Fordítóprogramok

Fordítóprogramok szerkezete

Előadó: Pozsgai Tamás

Ajánlott irodalom

- Aho-Sethi-Ullmann: Compilers
- Csörnyei Zoltán: Fordítóprogramok

A jegyzet Csörnyei Zoltán: Fordítóprogramok című könyvének a menetét követi

Jelölések 1

- G(T,N,S,P) : Nyelvtan, (grammatika)

ahol

T : Terminális szimbólumok halmaza
N : Nemterminális szimbólumok halmaza
S : Kezdő szimbólum
P : Helyettesítési szabályok halmaza

Jelölések 2

- L(G) : G nyelvtan által meghatározott nyelv.
- a,b,…∈ T : Terminális szimbólumok.
- A,B,…∈ N : Nemterminális szimbólumok.
- X,Y,Z,…∈ (T∪N) : Terminális vagy nemterminális szimbólumok.
- α,β,…∈ (T∪N)* : Terminális vagy nemterminális szimbólumok sorozata.
- x,y,z,…∈ T* :Terminális szimbólumok sorozata.
- ε : Üres szimbólumsorozat.

A programnyelvek

- A programnyelvek hierarchiája, a gépfüggetlenséget figyelembe véve a következő:
  - Gépi kód : A legalacsonyabb szintű programnyelv.
  - Assembly-nyelv : A gépi kód szimbolikus megfelelője.
  - Magasszintű nyelvek :
    - Felhasználó orientált
    - Probléma orientált nyelv

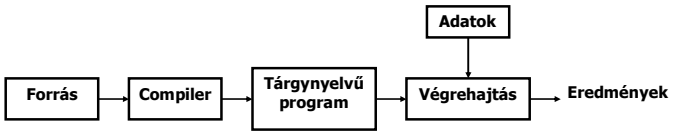
A fordítóprogram

- Input : Forrásnyelvű program. A fordítóprogram ezt lefordítja.
- Output : Tárgyprogram.
- A forrásnyelvű program forrásnyelven van megírva, a tárgyprogram pedig egy tárgynyelvű program.
- Ha a forrásnyelv egy assembly-nyelv és a tárgynyelv gépikód, akkor a fordítóprogramot ***assemblernek*** nevezzük.

Compiler

A forrásnyelv egy magasszintű nyelv, akkor a forrásnyelv és a végrehajtható gépikód nagyon különbözőek.

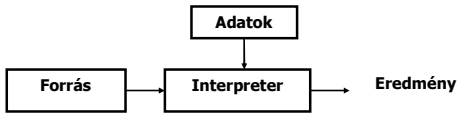
- Az első módszer az, hogy a magasszintű nyelven írt programot egy alacsonyabb szintű nyelvre (pl.: assembly nyelv) fordítjuk le (compiler).



Interpreter

Második módszer:

- Gép mely értelmez egy magassztintű nyelvet. (Gépi kódja magasszintű nyelv.)
- Az interpreter működésekor a fordítási- és a futási idő egybeesik.

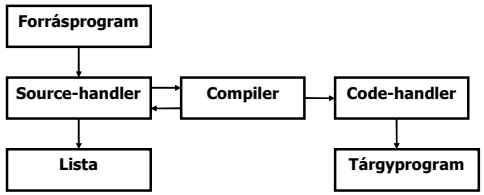


Fordítóprogram szerkezete

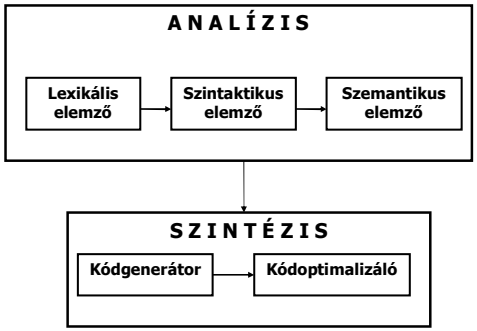
- Compiler(forrásprogram)(tárgypr.,lista) (jelölés: program(input)(output) formájú)
- Input-handler(forrás pr.)(karaktersorozat) A forrásnyelvű programot a fordítás számára könnyen hozzáférhető karaktersorozattá alakítja.
- Output-handler(forrás pr.,hibák)(lista) Ezekből a forrásnyelvű sorokból készül a lista.
- Code-handler(tárgykód)(tárgyprogram) Tárgykód elhelyezése a háttértáron.

Fordítóprogram szerkezete 2

- Az input-output-handlert összefoglalva szokás source handlernek nevezni.
  Source-handler(forrás pr.,hibák)(karaktersorozat,lista)



Compiler felépítése



Analízis

- **Lexikális elemző** (karakter sorozat)(szimbólumsorozat, lexikális hibák). A karaktersorozatból szimbólumsorozatot készít. Pl.:Kiszűri a szóközt, felismeri a kulcsszavakat.
- **Szintaktikus elemző** (szimbólumsorozat)(szintaktikusan elemzett pr.,szintaktikus hibák). Feladata a program struktúrájának a felismerése. Működésének eredménye lehet az elemzett program szintaxisfája.
- **Szemantikus elemző** (szintaktikusan elemzett pr.) (analizált pr,szintaktikus hibák). Feladata bizonyos szemantikai jellegű tulajdonságok vizsgálata.

Analízis 2

- Szintetizálás :
  - A szintézis első lépése a **kódgenerálás** melyet a kódgenerátor végez.
    Kódgenerátor(analizált pr.)(tárgykód)
  - A következő lépés a **kódoptimalizálás.**
    Kódoptimalizáló(tárgykód)(tárgykód)

Lexikális elemzés

- Alapvető feladata, hogy a forrásnyelvű programból a source-handler által készített karaktersorozatban a szimbolikus egységeket felismerje, azaz meghatározza a szimbólum szövegét és a szimbólum típusát.
- A lexikális elemző és a szintaktikus elemző elkülönítésének alapvető oka az, hogy a lexikális elemző reguláris kifejezésekkel leírható, míg a szintaktikus elemzés környezetfüggetlen grammatikával jellemezhető objektumokat használ.

Reguláris kifejezések

- A lexikális elemző számára a szimbólumok reguláris grammatikákkal adhatók meg.
- Lexikális elemzők létrehozásának menete :
  - Feltételezve, hogy a szimbolikus egységek leírása reguláris grammatikával vagy a reguláris kifejezések nyelvén adott, megkonstruáljuk az ekvivalens determinisztikus véges automatát.
  - Elkészítjük a determinisztikus véges automata implementációját.

Példa

- Legyenek a szimbólumok kódjai a következők:

Azonosító	01
Konstans	02
If	50
then	51

Else	52
=	80
:=	70
;	98

- Ekkor a lexikális elemző az:
  if a12 = 3 then i := 5 else j44 := 7;
  programsorból az
  50 01-0001 80 02-0002 51 01-0003 70 02-0004 52 01-0005 70 02-0006 98
  sorozatot készíti, ahol 0001, … 0006 a szimbólumtábla bejegyzésekre mutató pointerek.

Speciális problémák

- Minden programozási nyelvben vannak olyan azonosítók melyeknek speciális célra fenntartott nevük van, ezek a **kulcsszavak**.
- **Standard szavak** :Vannak olyan azonosítók, hogy előre definiált jelentésük van, de ez a jelentés a programban megváltoztatható.
- A kulcsszavak és a standard szavak száma programnyelvenként változik.

Speciális problémák 2

- A kulcsszavak kezelésére két módszert adunk:
  - Minden kulcsszót egy reguláris kifejezéssel írunk le és megadjuk a reguláris kifejezéshez tartozó automata implementációját. Hátrány: Nagyméretű programot kapunk.
  - A kulcsszavakat egy külön táblázatban tároljuk. A karaktersorozatban a szavakat egy általános azonosítófelismerővel határozzuk meg.
- A lexikális elemző a kulcsszavakra alkalmazott módszerrel meghatározhatja, hogy a vizsgált szimbólum standard szó-e.

Előreolvasás

- A lexikális elemző a leghosszabb karaktersorozatból álló szimbólum felismerésére törekszik, a szimbólum jobboldali végpontjának meghatározására gyakran egy vagy több karaktert is előre kell olvasnia.

  Do 10 I=1.1000

  Do 10 I=1,1000
- Mivel a szóköz karakterek nem játszanak szerepet így az 1 és 1000 közötti jel dönti el, hogy az utasítás egy Do szimbólummal kezdődő ciklusutasítás, vagy a D0101 azonosítóra vonatkozó értékadás.
- Az egész számok definíciója a valós számok definíciójának prefixe és a valós számok definíciója a hatványkitevős részt is tartalmazó valós számok definíciójának a prefixe.
- Pozitív egész : D^+ és $D^+ . D^+ e(+|-|\varepsilon) D^+$

Előreolvasás 2

- **Példa :** Tekintsük a 12.3e+f# karaktersorozatot, ahol a # karakter jelzi az elemezendő szöveg végét.
  A lexikális elemző pufferének tartalma:

1	egész szám
12	egész szám
12.	érvénytelen
12.3	valós szám
12.3e	érvénytelen
12.3e+	érvénytelen
12.3e+f	érvénytelen
12.3e+f#	

A felismert szimbólum a 12.3 és típusa valós szám. Az elemzés ezután az e+f szöveg elemzésével folytatódik.

Szimbólumtábla

- Vannak olyan programnyelvek (pl.: C) amelyek a kisbetűs és nagybetűs karaktereket különbözőnek tekintik. Ebben az esetben az azonosító szimbólum betűkaraktereit változtatás nélkül kell felhasználni. Ha a nyelv nem tesz különbséget akkor az összes karaktert vagy kisbetűs vagy nagybetűs alakra kell hozni és ilyen alakban kell tovább feldolgozni.

Direktívák

- A compiler működésének a vezérlésére szolgálnak.
- A direktívában szereplő szimbólumokat is a lexikális elemző határozza meg, de fel kell ismernie, hogy ezek egy direktívához tartoznak.
- Ha a direktíva egy **feltételes fordítás** direktívája, akkor fel kell ismernie a direktíva összes szimbólumát, majd ki kell értékelnie a feltételt.
- Egy másik direktíva a **makróhelyettesítés**.
- Előfeldolgozó program (pl.: C-ben) : Feladata a makróhelyettesítés, a feltételes fordítás feldolgozása és a megadott nevű állományoknak a forrásnyelvű szövegbe való beolvasása.

Hibakezelés

- Ha a lexikális elemző egy karaktersorozatnak nem tud egy szimbólumot sem megfeleltetni, akkor azt mondjuk, hogy a karaktersorozatban ***lexikális hiba*** van. A hiba oka legtöbbször, hogy illegális karakterek kerülnek a szimbólumba, karakterek felcserélődnek vagy hiányoznak.
- Leggyakrabban előforduló hibák:
  - Illegális karakter
  - Kulcsszavak helytelen használata
  - Hiányzó karakterek
  - Formátumhiba Pl.:számok
  - Karakterhiány és a komment-terminátorok hiánya

Lexikális elemző generátor

- A lexikális elemzés egy determinisztikus véges automatával valósítható meg.
- ***LEX*** program: A *lex* reguláris kifejezésekkel definiált programot állít össze. M.E.Lesk és E.Schmidt fejlesztette ki 1975-ben.
- A *lex* lexikális elemző generátor lényegében deterministikus véges automaták implementációját végzi kiegészítve azzal, hogy az automata terminális állapotaihoz egy-egy akciónak nevezett program is hozzárendelhető.
- A *lex* inputja három részből áll, és a következő formátumú:

definíciók
%%
fordítási szabályok
%%
felhasználói programok

Környezetfüggetlen nyelvtanok és a Szintaktikus elemzés

Szintaxis és szemantika

- Szintaktikus elemzés:
  - Környezetfüggetlen nyelvtan (Ide tartozik a programok struktúrájának leírása)
  - Környezetfüggő nyelv (Olyan megkötések tartoznak hozzá mint a típusazonosság egy értékadó utasítás két oldalán), (statikus szemantika)
- Szemantikus elemzés: A nyelv belső összefüggései. (run-time vagy végrehajtási szemantika).

A szintaktikus elemzés alapfogalmai

- Legyen G=(T,N,S,P) egy grammatika. Ha, $S \rightarrow^* \alpha$ akkor az α-t *mondatformának* nevezzük.
- Ha $S \rightarrow^* x$, akkor az x a grammatika egy monda-ta.
- Legyen G=(T,N,S,P) egy grammatikának $\alpha = \alpha_1 \beta \alpha_2$ egy mondatformája. A β-t az α egy *részmon-datának* nevezzük, ha van olyan *A* szimbólum, melyre $S \rightarrow^* \alpha_1 A \alpha_2$ és $A \rightarrow^* \beta$.
- A β egy egyszerű részmondata α-nak, ha a fen-tiekben az $A \rightarrow B$ teljesül.

Példa

- Tekintsük a következő nyelvtant:
  G = ({i,+, ∗,(,)} , {E,F,T} , E,P) ahol a P a következő szabályokat tartalmazza:

$$E \rightarrow T \mid E + T$$
$$T \rightarrow F \mid T * F$$
$$F \rightarrow i \mid (E)$$

Ekkor az E+T∗i+T∗F mondatformának az i+T vagy az i+T∗F nem részmondata, de az E+T∗i vagy a T∗i egy részmondata, és a T∗F egy egyszerű részmondata.

Definíciók

- Egy mondatforma legbaloldalibb egyszerű részmondatát a mondatforma *nyelének* nevezzük.
- A mondat szintaxisfájának levelei a nyelvtan terminális szimbólumai, a szintaxisfa többi eleme a nemterminális szimbólumokat reprezentálja, a gyökéreleme pedig a nyelvtan kezdőszimbóluma.

A szintaktikus elemzés alapkövetelményei

- A *nemegyértelmű nyelvtanban* van olyan mondat, amelyhez több szintaxisfa is tartozik. Ez az elemzés szempontjából azt jelenti, hogy ezt a mondatot többféleképpen is lehet elemezni, azaz a különböző elemzésekhez különböző tárgyprogramok tartozhatnak.
- A G nyelvtanra az alábbi feltételek teljesülését követeljük meg:
  - Ciklusmentes.
  - Redukált, azaz nincs benne „felesleges” nemterminális szimbólum.
  - ε mentes.

A szintaktikus elemzés feladata

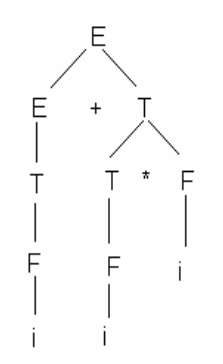
- Ha adott egy program akkor a szintaktikus elemzés feladata az, hogy eldöntse a program a nyelv egy mondata-e. A szintaktikus elemzőnek meg kell határoznia a programhoz tartozó szintaxisfát, ismerve a szintaxisfa gyökérelemét és a leveleit, elő kell állítania a szintaxisfa többi elemét és éleit, vagyis meg kell határoznia a program egy levezetését. Ha ez sikerül, akkor azt mondjuk, hogy a program eleme a nyelvnek, azaz a program szintaktikusan helyes.

Szintaxisfa

- A szintaxisfa felépítésére két lehetőség van:
  - Felülről-lefelé: Az S szimbólumból kiindulva építjük fel a szintaxisfát.
  - Alulról-felfelé: A szintaxisfa felépítése a levelekből kiindulva halad az S szimbólum felé.
- ***Legbaloldalibb helyettesítés*:** Ha $A \rightarrow \alpha$, akkor az $xA\beta$ mondatforma legbaloldalibb helyettesítése $x\alpha\beta$.
- ***Legjobboldalibb helyettesítés*:** A βAx mondatforma *legjobboldalibb helyettesítése* $\beta\alpha x$.

Szintaxisfa 2

- Ha az S→*x levezetésben minden helyettesítés legbaloldalibb helyettesítés, akkor ezt a levezetést *legbaloldalibb levezetésnek* nevezzük.
- Ha a levezetésben minden helyettesítés legjobboldalibb helyettesítés akkor ezt a levezetést *legjobboldalibb levezetésnek* nevezzük.



Példa

- Az előző ábrán látható i+i*i kifejezés legbaloldalibb és legjobboldalibb levezetése a következő:
  - Legbaloldalibb helyettesítés:

$E \rightarrow E+T \rightarrow T+T \rightarrow F+T \rightarrow i+T \rightarrow I+T*F \rightarrow i+F*F \rightarrow i+i*F \rightarrow i+i*i$

  - Legjobboldalibb helyettesítés:

$E \rightarrow E+T \rightarrow E+T*F \rightarrow E+T*i \rightarrow E+F*i \rightarrow E+i*i \rightarrow T+i*i \rightarrow F+i*i \rightarrow i+i*i$

Szintaxisfa 3

- A felülről-lefelé elemzés egyik módszere lehetne az, hogy előállítjuk az összes lehetséges szintaxisfát, azaz az összes lehetséges mondatot, és ha az elemezendő szöveg megegyezik egy mondattal, akkor a mondathoz tartozó szintaxisfáról az elemzés lépései leolvashatók.
- Egy másik módszer: Kiindulunk a kezdőszimbólumból és megpróbálunk legbaloldalibb helyettesítések egymás utáni alkalmazásával eljutni az elemezendő szöveghez.

Tétel

- **Tétel:** Ha $S \rightarrow^* x\alpha \rightarrow^* yz$, ahol $|x| = |y|$, akkor $x = y$.
- **Biz:** Az állítás triviális, mivel egy mondatforma baloldalán a terminálisokból álló x jelsorozatot a környezetfüggetlen nyelvtan helyettesítési szabályai nem változtat-ják meg.

Szintaxisfa 4

- Ha a szintaxisfa építésekor a baloldali terminálisok nem egyeznek meg az elemezendő szöveg baloldalán álló terminálisokkal, akkor a szintaxisfa felépítése már biztosan nem lesz jó. Ekkor egy lépést vissza kell lépni és más helyettesítési szabályt kell alkalmazni.
- Ha alulról-felfelé elemezünk, akkor minden lépésben a mondatforma nyelét kell visszahelyettesíteni a hozzátartozó nemterminális szimbólumra.

Balsarkos elemzés

- Az alulról-felfelé és a felülről-lefelé elemzésekel kívül más módszerek is léteznek, mint például a ***balsarkos elemzés***.
- Egy helyettesítési szabály *balsarkának* nevezzük a jobboldalán álló első szimbólumot.
- Az elemzés menete: A szövegben balról-jobbra, alulról-felfelé haladva meghatározzuk a balsarkokat, majd felülről-lefelé elemzéssel a szintaxisfa többi részét. A felülről-lefelé elemzéshez a helyettesítési szabályok nem balsarkos elemeit egy veremben tárolhatjuk.

