

Fordítóprogramok célja és szerkezete

Fordítóprogramok előadás (A,C,T szakirány)

1 Fordítóprogramok előadás (A,C,T szakirány) Fordítóprogramok célja és szerkezete

Fordítóprogramról általában Fordítóprogramok célja

Miért van szükség fordítóprogramokra?

Így kényelmes programozni

```
int sum = 0;
for( int i=0; i<len; ++i )
    sum += t[i];
```

2 Fordítóprogramok előadás (A,C,T szakirány) Fordítóprogramok célja és szerkezete

Fordítóprogramról általában Fordítóprogramok célja

Miért van szükség fordítóprogramokra?

Így kényelmes programozni

```
int sum = 0;
for( int i=0; i<len; ++i )
    sum += t[i];
```

Ezt tudja végrehajtani a számítógép

```
B9 00 00 00 00
B8 00 00 00 00
81 F9 0A 00 00 00
7D 06
03 04 8B
41
EB F2
```

2 Fordítóprogramok előadás (A,C,T szakirány) Fordítóprogramok célja és szerkezete

Fordítóprogramról általában Fordítóprogramok célja

Miért van szükség fordítóprogramokra?

- magasszintű programozási nyelvek
 - könnyebb programozni
 - közlebb a megoldandó problémához
 - platform-függetlenség
- gépi kód
 - gépközeli (numerikus utasítások, regiszterek, memóriahivatkozások ...)
 - erősen platform-függő
 - optimizált
- Fordítóprogramokat általában a kettő közötti átalakításra használunk.

3 Fordítóprogramok előadás (A,C,T szakirány) Fordítóprogramok célja és szerkezete

Fordítóprogramról általában Assembly, gépi kód

Assembly és assembler

- az assembly nyelvben
 - a gépi kód utasításaihoz neveket (*mnemonikokat*) rendelünk
 - a regiszterekre névvel hivatkozunk
 - az egyes memoriacímeket címkékkel azonosítjuk
 - gyakran közülös állomás a magasszintű nyelvről gépi kódra történő fordítás során
- assembler: assembly nyelvről gépi kódra fordít

4 Fordítóprogramok előadás (A,C,T szakirány) Fordítóprogramok célja és szerkezete

Fordítóprogramról általában Assembly, gépi kód

Assembly és assembler

Gépi kód	Assembly
----------	----------

```
B9 00 00 00 00
B8 00 00 00 00
81 F9 0A 00 00 00
7D 06
03 04 8B
41
EB F2
```

```
mov ecx,0
mov eax,0
eleje:
cmp ecx,10
jge vege
add eax,[ebx+4*ecx]
inc ecx
jmp eleje
vege:
```

5 Fordítóprogramok előadás (A,C,T szakirány) Fordítóprogramok célja és szerkezete

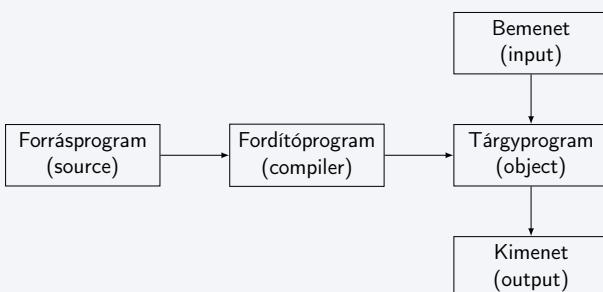
Példák

- magasszintű nyelvek fordítóprogramjai (compiler)
 - gcc, javac, ghc ...
- assembly nyelvek fordítóprogramjai (assembler)
 - nasm, masm, gas ...
- nyelvek közötti fordítás (translator)
 - Fortran \Rightarrow C, latex2html, dvips ...
- egyéb fordítóprogramok
 - latex
 - hardware leíró nyelvből áramkörök tervezés

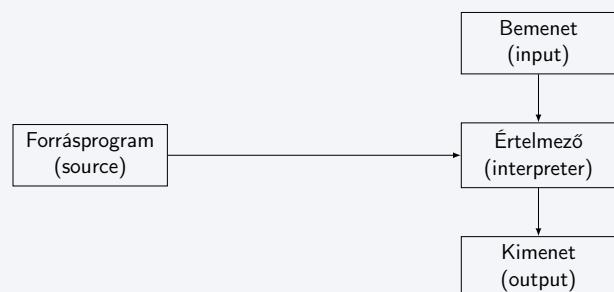
Fordítás és interpretálás

- Fordítás:** A forráskódot a fordítóprogram **tárgyprogrammá** alakítja. A tárgyprogramokból a szerkesztés során **futtatható állomány** jön létre.
 - Fordítási idő:** amikor a fordító dolgozik
 - Futási idő:** amikor a program fut
- Interpretálás:** A forráskódot az interpreter értelmezi és azonnal végrehajtja.
 - a fordítási és futási idő nem különbözik

Fordítás



Interpretálás



Fordítás vs. interpretálás

Fordítás

- gyorsabb végrehajtás
- a forrás alaposabb ellenőrzése
- minden platformra külön-külön le kell fordítani
- C++, Ada, Haskell ...

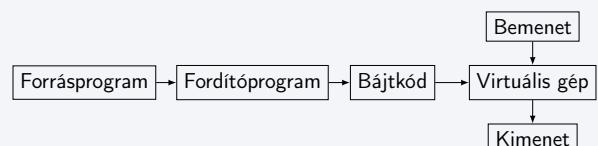
Interpretálás

- rugalmasabb (pl. utasítások fordítási időben történő összeállítása)
- minden platformon azonnal futtatható, ahol az interpreter rendelkezésre áll
- Perl, Php, Javascript ...

Fordítás + interpretálás

Fordítás és interpretálás egymásra építve (pl. Java):

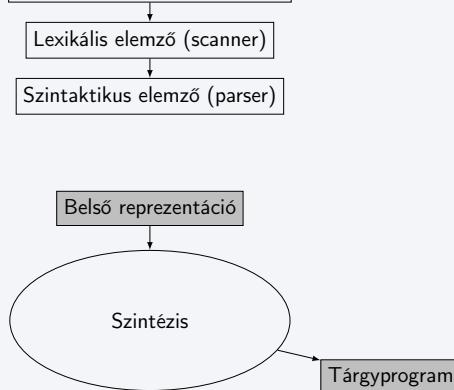
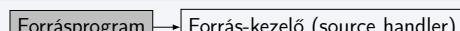
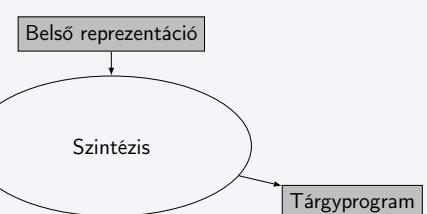
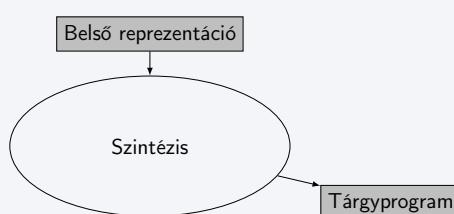
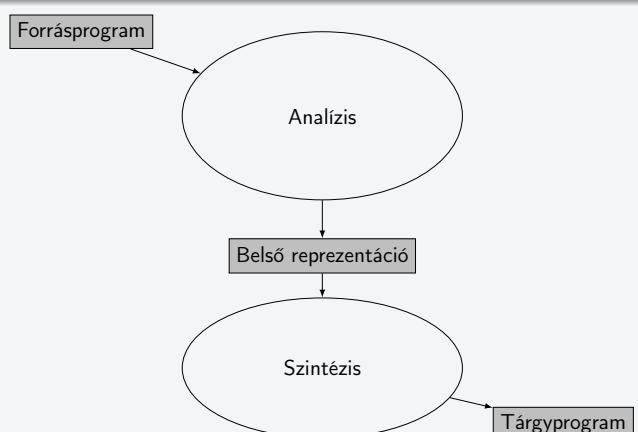
- A forráskód fordítása **bájtkódra**.
- A bájtkód interpretálása **virtuális géppel**.



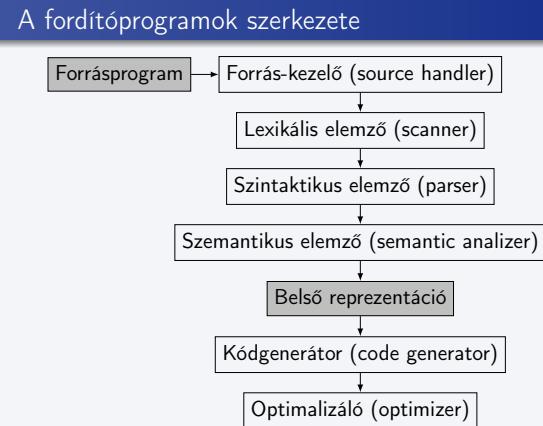
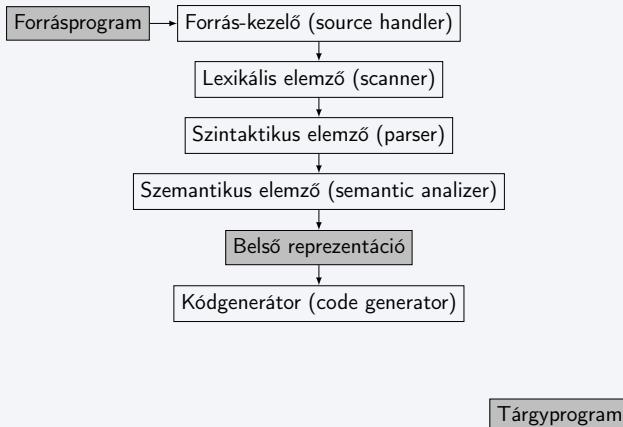
Virtuális gép: olyan gép szoftveres megvalósítása, amelynek a bájtkód a gépi kódja.

Fejlődés

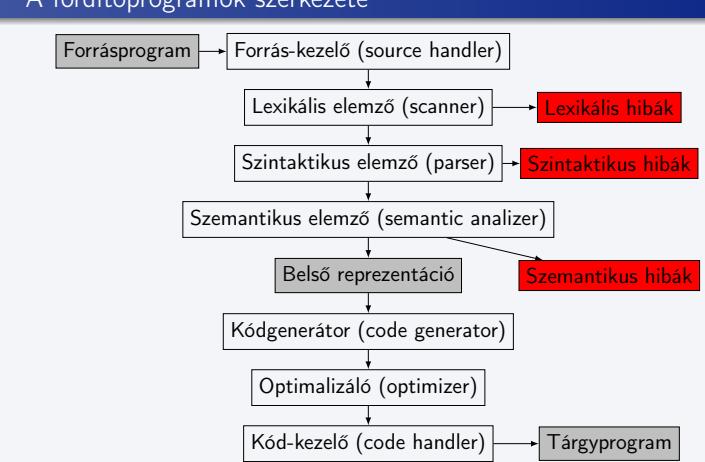
- 1957, az első Fortran compiler: 18 emberévnyi munka
- azóta fejlődött a formális nyelvek és automaták elmélete
- ma: a fordítóprogramok létrehozásának egy része automatizálható
 - a programszöveget (szintaktikusan) elemző program: automatikusan generálható
 - a szemantikus ellenőrzést (pl. típusok) és kódgenerálást végző program: nem generálható automatikusan, de nem túl bonyolult
 - kódoptimalizálás: nehéz feladat
- Példa: elemzőgenerátorok



A fordítóprogramok szerkezete



Tárgyprogram



Tárgyprogram

Forrás kezelő

- bemenet: fájl(ok)
- kimenet: karakterSORozat
- feladata:
 - fájlok megnyitása
 - olvasás
 - pufferelés
 - az operációsrendszer-specifikus feladatok elvégzése
 - ha a fordítóprogram készít listafájlt, akkor ezt is kezeli

Példa

input.cpp \Rightarrow $x_1 = x_1 + 1;$

Lexikális elemző

- bement: karakterSORozat
- kimenet: szimbólumsOROZAT, lexikális hIBÁK
- feladata:
 - lexikális egységek (szimbólumok) felismerése: azonosító, konstans, kulcsszó...

Példa

 $x_1 = x_1 + 1;$
 \Rightarrow
 változó[x] operátor[=] változó[x] operátor[+] konstans[1] utasításVÉG

Szintaktikus elemző

- bemenet: szimbólumsorozat
- kimenet: szintaxisfa, szintaktikus hibák
- feladata:
 - a program szerkezetének felismerése
 - a szerkezet ellenőrzése: megfelel-e a nyelv definíciójának?



Szemantikus elemző

- bemenet: szintaktikusan elemzett program (szintaxisfa, szimbólumtábla, ...)
- kimenet: szemantikusan elemzett program, szemantikus hibák
- feladata:
 - típusellenőrzés
 - változók láthatóságának ellenőrzése
 - eljárások hívása megfelel-e a szignatúrának?
 - stb.

Szintaktikus hibák

```
x = x 1;
x = x + ;
if x==0) {}
```

Szemantikus hibák

```
if( "alma" == 3.14 ) {}
void f(int x) {} f(3,4);
{ int y; } y++;
```

Kódgenerátor

- bemenet: szemantikusan elemzett program
- kimenet: tárgykód utasításai (pl. assembly, gépi kód)
- feladata:
 - a forrásprogrammal ekvivalens tárgyprogram készítése

Az $x = x + 1$; utasítás egy lehetséges megvalósítása

```
mov eax,1
push eax
mov eax,[x]
pop ebx
add eax,ebx
mov [x],eax
```

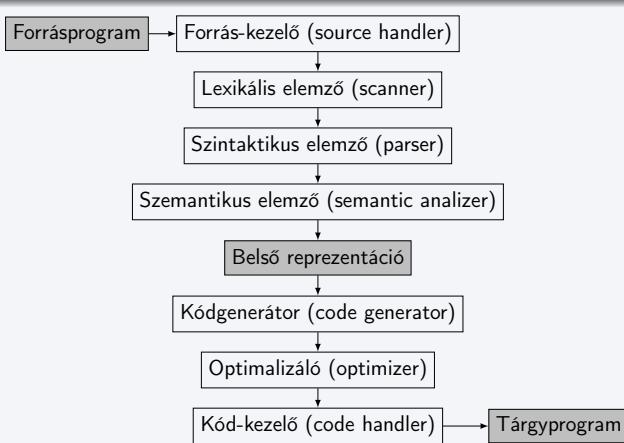
Kódpotimalizáló

- bemenet: tárgykód
- kimenet: optimalizált tárgykód
- feladata: valamelyen szempontok szerint jobb kód készítése
 - pl. futási sebesség növelése, méret csökktése
 - pl. felesleges utasítások megszüntetése, ciklusok kifejtése...
- optimalizáció végezhető
 - az eredeti programon (szemantikus elemzés után)
 - a tárgykódon (a kódgenerálás után)

Az $x = x + 1$; utasítás kódjának optimalizálása

```
mov eax,1
push eax
mov eax,[X]
pop ebx
add eax,ebx
mov [X],eax
      ==>
inc dword [X]
```

Emlékeztető: a fordítás komponensei



A fordítás menetei

- az egyes komponensek egymást követik
 - az egyik kimenete a másik bemenete
- ez nem jelenti azt, hogy pl. a lexikális elemzőnek be kell fejezni a szintaktikus elemzés előtt
 - a szintaktikus elemző meghívhatja a lexikálisat, amikor egy újabb szimbólumra van szüksége
- eredmény: egy utasításnak akár a tárgykódja is elkészülhet, amikor a következő utasítás még be sincs olvasva
- a kódoptimalizálásnál viszont jellemzően újra át kell olvasni az egész tárgykódot (akár többször is)

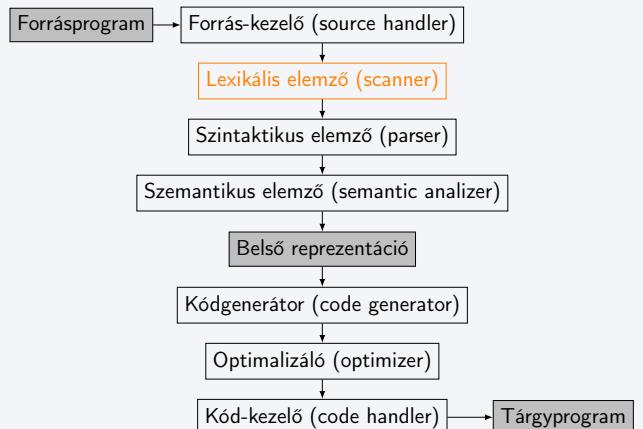
A fordítás menetszáma

A fordítás annyi menetes, ahányszor a programszöveget (vagy annak belső reprezentációját) végigolvassa a fordító a teljes fordítási folyamat során.

A lexikális elemzés

Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés helye



1 Fordítóprogramok előadás (A,C,T szakirány) A lexikális elemzés

2 Fordítóprogramok előadás (A,C,T szakirány) A lexikális elemzés

Elemzési lépések szétválasztása

- lexikális elemzés: megadható reguláris nyelvtannal (3-as)
- szintaktikus elemzés: megadható környezetfüggetlen nyelvtannal (2-es)
- szemantikus elemzés: környezetfüggő (1-es)
 - pl. egy változó használatának helyessége függhet a deklarációjától, azaz a környezettől

A három lépés szétválasztásának oka, hogy **egyszerű feladathoz ne használunk bonyolult eszközöket**.

