Construirea tabelelor de analiză sintactică în varianta SLR

O gramatică pentru care se pot construi tabele de ASLR prin tehnica SLR se numeşte **gramatică SLR**.

Se numeşte **element LR(0)** al unei gramatici G (sau, pe scurt, **element**), o producţie a acelei gramatici în care s-a plasat un punct într-o anumită poziţie a părţii drepte. De exemplu, producţiei A→XYZ i se pot ataşa următoarele patru elemente.

A→**.**XYZ A→X**.**YZ A→XY**.**Z A→XYZ**.**

Producţiei A→ε îi corespunde un singur element: A→**.**

Elementele pot fi reprezentate printr-o pereche de numere întregi corespunzătoare numărului producţiei şi respectiv poziţiei punctului. Elementele se utilizează în procesul de analiză sintactică pentru a arăta cât s-a văzut, la un moment dat, dintr-o anumită producţie. De exemplu, A→**.**XYZ arată că se aşteaptă ca în intrare să existe un şir derivat din XYZ iar A→X**.**YZ arată că tocmai s-a văzut în intrare un şir derivabil din X şi se aşteaptă, în continuare, un şir derivabil din YZ.

Metoda SLR de construire a tabelei de analiză se bazează pe construirea prealabilă, din gramatica dată, a unui AFD care să recunoască prefixele viabile. Stările acestui AFD sunt mulţimi de elemente. Elementele individuale pot fi considerate ca stările AFN ce recunoaşte, de asemenea, prefixele viabile iar gruparea elementelor în mulţimi, care trebuie efectuată aici, este similară construirii submulţimilor de la transformarea AFN în AFD.

La baza construirii tabelelor de analiză prin metoda SLR stă o colecţie de mulţimi de elemente LR(0) numită **colecţia LR(0) canonică**. Aceasta se obţine cu ajutorul a două funcţii, numite **închidere** şi **salt** aplicate asupra **gramaticii augmentate**, corespunzătoare gramaticii iniţiale.

Dacă G este o gramatică având simbolul de start S, atunci gramatica sa augmentată G' constă din gramatica G, completată cu un nou simbol de start S' şi cu producţia S'→S. Scopul introducerii acestei producţii suplimentare este de a permite oprirea analizei sintactice şi anunţarea acceptării şirului de intrare, atunci când se efectuează reducerea cu producţia S'→S.

Operaţia de închidere

Dacă I este o mulţime de elemente pentru o gramatică G, atunci *închidere* (I) este, de asemenea, o mulţime de elemente, construită din I prin următoarele două reguli:

1) Iniţial, toate elementele din I se introduc în *închidere* (I).
2) Dacă A→α**.**Bβ este un element din *închidere* (I) şi B→γ este o producţie a gramaticii G, atunci se adaugă la *închidere* (I) elementul B→**.**γ (dacă nu este deja prezent). Această regulă se aplică în mod repetat până nu se mai adaugă elemente noi la *închidere* (I).

Intuitiv, prezenţa elementului A→α**.**Bβ în *închidere* (I) arată că, la un moment dat în procesul de analiză ne aşteptăm să vedem în continuare, în intrare, un subşir derivabil din Bβ**.** Dacă în plus, B→γ este o producţie a gramaticii, înseamnă că s-ar putea să vedem mai întâi, în acest moment, un subşir derivabil din γ, motiv pentru care se va include în *închidere* (I) şi elementul B→**.**γ

.

Exemplu: Se consideră gramatica augmentată pentru expresii:

E'→E
E→E+T | T
T→T∗F | F
F→(E) | **id**

şi se ia I={[E'→**.**E]}

⇒închidere (I)={[E'→**.**E], [E→**.**E+T], [E→**.**T], [T→**.**T∗F], [T→**.**F], [F→**.**(E)], [F→**.id**]}

Cele două reguli de construire a funcţiei închidere (I) pot fi cuprinse în următorul algoritm:

```
function închidere (I);
  begin
    J:=I;
    repeat
        for ∗ fiecare element [A→α.Bβ] din J şi fiecare producţie B→γ din G
                astfel încât [B→.γ]∉ J do
              ∗ adaugă [B→.γ] la J
    until ∗ nu se mai pot adăuga elemente noi la J;
    return J
  end;
```

În vederea implementării funcției închidere se recomandă crearea unui tablou cu elemente logice numit *adăugat*, indexat de neterminalele gramaticii G, astfel încât *adăugat* [B] primeşte valoarea *true* atunci când se adaugă elementele B→.γ pentru acel neterminal B.