Felülről-lefelé elemzések

Teljes visszalépéses algoritmus

- Az *S* szimbólum helyettesítésére az első olyan szabályt alkalmazzuk, amelynek baloldalán az *S* áll.
- A legbaloldalibb nemterminálist a nemterminális első helyettesítési szabályával helyettesítjük. Az új mondatformákra ezt a műveletet ismételjük addig, amíg lehetséges.
- Két eset van, amikor az előző pont művelete nem alkalmazható tovább:
  - A mondatforma baloldalán álló terminálisok nem egyeznek meg az elemezendő szöveg prefixével.
  - Nincs több terminális a mondatformában. → Sikeres

- Ha a terminálisok nem egyeznek meg, akkor lépjünk egy helyettesítést vissza.
- Ha az **A** nemterminális szimbólumnak nincs már több helyettesítési szabálya, akkor lépjünk vissza az *A* helyettesítését megelőző lépésre, és folytassuk az elemzést úgy, hogy a következő helyettesítési szabályt alkalmazzuk.
- Ha egy ilyen visszalépéskor az **S** szimbólumhoz jutunk vissza, és nincs már az **S**-nek további helyettesítési szabálya, akkor az elemzést befejeztük, az elemezendő szöveg nem mondata a nyelvtan által definiált nyelvnek.

Példa

- **G = ({ A, B, S}, { a, b, c, d } , P, S)** nyelvtan helyettesítési szabályai a következők:

  S → aAd | aB

  A → b | c

  B → ccd | ddc

  és elemezzük az **accd** szöveget.

Jelöljük az elemezendő szövegben egy ponttal azt, hogy balról jobbra haladva meddig jutottunk el.

Az elemzés lépései:

S .accd
aAd a.ccd
abd a.ccd

A mondat nem azonos az elemezendő szöveggel, vissza kell lépni.

aAd a.ccd
acd acc.d

A mondat nem azonos az elemezendő szöveggel, vissza kell lépni.

aAd a.ccd

A-ra nincs több helyettesítési szabály, vissza kell lépni.

S .accd
aB a.ccd
accd accd. Az elemzés sikeres.

A teljes visszalépéses elemző algoritmusa

- Az algoritmus két vermet használ, az egyikben a vizsgált mondatforma van, a másik az elemzés lépéseinek adatait tartalmazza.
- Az elemzés algoritmusa műveletek szabályhalmazával van leírva, a műveletek az input szimbólumsorozat és a két verem közötti információcserét írják le.
- Az elemzés eredménye az, hogy vagy szintaktikus hiba van az input szövegben, vagy az elemzett szöveg a nyelv egy mondata. Ha az elemzett szöveg szintaktikusan helyes, akkor a mondat szintaxisfáját is megkapjuk.

- Ha a nyelvtanban van $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_k$ helyettesítési szabály, akkor jelöljük az alternatívák sorszámát a szabály baloldalán is: $A_1 \to \alpha_1 \mid A_2 \to \alpha_2 \mid \ldots \mid A_k \to \alpha_k$ ahol az $A_1, A_2, \ldots, A_k$ mindegyike az A szimbólumot jelöli, és az index csak a szabályok megkülönböztetésére szolgál.
- Az elemzés állapotait az **(S,i,α,β)** négyesekkel írjuk le, ahol az
  - **S:** Az állapot típusa. Ez lehet **q** = normál állapot, **b** = visszalépés , **t** = az elemzés vége.
  - **i:** Az input szövegre mutató pointer.
  - **α:** A vizsgált mondatforma elemzésének történetét tartalmazó verem tartalma.
  - **β:** A vizsgált mondatformát tartalmazó verem tartalma.

- Kezdőállapot: $(q,1,\varepsilon,S\#)$.
- Az állapotátmenetek a következők lehetnek:
  - Szintaxisfa építése az első szabállyal:

    Ha $A_1 \to \gamma_1$, akkor $(q,i,\alpha,A\beta) \to (q,i,\alpha A_1,\gamma_1\beta)$.
  - Az input szimbólum olvasása:

    Ha $c_i=a$, akkor $(q,i,\alpha,a\beta) \to (q,i+1,\alpha a,\beta)$.
  - **Az input szimbólum nem azonos a vizsgált szimbólummal:**

    Ha $c_i \neq a$, akkor $(q,i,\alpha,a\beta) \to (b,i,\alpha,a\beta)$.
  - Visszalépés az input szövegben:

    $(b,i,\alpha a,\beta) \to (b,i-1,\alpha,a\beta)$.

- A következő alternatív helyettesítési szabály keresése:
  - Ha i=1, A=S és az S-nek csak j helyettesítési szabálya van akkor az algoritmus szintaktikus hiba detektálásával befejeződik.

    $(b,1,\alpha S_j,\gamma_j\beta) \to (t,1,\alpha S_j,\gamma_j\beta)$
  - Ha van $A_{j+1} \to \gamma_{j+1}$ szabály, akkor

    $(b,i,\alpha A_j,\gamma_j\beta) \to (q,i,\alpha A_{j+1},\gamma_{j+1}\beta)$.
  - Egyébként, azaz ha az A minden alternatíváját már felhasználtuk $(b,i,\alpha A_j,\gamma_j\beta) \to (b,i,\alpha,A\beta)$.
- A sikeres befejezés: $(q,n+1,\alpha,\#) \to (t,n+1,\alpha,\varepsilon)$.

Példa

- $G = (\{E,E`,T,T`,F\},\{i,+,*,(,)\},P,E)$ ahol a P helyettesítési szabályok a következők:
  $E \to T|TE`$, $E` \to +T|+TE`$, $T \to F|FT`$, $T` \to *F|*FT$, $F \to i|(E)$.
- Elemezzük az i+i szöveget.

$(q,1,\varepsilon,E\#)$	1.	$(q,1,E_1,$ $T\#)$
	1.	$(q,1,E_1T_1,$ $F\#)$
	1.	$(q,1,E_1T_1F_1,$ $i\#)$
	2.	$(q,2,E_1T_1F_1i,$ $\#)$
	3.	$(b,2,E_1T_1F_1i,$ $\#)$
	4.	$(b,1,E_1T_1F_1,$ $i\#)$

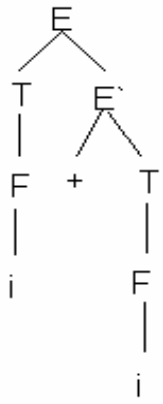
5b.	$(q,1,E_1T_1F_2,$	$(E)\#)$
3.	$(b,1,E_1T_1F_2,$	$(E)\#)$
5c.	$(b,1,E_1T_1,$	$F\#)$
5b.	$(q,1,E_1T_2,$	$FT`\#)$
1.	$(q,1,E_1T_2F_1,$	$iT`\#)$
2.	$(q,2,E_1T_2F_1i,$	$T`\#)$
1.	$(q,2,E_1T_2F_1iT`_1,$	$*T\#)$
3.	$(b,2,E_1T_2F_1iT`_1,$	$*T\#)$
5b.	$(q,2,E_1T_2F_1iT`_2,$	$*FT\#)$
3.	$(b,2,E_1T_2F_1iT`_2,$	$*FT`\#)$
5c.	$(b,2,E_1T_2F_1i,$	$T`\#)$
4.	$(b,1,E_1T_2F_1,$	$iT`\#)$
5c.	$(q,1,E1T_2F_2,$	$(E)T`\#)$
3.	$(b,1,E_1T_2F_2,$	$(E)T\#)$

5c.	$(b,1,E_1T_2,$	$FT`\#)$
5c.	$(b,1,E_1,$	$T\#)$
5b.	$(q,1,E_2,$	$TE`\#)$
1.	$(q,1,E_2T_1,$	$FE`\#)$
1.	$(q,1,E_2T_1F_1,$	$iE`\#)$
2.	$(q,2,E_2T_1F_1i,$	$E`\#)$
1.	$(q,2,E_2T_1F_1iE`_1,$	$+T\#)$
2.	$(q,3,E_2T_1F_1iE`_1+,$	$T\#)$
1.	$(q,3,E_2T_1F_1iE`_1+T_1,$	$F\#)$
1.	$(q,3,E_2T_1F_1iE`_1+T_1F_1,$	$i\#)$
2.	$(q,4,E_2T_1F_1iE`_1+T_1F_1i,$	$\#)$
6.	$(t,4,E_2T_1F_1iE`_1+T_1F_1i,$	$\varepsilon)$

- A szintaxisfa felépítése a harmadik elemből kiolvasható:

  $E_2 \rightarrow TE`$, $T_1 \rightarrow F$, $F_1 \rightarrow i$, $E`_1 \rightarrow +T$, $T_1 \rightarrow F_1$, $F_1 \rightarrow i$.

- A teljes visszalépéses elemzés rendkívül lassú. A szintaktikus hiba csak akkor derül ki, ha már minden lehetséges esetet végigpróbált.



Korlátozott visszalépéses elemzés

- Tegyük fel, hogy léteznek $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ helyettesítési szabályok. Tegyük fel továbbá, hogy a teljes visszalépéses algoritmus során eljutottunk egy $S \rightarrow^* xA\beta$ levezetéshez. Ezután a $A \rightarrow \alpha_k$ helyettesítést alkalmazva: $S \rightarrow^* xA\beta \rightarrow x\,\alpha_k\beta \rightarrow^* xy\beta$. Ha ez elemzendő szöveg prefixe xy, akkor $A \rightarrow \alpha_k$ jó helyettesítés volt.

Korlátozott visszalépéses elemzés

- De, ha $S \rightarrow^* xy\beta \rightarrow^* xyv\gamma$, ahol xyv már nem prefixe az elemzendő szövegnek, akkor visszalépéseket kell végrehajtani.
- Teljes visszalépéses elemzésnél ekkor $A \rightarrow \alpha_{k+1}$ lépést alkalmazzuk.
- Korlátozott visszalépéses elemzéskor nem lépünk vissza, hanem szintaktikus hibával leáll az algoritmus.

Példa

- G = ({A,B,S},{a,b,c},P,S) ahol a P helyettesítési szabályok a következők: S→AB, A→a|aa, B→b|ac.
- Elemezzük az aab szöveget. Erre a szintaktikusan helyes kifejezésre hibát jelez az algoritmus, mivel

  S→AB→aB

  levezetés után B egyik helyettese sem alkalmazható. Az A→aa szabályt pedig nem alkalmazzuk a mondatforma a szimbóluma miatt.

LL(k) nyelvtanok és elemzések

- **LL(k) nyelvtan:** Amikor **k** szimbólum előreolvasásával az elemzés lépései egyértelműen meghatározhatók, azaz visszalépésekre nincs szükség. Gyakorlatban a k=1 elemzők a fontosak (Left to right, tracing a Leftmost derivation).

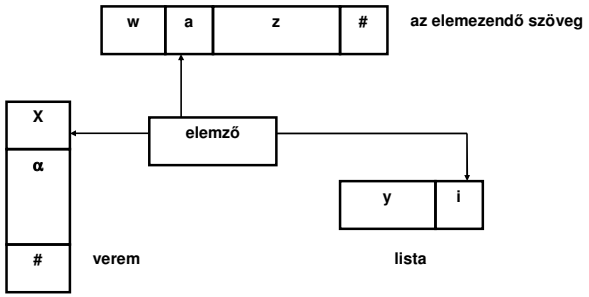
Egyszerű LL(1) nyelvtan

- A G nyelvtant egyszerű LL(1) nyelvtannak nevezzük, ha **ε**-mentes és ha minden A nemterminális szimbólumra a helyettesítési szabályok különböző terminális szimbólummal kezdődnek, azaz $k \geq 1$-re $A \rightarrow a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_k\alpha_k$, ahol $a_i \neq a_j$, ha $i \neq j$.

Példa

- **G = {A,B,S},({a,b,c,d},P,S)**, ahol a helyettesítési szabályok a következők:

  S→ aS | bA , A→ d | ccA
- Látható, hogy a fenti nyelvtan egy egyszerű LL(1) nyelvtan.
- Az elemzés állapotait az **(x,β,y)** hármassal jelöljük, ahol
  - **x**: A még nem elemzett szöveg.
  - **β**: Verem.
  - **Y**: Listát jelent.



- Az elemzéshez egy vermet fogunk használni. A verem alját # jellel jelöljük. Ebbe a verembe helyezzük el az elemzés folyamán az aktuális mondatformát.
- A verem kezdeti tartalma: S
- A verem tetején levő szimbólumot fogjuk összehasonlítani az input szöveg következő, még nem elemzett szimbólumával.
- Az elemzést egy elemző tábla segítségével fogjuk végezni. A tábla sorai a verem tetején álló szimbólumot, az oszlopai pedig a következő elemezendő szimbólumot jelölik.

	pop	ha X=a
	accept	Ha X=# és a=#
M(X,a) =	(aα,i)	Ha x→aα az i-edik helyettesítési szabály
	error	Egyébként

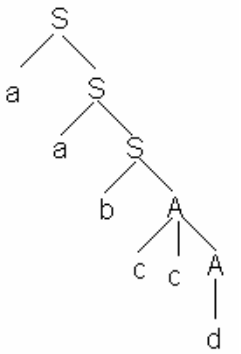
	a	b	c	d	#
S	(aS,1)	(bA,2)			
A			(ccA,4)	(d,3)	
a	pop				
b		pop			
c			pop		
d				pop	
#					accept

Példa

- Elemezzük az **aabccd** szöveget.

(aabccd#, S#, ε)	(aS,1)	(aabccd#, aS#, 1)
	pop	(abccd#, S#, 1)
	(aS,1)	(abccd#, aS#, 11)
	pop	(bccd#, S#, 11)
	(bA,2)	(bccd#, bA#, 112)
	pop	(ccd#, A#, 112)
	(ccA,4)	(ccd#, ccA#, 1124)
	pop	(cd#, cA#, 1124)
	pop	(d#, A#, 1124)
	(d,3)	(d#, d#, 11243)
	pop	(#, #, 11243)
	accept	o.k.

- Az **aabccd** mondat szintaxisfája az elemzés végállapotának harmadik komponense alapján a következő ábrán látható:



ε-mentes LL(1) nyelvtan

- **FIRST(α) = {a| α→*aβ }** azaz a **FIRST(α)** halmaz tartalmazza azokat a terminális szimbólumokat, amelyek az α -ból levezethető részmondatok baloldalán állnak.
- G ε-mentes LL(1) nyelvtan ⇔ ha G ε-mentes és minden A nemterminális szimbólum esetén **$FIRST(\alpha_i) \cap FIRST(\alpha_j)=0$** ha **$i \neq j$.**

Példa

- **G=({A,B,S}, {a,b,c,d,e},P,S)**, ahol a helyettesítési szabályok a következők:
  - **S→ ABe**
  - **A→ dB|aS|c**
  - **B→ AS|b**
- Ez a nyelvtan egy ε-mentes LL(1) nyelvtan, mivel ε-mentes.
- **FIRST(dB)={d},FIRST(aS)={a}, FIRST(c)={c} FIRST(AS)={a,c,d}, FIRST(b)={b},** és a megfelelő halmazok diszjunktak.

- **Tétel:** Ha a G nyelvtan egyszerű LL(1) nyelvtan, akkor a G egy ε -mentes LL(1) nyelvtan.
- **Tétel:** Ha a G nyelvtan ε -mentes LL(1) nyelvtan, akkor megadható egy vele ekvivalens G` egyszerű LL(1) nyelvtan.

Nem bizonyítjuk.

	pop	ha X=a
	accept	Ha X=# és a=#
M(X,a) =	(aα,i)	Ha x→α az i-edik helyettesítési szabály és a∈ FIRST(α)
	error	Egyébként

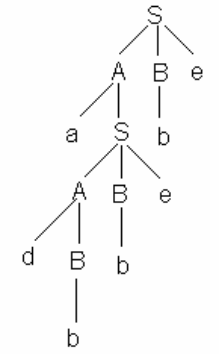
	a	b	c	d	e	#
S	(ABe,1)		(ABe,1)	(ABe,1)		
A	(aS,3)		(c,4)	(dB,2)		
B	(AS,5)	(b,6)	(AS,5)	(AS,5)		
a	pop					
b		pop				
c			pop			
d				pop		
e					pop	
#						accept

Példa

- Elemezzük az **adbbebe** szöveget.

(adbbebe#, S#, ε)	(ABe,1)	**(adbbebe#, ABe#, 1)**
	(aS,3)	**(adbbebe#, aSBe#, 13)**
	pop	**(dbbebe#, SBe#, 13)**
	(ABe,1)	**(dbbebe#, ABeBe#, 131)**
	(dB,2)	**(dbbebe#, dBBeBe#, 1312)**
	pop	**(bbebe#, BBeBe#, 1312)**
	(b,6)	**(bbebe#, bBeBe#, 13126)**
	pop	**(bebe#, BeBe#, 13126)**
	(b,6)	**(bebe#, beBe#, 131266)**
	pop	**(ebe#, eBe#, 131266)**
	pop	**(be#, Be#, 131266)**
	(b,6)	**(be#, be#, 1312666)**
	pop	**(e#, e#, 1312666)**
	pop	**(#, #, 1312666)**
	accept	**o.k.**

- Az **adbbebe** mondat szintaxisfája az elemzés végállapotának harmadik komponense alapján a következő ábrán látható:



LL(k) nyelvtanok

- $S \rightarrow^* \alpha A \beta$, $S \rightarrow uxy$.
- Legyen a $FIRST_k(\alpha)$ az α-ból levezethető részmondatok első k terminális szimbólumsorozatainak halmaza, azaz:

$$FIRST_k(\alpha) = x \mid \alpha \rightarrow^* x, \ |x|<k$$
$$\alpha \rightarrow^* x\beta, |x|=k.$$

- Így a $FIRST_k(x)$ az x első k darab szimbólumát, $|x|<k$ esetén pedig a teljes x-t jelenti.

- **Def:** A G nyelvtan LL(k) nyelvtan, ha
  - $S \rightarrow^* wA\beta \rightarrow w\alpha_1\beta \rightarrow^* wx$
  - $S \rightarrow^* wA\beta \rightarrow w\alpha_2\beta \rightarrow^* wx$
  - és $FIRST_k(x) = FIRST_k(y)$ esetén $\alpha_1=\alpha_2$.

- **1.Tétel:** Ha G egy nem ε-mentes $LL(k)$ nyelvtan, akkor létezik olyan ε-mentes $LL(k+1)$ nyelvtan, amelyik az $L(G)$ nyelvet generálja.

