functiei
$\to$ "RETURN" E // apare doar in corpul unei functii, la final
$\to \lambda$ // folosita doar in programul principal
$E \to SC$ // expresie
$C \to$ " = "S // comparatie (expresie booleana)

$\rightarrow \lambda$ // expresie simpla

$S \rightarrow FR$ // Factor urmat de Rest

$R \rightarrow$ " $*$ "FR // expresie de tip INTEGER

$\rightarrow$ "AND" FR // expresie de tip BOOLEAN

$\rightarrow \lambda$

$F \rightarrow$ "(" E ")"

$\rightarrow ID\ A$ // referinta la variabila/functie

$\rightarrow NUM$ // constanta numerica

$\rightarrow$ "TRUE"

$\rightarrow$ "FALSE"

$A \rightarrow$ "(" ")" // referinta la functie

$\rightarrow \lambda$ // referinta la identificator

În acest mini-limbaj există funcții fără parametri, două tipuri de instrucțiuni (asignare și return), două tipuri pentru variabile și funcții (INTEGER, BOOLEAN). Referința la funcție se face prin adăugarea "()" după numele funcției.

Vom atributa această gramatică pentru a verifica dacă:

- variabilele sunt declarate înainte de utilizare
- operanzii care apar în expresii au tipuri compatibile între ei și cu operatorii
- tipul returnat de expresia ce apare în instrucțiunea $RETURN\ E$ coincide cu tipul declarat al funcției în corpul căreia se află.

Pentru aceasta vom implementa ca atribut moștenit tabela de simboluri, în care vom introduce pentru fiecare identificator numărul unic asociat de scanner și o valoare împachetată, $value = tip + ref$, unde:

$$tip = \begin{cases} 1, daca\ tipul\ este\ INTEGER \\ 2, daca\ tipul\ este\ BOOLEAN \end{cases}$$

$$ref = \begin{cases} 30, daca\ este\ referita\ o\ variabila \\ 40, daca\ este\ referita\ o\ functie \end{cases}$$

Astfel, dacă identificatorul $ab1$ are asignată valoarea 42, aceasta înseamnă că $ab1$ este o funcție care returnează o valoare de tip boolean.

Pentru a introduce identificatori în tabelă vom folosi funcția $into$:

$[into \downarrow symbtab \downarrow idn \downarrow value \uparrow newtab]$

Astfel, în tabela curentă, $symbtab$, se introduce identificatorul cu numărul unic asociat idn și valoarea $value$. Se obține o nouă tabelă, $newtab$.

Pentru a extrage identificatori din tabelă vom folosi funcția $from$:

$[from \downarrow symbtab \downarrow idn \uparrow value]$

Din tabela curentă, $symbtab$, se extrage pentru identificatorul cu numărul unic asociat, idn, valoarea asociată, $value$. Tabela rămâne neschimbată.

Vom considera o variabilă globală, lex, care nu apare explicit în gramatica atributată și care reprezintă nivelul lexical al variabilelor din tabelă. Astfel:

- variabilele și funcțiile declarate în programul principal au nivel lexical 0;
- variabilele și funcțiile declarate în funcțiile din programul principal au nivel lexical 1;
- ș.a.m.d.

Pentru limbaje precum C, C++ există doar două nivele lexicale, 0 pentru main și 1 pentru variabilele declarate în funcții.

Atunci când se trece la un nou nivel lexical se va utiliza funcția

$[open \downarrow symbtab \uparrow newtab]$

în care $newtab$ diferă de $symbtab$ doar prin nivelul lexical asociat, care este cu 1 mai mare decât nivelul lexical al lui $symbtab$. Aceste nivele lexicale diferite permit ca identificatori cu același nume să fie utilizați la nivele lexicale diferite, deși este vorba de entități diferite (ca tipuri, adrese în memorie etc.).

Neterminalii gramaticii vor avea asociate atributele:

- $V \downarrow tabIn \uparrow tabOut$; V primește ca atribut moștenit tabela curentă, $tabIn$, și după ce se fac toate declarațiile de variabile se sintetizează $tabOut$.
- $T \uparrow type; type \in \{1,2\}; T$ este neterminalul pentru tip.
- $B \downarrow symbtab \downarrow typeR$; B este neterminalul pentru corpul ($body$) al programului principal sau al unei funcții. B primește ca atribut moștenit tabela de simboluri curentă, $symbtab$, și tipul întors de funcția respectivă (1 sau 2) dacă este utilizat pentru corpul unei funcții sau valoarea 0 când este utilizat pentru programul principal.
- $E \downarrow symbtab \uparrow type$; neterminalul pentru expresii, E, primește ca atribut moștenit tabela de simboluri curentă, $symbtab$, și sintetizează tipul expresiei respective (1 sau 2).