Reguláris nyelvtan

Reguláris nyelvtanokban (Chomsky 3) a szabályok a következő alakúak lehetnek:

Jobbrekurzív eset

$$\begin{aligned} A &\rightarrow a \\ A &\rightarrow aB \\ A &\rightarrow \epsilon \end{aligned}$$

Balrekurzív eset

$$\begin{aligned} A &\rightarrow a \\ A &\rightarrow Ba \\ A &\rightarrow \epsilon \end{aligned}$$

3 Fordítóprogramok előadás (A,C,T szakirány) A lexikális elemzés

4 Fordítóprogramok előadás (A,C,T szakirány) A lexikális elemzés

Reguláris nyelvtan

Reguláris nyelvtanokban (Chomsky 3) a szabályok a következő alakúak lehetnek:

Jobbrekurzív eset

$$\begin{aligned} A &\rightarrow a \\ A &\rightarrow aB \\ A &\rightarrow \epsilon \end{aligned}$$

Balrekurzív eset

$$\begin{aligned} A &\rightarrow a \\ A &\rightarrow Ba \\ A &\rightarrow \epsilon \end{aligned}$$

Példa: változónevek leírása

$$\begin{aligned} V &\rightarrow \underline{a} F \mid \underline{b} F \mid \dots \\ F &\rightarrow \epsilon \mid \underline{a} F \mid \underline{b} F \mid \dots \mid \underline{1} F \mid \underline{2} F \mid \dots \end{aligned}$$

Reguláris kifejezés

- kifejezőreje azonos a reguláris nyelvtanokéval
- alapelemek:
 - üres halmaz
 - üres szöveget tartalmazó halmaz
 - egy karaktert tartalmazó halmaz
- konstruktív műveletek:
 - konkatenáció
 - unió: \mid
 - lezárás: $*$
- további „kényelmi” műveletek: $+, ?$

Példák (flex szintaxissal)

változónév: `[a-zA-Z] [a-zA-Z0-9]*`

egész szám: `(\+|\-)?` $[0-9]^+$

törtszám: `[0-9]^+.\^+[0-9]^*`

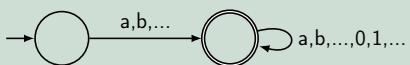
4 Fordítóprogramok előadás (A,C,T szakirány) A lexikális elemzés

5 Fordítóprogramok előadás (A,C,T szakirány) A lexikális elemzés

Véges determinisztikus automaták

- kifejezőreje azonos a reguláris nyelvtanokéval és a reguláris kifejezésekével
- elemei:
 - ábécé
 - állapotok halmaza
 - átmennetfüggvény
 - kezdőállapot
 - végállapotok halmaza

Változónevek elfogadása automatával



Véges determinisztikus automata implementációja

- egymásba ágyazott if vagy case utasításokkal egy cikluson belül
 - elágazunk a pillanatnyi állapot szerint
 - ezen belül elágazunk a következő karakter szerint
 - az egyes ágakban beállítjuk a következő állapotot és kezdjük elóról
- táblázattal
 - a táblázat soraihoz az állapotok, oszlopaihoz a karakterek vannak rendelve
 - celláiban a következő állapot sorszáma van
 - a következő állapot a pillanatnyi állapot sorában és az olvasott karakter oszlopában található

6 Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

7 Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

A lexikális elemző működése

- Az abc bemenet esetén a, ab és abc is legális változónév. Melyiket kellene felismerni?
 - A lexikális elemző minden a lehető leghosszabb karaktersorozatot ismeri fel.

A lexikális elemző működése

- Az abc bemenet esetén a, ab és abc is legális változónév. Melyiket kellene felismerni?
 - A lexikális elemző minden a lehető leghosszabb karaktersorozatot ismeri fel.
- A while input megfelel a változónév definíciójának és egy kulcsszó is egyben. Melyiket kellene felismerni?
 - A lexikális elemző megadásakor sorbarendezhetjük a szimbólumok definíciót. Ha egyszerre több szimbólum is felismerhető, a sorrendben korábbi lesz az eredmény. (Tehát a kulcsszavak definícióját kell előre venni.)

8 Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

8 Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

Kulcsszavak és standard szavak

Kulcsszó

A kulcsszavaknak előre adott jelentésük van, és ez nem definiálható felül.

Kulcsszavak és standard szavak

Kulcsszó

A kulcsszavaknak előre adott jelentésük van, és ez nem definiálható felül.

Standard szó

A standard szavaknak előre adott jelentésük van, de ez felüldefiniálható.

9 Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

9 Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

Kulcsszavak és standard szavak

Kulcsszó

A kulcsszavaknak előre adott jelentésük van, és ez nem definiálható felül.

Standard szó

A standard szavaknak előre adott jelentésük van, de ez felüldefiniálható.

- Ha a kulcsszavakat is véges determinisztikus automatával akarjuk felismerni, nagyon nagy méretű automatát kaphatunk.
- Jobb módszer egy táblázatban tárolni őket: akárhányszor a lexikális elemző egy azonosítót ismer fel, meg kell nézni, hogy benne van-e ebben a táblázatban. (Ha igen, akkor kulcsszó, különben azonosító.)

9

Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

Előreolvasás

A lexikális elemző időnként több karaktert is előreolvas a szimbólum felismeréséhez.

10

Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

Előreolvasás

A lexikális elemző időnként több karaktert is előreolvas a szimbólum felismeréséhez.

④ példa: az egyes szimbólomok egymás prefixei

- egész szám: [0-9]+
- valós szám: [0-9]+."[0-9]+
- 3.14 ⇒ valós szám
- 3.x ⇒ egész szám, majd lexikális hiba
- az elemző megjegyzí a legutóbbi érvényes állapotot

Előreolvasás

A lexikális elemző időnként több karaktert is előreolvas a szimbólum felismeréséhez.

④ példa: az egyes szimbólomok egymás prefixei

- egész szám: [0-9]+
- valós szám: [0-9]+."[0-9]+
- 3.14 ⇒ valós szám
- 3.x ⇒ egész szám, majd lexikális hiba
- az elemző megjegyzí a legutóbbi érvényes állapotot

④ példa: Fortran (a szóközöknek semmi szerepe nem volt!)

- D0 10 I = 1.1000 (ez egy értékkedés a D010I változónak)
- D0 10 I = 1,1000 (ez egy ciklus)

10

Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

10

Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

Előreolvasás

A lexikális elemző időnként több karaktert is előreolvas a szimbólum felismeréséhez.

④ példa: az egyes szimbólomok egymás prefixei

- egész szám: [0-9]+
- valós szám: [0-9]+."[0-9]+
- 3.14 ⇒ valós szám
- 3.x ⇒ egész szám, majd lexikális hiba
- az elemző megjegyzí a legutóbbi érvényes állapotot

④ példa: Fortran (a szóközöknek semmi szerepe nem volt!)

- D0 10 I = 1.1000 (ez egy értékkedés a D010I változónak)
- D0 10 I = 1,1000 (ez egy ciklus)
- megoldás: D0 / [0-9]+ [a-zA-Z0-9]* = [a-zA-Z0-9]* ,
 - a '/' az előreolvasási operátor
 - r/s jelentése: ismert fel r-t, de csak ha s követi
 - r felismerése után s visszakerüli a bemenetbe
 - a leghosszabb karakterszorozatról való döntéskor | r | + | s | számít

Szemantikus értékek és szimbólumtábla

- A lexikális elemzőnek a felismert szimbólum fajtáján kívül egyéb információkat is továbbítania kell.

- változó: a változó neve
- konstans: a konstans értéke

- Ezekre a **szemantikus értékekre** szemantikus elemzéshez és a kódgeneráláshoz van szükség.

- A változókat és azok adatait a **szimbólumtáblába** kell felírni.
 - Ezt általában a szintaktikus elemző teszi meg a változó deklarációjának felismerésekor.

10

Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

11

Fordítóprogramok előadás (A,C,T szakirány)

A lexikális elemzés

Direktívák

Példa direktívára

```
#include "my.h"
#define valami 42
#ifndef FELTETEL
int akarmi() { return valami; }
#endif
```

Célszerű egy előfeldolgozó fázis beiktatása a lexikális és a szintaktikus elemzés közé, ami

- feljegyzi a makródefiníciókat,
- elvégzi a makróhelyettesítéseket,
- meghívja a lexikális elemzést a beillesztett fájlokra,
- kiértékeli a feltételeket és dönt a kód részletek beillesztéséről vagy törléséről.

Hibatípusok és javítási lehetőségek

illegális karakter (pl. add?ress, ? legyen az illegális karakter)

- az éppen épített szimbólum eldobása és folytatás a következő karaktertől (eredmény: ress azonosító)
- a karakter kihagyása (eredmény: address azonosító)
- a karakter helyettesítése szóközzel (eredmény: add és ress azonosítók)

Hibatípusok és javítási lehetőségek

illegális karakter (pl. add?ress, ? legyen az illegális karakter)

- az éppen épített szimbólum eldobása és folytatás a következő karaktertől (eredmény: ress azonosító)
- a karakter kihagyása (eredmény: address azonosító)
- a karakter helyettesítése szóközzel (eredmény: add és ress azonosítók)

elgépelt kulcsszó (pl. while helyett wile whille wjile)

- a lexikális elemző azonosítónak fogja felismerni, de egy ügyes szintaktikus elemző kijavíthatja a hibát
- azokban a nyelveken, ahol a kulcsszavakat speciális módon jelölök, a lexikális elemző is felismerheti és javíthatja a hibát
 - speciális jelölés lehet pl. adott karakterrel való kezdés, zárójelezés, nagybetű használata
 - a szintaxiskiemelés (pl. vastagítás, színezés) nem használható erre a célra, mert az láthatatlan a lexikális elemző számára!

Hibatípusok és javítási lehetőségek

kihagyott szimbólum (pl. 1+a helyett 1a, a+1 helyett a1)

- ezeket csak a szintaktikus elemző tudja észrevenni

Hibatípusok és javítási lehetőségek

kihagyott szimbólum (pl. 1+a helyett 1a, a+1 helyett a1)

- ezeket csak a szintaktikus elemző tudja észrevenni

hibás számformátum (pl. 1.23.45)

- valamelyiket illegális karakternek lehet tekinteni

Hibatípusok és javítási lehetőségek

kihagyott szimbólum (pl. 1+a helyett 1a, a+1 helyett a1)

- ezeket csak a szintaktikus elemző tudja észrevenni

hibás számformátum (pl. 1.23.45)

- valamelyiket illegális karakternek lehet tekinteni

befejezetlen megjegyzések és sztringek (pl. 'alma ..., /* megjegyzés ...)

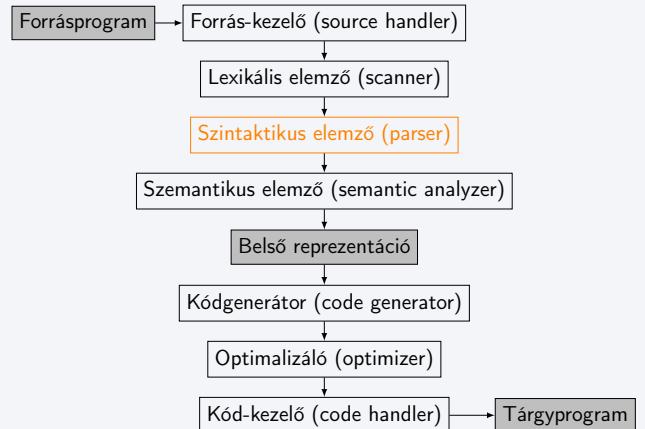
- könnyen az egész további program megjegyzésbe kerülhet
- sor végén, illetve fájl végén lehet jelezni a hibát

A szintaktikus elemzés

Fordítóprogramok előadás (A,C,T szakirány)

A szintaktikus elemzésről általában

A szintaktikus elemzés helye



1 Fordítóprogramok előadás (A,C,T szakirány)

A szintaktikus elemzés

2 Fordítóprogramok előadás (A,C,T szakirány)

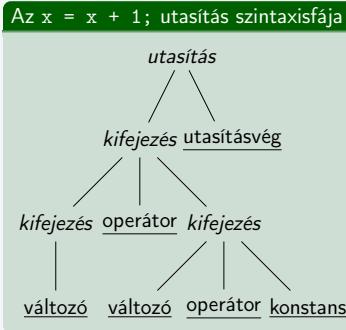
A szintaktikus elemzés

1 Fordítóprogramok előadás (A,C,T szakirány) 2 Fordítóprogramok előadás (A,C,T szakirány) A szintaktikus elemzés

A szintaktikus elemzésről általában

Szintaktikus elemzés

- bemenet:
szimbólumsorozat
- kimenet:
szintaxisfa,
szintaktikus hibák
- feladata:
 - a program szerkezetének felismerése
 - a szerkezet ellenőrzése:
megfelel-e a nyelv definíciójának?



3 Fordítóprogramok előadás (A,C,T szakirány)

A szintaktikus elemzés

4 Fordítóprogramok előadás (A,C,T szakirány)

A szintaktikus elemzés

3 Fordítóprogramok előadás (A,C,T szakirány) 4 Fordítóprogramok előadás (A,C,T szakirány) A szintaktikus elemzés

Jelölések és fogalmak

Környezetfüggetlen nyelvtanok

- a szintaktikus elemzés:
környezetfüggetlen nyelvtannal (Chomsky 2)
- $A \rightarrow \alpha$ alakú szabályok

„az A szimbólum a környezetétől függetlenül helyettesíthető α -ra”

Jelölések és fogalmak

Környezetfüggetlen nyelvtanok

- a szintaktikus elemzés:
környezetfüggetlen nyelvtannal (Chomsky 2)
- $A \rightarrow \alpha$ alakú szabályok

„az A szimbólum a környezetétől függetlenül helyettesíthető α -ra”

Jelölések és fogalmak

Levezetések és kapcsolódó fogalmak

Jelölés	Jelentés
$A \rightarrow \alpha$	szabály
$A \Rightarrow \alpha$	egy lépéses levezetés (1 szabály alk.)
$A \Rightarrow^* \alpha$	nulla, egy vagy több lépéses levezetés
$A \Rightarrow^+ \alpha$	legalább egy lépésből álló levezetés

Jelölés

Jelölés	Jelentés
$a, b, c\dots$	terminális szimbólum (ezek a lexikális elemek, tokenek)
$A, B, C\dots$	nemterminális szimbólum
$X, Y, Z\dots$	terminális vagy nemterminális
$x, y, z\dots$	terminális szimbólumok sorozata
$\alpha, \beta, \gamma\dots$	terminális vagy nemterminális szimbólumok sorozata

4 Fordítóprogramok előadás (A,C,T szakirány) A szintaktikus elemzés

5 Fordítóprogramok előadás (A,C,T szakirány) A szintaktikus elemzés

Levezetések és kapcsolódó fogalmak

Jelölés	Jelentés
$A \rightarrow \alpha$	szabály
$A \Rightarrow \alpha$	egy lépéses levezetés (1 szabály alk.)
$A \Rightarrow^* \alpha$	nulla, egy vagy több lépéses levezetés
$A \Rightarrow^+ \alpha$	legalább egy lépésből álló levezetés

- **mondat:** a startszimbólumból levezethető terminális sorozat ($S \Rightarrow^* x$)
- **mondatforma:** a startszimbólumból jelből levezethető bármilyen sorozat ($S \Rightarrow^* \alpha$)
- **részmondat:** β az $\alpha_1\beta\alpha_2$ mondatforma részmondata, ha $S \Rightarrow^* \alpha_1A\alpha_2 \Rightarrow^+ \alpha_1\beta\alpha_2$.
- **egyszerű részmondat:** β az $\alpha_1\beta\alpha_2$ mondatforma egyszerű részmondata, ha $S \Rightarrow^* \alpha_1A\alpha_2 \Rightarrow \alpha_1\beta\alpha_2$.

Követelmények

- **ciklusmentesség:** nincs $A \Rightarrow^+ A$ levezetés

- ellenpélda:

$$\begin{aligned} A &\rightarrow B \\ B &\rightarrow A \end{aligned}$$

Követelmények

- **ciklusmentesség:** nincs $A \Rightarrow^+ A$ levezetés
 - ellenpélda:

$$\begin{aligned} A &\rightarrow B \\ B &\rightarrow A \end{aligned}$$
- **redukáltság:** „nincsenek felesleges nemterminálisok”
 - minden nemterminális szimbólum előfordul valamelyik mondatformában
 - mindegyikból levezethető valamely terminális sorozat
 - ellenpélda: $A \rightarrow aA$, ha ez az egyetlen szabály A -ra

Követelmények

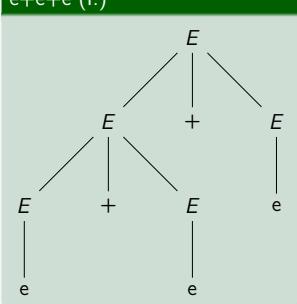
- **ciklusmentesség:** nincs $A \Rightarrow^+ A$ levezetés
 - ellenpélda:

$$\begin{aligned} A &\rightarrow B \\ B &\rightarrow A \end{aligned}$$
- **redukáltság:** „nincsenek felesleges nemterminálisok”
 - minden nemterminális szimbólum előfordul valamelyik mondatformában
 - mindegyikból levezethető valamely terminális sorozat
 - ellenpélda: $A \rightarrow aA$, ha ez az egyetlen szabály A -ra
- **egyértelműség:** minden mondathoz pontosan egy szintaxisfa tartozik

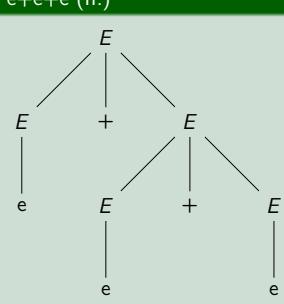
Példa nem egyértelmű nyelvtanra

$$E \rightarrow e \mid E + E$$

e+e+e (I.)



e+e+e (II.)



Legbal és legjobb levezetések

- **Legbal:** minden a *legbaloldalibb* nemterminálist helyettesítjük
- **Legjobb:** minden a *legjobboldalibb* nemterminálist helyettesítjük

Legbal és legjobb levezetések

- **Legbal:** minden a *legbaloldalibb* nemterminálist helyettesítjük

Legbal levezetés

$$S \Rightarrow AB \Rightarrow aaB \Rightarrow aab$$

- **Legjobb:** minden a *legjobboldalibb* nemterminálist helyettesítjük

Legjobb levezetés

$$S \Rightarrow AB \Rightarrow Ab \Rightarrow aab$$

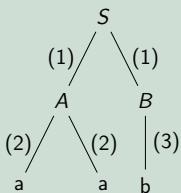
Elemzési irányok

- **Felülről lefelé:** A startszimbólumból indulva, felülről lefelé építjük a szintaxisfát. A mondatforma baloldalán megjelenő terminálisokat illesztjük az elemzendő szövegre.