- **2.Tétel:** Ha G egy ε-mentes $LL(k+1)$ nyelvtan, akkor van olyan nem ε-mentes nyelvtan, amelyik az $L(G)$ nyelvet generálja.

- **3.Tétel:** A G nyelvtan akkor és csak akkor $LL(k)$ nyelvtan, ha minden $S \rightarrow^* wA\beta$, és $A \rightarrow \gamma \mid \delta$ esetén

$$FIRST_k(\gamma\beta) \cap FIRST_k(\delta\beta) = 0.$$

- **Bizonyítás:**

$\Rightarrow$ Tegyük fel, hogy G $LL(k)$ nyelvtan.

$FIRST_k(\gamma\beta) \cap FIRST_k(\delta\beta) = X$

$S \rightarrow^* wA\beta \rightarrow w\gamma\beta \rightarrow^* wxy$

$\Rightarrow \gamma = \delta.$

$S \rightarrow^* wA\beta \rightarrow w\delta\beta \rightarrow^* wxz$

$\Leftarrow$ Tegyük fel, hogy G **nem** $LL(k)$ nyelvtan.

$S \rightarrow^* wA\beta \rightarrow w\gamma\beta \rightarrow^* wx$

$\Rightarrow FIRST_k(x)=FIRST_k(y).$

$S \rightarrow^* wA\beta \rightarrow w\delta\beta \rightarrow^* wy$

$A \rightarrow \gamma \quad FIRST_k(\gamma\beta)$

$A \rightarrow \delta \quad FIRST_k(\delta\beta)$ **Ellentmondás!**

$FOLLOW_k(\beta)$

- Legyen $FOLLOW_k(\beta)$ ($k \geq 1$) a β-t tartalmazó mondatformák β utáni szimbólumsorozatainak k hosszúságú terminális prefixeiből álló halmaz, azaz

  $FOLLOW_k(\beta)=\{x \mid S \rightarrow^* \alpha\beta\gamma$ és $x \in FIRST_k(\gamma)\}$, és ha $\varepsilon \in FOLLOW_k(\beta)$, akkor legyen $FOLLOW_k(\beta)=FOLLOW_k(\beta) \setminus \{\varepsilon\} \cup \{\#\}$.

- **4. Tétel:** A G nyelvtan akkor és csak akkor LL(1) nyelvtan, ha minden A nemterminális szimbólumra $A \to \gamma \mid \delta$ esetén

$FIRST_1(\gamma\ FOLLOW_1(A)) \cap FIRST_1(\delta\ \ FOLLOW_1(A)) = 0$

- Bizonyítás:

$\Rightarrow$"a" a közös elem.

1. $a \in FIRST_1(\gamma)$, $a \in FIRST_1(\delta)$
2. $a \in FIRST_1(\gamma)$, $a \in FIRST_1(\delta)$ és $a \in FOLLOW_1(A)$
3. $\varepsilon \in FIRST_1(\gamma)$, $a \in FIRST_1(\delta)$ és $a \in FOLLOW_1(A)$
4. $\varepsilon \in FIRST_1(\gamma)$, $\varepsilon \in FIRST_1(\delta)$ és $a \in FOLLOW_1(A)$

$\Leftarrow$ 1-4. közül legalább egy teljesül. Ez ellentmond a metszet üresség.

- **Def:** G nyelvtan erős LL(k) nyelvtan, ha
  - $S \to^* wA\beta \to w\alpha_1\beta \to^* wx$
  - $S \to^* wA\gamma \to w\alpha_2\gamma \to^* vy$
  - és $FIRST_k(x) = FIRST_k(y) \Rightarrow \alpha_1 = \alpha_2$.
- **5. Tétel:** A G nyelvtan akkor és csak akkor erős LL(k) nyelvtan, ha minden $A \to \gamma \mid \delta$ esetén $FIRST_k(\gamma\ FOLLOW_k(A)) \cap FIRST_k(\delta\ \ FOLLOW_k(A)) = 0$
- **Példa:** G=({a,b},{A,S},S,P) ahol a helyettesítési szabályok a következők:
  - $S \to aAaa \mid bAba$
  - $A \to b \mid \varepsilon$

  Ez LL(2) nyelvtan, de nem erős.

  $FIRST2(b\ FOLLOW_2(A)) \cap FIRST_2(\delta\ \ FOLLOW_2(A)) = \{ba\}$ miatt nem erős LL(2).

LL(1) nyelvek

- **Def:** $A \to \gamma$

$LEFT_1(A,\gamma) = \{FIRST_1(x) \mid S \to^* wA\beta \to^* w\gamma\beta \to^* wx\}$

- **Def:** Az „A" balfaktorizált, ha $\forall A \to \gamma \mid \delta$

$LEFT_1(A,\gamma) \cap LEFT_1(A,\delta) = 0$.

- **Tétel:** Ha a G nyelvtan egy LL(k) nyelvtan, akkor nem lehet balrekurzív nyelvtan.
- **Tétel:** Minden környezetfüggetlen balrekurzív G nyelvtanhoz megadható olyan vele ekvivalens G` nyelvtan, amelyik balrekurzív mentes.

- **Def:** Egy **G** nyelvtan akkor és csak akkor balfaktorizált, ha **LL(1)** nyelvtan.

- **Példa:** **G=({a,b,c,d},{A,B,S},S,P),** ahol a helyettesítési szabályok:
  - **S→A**
  - **A→Bc|Bd**
  - **B→a|bB**

  G`=({a,b,c,d},{A,A`,B,S},S,P), ahol a helyettesítési szabályok:
  - **S→A**
  - **A→BA`**
  - **A→c|d**
  - **B→a|bB**

$M(x,a) =$

pop	ha x=a
accept	ha x=# és a=#
(α,i)	Ha $x\rightarrow\alpha$ az i-edik szabály és $a\in FIRST_1(\alpha)$
(α,i)	Ha $x\rightarrow\alpha$ az i-edik szabály és $\varepsilon\in FIRST_1(\alpha)$ és $a\in FOLLOW_1(X)$
error	egyébként

a) $FIRST_1(\alpha)$

1. $FIRST_1(X)=0$
2. $X\in t$ $FIRST_1(X)= FIRST_1(X) \cup \{X\}$
3. $X\rightarrow\varepsilon$ $FIRST_1(X)= FIRST_1(X) \cup \{\varepsilon\}$
4. $X\rightarrow Y_1\ldots Y_m$ és $Y_1\ldots Y_k\rightarrow^*\varepsilon$ $(1\leq k<m)$ akkor $FIRST_1(X)= FIRST_1(X) \cup \{a\}$

Ha k=m, akkor
$FIRST_1(X)= FIRST_1(X) \cup \{\varepsilon\}$

1. $FIRST_1(\alpha)= FIRST_1(X_1) \setminus \{\varepsilon\}$
2. $\varepsilon\in FIRST_1(X_1)$, akkor $FIRST_1(\alpha)= FIRST_1(\alpha) \cup \{FIRST_1(X_2) \setminus \{\varepsilon\}\}$
3. $\varepsilon\in FIRST_1(X_1) \cup FIRST_1(X_2)$,akkor $FIRST_1(\alpha)= FIRST_1(\alpha) \cup \{FIRST_1(X_3) \setminus \{\varepsilon\}\}$
4. $\varepsilon\in FIRST_1(X_1)$ minden i-re, akkor $FIRST_1(\alpha)= FIRST_1(\alpha) \cup \{\varepsilon\}$

b) $FOLLOW_1(A)$

1. Legyen # $\in FOLLOW_1(S)$ és minden más nemterminális **A**-ra $FOLLOW_1(A)=0$;
2. Ha van $B\rightarrow\alpha A\beta$ szabály, akkor legyen $FOLLOW_1(A) = FOLLOW_1(A) \cup (FIRST_1(\beta) \setminus \{\varepsilon\})$
3. Ha van $B\rightarrow\alpha A\beta$ szabály, melyre $\beta\rightarrow^*\varepsilon$, akkor legyen $FOLLOW_1(A)= FOLLOW_1(A) \cup FOLLOW_1(\beta)$
4. Ha van $B\rightarrow\alpha A$ szabály, akkor legyen $FOLLOW_1(A)= FOLLOW_1(A) \cup FOLLOW_1(\beta)$

Jelöljük azoknak a nemterminális szimbólumoknak a halmazát, amelyekből az ε levezethető, N_ε-nal, és vezessük be az F relációt a következőképpen: ha $A\rightarrow x_1, x_2, \ldots, x_n$, akkor AFX_1, ha $x_1\in N_\varepsilon$, akkor AFX_2, ha $x_1, x_2\in N_\varepsilon$, akkor AFX_3 és így tovább. A $FIRST_1(A)$ az F^+-ból meghatározható, $FIRST_1(A)=\{a|AF^+a\} \cup \{\varepsilon|A\in N_\varepsilon\}$

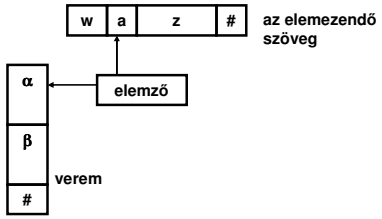
- A $\mathbf{FOLLOW_1(A)}$ meghatározásához további két reláció szükséges, ezek a relációk legyenek B és L.
  Ha $\mathbf{A \rightarrow x_1, x_2, \ldots, x_n}$, akkor legyen $\mathbf{x_i B x_{i+1}}$
  ha $\mathbf{x_{i+1}} \in N_\varepsilon$, akkor legyen $\mathbf{x_i B x_{i+2}}$
  ha $\mathbf{x_{i+1}}, \mathbf{x_{i+2}} \in N_\varepsilon$, akkor legyen $\mathbf{x_i B x_{i+3}}$ és így tovább.

  Ha $\mathbf{A \rightarrow x_1, x_2, \ldots, x_n}$, akkor legyen $\mathbf{X_n LA}$, ha $\mathbf{x_n} \in N_\varepsilon$, akkor $\mathbf{X_{n-1} LA}$, ha $\mathbf{X_{n-1}}, \mathbf{X_n} \in N_\varepsilon$, akkor $\mathbf{X_{n-2} LA}$ és így tovább.

  Az $a \in \mathbf{FOLLOW_1(A)}$ terminálisokat az $\mathbf{AL^*X}$, $\mathbf{XBY}$, $\mathbf{YF^*}$ relációkból lehet meghatározni azaz $\mathbf{FOLLOW_1(A) = \{a \mid A(L^*BF^*)a\}}$.

Alulról-felfelé elemzések

- A felülről-lefelé elemzések a gyökérből kiindulva a levelek felé haladva építik fel a szintaxisfát. Az alulról-felfelé elemzések ezt éppen fordított irányban teszik.
- Az alulról-felfelé elemzések egy vermet használnak. A vizsgált elemzések mindegyike léptetés-redukálás típusú elemzés lesz. Az elemző először mindig egy redukciót akar végrehajtani, azaz azt vizsgálja, hogy a verem tetején és az alatta levő szimbólumokból alkotott α mondatforma megfelel-e valamely szabály jobboldalának.



- Ha a redukció nem lehetséges, akkor az input szimbólumsorozat következő ***a*** elemét lépteti, azaz helyezi a verem tetejére, és megismétli a redukció lehetőségének vizsgálatát.

A visszalépéses elemzés

- Az elemzés redukciókon és léptetéseken keresztül eljut a vizsgált szimbólumsorozat végére. Ha ekkor a szintaxisfa építése is eljut az S kezdőszimbólumig, akkor az elemzésnek vége van, az elemzett szöveg eleme a nyelvtan által definiált nyelvnek. Ha a szintaxisfa még nincsen készen, akkor más úton kell megkísérelni a szintaxisfa felépítését, azaz visszalépéseket kell végrehajtani.
- A visszalépésekkel az utoljára végrehajtott redukcióig vissza kell menni. Ekkor meg kell kísérelni egy másik szabály szerinti redukció végrehajtását, és ezzel a mondatformával a szintaxisfa felépítését. Ha nincs több helyettesítési szabály akkor a redukció helyett egy léptetést kell végrehajtani.

- Ezt az eljárást kell folytatni addig amíg vagy a sikeres elemzés állapotába jutunk vagy már visszalépéseken keresztül eljutottunk odáig, hogy ismét a #-ra mutat a pointer, és nincs redukció, nem lehet hová visszalépni. Ekkor a szöveg nem eleme a nyelvnek.

- **Def:**A G nyelvtanban az A szimbólumot jobbrekurzív szimbólumnak nevezzük, ha $A \rightarrow^{+} \alpha A$. A G nyelvtant jobbrekurzív nyelvtannak nevezzük, ha legalább egy jobbrekurzív szimbóluma van.

A visszalépéses elemző algoritmusa

- Elve megegyezik a teljes visszalépéses felülről lefelé történő elemzés elvével. Eltérés csak az elemzés állapotaiban és az állapotát-meneteket leíró műveleti szabályokban van. A helyettesítési szabályokat 1-től kezdve sorszámozzuk.

- Az elemzés állapotait az (s,i,α,β) négyesekkel írjuk le.
  - **s**: Az állapot típusa. Ez lehet q, b és t.
  - **i**: Az input szövegre mutató pointer.
  - **α**: Verem mely α-t a vizsgált mondatformát tartalmazza.
  - **β**: Szintén verem, mely β-t a vizsgált mondatforma kialakulásának történetét tartalmazza.

- A kezdőállapot legyen $(q,1,\#,\varepsilon)$. Az algoritmus olyan állapotba visz át melyben az elemzés vége jelzés van. Ekkor az elemzés vagy sikeres vagy a vizsgált szöveg nem eleme a nyelvnek.

- Az $(s,i,\alpha,\beta) \rightarrow (s`,i`,\alpha`,\beta`)$ állapotátmenetek, az elemzés lépései a következők:

1. **Redukció:**
   ha $A \rightarrow \gamma$ a j-ik szabály akkor $(q,i,\alpha\gamma,\beta) \rightarrow (q,i,\alpha A,j\beta)$.

   Ha a redukció végrehajtható, akkor ezzel az állapotátmenettel megkapott új állapotra ismét a redukálást kell alkalmazni.
2. **Léptetés:**
   ha $c_i=a$, akkor $(q,i,\alpha,\beta) \rightarrow (q,i+1,\alpha a,s\beta)$ ahol s a léptetés műveletét jelenti. Ha a léptetés után $i \neq n+1$, akkor ismét az 1. pontban leírt redukciót kell megkísérelni. Ha az i pointer a #-ra mutat és a harmadik komponens #S, akkor az elemzés sikeres.
3. **Sikeres befejezés:**
   $(q,n+1,\#S,\beta) \rightarrow (t,n+1,\#S,\beta)$

Ha eljutottunk a # jelig de a 3. pont művelete nem alkalmazható akkor visszalépés következik.

4. **Visszalépés művelet kijelölése:**
   $(q,n+1,\alpha,\beta) \rightarrow (b,n+1,\alpha,\beta)$

5. **A visszalépés végrehajtása:**

a) Ha az $A \rightarrow \gamma$ a j-edik helyettesítési szabály, $B \rightarrow \delta$ egy még nem alkalmazott, k-adik sorszámú szabály, és $\alpha\gamma=\alpha`\delta$ azaz a $\delta$ az $\alpha\gamma$ **prefixe, akkor** $(b,i,\alpha A,j\beta) \rightarrow (q,i,\alpha` B,k\beta)$
Ez azt jelenti, ha a vizsgált mondatformára van másik lehetséges redukció, akkor ezt hajtsuk végre. Ez után ismét az 1. pont következik.

b) Ha i=n+1, az $A \rightarrow \gamma$ a j-edik helyettesítési szabály és az $\alpha\gamma$–ra nincs másik olyan szabály melynek jobboldala az $\alpha\gamma$ prefixe, akkor $(b,i,\alpha A,j\beta) \rightarrow (b,n+1,\alpha\gamma,\beta)$.
Mivel az állapot **b** maradt ezután az 5.pont következik.

c) Ha $i \neq n+1$, az $A \rightarrow \gamma$ a j-edik helyettesítési szabály, és az $\alpha\gamma$–ra nincs másik itt még nem alkalmazott szabály, melynek jobboldala az $\alpha\gamma$ prefixe, valamint $c_i=a$, akkor

$$(b,i,\alpha A,j\beta) \rightarrow (q,i+1,\alpha\gamma a,s\beta)$$

azaz az utoljára végrehajtott redukcióhoz menjünk vissza, állítsuk vissza a redukció előtti állapotot és végezzünk el egy léptetés műveletet. Az elemzés az 1. ponttal folytatódik.

d) Ha az állapot negyedik komponense egy s jellel kezdődik, akkor a léptetés műveletével is vissza kell lépni, azaz

$$(b,i,\alpha a,s\beta) \rightarrow (b,i-1,\alpha,\beta)$$

Az elemzés állapota **b** maradt, az elemzést az 5.ponttal kell folytatni.

6. Ha a fenti 1-5. esetek közül egyiknek sem teljesül a feltétele akkor az elemzés szintaktikus hiba detektálásával befejeződik.

$$(s,i,\alpha,\beta) \rightarrow (t,i,\alpha,\beta)$$

Az i,α és β a kezdőállapot értékét veszi fel. Ha az elemzés a $(t,n+1,\#S,\beta)$ állapottal fejeződik be, akkor a β tartalmazza a szintaxisfa leírását. Ha $\beta=\beta_1,\beta_2,\ldots,\beta_m$ akkor legyen

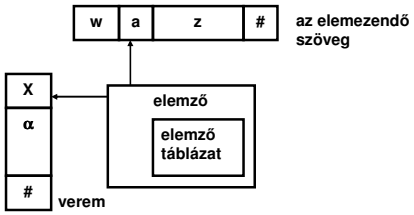
$$h(\beta_i)=\begin{cases}\varepsilon & \text{Ha } \beta_i=s\\ A\rightarrow\gamma, & \text{Ha } \beta_i=j \text{ és } A\rightarrow\gamma \text{ a j-edik szabály.}\end{cases}$$

Ekkor a $h(\beta_1), h(\beta_2),\ldots, h(\beta_m)$ helyettesítési szabályokból álló sorozat a szintaxisfa felépítését adja.

- A visszalépéses elemzés rendkívül lassú, a szintaktikus hiba csak akkor derül ki, ha már minden lehetséges esetet végigpróbáltunk, és az algoritmus nem mond arról semmit, hogy hol van a hiba, mivel a hiba felfedezésekor az i pointer már a szöveg elejére mutat.
- A visszalépések gyorsíthatók, például az állapotok eltárolásával visszalépések esetén azonnal visszaállítható a szükséges korábbi állapot a megfelelő, eltárolt állapot olvasásával.
- A visszalépések számának csökkentésére néhány ellenőrző tevékenység is beépíthető az elemző algoritmusba. Ez azonban nem alkalmazható tetszőleges környezetfüggetlen nyelvtanra.

