5

ANALIZA SINTACTICĂ ASCENDENTĂ

Se va prezenta un tip general de analiză numit **cu deplasare şi reducere (ASDR)**. Aceasta încearcă să construiască arborele sintactic pentru şirul de intrare începând de la frunze spre rădăcină (de jos în sus–ascendent). Procesul se poate privi ca reducerea unui şir de terminale, "w", la simbolul de start al gramaticii. În fiecare pas al reducerii un subşir particular, corespunzător părţii dreapta a unei producţii este înlocuit cu simbolul din partea stângă a producţiei respective. Dacă subşirul este corect ales, la fiecare pas se va parcurge, în sens invers, o derivare dreapta.

Exemplu: Se consideră gramatica:

$S \rightarrow aABe$ $\quad$ $A \rightarrow Abc \mid b$ $\quad$ $B \rightarrow d$

Propoziţia "abbcde" poate fi redusă la S în următorii paşi:

abbcde $\quad$ aAbcde $\quad$ aAde $\quad$ aABe $\quad$ S

Înlocuirile s-au realizat căutând subşiruri care să corespundă părţii dreapta a unei producţii şi înlocuind subşirurile respective cu partea stângă a acelei producţii. Şirul de reduceri urmăreşte, în sens invers, următoarea derivare dreapta:

$$S \underset{d}{\Rightarrow} aABe \underset{d}{\Rightarrow} aAde \underset{d}{\Rightarrow} aAbcde \underset{d}{\Rightarrow} abbcde$$

5.1 Noţiuni de bază

Un **capăt al unui şir** este un subşir care corespunde părţii drepte a unei producţii şi a cărui reducere la neterminalul din partea stângă a producţiei reprezintă un pas în parcurgerea în sens invers a unei derivări dreapta. În multe cazuri, subşirul stâng β care corespunde părţii drepte a unei producţii $A \rightarrow \beta$ nu este un **capăt** deoarece o reducere prin producţia $A \rightarrow \beta$ produce un şir care nu poate fi redus în continuare, până la simbolul de start al gramaticii.

Exemplu: înlocuirea lui *b* cu A în pasul 2 al exemplului de mai sus

aAbcde→aAAcde (care nu poate fi redus la S)

⇒ Trebuie dată o definiţie mai precisă pentru noţiunea de "capăt".

Un **capăt** al unei forme propoziţionale drepte γ constă, formal dintr-o **producţie** A→β şi o **poziţie** în γ în care şirul β poate fi găsit şi înlocuit prin A pentru a produce forma propoziţională anterioară a unei derivări dreapta a lui γ.

Dacă $S \overset{*}{\underset{d}{\Rightarrow}} \alpha A w \underset{d}{\Rightarrow} \alpha\beta w$, atunci A→β şi poziţia care urmează lui α, defineşte un capăt al lui $\alpha\beta w$. Subşirul *w*, situat în dreapta unui capăt, conţine numai terminale.

În cazul în care gramatica este ambiguă, este posibil să existe mai multe derivări dreapta ale şirului $\alpha\beta w$ ca atare pot exista mai multe capete. Dacă gramatica este neambiguă, capătul corespunzător unei forma propoziţionale este unic.

Grafic, un capăt β, al unei forme propoziţionale $\alpha\beta w$, se poate reprezenta astfel:

