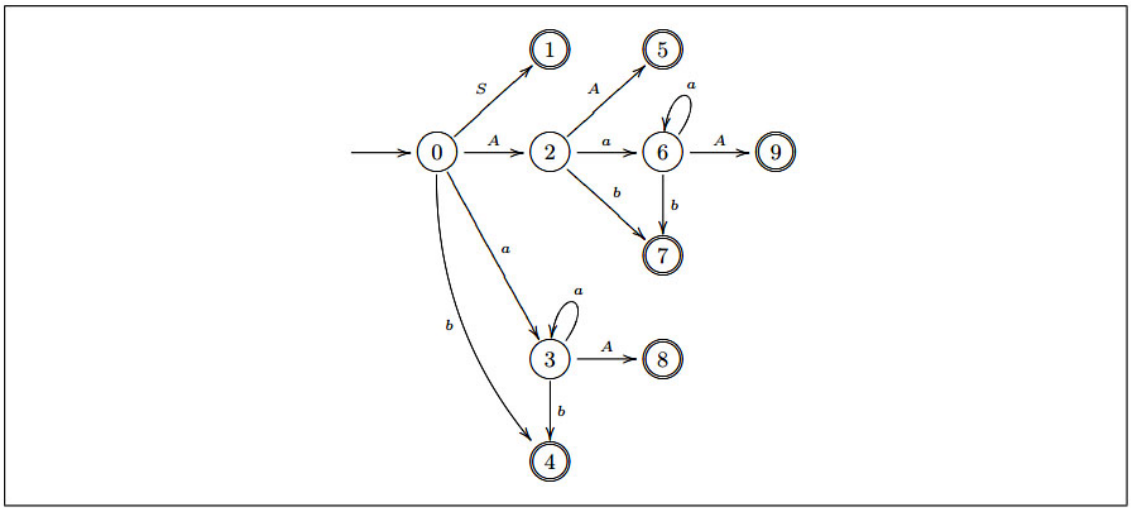
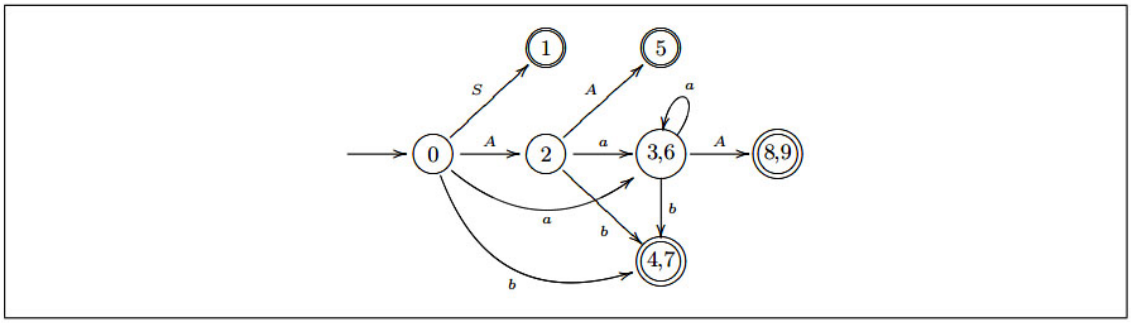
LALR(1) elemzés

LALR: Look-Ahead LR parser

Fontos az állapotok számának csökkentése.

Az *LR*(1)-elemek kanonikus halmazai között vannak olyan halmazpárok, hogy az egyik halmazban levő minden *LR*(1)-elemnek van egy megfelelője a másik halmazban, úgy, hogy ezeknek az elemeknek a magjuk azonos, és legfeljebb csak az előreolvasási szimbólumokban különböznek. Egyesítsük ezeket a halmazpárokat.





Ha a $\mathcal{H}_i$ és a $\mathcal{H}_j$ halmazok egyesíthetők, akkor legyen $\mathcal{K}_{[i,j]} = \mathcal{H}_i \cup \mathcal{H}_j$.

Végezzük el az *LR*(1)-kanonikus halmazok összes lehetséges egyesítését, az indexek átsorszámozása után az így kapott $\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_n$ halmazokat nevezzük *egyesített LR*(1) *kanonikus halmazoknak,* vagy *LALR*(1)*-kanonikus halmazoknak*.

Ezekből az egyesített halmazokból létrehozható elemzőt fogjuk majd *LALR*(1) elemzőnek nevezni.

Példa. A példában szereplő kanonikus halmazok közül a következőket lehet egyesíteni:

$\mathcal{H}_3$ és $\mathcal{H}_6$, $\mathcal{H}_4$ és $\mathcal{H}_7$, $\mathcal{H}_8$ és $\mathcal{H}_9$.

$$\mathcal{H}_0 = \left\{ [S' \to .S, \#], [S \to .AA, \#], [A \to .aA, a/b], [A \to .b, a/b] \right\}$$

$$\mathcal{H}_1 = \left\{ [S' \to S., \#] \right\}$$

$$\mathcal{H}_2 = \left\{ [S \to A.A, \#], [A \to .aA, \#], [A \to .b, \#] \right\}$$

$$\mathcal{H}_3 = \left\{ [A \to a.A, a/b], [A \to .aA, a/b], [A \to .b, a/b] \right\}$$

$$\mathcal{H}_4 = \left\{ [A \to b., a/b] \right\}$$

$$\mathcal{H}_5 = \left\{ [S \to AA., \#] \right\}$$

$$\mathcal{H}_6 = \left\{ [A \to a.A, \#], [A \to .aA, \#], [A \to .b, \#] \right\}$$

$$\mathcal{H}_7 = \left\{ [A \to b., \#] \right\}$$

$$\mathcal{H}_8 = \left\{ [A \to aA., a/b] \right\}$$

$$\mathcal{H}_9 = \left\{ [A \to aA., \#] \right\}$$

Az ábrán is látható, hogy az összevonható halmazok az automatában azonos, vagy legalábbis hasonló részstruktúrát alkotnak.

A *read* függvény az egyesített halmazokra nem okozhat problémát, azaz ha

$$\mathcal{K} = \mathcal{H}_1 \cup \mathcal{H}_2 \cup \ldots \cup \mathcal{H}_k,$$

$$\textbf{\textit{read}}(\mathcal{H}_1, X) = \mathcal{H}_1^{'}, \textbf{\textit{read}}(\mathcal{H}_2, X) = \mathcal{H}_2^{'}, \ldots, \textbf{\textit{read}}(\mathcal{H}_k, X) = \mathcal{H}_k^{'},$$

$$\mathcal{K}' = \mathcal{H}_1^{'} \cup \mathcal{H}_2^{'} \cup \ldots \cup \mathcal{H}_k^{'},$$

akkor

$$\textbf{\textit{read}}(\mathcal{K}, X) = \mathcal{K}'.$$

Ezt a következőképpen lehet belátni. A *read* függvény értelmezése szerint a *read*$(\mathcal{H}, X)$ csak a $\mathcal{H}$ *LR*(1)-elemeinek magjaitól függ, és nem függ az előreolvasási szimbólumoktól. Így, mivel a $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k$ halmazokban az *LR*(1)-elemek magjai azonos halmazokat alkotnak, a

$$\textbf{\textit{read}}(\mathcal{H}_1, X), \textbf{\textit{read}}(\mathcal{H}_2, X), \ldots, \textbf{\textit{read}}(\mathcal{H}_k, X)$$

***LR*(1)-elemeinek magjaiból alkotott halmazok is azonosak, tehát ezek a halmazok is egyesíthetők egy $\mathcal{K}'$ halmazba, és így valóban *read*$(\mathcal{K}, X) = \mathcal{K}'$.**

Az *LR*(1)-elemek kanonikus halmazainak egyesítése után azonban az egyesített halmazon belül maguk az *LR*(1)-elemek okozhatnak problémát. Tegyük fel, hogy

$\mathcal{K}_{[i,j]} = \mathcal{H}_i \cup \mathcal{H}_j$**.**

- ***Léptetés-léptetés konfliktus* az összevonás után nem léphet fel. Ha**

  $$[A \to \alpha.a\beta, b] \in \mathcal{H}_i,$$

  $$[B \to \gamma.a\delta, c] \in \mathcal{H}_j,$$

  akkor az összevonás után az a szimbólumra továbbra is egy léptetést írunk elő, és a fentiekben láttuk, hogy a *read* függvény sem okoz problémát, azaz a *read*$(\mathcal{K}_{[i,j]}, a)$ éppen a *read*$(\mathcal{H}_i, a) \cup$*read*$(\mathcal{H}_j, a)$-val egyenlő.