LR(k) nyelvtanok és elemzések

- Az elemzést **LR(k)** elemzésnek, a nyelvtant **LR(k)** nyelvtannak nevezzük, ahol az **LR** a balról jobbra történő elemzésre utal, a **k** pedig azt jelenti, hogy k szimbólumot előreolvasva egyértelműen meghatározható a mondatforma nyele. Az **LR(k)** elemzés vissza-lépés nélküli típusú elemzés.



- **Def:** Legyen a G=(T,N,S,P) nyelvtanhoz tartozó G` kiegészített nyelvtan a következő: G`=(T,N∪{S`},S`,P∪{S`→S})
- **Def:** Egy G` kiegészített nyelvtan LR(k) nyelvtan (k≥0), ha
  - S`→*αAw →αβw
  - S`→*γBx→ γδx=αβy és
  - $\mathbf{FIRST_k(w)=FIRST_k(y)}$ esetén α=γ, A=B és x=y.
- **Feladat:** Legyen a G=({a},{S`,S},S`,P) nyelvtan P helyettesítési szabályai a következők:
  - S`→S
  - S`→aSa|a

Ekkor minden k-ra
- $S` \rightarrow^* a^k S a^k \rightarrow a^k a a^k = a^{2k+1}$
- $S` \rightarrow^* a^{k+1} S a^{k+1} \rightarrow a^{k+1} a a^{k+1} = a^{2k+3}$ és
- $\mathbf{FIRST_k}(a^k)=\mathbf{FIRST_k}(a a^{k+1})=a^k$ de $a^{k+1} S a^{k+1} \neq a^k S a^{k+2}$

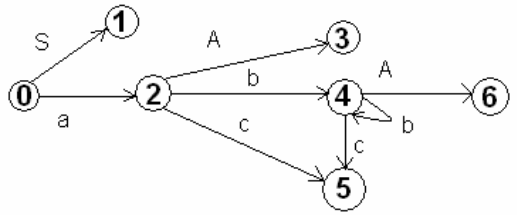
- **Feladat:** G=({a,b},{S`,S},S,P). Ez LR(1) nyelvtan ahol a helyettesítési szabályok:
  - S`→S
  - S`→SaSb|ε
- **Feladat:** G=({a,b,c},{S`,S,A,B},S`,P) nyelvtan helyettesítési szabályai:
  - S`→S
  - S→Ab|Bc
  - A→Aa|ε
  - B→Ba|ε

Ekkor
- $S` \rightarrow S \rightarrow A a^k b \rightarrow a^k b$
- $S` \rightarrow S \rightarrow B a^k c \rightarrow a^k c$ és
- $\mathbf{FIRST_k}(a^k b)=\mathbf{FIRST_k}(a^k c)=a^k$,de A≠B.

- **Tétel:** Minden LL(k) nyelvtan LR(k) nyelvtan, de létezik olyan LR(k) nyelvtan, amelyik nem LL(k`) egyetlen k`-ra sem.
- **Tétel:** Minden LR(k) nyelvtanhoz létezik vele ekvivalens LR(1) nyelvtan.

LR(0) elemzés elve

- **Def:** Legyen az **αβx** mondatforma nyele **β**. Ekkor az **αβ** prefixeit az **αβx járható prefixeinek** nevezzük.
- **Példa:** G=({a,b,c},{A,S,S`},S`,P)
  - S`→S
  - S→aA — 0,2,4 léptetés
  - S→bA — 1,3,5,6 redukció
  - A→c



állapot	action	goto				
		S	A	a	b	c
0	S	1		2		
1	accept					
2	S		3		4	5
3	1					
4	S		6		4	5
5	3					
6	2					

Verem	Input	Járható prefix	Nyél
#0	abbc#		
#0a2	bbc#	a	
#0a2b4	bc#	ab	
#0a2b4b4	c#	abb	
#0a2b4b4c5	#	abbc	c
#0a2b4b4A6	#	abbA	bA
#0a2b4A6	#	abA	bA
#0a2A3	#	aA	aA
#0S1	#	S	S
#0	accept		

Az LR(0) elemző felépítése

- **Def:** Legyen a **"."** karakter egy metszetszimbólum. Ha egy **G** nyelvtan helyettesítési szabálya **A→αβ** , akkor a nyelvtan **LR(0)**–eleme legyen **[A→α.β]**
- Ha a nyelvtan egy szabályának jobboldala n szimbólumot tartalmaz, akkor a definíció alapján a szabályból **n+1** darab **LR(0)**–elem képezhető. Az **A→ε** szabályhoz az **[A→ .]** **LR(0)**–elem tartozik.
- **Def:** Egy G nyelvtan **[A→α.β]** LR(0)-eleme érvé-nyes a **γα** járható prefixre nézve, ha **S`→*γAx →γαβx**

- **Def:** Legyen az **τ** halmaz egy nyelvtan egy LR(0)-elemhalmaza. Ekkor a **closure(τ)** halmaz tartalmazza a következő LR(0)-elemeket:
- Az τ halmaz minden eleme legyen eleme a closure(τ) halmaznak is.
- Ha **[A→α.Bβ]** ∈ closure(τ) és B→γ a nyelvtan egy helyettesítési szabálya, akkor legyen [B→.γ] ∈ closure(τ).
- A closure(τ) halmazt a 2. pontban leírt művelettel addig kell bővíteni ameddig az lehetséges.

- **Def:** Legyen az **τ** halmaz egy nyelvtan egy LR(0)-elemhalmaza. Ekkor a read(τ,X) halmaz tartalmazza a következő LR(0) elemeket:
1. Ha **[A→α.Xβ] ∈ τ** akkor a **closure([A→αX.β])** minden eleme legyen a **read(τ,X)**.
2. A **read(τ,X)** halmazt az (1.) a művelettel addig kell bővíteni ameddig az lehetséges.
- **LR(0) elemzés nagytétele:** Egy **γ** járható prefix érvé-nyes elemeinek halmaza az a kanonikus elemhalmaz, amelyik az elemző determinisztikus véges automatájának ahhoz az állapotához tartozik, amelyikbe az automata a kezdőállapotból a **γ** hatására kerül.

SLR(1) nyelvtanok és elemzések

- Ha a léptetés vagy a redukció művelete nem áll fenn egy állapotban akkor azt mondjuk, hogy ez az állapot egy **indekvát** állapot azaz az elemzés számára kevés információt tartalmaz. Ezt a hiányos információt egészítjük ki az elemezendő szöveg következő szimbólumának megvizsgálásával, és ha ezáltal olyan információhoz jutunk amellyel az állapotok indekvátsága megszűntethető, akkor a nyelvtant SLR(1) nyelvtannak nevezzük.
- Példa: A **G ={{ i,+,(,)},{E,T,S`} ,S`,P)** nyelvtan LR(0) elemeinek kanonikus halmazai a következők:

τ_0=closure([S`→.E])={[S`→.E],
[E →.T],
[E →.E+T],
[T →.i],
[T →.(E)]}

τ_1=read(τ_0,E)=closure([S`→.E])∪closure([E →E.+T])={[S`→E.],[E →E.+T]}

τ_2=read(τ_0,T)=closure([E→T.])={[E→T.]}.
τ_3=read(τ_0,i)=closure([T→i.])={[T→i.]}.
τ_4=read(τ_0,()=closure([T→(.E)])={[T→(.E)],
[E→.T],
[E→.E+T],
[T→.i],
[T→.(E)]}.
τ_5=read(τ_1,+)=closure([E→E+.T])={[E→E+.T],
[T→.i],
[T→.(E)]}.
τ_6=read(τ_4,E)=closure([T→(E.)])∪closure([E →E.+T])={[T→(E.)],[E →E.+T]}.

read(τ_4,T)=τ_2
read(τ_4,i)=τ_3
read(τ_4,()=τ_4

τ_7=read(τ_5,T)=closure([E→E+T.])={[E→E+T.]}.
read(τ_5,i)=τ_3
read(τ_5,()=τ_4

τ_8=read(τ_6,))=closure([T→(E).])={[T→(E).]}.
read(τ_6,+)=τ_5

- A példában az 1. állapot egy indekvát állapot, mivel az **S`→.E**–ból redukció, az **E → E.+T**-ből pedig egy léptetés következik. Ha a szövegben + jel következik, akkor léptetést kell végrehajtani. Redukciót pedig akkor kell végrehajtani, ha a FOLLOW$_1$(S`) halmaz szimbóluma, azaz a # a következő jel. Mivel a {+} és a FOLLOW$_1$(S`) diszjunkt halmazok, az 1. állapot indekvátsága egy szimbólum előreolvasásával megszűntethető.
- Az elemzés determinisztikus véges automatáját, az *action* és a *goto* táblázattal fogjuk leírni. Mivel egy jel előreolvasása határozza meg az elvégzendő műveletet , az *action* táblázatot bővíteni kell. Az *action* táblázat oszlopaihoz a terminális szimbólumokat rendeljük.

- A goto táblázat oszlopaihoz a terminális és nemterminális szimbólumokat rendeljük és a goto táblázat egy eleme azt jelenti, hogy az adott állapotból az oszlophoz tartozó szimbólum hatására melyik állapotba jutunk.
- A két táblázatot összevonjuk úgy, hogy a *goto* táblázat oszlopaihoz csak a nemterminális szimbólumok tartoznak, és a terminális szimbólumokhoz tartozó állapotátmenetek az *action* táblázatba kerülnek.
- Ekkor az action tábla a következőket tartalmazza:
  - *Léptetés* (s), és adjuk meg mellette a köv. állapot sorszámát.
  - *Redukció* (r), és a mellette levő szám jelentse a nyelvtan helyettesí-tési szabályának sorszámát.
  - Jelöljük *accept*-tel a nulladik szabály szerinti redukciót.
- A *goto* táblába elég egyszerűen csak a köv. állapot sorszámát beírni

- Az *action* táblázat **i**-ik sorát a következőképpen kell kitölteni:

i. Ha $[A \to \alpha.a\beta] \in \tau_i$ és $read(\tau_i, a) = \tau_j$, akkor az $action[i,a] = sj$.

ii. Ha $[A \to \alpha.] \in \tau_i, A \neq S'$ és $a \in FOLLOW_1(A)$, akkor az $action[i,a] = rl$, ahol az $A \to \alpha$ az l-ik szabály.

iii. Ha $[S' \to S.] \in \tau_i$, akkor legyen $action[i,\#] = accept$.

- A goto táblázat i-ik sora a következő:

iv. Ha $read(\tau_i, A) = \tau_j$, akkor legyen $goto[i,A] = j$.

- Ezek után töltsük ki az üresen maradt helyeket:

v. Mindkét táblázat összes üresen maradt helyére írjuk be, hogy *error*.

vi. Az elemzés kezdeti állapota legyen az az állapot, amelyik az $[S' \to .S]$ elemet tartalmazza.

- **Def:** Ha egy G nyelvtanra az *action* és a *goto* táblázat minden egyes elemére a fenti eljárással legfeljebb egy értéket adunk meg, akkor a táblázatok kitöltését **konfliktusmentesnek** nevezzük.
- **Def:** Ha egy G kiegészített nyelvtanra az *action* és a *goto* táblázatok kitöltése konfliktusmentes, akkor a nyelvtant **SLR(1) nyelvtannak** nevezzük.
- Az *action* és *goto* táblázatok a következők:

állapot	action					goto	
	i	+	(	)	#	E	T
0	s3		s4			1	2
1		s5			acc		
2		r1		r1	r1		
3		r3		r3	r3		
4	s3		s4			6	2
5	s3		s4				7
6		s5		s8			
7		r2		r2	r2		
8		r4		r4	r4		

- Az SLR(1) elemzés egy (α, x) kettőssel írható le, ahol α a verem tartalmát, **x** pedig a még nem elemzett input szöveget jelöli. A verem szimbólumpárokat tartalmaz., első eleme a nyelvtan egy szimbóluma, a második pedig az automata egy állapotának sorszáma. Az elemzés elején a kettős értéke legyen $(\#0, x\#)$, ahol **0** az automata kezdőállapota és **x** az elemezendő szöveg. SLR(1) elemzés menete: Tegyük fel, hogy az elemző pillanatnyi állapota a $(\#0X_1s_1x_2s_2\ldots x_ks_k, a_ia_{i+1}\ldots a_n\#)$ kettőssel írható le. Ekkor az elemző következő lépését az *action* és *goto* táblázatok alapján az s_k és a_i adatok határozzák meg.
  - Ha $action[s_k, a_i] = sj$, akkor az elemző új állapota $(\#0X_1s_1x_2s_2\ldots x_ks_k, a_ija_{i+1}\ldots a_n\#)$.
  - Ha $action[s_k, a_i] = rl$, az l-ik szabály $A \to \alpha$ és $|\alpha| = m$, akkor az elemző új állapota $(\#0X_1s_1x_2s_2\ldots x_{k-m}s_{k-m}\, As, a_ia_{i+1}\ldots a_n\#)$, ahol $s = goto[s_{k-m}, A]$.
  - Ha $action[s_k, a_i] = accept$, akkor az elemzés befejeződik, az elemző az elemzett szöveget elfogadja.
  - Ha $action[s_k, a_i] = error$, akkor az elemzés befejeződik, az elemző az elemzett szövegben szintaktikus hibát detektált.

A kanonikus LR(1) elemző

- **Def:** Ha egy G nyelvtan helyettesítési szabálya $A \to \alpha\beta$, akkor a nyelvtan LR(1)-eleme $[A \to \alpha.\beta, a]$ $(a \in T \cup \{\#\})$, ahol az $\alpha.\beta$ az LR(1)-elem magja, és az a az LR(1)-elem előreolvasási szimbóluma.
- **Def:** Egy G nyelvtan $[A \to \alpha.\beta, a]$ LR(1)-eleme érvényes a $\gamma\alpha$ járható prefixre nézve, ha $S` \to^* \gamma Ax \to \gamma\alpha\beta x$ és az a az x első szimbóluma, vagy ha $x=\varepsilon$, akkor $a=\#$.
- **Def:** Legyen az ζ halmaz egy nyelvtan egy LR(1)-elemhalmaza. Ekkor a closure(ζ) halmaz tartalmazza a következő LR(1)-elemeket:
  - Az ζ halmaz minden eleme legyen eleme a closure(ζ) halmaznak is,
  - Ha $[A \to \alpha.B\beta, a] \in closure(\zeta)$ és $B \to \gamma$ a nyelvtan egy helyettesítési szabálya, akkor legyen $[B \to .\gamma, b] \in closure(\zeta)$ minden $b \in FIRST_1(\beta a)$–ra.
  - A $closure(\zeta)$ halmazt az előző pontban leírt művelettel addig kell bővíteni, ameddig az lehetséges.

- **Def:** Legyen az ζ halmaz egy nyelvtan egy LR(1)-elemhalmaza. Ekkor a $read(\zeta, X)$ halmaz tartalmazza a következő LR(1)-elemeket:
  - Ha $[A \to \alpha.X\beta, a] \in \zeta$, akkor a $closure([A \to \alpha X.\beta, a])$ minden eleme legyen a $read(\zeta, X)$ halmaz eleme.
  - A $read(\zeta, X)$ halmazt az előző művelettel addig kell bővíteni ameddig az lehetséges.

Az LR(1)-elemek felsorolásának rövidítésére vezessük be a következő jelölést:

$[A \to \alpha.X\beta, a/b]$ jelentse az $[A \to \alpha.X\beta, a]$ és $[A \to \alpha.X\beta, b]$ LR(1)-elemeket.

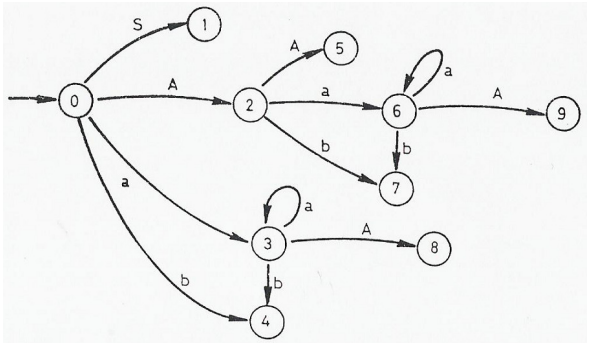
- **Példa:** A $G=(\{a,b\},\{S`,S,A\},A`,P)$ nyelvtan LR(1)-elemeinek kanonikus halmazai a következők:
  - $\zeta_0 = closure([S' \to .S, \#]) = \{[S' \to .S, \#], [S \to .AA, \#], [A \to .aA, a/b], (A \to .b, a/b]\}$.
  - $\zeta_1 = read(\zeta_0, S) = closure([S' \to S., \#]) = \{[S` \to S., \#]\}$.
  - $\zeta_2 = read(\zeta_0, A) = closure([S \to A.A, \#]) = \{(S \to A.A, \#], (A \to .aA, \#], [A \to .b, \#]\}$.
  - $\zeta_3 = read(\zeta_0, a) = closure((A \to a.A, alb]) = \{[A \to a.A, alb], (A \to .aA, a/b], (A \to .b, a/b]\}$.
  - $\zeta_4 = read(\zeta_0, b) = closure([A \to b., a/b]) = \{(A \to b., a/b]\}$.
  - $\zeta_5 = read(\zeta_2, A) = closure([S \to AA., \#]) = \{(S \to AA., \#]\}$.

- ζ_6=read(ζ_2,a)=closure([A → a.A, #])={[A → a.A,#], [A → .aA, #], [A → .b, #]}.
- ζ_7=read(ζ_2,b)=closure([A →b.,#])={[A → b.,#]}.
- ζ_8=read(ζ_3,A)= closure([A→aA.,a/b])={[A → aA., a/b]}.

  read(ζ_3,a) = ζ_3
  read(ζ_3,b) = ζ_4
- ζ_9=read(ζ_6,A)=closure([A → aA.,#])={[A→aA.,#]}.

  read(ζ_6,a) = ζ_6
  read(ζ_6,b) = ζ_7

Az elemző automatája a következő ábrán látható.



- Szabályok, melyekkel az *action* és *goto* táblázatának elemeit lehet meghatározni:
  Az *action* táblázat i-ik sora:

i. Ha [A→α.aβ,b]∈ζ_i és read(ζ_i,a)= ζ_j , akkor az action[i, a]=sj.

ii. Ha [A→α.,a]∈ζ_i és A≠S` akkor az action[i,a]=rl, ahol az A→α az l-ik szabály.

iii. Ha [S`→S.,#]∈ζ_i, akkor legyen action[i,#]=accept.