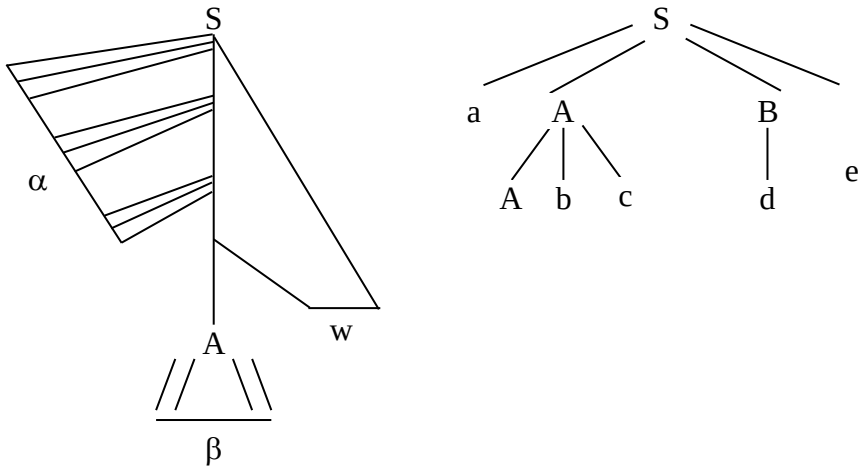

Fig.5.1 Capătul A→β în Arborele Sintactic pentru $\alpha\beta w$

Capătul reprezintă cel mai din stânga subarbore complet format dintr-un nod şi toţi fii săi. A este nodul interior cel mai de jos şi cel mai din stânga având toţi fiii prezenţi în arbore. Reducerea lui β la *A* în $\alpha\beta w$, numită **fasonarea** capătului β, constă în îndepărtarea fiilor lui A din arborele sintactic.

Exemplu. Se consideră gramatica:

(1) E→E+E (2) E→E∗E (3) E→(E) (4) E→**id**

şi derivarea dreaptă:

$$E \underset{d}{\Rightarrow} \underline{E+E} \underset{d}{\Rightarrow} E+\underline{E*E} \underset{d}{\Rightarrow} E+E*\underline{\mathbf{id}_3} \underset{d}{\Rightarrow} E+\underline{\mathbf{id}_2}*\mathbf{id}_3 \underset{d}{\Rightarrow} \underline{\mathbf{id}_1}+\mathbf{id}_2*\mathbf{id}_3$$

Sublinierile marchează capetele din fiecare formă propoziţională. Se remarcă şi aici faptul că şirurile care apar la dreapta unui capăt conţin numai simboluri terminale.

Gramatica dată fiind ambiguă, se poate obţine acelaşi şir şi prin următoarea derivare dreapta:

$$E \underset{d}{\Rightarrow} \underline{E*E} \underset{d}{\Rightarrow} \mathbf{E*}\underline{\mathbf{id}_3} \underset{d}{\Rightarrow} \underline{E+E}*\mathbf{id}_3 \underset{d}{\Rightarrow} E+\underline{\mathbf{id}_2}*\ \mathbf{id}_3 \underset{d}{\Rightarrow} \underline{\mathbf{id}_1}+\mathbf{id}_2*\mathbf{id}_3$$

În forma propoziţională E+E∗$\mathbf{id}_3$ sunt două capete, $\underline{E+E}$ şi $\underline{\mathbf{id}_3}$, corespunzând primei şi respectiv celei de-a doua derivări prezentate. Ca şi în exemplul prezentat la gramatici ambigue, existenţa celor două derivări semnifică priorităţi relative diferite ale operaţiilor de ∗ şi + (în primul caz, ∗ mai prioritar ca +).

Fasonarea capetelor

Procesul de parcurgere în sens invers a unei derivări dreapta se numeşte **fasonarea capetelor**. Se porneşte de la un şir de terminale, notat cu **w**, care urmează să fie analizat. Dacă w aparţine limbajului descris prin gramatica dată atunci el se poate obţine după un pas oarecare **n** într-o derivare dreapta încă necunoscută:

$$S=\gamma_0 \underset{d}{\Rightarrow} \gamma_1 \underset{d}{\Rightarrow} \gamma_2 \underset{d}{\Rightarrow} \ldots \underset{d}{\Rightarrow} \gamma_{n-1} \underset{d}{\Rightarrow} \gamma_n = w$$

Pentru a reconstitui această derivare în ordine inversă, se localizează capătul β_n în forma propoziţiei γ_n, după care se înlocuieşte β_n cu partea stângă a unei producţii $A_n \rightarrow \beta_n$, obţinându-se forma propoziţională dreapta γ_{n-1}. Apoi se continuă acest proces pentru γ_{n-1}, ş.a.m.d. Procesul se termină cu succes dacă, în final, se obţine ca formă propoziţională simbolul de start **S**.