- **Ha $\mathcal{H}_i$ kanonikus halmazban egy $[A \to \alpha.a\beta, b]$, a $\mathcal{H}_j$-ben egy $[B \to \gamma., a]$ elem szerepelne, akkor az egyesítés után az a szimbólum miatt egy inadekvát állapotot kapnánk, *léptetés-redukálás konfliktus* lépne fel. Ez az eset azonban sohasem állhat fenn, mivel ekkor mindkét elemnek szerepelnie kell mind a $\mathcal{H}_i$, mind a $\mathcal{H}_j$ halmazban, legfeljebb csak az előreolvasási szimbólumokban különbözhetnek, hiszen ezért tudtuk egyesíteni őket. Tehát a $\mathcal{H}_j$ halmazban is kell lennie egy $[A \to \alpha.a\beta, c]$ elemnek. Ekkor pedig a tétel alapján a nyelvtan nem lenne *LR*(1) nyelvtan; már a $\mathcal{H}_j$ halmazból léptetés-redukálás konfliktus következne, az a szimbólumot előreolvasva nem lehetne eldönteni, hogy az elemzésben milyen műveletet kell alkalmazni.**
- **Az összevonás után azonban *redukálás-redukálás konfliktus* előfordulhat, az *LR*(1) nyelvtan tulajdonságai ezt nem zárják ki. A következő példában egy ilyen esetet mutatunk be.**

Példa. **Tekintsük a $G' = (\{S', S, A, B\}, \{a, b, c, d, e\}, P', S')$ nyelvtant, ahol a helyettesítési szabályok:**

$S' \to S$

$S \to aAd \mid bBd \mid aBe \mid bAe$

$A \to c$

$B \to c$

A nyelvtan *LR*(1) nyelvtan. Az ac járható prefixre az

$\{[A \to c., d], [B \to c., e]\}$,

valamint a bc járható prefixre az

$\{[A \to c., e], [B \to c., d]\}$

***LR*(1)-elemek egy-egy kanonikus halmazt alkotnak.**

A két halmaz egyesítése után redukálás-redukálás konfliktus lép fel. Ha a bemenő szimbólum d vagy e, a c mondatnyél azonosítható, de nem dönthető el, hogy az $A \to c$ és a $B \to c$ szabály szerinti redukciók közül melyiket kell végrehajtani.

Most az *LALR*(1) elemző táblázatainak kitöltési szabályait adjuk meg. Miután meghatároztuk az *LR*(1)-elemek

$$\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$$

kanonikus halmazait, egyesítsük egy halmazba azokat a kanonikus halmazokat, amelyekben az *LR*(1)-elemek magjaiból alkotott halmazok azonosak. Legyenek ezek a halmazok

$$\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n \quad (n \leq m).$$

Az *action* és a *goto* táblázatok méretének a meghatározására és a táblázatok kitöltésére a $\mathcal{K}_i \quad (1 \leq i \leq n)$ halmazokat kell használni, a táblázatok kitöltésének módszere teljesen megegyezik az *LR*(1) elemzőnél leírtakkal. A táblázatokat *LALR*(1) *elemző táblázatoknak* nevezzük.

Ha a G' kiegészített nyelvtanra a LALR(1) elemző táblázatok kitöltése konfliktusmentes, akkor a nyelvtant LALR(1) nyelvtannak nevezzük.

Az *LALR*(1) elemző működése az *LR*(1) elemző működésével egyezik meg.

Példa.

Az előző példában szereplő nyelvtan
$G' = (\{S', S, A\}, \{a, b\}, P', S')$ **helyettesítési szabályai:**
(0) $S' \to S$
(1) $S \to AA$
(2) $A \to aA$
(3) $A \to b$

a táblázata pedig

állapot	*action*			*goto*	
	a	b	#	S	A
0	$s3$	$s4$		1	2
1			*elfogad*		
2	$s6$	$s7$			5
3	$s3$	$s4$			8
4	$r3$	$r3$			
5			$r1$		
6	$s6$	$s7$			9
7			$r3$		
8	$r2$	$r2$			
9			$r2$		

(0) $S' \to S$
(1) $S \to AA$
(2) $A \to aA$
(3) $A \to b$

A $\mathcal{H}_i$ és a $\mathcal{H}_j$ kanonikus halmazok egyesítéséből származó $\mathcal{K}_{[i,j]}$ halmazhoz tartozó állapotot jelöljük $[i, j]$ -vel.

A nyelvtanhoz a következő *LALR*(1)-elemző táblázatot lehet elkészíteni.

állapot	*action*			*goto*	
	a	b	#	S	A
0	$s\,[3,6]$	$s\,[4,7]$		1	2
1			$elfogad$		
2	$s\,[3,6]$	$s\,[4,7]$			5
$[3,6]$	$s\,[3,6]$	$s\,[4,7]$			$[8,9]$
$[4,7]$	$r3$	$r3$	$r3$		
5			$r1$		
$[8,9]$	$r2$	$r2$	$r2$		

Az *LALR*(1)-táblázatok kitöltése konfliktusmentes, a nyelvtan tehát egy *LALR*(1) nyelvtan.

Átszámozhatjuk az állapotokat, hogy könnyebben tudjuk kezelni a táblázatot:

állapot	*action*			*goto*	
	a	*b*	#	*S*	*A*
0	*s* [3, 6]	*s* [4, 7]		1	2
1			*elfogad*		
2	*s* [3, 6]	*s* [4, 7]			5
[3, 6]	*s* [3, 6]	*s* [4, 7]			[8, 9]
[4, 7]	*r*3	*r*3	*r*3		
5			*r*1		
[8, 9]	*r*2	*r*2	*r*2		

állapot	*action*			*goto*	
	a	*b*	#	*S*	*A*
0	*s*3	*s*4		1	2
1			*elfogad*		
2	*s*3	*s*4			5
3	*s*3	*s*4			6
4	*r*3	*r*3	*r*3		
5			*r*1		
6	*r*2	*r*2	*r*2		

Példa. Elemezzük az előző példában megadott táblázat felhasználásával az *abb#* szöveget.

állapot	*action*			*goto*	
	a	*b*	#	*S*	*A*
0	*s* [3, 6]	*s* [4, 7]		1	2
1			*elfogad*		
2	*s* [3, 6]	*s* [4, 7]			5
[3, 6]	*s* [3, 6]	*s* [4, 7]			[8, 9]
[4, 7]	*r*3	*r*3	*r*3		
5			*r*1		
[8, 9]	*r*2	*r*2	*r*2		

			szabály
(#0, *abb#*)	$\xrightarrow{s[3,6]}$	(#0*a* [3, 6] , *bb#*)	
	$\xrightarrow{s[4,7]}$	(#0*a* [3, 6] *b* [4, 7] , *b#*)	
	$\xrightarrow{r3}$	(#0*a* [3, 6] *A*[8, 9], *b#*)	$A \to b$
	$\xrightarrow{r2}$	(#0*A*2, *b#*)	$A \to aA$
	$\xrightarrow{s[4,7]}$	(#0*A*2*b* [4, 7] , #)	
	$\xrightarrow{r3}$	(#0*A*2*A*5, #)	$A \to b$
	$\xrightarrow{r1}$	(#0*S* 1, #)	$S \to AA$
	$\xrightarrow{\textit{elfogad}}$	***O.K.***	

Ugyanaz átszámozással:

állapot	*action*			*goto*	
	a	b	#	S	A
0	$s3$	$s4$		1	2
1			$elfogad$		
2	$s3$	$s4$			5
3	$s3$	$s4$			6
4	$r3$	$r3$	$r3$		
5			$r1$		
6	$r2$	$r2$	$r2$		

				szabály
$(\#0,$	$abb\#)$	$\xrightarrow{s3}$	$(\#0a3,\quad bb\#)$	
		$\xrightarrow{s4}$	$(\#0a3b4,\quad b\#)$	
		$\xrightarrow{r3}$	$(\#0a3A6,\quad b\#)$	$A \to b$
		$\xrightarrow{r2}$	$(\#0A2,\quad b\#)$	$A \to aA$
		$\xrightarrow{s4}$	$(\#0A2b4,\quad \#)$	
		$\xrightarrow{r3}$	$(\#0A2A5,\quad \#)$	$A \to b$
		$\xrightarrow{r1}$	$(\#0S\,1,\quad \#)$	$S \to AA$
		$\xrightarrow{\textbf{elfogad}}$	***O.K.***	

Az *LALR*(1) nyelvtanok egyben *LR*(1) nyelvtanok is, mint az a fenti példából is látható, de ez fordítva nem áll fenn. Például
$G' = (\{S', S, A, B\}, \{a, b, c, d, e\}, P', S')$**, ahol a helyettesítési szabályok:**
$S' \to S$
$S \to aAd \mid bBd \mid aBe \mid bAe$

$A \to c$
$B \to c$

olyan nyelvtan, amely *LR*(1), de nem *LALR*(1) nyelvtan.