- **Alulról felfelé:** Az elemzendő szöveg összetartozó részeit helyettesítjük nemterminális szimbólumokkal (redukció) és így alulról, a startszimbólum felé építjük a fát.

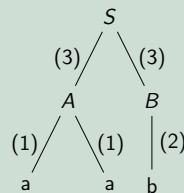
Elemzési irányok

Felülről lefelé



$$S \rightarrow^{(1)} AB \rightarrow^{(2)} aaB \rightarrow^{(3)} aab$$

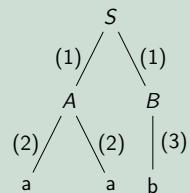
Alulról felfelé



$$aab \leftarrow^{(1)} Ab \leftarrow^{(2)} AB \leftarrow^{(3)} S$$

Elemzési irányok

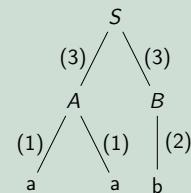
Felülről lefelé



$$S \rightarrow^{(1)} AB \rightarrow^{(2)} aaB \rightarrow^{(3)} aab$$

• Ez egy *legbal* levezetés!

Alulról felfelé



$$aab \leftarrow^{(1)} Ab \leftarrow^{(2)} AB \leftarrow^{(3)} S$$

• Ez egy *legjobb* levezetés inverze!

Felülről lefelé elemzések

- Probléma: „Melyik helyettesítési szabályt kell alkalmazni?”

Felülről lefelé elemzések

- Probléma: „Melyik helyettesítési szabályt kell alkalmazni?”

Stratégiák:

- visszalépéses keresés (backtrack): ha nem illeszkednek a szövegre a mondatforma baloldalán megjelenő terminálisok, lépjünk vissza, és válasszunk másik szabályt
⇒ **visszalépéses felülről lefelé elemzések** (nem tananyag)

- lassú
- ha hibás a szöveg, az csak túl későn derül ki

Felülről lefelé elemzések

- Probléma: „Melyik helyettesítési szabályt kell alkalmazni?”
- Stratégiák:
 - visszalépéses keresés (backtrack): ha nem illeszkednek a szövegre a mondatforma baloldalán megjelenő terminálisok, lépjünk vissza, és válasszunk másik szabályt
⇒visszalépéses felülről lefelé elemzések (nem tananyag)
 - lassú
 - ha hibás a szöveg, az csak túl későn derül ki
 - előreolvasás: olvassunk előre a szövegen valahány szimbólumot, és az alapján döntsünk az alkalmazandó szabályról
⇒LL elemzések
 - csak szűk nyelvosztályra alkalmazható

Alulról felfelé elemzések: „Mit redukálunk?”

Alulról felfelé elemzések: „Mit redukálunk?”

- visszalépéses keresés (backtrack): ha nem sikerül eljutni a startszimbólumig, lépjünk vissza, és válasszunk másik redukciót
⇒visszalépéses alulról felfelé elemzések (nem tananyag)
 - lassú
 - ha hibás a szöveg, az csak túl későn derül ki

Alulról felfelé elemzések: „Mit redukálunk?”

- visszalépéses keresés (backtrack): ha nem sikerül eljutni a startszimbólumig, lépjünk vissza, és válasszunk másik redukciót
⇒visszalépéses alulról felfelé elemzések (nem tananyag)
 - lassú
 - ha hibás a szöveg, az csak túl későn derül ki
- precedenciák használata: az egyes szimbólumok között adjunk meg precedenciarelációkat és ennek segítségével határozzuk meg a megfelelő redukciót
⇒precedencia elemzések (nem tananyag)
 - ma már kevessé használt
 - operátorokkal felépített kifejezések esetén természetes a használata

Alulról felfelé elemzések: „Mit redukálunk?”

- visszalépéses keresés (backtrack): ha nem sikerül eljutni a startszimbólumig, lépjünk vissza, és válasszunk másik redukciót
⇒visszalépéses alulról felfelé elemzések (nem tananyag)
 - lassú
 - ha hibás a szöveg, az csak túl későn derül ki
- precedenciák használata: az egyes szimbólumok között adjunk meg precedenciarelációkat és ennek segítségével határozzuk meg a megfelelő redukciót
⇒precedencia elemzések (nem tananyag)
 - ma már kevessé használt
 - operátorokkal felépített kifejezések esetén természetes a használata
- előreolvasás: olvassunk előre a szövegen valahány szimbólumot, és az alapján döntünk a redukcióról
⇒LR elemzések
 - minden programozási nyelvhez lehet (LR) elemzőt készíteni
 - majdnem mindenhez lehet gyors (LALR) elemzőt készíteni

LL elemzések

Fordítóprogramok előadás (A,C,T szakirány)

Az LL elemzésekéről általában

LL elemzések

- felülről lefelé elemzés
- alapötlet: k terminális szimbólum előreolvasásával döntünk az alkalmazandó szabályról
- név: Left to right, using a Leftmost derivation (balról jobbra, legbal levezetéssel)

Ellenpélda

Nem minden grammatika esetén alkalmazható!

Nem elemezhető LL módszerrel

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow a \mid aA \\ B &\rightarrow ab \mid aBb \end{aligned}$$

Az LL elemzésekéről általában

Ellenpélda

Nem minden grammatika esetén alkalmazható!

Nem elemezhető LL módszerrel

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow a \mid aA \\ B &\rightarrow ab \mid aBb \end{aligned}$$

Akárhogyan rögzítjük az előreolvasandó szimbólumok számát (k), a

$$\underbrace{aa\dots a}_{k}\dots$$

inputra nem tudjuk eldönten, hogy $S \rightarrow A$ vagy $S \rightarrow B$ a jó választás.

FIRST halmazok

$FIRST_k(\alpha)$: az α mondatformából levezethető terminális sorozatok k hosszúságú kezdőszelései

(ha a sorozat hossza kisebb mint k , akkor az egész sorozat eleme $FIRST_k(\alpha)$ -nak, akár $\epsilon \in FIRST_k(\alpha)$ is előfordulhat)

Az LL elemzésekéről általában

FIRST halmazok

$FIRST_k(\alpha)$: az α mondatformából levezethető terminális sorozatok k hosszúságú kezdőszelései

(ha a sorozat hossza kisebb mint k , akkor az egész sorozat eleme $FIRST_k(\alpha)$ -nak, akár $\epsilon \in FIRST_k(\alpha)$ is előfordulhat)

Definíció: $FIRST_k(\alpha)$

$$\begin{aligned} FIRST_k(\alpha) = \{x \mid \alpha \Rightarrow^* x\beta \wedge |x| = k\} \cup \\ \{x \mid \alpha \Rightarrow^* x \wedge |x| < k\} \end{aligned}$$

LL(k) grammatikák

LL(k) grammatika: a levezetés tetszőleges pontján a szöveg következő k terminálisa meghatározza az alkalmazandó levezetési szabályt

LL(k) grammatikák

LL(k) grammatika: a levezetés tetszőleges pontján a szöveg következő k terminálisa meghatározza az alkalmazandó levezetési szabályt

Definíció: LL(k) grammatica

Tetszőleges

$$S \Rightarrow^* wA\beta \Rightarrow w\alpha_1\beta \Rightarrow^* wx$$

$$S \Rightarrow^* wA\beta \Rightarrow w\alpha_2\beta \Rightarrow^* wy$$

levezetéspárra $FIRST_k(x) = FIRST_k(y)$ esetén $\alpha_1 = \alpha_2$.

LL elemzések

- példa szoftverek:
 - ANTLR parser generátor: <http://www.antlr.org/>
 - Coco/R: <http://www.ssw.uni-linz.ac.at/coco/>
- gyakorlatban: az LL(1) elemzést könnyű megvalósítani
 - egyszerű LL(1)
 - epszilonmentes LL(1)
 - általános

Definíció: Egyszerű LL(1)

Olyan LL(1) grammatika, amelyben a szabályok jobboldala terminális szimbólummal kezdődik (ezért ϵ -mentes is). (Az összes szabály $A \rightarrow a\alpha$ alakú.)

Következmény: Egyszerű LL(1) grammatika esetén az azonos nemterminálhoz tartozó szabályok jobboldalai különböző terminálissal kezdődnek.

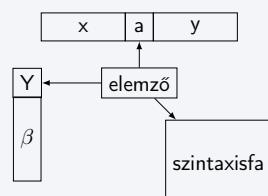
Tétel

Egy grammatika pontosan akkor egyszerű LL(1), ha

- csak $A \rightarrow a\alpha$ alakú szabályai vannak
- és ha $A \rightarrow a_1\alpha_1$ és $A \rightarrow a_2\alpha_2$ két különböző szabály, akkor $a_1 \neq a_2$.

(A fordítóprogramok könyvben ez szerepel definícióként!)

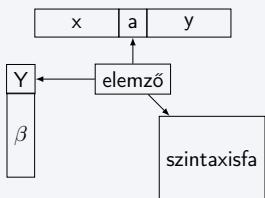
Egyszerű LL(1) elemzés



- xay : bemenet

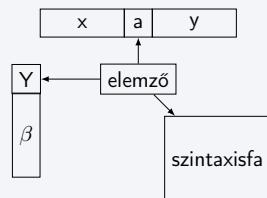
- $Y\beta$: aktuális mondatforma (veremben tároljuk)

Egyszerű LL(1) elemzés



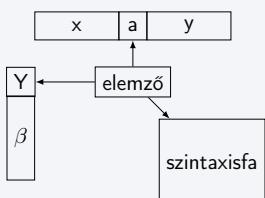
- ha a verem tetején terminális szimbólum van:
 - ha egyezik a szöveg következő karakterével: pop és lépés a szövegen
 - különben: hiba

Egyszerű LL(1) elemzés



- ha a verem tetején nemterminális szimbólum (A) van:
 - ha van $A \rightarrow a\alpha$ szabály: A helyére $a\alpha$ és bejegyzés a szintaxisfába
 - különben: hiba

Egyszerű LL(1) elemzés



- ha a verem üres:
 - ha a szöveg végére értünk: OK
 - különben hiba

Elemző táblázat

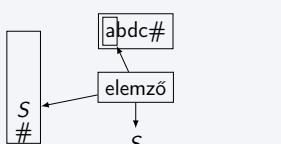
Az egyes akciók táblázatba foglalhatók.

Példa nyelvtan

$S \rightarrow aS$	(1)	$ $	bAc	(2)
$A \rightarrow bAc$	(3)	$ $	d	(4)

- #: veremalja és fájlvége
- üres helyek: hiba

Példa: abdc

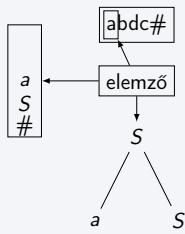


• $(abdc\#, S\#, \epsilon)$

Példa: abdc

	a	b	c	d	#
S	$S \rightarrow^{(1)} aS$	$S \rightarrow^{(2)} bAc$			
A		$A \rightarrow^{(3)} bAc$		$A \rightarrow^{(4)} d$	
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

Példa: abdc

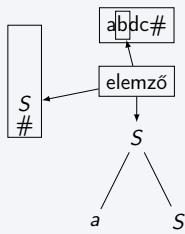


- $(abdc\#, S\#, \epsilon)$
- $(abdc\#, aS\#, 1)$

Példa: abdc

	a	b	c	d	#
S	$S \rightarrow^{(1)} aS$	$S \rightarrow^{(2)} bAc$			
A		$A \rightarrow^{(3)} bAc$		$A \rightarrow^{(4)} d$	
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

Példa: abdc

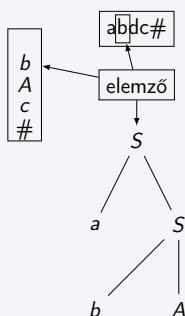


- $(abdc\#, S\#, \epsilon)$
- $(abdc\#, aS\#, 1)$
- $(bdc\#, S\#, 1)$

Példa: abdc

	a	b	c	d	#
S	$S \rightarrow^{(1)} aS$	$S \rightarrow^{(2)} bAc$			
A		$A \rightarrow^{(3)} bAc$		$A \rightarrow^{(4)} d$	
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

Példa: abdc

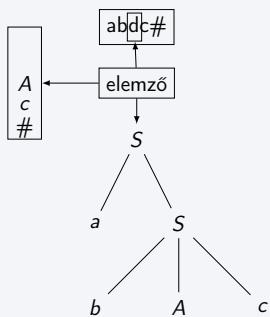


- $(abdc\#, S\#, \epsilon)$
- $(abdc\#, aS\#, 1)$
- $(bdc\#, S\#, 1)$
- $(bdc\#, bAc\#, 12)$

Példa: abdc

	a	b	c	d	#
S	$S \rightarrow^{(1)} aS$	$S \rightarrow^{(2)} bAc$			
A		$A \rightarrow^{(3)} bAc$		$A \rightarrow^{(4)} d$	
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

Példa: abdc

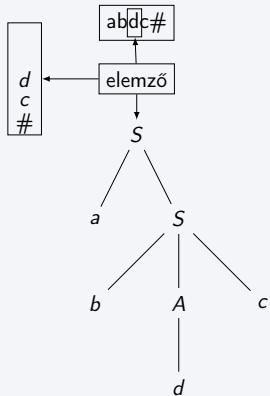


- $(abdc\#, S\#, \epsilon)$
- $(abdc\#, aS\#, 1)$
- $(bdc\#, S\#, 1)$
- $(bdc\#, bAc\#, 12)$
- $(dc\#, Ac\#, 12)$

Példa: abdc

	a	b	c	d	#
S	$S \rightarrow^{(1)} aS$	$S \rightarrow^{(2)} bAc$			
A		$A \rightarrow^{(3)} bAc$		$A \rightarrow^{(4)} d$	
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

Példa: abdc

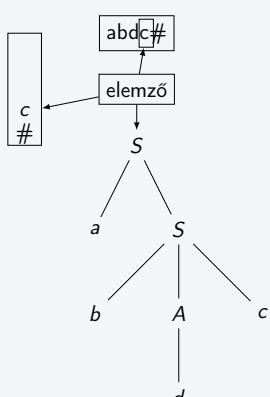


- $(abdc\#, S\#, \epsilon)$
- $(abdc\#, aS\#, 1)$
- $(bdc\#, S\#, 1)$
- $(bdc\#, bAc\#, 12)$
- $(dc\#, Ac\#, 12)$
- $(dc\#, dc\#, 124)$

Példa: abdc

	a	b	c	d	#
S	$S \rightarrow^{(1)} aS$	$S \rightarrow^{(2)} bAc$			
A		$A \rightarrow^{(3)} bAc$		$A \rightarrow^{(4)} d$	
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

Példa: abdc

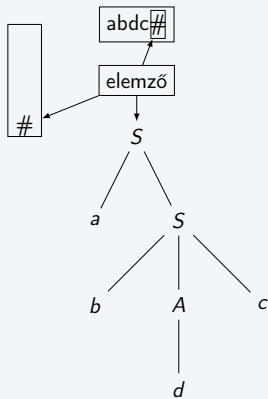


- $(abdc\#, S\#, \epsilon)$
- $(abdc\#, aS\#, 1)$
- $(bdc\#, S\#, 1)$
- $(bdc\#, bAc\#, 12)$
- $(dc\#, Ac\#, 12)$
- $(dc\#, dc\#, 124)$
- $(c\#, c\#, 124)$

Példa: abdc

	a	b	c	d	#
S	$S \rightarrow^{(1)} aS$	$S \rightarrow^{(2)} bAc$			
A		$A \rightarrow^{(3)} bAc$		$A \rightarrow^{(4)} d$	
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

Példa: abdc

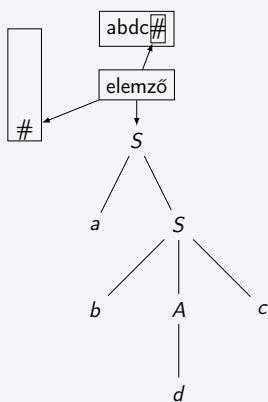


- ($abdc\#, S\#, \epsilon$)
- ($abdc\#, aS\#, 1$)
- ($bdc\#, S\#, 1$)
- ($bdc\#, bAc\#, 12$)
- ($dc\#, Ac\#, 12$)
- ($dc\#, dc\#, 124$)
- ($c\#, c\#, 124$)
- ($\#, \#, 124$)

Példa: abdc

	a	b	c	d	#
S	$S \rightarrow^{(1)} aS$	$S \rightarrow^{(2)} bAc$			
A		$A \rightarrow^{(3)} bAc$		$A \rightarrow^{(4)} d$	
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

Példa: abdc



- ($abdc\#, S\#, \epsilon$)
- ($abdc\#, aS\#, 1$)
- ($bdc\#, S\#, 1$)
- ($bdc\#, bAc\#, 12$)
- ($dc\#, Ac\#, 12$)
- ($dc\#, dc\#, 124$)
- ($c\#, c\#, 124$)
- ($\#, \#, 124$)
- OK

ε-mentes LL(1) nyelvtan

Definíció: ε-mentes LL(1)

Olyan LL(1) grammatika, amely ε-mentes.
(Nincs $A \rightarrow \epsilon$ szabály.)

ε-mentes LL(1) nyelvtan

Definíció: ε-mentes LL(1)

Olyan LL(1) grammatika, amely ε-mentes.
(Nincs $A \rightarrow \epsilon$ szabály.)

Következmény: ε-mentes LL(1) grammatika esetén az egy nemterminálishoz tartozó szabályok jobboldalainak $FIRST_1$ halmazai diszjunktak.

Tétel

Egy grammatika pontosan akkor ε-mentes LL(1), ha

- ε-mentes
- és ha $A \rightarrow \alpha_1$ és $A \rightarrow \alpha_2$ két különböző szabály, akkor $FIRST_1(\alpha_1) \cap FIRST_1(\alpha_2) = \emptyset$.

(A fordítóprogramok könyvben ez szerepel definícióként.)