- **Def:** Az LR(1)-elemek kanonikus halmazaiból létrehozott *action* és *goto* táblázatokat **kanonikus elemző táblázatoknak** nevezzük.

Az LALR(1) elemző

- Cél: az állapotok számának csökkentése.
- **Példa:** A ζ_3 és ζ_6 . a ζ_4 és ζ_7 a ζ_8 és ζ_9 -et lehet egyesíteni. A read függvény definíciója szerint a **read(ζ,X)** csak az ζ LR(1)-elemeinek magjától függ. Ha $\zeta = \zeta_i \cup \zeta_j \cup \ldots \cup \zeta_k$ akkor az $\zeta_i, \zeta_j, \zeta_k$ halmazokban az LR(1)-elemek magjai azonos halmazt alkotnak. Ezért a **read(ζ_i,X)** , **read(ζ_j,X)** ,…, **read(ζ_k,X)** halmazok LR(1)-elemeinek magjaiból alkotott halmazok is azonosak.
  Ha az egyik kanonikus halmazban egy **[A→α.aβ,b]**, a másikban egy **[A→α.,a]** elem szerepelne, akkor az egyesítés után *léptetés/redukálás konfliktus* lépne fel.
- **Példa:** Tekintsük a **G=({a,b,c,d,e},{S`,S,A,B},S`,P)** nyelvtant, ahol a helyettesítési szabályok:
  - **S`→S**
  - **S→aAd | bBd | aBe | bAe**
  - **A→c**
  - **B→c**

- Az **ac** járható prefixre az **{ [A→c.,d], [B→c.,e] }**, valamint a **bc** járható prefixre az **{ [A→c.,e], [B→c.,d] }**, LR(1)-elemek egy-egy kanonikus halmazt alkotnak.
- A két halmaz egyesítése után redukálás/redukálás konfliktus adódik, ha az input szimbólum d vagy e, a c mondatnyél azonosítható, de nem dönthető el, hogy az **A→c,** vagy a **B→c** redukciót kell végrehajtani.
- **Def:** Ha a **G** kiegészített nyelvtanra a **LALR(1)** elemző táblázatok kitöltése konfliktusmentes, akkor a nyelvtant ***LALR(1) nyelvtannak*** nevezzük.

A Szimbólumtábla

- A szimbólumtábla legegyszerűbben egy olyan táblázatnak tekinthető, melyben egy sor egy szimbólum programbeli jellemzőit, attribútumait tartalmazza.
- Leggyakrabban tárolt attribútumok:
  - Szimbólum neve
  - Szimbólum definíciójának adatai
  - Szimbólum típusdescriptora
  - Szimbólum tárgyprogrambeli címe
  - Annak a forrásnyelvi sornak a sorszáma, amelyben a szimbólumot definiálták
  - Azoknak a forrásnyelvi soroknak a sorszámai, amelyekben a szimbólumra hivatkoztak
  - A szimbólum ábécé-sorrendbeli láncolási címe
- A táblának mindig kell tartalmaznia a szimbólum nevét, mert a szimbólumot a nevével azonosítja.
- Problémát csak a változó hosszúságú nevek okozhatnak.

- A hosszabb szimbólumnevek kezelésére két módszer alkalmazható:
  - Az első módszernél a szimbólum nevének a hossza tetszőleges lehet, de két név csak akkor tekinthető különbözőnek, ha az első adott darabszámú karaktere különbözik. Ez annyit jelent, hogy jobbról szóköz karakterek-kel kiegészítve a szimbólumok első adott darabszámú karakterét tárolják a táblában.
  - A másik módszernél a változó hosszúságú szimbólumnevek egy „szimbólumnév-string”-be kerülnek. A táblába csak a szimbólumnév stringbeli kezdőcímét és a hosszát tartalmazza. Ez jelentősen csökkentheti a tábla méretét.
- A tárgyprogrambeli cím a kódgeneráláshoz szükséges információ. Ez a cím azt a memóriahelyet jelöli ki, ahová az adott nevű szimbólumot a compiler elhelyezi. Ez a cím akkor kerül be a táblába amikor a szimbólumot definiálják.

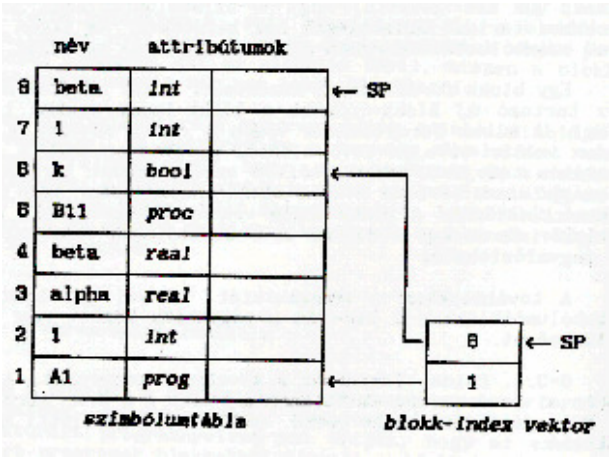
- Műveletek a szimbólumtáblában:
  - **Beszúrás:** Beszúrást kell végrehajtani ha a fordítóprogram egy szimbólum deklarációját dolgozza fel.
  - **Keresés:** Minden további hivatkozás egy keresés végrehajtását jelenti.
- A blokkstruktúrájú műveleteknél további két művelet szükséges:
  - **Set**: Egy blokk elejének feldolgozásakor a set egy új blokkhoz tartozó altáblát nyit meg.
  - **Reset**: A blokk feldolgozásának végén a reset ezt az altáblát törli

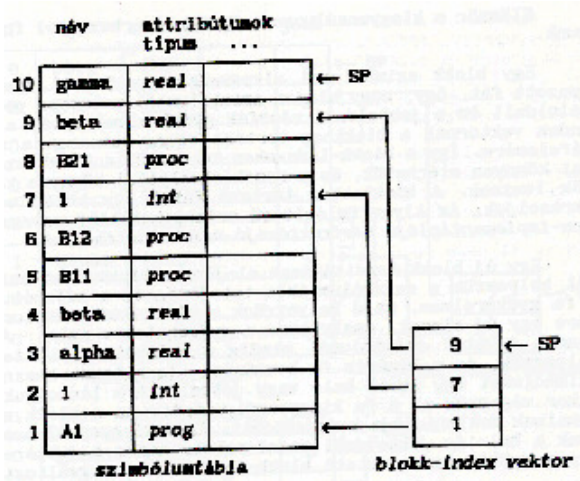
A verem-szimbólumtábla

- Blokkstruktúrált programnyelven azt értjük, hogy a programok blokkokból állnak, a blokkok belsejükben blokkokat tartalmazhatnak. Így beszélhetünk egy változó deklarációjának *hatásköréről, láthatóságáról, élettartamáról.*
- Biztosítani kell, hogy az egy blokkban deklarált szimbólumok össze legyenek kapcsolva, azért hogy a törlés ne okozzon problémát. Továbbá biztosítani kell, hogy egy szimbólum újradefiniálása esetén a szimbólumra történő hivatkozás a belső blokkban definiált szimbólumot azonosítsa.
- Az egy blokkhoz tartozó szimbólumok összekapcsolás úgy valósítható meg, hogy bevezetünk egy másik vermet, a változó hosszú *blokk-index* vektort. Ennek elemei a szimbólumtáblára mutató pointerek. Az ilyen felépítésű szimbólumtáblát **`verem-szimbólumtáblának`** nevezzük.

- **Példa:**

```
program A1;
  var i: integer;
  alpha, beta: real;
procedure B11;
  var k: boolean;
  i,beta: integer;
end;
procedure B12; var i: boolean;
procedure B21;
  var beta, gamma: real;
end;
end
end.
```

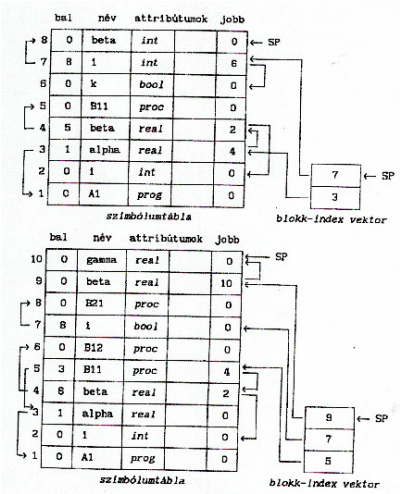




Kiegyensúlyozott fa

- Egy blokk szimbólumai alkossanak egy önálló kiegyensúlyozott fát, úgy, hogy minden szimbólumból mutasson pointer a baloldali és a jobboldali részfák gyökérelemére, és a blokk-index vektornak a blokkhoz tartozó eleme mutasson a fa gyö-kérelemére. így a blokk-indexeken keresztül a blokk szimbólumai könnyen elérhetők, és a reset művelettel könnyen törölhetők lesznek. A blokkokhoz tartozó fákat egy közös veremben ábrázoljuk. Az ilyen felépítésű szimbólumtáblát nevezzük **verem-implementációjú fastruktúrájú szimbólumtáblának.**
- Egy új blokk fordításának elején, a blokk első szimbólumát helyezzük a szimbólumtábla tetejére, ez a szimbólum lesz a fa gyökéreleme, majd helyezzünk a blokk-index vektor tete-jére egy új elemet, amely erre a gyökérelemre mutat. A blokk minden további szimbólumát mindig a szimbólumtábla tetejére helyezzük, és a bináris fa beszúrási algoritmust használva a szimbólumot egy levél bal- vagy jobboldalára láncoljuk. Csak akkor. végezzük el a fa kiegyensúlyozását, ha a blokk szimbólumainak deklarációja befejeződött.

- Egy szimbólum keresésénél, a láthatóság szabályát figyelembe véve, mindig a legutoljára beírt blokk szimbólumait. kell először vizsgálni. Ez azt jelenti, hogy a blokk-index vektor tetején kijelölt fában kell a keresést kezdeni. A keresést ezután a blokk-index vektorban lefelé haladva, a blokk-indexek által meghatározott fákban kell folytatni, és a keresés mindig a szimbólum első előfordulásáig tart.
- A blokk végén a szimbólumok törlése a verem-szimbólumtáblánál leírtakkal azonos.
- **Példa:** A következő ábrákon a verem-implementációjú fastruktúrájú szimbólumtáblának a B11 és a B21 procedúrát befejező *end* fordítása előtti állapota látható.

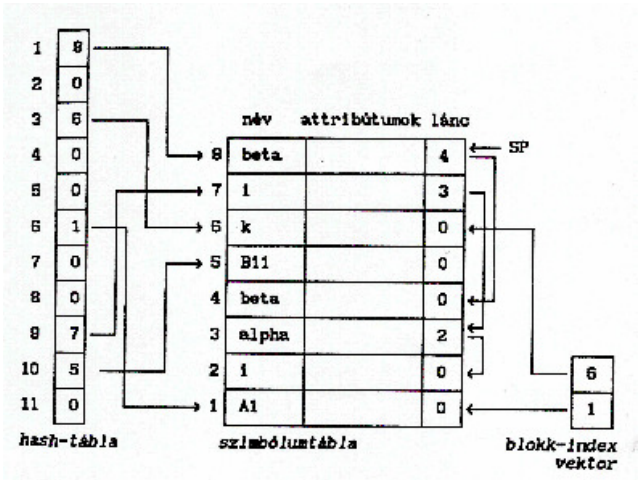


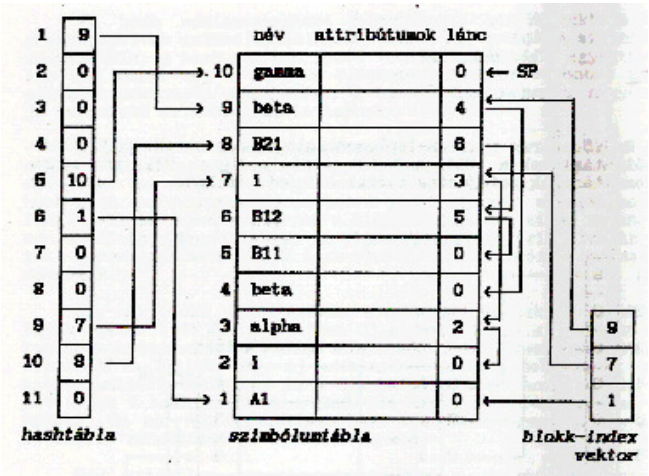
- Ha hash adatszerkezetet alkalmazunk blokkstruktúrált programnyelvek szimbólumtáblájára, akkor egy probléma azonnal szembetűnik: a hash adatszerkezet "rendezetlensége" egyáltalán nem felel meg a program blokkszerkezetének. Azonban az altáblák módszerét alkalmazva, a hashmódszerrel egy nagyon jó hatásfokú szimbólumtáblát kapunk.
- Ha egy blokk szimbólumait egy altáblában helyezzük el, akkor a blokk szimbólumai könnyen elérhetők, és könnyen törölhetők. Az altáblákat egy veremben építjük fel, az azonos hash-kódú szimbólumok láncolására egy pointert alkalmazhatunk. A blokk-index vektornak a blokkhoz tartozó eleme mutasson az altábla kezdőcímére. Az ilyen felépítésű szimbólumtáblát *verem-implementációjú hash-struktúrájú szimbólumtáblának* nevezzük.
- Egy szimbólum keresése egyszerűen a szimbólum hash-kódjával megcímzett hash-tábla pozíción keresztül történhet, és a láthatóság szabályának biztosítására, a szimbólum első előfordulásáig tart. Egy blokk fordításának végén a blokk szimbólumait törölni kell. Ez a művelet az első két táblatípus reset műveleteinél kissé bonyolultabb.

- **Példa:** A hash-függvény képezze le a szimbólumneveket az [1-11] tartományra, és ez a leképezés legyen a következő:

```
A1          →6
alpha,i     →9
B11,B12,B21 →10
beta        →1
gamma       →5
k           →3
```

Az első ábrán a verem-implementációjú hash-struktúrájú szimbólumtáblának a B11, a következő ábrán pedig a B21 procedúra fordításának befejezése előtti állapot látható.





Postdefinit hivatkozások

- A magasszintű programnyelvek fordítóprogramjai a postdefinit hivatkozásokra könnyen egy hibás, vagy esetleg egy nem várt eredményt produkálnak.
- **Példa:** A következő Pascal nyelvű programban a pointer milyen típusú dinamikus változóra mutat?

```
type i=integer;
Procedure P;
  type T=↑i;
       i=real;
  …
```

A T mutatótípus alaptípusa most valós, de ebben a példában könnyen interpretálható egész alaptípusnak is.

Átlapolás

- A legtriviálisabb eset az amikor egy függvénynek és a függvényértéket reprezentáló változónak azonos a neve. Ha egy f függvénynek nincs paramétere, akkor pl. az f függvényeljárás törzsében az f:=f+1; értékadó utasítás baloldalán az f változó, a jobboldalán pedig az f függvény rekurzív meghívása látható.
- Az ADA programnyelv még további átlapolásra ad lehetőséget: függvényeket, operátorokat, és felsorolási típusok konstansait lehet átlapolni, és mindegyik átlapolás érvényes, azaz a hivatkozások környezete határozza meg, hogy melyik értelmezést kell aktivizálni.
- Minden átlapolást a szimbólumtáblába külön elemként fel kell írni, hiszen a szimbólumok attribútumai biztosan különböznek. Az átlapolásokat azonban össze kell láncolni, ezzel biztosítható, hogy kereséskor az összes átlapolás elérhető legyen. Kereséskor a láncban addig kell keresni, amíg az adott programkörnyezetnek megfelelő átlapolást nem találunk. A lánc egyes elemeinek törlését is biztosítani kell.

A szintaktikus elemzés és a szimbólumkezelés

- A szimbólumtábla fastruktúrájú, a szimbólumok típusát típusdescriptorok írják le. A szimbólumok a fában SYMTAB_NODE típusú pontok:

```
typedef struct symtab_node {
struct symtab_node *left, *right;
struct symtab_node *next;
char *name;
DEFN_STRUCT  defn;
TYPE_STRUCT_PTR typep;
int level;
char name_index;
} SYMTAB_NODE, *SYHTAB_NODE_PTR;
```

- A **left** és ***right*** változók a faszerkezethez kellenek, a **next** változót például a felsorolási típus típusértékeinek láncolására lehet használni. A **name** a szimbólum nevére mutató pointer, a **defn** pedig a szimbólum definíciójából meghatározható adatokat tartalmazza.
- A *DEFN_STRUCT* típus a következő:

```
typedef union{
Int integer;
Float real;
Char character;
Char *stringp;
}VALUE;
typedef enum {
UNDEFINED,
CONST _DEFN, TYPE_DEFN, VAR_DEFN,FIELD_DEFN, VALPARM_DEFN,
   VARPARM_DEFN, PROG_DEFN, PROC_DEFN,FUNC_DEFN, } DEFNKEY;
```

```
typedef enum {
DECLARED, FORWARD,
READ, READLN, WRITE, WRITELN, ABS, ARCTAN, ... , SQR, SQRT, SUCC,
   TRUNC} ROUTINE_KEY;
typedef struct { DEFN_KEY key;
union {
struct {
VALUE value; } constant;
 struct {
ROUTINE_KEY key;
int parm_count;
Int total_parm_size;
Int total_local_size;
 struct symtab_node *parms;
struct symtab_node *locals;
struct symtab_node *local_symtab;
} routlne;
struct {
Int offset;
struct symtab_node *record_idp;
   } data;
   } info;
}DEFN_STRUCT;
```

- A **typep** a típusdescriptorra mutató pointer, a **level** a szimbólum szintszáma, a **name_index** pedig a szimbólumnak az assembly nyelvű kódgenerálásához szükséges azonosítója.
- A típusdescriptor a következőképpen adható meg:

```
typedef enum {
NO_FORM,
SCALAR_FORM, ENUM_FORM, SUBRANGE_FORM, ARRAY_FORM,
   RECORD_FORM,
} TYPE_FORH;
typedef struct type_struct {
TYPE_FORM form;
int size;
struct symtab_node *type_idp;
union {
struct {
struct symtab_node *const_idp;
   int max;
} enumeration;
```

```
struct {
struct type_struct *range_typep;
   int min, max;
} subrange;
struct {
Struct type_struct *index_typep, *elmt_typep;
Int min_index, max_index;
   int elmt_count;
} array;
struct {
   struct symtab_node *field_symtab;
   } record;
   } info;
} TYPE_STRUCT, *TYPE_STRUCT_PTR;
```

- **size** a típushoz tartozó bytehosszot tartalmazza, a **type_idp** pedig a típusdefiníció típusnevét tartalmazó pontra mutató pointer.