A programnyelvek generálhatók *LALR*(1) nyelvtannal, a programnyelvek fordítóprogramjaiban leggyakrabban alkalmazott elemzési módszer az *LALR*(1) elemzés. Az *LALR*(1) elemző előnye az *LR*(1) elemzővel szemben az, hogy táblázatainak mérete lényegesen kisebb.

Például, a Pascal nyelvre az *LALR*(1) táblázatok néhányszor száz sort tartalmaznak, míg az *LR*(1) elemző táblázatai több ezer sorból állnak.

Szintaktikai elemzés — összefoglalás

– felülről lefelé (top down) elemzők

- **– LL(1) elemző**
- **– rekurzív leszállás módszere**

– alulról felfelé (botom up) elemzők

- **– LR(1), LALR(1) elemzők**
- **– precedencia (elsőbbségi) nyelvtanon alapuló elemzés**

Példák

LL(1) elemzés

$$S \longrightarrow aS \mid A$$
$$A \longrightarrow bAc \mid d \mid \epsilon$$

$$FOLLOW_1(S) = \{\#\}$$
$$FOLLOW_1(A) = \{c, \#\}$$

$$FIRST_1(aS\ FOLLOW_1(S)) = \{a\}$$
$$FIRST_1(A\ FOLLOW_1(S)) = \{b, d, \#\}$$

$$FIRST_1(bAc\ FOLLOW_1(A)) = \{b\}$$
$$FIRST_1(d\ FOLLOW_1(A)) = \{d\}$$
$$FIRST_1(\epsilon\ FOLLOW_1(A)) = \{c, \#\}$$

	a	b	c	d	#
S	$(aS,1)$	$(A,2)$		$(A,2)$	$(A,2)$
A		$(bAc,3)$	$(\varepsilon,5)$	$(d,4)$	$(\varepsilon,5)$
a	pop				
b		pop			
c			pop		
d				pop	
#					$elfogad$

Az elemzés az abc szóra a következő:

$$
\begin{array}{lll}
(abc\#,\ S\#,\ \varepsilon) & \xrightarrow{(aS,1)} & (\ abc\#,\ aS\#,\ 1\ ) \\
& \xrightarrow{pop} & (\ bc\#,\ S\#,\ 1\ ) \\
& \xrightarrow{(A,2)} & (\ bc\#,\ A\#,\ 12\ ) \\
& \xrightarrow{(bAc,3)} & (\ bc\#,\ bAc\#,\ 123\ ) \\
& \xrightarrow{pop} & (\ c\#,\ Ac\#,\ 123\ ) \\
& \xrightarrow{(\varepsilon,5)} & (\ c\#,\ c\#,\ 1235\ ) \\
& \xrightarrow{pop} & (\ \#,\ \#,\ 1235\ ) \\
& \xrightarrow{elfogad} & \textbf{OK}
\end{array}
$$

Az abc szó levezetése az 1, 2, 3, 5 helyettesítési szabályok alkalmazásával:

$S \Rightarrow aS \Rightarrow aA \Rightarrow abAc \Rightarrow abc$

Elemzés az aba szóra:

$$
\begin{array}{lll}
(aba\#, S\#, \varepsilon) & \xrightarrow{(aS,1)} & (\ aba\#,\ aS\#,\ 1\ ) \\
& \xrightarrow{pop} & (\ ba\#,\ S\#,\ 1\ ) \\
& \xrightarrow{(A,2)} & (\ ba\#,\ A\#,\ 12\ ) \\
& \xrightarrow{(bAc,3)} & (\ ba\#,\ bAc\#,\ 123\ ) \\
& \xrightarrow{pop} & (\ a\#,\ Ac\#,\ 123\ ) \\
& \rightarrow & \textbf{HIBA}
\end{array}
$$

Rekurzív leszállás módszere Példa C++-ban

Helyettesítési szabályok:

$S' \rightarrow S$

$S \rightarrow aS \mid A$

$A \rightarrow bA \mid \varepsilon$

```
# include < iostream >
using namespace std;
char aktszimb;
char w[1024];
int i;
void S();
void A();

void Vizsgal(char a){
if ( a == aktszimb )
aktszimb = w[++i];
else {
      cout << "HIBA ! " << endl;
      exit(1);
     }
```

```
}

void S(){
switch (aktszimb){
 case 'a' : Vizsgal('a') ; S(); break;
 case 'b' : A(); break;
 case '#' : ;
 }
}

void A(){
switch (aktszimb) {
 case 'b' : Vizsgal('b'); A(); break;
 case '#' : ;
 }
}

int main(){
 cout << "w = ";
 cin >> w;
 i = 0 ;
 aktszimb = w[i];
 S() ;
 Vizsgal ('#');
 return 0;
}
```

LR(1) elemzés

állapot	*action*			*goto*	
	a	b	#	S	A
0	$s3$	$s4$		1	2
1			$elfogad$		
2	$s3$	$s4$			5
3	$s3$	$s4$			6
4	$r3$	$r3$	$r3$		
5			$r1$		
6	$r2$	$r2$	$r2$		

			szabály
$(\#0,\ abb\#)$	$\xrightarrow{s3}$	$(\#0a3,\ bb\#)$	
	$\xrightarrow{s4}$	$(\#0a3b4,\ b\#)$	
	$\xrightarrow{r3}$	$(\#0a3A6,\ b\#)$	$A \to b$
	$\xrightarrow{r2}$	$(\#0A2,\ b\#)$	$A \to aA$
	$\xrightarrow{s4}$	$(\#0A2b4,\ \#)$	
	$\xrightarrow{r3}$	$(\#0A2A5,\ \#)$	$A \to b$
	$\xrightarrow{r1}$	$(\#0S\,1,\ \#)$	$S \to AA$
	$\xrightarrow{\textbf{elfogad}}$	**O.K.**	

Precedencia elemzés

	S	A	B	C	$+$	$*$	a	$\$$
S								
A					$\doteq$			$\gtrdot$
B					$\gtrdot$			$\gtrdot$
C					$\gtrdot$	$\doteq$		$\gtrdot$
$+$			$\doteq$	$\lessdot$			$\lessdot$	
$*$			$\doteq$	$\lessdot$			$\lessdot$	
a					$\gtrdot$	$\gtrdot$		$\gtrdot$
$\$$		$\lessdot$	$\lessdot$	$\lessdot$			$\lessdot$	

Az $a + a * a$ szót vizsgáljuk.