Din modul de calcul al funcției *închidere* rezultă că dacă se adaugă o producție B cu punct la extremitatea stângă, atunci vor fi adăugate toate producțiile pentru B. În concluzie, nu este necesar ca elementele de forma B→**.**γ să se rețină în totalitate în *închidere* (I), fiind suficientă o listă de neterminale B ale căror producții trebuie adăugate în acest mod. Rezultă că elementele dintr-o mulțime pot fi împărțite în două clase astfel:

1) **Elemente nucleu**: care includ elementul inițial S'→**.**S şi toate elementele ale căror puncte nu sunt la extremitatea stângă.
2) **Elemente nenucleu**: cele care au punctul la extremitatea stângă.

Mulțimile de elemente care apar în procesul de calcul prin metoda SLR se obțin întotdeauna prin închiderea unei mulțimi de *elemente nucleu* iar elementele care se adaugă prin închidere sunt *elemente nenucleu*. În concluzie, mulțimile de elemente care interesează în această metodă pot fi reprezentate în memorie doar prin elemente nucleu, cele nenucleu urmând să fie regenerate ori de câte ori ar fi necesar.

Operația salt

Argumentele funcției *salt* (I,X) au următoarea semnificație:

I – o mulțime de elemente LR(0);

X – un simbol al gramaticii G(neterminal sau terminal).

Rezultatul funcției *salt* (I,X) este reprezentat de închiderea mulțimii tuturor elementelor [A→αX**.**β] pentru care [A→α**.**Xβ] este în *I*. Intuitiv, aceasta înseamnă că, dacă *I* este mulțimea de elemente valide pentru un anumit prefix viabil γ, atunci *salt* (I,X) este mulțimea de elemente valide pentru prefixul viabil γX.

Exemplu: I={[E'→E**.**], [E→E**.**+T]}
salt (I,+)={[E→E+**.**T], [T→**.**T∗F], [T→**.**F], [F→**.**(E)], [F→**.id**]}

Procedeul de calcul este următorul: se caută în mulțimea *I* acele elemente care au punctul în stânga simbolului +. Din aceste elemente se obțin elemente noi mutând punctul în dreapta simbolului +, după care se determină închiderea acestei din urmă mulțimi.

Construirea colecției LR(0) canonice

Algoritmul pentru construirea colecției LR(0) canonice, notată cu C, este următorul:

procedure colecție (G');
  begin
    C :={închidere ({[S'→**.**S]})};
    repeat
      for ∗ fiecare mulțime de elemente I din C şi fiecare simbol gramatical X astfel încât *salt* (I,X) nu e vidă şi nu e prezentă în C **do**
        ∗ adaugă *salt* (I,X) la C
    until ∗ nu se mai pot adăuga la C mulțimi noi de elemente
  end;

Exemplu: Se consideră gramatica pentru expresii din exemplele anterioare. Colecția canonică de mulțimi de elemente LR(0) este următoarea:

I_0=închidere({[E'→.E]})
E'→**.**E
E→**.**E+T
E→**.**T
T→**.**T∗F
T→**.**F
F→**.**(E)
F→**.id**

I_5=salt(I_0,id)
F→**id.**

I_6=salt(I_1,+)
E→E+**.**T
T→**.**T∗F
T→**.**F
F→**.**(E)
F→**.id**

I_7=salt(I_2,∗)
T→T∗**.**F
F→**.**(E)
F→**.id**

I_1=salt(I_0,E)
E'→E**.**
E→E**.**+T

I_2=salt(I_0,T)
E→T**.**
T→T**.**∗F

I_3=salt(I_0,F)
T→F**.**

I_8=salt(I_4,E)
F→(E**.**)
E→E**.**+T

salt(I_4,T)≡I_2
salt(I_4,F)≡I_3
salt(I_4,( )≡I_4
salt(I_4,id)≡I_5

I_9=salt(I_6,T)
E→E+T**.**
T→T**.**∗F

salt(I_6,F)≡I_3
salt(I_6,( )≡I_4
salt(I_6,id)≡I_5

I_4=salt(I_0,( )
F→(**.**E)
E→**.**E+T
E→**.**T
T→**.**T∗F
T→**.**F
F→**.**(E)
F→**.id**