Exemplu: se consideră gramatica din exemplul anterior şi şirul de intrare $\mathbf{id_1+id_2*id_3}$. Secvenţa de reduceri prezentată în figură este inversul secvenţei din prima derivare a exemplului anterior (cu $*$ prioritar faţă de +).

Forma propoziţională dreapta	Capăt	Producţia pentru reducere
$\mathbf{id_1+id_2*id_3}$	$\mathbf{id_1}$	$E \rightarrow \mathbf{id}$
$E+\mathbf{id_2*id_3}$	$\mathbf{id_2}$	$E \rightarrow \mathbf{id}$
$E+E*\mathbf{id_3}$	$\mathbf{id_3}$	$E \rightarrow \mathbf{id}$
$E+E*E$	$E*E$	$E \rightarrow E*E$
$E+E$	$E+E$	$E \rightarrow E+E$
E		

Fig. 5.2. Reduceri efectuate de un analizor sintactic cu deplasare şi reducere

5.2 Implementarea cu stivă a analizei sintactice de tip deplasare – reducere (ASDR)

Pentru implementarea AS prin fasonarea capetelor, trebuie rezolvate două probleme:

- localizarea subşirului care urmează să fie redus, într-o formă propoziţională dreaptă;
- alegerea producţiei care se aplică, dacă gramatica are mai multe producţii cu aceeaşi parte dreaptă.

Ca structură de date de bază se poate utiliza pentru implementare o stivă în care se păstrează simbolurile gramaticale şi un tampon de intrare care conţine şirul de analizat, **w**. Baza stivei şi extremitatea dreaptă a şirului de intrare va fi marcată prin simbolul $. Iniţial, stiva este goală (conţine doar $) iar la intrare este şirul **w** întreg:

STIVA	**INTRARE**
... ab	cd ...$

STIVA	INTRARE
$	w$

ANSIN deplasează în stivă zero sau mai multe simboluri de intrare până când în vârful stive apare un capăt, β. Apoi se reduce β la partea stângă a producţiei corespunzătoare şi se continuă în mod ciclic aceste operaţii până când se detectează o eroare sau până când stiva conţine doar simbolul de start iar intrarea este vidă:

STIVA	INTRARE
$S	w$

În acest caz AS se opreşte şi semnalează terminarea cu succes a analizei.

Exemplu. Se urmăresc pas cu pas acţiunile ANSIN deplasare – reducere la analiza şirului de intrare $id_1+id_2*id_3$ (gramatica expresiilor din exemplul anterior).

STIVA	INTRARE	ACŢIUNE
(1) $	$id_1+id_2*id_3$$	deplasare
(2) $$id_1$	$+id_2*id_3$$	reducere E→**id**
(3) $E	$+id_2*id_3$$	deplasare
(4) $E+	$id_2*id_3$$	deplasare
(5) $E+$id_2$	$*id_3$$	reducere E→**id**
(6) $E+E	$*id_3$$	deplasare
(7) $E+E*	$id_3$$	deplasare
(8) $E+E*$id_3$	$$	reducere E→**id**
(9) $E+E*E	$$	reducere E→E*E
(10) $E+E	$$	reducere E→E+E
(11) $E	$$	acceptă

Fig. 5.3. Configuraţiile ANSIN deplasare – reducere pentru intrarea $id_1+id_2*id_3$

ANSIN cu deplasare şi reducere execută următoarele patru acţiuni:

1) **Deplasare**:Simbolul de intrare următor este introdus în vârful stivei
2) **Reducere**: Se realizează în situaţia în care extremitatea dreaptă a unui capăt se află în vârful stivei. Analizorul trebuie să localizeze extremitatea sa stângă, să stabilească neterminalul cu care se înlocuieşte capătul şi să realizeze efectiv reducerea lui.
3) **Acceptare**:Analizorul semnalează terminarea cu succes a analizei.
4) **Eroare**: Analizorul descoperă apariţia unei erori şi se apelează o rutină de revenire.

Există o motivaţie importantă care justifică utilizarea stivei la acest tip de analiză sintactică: capătul va fi întotdeauna în vârful stivei şi nu în interior. Modalitatea de alegere a acţiunii, astfel încât analizorul să lucreze corect, depinde de tipul concret de analiză. În continuare, pe parcursul acestui capitol, vor fi abordate două astfel de tehnici: precedenţa operatorilor şi analiza sintactică LR.