$\$ \lessdot a \gtrdot + a * a \$$ **redukálás** $C \to a$

$\$ \lessdot C \gtrdot + a * a \$$ **redukálás** $B \to C$

$\$ \lessdot B \gtrdot + a * a \$$ **redukálás** $A \to B$

$\$ \lessdot A \doteq + \lessdot a \gtrdot * a \$$ **redukálás** $C \to a$

$\$ \lessdot A \doteq + \lessdot C \doteq * \lessdot a \gtrdot \$$ **redukálás** $C \to a$

$\$ \lessdot A \doteq + \lessdot C \doteq * \lessdot C \gtrdot \$$ **redukálás** $B \to C$

$\$ \lessdot A \doteq + \lessdot C \doteq * \doteq B \gtrdot \$$	**redukálás** $B \to C * B$
$\$ \lessdot A \doteq + \doteq B \gtrdot \$$	**redukálás** $A \to A + B$
$\$ \lessdot A \gtrdot \$$	**redukálás** $S \to A$
$\$ \; S \; \$$	

Ez megfelel a következő (legjobboldalibb) levezetésnek:

$S \Rightarrow A \Rightarrow A + B \Rightarrow A + C * B \Rightarrow A + C * C \Rightarrow A + C * a \Rightarrow A + a * a$
$\Rightarrow B + a * a \Rightarrow C + a * a \Rightarrow a + a * a$

Szintaktikai (~~szintaktikus~~) elemzés

- **felülről lefelé (top-down) elemzés:**
  - **LL(1) elemzés**
  - **rekurzív leszállás módszere**
- **alulról felfelé (bottom-up) elemzés:**
  - **LR(1) elemzés**
  - **elsőbbségi nyelvtanon alapuló elemzés**

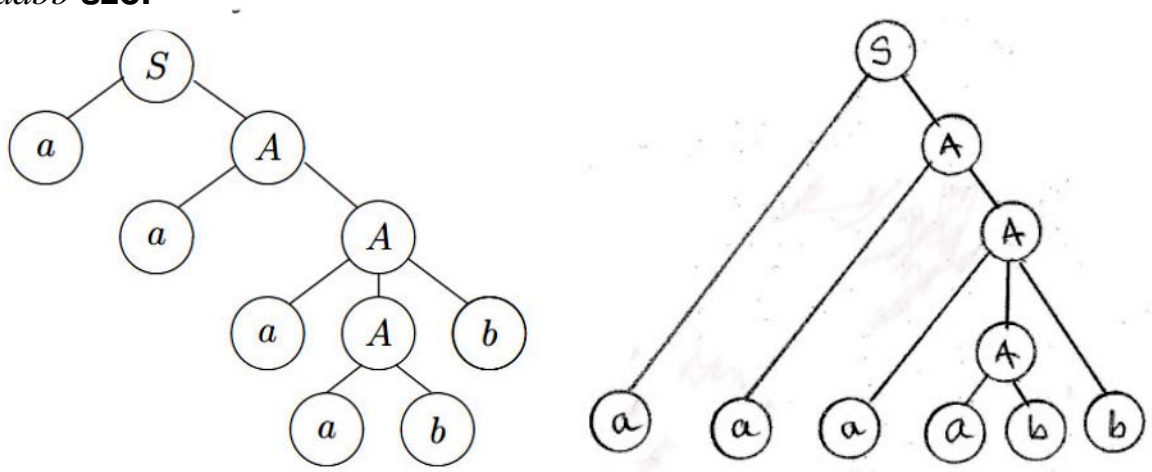
Szemantikai elemzés

A szintaktikai elemzés meghatározta az elemezendő szöveg szintaxisfáját.

Tekintsük a $G = (\{S, A\}, \{a, b\}, \{S \to aA,\ S \to a,\ S \to \varepsilon,\ A \to aA,\ A \to aAb,\ A \to ab,\ A \to b\}, S)$ **környezetfüggetlen nyelvtant. Ez a nyelvtan az** $L(G) = \{a^n b^m \mid n \geq m \geq 0\}$ **nyelvet generálja. Az** $a^4b^2 \in L(G)$ **szó levezetése a következő:**

$$S \Longrightarrow aA \Longrightarrow aaA \Longrightarrow aaaAb \Longrightarrow aaaabb.$$

Ezt a levezetést ábrázolhatjuk levezetési fával, amelynek eredménye az *aaaabb* **szó.**



Minden levezetéshez hozzárendelhető egy levezetési fa. Fordítva, egy levezetési fához több levezetés is rendelhető.

A szintaxisfa pontjaihoz olyan attribútumokat rendelünk, amelyek leírják az adott pont tulajdonságait.
(*Az attribútum a jelenségek, tárgyak lényegi, szükségszerű, elválaszthatatlan tulajdonsága. Latinul: mellérendel, neki tulajdonít, részesít.*)

Ezeknek az attribútumoknak a meghatározása és az attribútumok konzisztenciájának vizsgálata a szemantikai elemzés feladata lesz.
(*konzistens = belső ellentmondásoktól mentes, egységes, következetes, tartalmas*)

A szemantikai elemző a *statikus szemantikával*, azaz az olyan tulajdonságok vizsgálatával foglalkozik, amelyek nem írhatók le környezetfüggetlen nyelvtannal.

A szemantikai elemzők általában a következő tulajdonságokat vizsgálják:

- **változók deklarációja és a változók hatásköre, láthatósága,**
- **változók többszörös deklarációja, a deklaráció hiánya,**
- **operátorok és operandusaik közötti típuskompatibilitás,**
- **eljárások, tömbök formális és aktuális paraméterei közötti kompatibilitás,**
- **túlterhelések egyértelműsége (túlterhelés pl. a szimbólumtábla betelése).**

Bár történtek kísérletek arra, hogy a szemantikai elemzőt környezetfüggő nyelvtanból készítsék el, ezt a módszert elvetették, mert az elemzők rendkívül bonyolultak és lassúak voltak. A szemantikai elemzésre az a módszer terjedt el, hogy az egyes szemantikai tulajdonságok vizsgálatára önálló programokat írnak.

A fordítási nyelvtanok

A helyettesítési szabályokban speciális jelekkel, *akciószimbólumokkal* jelölhetjük azt, hogy a szintaxisfa felépítése folyamán az adott szabály alkalmazása esetén szemantikai elemzési tevékenységeket is kell végezni. Ha a szemantikai tevékenységet egy *proc* eljárás írja le, akkor ezt az eljárást *szemantikai rutinnak* nevezzük és a hozzátartozó akciószimbólumot *@proc*-cal jelöljük.

A $@s$ akciószimbólum az $S \overset{*}{\Longrightarrow} \alpha@s\beta \overset{*}{\Longrightarrow} x$ levezetésben azt jelenti, hogy az elemzéskor a $@s$ sorra kerülése esetén az s eljárást kell meghívni, és az elemzés csak ennek a programelemnek a lefutása után folytatódhat.

Ha egy G környezetfüggetlen nyelvtan szabályainak jobb oldalait akciószimbólumokkal egészítjük ki, akkor a nyelvtant fordítási nyelvtannak nevezzük.
A fordítási nyelvtanokat *TG*-vel (translational grammar) jelöljük.[1]

Példa.
Legyen egy nyelvtannak az
⟨***értékadó-utasítás***⟩ → ⟨***változó***⟩ := ⟨***kifejezés***⟩
egy szabálya, és az ebből kialakított fordítási nyelvtannak a szabálya legyen
⟨***értékadó-utasítás***⟩ → ⟨***változó***⟩ := ⟨***kifejezés***⟩ ***@CheckType.***
A *@CheckType* akciószimbólumhoz tartozó program a ⟨*változó*⟩ és a ⟨*kifejezés*⟩ típusának azonosságát vizsgálja.

A szemantikai elemzést megnehezítheti az, hogy az akciószimbólumokkal jelölt szemantikai rutinoknak nincs paraméterük, az akciószimbólumok által jelölt eljárásokba bele kell építeni azt, hogy az eljárások a szükséges paramétereket hol találják meg.

[1] Angol–magyar informatikai szótár, http://www.tintakiado.hu/informatikai_szotar.php

Mivel mind a felülről lefelé, mind az alulról felfelé elemzések az elemzéshez egy vermet használnak, célszerűnek látszik, hogy a szimbólumok attribútumait is egy verembe helyezzük el, és ez a verem a szemantikai rutinok egymás közötti paraméterátadására is szolgálhat. Ezt a vermet *szemantikai veremnek* hívjuk.

Egy általános módszerrel foglalkozunk. A *TG* szimbólumaihoz attribútumokat rendelünk, ezek az attribútumok szolgálnak majd arra, hogy a szemantikai információt továbbítsák, és ebből majd a szemantikai elemző programot is generálni tudjuk.

Attribútum fordítási nyelvtanok

Legyen *TG* egy fordítási nyelvtan. A továbbiakban jelöljük a *TG* fordítási nyelvtan *akciószimbólumainak halmazát* $@\mathcal{S}$-sel, és jelöljük X-szel a nyelvtan tetszőleges terminális vagy nemterminális szimbólumát, vagy egy tetszőleges akciószimbólumát, azaz legyen $X \in T \cup N \cup @\mathcal{S}$.