ε-mentes LL(1) elemzés

- ha a verem tetején terminális szimbólum van:

- ha egyezik a szöveg következő karakterével:
pop és lépés a szövegben
- különben: hiba

ε-mentes LL(1) elemzés

- ha a verem tetején terminális szimbólum van:
 - ha egyezik a szöveg következő karakterével:
pop és lépés a szövegen
 - különben: hiba
- ha a verem tetején nemterminális szimbólum (A) van (és a szövegen a következik):
 - ha van $A \rightarrow \alpha$ szabály, amelyre $a \in FIRST_1(\alpha)$:
 A helyére α és bejegyzés a szintaxisfába
 - különben: hiba

ε-mentes LL(1) elemzés

- ha a verem tetején terminális szimbólum van:
 - ha egyezik a szöveg következő karakterével:
pop és lépés a szövegen
 - különben: hiba
- ha a verem tetején nemterminális szimbólum (A) van (és a szövegen a következik):
 - ha van $A \rightarrow \alpha$ szabály, amelyre $a \in FIRST_1(\alpha)$:
 A helyére α és bejegyzés a szintaxisfába
 - különben: hiba
- ha a verem üres:
 - ha a szöveg végére értünk: OK
 - különben hiba

- ugyanolyan szerkezetű, mint az egyszerű LL(1)-es
- az $A \rightarrow \alpha$ szabályt az A sorába és a $FIRST_1(\alpha)$ elemeinek oszlopába kell beírni

Példa nyelvtan és a FIRST halmazok

$$\begin{array}{l} S \rightarrow aS \mid A \\ A \rightarrow bAc \mid d \end{array}$$

$$\begin{aligned} FIRST_1(aS) &= \{a\} & FIRST_1(A) &= \{b, d\} \\ FIRST_1(bAc) &= \{b\} & FIRST_1(d) &= \{d\} \end{aligned}$$

	a	b	c	d	#
S	$S \xrightarrow{(1)} aS$	$S \xrightarrow{(2)} A$		$S \xrightarrow{(2)} A$	
A		$A \xrightarrow{(3)} bAc$		$A \xrightarrow{(4)} d$	
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

LL(1)

Definíció: LL(1) grammatika

Tetszőleges

$$S \Rightarrow^* wA\beta \Rightarrow w\alpha_1\beta \Rightarrow^* wx$$

$$S \Rightarrow^* wA\beta \Rightarrow w\alpha_2\beta \Rightarrow^* wy$$

levezetéspárra $FIRST_1(x) = FIRST_1(y)$ esetén $\alpha_1 = \alpha_2$.

LL(1)

Definíció: LL(1) grammatika

Tetszőleges

$$S \Rightarrow^* wA\beta \Rightarrow w\alpha_1\beta \Rightarrow^* wx$$

$$S \Rightarrow^* wA\beta \Rightarrow w\alpha_2\beta \Rightarrow^* wy$$

levezetéspárra $FIRST_1(x) = FIRST_1(y)$ esetén $\alpha_1 = \alpha_2$.

Probléma:

 $A \rightarrow \alpha_1 \mid \dots \mid \alpha_n$ szabályban előfordulhat, hogy $\alpha_i \Rightarrow^* \epsilon$ valamelyik α_i -re.Ezért az előreolvasott szimbólum a $S \Rightarrow^* wA\beta$ levezetésesetén a β -ból származhat.Meg kell tehát vizsgálni azt is, hogy milyen terminálisok követhetik az A -t!

FOLLOW halmazok

 $FOLLOW_k(\alpha)$: a levezetésekben az α után előforduló k hosszúságú terminális sorozatok(ha a sorozat hossza kisebb mint k , akkor az egész sorozat eleme $FOLLOW_k(\alpha)$ -nak,
ha α után vége lehet a szövegnek, akkor $\# \in FOLLOW_k(\alpha)$)

FOLLOW halmazok

FOLLOW_k(α): a levezetésekben az α után előforduló k hosszúságú terminális sorozatok

(ha a sorozat hossza kisebb mint k , akkor az egész sorozat eleme FOLLOW_k(α)-nak,
ha α után vége lehet a szövegnek, akkor $\# \in FOLLOW_k(\alpha)$)

Definíció: FOLLOW_k(α)

$$FOLLOW_k(\alpha) = \{x \mid S \Rightarrow^* \beta\alpha\gamma \wedge x \in FIRST_k(\gamma) \setminus \{\epsilon\} \cup \{\#\mid S \Rightarrow^* \beta\alpha\}$$

LL(1) elemzést megalapozó téTEL

TéTEL

Egy grammata pontosan akkor LL(1) grammata, ha bármely $A \rightarrow \alpha_1$ és $A \rightarrow \alpha_2$ különböző szabályok esetén $FIRST_1(\alpha_1 FOLLOW_1(A)) \cap FIRST_1(\alpha_2 FOLLOW_1(A)) = \emptyset$.

FIRST₁($\alpha FOLLOW_1(A)$) jelentése: α -hoz egyenként konkatenáljuk FOLLOW₁(A) elemeit és az így kapott halmaz minden elemére alkalmazzuk a FIRST₁ függvényt.

Példa

Példa nyelvtan és a FIRST, FOLLOW halmazok

$$\begin{aligned} S &\rightarrow aS \mid A \\ A &\rightarrow bAc \mid d \mid \epsilon \end{aligned}$$

$$\begin{aligned} FOLLOW_1(S) &= \{\#\} \\ FOLLOW_1(A) &= \{c, \#\} \end{aligned}$$

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

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

LL(1) elemző táblázat

Mint az egyszerű és az ϵ -mentes LL(1) esetén. Az $A \rightarrow \alpha$ szabályt az A sorába és a $FIRST_1(\alpha FOLLOW_1(A))$ halmaz oszlopáiba kell írni.

	a	b	c	d	#
S	$S \xrightarrow{(1)} aS$	$S \xrightarrow{(2)} A$		$S \xrightarrow{(2)} A$	$S \xrightarrow{(2)} A$
A		$A \xrightarrow{(3)} bAc$	$A \xrightarrow{(5)} \epsilon$	$A \xrightarrow{(4)} d$	$A \xrightarrow{(5)} \epsilon$
a	pop				
b		pop			
c			pop		
d				pop	
#					OK

LL(1) elemzés

- ha a verem tetején terminális szimbólum van:
 - ha egyezik a szöveg következő karakterével: pop és lépés a szövegben
 - különben: hiba

LL(1) elemzés

- ha a verem tetején terminális szimbólum van:
 - ha egyezik a szöveg következő karakterével: pop és lépés a szövegben
 - különben: hiba
- ha a verem tetején nemterminális szimbólum (A) van (és a szövegben a következik):
 - ha van $A \rightarrow \alpha$ szabály, amelyre $a \in FIRST_1(\alpha FOLLOW_1(A))$: A helyére α és bejegyzés a szintaxisfába
 - különben: hiba

LL(1) elemzés

- ha a verem tetején terminális szimbólum van:
 - ha egyezik a szöveg következő karakterével: pop és lépés a szövegben
 - különben: hiba
- ha a verem tetején nemterminális szimbólum (A) van (és a szövegben a következik):
 - ha van $A \rightarrow \alpha$ szabály, amelyre $a \in FIRST_1(\alpha FOLLOW_1(A))$:
A helyére α és bejegyzés a szintaxisfába
 - különben: hiba
- ha a verem üres:
 - ha a szöveg végére értünk: OK
 - különben hiba

Rekurzív leszállás

- az (általános) LL(1) elemzés egy másik implementációja
- minden nemterminálhoz egy eljárást készítünk
- az eljárásokon keresztül a futási idejű verem valósítja meg az elemzés vermét

Az elfogad eljárás

A terminális szimbólumok ellenőrzéséhez:

```
void elfogad( terminalis t )
{
    if( aktualis_szimbolum == t )
        aktualis_szimbolum = lexikalis_elemzo.kovetkezo();
    else
        hiba(...);
}
```

(A lexikális elemző kovetkezo függvénye visszaadja a soron következő lexikális elemet.)

$A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ programja

```
void A()
{
    if( aktualis_szimbolum ∈ FIRST_1(α1 FOLLOW1(A)))
    {
        // alfa_1 programja
    }
    ...
    else if( aktualis_szimbolum ∈ FIRST_1(αn FOLLOW1(A)))
    {
        // alfa_n programja
    }
    else
    {
        hiba(...);
    }
}
```

A szabályalternatívák programja

Az $\alpha = \epsilon$ alternatíva programja a „skip”.

Az $\alpha = X_1 X_2 \dots X_n$ alternatíva programja az X_1, X_2, \dots, X_n szimbólumokhoz tartozó utasítások szekvenciája.

- ha $X_i = a$ (terminális), akkor az X_i -hez tartozó utasítás
`elfogad(a);`
- ha $X_i = B$ (nemterminális), akkor
`B();`

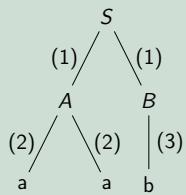
- hiba detektálása esetén:
 - segítő hibajelzést kell adni
 - meg kell próbálni folytatni az elemzést
(az elemzőt szinkronizálni kell a bemenettel)
- a szinkronizáció szimbólumok kihagyását jelenti egy olyan szimbólumig, ahonnan folytatni lehet az elemzést
 - „pánik módszer”: az elemző pánikszerűen menekül a hiba helyétől
- például: a rekurzív leszállás eljárásai elején megvizsgálhatjuk, hogy megfelelő-e az aktualis_szimbolum, és szükség esetén addig ugorjuk át a szimbólumokat a bemeneten, amíg megfelelő nem lesz

LR elemzések (az LR(0) elemzés)

Fordítóprogramok előadás (A,C,T szakirány)

Emlékeztető: elemzési irányok

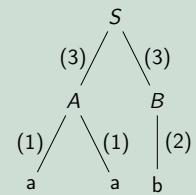
Felülről lefelé



$$S \xrightarrow{(1)} AB \xrightarrow{(2)} aaB \xrightarrow{(3)} aab$$

- Ez egy *legbal* levezetés!

Alulról felfelé



$$aab \xleftarrow{(1)} Ab \xleftarrow{(2)} AB \xleftarrow{(3)} S$$

- Ez egy *legjobb* levezetés inverze!

Emlékeztető: alulról felfelé elemzések

- Az elemzendő szöveg összetartozó részeit helyettesítjük nemterminális szimbólumokkal (redukció) és így alulról, a kezdőszimbólum felé építjük a fát.
- Fő kérdés: „Mit redukálunk?”

Emlékeztető: alulról felfelé elemzési stratégiák

- **visszalépéses keresés (backtrack):** ha nem sikerül eljutni a startszimbólumig, lépjünk vissza, és válasszunk másik redukciót
⇒visszalépéses alulról felfelé elemzések (nem tananyag)
 - lassú
 - ha hibás a szöveg, az csak túl későn derül ki

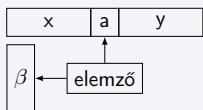
Emlékeztető: alulról felfelé elemzési stratégiák

- **visszalépéses keresés (backtrack):** ha nem sikerül eljutni a startszimbólumig, lépjünk vissza, és válasszunk másik redukciót
⇒visszalépéses alulról felfelé elemzések (nem tananyag)
 - lassú
 - ha hibás a szöveg, az csak túl későn derül ki
- **precedenciák használata:** az egyes szimbólumok között adjunk meg precedenciarelációkat és ennek segítségével határozzuk meg a megfelelő redukciót
⇒precedencia elemzések (nem tananyag)
 - ma már kevessé használt
 - operátorokkal felépített kifejezések esetén természetes a használata

Emlékeztető: alulról felfelé elemzési stratégiák

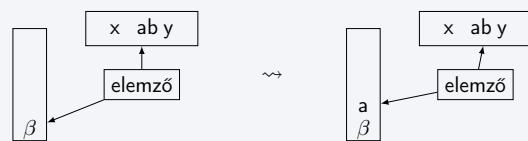
- **visszalépéses keresés (backtrack):** ha nem sikerül eljutni a startszimbólumig, lépjünk vissza, és válasszunk másik redukciót
⇒visszalépéses alulról felfelé elemzések (nem tananyag)
 - lassú
 - ha hibás a szöveg, az csak túl későn derül ki
- **precedenciák használata:** az egyes szimbólumok között adjunk meg precedenciarelációkat és ennek segítségével határozzuk meg a megfelelő redukciót
⇒precedencia elemzések (nem tananyag)
 - ma már kevessé használt
 - operátorokkal felépített kifejezések esetén természetes a használata
- **előreolvasás:** olvassunk előre a szövegen valahány szimbólumot, és az alapján döntsünk a redukcióról
⇒LR elemzések
 - minden programozási nyelvhez lehet (LR) elemzést készíteni
 - majdnem mindenhez lehet gyors (LALR) elemzést készíteni

Az LR elemző felépítése



- xay : bemenet
- β : aktuális mondatforma egy része (veremben tároljuk)
- két lehetséges művelet: léptetés és redukálás
- *LR*: Left to right, using a Rightmost derivation
(Balról jobbra, legjobb vezetéssel)

Léptetés



A bemenet következő szimbólumát a verem tetejére helyezzük.

Redukálás

Az $A \rightarrow \alpha$ szabály szerinti redukció.



A verem tetején lévő szabály-jobboldalt helyettesítjük a megfelelő nemterminális szimbólummal.

Kiegészített grammatika

- Az elemzés végét arról fogjuk felismerni, hogy egy redukció eredménye a kezdőszimbólum lett.
- Ez csak akkor lehet, ha a kezdőszimbólum nem fordul elő a szabályok jobboldalán.

Kiegészített grammatika

- Az elemzés végét arról fogjuk felismerni, hogy egy redukció eredménye a kezdőszimbólum lett.
- Ez csak akkor lehet, ha a kezdőszimbólum nem fordul elő a szabályok jobboldalán.
- Ezt nem minden grammatika teljesíti, de mindegyik kiegészíthető:
 - legyen S' az új kezdőszimbólum
 - legyen $S' \rightarrow S$ egy új szabály
- az *LR* elemzésekhez minden kiegészített nyelvtanokat fogunk használni

Mit kell redukálni?

Mit kell redukálni?

- **Egyszerű részmondat** (emlékeztető): α a $\gamma\alpha\beta$ mondatforma egyszerű részmondata, ha $S \Rightarrow^* \gamma A \beta \Rightarrow \gamma\alpha\beta$.

Mit kell redukálni?

- **Egyszerű részmondat** (emlékeztető): α a $\gamma\alpha\beta$ mondatforma egyszerű részmondata, ha $S \Rightarrow^* \gamma A \beta \Rightarrow \gamma\alpha\beta$.
- **Nyél:** a mondatformában a legbaloldalibb egyszerű részmondat.

Mit kell redukálni?

- **Egyszerű részmondat** (emlékeztető): α a $\gamma\alpha\beta$ mondatforma egyszerű részmondata, ha $S \Rightarrow^* \gamma A \beta \Rightarrow \gamma\alpha\beta$.
- **Nyél:** a mondatformában a legbaloldalibb egyszerű részmondat.
- **Épp a nyelet kell megtalálni a redukcióhoz.**

A nyél meghatározása

- **Probléma: „Mi a nyél?”**
 - léptetni vagy redukálni kell?
 - ha több lehetőség is van, melyik szabály szerint kell redukálni?

A nyél meghatározása

- **Probléma: „Mi a nyél?”**
 - léptetni vagy redukálni kell?
 - ha több lehetőség is van, melyik szabály szerint kell redukálni?
- **$LR(k)$ grammatika:** k szimbólum előreolvasásával eldönthető, hogy mi legyen az elemzés következő lépése.

Magyarázat az $LR(k)$ definíciójához

- Tegyük fel, hogy léptetésekkel és redukálásokkal eljutottunk az $\alpha\beta w$ mondatformához, és itt β a nyél: $S \Rightarrow^* \alpha A w \Rightarrow \alpha\beta w$.

Magyarázat az $LR(k)$ definíciójához

- Tegyük fel, hogy léptetésekkel és redukálásokkal eljutottunk az $\alpha\beta w$ mondatformához, és itt β a nyél: $S \Rightarrow^* \alpha Aw \Rightarrow \alpha\beta w$.
- Tegyük fel, hogy egy ugyanígy kezdődő mondatforma, az $\alpha\beta y$ felbontható $\alpha\beta y = \gamma\delta x$ módon, és ebben δ a nyél, azaz $S \Rightarrow^* \gamma Bx \Rightarrow \gamma\delta x$.

Magyarázat az $LR(k)$ definíciójához

- Tegyük fel, hogy léptetésekkel és redukálásokkal eljutottunk az $\alpha\beta w$ mondatformához, és itt β a nyél: $S \Rightarrow^* \alpha Aw \Rightarrow \alpha\beta w$.
- Tegyük fel, hogy egy ugyanígy kezdődő mondatforma, az $\alpha\beta y$ felbontható $\alpha\beta y = \gamma\delta x$ módon, és ebben δ a nyél, azaz $S \Rightarrow^* \gamma Bx \Rightarrow \gamma\delta x$.
- Az $LR(k)$ tulajdonság azt mondja, hogy w -ból és y -ból előreolvasva k szimbólumot, egyértelműen előntható az elemzés következő lépése.

Magyarázat az $LR(k)$ definíciójához

- Tegyük fel, hogy léptetésekkel és redukálásokkal eljutottunk az $\alpha\beta w$ mondatformához, és itt β a nyél: $S \Rightarrow^* \alpha Aw \Rightarrow \alpha\beta w$.
- Tegyük fel, hogy egy ugyanígy kezdődő mondatforma, az $\alpha\beta y$ felbontható $\alpha\beta y = \gamma\delta x$ módon, és ebben δ a nyél, azaz $S \Rightarrow^* \gamma Bx \Rightarrow \gamma\delta x$.
- Az $LR(k)$ tulajdonság azt mondja, hogy w -ból és y -ból előreolvasva k szimbólumot, egyértelműen előntható az elemzés következő lépése.
- Ezért ha $FIRST_k(w) = FIRST_k(y)$, akkor $\alpha\beta w$ és $\alpha\beta y$ esetén is ugyanazt kell csinálni:
 - mivel $\alpha\beta w$ esetén az $A \rightarrow \beta$ szabály szerint redukáltunk,
 - ugyanezt kellett csinálni $\alpha\beta y$ esetén is,
 - vagyis $\alpha = \gamma$, $\beta = \delta$, $A = B$ és $y = x$.