Prefixe viabile

Mulţimea prefixelor formelor propoziţionale dreapta care pot apărea în stiva unui ASDR se numesc **prefixe viabile**. Altfel spus, un prefix viabil este un prefix al

unei forme propoziţionale dreapta care nu continuă dincolo de extremitatea dreaptă a celui mai din dreapta capăt al acelei forme propoziţionale. Conform acestei definiţii este întotdeauna posibil să se adauge simboluri terminale la extremitatea unui prefix viabil, pentru a obţine o formă propoziţională dreapta. De aceea, pe parcursul analizei, nu va apărea nici o eroare atâta timp cât porţiunea din intrare văzută până la un anumit punct poate fi redusă la un prefix viabil.

5.3 Analizoare sintactice LR

Tehnica LR reprezintă o metodă eficientă de AS ascendentă care poate fi aplicată pentru o clasă largă de GIC. Tehnica este numită ASLR(k). Pentru k=1, el poate fi omis.

Principalele *avantaje* ale analizei LR sunt:

- se pot elabora analizoare LR practic pentru toate construcţiile de limbaje exprimabile prin GIC;
- deşi este cea mai generală metodă fără reveniri, dintre cele cunoscute, ASLR poate fi implementată la fel de eficient ca şi alte metode de tipul deplasare – reducere;
- un ASLR poate detecta o eroare imediat ce este posibil, într-o baleiere de la stânga la dreapta a şirului de intrare (mai repede şi mai exact decât alte metode).

Principalele *dezavantaje* ale metodei sunt:

- cantitatea relativ mare de memorie necesară;
- volumul mare de muncă pentru construcţia manuală a unui analizor, pentru gramatica unui limbaj de programare tipic.

Există însă în prezent generatoare de ASLR care primesc la intrare o GIC şi produc la ieşire tabelele de analiză necesare analizorului. Dacă gramatica conţine ambiguităţi sau alte construcţii dificil de analizat într-o baleiere stânga – dreapta a intrării, generatorul poate localiza aceste construcţii şi poate informa proiectantul compilatorului.

În prezent sunt utilizate trei variante de construire a tabelei de ASLR pentru o gramatică:

1) LR simplu (SLR): cea mai uşor de implementat dar şi cea mai slabă (pot fi analizate doar o clasă mai redusă de gramatici);

2) LR canonic (LR): cea mai puternică dar şi cea mai costisitoare (efort + memorie);

3) LR cu anticipare (LALR): varianta de compromis, intermediară; corespunde cel mai bine LP uzuale.

5.3.1 Algoritmul de analiză sintactică LR

Schema de principiu a unui ASLR este următoarea:

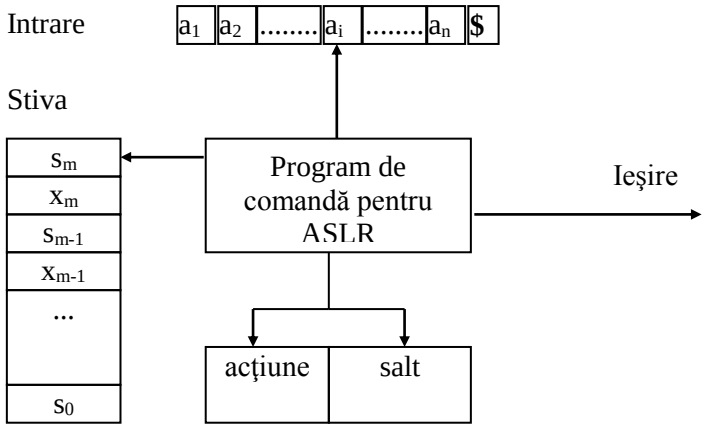

Fig. 5.4. Modelul unui ASLR

Un ASLR constă din: - tampon de intrare
- stivă
- un program de comandă
- o tabelă de analiză cu două părţi { acţiune / salt
- ieşire