Legyen $\mathcal{A}$ egy véges, nem üres halmaz, az *attribútumok halmaza*.

Ekkor minden X szimbólumhoz rendeljük hozzá az $\mathcal{A}$ egy $\mathcal{A}(X)$ részhalmazát, speciálisan, a $@s$ szimbólumra $\mathcal{A}(@s)$ legyen az s szemantikai rutin paramétereinek halmaza.

Az $x \in L(TG)$ mondat szintaxisfájában az X ponthoz az $\mathcal{A}(X)$-beli attribútumok *egy-egy példányát* rendeljük, tehát a szintaxisfa két különböző helyén levő X pontban két azonos nevű attribútum között nincs semmilyen kapcsolat.

A továbbiakban jelöljük az X szimbólum $a \in \mathcal{A}(X)$ attribútumát $X.a$-val. Az attribútumokat a nyelvtan helyettesítési szabályaiba, a szimbólumok megismétlése nélkül, közvetlenül a szimbólum mellé is beírhatjuk. Ha egy szimbólumnak több attribútuma is van, akkor ezeket vesszővel választjuk el.

Ha egy helyettesítési szabályban egy szimbólum többször is előfordul, akkor a szimbólumokat indexeléssel különböztetjük meg, utalva arra, hogy azonos nevű attribútumaik különbözőek.

Ha két különböző szimbólumhoz azonos nevű attribútumokat rendelünk, akkor az attribútum elé a hozzátartozó szimbólumot mindig meg kell adni. Az a azonossága az $X.a$ és az $Y.a$ $(X \neq Y)$ attribútumokban semmilyen összefüggésre nem utal, az attribútumok között lévő kapcsolatot a szemantikai függvények írják le.

Az *attribútumértékek halmazát* jelöljük $\mathcal{V}$-vel, a $\mathcal{V}$-re semmilyen megkötést nem teszünk.

Legyen $\mathcal{R}$ a *szemantikai függvények halmaza*, és minden $p \in P$ helyettesítési szabályhoz rendeljük hozzá az $\mathcal{R}$ egy $\mathcal{R}(p)$ részhalmazát. Ha a $p \in P$ helyettesítési szabályhoz tartozó attribútum-előfordulások halmazát $\mathcal{A}(p)$-vel jelöljük, akkor az $\mathcal{R}(p)$ minden szemantikai függvényének minden argumentuma az $\mathcal{A}(p)$ egy eleme, és értéke az $\mathcal{A}(p)$ egy elemének az értéke. A $@s$ akciószimbólumhoz tartozó s eljárást egy szemantikai függvény programjának tekintjük.

Az X szimbólum attribútumértékeinek egyértelműeknek kell lenniük, azaz minden $x \in L(TG)$ mondatra az x-hez tartozó szintaxisfa minden X pontjában minden attribútum értékét legfeljebb csak egy szemantikai függvény határozhatja meg.

Megjegyezzük, ha $p : X \to \alpha$ és $q : Y \to \beta X \gamma$ két helyettesítési szabály, akkor lehetnek olyan $\mathcal{A}(X)$-beli attribútumok is, amelyek értékének meghatározására sem $\mathcal{R}(p)$-ben, sem $\mathcal{R}(q)$-ban nincs szemantikai függvény.

Legyen $\mathcal{C}$ a *logikai feltételek halmaza*, és minden $p \in P$ helyettesítési szabályhoz rendeljük hozzá a $\mathcal{C}$ egy $\mathcal{C}(p)$ elemét. A $\mathcal{C}(p)$ egy logikai állítást mond ki a p helyettesítési szabályhoz tartozó $\mathcal{A}(p)$ attribútumokra.

Ha az állítás a p szabály feldolgozásakor az $\mathcal{A}(p)$ attribútumokra nem teljesül, akkor a szemantikai elemzőnek hibajelzést kell adnia. A $\mathcal{C}$ halmaz lehet üres halmaz is. Tehát a p szabályhoz tartozó szemantikai tulajdonságokat a $\mathcal{C}(p)$ kiértékelésével tudjuk ellenőrizni.

Az ATG **$= (TG, \mathcal{A}, \mathcal{V}, \mathcal{R}, \mathcal{C})$** ***ötöst attribútum fordítási nyelvtannak nevezzük, ahol***

- ***TG egy fordítási nyelvtan,***
- **$\mathcal{A}$** ***az attribútumok halmaza,***
- **$\mathcal{V}$** ***az attribútumértékek halmaza,***
- **$\mathcal{R}$** ***a szemantikai függvények halmaza, és***
- **$\mathcal{C}$** ***a logikai feltételek halmaza.***

Az $X \in T \cup N$ szimbólum egy $X.a$ attribútumát *szintetizáltnak* nevezzük, ha értékét egy szemantikai függvény abban az esetben határozza meg, amikor az X szimbólum egy helyettesítési szabály bal oldalán áll. Az $X.a$ attribútum *örökölt*, ha értékét egy szemantikai függvény akkor határozza meg, amikor az X szimbólum egy helyettesítési szabály jobb oldalán áll. Tehát az információt egy szintaxisfában a szintetizált attribútumok alulról-felfelé, az örökölt attribútumok pedig felülről-lefelé és egy helyettesítési szabály jobb oldalát alkotó pontokon továbbítják.

A nyelvtan S kezdőszimbólumához nem tartoznak örökölt attribútumok, és a terminális szimbólumokhoz nem tartoznak szintetizált attribútumok. A terminális szimbólumoknak azonban vannak „szintetizált jellegű”, úgynevezett *kitüntetett szintetizált attribútumaik*. Ilyen attribútum például egy konstans terminális szimbólum esetén a konstans értéke és típusa, vagy egy azonosító szimbólum esetén a szimbólum neve. Feltehetjük azt, hogy a kitüntetett szintetizált attribútumok értékét konstans értékű szemantikai függvények határozzák meg, és ezeket a konstans értékeket egy, az attribútum nyelvtantól független külső eljárás, a lexikális elemző adja át.

A $@s$ akciószimbólum $@s.a$ attribútuma *örökölt*, ha az a az s eljárás bemenő paramétere, és a $@s.a$ attribútum *szintetizált*, ha az a az s eljárás kimenő paramétere, azaz értékét az s eljárás határozza meg.

Jelölje a továbbiakban egy *ATG* nyelvtan $X \in T \cup N \cup @\mathcal{S}$ szimbólumának szintetizált és örökölt attribútumait $\mathcal{S}(X)$ és $\mathcal{I}(X)$.

Legyen *AF*(p) az $\mathcal{R}(p)$ szemantikai függvények által *meghatározott értékű attribútumok halmaza*, azaz legyen
AF$(p) = \{X.a \mid X.a = f(\ldots) \in \mathcal{R}(p)\}$**.**

Az $X.a$ szintetizált attribútum, ha létezik egy olyan $p : X \to \alpha$ helyettesítési szabály, melyre $X.a \in$ *AF*(p), és az $X.a$ örökölt attribútum, ha létezik egy olyan $q : A \to \alpha X \beta$ helyettesítési szabály, melyre $X.a \in$ *AF*(q).

Ha az X szimbólum $X.a$ attribútumának szintetizált vagy örökölt jellegét is meg akarjuk különböztetni, akkor az attribútumot $X{\uparrow}a$ vagy $X{\downarrow}a$-val jelöljük. Az $X{\uparrow}a, b{\downarrow}c, d$ az $X \uparrow a$, $X \uparrow b$, $X \downarrow c$ és $X \downarrow d$ rövid jelölése. A nyelvtanok helyettesítési szabályaiban az $X{\uparrow}a, b{\downarrow}c, d$ az X szimbólumot és a szimbólum négy attribútumát jelöli. Az akciószimbólumok attribútumait, utalva arra, hogy ezek egy eljárás paraméterei, zárójelbe tesszük, például $@s({\uparrow}a, b, {\downarrow}c, d)$.