- Egy program fordításának megkezdésekor a szimbólumtábla nem üres, hanem tartalmazza az *integer, real, char* és *boolean* predefinit típusokra, valamint a *FALSE* és *TRUE* konstansokra vonatkozó bejegyzéseket.

Változódeklarációk

- A fordítóprogram a változóhoz a már deklarált típus típusdescriptorát rendeli hozzá. A szimbólum offset értékének meghatározására egy nulla kezdőértékű változót használ, amelynek értékét a feldolgozás után a deklarált változó típusdescriptorában levő **`size`** értékkel inkrementálja.
- Ha egy deklarációban több azonosító is szerepel, akkor az azonosítókhoz tartozó pontok a next pointeren keresztül láncot alkotnak. A típusdescriptor meghatározása után a fordítóprogram ezt a láncot használja akkor, amikor a típudescriptorra mutató pointert minden egyes pontba beírja.

Program és alprogram deklarációk

- A verem szimbólumtáblában egy alprogram első szimbólumára a **`blokk-index`** vektor egy eleme mutat. Egy program fordításának megkezdésekor a szimbólumtábla nem üres, hiszen a predefinit típusok és predefinit standard alprogramok azonosítóinak bejegyzéseit már tartalmazza, ezért definiáljuk a **`blokk_index[0]`** elemet úgy, hogy az az első predefinit bejegyzésre mutasson.
- A szimbólumtáblában a lefordítandó programból az első bejegyzés a program azonosító neve lesz. erre mutat a *blokk_index[1]* pointer.
- Procedúra vagy függvény deklarációjának fordításakor a **`defn.info.routine`** mezők kitöltése a fő feladat.
- A **`name-index`** mező lehet a szimbólumnak a programozó által megadott azonosító neve.

- Az alprogramok szimbólumai, azaz paraméterei és változói, a rekord mezőszimbólumaihoz hasonlóan egy saját szimbólumtáblába kerülnek. Erre a privát szimbólumtáblára mutat a **`local_symtab`** pointer. A paraméterek darabszámát a **`parm_count`** tartalmazza.
- A paraméterek pontjai a **`next`** pointeren keresztül láncolódnak, az első paraméterre a **`parms`** pointer mutat. Ez a lánc az aktuális paraméterek feldolgozá-sához szükséges.
- Az első lokális változóra a **`locals`** pointer mutat. A paraméterek és a lokális változók által elfoglalt memóriaterület byteszámát a **`total-parm_size`** és a **`total_local_size`** mezők tartalmazzák.
- egy alprogramba történő belépéskor a **`set`** művelet inkrementálja, kilépéskor a **`reset`** dekrementálja a szintszámot, és ez a szintszám a szimbólumtábla blokk-index vektorának indexe is. A szintszámot az alprogram összes szimbólumában a **`level`** mező tárolja.

A Szemantikus elemzés

- A továbbiakban olyan fordítóprogramokkal foglalkozunk, melyekben a szintaxisfa alapján történik a további feldolgozás, a szemantikus elemzés. A szintaxisfa pontjaihoz olyan attribútumokat rendelünk, melyek leírják az adott pont tulajdonságait. Ezek meghatározása a szemantikus elemzés feladata lesz.
- A szemantikus elemzők a következő tulajdonságokat vizsgálják:
  - Változók deklarációja, hatásköre. láthatósága.
  - Változók többszörös deklarációja, a deklaráció hiánya.
  - Operátorok és operandusok közötti kompatibilitás.
  - procedúrák. tömbök formális és aktuális paraméterei közötti kompatibilitás.

Az akciószimbólumok és a fordítási nyelvtanok

- Ha *x* a nyelvtan által definiált nyelvnek egy mondata, akkor a szintaktikus elemzés egy **S→*αβ→*x** levezetést állit elő.
- A helyettesítési szabályokban speciális jelekkel jelölhetjük azt, hogy a szintaxisfa felépítése folyamán az adott szabály alkalmazása esetén szemantikus elemzési tevékenységeket is kell végezni. Ezeket a speciális jeleket hívjuk **akciószimbólumoknak**. Ha a szemantikus tevékenységet egy *prog* procedúra írja le, akkor ezt a procedúrát **szemantikus rutinnak** nevezzük és a hozzátartozó akciószimbólumot **@prog** -gal jelöljük.
- Ha a nyelvtan szabályait a szemantikus elemzés elvégzése céljából akciószimbólumokkal egészítjük ki, akkor a nyelvtant **fordítási nyelvtannak** nevezzük. A továbbiakban a G környezetfüggetlen nyelvtankból készített fordítási nyelvtanokat **TG** -vel Jelöljük.

- Mivel mind a felülről lefelé, mind az alulról felfelé elemzések az elemzéshez egy vermet használnak, ezért a szemantikus rutinok egymás közötti paraméterátadására is egy verem szolgáljon. Ezt a vermet **szemantikus veremnek** hívjuk, a szintaktikus elemző által használt vermet **szintaktikus veremnek** nevezzük.
- Az akcióvezérelt szemantikus veremre az a jellemző, hogy az akciószimbólumokhoz tartozó szemantikus rutinok kezelik, a veremre vonatkozó **push** és **pop** műveleteket ezek a rutinok tartalmazzák.
- Az elemző vezérelt szemantikus verem kezelése a szintaktikus elemzéshez használt verem kezelésével párhuzamos.

Az elemzővezérelt szemantikus verem

1. **Alulról-felfelé elemzés:**
   Alulról felfelé elemezve, ha α a mondatforma nyele, és az $A \rightarrow \alpha$ szerinti redukció következik, akkor az α szimbólumai a szintaktikus veremben vannak. így, a szemantikus vermet a szintaktikus veremmel párhuzamosan kezelve, az α minden szimbólumához tartozik egy bejegyzés a szemantikus veremben is. Ezek az értékek csak akkor használhatók egy szemantikus rutin paramétereiként, azaz csak akkor kerülnek ki a szemantikus veremből. ha az α szimbólumai kikerülnek a szintaktikus veremből, azaz, ha az $A \rightarrow \alpha$ alapján egy redukálást végzünk.

- **Felülről-lefelé elemzés**
  Az elemzés folyamán, ha a fordítási nyelvtannak egy $A \rightarrow \alpha$ szabálya szerinti helyettesitést alkalmazunk, akkor először egy *pop* művelettel az *A* szimbólumot a veremböl kiolvassuk, majd ezután az a szimbólumait *push* műveletekkel a verembe helyezzük. A szintaktikus és a szemantikus vermet párhuzamosan kezelve, az $A \rightarrow \alpha$ szabály alkalmazásakor az A szimbólumhoz tartozó attribútumok a szemantikus veremből kikerülnek.

- Ha a szintaktikus verem tetején álló terminális szimbólum megegyezik az input szimbólumsorozat soronkövetkező szimbólumával, akkor is egy *pop* műveletet kell végrehajtani. A szemantikus veremre ez a *pop* művelet sem hajtható végre, mivel a törlendő szimbólum attribútuma még egy akciószimbólumhoz tartozó szemantikus rutin paramétere lehet.
- Egy $A \rightarrow X_1 \ X_2 \ \ldots \ X_n$ szabály szerinti helyettesítéskor se a szintaktikus veremre, se a szemantikus veremre ne hajtsunk végre *pop* műveletet. Vezessünk be egy EOP kódot, és a szintaktikus verembe az *A* szimbólum helyére ezt írjuk be, a szemantikus veremben pedig egy *push* művelettel rezerváljunk helyet az *A* attribútumainak elhelyezésére.
- Ezután már végrehajthatjuk a helyettesítési szabály jobboldalának megfelelő *push* műveleteket. Az akciószimbólumokat is beírhatjuk a szintaktikus verembe.
- Ha egy akciószimbólum a szintaktikus verem tetejére kerül, akkor az elemző program számára ez azt jelenti, hogy az akciószimbólum kiolvasása után a hozzátartozó szemantikus rutint kell futtatni.

Attribútum fordítási nyelvtanok

- Az előző fejezetben láttuk, hogy a nyelvtan szimbólumaihoz rendelt procedúrákhoz paraméterek tartoztak, és ezek a paraméterek továbbították a szemantikus rutinok közötti információt. Most úgy oldjuk meg a szemantikus rutinok közötti információátadást, hogy a fordítási nyelvtan nyelvtani és akciószimbólumaihoz attribútumokat rendelünk.
- Jelöljük a TG fordítási nyelvtan szimbólumainak halmazát $@\zeta$-vel. Legyen TG egy fordítási nyelvtan. A pedig az attribútumok véges nemüres halmaza. Ekkor minden x szimbólumhoz rendeljük hozzá az A egy $A(x)$ részhalmazát. Ez jelentse azt, hogy az X szimbólum a szintaxisfa egy pontjában előfordul, akkor ehhez a ponthoz az $A(x)$ attribútumok egy példányát rendeljük. Minden $@s$ szimbólumhoz rendeljük hozzá az A egy $A(@s)$ részhalmazát, az $A(@s)$ halmaz az s szemantikus rutin paramétereinek halmaza lesz.

- Az attribútumértékek halmazát **V**-vel jelöljük.
- **R:** Szemantikus szabályok halmaza és minden **p∈P** helyettesítési szabályhoz rendeljük hozzá az **R** egy **R(p)** részhalmazát. Az **R(p)** elemeit **szemantikus függvényeknek** nevezzük.
- Ha **p: X→α** és **q: Y→βxγ** két helyettesítési szabály, akkor lehetnek olyan **A(x)** –beli attribútumok is, amelyek értékének meghatározására sem **R(p)**–ben sem **R(q)**–ban nincs szemantikus függvény.
- **C:** A logikai feltételek halmaza és minden **p∈P** helyettesítési szabályhoz rendeljük hozzá a **C** egy **C(p)** elemét. A **C(p)** egy logikai állítást mond ki a **p** helyettesítési szabályhoz tartozó **A(p)** attribútumokra. Ha az állítás a **p** szabály feldolgozásakor az **A(p)** attribútumokra nem teljesül, akkor a szemantikus elemzőnek hibajelzést kell adnia.

- **Def:** Az ATG=(TG,A,V,R,C)-t **attribútum fordítási nyelvtannak nevezzük**, ahol
  - **TG:** fordítási nyelvtan
  - **A:** Attribútumok véges halmaza
  - **V:** Attribútum értékek halmaza
  - **R:** Szemantikus szabályok halmaza
  - **C:** Logikai feltételek halmaza
- Az **X** szimbólum egy attribútumát **szintetizáltnak nevezzük** ha értékét egy szemantikus függvény abban az esetben határozza meg, amikor az **X** szimbólum egy helyettesítési szabály baloldalán áll.
- Az attribútum **örökölt**, ha értékét egy szemantikus függvény akkor határozza meg, amikor az **X** szimbólum egy helyettesítési szabály jobboldalán áll.

- A terminális szimbólumokhoz az örökölt attribútumokon kívül tartozhatnak úgynevezett **kitüntetett szintetizált attribútumok**. A kitüntetett szintetizált attribútumok érté-két egy konstans értékű függvény határozza meg.
- A továbbiakban jelöljük az **X** szimbólumhoz tartozó **a** attribú-tumot **X.a** -val.
- Jelöljük az **R(p)** szemantikus függvények által meghatározott attribútumok halmazát **AF(p)** -vel, azaz legyen ***AF(p) = {X.a | X.a ← f(...)∈R(p)***
- A **@s** akciószimbólum **@s.a** attribútuma **örökölt**, ha a **@s** -t tartalmazó **P** helyettesítési szabályra ***@s.a ∈ AF(p)***, és a **@s.a** attribútum **szintetizált**, ha értékét az s szemantikus rutin határozza meg.
- Ha az **X (X ∈ T∪N∪@γ)** szimbólum **X.a** attribútumának szintetizált vagy örökölt jellegét is meg akarjuk különböztetni, akkor az attribútumot **X↑a** vagy **X↓a** -val jelöljük.

- **Tétel:** Jelölje egy *ATG* nyelvtan **X (X∈T∪N∪@γ)** szimbólumának szintetizált és örökölt attribútumait **γ(X)** és **τ(X)**. Ekkor ***γ(X)∩ τ(X)=0***.
- **Def:** Egy attribútum fordítási nyelvtant *teljesnek* nevezünk, ha minden
  - ***p: A→α*** **-ra** ***γ(A) ⊆ AF(p),*** **és**
  - ***p: A→α @sβ*** **-ra** ***γ(@s) ⊆ AF(p),***
  - ***p: A→α Xβ*****-ra** ***τ(X) ⊆ AF(p)*** **(X∈T∪N∪@γ)**
  - ***X*** **(X∈T∪N∪@γ)** **szimbólumra** ***A(X)= γ(A)∪ τ(X).***
- **Def:** Egy fordítási attribútum nyelvtant **jóldefiniált attribútum fordítási nyelvtannak** nevezünk, ha minden szintaxisfa minden pontjában minden attribútum értéke egyértelműen kiszámítható.
- Minden jóldefiniált attribútum fordítási nyelvtan teljes, de ez fordítva nem áll fenn.

- **Def:** A $p: x_0 \rightarrow x_1\ x_2 \ldots x_n$ helyettesítési szabályhoz tartozó direkt attribútum függőségek a következők:
  $DP(p) = \{(X_i.a, X_j.b) \mid X_j.b = f(\ldots, X_i.a, \ldots) \in R(p), 0 \le i, j \ge n\}$

- A direkt attribútum függőségek egy adott szintaxisfára egy **függőségi gráfot** generálnak, ahol a gráf pontjai az attribútomok, az irányított élek pedig a fenti kapcsolatot leíró relációk.
- Egy *ATG* -t **lokálisan aciklikusnak** nevezünk akkor, ha minden p∈P helyettesítési szabályra a *DP(p)* gráf körmentes.

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

Attribútum kiértékelő stratégiák

- A **menetvezérelt kiértékelők** az attribútumok értékét szintaxisfa postorder bejárásaival határozzák meg. A bejárások száma szerint megkülönböztethetünk egy- vagy többmenetes stratégiákat.
- Ha a kiértékelő a szintaxisfát minden menetben postorder bejárással járja be, akkor **L–R kiértékelőről** beszélünk.
- Az $A \rightarrow X_1\ X_2 \ldots X_n$ helyettesítési szabályra az L-R kiértékelés a következő eljárással adható meg:

```
Procedure eval(A);
  begin
    for i:=1 to n do
      begin Iatt(Xi); eval(Xi) end;
    Satt(A)
  End;
```

- Ha a postorder bejárást megváltoztatjuk úgy, hogy a leveleket jobbról-balra haladva járjuk be, akkor a kiértékelőt *R-L* **kiértékelőnek** nevezzük.

- Azokat a többmenetes attribútum kiértékelőket, amelyek az attribútumokat páratlan menetekben *L-R*, a páros menetekben *R-L* stratégiával értékelik ki, **alternáló szemantikus kiértékelőknek**, röviden **ASE kiértékelőknek** nevezzük.

- Ha egy *ATG* attribútum fordítási nyelvtan minden szintaxisfája kiértékelhető véges lépésszámú *ASE* kiértékelővel, akkor azt mondjuk, hogy az *ATG az* **ASE nyelvosztályba** tartozik.

Rendezett attribútum fordítási nyelvtanok.

- A rendezett attribútum nyelvtanok elve az, hogy a nyelv mondatain értelmezett, nem feltétlenül véges attribútum függőségeket a helyettesítési szabályokhoz és a szimbólumokhoz kapcsolódó, az attribútumok közötti véges darabszámú függőségre képezzük le.
- A DS(X) határozza meg az attribútumok kiszámítási sorrendjét.
- Relációk DS(X)-ben:
  $p: X \to \alpha$ és $q: B \to \beta X \gamma$ szabályok esetén az X szimbólum attribútumainak meghatározásához mind az R(p), mind az R(q) szükséges, hiszen $\delta(X) \subseteq AF(p)$ és $\gamma(X) \subseteq AF(q)$

- Az A(X) egy $A_1(X), A_2(X), \ldots, A_{m(x)}(X)$ particionálását **megengedett particionálásnak** nevezzük, ha
  - $A_{m(x)}(X), A_{m(x)-2}(X), A_{m(x)-4}(X), \ldots \subseteq \delta(X)$,
  - $A_{m(x)-1}(X), A_{m(x)-3}(X), A_{m(x)-5}(X), \ldots \subseteq \gamma(X)$.
- **Def:** Egy ATG-t particionáltnak nevezünk, ha lokálisan acikli-kus, és minden X szimbólumhoz létezik egy olyan $A_1(X), A_2(X), \ldots, A_{m(x)}(X)$ megengedett particionálás, hogy az X attribútu-mai a partíciók sorrendjében meghatározhatók.
- **Def:** Tekintsük az $NDP(p) = DP^+(p) \setminus \{X.a, X.b) \mid X.a, X.b \in AF(p)\}$ halmazt. Az NDP(p) relációkat normalizált direkt függőségeknek nevezzük.
- **Def:** Egy ATG indukált attribútum függőségeit a következő-képpen határozzuk meg:
  - Minden $p \in P$ -re $IDP(p) = NDP(p)$
  - Minden x-re $IDS(X) = \{(X.a, X.b \mid \exists\ q,\ (X.a, X.b) \in IDP^+(q))\}$,
  - Minden $p: X_0 \to X_1 X_2 \ldots X_n$-re $IDP(p) = IDP(p) \cup IDS(X_n) \cup IDS(X_1) \cup \ldots \cup IDS(X_n)$
  - A 2. és 3. lépést addig kell ismételgetni amíg az IDP és IDS halmazok változnak.

- **Tétel:** Ha egy ATG partícionált, akkor minden p-re és X-re az IDP(p) és IDS(X) gráfja nem tartalmaz kört. Ha $(X.a, X.b) \in IDS(X)$, akkor $X.a \in A(X_i)$ és $X.b \in A(X_j)$, ahol $i \le j$
- **Def:** Legyen $A_1(X), A_2(X), \ldots, A_{m(X)}(X)$ az A(X) egy megengedett partícionálása és legyen $p: X_0 \to X_1 X_2 \ldots X_n$. Az $EDP(p)$ kiterjesztett attribútum függőségek legyenek a következők:

  $EDP(p) = IDP(p) \cup \{(X_i.a, X_i.b) \mid X_i.a \in A_j(X_i), X_i.b \in A_k(X_i), 0 \le i \le n, j < k\}$
- **Tétel:** Egy ATG nyelvtan akkor és csak akkor partícionált, ha minden $p \in P$-re az EDP(p) gráf körmentes.