Programul de comandă este acelaşi pentru toate ASLR; ceea ce diferă de la o variantă de analizor la alta sunt doar tabelele de analiză. Programul de analiză citeşte, unul câte unul, caracterul din tamponul de intrare. În stivă se memorează şiruri de forma:

s_0 x_1 s_1 x_2 s_2 .... x_m s_m

cu sau s_m în vârf

Fiecare x_i este un simbol gramatical şi fiecare s_i este un simbol numit **stare**. Se consideră că fiecare stare conţine, în rezumat, informaţia prezentă în stivă sub acea stare. Combinaţia dintre simbolul de stare din vârful stivei (s_m) şi simbolul de intrare curent (a_i), utilizată ca indici în tabela de analiză, determină comportarea analizorului (decizia deplasare – reducere). Într-o implementare concretă, simbolurile gramaticale nu trebuie să apară în stivă (sunt suficiente stările). Prezenţa lor contribuie doar la înţelegerea comportării analizorului.

Programul de comandă lucrează astfel: La început se consultă *acţiune* $[s_m, a_i]$, care poate să conţină una din următoarele valori:

1) **deplasează** şi acoperă cu starea *s*;
2) **reduce** prin producţia $A \rightarrow \beta$;
3) **acceptă** şirul;
4) **eroare**.

Tabela "salt" se comportă ca o funcţie care primeşte ca argument o stare şi un simbol gramatical şi produce o stare nouă (valorile din această tabelă sunt stări). Această tabelă este funcţia de tranziţie a unu AFD care recunoaşte prefixele viabile ale

gramaticii date, G. Starea iniţială a acestui AFD este starea pusă iniţial în vârful stivei (s_0).

Se numeşte **configuraţie** a unui ASLR o pereche în care prima componentă este conţinutul momentan al stivei iar a doua componentă este partea rămasă neparcursă din intrare:

$$(s_0\ x_1\ s_1 \ldots x_m\ s_m, a_i\ a_{i+1}\ \ldots\ a_n\ \$)$$

Următoarea operaţie efectuată de analizor este determinată de citirea următorului simbol de intrare a_i, şi de starea din vârful stivei, s_m, care conduce la consultarea tabelei "acţiune". În funcţie de conţinutul elementului *acţiune* $[s_m,a_i]$, rezultă în final una din următoarele configuraţii:

1) Dacă *acţiune* $[s_m, a_i]$="deplasare *s*", se realizează, deplasarea lui a_i din intrare în stivă, urmată de acoperirea lui a_i în stivă, cu starea $s=salt[s_m, a_i]$; a_{i+1} devine noul simbol de intrare. Prin aceste mişcări se obţine configuraţia:

$$(s_0\ x_1\ s_1 \ldots x_m\ s_m a_i\ s,\ a_{i+1}\ a_{i+2} \ldots\ a_n\ \$)$$

2) Dacă *acţiune* $[s_m, a_i]$="reducere $A \rightarrow \beta$" atunci se obţine configuraţia:

$$(s_0\ x_1\ s_1 \ldots x_{m-r}\ s_{m-r}\ As,\ a_i\ a_{i+1} \ldots\ a_n\ \$)$$

unde $s= salt[s_{m-r}, A]$ iar r este $|\beta|$ (lungimea lui β).

S-au extras 2r simboluri din stivă (**r** simboluri de stare +**r** simboluri gramaticale), expunând starea s_{m-r}, completând stiva cu A şi s. Simbolul de intrare curent nu este modificat. Ieşirea constă în executarea acţiunii semantice asociate cu producţia de reducere (în exemplele de mai jos – o simplă tipărire a producţiei utilizate).

3) Dacă *acţiune* $[s_m, a_i]$="**acceptare**", analiza sintactică se termină.

4) Dacă *acţiune* $[s_m, a_i]$="**eroare**", se va apela o rutină de revenire din erori, iar configuraţia rezultată depinde de activitatea acestei rutine.

Algoritmul de analiză este prezentat schematic mai jos:

Intrare: un şir de intrare **w** şi o tabelă de ASLR cu funcţiile **acţiune** şi **salt** corespunzătoare gramaticii G.