Ha $X \in T \cup N$, és $X.a \in \mathcal{S}(X) \cap \mathcal{I}(X)$ lenne, akkor létezne egy olyan $p : X \to \alpha$ helyettesítési szabály, melyre $X.a \in$ *AF*(p), és létezne egy olyan $q : Y \to \beta X \gamma$ szabály, melyre $X.a \in$ *AF*(q). Mivel $S \overset{*}{\Longrightarrow} \varphi Y \psi \Longrightarrow \varphi\beta X\gamma\psi \Longrightarrow \varphi\beta\alpha\gamma\psi \overset{*}{\Longrightarrow} x$, az $X.a$ kiszámítására legalább kettő szemantikai függvény lenne, ami ellentmond a kiszámíthatóságra tett feltételnek.

Hasonlóan, ha $@s \in @\mathcal{S}$, és $@s.a \in \mathcal{I}(@s)$, akkor létezik olyan p szabály, amelyre $@s \in$ *AF*(p). Ha $@s.a \in \mathcal{S}(@s)$ is fennállna, akkor a $@s.a$ értékét az s szemantikai rutin is meghatározná, ami szintén ellentmond a kiszámíthatóságára tett feltételnek.

Így a szintetizált és örökölt attribútumokra érvényes a következő állítás:

Tétel. Az *ATG* nyelvtan minden $X \in T \cup N \cup @\mathcal{S}$ szimbólumára $\mathcal{S}(X) \cap \mathcal{I}(X) = \emptyset$.

Ha egy szintaxisfa pontjaiban meg akarjuk határozni az attribútumok értékét, akkor először csak a levelekhez tartozó kitüntetett szintetizált attribútumok értékei ismertek. A szemantikai elemzés fogja az összes többi attribútum értéket meghatározni. Az egyes értékek meghatározásának sorrendje közömbös, de azt megköveteljük, hogy egy szemantikai függvény alkalmazása esetén az argumentumok értéke már ismert legyen.

Először vizsgáljuk meg azt, hogy egy helyettesítési szabály attribútumai közül melyeket kell a szabály alkalmazásakor meghatároznunk.

Egy attribútum fordítási nyelvtant teljes attribútum fordítási nyelvtannak nevezünk, ha minden $p : X_0 \to X_1 X_2 \dots X_n$ helyettesítési szabályra

- $\mathcal{S}(X_i) \subseteq$ ***AF***(p)***, ha*** $i = 0$***, vagy*** $X_i = @s$***,***
- $\mathcal{I}(X_i) \subseteq$ ***AF***(p) $(1 \le i \le n)$***,***
- $\mathcal{A}(X_i) = \mathcal{S}(X_i) \cup \mathcal{I}(X_i)$ $(0 \le i \le n)$**.**

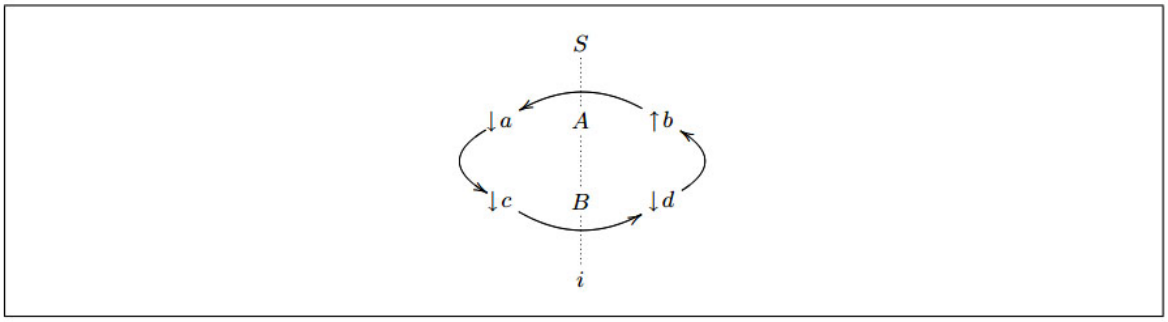
A teljes *ATG* azonban csak az egyszintű részfákra biztosítja az attribútumok kiszámíthatóságát, ami még nem jelenti azt, hogy egy szintaxisfa minden attribútumának az értéke meghatározható.

Példa.
Legyen egy *ATG*-ben
$N = \{S, A, B\}$, $T = \{i\}$,
$\mathcal{A} = \{A\downarrow a, A\uparrow b, B\downarrow c, B\downarrow d\}$,
és a P és az $\mathcal{R}$ halmaz a következő:

$$
\begin{array}{lcll}
S & \to & A{\downarrow}a{\uparrow}b & \{A{\downarrow}a \leftarrow A{\uparrow}b\} \\
A{\downarrow}a{\uparrow}b & \to & B{\downarrow}c,d & \{A{\uparrow}b \leftarrow B{\downarrow}d,\ B{\downarrow}d \leftarrow B{\downarrow}c,\ B{\downarrow}c \leftarrow A{\downarrow}a\} \\
B{\downarrow}c,d & \to & i & \emptyset
\end{array}
$$

Látható, hogy a nyelvtan egy teljes attribútum nyelvtan. Az $S \stackrel{+}{\Longrightarrow} i$ levezetéshez tartozó, attribútumokkal kiegészített szintaxisfa a ?? ábrán látható. Az ábrán (és a további ábrákon is,) a szintaxisfa éleit pontokkal, az attribútum értékadásokat nyilakkal ábrázoljuk. Az attribútumok egy gráf csúcspontjai, és az attribútumok értékeinek meghatározását a gráf élei jelölik.

Az ábráról leolvasható, hogy az attribútumok értékeit nem tudjuk meghatározni, mivel az attribútumok értékeit meghatározó gráf egy kört tartalmaz.



Az $S \stackrel{+}{\Longrightarrow} i$ levezetés attribútumai

A fordításhoz természetesen olyan *ATG*-k kellenek, amelyekben a szintaxisfák összes attribútumának az értéke meghatározható.

Egy attribútum fordítási nyelvtant jól definiált attribútum fordítási nyelvtannak nevezünk, ha a nyelvtan által generált nyelv minden mondatára teljesül, hogy a mondat szintaxisfájának minden pontjában minden attribútum értéke egyértelműen kiszámítható.

Nyilvánvaló, hogy minden jól definiált attribútum fordítási nyelvtan teljes, de ez fordítva nem áll fenn, mint azt a példában is láttuk.

Ha egy $X.a$ attribútum értéke szükséges az $Y.b$ attribútum értékének meghatározásához, akkor ezt az $(X.a, Y.b)$ párossal jelöljük.

A $p : X_0 \to X_1 X_2 \dots X_n$ helyettesítési szabályhoz tartozó direkt attribútumfüggőségek a következők:
DP$(p) = \{(X_i.a, X_j.b) \mid X_j.b = f(\dots, X_i.a, \dots) \in \mathcal{R}(p), \quad (0 \le i, j \le n)\}$**.**

A direkt attribútumfüggőségek egy adott szintaxisfára egy *függőségi gráfot* generálnak, ahol a gráf pontjai az attribútumok, az irányított élek pedig a fenti kapcsolatot leíró relációk. A példában éppen egy ilyen függőségi gráfot ábrázoltunk.

Egy attribútum fordítási nyelvtant *lokálisan aciklikusnak* nevezünk akkor, ha minden $p \in P$ helyettesítési szabályra a *DP*(p) függőségi gráf körmentes.

Legyen *DT*(x) az x mondat levezetésében felhasznált összes p szabályhoz tartozó *DP*(p) direkt attribútum függőségek halmaza. A *DT*(x)-hez tartozó direkt attribútumfüggőségeket gráfban is ábrázolhatjuk.

A teljesség, a jól definiáltság és a *DT*(x) direkt attribútumfüggőségre teljesül a következő állítás.

Tétel. Egy teljes attribútum fordítási nyelvtan jól definiált, ha a nyelvtan által generált nyelv minden x mondatára a *DT*(x) gráf nem tartalmaz kört.

A jól definiáltság tehát azt jelenti, hogy a nyelvtan nem csak lokálisan aciklikus, hanem minden mondatának szintaxisfájára teljesül az is, hogy az attribútumok között nincs cirkuláris függőség. A fordítóprogramokban nyilvánvaló, hogy csak jól definiált attribútum fordítási nyelvtanok alkalmazhatók.