- **Def:** Egy ATG nyelvtan rendezett attribútum nyelvtan, ha a következő, az **A** partícionálását végző algoritmussal particionált nyelvtant kapunk:

  Minden X-re a partíciók legyenek: $A_1(X), A_2(X), \ldots, A_{m(X)}(X)$, ahol $A_i(X) = T_{m-i+1}(X) - T_{m-i-1}(X)$ $i = 1, 2, \ldots, m(x)$, ahol $T_{-1}(X) = T_0(X) = 0$, $m(X)$ az a legkisebb olyan n index, amelyre $T_{n-i}(X) \cup T_n(X) = A(X)$ és minden k>0-ra legyen $T_{2k-1}(X) = \{X.a \in \gamma(X) \mid ha\ (X.a, X.b) \in IDS(X)$ akkor $X.b \in Tj(X), j \le 2k-1\}$

  $T_{2k}(X) = \{X.a \in \delta(X) \mid ha\ (X.a, X.b) \in IDS(X)$ akkor $X.b \in Tj(X), j \le 2k\}$

  Látható, hogy minden rendezett attribútum nyelvtan partícionált, ez azonban fordítva nem áll fenn.

S-attribútum fordítási nyelvtanok

- **Def:** Egy ATG-t **S-attribútum fordítási nyelvtannak nevezzük** és S-ATG-vel jelöljük, ha minden $A \rightarrow X_1\ X_2 \ldots X_n$ helyettesítési szabályhoz csak $A^{\uparrow}a=f(X_1.b, \ldots, X_n.c)$ alakú szemantikus függvény tartozik, és akció-szimbólum a helyettesítési szabály jobboldalának első szimbóluma.
- Az S-ATG nyelvtanok jól használhatók alulról-felfelé elemzéseknél. Az elemző program a szimbólumhoz tartozó attribútumokat az elemző vermében tárolhatja. Ha az elemző egy $A \rightarrow \alpha$ szabály szerinti redukciót hajt végre akkor az A-hoz tartozó szintetizált attribútum értékét a verem tetején lévő, az α -hoz tartozó attribútum értékéből meghatározhatja.
- Az *S-ATG* nyelvtank az attribútum nyelvtanoknak csak nagyon szűk osztályát alkotják, hiszen örökölt attribútumokat nem tartalmazhatnak.

L-attribútum fordítási nyelvtanok

- **Def:** Egy ATG -t **L-attribútum fordítási nyelvtannak nevezünk** és L-ATG -vel jelölünk, ha minden $A \rightarrow X_1\ X_2 \ldots X_n$ helyettesítési szabályra az attribútumok kiszámítási sorrendje a következő:
  $\delta(A), \delta(X_1), \gamma(X_1), \delta(X_2), \ldots, \gamma(X_n), \gamma(A)$.
- Egy *L-ATG* nyelvtanban tehát a helyettesítési szabály jobboldalán álló X_i szimbólum örökölt attribútumai csak a bal-oldal örökölt és az $X_1\ X_2 \ldots X_{i-1}$ szimbólumok örökölt vagy szintetizált attribútumaitól függnek, és az X_i szimbólum szintetizált attribútuma még a saját örökölt attribútumának is lehet függvénye.
- **Tétel:** Egy *ATG* akkor és csak akkor *L-ATG,* ha lokálisan aciklikus, és minden p: $A \rightarrow X_1\ X_2 \ldots X_n$-re minden $(X_i.a, X_j.b) \in DP(p)$ direkt függőségi relációra a következő esetek egyike fennáll: $i<j$, $i=j$ és $X_i.a \in \delta(X_i)$, $X_i=A$ és $X_i.a \in \delta(A)$, $X_j=A$

- **Def:** Egy *L-ATG* nyelvtan *egyszerű értékadó formában* van megadva, ha minden $A \rightarrow X_1\ X_2 \ldots X_n$ helyettesítési szabályra
  - az $X_i^{\downarrow}a$ $(1 \le i \le n)$ attribútum értéke konstans, vagy megegyezik az $\delta(A)$, vagy az $\gamma(X_j)$ $(1 \le j < i)$ egy értékével, továbbá
  - az $A^{\uparrow}b$ attribútum értéke konstans, vagy megegyezik az $\delta(A)$, vagy az $\gamma(X_i)$ $(1 \le i \le n)$ egy értékével.
- Az értékadó utasítások elhagyhatók, ha az $X_0 \rightarrow X_1\ X_2 \ldots X_n$ ($X_0 \in N$) helyettesítési szabályba beírt attribútumok értékének meghatározásakor a következő szabályokat alkalmazzuk:
  - az $X_i^{\downarrow}a$ $(1 \le i \le n)$ értéke az $X_j(0 \le j < i)$ szimbólumokhoz tartozó legjobboldalibb azonos nevű örökölt vagy szintetizált attribútum értéke lesz,
  - az $X_0^{\downarrow}a$ az elemzés egy másik lépésében kap értéket, akkor, amikor a szintaxisfa egy egyszintű részfájának X_0 leveléhez tartozó örökölt attribútumokat határozzuk meg.

- ha X_i egy akciószimbólum, akkor az $X_i^{\uparrow}a$ $(1 \le i \le n)$ értékét szemantikus rutin határozza meg, egyébként az $X_i^{\uparrow}a$ $(1 \le i \le n)$ az elemzés egy későbbi lépésében kap értéket, akkor. amikor a szintaxisfa X_i gyökérelemű egy szintű részfájának szintetizált attribútumértékét határozzuk meg.
- az $X_0^{\uparrow}a$ értéke az X_j $(1 \le j \le n)$ szimbólumokhoz tartozó legjobboldalibb azonos nevű attribútum értéke lesz.

Hibakezelés

Hibák, szimptómák, anomáliák.

- A fordítóprogram szempontjából meg kell különböztetnünk a **hibát** és a hiba **szimptómáját**. A hibakezelőnek a hiba szimptómáiból kell diagnosztizálni a program tényleges hibáját. A diagnózis azonban mindig magában hordoz egy bizonytalanságot, ezért a legtöbb compiler nem is végez többet, mint azt, hogy hibajelzés formájában közli a felfedezett szimptómát és a programozóra bízza a hiba pontos diagnosztizálását.
- Egy hibát **detektálható hibának** nevezünk, ha legalább egy olyan szimptómát okoz, amelyik megsérti a programnyelv definícióját.
- A fordítóprogramnak a program összes detektálható hibáját fel kell ismernie, és ezt a lehető legkorábban meg kell tennie. Nem szabad viszont a szimptóma jelentkezése előtt hibajelzést adni.

- Az **`anomáliák`** olyan rendellenességek, amelyeknek az oka nem feltétlenül programhiba. Az anomáliákat nem lehet a programnyelvből levezetni, a fordítóprogram írójának programozói gyakorlatában összegyűjtött tudás mutatkozik meg felismerésükben. Gyakran egy detektálható hiba először mint anomália jelentkezik, és csak utána jelenik meg a hiba szimptómája
- Egy szimptóma megjelenése esetén a fordítóprogram három tevékenységet végezhet:
  - hibajelzést ad, azaz jelzi, hogy a forrásnyelvi programban hiba van,
  - hibaelfedést végez, azaz a fordítás folytatása érdekében szinkronizálja a fordítóprogramot és a forrásnyelvi programot alkotó szimbólumsorozatot,
  - hibajavítást végez, azaz a szimptóma alapján diagnosztizál, majd a feltételezett hibát a forrásnyelvi programot megváltoztatva megszűnteti.

Hibajelzés

- A fordítás folyamán jelentkező szimptómákat a *source*-handler program hibajelző modulja kezeli, a modulnak a szimptóma jelentkezésének helyét, súlyosságának szintszámát, és a szimptóma kódját kell átadni. A hibajelző modul ebből az információból olyan listasort állít elő, amelyben pontosan meg van jelölve a szimptóma helye és az utóbbi két adatból a hiba szöveges leírása. A szimptóma helyeként mindig az eredeti forrásnyelvű program egy pontját kell megadni.
- Az egyes hibakódokhoz tartozó szövegeket egy táblázat tartalmazhatja.
- A szimptómák súlyosságát egy szintszámmal jelöljük, ezek a következők lehetnek:

  1.szint: észrevétel 2.szint:megjegyzés 3.szint:figyelmeztetés 4.szint:hiba 5.szint:súlyos hiba 6.szint:túlterhelés

- Az 1-3. szint az anomáliákhoz kapcsolódik. Az 1. szint a nem szabványos konstrukciót jelzi. A 2. szint a programozói stílust kifogásolja. A 3. szint a lehetséges hibákra utal. A 4. szintű hibák a lexikális, szintaktikus és szemantikus hibák, ezek legtöbbször a hibajavító programmal is javíthatók. Az 5. szintű hiba esetén a fordítóprogram már nem készít tárgyprogramot, a 6. szintű hiba jelentkezése, például a szimbólumtábla megtelése esetén, a fordítóprogram kénytelen azonnal befejezni a fordítást.

- A felhasználónak továbbá lehetőséget kell adni arra, hogy az általa megadott szintnél kisebb hibákat a fordítóprogram ne írja ki.

Hibaelfedés

- A *hibaelfedés* célja az, hogy a hiba szimptómájának jelentkezése után a hibát követő szöveg elemzése tovább folytatódjon. Egy rosszul megválasztott hibaelfedéssel az elemezendő szövegbe újabb hibák kerülhetnek. A hibaelfedő programnak az ilyen indukált, újabb hibák számát minimalizálnia kell.
- A legegyszerűbb hibaelfedés a **`pánik módszer`**. A módszer lényege az, hogy az elemző "kiugrik" a hibás programrészből. A grammatikához definiálni kell a **`biztonságos szimbólumok`** halmazát, ez a halmaz általában az utasítás vége szimbólumot, az utasítások első szimbólumait és a program vége szimbólumot tartalmazza.
- Egy jó hibakezelőjű fordítóprogramnak a hibát azonnal fel kell ismernie. Sajnos, az LL(1),SLR(1), LALR(1) elemzők sem rendelkeznek ezzel a tulajdonsággal, a hibát csak akkor ismerik fel, amikor az illegális szimbólumot a verembe akarják tenni.

Ad hoc hibafelfedési módszerek

- Az ad hoc hibaelfedési módszerek speciálisan egy-egy adott elemzőhöz készülnek, az egyes hibajelenségekre a nyelv szabályai alapján külön-külön meghatározva a hibaelfedést megvalósító egyszerű tevékenységet. Ezek a tevékenységek a **hibarutinok**, amelyek a pánik módszert csak ritkán használják.
- Az elemző táblázatok üres helyein helyezzünk el hibarutinokra mutató pointereket. A hibarutinok megváltoztathatják az input szöveget, cserélhetnek, beszúrhatnak, törölhetnek szimbólumokat, és az elemző vermének tartalmát is módosíthatják. Arra azonban ügyelni kell, hogy ezek a változtatások nehogy ciklust okozzanak az elemző működésében, és azt is biztosítani kell, hogy ha az elemző egy hibaelfedési művelettel az input szöveg végére kerül, akkor az elemző verme is üres legyen, és ha a verem kiürül, akkor az elemző olvasásokkal az input szöveg végére kerüljön.

Hibajavítás

- A **hibajavítás** a hibaelfedést is magában foglalja, hiszen a kijavított hibától kezdve az elemzés tovább folytatódik. A hibajavítás csupán a hibás programnak egy érvényes, az adott nyelv mondatává való legkisebb költségű átalakítását jelenti.
- Legyen $xay \notin L(G)$, ahol **x** a már elemzett szöveg, és az a szimbólumnál van a detektált hiba. Ekkor a hibajavításnak három módszere lehetséges:
  - az **x** módosítása úgy, hogy $x'ay \in L(G)$
  - egy **w** beszúrása úgy, hogy $xway \in L(G)$
  - az **a** törlése és az **xy** szöveg elemzése, újabb hiba esetén pedig az 1.-3. lépések ismétlése.
- Az első módszer alkalmazása nem célszerű, hiszen akkor az elemzés egy korábbi állapotához kell visszalépni, és a kapott eredmény egy részét is törölni kell

LL(1) elemzők hibajavítása

- Egy *L(G)* nyelvet *beszúrással javíthatónak* nevezünk akkor,ha $S \to^+ xz$, és $xay \notin L(G)$ esetén mindig van olyan $w \in T^+$, amelyre $xwav \in L(G)$. Az Ada programnyelv például egy ilyen nyelv, mert az **x** után a **package** lezárható, és egy **új package** kezdhető úgy, hogy az **ay** ennek postfixe legyen.
- Az FMQ-algoritmust először csak a beszúrással javítható nyelvekre definiáljuk.
- Egy hibajavítás minőségét a javítás költségével fejezzük ki. Legyen minden a-ra az a beszúrásának költsége $C(a) \geq 0$, és legyen $C(\varepsilon)=0$, $C(\#)=\infty$. Ha $x=a_1a_2\ldots a_n$, akkor legyen $C(x)=C(a_1)+C(a_2)+\ldots+C(a_n)$. Továbbá legyen ? Egy új terminális szimbólum, amelyre $C(?)=\infty$.

- Egy **A** szimbólumra legyen **M(A)** a legkisebb költségű levezetés, azaz $M(A)=x_0$, ha $C(x_0)=\min C(x)$ ahol $A \to^+ x$, és legyen az **A** szimbólumból az a -t tartalmazó levezetések *legkisebb költségű prefixe* **N(A,a)**, azaz

N(A,a)=	w_0	Ha $C(w_0)$=min C(w), ahol $A \to^+$ wau
	?	Ha A not $\to^+$ wau

- Legyen továbbá
  - $M(a)=a$,
  - $N(a,b)=?$, ha $a \neq b$,
  - $N(a,a)=\varepsilon$.

- Az FMQ-algoritmus programja:

```
function FHQ (ParseStack:Stack of Symbol; a:Terminal)
   return Term1nalStr1ng is
   Insert, LeastCost: Term1nalStr1ng;
    begin
        Insert:= "?"; LeastCost:= ε;
        for 1 in O..Depth(ParseStack)-l loop
        exit when C(LeastCost) >= C(Insert);
        if C(LeastCost & N(ParseStack(Top-I) ,a)) <
        C(Insert) then
        Insert:= LeastCost & N(ParseStack(Top-I),a);
        end if;
        LeastCost:= LeastCost & M(ParseStack(Top -1));
        end loop;
        return Insert;
end FHQ;
```

- A minimális költségű hibajavítás algoritmusát **LL1-FMQ-algoritmusnak** nevezzük.

```
procedure LL1FMQ (Wd:out TerminalStr1ng; d:out Integer) is
begin
Wd:= "?"; d:= O;
for 1 in 1..z` Length loop
exit when D(z(1..I-1))>= C(Wd) + D(z(1..d));
 if C(FMQ(ParseStack,z(I))) + D(z(1..I-1)) < C(Wd) +
   D(z(1..d)) then
Wd:= FMQ(ParseStack,z(I);
d:=I-1;
end if;
   end loop;
end LL1FMQ;
```

- Ha a nyelv egy beszúrással nem javítható nyelv, akkor az algoritmus minden lépésében meg kell vizsgálni, hogy az $\mathbf{xw_d b_{d+1} b_{d+2} \ldots b_n \in L(G)}$ teljesül-e. (Ezt a műveletet hívjuk *érvényesítésnek.)* A gyakorlatban használt elemzőkben azonban ennél gyengébb feltétel teljesülését is elegendőnek tekintik, és csak azt vizsgálják, hogy a $\mathbf{b_{d+1} b_{d+2} \ldots b_n}$ első *k* szimbóluma hiba nélkül elemezhető-e. Ekkor **k-érvényesítésről** beszélünk, a szimbólumsorozatot k-érvényesnek, a hibajavítást **regionálisan optimális javításnak** nevezzük.

- Tetszőleges LL(1) nyelvhez alkalmazható regionálisan optimális hibajavítás tehát a következő:

```
procedure kValidLL1FMQ (Wd: out TerminalString;
    d: out Integer; k: Integer) is
 begin
 Wd="?"; d:=0;
  for I in 1..z'Length loop
 exit when D(z(1..I-1)) >= C(Wd) + D(z(1..d));
   len:= min(I+k-1, z'Length);
```

```
if C(kValidFMQ(ParseStack,z(I..len)))
+ D(z(1..I-1)) < C(Wd) + D(z(1..d)) then
Wd:= kValidFMQ(ParseStack,z(I..len));
d:= 1-1;
   end if;
   end loop;
end kValidLL1FHQ;
```

LR(1) elemzők hibajavítása

- Az algoritmus hasonló az FHQ-algoritmushoz, hiszen most is a minimális költségű beszúrást és törlést határozzuk meg, de itt már nem tudjuk kihasználni azt, hogy az elemző verme tartalmazza a még nem elemzett szövegre vonatkozó, nemterminális és terminális. szimbólumok sorozatával "kódolt" információt.
- AZ LR(1) elemzők hibajavítása a *folytathatóságon* alapul. Meg kell határozni az x szöveghez a lehetséges legolcsóbb **w** *folytatást,* azaz egy olyan legkisebb költségű terminális sorozatot, melyre **xw∈*L(G)*,** és a teljes hátralévő **ay** szöveget a **w** -vel kell helyettesíteni.
- Ennél azonban jobb megoldás az, hogy a **w** szimbólumsorozat w_d prefixeit a hiba helyére beszúrva, és csak **d** darab szimbólumot törölve próbálunk optimális költségű $w_d b_{d+1} b_{d+2} \ldots b_n$ sorozatot előállítani.

- Ha az elemző a hiba detektálásakor az s állapotban van, akkor az τ_s LR(0)-kanonikus halmazból kiolvasható, hogy milyen terminális szimbólummal folytatható az elemzés a sikeres befejezés felé.
- **Def:** Legyen az τ halmaz egy grammatika egy rendezett **LR(0)**–kanonikus elemhalmaza. Ekkor a *list-closure(τ)* halmaz tartalmazza a következő elemeket:
  - Az **τ halmaz minden eleme legyen eleme a list-closure(τ)** halmaznak is.
  - Ha **[A→α.Bβ]∈list-closure(τ) és** B→γ | δ esetén **C(M(γ))<C(M(δ)),** akkor legyen **[B→.γ]∈ list-closu-re(τ), [B→.δ]∈list-closure(τ)**
  - A **list-read(τ,X)** halmazt az előző pontban leírt művelettel addig kell bővíteni, ameddig az lehetséges.