 $LR(k)$ definíciójaDefiníció: $LR(k)$ grammatika

Egy kiegészített grammata $LR(k)$ grammata ($k \geq 0$), ha

$$S \Rightarrow^* \alpha Aw \Rightarrow \alpha\beta w$$

$$S \Rightarrow^* \gamma Bx \Rightarrow \gamma\delta x$$

$\alpha\beta y = \gamma\delta x$ és $FIRST_k(w) = FIRST_k(y)$ esetén

$\alpha = \gamma$, $\beta = \delta$ és $A = B$.

 $LR(k)$ definíciójaDefiníció: $LR(k)$ grammatika

Egy kiegészített grammata $LR(k)$ grammata ($k \geq 0$), ha

$$S \Rightarrow^* \alpha Aw \Rightarrow \alpha\beta w$$

$$S \Rightarrow^* \gamma Bx \Rightarrow \gamma\delta x$$

$\alpha\beta y = \gamma\delta x$ és $FIRST_k(w) = FIRST_k(y)$ esetén

$\alpha = \gamma$, $\beta = \delta$ és $A = B$.

LR(0) grammata: előreolvasás nélkül előntható az elemzés következő lépése.

Példa

Grammatika

- 1 $S' \rightarrow S$
- 2 $S \rightarrow aAd$
- 3 $A \rightarrow bA$
- 4 $A \rightarrow c$

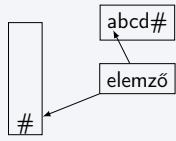
$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abc$$

Példa

Grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$

$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$$



Példa

Grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$

$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$$

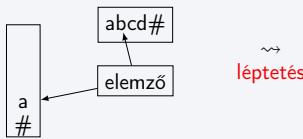


Példa

Grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$

$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$$



Példa

Grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$

$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$$

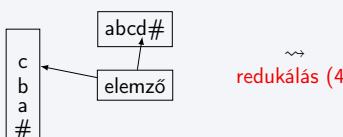


Példa

Grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$

$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$$

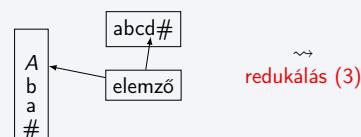


Példa

Grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$

$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$$



Példa

Grammatika

- ① $S' \rightarrow S$
- ② $S \rightarrow aAd$
- ③ $A \rightarrow bA$
- ④ $A \rightarrow c$

$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$$

léptetés



Példa

Grammatika

- ① $S' \rightarrow S$
- ② $S \rightarrow aAd$
- ③ $A \rightarrow bA$
- ④ $A \rightarrow c$

$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$$

redukálás (2)



Példa

Grammatika

- ① $S' \rightarrow S$
- ② $S \rightarrow aAd$
- ③ $A \rightarrow bA$
- ④ $A \rightarrow c$

$$S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$$

elfogadás

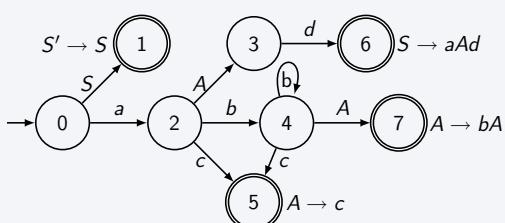


Léptetés vagy redukálás?

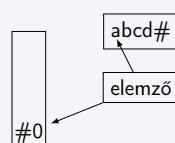
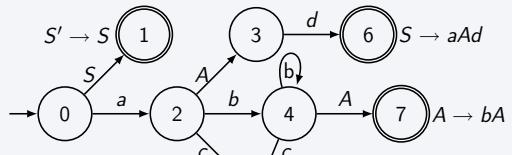
- LR(0) elemzés: előreolvasás nélkül kell döntenünk!

Léptetés vagy redukálás?

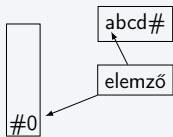
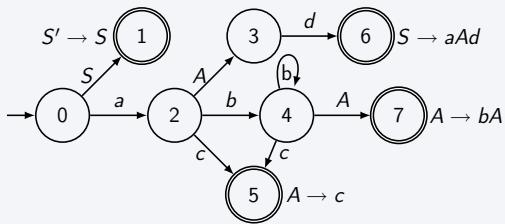
- LR(0) elemzés: előreolvasás nélkül kell döntenünk!
- Ötlet: készítsünk egy véges determinisztikus automatát
 - az átmeneteit a verembe kerülő szimbólumok határozzák meg
 - léptetéskor terminális
 - redukáláskor nemterminális
 - amikor elfogadó állapotba jut, akkor kell redukálni



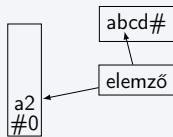
Példa



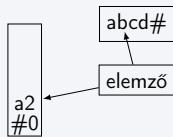
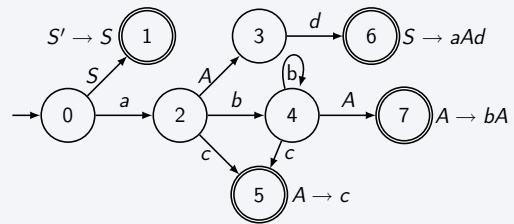
Példa



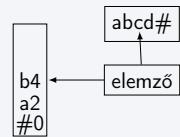
~~> léptetés



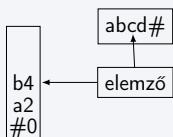
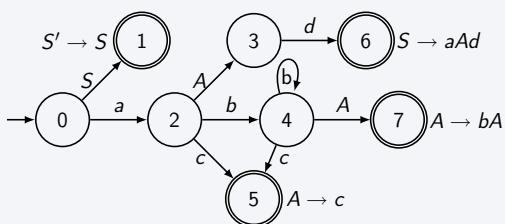
Példa



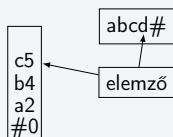
~~> léptetés



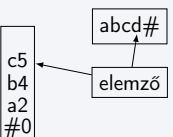
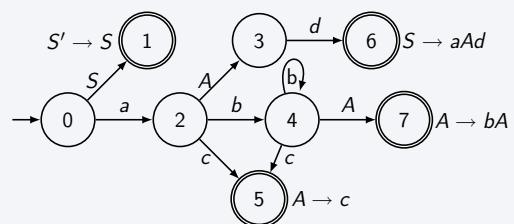
Példa



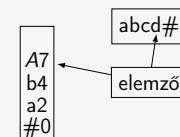
~~> léptetés



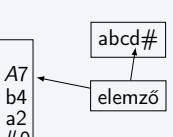
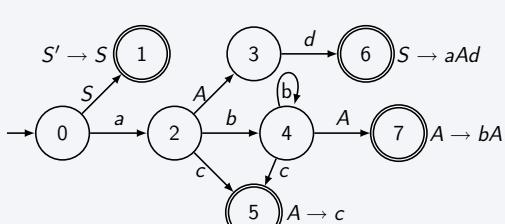
Példa



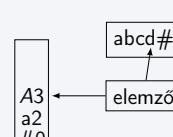
~~> redukálás



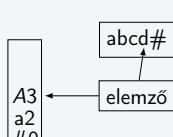
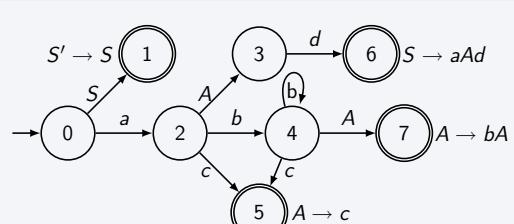
Példa



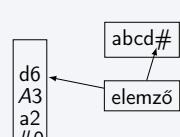
~~> redukálás



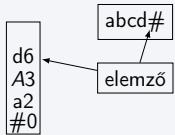
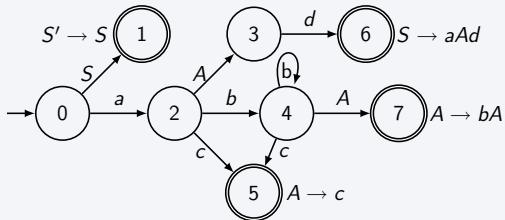
Példa



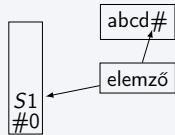
~~> léptetés



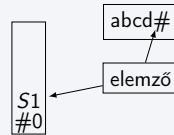
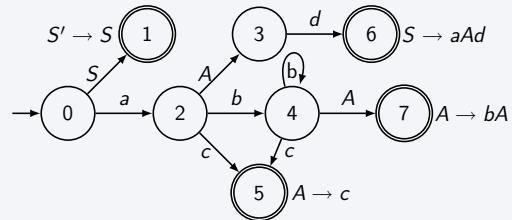
Példa



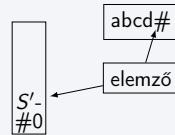
redukálás



Példa



elfogadás



LR(0) elemek

- Hogyan építsük fel az automatát?

LR(0) elemek

- Hogyan építsük fel az automatát?
- Az automata állapotai azt mutatják meg, hogy melyik szabály építésében hol tartunk.
- Például: $[S \rightarrow a.Ad]$ jelentse azt, hogy
 - az a szimbólumot már elemeztük,
 - az Ad rész még hátra van.

LR(0) elemek

- Hogyan építsük fel az automatát?
- Az automata állapotai azt mutatják meg, hogy melyik szabály építésében hol tartunk.
- Például: $[S \rightarrow a.Ad]$ jelentse azt, hogy
 - az a szimbólumot már elemeztük,
 - az Ad rész még hátra van.

Definíció: LR(0) elem

Ha $A \rightarrow \alpha$ a grammatika egy helyettesítési szabálya, akkor az $\alpha = \alpha_1\alpha_2$ tetszőleges felbontás esetén $[A \rightarrow \alpha_1.\alpha_2]$ a grammatika egy LR(0) eleme.

LR(0) elemek

- Hogyan építsük fel az automatát?
- Az automata állapotai azt mutatják meg, hogy melyik szabály építésében hol tartunk.
- Például: $[S \rightarrow a.Ad]$ jelentse azt, hogy
 - az a szimbólumot már elemeztük,
 - az Ad rész még hátra van.

Definíció: LR(0) elem

Ha $A \rightarrow \alpha$ a grammatika egy helyettesítési szabálya, akkor az $\alpha = \alpha_1\alpha_2$ tetszőleges felbontás esetén $[A \rightarrow \alpha_1.\alpha_2]$ a grammatika egy LR(0) eleme.

- Ha a szabály jobboldala n szimbólumot tartalmaz, akkor $n + 1$ darab LR(0)-elem tartozik hozzá.

$A \rightarrow .aAd, A \rightarrow a.Ad, A \rightarrow aA.d, A \rightarrow aAd.$

A lezárás művelet

- Az automata egy állapotához több $LR(0)$ -elem is tartozhat.
 - Ezeket a halmazokat fogjuk **kanonikus halmazoknak** hívni.

A lezárás művelet

- Az automata egy állapotához több $LR(0)$ -elem is tartozhat.
 - Ezeket a halmazokat fogjuk **kanonikus halmazoknak** hívni.
- Milyen $LR(0)$ elemek tartoznak egy halmazba?
 - Például: Ha $[A \rightarrow a.Ad]$ állapotban vagyunk, akkor az $A \rightarrow bA$ és $A \rightarrow c$ szabályokat kezdhetjük építeni, azaz $[A \rightarrow .bA]$ és $[A \rightarrow .c]$ is hozzátarozik a halmazhoz.

A lezárás művelet

- Az automata egy állapotához több $LR(0)$ -elem is tartozhat.
 - Ezeket a halmazokat fogjuk **kanonikus halmazoknak** hívni.
- Milyen $LR(0)$ elemek tartoznak egy halmazba?
 - Például: Ha $[A \rightarrow a.Ad]$ állapotban vagyunk, akkor az $A \rightarrow bA$ és $A \rightarrow c$ szabályokat kezdhetjük építeni, azaz $[A \rightarrow .bA]$ és $[A \rightarrow .c]$ is hozzátarozik a halmazhoz.

Definíció: lezárás (closure)

Ha \mathcal{I} a grammatika egy $LR(0)$ elemhalmaza, akkor $closure(\mathcal{I})$ a legszűkebb olyan halmaz, amely az alábbi tulajdonságokkal rendelkezik:

- $\mathcal{I} \subseteq closure(\mathcal{I})$
- ha $[A \rightarrow \alpha.B\gamma] \in closure(\mathcal{I})$, és $B \rightarrow \beta$ a grammatika egy szabálya, akkor $[B \rightarrow .\beta] \in closure(\mathcal{I})$.

A olvasás művelet

- Hogyan lépünk át az automata egyik állapotából a másikba?

Az olvasás művelet

- Hogyan lépünk át az automata egyik állapotából a másikba?
- Ha $[A \rightarrow a.Ad]$ állapotban vagyunk, és A kerül a verem tetejére, akkor $[A \rightarrow aA.d]$ állapotba jutunk.

Az olvasás művelet

- Hogyan lépünk át az automata egyik állapotából a másikba?
- Ha $[A \rightarrow a.Ad]$ állapotban vagyunk, és A kerül a verem tetejére, akkor $[A \rightarrow aA.d]$ állapotba jutunk.

Az olvasás művelet

Definíció: olvasás (read)

Ha \mathcal{I} a grammatika egy $LR(0)$ elemhalmaza, X pedig terminális vagy nemterminális szimbóluma, akkor $read(\mathcal{I}, X)$ a legszűkebb olyan halmaz, amely az alábbi tulajdonsággal rendelkezik:

- ha $[A \rightarrow \alpha.X\beta] \in \mathcal{I}$, akkor
 $closure([A \rightarrow \alpha.X\beta]) \subseteq read(\mathcal{I}, X)$.

LR(0) kanonikus halmazok

Definíció: LR(0) kanonikus halmazok

- ❶ $\text{closure}([S' \rightarrow .S])$ a grammatika egy kanonikus halmaza.
- ❷ Ha \mathcal{I} a grammatika egy kanonikus elemhalmaza, X egy terminális vagy nemterminális szimbóluma, és $\text{read}(\mathcal{I}, X)$ nem üres, akkor $\text{read}(\mathcal{I}, X)$ is a grammatika egy kanonikus halmaza.
- ❸ Az első két szabálytal az összes kanonikus halmaz előáll.

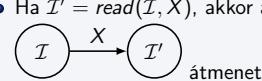
LR(0) kanonikus halmazok

Definíció: LR(0) kanonikus halmazok

- ❶ $\text{closure}([S' \rightarrow .S])$ a grammatika egy kanonikus halmaza.
- ❷ Ha \mathcal{I} a grammatika egy kanonikus elemhalmaza, X egy terminális vagy nemterminális szimbóluma, és $\text{read}(\mathcal{I}, X)$ nem üres, akkor $\text{read}(\mathcal{I}, X)$ is a grammatika egy kanonikus halmaza.
- ❸ Az első két szabálytal az összes kanonikus halmaz előáll.

Az automata felépítése:

- A $\text{closure}([S' \rightarrow .S])$ legyen a kezdőállapot.
- Ha $\mathcal{I}' = \text{read}(\mathcal{I}, X)$, akkor az automatában legyen



átmenet.

- A végállapotok azok a kanonikus halmazok, amelyekben olyan elemek vannak, ahol a pont a szabály végén van.

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$

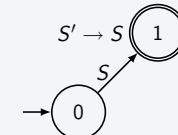


$$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .aAd]\}$$

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$



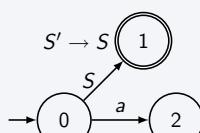
$$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .aAd]\}$$

$$\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \text{closure}([S' \rightarrow S.]) = \{[S' \rightarrow S.]\}$$

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$



$$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .aAd]\}$$

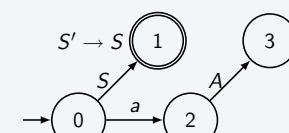
$$\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \{[S' \rightarrow S.]\}$$

$$\begin{aligned} \mathcal{I}_2 &= \text{read}(\mathcal{I}_0, a) = \text{closure}([S \rightarrow a.Ad]) \\ &= \{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\} \end{aligned}$$

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$



$$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .aAd]\}$$

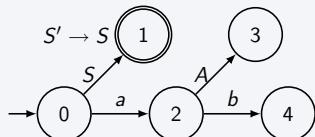
$$\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \{[S' \rightarrow S.]\}$$

$$\begin{aligned} \mathcal{I}_2 &= \text{read}(\mathcal{I}_0, a) = \{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\} \\ \mathcal{I}_3 &= \text{read}(\mathcal{I}_2, A) = \text{closure}([S \rightarrow aA.d]) = \{[S \rightarrow aA.d]\} \end{aligned}$$

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$



$$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .aAd]\}$$

$$\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \{[S' \rightarrow S.]\}$$

$$\mathcal{I}_2 = \text{read}(\mathcal{I}_0, a) = \{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\}$$

$$\mathcal{I}_3 = \text{read}(\mathcal{I}_2, A) = \{[S \rightarrow a.A.d]\}$$

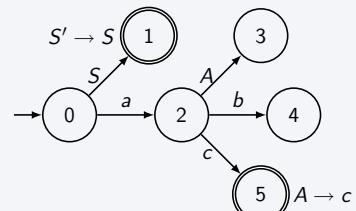
$$\mathcal{I}_4 = \text{read}(\mathcal{I}_2, b) = \text{closure}([A \rightarrow b.A])$$

$$= \{[A \rightarrow b.A], [A \rightarrow .bA], [A \rightarrow .c]\}$$

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$



$$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .aAd]\}$$

$$\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \{[S' \rightarrow S.]\}$$

$$\mathcal{I}_2 = \text{read}(\mathcal{I}_0, a) = \{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\}$$

$$\mathcal{I}_3 = \text{read}(\mathcal{I}_2, A) = \{[S \rightarrow a.A.d]\}$$

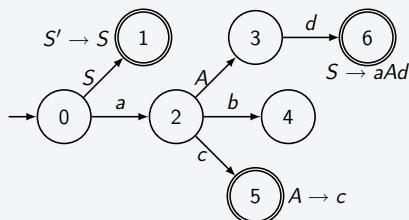
$$\mathcal{I}_4 = \text{read}(\mathcal{I}_2, b) = \{[A \rightarrow b.A], [A \rightarrow .bA], [A \rightarrow .c]\}$$

$$\mathcal{I}_5 = \text{read}(\mathcal{I}_2, c) = \text{closure}([A \rightarrow c.]) = \{[A \rightarrow c.\]\}$$