element nucleu

elemente nenucleu

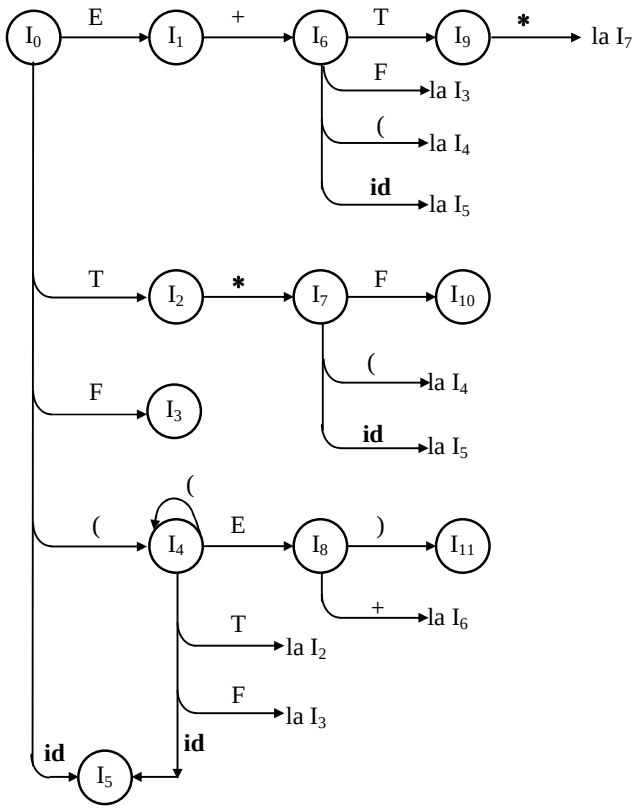
I_{10}=salt(I_7,F)
T→T∗F**.**

salt(I_7,( )≡I_4
salt(I_7,id)≡I_5

I_{11}=salt(I_8,))
F→(E)**.**

salt(I_8,+)≡I_6
salt(I_9,∗)≡I_7

Funcția *salt* pentru întreaga colecție canonică se poate reprezenta sub forma unui AFD ca în **fig**. Dacă I_0 este stare inițială şi orice stare a AFD este considerată stare finală, atunci automatul recunoaşte exact prefixele viabile ale gramaticii G'. Explicația acestui fapt este următoarea: se consideră un AFN ale cărui stări sunt chiar elementele individuale, înzestrat cu tranziții de la $[A\rightarrow\alpha.X\beta]$ la $[A\rightarrow\alpha X.\beta]$ etichetate cu X şi cu tranziții de la $[A\rightarrow\alpha.B\beta]$ la $[B\rightarrow.\gamma]$ etichetate cu ε. *Închidere* (I) este exact funcția *ε-închidere* aplicată mulțimii de stări corespunzătoare a AFN în procedeul de transformare a AFN în AFD echivalent. În concluzie, *salt* (I,X) va furniza tranziția de ieşire din starea I pentru simbolul X în AFD construit din AFN prin aplicarea algoritmului de conversie bazat pe construirea de submulțimi. În acest context, procedura *colecție* (G') corespunde exact construirii funcției de tranziție pentru AFD rezultat şi echivalent cu AFN inițial, definit ca mai sus.



Se spune că **elementul** $A\rightarrow\beta_1.\beta_2$ **este valid** pentru un prefix viabil $\alpha\beta_1$ dacă există o derivare $S'\overset{*}{\underset{d}{\Rightarrow}}\alpha Aw\overset{*}{\underset{d}{\Rightarrow}}\alpha\beta_1\beta_2 w$. În general, un element va fi valid pentru mai multe prefixe viabile. Faptul că $A\rightarrow\beta_1.\beta_2$ este valid pentru $\alpha\beta_1$ indică dacă trebuie făcută *deplasare* sau *reducere* în situația în care în stivă apare prefixul viabil $\alpha\beta_1$, şi anume:

a) **dacă $\beta_2\neq\varepsilon$** $\Rightarrow$ capătul nu este complet deplasat în stivă $\Rightarrow$ **<u>deplasare</u>**
b) **dacă $\beta_2=\varepsilon$** $\Rightarrow$ $A\rightarrow\beta_1$ ar trebui să fie capăt $\Rightarrow$ **<u>reducere</u>**

Dacă există două sau mai multe elemente valide pentru acelaşi prefix viabil, examinarea lor ar putea să ne ducă la concluzii diferite privind operația (deplasare sau reducere) ce trebuie efectuată. Aceste situații sunt **conflicte de analiză sintactică**. Unele dintre aceste conflicte se pot rezolva luând în considerare următorul simbol de intrare iar altele aplicând metode de analiză mai puternice.

Pentru fiecare prefix viabil care poate apărea în stiva unui ASLR se poate calcula mulțimea de elemente valide corespunzătoare. O teoremă importantă a teoriei ASLR, care se poate şi demonstra, arată că mulțimea de elemente valide pentru un prefix viabil γ este chiar mulțimea de elemente la care se ajunge din starea inițială, pe un drum etichetat γ în AFD construit din colecția canonică de mulțimi de elemente cu tranzițiile date de funcția *salt*. În esență, mulțimea de elemente valide cuprinde toată informația utilă care poate fi culeasă din stivă.