- **Def:** Legyen az τ halmaz egy grammatika egy rendezett **LR(0)**–kanonikus elemhalmaza. Ekkor a *list-closure(τ)* halmaz tartalmazza a következő elemeket:
  - Ha **[A→α.Xβ]∈τ** akkor a **list-closure([→αX.β])** minden eleme legyen a **list-read(τ,X)** halmaz eleme, és az elemek sorrendje ne változzon meg.
  - A **list-read(τ,X)** halmazt az előző pontbeli művelettel addig kell bővíteni, ameddig az lehetséges.
- Az elemző automata i állapotához tartozó rendezett kanonikus elemhalmazban legyen I az első **[A→α.cβ]** vagy **[A→α.]** típusú LR(0)-elem. A grammatika redukáltsága biztosítja, hogy legalább az egyik típusú elem biztosan létezik.
  - Ha I= **[A→α.cβ], akkor cont(i)=c.**
  - Ha I= **[S`→S.], akkor cont(i)=accept.**
  - Ha I= **[A→α.], akkor cont(i)=r1, ahol r a redukálás jele és A→α a grammatika I-ik szabálya.**

Az értékadó utasítás fordítása

- Ebben az utasításban a változó értéke és a változó címe a változó jelölésmódjából, a változóra való hivatkozásból nem különböztethető meg.
- A legtöbb magasszintű programnyelvben nincs is lehetőség egy változó címének elérésére, kivéve a címszerinti paraméterátadás, az értékadó utasítás és az input utasítások paramétereinek esetét.
- Csupán a változó helye határozza meg azt, hogy az értékét vagy a címét kell-e figyelembe venni. Az ***X := Y + Z*** értékadó utasításban a **:=** jel baloldalán a változó címére, a jel jobboldalán a változók értékére hivatkozunk.

- Az értékadó utasítás a következőképpen adható meg:

  <értékadó-utasitás> →@Set(↑r)
  <változó>↓r↑t := @Reset(↓r) <kifejezés> ↑s
  @Store(↓r,t,s)

- Az r attribútum annak a jelzésére szolgál, hogy a változók címét vagy értékét kell meghatározni. A ***@Set(↑r)*** és a ***@Reset(↓r)*** szemantikus rutinok ezt az attribútumot állítják.
- A ***@Store(↓r,t,s)*** rutin ellenőrzi a *változó* és a *kifejezés* **t** és **s** típusát, ha kell, generálja a típuskonverziót végző procedúra hívását, és kiadja a kifejezés értékét a változó címére helyező utasítást.

A goto fordítása

- A fordítás nehézségét itt az okozza, hogy a programnyelvek megengedik ugyanannak a címkének a többszörös használatát.
- A Pascal és a C csak egy alprogramon belüli ugrásokat enged meg. A Pascal még az ugráscímek előzetes deklarálását is megköveteli. A címkék csak a deklarációt tartalmazó blokkban definiálhatók és a címkékre csak ebben a blokkban lehet hivatkozni. A címke nem lehet belső blokkban sem, és a címkére belső blokkból sem lehet ugrani. Ilyen megkötések mellet a goto utasítások fordítása rendkívül egyszerű, hiszen a láthatóság és hatáskör követelményei a szimbólumtábla már ismertetett kezelési módszereivel megvalósíthatók.

- Ha a vezérlésátadás egy külső blokkba történik, akkor az ugrás végrehajtása után a címkét tartalmazó blokk aktivációs rekordja lesz az aktuális aktivációs rekord.
- A szimbólumtáblából meghatározható, hogy a címke melyik blokkban van. A goto végrehajtásakor vissza kell lépni ehhez a blokkhoz, azaz a fordításkor annyi alprogramból történő visszalépés utasítássorozatot kell generálni, ahány szinttel lejjebb van az új aktivációs rekord.
- Ha a procedúrahivások rekurzív hívások, akkor fordítási időben csak olyan visszalépés generálható , amelyik a közvetlenül megelőző procedúrahiváshoz tér vissza.

Az alprogramok fordítása

- A programokban kétféle alprogram írható:
  - Procedúra
  - Függvény
- Ez a két dolog a generált kód szempontjából csak annyiban különbözik, hogy függvény esetében az aktivációs rekord egy a másik függvényérték számára fenntartott mezőt is tartalmaz . Mivel a függvényérték egy implicit paraméternek tekinthető ez a mező az aktivációs rekord paraméter területén helyezkedik el. A függvényérték kilépéskor innen olvasható ki.
- Egy alprogram fordítása az alprogram deklarációjának és hívásainak fordításából áll. Míg a deklaráció fordítása hasonló a változók deklarációjának a fordításához, a hívás fordítása a vezérlésátadás és a paraméterátadás miatt sokkal bonyolultabb feladat, mint egy változóra történő hivatkozás fordítása.

Paraméter nélküli alprogramok

- Az alprogramhoz tartozó aktivációs rekord építése az alprogram hívásakor kezdődik, és az alprogramba történő belépés után, az alprogram első utasításainak végrehajtásával fejeződik be.
- Egy procedúra hívása legyen a következő:

  <subpr-call> **↓i** → ***<id>*** **↑n** ***@SearchSubpr(*****↓i,n,** **↑p ,l)** ***@StCall*** **(↓i, n,l)**

  Az **i** attribútum az aktuális szintszámot tartalmazza.
- Az n nevü procedúrát a *@SearchSubpr(*↓**i**,*n,* ↑**p**,*l)* szemantikus rutin megkeresi a szimbólumtáblában. Ha van ilyen nevű procedúra, akkor a láthatóság ellenőrzése után a **p** attribútumba tölti a procedúra szimbólumtáblabeli pontjának **defn** mezőjét, és **l** -be a procedúra szintszámát.

- A *@StCall(↓i,n,l)* szemantikus rutin először generálja a statikus pointer feltöltését végző utasításokat. Ha **`l= i+1`**, akkor a rutin a PUSH BP ; *@StCall(↓i,n,l)* utasítást generálja. Ezután az alprogram hívását végző **`CALL n`** utasítás generálása következik.

Paraméterátadási módszerek

- A paraméterátadási módokat a következőképpen csoportosíthatjuk:
  - Érték, érték-read-only, eredmény,érték-eredmény,
  - Hivatkozás, hivatkozás-read-only,
  - Név, címke, procedúra-név szerinti paraméterátadás.
- Az **`érték szerinti`** paraméterátadás esetén a procedúra hívásakor az aktuális paraméter értéke ebbe a lokális változóba másolódik át. Az **`érték-read-only`** paraméterátadásnál a lokális változót csak olvasni lehet. **`Eredmény szerinti`** paraméterátadásnál a procedúra hívásakor ennek a lokális változónak a kezdőértéke definiálatlan, és a procedúrából történő kilépéskor a lokális változó értéke az aktuális paraméterbe kerül.

- **`Érték-eredmény szerinti`** paraméterátadás esetén a procedúra hívásakor az aktuális paraméter értéke a lokális változóba íródik, a procedúrából való kilépéskor pedig a lokális változó értéke az aktuális paraméterbe kerül.
- **`Eredmény és érték-eredmény`** típusú paraméterátadás esetén az aktuális paraméter csak olyan típusú lehet, amelyik értékadó utasítás baloldalán is megengedett.
- A második csoport paraméterátadására az a jellemző, hogy az aktuális paraméter címe kerül átadásra, ezért ezt a paraméterátadási módot gyakran cím szerinti paraméterátadásnak is nevezik.
- A **`hivatkozás szerinti`** paraméterátadáskor a formális paraméter megváltoztatása egyben az aktuális paraméter megváltozását is jelenti, míg a **`hivatkozás-read-only`** paraméterátadásnál a paraméter címe csak az aktuális paraméter olvasására szolgálhat.

- A harmadik csoportban szereplő paraméterátadási módszerek a FORTRAN és ALGOL-60 nyelvekben jelentek meg, a modern programnyelvek ezeket a módszereket már nem preferálják.
- **`Név szerinti`** paraméterátadás esetén az aktuális paraméter egy tetszőleges kifejezés lehet. amellyel az alprogram törzsében a formális paraméter összes előfordulását helyettesíteni kell. Ez szöveghelyettesítést jelent. az assembly nyelv szóhasználatát alkalmazva, az alprogram a formális paraméterekkel egy makródefiníciónak, az alprogram hívása pedig a makróhívásnak felel meg.
- A **`címke paraméterek`** *goto* utasítások operandusai lehetnek.
- Hasonlóan. a **`procedúra-név`** átadásakor is tárolni kell a híváshoz tartozó aktivációs rekord elmét .

Kódoptimalizálás

- Az optimalizálás célja az, hogy a generált kód minősége javuljon, ami alatt azt értjük, hogy a program végrehajtási ideje és a program mérete csökkenjen. Az optimalizáló eljárásokat a kódgenerálás lépésében vagy a kódgenerálás után alkalmazhatjuk, az optimalizálandó program általában az **`object code`**.
- Az optimalizálással szemben támasztott legfontosabb követelmény a biztonság, amely azt jelenti, hogy az optimalizált programnak ugyanazt az eredményt kell adnia, mint az eredetinek. Az optimalizálás azonban nem jelenti az optimális kódú program meghatározását.
- Az optimalizálás lehet **`gépfüggő`**, amikor például egy jobb regiszterfelhasználással gyorsítjuk a program futását, vagy lehet **`gépfüggetlen`**, amikor olyan stratégiákat alkalmazunk, amelyek függetlenek a kódot végrehajtó környezettől.

A lokális optimalizálás

- A programot *alapblokkokra* bontjuk, és egy alapblokkon belül optimalizálunk. Ezt **`lokális optimalizálásnak`** nevezzük. Az alapblokkokat a következő módszerrel határozzuk meg.
- Jelöljük meg a program első utasítását, és azokat az utasításokat, amelyekre ugró utasítással vezérlést lehet átadni, és jelöljük meg a vezérlésátadó utasítások után következő utasításokat is. Ekkor minden megjelölt utasításhoz egy alapblokk tartozik, az alapblokk a megjelölt utasítást és a következő megjelölt utasításig, vagy a program végéig szereplő utasításokat tartalmazza.
- így minden alapblokknak egy belépési pontja van, és benne az utasítások szekvenciálisan hajtódnak végre.

A tömörítés

- A legegyszerűbb lokális optimalizálási technika a **`tömörítés`**, amelynek célja az, hogy az alapblokk utasításaiban a lehető legkevesebb konstans és konstans értékű változó legyen.
- A tömörítés egyik módszere a **`konstansok összevonása`**. Ha egy kifejezés valamely részének vagy részeinek értéke konstans, és már ismert a fordítás adott lépésében, akkor a compiler a kifejezést módosítja. Elvégezve az ismert értékekre vonatkozó műveleteket, a módosított kifejezésben csak egy konstans szerepel. A kijelölt műveletek el végzésekor természetesen a fordítóprogramnak figyelembe kell venni a műveletek precedenciáját, a kommutativitás és asszociativitás szabályait.

- A tömörítés második módszere a konstans értékű azonosítóknak az értékükkel való helyettesítése. Ha egy azonosító értéke már a fordítási időben ismert, akkor a fordítóprogram ezt az értéket terjeszti tovább, ezért ezt a módszert a **`konstans továbbterjesztésének`** nevezzük.

Az azonos kifejezések kiszámítása

- A lokál is optimalizálás másik széleskörűen alkalmazott módszere az **`azonos kifejezések kiszámításának eliminálása`**. Alkalmazásának feltétele az, hogy a teljes alap blokk már elemezve legyen, és a tömörítések elvégzése már egy korábbi lépésben megtörténjen.
- Egy alapblokk vizsgálatához jó adatszerkezet az irányított körmentes gráf. Az irányított körmentes gráfot az alapblokk absztrakt szintaxis-fájából, a kiszámítási fájából is felépíthetjük.
- A kiszámított fában egy **`b+c`** kifejezésnek a

fa felel meg, ahol b és c a fa levelei, vagy egy-egy részfa gyökérpontjai. .

- A b + c kifejezés irányított körmentes gráfjában a b és c csak akkor lesz levél, ha értéküket eddig még nem határoztuk meg. Ha a b és/vagy c értéke már meghatározott, akkor a + ponthoz tartozó élek a b és vagy c -t meghatározó gráfpontokba vezetnek, és ha már a b + c értéke is adott, akkor egyszerűen egy él vezet a gráf b + c -t már meghatározó részgráfjának gyökérpontjára.
- Az a:= b + c értékadást úgy jelöljük, hogy a gráf pontjához az a címkét írjuk. Ha már volt a gráfban a címkéjű pont, akkor onnan az a címkét törölni kell. A d:=a utasítást külön nem jelöljük, hanem az a címkéjű gráfponthoz a d címkét is odaírjuk.
- Egy alapblokkhoz tartozó irányított körmentes gráfot az alapblokk utasításaiból a következő módszerrel határozzuk meg.

- Legyen a *node(i)* függvény értéke egy pointer, amely a gráf i címkéjű pontjára mutat, vagy **ε**, ha a gráfban nincs ilyen pont. Ez a pointer a szimbólumtáblába is felírható, ekkor a *node(i)* függvény ezt az értéket onnan olvashatja ki.
- Tegyük fel, hogy az alapblokk utasításai

i. **`X : = Y op Z;`**

ii. **`x : = op Y;`**

iii. **`X := Y alakúak.`**

- A gráf legyen üres, a gráf pontjait és éleit az (i)-(iii) típusú utasítások szekvenciális feldolgozásával hozzuk létre. Minden utasításra a következő lépéseket kell végrehajtani:
  - Ha node(y) = **ε**, akkor a gráfot bővítsük egy levéllel, és a levél címkéje legyen y. Az (i) esetben, ha node(z) = **ε**, akkor a gráfot bővítsük egy levéllel, és a levél címkéje legyen z.

- Az (i) esetben vizsgáljuk meg, hogy már létezike a fában *op* gyökerű, y és z bal- és jobbleszármazottú részfa. Ha nincs, akkor bővítsük a gráfot egy ilyen részfával. Az (ii) esetben vizsgáljuk meg, hogy már létezik-e a fában op gyökerű, y leszármazottú részfa. Ha nincs, akkor bővítsük a gráfot egy ilyen részfával. Az (iii) esetben vizsgáljuk meg, hogy már létezik-e a fában y címkéjű pont. Ha nincs, akkor bővítsük a fát egy ilyen ponttal. Mindegyik esetben mutasson p a megtalált vagy létrehozott részfa gyökérelemére.

- Ha node(x)≠ε és az x nem levél, akkor töröljük az x címkét a gráf node(x) által meghatározott pontjáról. Írjuk az x címkét a p pointer által megadott pontra, és legyen node(x)=p

Ciklusutasítások optimalizálása

- Egy ciklusutasítás a ciklusváltozó inicializálását, a lépésközzel való módosítását, a ciklusfeltétel teljesülésének vizsgálatát és a ciklus magját alkotó utasításokat leíró kódokat tartalmazza. A globális optimalizáláshoz tartozik.
- A legegyszerűbb optimalizálás a **`ciklus kifejtése`**. Ha a fordítási időben ismert a ciklusváltozó kezdő- és végértéke, és a lépésköz, akkor a programban a ciklusutasítás helyettesíthető a ciklusmagnak a megadott darabszámú megismétlésével. A ciklusmagokból származó programot a ciklusváltozó megfelelő értékeivel módosítani kell. így az eredetinél biztosan gyorsabban végrehajtható kódot kapunk, azonban a program mérete jelentősen megnőhet.
- A ciklus kifejtése előtt el kell dönteni, hogy a memóriaméretre és a végrehajtási időre vonatkozó követelményeket figyelembevéve, célszerű-e a módszert alkalmazni. Egy adott memóriaméret és futási idő követelmény kielégítésére alkalmazható a **`parciális cikluskifejtés`**.

Ablak-optimalizálás

- Az **`ablak-optimalizálás`** célja a programkód "megtisztítása". Mindig csak egy 1-3 utasításból álló programrészt vizsgálunk, és ezt az "ablakot" végigcsúsztatva a programon, az ablakban látszó utasításokra előre megadott mintákat illesztünk. Ha az utasítások megfelelnek valamelyik mintának, akkor ezeket az utasításokat egyelőre megadott utasítássorozattal helyettesítjük.
- Az ablak-optmalizálás egy halmazzal adható meg. Az ablak-optimalizálást leíró halmaz elemszáma csökkenthető, ha a leírásban formális paramétereket használunk.
- Az ablak-optimalizálás néhány tipikus programegyszerűsítése a következő:
  - felesleges műveletek törlése:
    - nulla hozzáadásának ,
    - nulla kivonásának törlése,
  - műveletek egyszerűsítése:
    - nullával való szorzás helyett értékadás,
    - nulla regiszterbe töltése *(move)* helyett a regiszter tartalmának törlése *(clear),*
  - felesleges utasításcsoportok törlése:
    - regiszterbe töltés és az eredeti helyre visszaírás esetén a visszaírás törlése,
    - felesleges vezérlésátadások törlése, vezérlésátadási láncok törlése,
    - utasításismétlések vizsgálata és ha lehetséges, a felesleges ismétlések törlése.

A globális optimalizálás

- A **globális optimalizálás** az alapblokkokra osztott teljes programot vizsgálja. A program egy irányított gráfnak tekint-hető, ahol a blokkok a gráf pontjai, és a blokkok közötti vezérlésátadás határozza meg a pontok közötti irányított éleket. A program végrehajtása a gráf egy útjának bejárása lesz, ahol az utat a pontok sorozatával adjuk meg.

Az adatáram-analizis

- A **globális adatáram-analízis** azt vizsgálja, hogy a blokksorozatok végrehajtásakor a változók értéke melyik ponton változik meg. Ezt az analízist felhasználva lehet optimalizálni a teljes programot. A globális adatáram-analízis két részből áll. Az első rész az alapblokkokra vonatkozó információt gyűjti össze, meghatározva az alapblokkok belépési és kilépési pontjaiban az értékeket reprezentáló szimbólumhalmazokat , az analízis második része pedig ezeket a halmazokat felhasználva, végigköveti az információ terjedését a teljes programban.
- Problémák:
  - Rendelkezésre álló kifejezések analízise
  - Nagyon foglalt kifejezések analízise
  - Értékadások elérhetőségének analízise
  - Élő változók analízise.

Az adatáram-analízis felhasználása az optimalizálásban

- A globális optimalizálásban
  - az azonos kifejezések kiszámításának eliminálása,
  - a változók továbbterjesztése,
  - a ciklus-invariánsok detektálása és a ciklus elé helyezése
  - az indukciós változók eliminálása

  a leggyakrabban alkalmazott optimalizációs módszer.