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$



$$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .aAd]\}$$

$$\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \{[S' \rightarrow S.]\}$$

$$\mathcal{I}_2 = \text{read}(\mathcal{I}_0, a) = \{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\}$$

$$\mathcal{I}_3 = \text{read}(\mathcal{I}_2, A) = \{[S \rightarrow a.A.d]\}$$

$$\mathcal{I}_4 = \text{read}(\mathcal{I}_2, b) = \{[A \rightarrow b.A], [A \rightarrow .bA], [A \rightarrow .c]\}$$

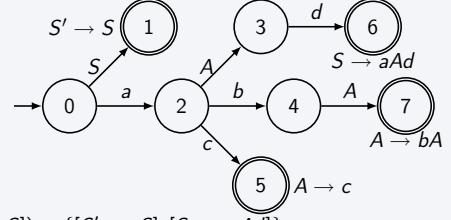
$$\mathcal{I}_5 = \text{read}(\mathcal{I}_2, c) = \{[A \rightarrow c.\]\}$$

$$\mathcal{I}_6 = \text{read}(\mathcal{I}_3, d) = \text{closure}([S \rightarrow aAd.]) = \{[S \rightarrow aAd.\]\}$$

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$



$$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .aAd]\}$$

$$\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \{[S' \rightarrow S.]\}$$

$$\mathcal{I}_2 = \text{read}(\mathcal{I}_0, a) = \{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\}$$

$$\mathcal{I}_3 = \text{read}(\mathcal{I}_2, A) = \{[S \rightarrow a.A.d]\}$$

$$\mathcal{I}_4 = \text{read}(\mathcal{I}_2, b) = \{[A \rightarrow b.A], [A \rightarrow .bA], [A \rightarrow .c]\}$$

$$\mathcal{I}_5 = \text{read}(\mathcal{I}_2, c) = \{[A \rightarrow c.\]\}$$

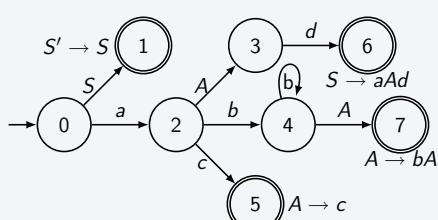
$$\mathcal{I}_6 = \text{read}(\mathcal{I}_3, d) = \{[S \rightarrow aAd.\]\}$$

$$\mathcal{I}_7 = \text{read}(\mathcal{I}_4, A) = \text{closure}([A \rightarrow bA.]) = \{[A \rightarrow bA.\]\}$$

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$



$$\mathcal{I}_4 = \text{read}(\mathcal{I}_2, b) = \{[A \rightarrow b.A], [A \rightarrow .bA], [A \rightarrow .c]\}$$

$$\mathcal{I}_5 = \text{read}(\mathcal{I}_2, c) = \{[A \rightarrow c.\]\}$$

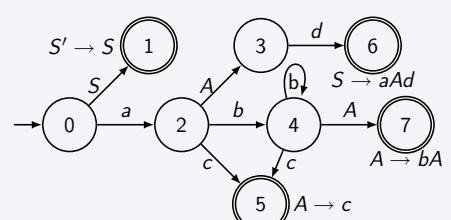
$$\dots \text{read}(\mathcal{I}_4, b) = \text{closure}([A \rightarrow b.A])$$

$$= \{[A \rightarrow b.A], [A \rightarrow .bA], [A \rightarrow .c]\} = \mathcal{I}_4$$

Példa

Példa grammatika

- ❶ $S' \rightarrow S$
- ❷ $S \rightarrow aAd$
- ❸ $A \rightarrow bA$
- ❹ $A \rightarrow c$



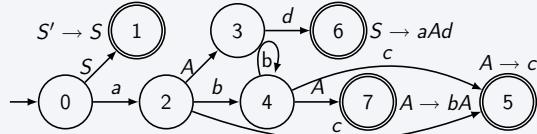
$$\mathcal{I}_4 = \text{read}(\mathcal{I}_2, b) = \{[A \rightarrow b.A], [A \rightarrow .bA], [A \rightarrow .c]\}$$

$$\mathcal{I}_5 = \text{read}(\mathcal{I}_2, c) = \{[A \rightarrow c.\]\}$$

$$\dots \text{read}(\mathcal{I}_4, b) = \mathcal{I}_4$$

$$\text{read}(\mathcal{I}_4, c) = \text{closure}([A \rightarrow c.\]) = \{[A \rightarrow c.\]\} = \mathcal{I}_5$$

LR(0) elemző táblázat



állapot	akció	S	A	a	b	c	d
0	léptetés	1		2			
1	OK						
2	léptetés	3		4	5		
3	léptetés					6	
4	léptetés	7		4	5		
5	redukálás ($A \rightarrow c$)						
6	redukálás ($S \rightarrow aAd$)						
7	redukálás ($A \rightarrow bA$)						

Véges-e az elemző létrehozása?

- a grammatikának véges sok szabálya van
- véges sok LR(0) eleme van

Véges-e az elemző létrehozása?

- a grammatikának véges sok szabálya van
- véges sok LR(0) eleme van
- a closure függvény kiszámítása véges sok lépében befejeződik
- a read függvény kiszámítása véges sok lépében befejeződik

Véges-e az elemző létrehozása?

- a grammatikának véges sok szabálya van
- véges sok LR(0) eleme van
- a closure függvény kiszámítása véges sok lépében befejeződik
- a read függvény kiszámítása véges sok lépében befejeződik
- a véges sok LR(0)-elem hatványhalmaza is véges
- a lehetséges kanonikus halmazok száma is véges

Véges-e az elemző létrehozása?

- a grammatikának véges sok szabálya van
- véges sok LR(0) eleme van
- a closure függvény kiszámítása véges sok lépében befejeződik
- a read függvény kiszámítása véges sok lépében befejeződik
- a véges sok LR(0)-elem hatványhalmaza is véges
- a lehetséges kanonikus halmazok száma is véges

Az elemző táblázat (az automata) létrehozása véges sok lépében befejeződik.

Helyes-e az elemző?

- Az automata pontosan akkor jut-e végállapotba, ha redukálni kell?
- A megfelelő a redukciót írja-e elő?

Járható prefix

Definíció: járható prefix

Ha az $\alpha\beta x$ mondatforma nyele β , akkor az $\alpha\beta$ prefixeit az $\alpha\beta x$ járható prefixeinek nevezzük.

Járható prefix

Definíció: járható prefix

Ha az $\alpha\beta x$ mondatforma nyele β , akkor az $\alpha\beta$ prefixeit az $\alpha\beta x$ járható prefixeinek nevezzük.

- A járható prefixeket olvassuk végig a nyél végének eléréséhez.
- Példa: $S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$
Az $abAd$ mondatforma nyele a bA .
A járható prefixei: a , ab , abA

Járható prefix

Definíció: járható prefix

Ha az $\alpha\beta x$ mondatforma nyele β , akkor az $\alpha\beta$ prefixeit az $\alpha\beta x$ járható prefixeinek nevezzük.

- A járható prefixeket olvassuk végig a nyél végének eléréséhez.
- Példa: $S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow abcd$
Az $abAd$ mondatforma nyele a bA .
A járható prefixei: a , ab , abA
- Maximális járható prefix:** olyan járható prefix, amihez nem lehet újabb szimbólumot hozzávenni, hogy járható prefixet kapunk.
- Egy járható prefix épp akkor maximális, ha a végén van a nyél.

Járható prefixre érvényes $LR(0)$ elemek

Definíció: járható prefixre érvényes $LR(0)$ elemek

A grammatika egy $[A \rightarrow \alpha.\beta]$ $LR(0)$ -eleme érvényes a $\gamma\alpha$ járható prefixre nézve, ha

$$S' \Rightarrow^* \gamma A x \Rightarrow \gamma \alpha \beta x$$

Járható prefixre érvényes $LR(0)$ elemek

Definíció: járható prefixre érvényes $LR(0)$ elemek

A grammatika egy $[A \rightarrow \alpha.\beta]$ $LR(0)$ -eleme érvényes a $\gamma\alpha$ járható prefixre nézve, ha

$$S' \Rightarrow^* \gamma A x \Rightarrow \gamma \alpha \beta x$$

- A érvényes $LR(0)$ elemek az adott járható prefix „lehetséges folytatásait” adják meg.
- Példa: az ab járható prefixre érvényes $LR(0)$ -elemek: $[A \rightarrow b.A]$, $[A \rightarrow .bA]$, $[A \rightarrow .c]$.
 $S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd$
 $S' \Rightarrow S \Rightarrow aAd \Rightarrow abAd \Rightarrow ab\underline{b}Ad$
 $S' \Rightarrow S \Rightarrow aAd \Rightarrow abc\underline{d}$
- A maximális járható prefixekre érvényes $LR(0)$ elemek azok, ahol a pont a szabály végén van.

Az $LR(0)$ elemzés helyessége

Tétel: az $LR(0)$ elemzés nagy tétele

Egy γ járható prefix hatására az elemző automatája a kezdőállapotból olyan állapotba kerül, amelyhez tartozó kanonikus halma az éppen a γ járható prefixre érvényes $LR(0)$ elemeket tartalmazza.

Az $LR(0)$ elemzés helyessége

Tétel: az $LR(0)$ elemzés nagy tétele

Egy γ járható prefix hatására az elemző automatája a kezdőállapotból olyan állapotba kerül, amelyhez tartozó kanonikus halmaz éppen a γ járható prefixre érvényes $LR(0)$ elemeket tartalmazza.

- Az elemző tehát addig fog léptetést előírni, amíg a veremben lévő járható prefix maximális nem lesz.
- Ha a járható prefix maximális, akkor az elemző redukálást ír elő.

Konfliktusok a táblázatban

- Hol használtuk ki, hogy $LR(0)$ a grammaтика?

Konfliktusok a táblázatban

- Hol használtuk ki, hogy $LR(0)$ a grammaтика?
- **Konfliktus:** Ha egy \mathcal{I}_k kanonikus halmaz alapján nem lehet egyértelműen eldönteni, hogy az adott állapotban milyen akciót kell végrehajtani.
 - léptetés/redukálás konfliktus: az egyik elem léptetést, egy másik redukálást ír elő
 - redukálás/redukálás konfliktus: az egyik elem az egyik szabály szerinti, a másik egy másik szabály szerinti redukciót ír elő

Konfliktusok a táblázatban

- Hol használtuk ki, hogy $LR(0)$ a grammaтика?
- **Konfliktus:** Ha egy \mathcal{I}_k kanonikus halmaz alapján nem lehet egyértelműen eldönteni, hogy az adott állapotban milyen akciót kell végrehajtani.
 - léptetés/redukálás konfliktus: az egyik elem léptetést, egy másik redukálást ír elő
 - redukálás/redukálás konfliktus: az egyik elem az egyik szabály szerinti, a másik egy másik szabály szerinti redukciót ír elő
- **Az $LR(0)$ tulajdonság biztosítja a táblázat konfliktusmentes kitöltését!**

LR elemzések (SLR(1) és LR(1) elemzések)

Fordítóprogramok előadás (A,C,T szakirány)

Emlékeztető

Emlékeztető: $LR(0)$ elemzés

- A lexikális elemző által előállított szimbólumsorozatot balról jobbra olvassuk, a szimbólumokat az elemző vermébe tesszük.

Emlékeztető

Emlékeztető: $LR(0)$ elemzés

- A lexikális elemző által előállított szimbólumsorozatot balról jobbra olvassuk, a szimbólumokat az elemző vermébe tesszük.
- Léptetés: egy új szimbólumot teszünk a bemenetről a verem tetejére.
- Redukálás: a verem tetején lévő szabály-jobboldalt helyettesítjük a szabály bal oldalán álló nemterminálissal.

Emlékeztető

Emlékeztető: $LR(0)$ elemzés

- A lexikális elemző által előállított szimbólumsorozatot balról jobbra olvassuk, a szimbólumokat az elemző vermébe tesszük.
- Léptetés: egy új szimbólumot teszünk a bemenetről a verem tetejére.
- Redukálás: a verem tetején lévő szabály-jobboldalt helyettesítjük a szabály bal oldalán álló nemterminálissal.
- $LR(0)$: az alkalmazandó műveletről előreolvasás nélkül döntünk.
- A háttérben egy véges determinisztikus automata működik:
 - az automata átmeneteit a verem tetejére kerülő szimbólumok határozzák meg
 - ha az automata végállapotba jut, redukálni kell
 - egyéb állapotban pedig léptetni

Emlékeztető

Emlékeztető: $LR(0)$ elemzés

- Az automata állapotai a *kanonikus halmazok*.
 - „Melyik szabály építésében hol tartunk éppen?”
 - elemei az $LR(0)$ -elemek

Kanonikus halmaz és jelentése

A $\{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\}$ kanonikus halmaz jelentése:

„Az adott állapotban az $S \rightarrow a.Ad$, $A \rightarrow bA$ és $A \rightarrow c$ szabályok jobboldalait építhetjük. A $S \rightarrow a.Ad$ szabályból az a szimbólumot már elemezük, az Ad rész még hátra van. A másik két szabály építése most kezdődhet.”

Emlékeztető

Emlékeztető: $LR(0)$ elemzés

- Lezáras (closure)* művelet: segítségével adhatók meg az egy kanonikus halmazba tartozó $LR(0)$ -elemek.

Lezáras

$$\text{closure}([S \rightarrow a.Ad]) = \{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\}$$

Emlékeztető: LR(0) elemzés

- Lezáras (closure) művelet: segítségével adhatók meg az egy kanonikus halmazba tartozó LR(0)-elemek.

Lezáras

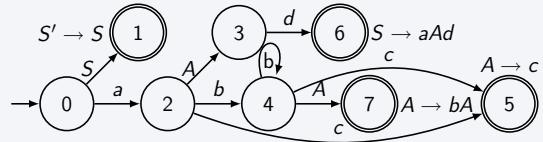
$$\text{closure}([S \rightarrow a.Ad]) = \{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\}$$

- Olvasás (read) művelet: megadja, hogy egy kanonikus halmazból egy adott szimbólum olvasásával melyik kanonikus halmazba jutunk. Ezek lesznek az automata átmenetei.

Olvasás

$$\begin{aligned} \text{read}(\{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\}, b) &= \\ &= \text{closure}([A \rightarrow b.A]) = \\ &= \{[A \rightarrow b.A], [A \rightarrow .bA], [A \rightarrow .c]\} \end{aligned}$$

Emlékeztető: LR(0) elemzés



...

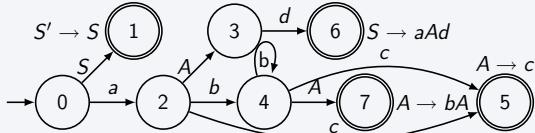
$$I_2 = \{[S \rightarrow a.Ad], [A \rightarrow .bA], [A \rightarrow .c]\}$$

...

$$\begin{aligned} I_4 &= \text{read}(I_2, b) = \{[A \rightarrow b.A], [A \rightarrow .bA], [A \rightarrow .c]\} \\ I_5 &= \text{read}(I_4, c) = \text{read}(I_4, c) = \{[A \rightarrow c]\} \end{aligned}$$

Elfogadó állapot: „a hozzá tartozó elemeknek a végén van a pont”

Emlékeztető: LR(0) elemzés



állapot	akció	S	A	a	b	c	d
0	léptetés	1		2			
1	OK						
2	léptetés		3	4	5		
3	léptetés					6	
4	léptetés		7	4	5		
5	redukálás ($A \rightarrow c$)						
6	redukálás ($S \rightarrow aAd$)						
7	redukálás ($A \rightarrow bA$)						

Konfliktusok

- Az LR(0) tulajdonság biztosította, hogy a táblázat egy cellájába sem kerül két különböző műveletet, azaz a táblázat konfliktusmentes.
- Mi történik, ha nem LR(0) a grammatika?

Konfliktusok

- Az LR(0) tulajdonság biztosította, hogy a táblázat egy cellájába sem kerül két különböző műveletet, azaz a táblázat konfliktusmentes.
- Mi történik, ha nem LR(0) a grammatika?

A helyes zárójelezés

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow \epsilon \mid (S)S \end{aligned}$$

Konfliktusok

- Az LR(0) tulajdonság biztosította, hogy a táblázat egy cellájába sem kerül két különböző műveletet, azaz a táblázat konfliktusmentes.
- Mi történik, ha nem LR(0) a grammatika?

A helyes zárójelezés

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow \epsilon \mid (S)S \\ S' &\Rightarrow S \Rightarrow (S)S \Rightarrow ((S)) \Rightarrow (\textcolor{red}{}) \\ S' &\Rightarrow S \Rightarrow (S)S \Rightarrow (S) \Rightarrow ((S)S) \Rightarrow ((\textcolor{red}{})) \Rightarrow (\textcolor{red}{}) \end{aligned}$$

Konfliktusok

- Az LR(0) tulajdonság biztosította, hogy a táblázat egy cellájába sem kerül két különböző műveletet, azaz a táblázat *konfliktusmentes*.
- Mi történik, ha nem LR(0) a grammatika?

A helyes zárójelezés

 $S' \rightarrow S$
 $S \rightarrow \epsilon \mid (S)S$
 $S' \Rightarrow S \Rightarrow (S)S \Rightarrow (S) \Rightarrow ()$
 $S' \Rightarrow S \Rightarrow (S)S \Rightarrow (S) \Rightarrow ((S)S) \Rightarrow ((S)) \Rightarrow ((())$

Az piros részek elolvasása után:

- az első esetben $S \rightarrow \epsilon$ szerinti redukciót kell végrahajtani,
- a második esetben léptetni kell.

Előreolvasás nélkül nem tudunk dönteneni, nem LR(0) grammatika.