Ieşire: Dacă $\mathbf{w} \in L(G) \Rightarrow$ o analiză ascendentă pentru w altfel $\Rightarrow$ o indicaţie de eroare.

Metoda: Se porneşte de la configuraţia:

$$(s_0, w\$)$$

$\uparrow$ în stivă $\uparrow$ în tamponul de intrare

după care se execută programul de mai jos, până se întâlneşte o acţiune de **acceptare** sau **eroare**.

∗ poziţionează *ip* la primul simbol de intrare;
repeat forever begin
∗ fie *s* starea din vârful stivei şi *a* simbolul indicat de *ip*;

if acţiune [s,a]=deplasează *s'* **then begin**

∗ introdu *a* apoi *s'* în vârful stivei; {deplasează şi acoperă cu *s'*}

∗avansează *ip* la următorul simbol de intrare

end

else if acţiune [s,a]=reduce A→β **then begin**

∗ extrage 2∗│β│ simboluri din stivă;

∗ fie *s'* starea rămasă, după extragere, în vârful stivei;

∗ introdu *A*, apoi *salt* [s', A] în stivă;

∗ tipăreşte producţia A→β

end

else if acţiune [s,a]=acceptă **then return**

else eroare()

end

Exemplu: Se consideră gramatica pentru expresii:

(1) `E→E+T`

(2) `E→T`

(3) `T→T*F`

(4) `T→F`

(5) `F→(E)`

(6) `F→`**`id`**

Tabelele (funcţiile) acţiune şi salt corespunzătoare sunt:

stare	acţiune						salt		
	id	+	∗	(	)	$	E	T	F
0	d_5			d_4			1	2	3
1		d_6				acc			
2		r_2	d_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	d_5			d_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	d_5			d_4				9	3
7	d_5			d_4					10
8		d_6			d_{11}				
9		r_1	d_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

Codurile pentru acţiuni au următoarele semnificaţii:

1) d_i – deplasează şi introdu în stivă starea *i*;
2) r_j – reduce prin producţia cu numărul *j*;
3) acc – acceptă;
4) spaţiu – eroare

Observaţie: Valoarea lui `salt[s,a]`, pentru a–terminal, se găseşte în `acţiune[s,a]`, legată cu acţiunea de deplasare. În consecinţă, tabela *salt* are prevăzute coloane doar pentru neterminale.

În continuare se prezintă conţinutul stivei şi al intrării precum şi acţiunile efectuate de analizor la analiza şirului de intrare: **id∗id+id.**

STIVA	INTRARE	ACŢIUNE
(1) 0	**id∗id+id**$	depl (5)
(2) 0 **id** 5	∗**id+id**$	red (6) : F→**id**
(3) 0 F 3	∗**id+id**$	red (4) . T→F
(4) 0 T 2	∗**id+id**$	depl (7)
(5) 0 T 2 ∗ 7	**id+id**$	depl (5)
(6) 0 T 2 ∗ 7 **id** 5	+**id**$	red (6) F→**id**
(7) 0 T 2 ∗ 7 F 10	+**id**$	red (3) T→T ∗ F
(8) 0 T 2	+**id**$	red (2) E→T
(9) 0 E 1	+**id**$	depl (6) :
(10) 0 E 1 + 6	**id**$	depl (5)
(11) 0 E 1 + 6 **id** 5	$	red (6) : F→**id**
(12) 0 E 1 + 6 F 3	$	red (4) . T→F
(13) 0 E 1 + 6 T 9	$	red (1) . E→E+T
(14) 0 E 1	$	acceptă

5.3.2 Gramatici LR

O gramatică pentru care se poate construi o tabelă de analiză sintactică pentru ASDR se numeşte gramatică LR. Există GIC care nu sunt LR dar, pentru construcţiile tipice ale limbajelor de programare uzuale, acestea pot fi în general evitate sau înlocuite prin producţii echivalente care se încadrează în clasa LR. Intuitiv, pentru ca o gramatică să fie LR este suficient ca un ASDR, parcurgând şirul de intrare de la stânga la dreapta, să fie capabil să recunoască toate capetele, atunci când ele apar în vârful stivei. Pentru aceasta, un ASLR nu trebuie să baleieze întreaga stivă întrucât simbolul de stare din vârful stivei conţinând toată informaţia necesară.