- $S \downarrow symbtab \uparrow type$; neterminalul S este folosit pentru introducerea unei expresii numerice și booleene.
- $C \downarrow symbtab \downarrow typeIn \uparrow typeOut$; neterminalul C este folosit pentru a introduce o expresie simplă ($C \rightarrow \lambda$) sau o comparație ($C \rightarrow$ " $=$ "S); $typeIn$ este tipul moștenit de la primul termen al comparației, care este o expresie simplă.
- $R \downarrow symbtab \downarrow type$; neterminalul pentru restul expresiei, R, primește ca atribut moștenit tipul de la factorul anterior.
- $F \downarrow symbtab \uparrow type$; neterminalul pentru factor.
- $A \downarrow value \uparrow type$; A primește ca atribut moștenit valoarea extrasă din tabelă pentru identificatorul ce apare înaintea lui A; sintetizează $type$.
- $H \downarrow tabIn \uparrow tabOut$;
- $ID \uparrow idn$; idn este numărul unic asociat de scanner identificatorului.

Gramatica atributată este:

$Z \rightarrow$ "PROGRAM" $ID \uparrow idn$ ";" "VAR" $V \downarrow \emptyset \uparrow tabnext$

$H \downarrow tabnext \uparrow tabH$ "BEGIN" $B \downarrow tabH \downarrow 0$ "END" "."

$V \downarrow tabIn \uparrow tabOut \rightarrow ID \uparrow idn$ ":" $T \uparrow type$ ";"
$[into \downarrow tabIn \downarrow idn \downarrow type + 30 \uparrow tabNext]$
$V \downarrow tabNext \uparrow tabOut$
$\rightarrow [tabOut \leftarrow tabIn]$

$T \uparrow type \rightarrow$ "INTEGER" $[type \leftarrow 1]$
$\rightarrow$ "BOOLEAN"$[type \leftarrow 2]$

$H \downarrow tabIn \uparrow tabOut \rightarrow$"FUNCTION"$\uparrow ID \uparrow idn$ ":" $T \uparrow type$";"
$[into \downarrow tabIn \downarrow idn \downarrow type + 40 \uparrow tabNext]$
$[open \downarrow tabNext \uparrow newTab]$
"VAR" $V \downarrow newTab \uparrow vtab$ $H \downarrow vtab \uparrow bodyTab$
"BEGIN" $B \downarrow bodyTab \downarrow type$ "END" ";"
$H \downarrow tabNext \uparrow tabOut$ // acelasi nivel lexical ca inainte
// de $open$
$[tabOut \leftarrow tabIn]$

$B \downarrow symbtab \downarrow typeR \rightarrow ID \uparrow idn$ " $:=$ " $E \downarrow symbtab \uparrow type$ ";"
$[from \downarrow symbtab \downarrow idn \uparrow value]$
$[type == value - 30]$
$B \downarrow symbtab \downarrow typeR$
$\rightarrow$ "RETURN" $E \downarrow symbtab \uparrow type$
$[type == typeR]$

$\rightarrow [typeR == 0]$ // apare doar in progr principal

$E \downarrow symbtab \uparrow type \rightarrow S \downarrow symbtab \uparrow stype\ C \downarrow symbtab \downarrow stype \uparrow type$

$C \downarrow symbtab \downarrow typeIn \uparrow typeOut \rightarrow$ " = "$S \downarrow symbtab \uparrow stype$
$[stype == typeIn; typeOut \leftarrow 2]$
$\rightarrow [typeOut \leftarrow typeIn]$

$S \downarrow symbtab \uparrow type \rightarrow F \downarrow symbtab \uparrow type\ R \downarrow symbtab \downarrow type$

$R \downarrow symbtab \downarrow type \rightarrow$ " $*$ "$F \downarrow symbtab \uparrow ftype$
$[ftype == type; type == 1]$
$R \downarrow symbtab \downarrow ftype$
$\rightarrow$ "AND" $F \downarrow symbtab \uparrow ftype$
$[ftype == type; type == 2]$
$R \downarrow symbtab \downarrow ftype$
$\rightarrow \lambda$

$F \downarrow symbtab \uparrow type \rightarrow$ "(" $E \downarrow symbtab \uparrow type$ ")"
$\rightarrow ID \uparrow idn$
$[from \downarrow symbtab \downarrow idn \uparrow value]$
$A \downarrow value \uparrow type$
$\rightarrow NUM \uparrow val\ [type \leftarrow 1]$
$\rightarrow$ "TRUE" $[type \leftarrow 2]$
$\rightarrow$ "FALSE" $[type \leftarrow 2]$

$A \downarrow value \uparrow type \rightarrow$ "(" ")" $[value\ div\ 10 == 4;\ type \leftarrow value - 40]$
$\rightarrow [value\ div\ 10 == 3;\ type \leftarrow value - 30]$

Observație. Gramatica mini-limbajului like-Pascal de mai sus este $LL(1)$, deci este și $LR(1)$. Prin adăugarea marcatorilor gramatica obținută este $LR(1)$ și deci este posibilă implementarea unui parser LR în care pot fi păstrate pe stivă și evaluate toate atributele, moștenite sau sintetizate.