Exemplu: Se consideră din nou gramatica pentru expresii, ale cărei mulțimi de elemente şi funcție *salt* au fost deja calculate. Pentru această gramatică, şirul "E+T∗" este un prefix viabil. După citirea elementelor şirului, AFD ajunge din starea I_0 în starea I_7. Starea I_7 conține elementele $T\rightarrow T*.F$ $F\rightarrow.(E)$ $F\rightarrow.\mathbf{id}$ care sunt toate elementele valide pentru prefixul E+T∗, aşa cum rezultă din următoarele trei derivări dreapta:

$E'\Rightarrow E\Rightarrow E+T\Rightarrow E+T*F\Rightarrow E+T*\mathbf{id}\Rightarrow E+T*F*\mathbf{id}$
$E'\Rightarrow E\Rightarrow E+T\Rightarrow E+T*F\Rightarrow E+T*(E)$
$E'\Rightarrow E\Rightarrow E+T\Rightarrow E+T*F\Rightarrow E+T*\mathbf{id}$

Prima derivare confirmă validitatea lui $T\rightarrow T*.F$, cea de a doua confirmă validitatea elementului $F\rightarrow.(E)$ iar a treia, validitatea lui $F\rightarrow.\mathbf{id}$ pentru prefixul viabil E+T∗. Se poate şi arăta că pentru acest prefix viabil nu există alte elemente valide.

Algoritmul de construire a tabelei de analiză

Din AFD care recunoaşte prefixele viabile se pot construi cele două părți, *acțiune* şi *salt*, ale tabelei de analiză sintactică prin metoda SLR. Algoritmul prezentat în continuare se poate aplica unei clase destul de largă de gramatici. Sunt însă şi situații în care el eşuează: produce tabele cu intrări multiplu definite.

Se porneşte de la gramatica **G**, pentru care se obține gramatica augmentată **G'**. Din **G'** se construieşte colecția LR(0) canonică, notată **C**. Din colecția **C**, pe baza algoritmului următor, se construiesc tabelele *acțiune* şi *salt*:

1. Se construieşte colecția de mulțimi de elemente LR(0) pentru G' : $C=\{I_0, I_1, ..., I_n\}$
2. Din mulțimea I_i, a colecției se construieşte starea **i** a analizorului sintactic. Acțiunile de analiză pentru starea **i** se determină astfel:
   a) Dacă $[A\rightarrow\alpha.a\beta]\in I_i$ şi $salt(I_i,a)=I_j$ atunci *acțiune* [i,a] se pune pe valoare "depl. j" (prescurtat d_j); **a**–terminal.
   b) Dacă $[A\rightarrow\alpha.]\in I_i$ atunci *acțiune* [i,a] ia valoarea "reduce cu $A\rightarrow\alpha$" (prescurtat r_k), unde $a\in URM(A)$; $A\neq S'$.
   c) Dacă $[S'\rightarrow S.]\in I_i$ atunci *acțiune* [i,$]=acceptă.

Dacă prin regulile de mai sus se generează acțiuni în conflict (intrări multiplu definite ale tabelei *acțiune*) se spune că gramatica nu este SLR(1) iar algoritmul nu poate construi tabela de analiză sintactică.

3. Tranzițiile *salt* pentru starea **i** se vor construi pentru toate neterminalele A, utilizând următoarea regulă:
   Dacă $salt(I_i,A)= I_j$ atunci *salt* [i,A]=j
   (funcția) (tabela)
4. Toate intrările (elementele din tabele) nedefinite prin **regulile 2)** şi **3)** se consideră intrări de eroare.
5. Starea inițială a analizorului sintactic este construită din mulțimea de elemente care conține elementul [S'→**.**S].

Exemplu: Construirea tabelei SLR pentru gramatica expresiilor din exemplele anterioare.

Colecția canonică de mulțimi de elemente LR(0) a fost determinată anterior. Mai întâi considerăm mulțimea de elemente I_0: E'→**.**E
E→**.**E+T
E→**.**T
T→**.**T∗F
T→**.**F
F→**.**(E)
F→**.id**

Elementul F→**.**(E) dă naştere intrării *acțiune* [0,(]=d_4 (**regula 2**) iar elementul F→**.id** conduce la acțiune *acțiune* [0,**id**]=d_5 (**regula 2**). Celelalte elemente din I_0 nu produc acțiuni.

Considerăm apoi I_1 : E'→E**.**
E→E**.**+T

Primul element produce *acțiune* [1,$]=acceptă (**regula 2.c)**, prezentată mai sus) iar al doilea determină acțiune [1,+]=d_6.

În continuare se consideră I_2 : E→T**.**
T→T**.**∗F

Deoarece URM(E)={**$**,**+**,**)**}, din primul element
⇒ *acțiune* [2,$]=*acțiune* [2,+]=*acțiune* [2,)]=reduce E→T.
Din al doilea element ⇒ *acțiune* [2,∗]= d_7.
Continuând în acest mod se vor obține tabelele *acțiune* şi *salt*.

Observație: Orice gramatică SLR(1) este neambiguă. Există însă multe gramatici neambigue care nu sunt SLR(1).