Posibilitatea de recunoaştere a unui capăt cunoscând numai simbolurile gramaticale din stivă este echivalentă cu existenţa unui automat finit care, prin citirea simbolurilor gramaticale din stivă, de la vârf spre bază, poate localiza capătul în vârful stivei. Acest automat finit este în esenţă funcţia *salt* a tabelei ASLR. Automatul nu trebuie însă să parcurgă stiva la fiecare mişcare. Simbolul de stare memorat în vârful stivei corespunde stării în care ar ajunge automatul finit de recunoaştere a capetelor în urma citirii simbolurilor gramaticale din stivă, de la bază la vârf. Din această cauză ASLR nu are nevoie decât de starea din vârful stivei pentru a "cunoaşte" conţinutul stivei şi pentru a lua decizia corectă de continuare a analizei.

Alte surse de informaţii utilizate de ASLR în luarea deciziilor deplasare – reducere sunt următoarele *k* simboluri din intrare. O gramatică ce poate fi analizată de un ASLR ce examinează până la *k* simboluri de intrare la fiecare mişcare se numeşte gramatică LR(k). de interes practic sunt cazurile k=0 şi k=1 (Exemplu: tabela de acţiuni din exemplul anterior utilizează un simbol de anticipare, deci k=1).

Diferenţa de principiu între gramaticile LL şi LR este următoarea: pentru ca o gramatică să fie LR(k) analizorul trebuie să fie capabil să recunoască apariţia în stivă a părţii drepte a unei producţii, în stivă fiind chiar acea parte dreaptă sau tot ce derivă din ea, plus *k* simboluri de anticipare în şirul de intrare; în cazul unei gramatici LL(k), analizorul trebuie să recunoască utilizarea unei producţii doar pe baza primelor *k*

simboluri din subşirul derivat în partea dreaptă a producţiei respective. Rezultă că cerinţa ca o gramatică să fie LR(k) este mult mai puţin restrictivă decât în cazul LL(k) iar clasa gramaticilor LR este mai cuprinzătoare decât a gramaticilor LL.

5.4 Întrebări şi probleme

1. Definiţi analiza sintactică ascendentă.
2. Ce se înţelege prin *capăt*? Dar prin *fasonarea capetelor*?
3. Descrieţi operaţiile pe care le efectuează un analizor sintactic cu deplasare şi reducere.
4. Ce avantaj are implementarea cu stivă a analizei sintactice cu deplasare şi reducere?
5. Descrieţi structura şi funcţionarea unui analizor sintactic LR.
6. Ce este o gramatică LR (k)?

Bibliografie

1. Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman: Compilers. Principles, Techniques and Tools, Addison-Wesley Publishing Company, 1986

2. Dick Grune, Henri E. Bal, Ceriel J.H. Jacobs, Koen Langendoen: Modern Compiler Design, John Wiley, 2003

3. Horia Ciocârlie: Limbaje de programare. Concepte fundamentale, Editura de Vest, 2007

4. Irina Athanasiu: Limbaje formale şi compilatoare, Universitatea Politehnica Bucureşti, 1992

5. Luca-Dan Şerbănaţi: Limbaje de programare şi compilatoare, Editura Academiei, Bucureşti, 1987

6. Teodor Rus: Mecanisme formale pentru specificarea limbajelor, Editura Academiei, 1983

7. David A. Watt, Deryck F. Brown: Programming Language Processors in Java - Compilers and Interpreters, Prentice Hall, 2000

8. Gabriel V. Orman: Limbaje formale, Editura Tehnică, Bucureşti, 1982

9. Alexandru Mateescu, Dragoş Vaida: Structuri matematice discrete. Aplicaţii, Editura Academiei, Bucureşti, 1989

10. Carmen De Sabata: Limbaje formale şi translatoare - îndrumător de laborator, Casa Cărţii de Ştiinţă, Timişoara, 1999

11. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns – Şabloane de proiectare, Editura Teora, 2002