Példa: helyes zárójelezés

Példa grammatika

- ① $S' \rightarrow S$
 ② $S \rightarrow \epsilon \mid (S)S$



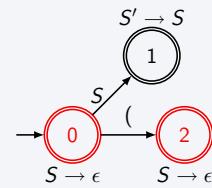
$\mathcal{I}_0 = closure([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

Példa: helyes zárójelezés

Példa: helyes zárójelezés

Példa grammatika

- ① $S' \rightarrow S$
 ② $S \rightarrow \epsilon \mid (S)S$



$\mathcal{I}_0 = closure([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S.] \}$

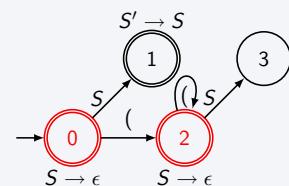
$\mathcal{I}_2 = read(\mathcal{I}_0, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

Példa: helyes zárójelezés

Példa: helyes zárójelezés

Példa grammatika

- ① $S' \rightarrow S$
 ② $S \rightarrow \epsilon \mid (S)S$



$\mathcal{I}_0 = closure([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S.] \}$

$\mathcal{I}_2 = read(\mathcal{I}_0, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

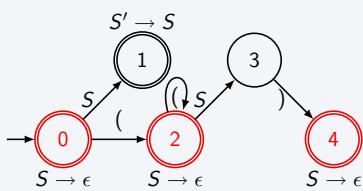
$\mathcal{I}_3 = read(\mathcal{I}_2, S) = \{[S \rightarrow (S.)S] \}$

$read(\mathcal{I}_2, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\} = \mathcal{I}_2$

Példa: helyes zárójelezés

Példa grammatika

- ➊ $S' \rightarrow S$
- ➋ $S \rightarrow \epsilon \mid (S)S$

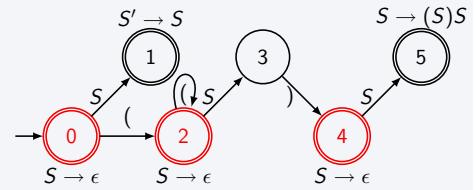


$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .], [S \rightarrow .(S)S]\}$
 $\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \{[S' \rightarrow S.\}\}$
 $\mathcal{I}_2 = \text{read}(\mathcal{I}_0, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$
 $\mathcal{I}_3 = \text{read}(\mathcal{I}_2, S) = \{[S \rightarrow (S.)S]\}$
 $\text{read}(\mathcal{I}_2, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\} = \mathcal{I}_2$
 $\mathcal{I}_4 = \text{read}(\mathcal{I}_3, ()) = \{[S \rightarrow (S).S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

Példa: helyes zárójelezés

Példa grammatika

- ➊ $S' \rightarrow S$
- ➋ $S \rightarrow \epsilon \mid (S)S$

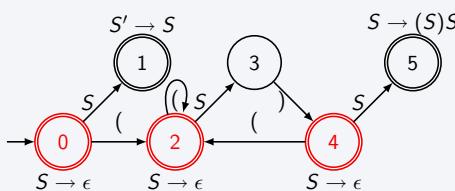


$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .], [S \rightarrow .(S)S]\}$
 $\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \{[S' \rightarrow S.\}\}$
 $\mathcal{I}_2 = \text{read}(\mathcal{I}_0, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$
 $\mathcal{I}_3 = \text{read}(\mathcal{I}_2, S) = \{[S \rightarrow (S.)S]\}$
 $\text{read}(\mathcal{I}_2, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\} = \mathcal{I}_2$
 $\mathcal{I}_4 = \text{read}(\mathcal{I}_3, ()) = \{[S \rightarrow (S).S], [S \rightarrow .], [S \rightarrow .(S)S]\}$
 $\mathcal{I}_5 = \text{read}(\mathcal{I}_4, S) = \{[S \rightarrow (S)S.\}\}$

Példa: helyes zárójelezés

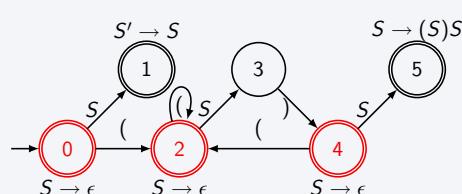
Példa grammatika

- ➊ $S' \rightarrow S$
- ➋ $S \rightarrow \epsilon \mid (S)S$



$\mathcal{I}_0 = \text{closure}([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .], [S \rightarrow .(S)S]\}$
 $\mathcal{I}_1 = \text{read}(\mathcal{I}_0, S) = \{[S' \rightarrow S.\}\}$
 $\mathcal{I}_2 = \text{read}(\mathcal{I}_0, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$
 $\mathcal{I}_3 = \text{read}(\mathcal{I}_2, S) = \{[S \rightarrow (S.)S]\}$
 $\text{read}(\mathcal{I}_2, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\} = \mathcal{I}_2$
 $\mathcal{I}_4 = \text{read}(\mathcal{I}_3, ()) = \{[S \rightarrow (S).S], [S \rightarrow .], [S \rightarrow .(S)S]\}$
 $\mathcal{I}_5 = \text{read}(\mathcal{I}_4, S) = \{[S \rightarrow (S)S.\}\}$
 $\text{read}(\mathcal{I}_4, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\} = \mathcal{I}_2$

Konfliktusok az LR(0) elemző táblázatban



	akció	S	$($	$)$
0	léptetés / redukálás ($S \rightarrow \epsilon$)	1	2	
1	OK			
2	léptetés / redukálás ($S \rightarrow \epsilon$)	3	2	
3	léptetés			4
4	léptetés / redukálás ($S \rightarrow \epsilon$)	5	(
5	redukálás ($S \rightarrow (S)S$)			

Az SLR(1) elemzés alapötlete

- Olvassunk előre egy szimbólumot!
- léptessünk, ha az automata tud lépni az előreolvasott szimbólummal
- redukáljunk, ha az előreolvasott szimbólum benne van a szabályhoz tartozó nemterminális $FOLLOW_1$ halmazában

Az SLR(1) elemzés alapötlete

- Olvassunk előre egy szimbólumot!
- léptessünk, ha az automata tud lépni az előreolvasott szimbólummal
- redukáljunk, ha az előreolvasott szimbólum benne van a szabályhoz tartozó nemterminális $FOLLOW_1$ halmazában

A helyes zárójelezés

$S' \Rightarrow S \Rightarrow (S)S \Rightarrow ((S)) \Rightarrow (\cdot)$
 $S' \Rightarrow S \Rightarrow (S)S \Rightarrow ((S)) \Rightarrow ((\cdot)) \Rightarrow (\cdot)$

$\mathcal{I}_2 = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

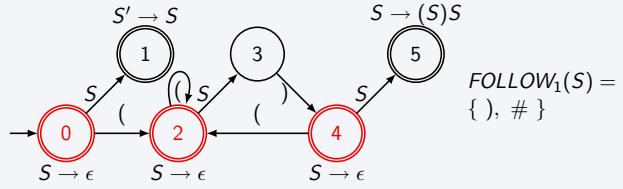
Az piros részek elolvasása után \mathcal{I}_2 állapotban van az automata:

- az első esetben $S \rightarrow \epsilon$ szerinti redukciót kell végrahajtani, mert $\cdot \in FOLLOW_1(S)$.
- a második esetben léptetni kell, mert $[S \rightarrow .(S)S] \in \mathcal{I}_2$.

Az SLR(1) elemzés szabályai

- Ha az aktuális állapot i , és az előreolvasás eredménye az a szimbólum:
 - ha $[A \rightarrow \alpha.a\beta] \in \mathcal{I}_i$ és $\text{read}(\mathcal{I}_i, a) = \mathcal{I}_j$, akkor léptetni kell, és átlépni a j állapotba,
 - ha $[A \rightarrow \alpha.] \in \mathcal{I}_i$ ($A \neq S'$) és $a \in FOLLOW_1(A)$, akkor redukálni kell $A \rightarrow \alpha$ szabály szerint,
 - ha $[S' \rightarrow S] \in \mathcal{I}_i$ és $a = \#$, akkor el kell fogadni a szöveget,
 - minden más esetben hibát kell jelezni.
- Ha az i állapotban A kerül a verem tetejére:
 - ha $\text{read}(\mathcal{I}_i, A) = \mathcal{I}_j$, akkor át kell lépni a j állapotba,
 - egyébként hibát kell jelezni.

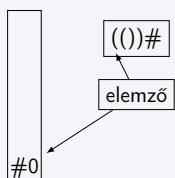
A helyes zárójelezés SLR(1) elemző táblázata



	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	

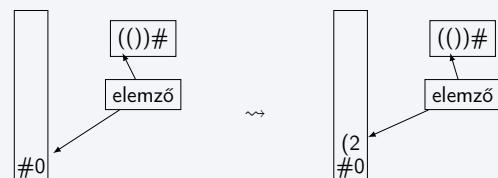
Példa

	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



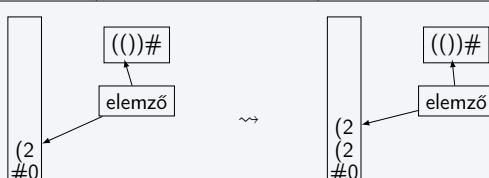
Példa

	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



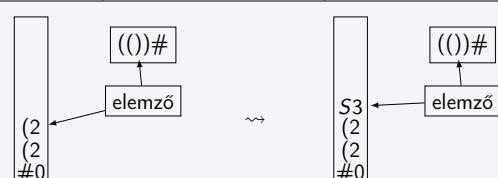
Példa

	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



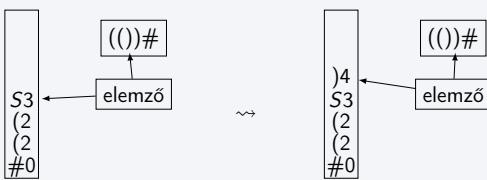
Példa

	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



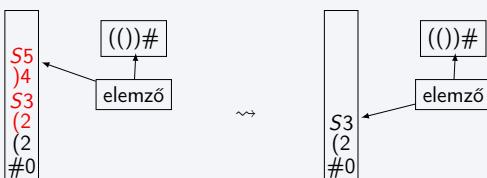
Példa

	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



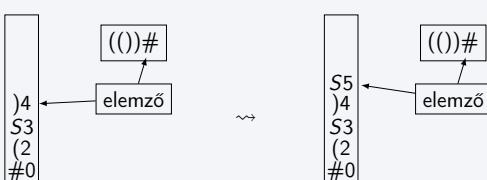
Példa

	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



Példa

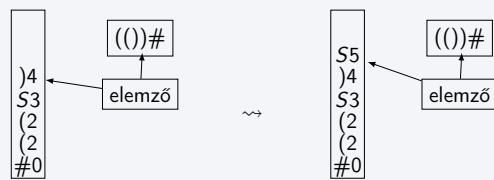
	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



Példa

Példa

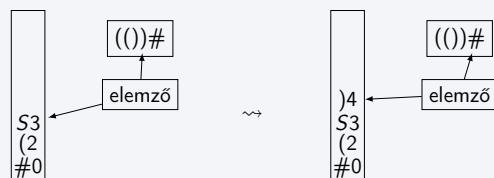
	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



Példa

Példa

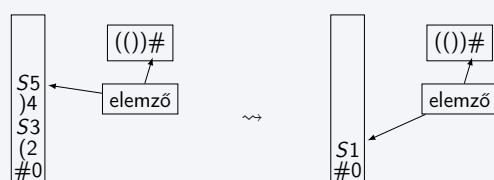
	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



1

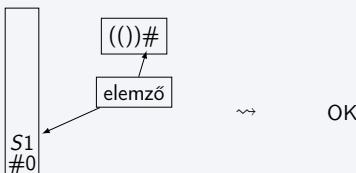
()

	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



Példa

	()	#	S
0	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	1
1			OK	
2	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	3
3		léptetés, 4		
4	léptetés, 2	redukálás ($S \rightarrow \epsilon$)	redukálás ($S \rightarrow \epsilon$)	5
5		redukálás ($S \rightarrow (S)S$)	redukálás ($S \rightarrow (S)S$)	



SLR(1) grammatika

Definíció: SLR(1) grammatika

Egy kiegészített grammatika $SLR(1)$ grammatika, ha az $SLR(1)$ elemző táblázata konfliktusmentes.

- elnevezés: „Simple LR”
- jobb, mint az $LR(0)$
- a valódi programnyelvek nyelvtanai általában nem $SLR(1)$ nyelvtanok

Probléma az SLR(1) elemzéssel

Példa grammatika

$S' \rightarrow S$	Egy program
$S \rightarrow U \mid E$	az egy utasítás vagy egy értékadás.
$U \rightarrow a$	Egy utasítás az egy azonosító szimbólum.
$E \rightarrow V = V$	Egy értékadás változó legyen egyenlő változó alakú.
$V \rightarrow a$	Egy változó az egy azonosító szimbólum.

Probléma az SLR(1) elemzéssel

Példa grammatika

$S' \rightarrow S$	Egy program
$S \rightarrow U \mid E$	az egy utasítás vagy egy értékadás.
$U \rightarrow a$	Egy utasítás az egy azonosító szimbólum.
$E \rightarrow V = V$	Egy értékadás változó legyen egyenlő változó alakú.
$V \rightarrow a$	Egy változó az egy azonosító szimbólum.

$$\mathcal{I}_0 = \{[S' \rightarrow .S], [S \rightarrow .U], [S \rightarrow .E], [U \rightarrow .a], [E \rightarrow .V = V], [V \rightarrow .a]\}$$

...

$$\mathcal{I}_4 = \text{read}(\mathcal{I}_0, a) = \{[U \rightarrow a], [V \rightarrow a]\}$$

Példa grammatika

$S' \rightarrow S$	Egy program
$S \rightarrow U \mid E$	az egy utasítás vagy egy értékadás.
$U \rightarrow a$	Egy utasítás az egy azonosító szimbólum.
$E \rightarrow V = V$	Egy értékadás változó legyen egyenlő változó alakú.
$V \rightarrow a$	Egy változó az egy azonosító szimbólum.

$$\mathcal{I}_0 = \{[S' \rightarrow .S], [S \rightarrow .U], [S \rightarrow .E], [U \rightarrow .a], [E \rightarrow .V = V], [V \rightarrow .a]\}$$

...

$$\mathcal{I}_4 = \text{read}(\mathcal{I}_0, a) = \{[U \rightarrow a], [V \rightarrow a]\}$$

$$\text{FOLLOW}_1(U) = \{\#\} \text{ és } \text{FOLLOW}_1(V) = \{=, \#\}$$

Redukálás / redukálás konfliktus!

Probléma az SLR(1) elemzéssel

$$\mathcal{I}_4 = \text{read}(\mathcal{I}_0, a) = \{[U \rightarrow a], [V \rightarrow a]\}$$

$$\text{FOLLOW}_1(U) = \{\#\} \text{ és } \text{FOLLOW}_1(V) = \{=, \#\}$$

Redukálás / redukálás konfliktus!

Probléma az SLR(1) elemzéssel

$$\mathcal{I}_4 = \text{read}(\mathcal{I}_0, a) = \{[U \rightarrow a], [V \rightarrow a]\}$$

$\text{FOLLOW}_1(U) = \{\#\}$ és $\text{FOLLOW}_1(V) = \{=, \#\}$
Redukálás / redukálás konfliktus!

Az $a\#$ szövegből az a elolvasása után az SLR(1) elemző nem tud dönteni a $U \rightarrow a$ és $V \rightarrow a$ szerinti redukciók között, mert a következő szimbólum ($\#$) benne van az U és a $V \text{ FOLLOW}_1$ halmazában is.

Probléma az SLR(1) elemzéssel

$$\mathcal{I}_4 = \text{read}(\mathcal{I}_0, a) = \{[U \rightarrow a], [V \rightarrow a]\}$$

$\text{FOLLOW}_1(U) = \{\#\}$ és $\text{FOLLOW}_1(V) = \{=, \#\}$
Redukálás / redukálás konfliktus!

Az $a\#$ szövegből az a elolvasása után az SLR(1) elemző nem tud dönteni a $U \rightarrow a$ és $V \rightarrow a$ szerinti redukciók között, mert a következő szimbólum ($\#$) benne van az U és a $V \text{ FOLLOW}_1$ halmazában is.

Pedig a szöveg elején a V -t csak az $=$ szimbólum követheti...

Az $LR(1)$ elemzés alapötlete

- a FOLLOW_1 halmaz globális az egész grammaticákra
- előfordulhat, hogy egy adott állapotban a FOLLOW_1 halmaznak nem minden eleme követheti a szabályt

Az $LR(1)$ elemzés alapötlete

- a FOLLOW_1 halmaz globális az egész grammaticákra
- előfordulhat, hogy egy adott állapotban a FOLLOW_1 halmaznak nem minden eleme követheti a szabályt
- Vegyük hozzá az $LR(0)$ elemekhez azokat a szimbólumokat, amik követhetik a szabályt az adott állapotban!

 $LR(1)$ elemekDefiníció: $LR(1)$ elem

Ha $A \rightarrow \alpha$ a grammatica egy helyettesítési szabálya, akkor az $\alpha = \alpha_1\alpha_2$ tetszőleges felbontás és a terminális szimbólum (vagy $a = \#$) esetén $[A \rightarrow \alpha_1.\alpha_2, a]$ a grammatica egy $LR(1)$ -eleme.

$A \rightarrow \alpha_1.\alpha_2$ az $LR(1)$ elem magja, a pedig az előreolvasási szimbóluma.

 $LR(1)$ elemekDefiníció: $LR(1)$ elem

Ha $A \rightarrow \alpha$ a grammatica egy helyettesítési szabálya, akkor az $\alpha = \alpha_1\alpha_2$ tetszőleges felbontás és a terminális szimbólum (vagy $a = \#$) esetén $[A \rightarrow \alpha_1.\alpha_2, a]$ a grammatica egy $LR(1)$ -eleme.

$A \rightarrow \alpha_1.\alpha_2$ az $LR(1)$ elem magja, a pedig az előreolvasási szimbóluma.

- $[V \rightarrow a., =]$ jelentése: a $V \rightarrow a$ szabály építését befejeztük és a szabályt az $=$ szimbólum követheti.