A jól definiált *ATG*-kben az attribútumok értékei meghatározhatók. Ezekkel a nyelvtanokkal a problémát azonban az okozza, hogy a függőségi gráfok körmentességét csak exponenciális idejű algoritmussal lehet eldönteni. Ezért a gyakorlatban az attribútum fordítási nyelvtanokra a jól definiáltságon kívül további megszorításokat kell tennünk.

A kódgenerálás

Azt vizsgáljuk, hogy a fordítóprogramok milyen módszerek felhasználásával építik fel a program kódját. A már megismert *L-ATG* nyelvtanok felhasználhatóságát mutatja, hogy kódgenerálás feladata is leírható *L-ATG* nyelvtannal.

A kódgenerálás feladatai közül csak néhányat mutatunk be.

Az aktivációs rekord

Futási időben a program i-edik blokkjának adatait az AR_i *aktivációs rekord* tartalmazza. Az éppen futó blokk aktivációs rekordját *aktív aktivációs rekordnak* nevezzük.

A fordítóprogram olyan kódot generál, hogy az aktivációs rekord a program futtatásakor, a blokk hívásakor épüljön fel a számítógép run-time vermébe.

Az aktivációs rekord három részből áll, a *lokális változók területéből*, a *display-területből (megjelenítési terület)* és a *paraméterterületből*.

A display-terület egyik adata a *call* utasítással indított blokkból a hívó programra való visszatéréshez szükséges utasításcím, az a cím, amit a *call* utasítás végrehajtásakor a processzor ír a verembe. A *statikus pointer* a blokkból elérhető változókat tartalmazó aktivációs rekordra mutat. A display-területen található meg a hívó blokk aktivációs rekordjának címe is. A hívó blokk aktivációs rekordjának címére azért van szükség, hogy a blokkból való kilépéskor visszaállítható legyen a run-time veremnek a blokkba való belépéskor fennálló állapota. Ezt az adatot *dinamikus pointernek* nevezzük.

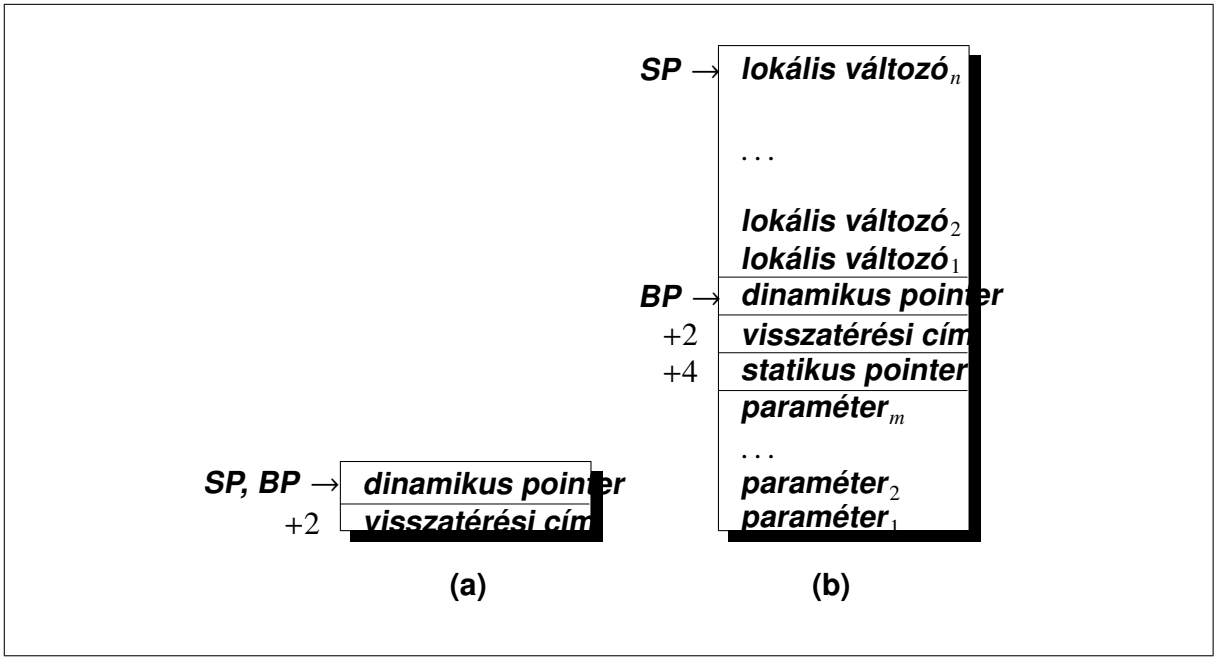
Az aktivációs rekord paraméterterülete a blokk aktuális paramétereit tartalmazza. Érték szerinti paraméterátadásnál a paraméter értéke, hi-

vatkozás szerinti paraméterátadásnál a paraméter memóriacíme található ezen a területen.

Az IBM PC gépeken a verem tetejét az *SP* regiszter jelzi. Az aktív aktivációs rekord dinamikus pointerére a *BP* regiszter mutat, a $[BP] + 2$ címen a visszatérési cím, a $[BP] + 4$ címen a statikus pointer található.

A fordítóprogramok a főprogram lokális változóit, azaz a globális változókat az adatszegmensbe helyezik, így a főprogram aktivációs rekordja csak a dinamikus pointert és a főprogramból a rendszerhez való visszatéréshez egy visszatérési címet tartalmaz.

Az alprogramok lokális változói az aktivációs rekordba kerülnek, azaz majd a futási időben a run-time verembe. Egy n paramétert és m lokális változót tartalmazó eljáráshoz az ábrán látható aktivációs rekord tartozik.



Az aktivációs rekordok szerkezete: (a) főprogram, (b) alprogram

A kifejezések fordítása

A kifejezések fordítását egy egyszerű, a példában szereplő nyelvtannal generált kifejezéseken tanulmányozzuk.

Ha az $F \rightarrow i$ helyettesítési szabályban szereplő i egy egyszerű változó, akkor legyen

$F \rightarrow i\uparrow n$ ***@SearchVar***$(\downarrow n,\uparrow t, q)$ ***@StLoadAX***$(\downarrow q)$,

ahol az n attribútum a változó azonosítója, t a típusa, és q a deklarációban meghatározott címe. A *@SearchVar*$(\downarrow n,\uparrow t, q)$ megkeresi az n szimbólumot a szimbólumtáblában. Ha van ilyen nevű szimbólum, akkor ellenőrzi a láthatóságát, és kimenetként adja a szimbólum típusát és a szimbólumtáblában levő címét. A *@StLoadAX*$(\downarrow q)$ szemantikus rutin a változó értékét az *AX* regiszterbe töltő utasítást generálja:

```
mov     ax,q        ; @StLoadAX(↓q)
```

Így a p nevű globális változóra a

```
mov     ax,p        ; @StLoadAX(↓q)
```

a lokális, azaz a veremben elhelyezett aktivációs rekord változójára a

```
mov     ax,word ptr [bp]-k   ; @StLoadAX(↓q)
```

utasítás fog generálódni.

A példában az s attribútumot használtuk közbülső eredmények tárolására. Mivel egy művelet eredménye mindig az *AX* regiszterben van, most ezt a funkciót a *push AX* utasítással tudjuk megvalósítani. Generálja a *@StPushAX* szemantikus rutin ezt az utasítást. A *@add*$(\downarrow s, x,\uparrow x)$ és a *@mul*$(\downarrow s, v,\uparrow v)$ szemantikus rutinok végezték a műveletek végrehajtását, ezt a *pop BX*; *add AX,BX* és *pop BX*; *imul BX* utasításpárokkal tudjuk leírni. Generálják a *@StPopBX*, *@StAddBX* és *@StImulBX* szemantikus rutinok ezeket az utasításokat.

A kétbájtos *integer* kifejezést leíró nyelvtant tehát a következőképpen adhatjuk meg:

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow + \textbf{\textit{@StPushAX}}\ T\ \textbf{\textit{@StPopBX @StAddBX}}\ E' \mid \varepsilon \\
T &\rightarrow FT' \\
T' &\rightarrow * \textbf{\textit{@StPushAX}}\ F\ \textbf{\textit{@StPopBX @StImulBX}}\ T' \mid \varepsilon \\
F &\rightarrow (E) \\
&\mid i{\uparrow}n\ \textbf{\textit{@SearchVar}}({\downarrow}n,{\uparrow}t,q)\ \textbf{\textit{@StLoadAX}}({\downarrow}q).
\end{aligned}
$$

Példa.

Határozzuk meg a $2+j*(3+k)$ kifejezéshez a fenti nyelvtannal generálható programot. Legyen j egy globális szimbólum, k pedig az aktivációs rekord első lokális változója.

```
        mov     ax,2        ; 2 @StLoadAX
        push    ax          ; + @StPushAX
        mov     ax,j        ; j @StLoadAX
        push    ax          ; * @StPushAX (
        mov     ax,3        ; 3 @StLoadAX
        push    ax          ; + @StPushAX
        mov     ax,word ptr [bp]-2 ; k @StLoadAX
        pop     bx          ; @StPopBX
        add     ax,bx       ; @StAddBX )
        pop     bx          ; @StPopBX
        imul    bx          ; @StImulBX
        pop     bx          ; @StPopBX
        add     ax,bx       ; @StAddBX
```

Az *if* utasítás fordítása

Vizsgáljuk az *if* utasítás következő formáját:

⟨***if-utasítás***⟩ → ***if*** ⟨***kifejezés***⟩ ***then*** ⟨***utasítás***$_1$⟩ ⟨***if-tail***⟩

⟨***if-tail***⟩ → ***else*** ⟨***utasítás***$_2$⟩ ***endif***

| ***endif***

Az utasításból a következő felépítésű kódok egyikét kell generálnunk:

```
        <kifejezés>
        cmp     ax,0
        jz      L_0001
        <utasítás1>
L_0001:
```

vagy

```
        <kifejezés>
        cmp     ax,0
        jz      L_0001
        <utasítás1>
        jmp     L_0002
L_0001:
        <utasítás2>
L_0002:
```

Látható, hogy az *if* utasításhoz egyedi címkét kell generálni. A címke nevének generálását végezze a *@GenLabel*$(\uparrow r)$ szemantikus rutin.

A *@GenLabel*$(\uparrow r)$ feladata hasonló az assemblerek makrófordításkor működő címkegenerálásához, azaz amikor a *label* direktívával megadott címkékre a makróhelyettesítésekben az assembler a *??nnnn* címkéket állítja elő, ahol *nnnn* a címkéket megkülönböztető, minden helyettesítésben eggyel megnövelt természetes szám. Az *nnnn* érték kerül az r attribútumba, és a generált címke *L_nnnn* alakú.

A *@GenLabel*$(\uparrow r)$ szemantikus rutinnal meghatározott r címkenévvel a *@StLabel*$(\downarrow r)$ szemantikus rutin az

```
L_r     equ     $           @StLabel(↓r)$
```

vagy az ezzel ekvivalens

```
L_r:                        @StLabel(↓r)
```

sort állítja elő. A *@StJmp*$(\downarrow r)$ rutin generálja a következő feltétel nélküli ugró utasítást:

```
        jmp     L_r         ; @StJmp(↓r)
```

A ⟨*kifejezés*⟩ feldolgozása az *AX* regiszterben nullát ad, ha a kifejezés értéke *false*. Ezért generálja a *@StJFalse*$(\downarrow r)$ szemantikus rutin a következő utasításokat:

```
        cmp     ax,0        ; @StJFalse(↓r)
        jnz     $+5
        jmp     L_r
```

A *jnz* és *jmp* utasításpárra azért van szükség, mert az assembly nyelvben nincs *near* típusú feltételes ugró utasítás. Ha az ugrás távolsága megfelel a *short* típusú ugrásnak, akkor a *@StJFalse*$(\downarrow r)$ rutinnal generált utasítások egyszerűbben is megadhatók:

```
        cmp     ax,0        ; @StJFalse(↓r)
        jz      L_r
```

Így a kódgeneráláshoz az *if* utasítás fenti leírása a következőképpen alakítható át:

⟨***if-utasítás***⟩ → ***if*** ⟨***kifejezés***⟩ ***@GenLabel***$(\uparrow r)$ ***@StJFalse***$(\downarrow r)$
then ⟨***utasítás***$_1$⟩ ⟨***if-tail***⟩$\downarrow r$

⟨***if-tail***⟩$\downarrow r$ → ***else @GenLabel***$(\uparrow s)$ ***@StJmp***$(\downarrow s)$ ***@StLabel***$(\downarrow r)$
⟨***utasítás***$_2$⟩ ***endif @StLabel***$(\downarrow s)$
| ***endif @StLabel***$(\downarrow r)$

Példa.

Az *if a then i := 1 else i := 2 endif* forrásnyelvű programsorból a következő assembly nyelvű programot kapjuk:

```
                        ; if
        mov     ax,a    ; <kifejezés>
                        ; @GenLabel(↑r)   (r = 0001)
        cmp     ax,0    ; @StJFalse(↓r)
        jz      L_0001
                        ; then
        mov     i,1     ; <utasítás1>
                        ; else @GenLabel(↑s)  (s = 0002)
        jmp     L_0002  ; @StJmp(↓s)
L_0001:                 ; @StLabel(↓r)
        mov     i,2     ; <utasítás2>
                        ; endif
L_0002:                 ; @StLabel(↓s)
```

Fordított lengyelforma

en: Polish notation (prefix/reverse Polish notation)
ro: notaţia poloneză (prefixată/sufixată)

Egy adott kifejezést zárójel nélküli alakra hozunk. Adott $\circ$ bináris műveletre $a \circ b$ helyett $ab\circ$ alakot írunk.

X_i**,** $i = 1, 2, \ldots, n$ **kifejezés**
F_j**,** $j = 1, 2, \ldots, m$ **fordított lengyelforma**
$prior$ **megadja a műveletek (operátorok) prioritását**

például:

művelet	***prioritás***
(	**0**
$+, -$	**1**
$*, /$	**2**
$\hat{}$	**3**

$(a + a) * (a - a * a) + a\hat{}a$

$$\underbrace{\underbrace{\underbrace{(a+a)}_{aa+} * \underbrace{(a - \underbrace{a * a}_{aa*})}_{aaa*-}}_{aa+aaa*-*} + \underbrace{a\hat{}a}_{aa\hat{}}}_{aa+aaa*-*aa\hat{}+}$$

tehát fordított lengyelformában: $aa+\ aaa*-*\ aa\hat{}+$

```
FORDÍTOTT_LENGYEL_FORMA(X)
 1  j ← 0
 2  for i = 1 to n
 3      do if X_i operandus
 4            then j ← j + 1
 5                 F_j ← X_i
 6         if X_i operátor
 7            then while (verem nem üres) és (prior(veremcsúcs) ≥ prior(X_i))
 8                       do pop(verem, op)
 9                          j ← j + 1
10                          F_j ← op
11                 push(verem, X_i)
12         if X_i = '('
13            then push(verem, X_i)
14         if X_i = ')'
15            then repeat
16                       pop(verem, op)
17                       if op ≠ '('
18                          then j ← j + 1
19                               F_j ← op
20                 until op = '('
21  while verem nem üres
22        pop(verem, op)
22        j ← j + 1
22        F_j ← op
23  return F
```

A $(a - a * a) * a$ kifejezés átalakítása az algoritmus szerint:

bemenet: $(a - a * a) * a$

$\uparrow$

$i = 1$

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$
verem	(	(	− (	− (	* − (	* − (		*	*
kimenet		a	a	aa	aa	aaa	$aaa * -$	$aaa * -$	$aaa * -a$	$aaa * -a*$

Visszaalakítás:

$aa + aaa * - * aa\hat{}+$

$\boxed{aa+}aaa * - * aa\hat{}+$

$(a + a)$

$\boxed{aa+}a\boxed{aa*} - *aa\hat{}+$

$a * a$

$\boxed{aa+}\boxed{a\boxed{aa*}-} * aa\hat{}+$

$a - a * a$

$\boxed{\boxed{aa+}\boxed{a\boxed{aa*}-} * aa\hat{}+}$

$(a + a) * (a - a * a)$

$\boxed{\boxed{aa+}\boxed{a\boxed{aa*}-} * \boxed{aa\hat{}}+}$

$a\hat{}a$

$(a + a) * (a - a * a) + a\hat{}a$