A lezárás művelet

- Ha $[V \rightarrow .V = V, \#]$ állapotban vagyunk, akkor a $V \rightarrow a$ szabályt kezdhetjük építeni, amit az = szimbólum követhet.
- Tehát az adott kanonikus halmazhoz $[V \rightarrow .a, =]$ is hozzátarozik.

A lezárás művelet

- Ha $[V \rightarrow .V = V, \#]$ állapotban vagyunk, akkor a $V \rightarrow a$ szabályt kezdhetjük építeni, amit az = szimbólum követhet.
- Tehát az adott kanonikus halmazhoz $[V \rightarrow .a, =]$ is hozzátarozik.

Definíció: lezárás (closure)

Ha \mathcal{I} a grammatika egy **LR(1)** elemhalmaza, akkor $closure(\mathcal{I})$ a legszűkebb olyan halmaz, amely az alábbi tulajdonságokkal rendelkezik:

- $\mathcal{I} \subseteq closure(\mathcal{I})$
- ha $[A \rightarrow \alpha.B\gamma.a] \in closure(\mathcal{I})$, és $B \rightarrow \beta$ a grammatika egy szabálya, akkor $\forall b \in FIRST_1(\gamma a)$ esetén $[B \rightarrow .\beta.b] \in closure(\mathcal{I})$

Az olvasás művelet

- Ha $[V \rightarrow .V = V, \#]$ állapotban vagyunk, és V kerül a verem tetejére, akkor $[V \rightarrow V. = V, \#]$ állapotba jutunk.

Az olvasás művelet

- Ha $[V \rightarrow .V = V, \#]$ állapotban vagyunk, és V kerül a verem tetejére, akkor $[V \rightarrow V. = V, \#]$ állapotba jutunk.

Definíció: olvasás (read)

Ha \mathcal{I} a grammatika egy **LR(1)** elemhalmaza, X pedig terminális vagy nemterminális szimbóluma, akkor $read(\mathcal{I}, X)$ a legszűkebb olyan halmaz, amely az alábbi tulajdonsággal rendelkezik:

- ha $[A \rightarrow \alpha.X\beta.a] \in \mathcal{I}$, akkor $closure([A \rightarrow \alpha.X.\beta.a]) \subseteq read(\mathcal{I}, X)$.

LR(1) kanonikus halmazok

Definíció: LR(1) kanonikus halmazok

- $closure([S' \rightarrow .S, \#])$ a grammatika egy kanonikus halmaza.
- Ha \mathcal{I} a grammatika egy kanonikus elemhalmaza, X pedig terminális vagy nemterminális szimbóluma, és $read(\mathcal{I}, X)$ nem üres, akkor $read(\mathcal{I}, X)$ a grammatika egy kanonikus halmaza.
- Az első két szabálytal a összes kanonikus halmaz előáll.

Az LR(1) elemzés szabályai

- Ha az aktuális állapot i , és az előreolvasás eredménye az a szimbólum:
 - ha $[A \rightarrow \alpha.a\beta, b] \in \mathcal{I}_i$ és $read(\mathcal{I}_i, a) = \mathcal{I}_j$, akkor léptetni kell, és átlépni a j állapotba,
 - ha $[A \rightarrow \alpha., a] \in \mathcal{I}_i$ ($A \neq S'$), akkor redukálni kell $A \rightarrow \alpha$ szabály szerint,
 - ha $[S' \rightarrow S., \#] \in \mathcal{I}_i$ és $a = \#$, akkor el kell fogadni a szöveget,
 - minden más esetben hibát kell jelezni.
- Ha az i állapotban A kerül a verem tetejére:
 - ha $read(\mathcal{I}_i, A) = \mathcal{I}_j$, akkor át kell lépni a j állapotba,
 - egyébként hibát kell jelezni.

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

$$\begin{aligned} \mathcal{I}_0 &= closure([S' \rightarrow .S, \#]) = \\ &= \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#], \\ &\quad [E \rightarrow .V = V, \#], [V \rightarrow .a, =]\} \end{aligned}$$

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

$$\begin{aligned} \mathcal{I}_0 &= closure([S' \rightarrow .S, \#]) = \\ &= \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#], \\ &\quad [E \rightarrow .V = V, \#], [V \rightarrow .a, =]\} \end{aligned}$$

$$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S., \#]\}$$

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

$$\begin{aligned} \mathcal{I}_0 &= closure([S' \rightarrow .S, \#]) = \\ &= \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#], \\ &\quad [E \rightarrow .V = V, \#], [V \rightarrow .a, =]\} \end{aligned}$$

$$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S., \#]\}$$

$$\mathcal{I}_2 = read(\mathcal{I}_0, U) = \{[S \rightarrow U., \#]\}$$

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

$$\begin{aligned} \mathcal{I}_0 &= closure([S' \rightarrow .S, \#]) = \\ &= \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#], \\ &\quad [E \rightarrow .V = V, \#], [V \rightarrow .a, =]\} \end{aligned}$$

$$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S., \#]\}$$

$$\mathcal{I}_2 = read(\mathcal{I}_0, U) = \{[S \rightarrow U., \#]\}$$

$$\mathcal{I}_3 = read(\mathcal{I}_0, E) = \{[S \rightarrow E., \#]\}$$

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

$$\begin{aligned} \mathcal{I}_0 &= closure([S' \rightarrow .S, \#]) = \\ &= \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#], \\ &\quad [E \rightarrow .V = V, \#], [V \rightarrow .a, =]\} \end{aligned}$$

$$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S., \#]\}$$

$$\mathcal{I}_2 = read(\mathcal{I}_0, U) = \{[S \rightarrow U., \#]\}$$

$$\mathcal{I}_3 = read(\mathcal{I}_0, E) = \{[S \rightarrow E., \#]\}$$

$$\mathcal{I}_4 = read(\mathcal{I}_0, a) = \{[U \rightarrow a., \#], [V \rightarrow a., =]\} \text{ Nincs konfliktus!}$$

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

$\mathcal{I}_0 = closure([S' \rightarrow .S, \#]) =$
 $= \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#],$
 $[E \rightarrow .V = V, \#], [V \rightarrow .a, =]\}$

$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S, \#]\}$

$\mathcal{I}_2 = read(\mathcal{I}_0, U) = \{[S \rightarrow U, \#]\}$

$\mathcal{I}_3 = read(\mathcal{I}_0, E) = \{[S \rightarrow E, \#]\}$

$\mathcal{I}_4 = read(\mathcal{I}_0, a) = \{[U \rightarrow a, \#], [V \rightarrow a, =]\}$ Nincs konfliktus!

$\mathcal{I}_5 = read(\mathcal{I}_0, V) = \{[E \rightarrow V = V, \#]\}$

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

$\mathcal{I}_0 = closure([S' \rightarrow .S, \#]) =$
 $= \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#],$
 $[E \rightarrow .V = V, \#], [V \rightarrow .a, =]\}$

$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S, \#]\}$

$\mathcal{I}_2 = read(\mathcal{I}_0, U) = \{[S \rightarrow U, \#]\}$

$\mathcal{I}_3 = read(\mathcal{I}_0, E) = \{[S \rightarrow E, \#]\}$

$\mathcal{I}_4 = read(\mathcal{I}_0, a) = \{[U \rightarrow a, \#], [V \rightarrow a, =]\}$ Nincs konfliktus!

$\mathcal{I}_5 = read(\mathcal{I}_0, V) = \{[E \rightarrow V = V, \#]\}$

$\mathcal{I}_6 = read(\mathcal{I}_5, =) = \{[E \rightarrow V = .V, \#], [V \rightarrow .a, =]\}$

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

$\mathcal{I}_0 = closure([S' \rightarrow .S, \#]) =$
 $= \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#],$
 $[E \rightarrow .V = V, \#], [V \rightarrow .a, =]\}$

$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S, \#]\}$

$\mathcal{I}_2 = read(\mathcal{I}_0, U) = \{[S \rightarrow U, \#]\}$

$\mathcal{I}_3 = read(\mathcal{I}_0, E) = \{[S \rightarrow E, \#]\}$

$\mathcal{I}_4 = read(\mathcal{I}_0, a) = \{[U \rightarrow a, \#], [V \rightarrow a, =]\}$ Nincs konfliktus!

$\mathcal{I}_5 = read(\mathcal{I}_0, V) = \{[E \rightarrow V = V, \#]\}$

$\mathcal{I}_6 = read(\mathcal{I}_5, =) = \{[E \rightarrow V = .V, \#], [V \rightarrow .a, =]\}$

$\mathcal{I}_7 = read(\mathcal{I}_6, V) = \{[E \rightarrow V = V, \#]\}$

Példa

Példa grammatika

$$S' \rightarrow S \quad S \rightarrow U \mid E \quad U \rightarrow a \quad E \rightarrow V = V \quad V \rightarrow a$$

$\mathcal{I}_0 = closure([S' \rightarrow .S, \#]) =$
 $= \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#],$
 $[E \rightarrow .V = V, \#], [V \rightarrow .a, =]\}$

$\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S, \#]\}$

$\mathcal{I}_2 = read(\mathcal{I}_0, U) = \{[S \rightarrow U, \#]\}$

$\mathcal{I}_3 = read(\mathcal{I}_0, E) = \{[S \rightarrow E, \#]\}$

$\mathcal{I}_4 = read(\mathcal{I}_0, a) = \{[U \rightarrow a, \#], [V \rightarrow a, =]\}$ Nincs konfliktus!

$\mathcal{I}_5 = read(\mathcal{I}_0, V) = \{[E \rightarrow V = V, \#]\}$

$\mathcal{I}_6 = read(\mathcal{I}_5, =) = \{[E \rightarrow V = .V, \#], [V \rightarrow .a, =]\}$

$\mathcal{I}_7 = read(\mathcal{I}_6, V) = \{[E \rightarrow V = V, \#]\}$

$\mathcal{I}_8 = read(\mathcal{I}_6, a) = \{[V \rightarrow a, \#]\}$

Az elemző táblázat kitöltése

$\mathcal{I}_0 = \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#],$
 $[E \rightarrow .V = V, \#], [V \rightarrow .a, =]\}$

$read(\mathcal{I}_0, S) = \mathcal{I}_1$

	x	=	#	S	U	E	V
0			1				
1							
2							
3							
4							
5							
6							
7							
8							

Az elemző táblázat kitöltése

$\mathcal{I}_0 = \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#],$
 $[E \rightarrow .V = V, \#], [V \rightarrow .a, =]\}$

$read(\mathcal{I}_0, U) = \mathcal{I}_2$

	x	=	#	S	U	E	V
0				1	2		
1							
2							
3							
4							
5							
6							
7							
8							

Az elemző táblázat kitöltése

 $\mathcal{I}_0 = \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#], [E \rightarrow .V = V, \#], [V \rightarrow .a, =]\}$
 $\text{read}(\mathcal{I}_0, E) = \mathcal{I}_3$

	x	=	#	S	U	E	V
0				1	2	3	
1							
2							
3							
4							
5							
6							
7							
8							

Az elemző táblázat kitöltése

 $\mathcal{I}_0 = \{[S' \rightarrow .S, \#], [S \rightarrow .U, \#], [S \rightarrow .E, \#], [U \rightarrow .a, \#], [E \rightarrow .V = V, \#], [V \rightarrow .a, =]\}$
 $\text{read}(\mathcal{I}_0, V) = \mathcal{I}_5$

	a	=	#	S	U	E	V
0	léptetés, 4			1	2	3	5
1							
2							
3							
4							
5							
6							
7							
8							

Az elemző táblázat kitöltése

 $\mathcal{I}_2 = \{[S \rightarrow U., \#]\}$

	a	=	#	S	U	E	V
0	léptetés, 4			1	2	3	5
1		OK					
2		redukálás, $S \rightarrow U$					
3							
4							
5							
6							
7							
8							

Az elemző táblázat kitöltése

 $\mathcal{I}_3 = \{[S \rightarrow E., \#]\}$

	a	=	#	S	U	E	V
0	léptetés, 4			1	2	3	5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4							
5							
6							
7							
8							

Az elemző táblázat kitöltése

$$\mathcal{I}_4 = \{[U \rightarrow a., \#], [V \rightarrow a., =]\}$$

	a	=	#	S	U	E	V
0	léptetés, 4			1	2	3	5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $U \rightarrow a$					
5							
6							
7							
8							

Az elemző táblázat kitöltése

$$\mathcal{I}_4 = \{[U \rightarrow a., \#], [V \rightarrow a., =]\}$$

	a	=	#	S	U	E	V
0	léptetés, 4						1 2 3 5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5							
6							
7							
8							

Az elemző táblázat kitöltése

$$\mathcal{I}_5 = \{[E \rightarrow V. = V, \#]\}$$

$\text{read}(\mathcal{I}_5, =) = \mathcal{I}_6$

	a	=	#	S	U	E	V
0	léptetés, 4			1	2	3	5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6							
7							
8							

Az elemző táblázat kitöltése

$$\mathcal{I}_6 = \{[E \rightarrow V = .V, \#], [V \rightarrow .a, \#]\}$$

$\text{read}(\mathcal{I}_6, V) = \mathcal{I}_7$

	a	=	#	S	U	E	V
0	léptetés, 4						1 2 3 5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6							7
7							
8							

Az elemző táblázat kitöltése

$$\mathcal{I}_6 = \{[E \rightarrow V = .V, \#], [V \rightarrow .a, \#]\}$$

$\text{read}(\mathcal{I}_6, a) = \mathcal{I}_8$

	a	=	#	S	U	E	V
0	léptetés, 4			1	2	3	5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6	léptetés, 8						7
7							
8							

Az elemző táblázat kitöltése

$$\mathcal{I}_7 = \{[E \rightarrow V = V, \#]\}$$

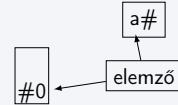
	a	=	#	S	U	E	V
0	léptetés, 4						1 2 3 5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6	léptetés, 8						7
7							
8							

Az elemző táblázat kitöltése

$$\mathcal{I}_7 = \{[V \rightarrow a, \#]\}$$

	a	=	#	S	U	E	V
0	léptetés, 4			1	2	3	5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6	léptetés, 8						7
7		redukálás, $E \rightarrow V = V$					
8		redukálás, $V \rightarrow a$					

	a	=	#	S	U	E	V
0	léptetés, 4						1 2 3 5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6	léptetés, 8						7
7		redukálás, $E \rightarrow V = V$					
8		redukálás, $V \rightarrow a$					



Az a# szöveg elemzése

	a	=	#	S	U	E	V
0	léptetés, 4			1	2	3	5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6	léptetés, 8						7
7		redukálás, $E \rightarrow V = V$					
8		redukálás, $V \rightarrow a$					



Az a# szöveg elemzése

	a	=	#	S	U	E	V
0	léptetés, 4						1 2 3 5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6	léptetés, 8						7
7		redukálás, $E \rightarrow V = V$					
8		redukálás, $V \rightarrow a$					



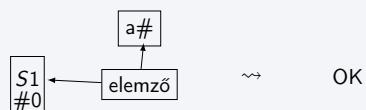
Az a# szöveg elemzése

	a	=	#	S	U	E	V
0	léptetés, 4			1	2	3	5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6	léptetés, 8						7
7		redukálás, $E \rightarrow V = V$					
8		redukálás, $V \rightarrow a$					



Az a# szöveg elemzése

	a	=	#	S	U	E	V
0	léptetés, 4						1 2 3 5
1		OK					
2		redukálás, $S \rightarrow U$					
3		redukálás, $S \rightarrow E$					
4		redukálás, $V \rightarrow a$	redukálás, $U \rightarrow a$				
5		léptetés, 6					
6	léptetés, 8						7
7		redukálás, $E \rightarrow V = V$					
8		redukálás, $V \rightarrow a$					



Az $LR(1)$ elemző táblázat konfliktusmentessége

Tétel

Egy grammatika pontosan akkor $LR(1)$ grammatika, ha az $LR(1)$ elemző táblázatai konfliktusmentesek.

Az $LR(1)$ elemző létrehozása véges

- az $LR(1)$ -elemek száma véges
- a closure és read függvények kiszámítása véges
- a kanonikus halmazok száma véges
- a kanonikus halmazok kiszámítása véges

Az $LR(1)$ elemző a megadott módon véges sok lépében és teljesen automatikusan létrehozható.

Járható prefix, érvényes LR -elem

- járható prefix: a mondatformának olyan prefixei, amelyek legfeljebb a nyél végéig tartanak
 - ezeket járja be az elemző automata
 - a maximális járható prefixnek a végén ott a teljes nyél

Járható prefix, érvényes LR -elem

- járható prefix: a mondatformának olyan prefixei, amelyek legfeljebb a nyél végéig tartanak
 - ezeket járja be az elemző automata
 - a maximális járható prefixnek a végén ott a teljes nyél
- járható prefixre érvényes LR -elemek: a járható prefix „lehetséges folytatásai”
 - melyik szabályok építésében hol tarthatunk egy adott járható prefix elemzése után?

Járható prefix, érvényes LR -elem

- járható prefix: a mondatformának olyan prefixei, amelyek legfeljebb a nyél végéig tartanak
 - ezeket járja be az elemző automata
 - a maximális járható prefixnek a végén ott a teljes nyél
- járható prefixre érvényes LR -elemek: a járható prefix „lehetséges folytatásai”
 - melyik szabályok építésében hol tarthatunk egy adott járható prefix elemzése után?

Definíció: Járható prefixre érvényes $LR(1)$ -elem

A grammatika egy $[A \rightarrow \alpha.\beta, a]$ $LR(1)$ -eleme érvényes a $\gamma\alpha$ járható prefixre nézve, ha

$$S' \Rightarrow^* \gamma A x \Rightarrow \gamma \alpha \beta x,$$

és $x \neq \epsilon$ esetén a az x első szimbóluma, $x = \epsilon$ esetén pedig $a = \#$.

Az $LR(1)$ -elemzés nagy tétele

Egy γ járható prefix hatására az elemző automatája a kezdőállapotból olyan állapotba kerül, amelyhez tartozó kanonikus halmaz éppen a γ járható prefixre érvényes $LR(1)$ elemeket tartalmazza.

Az $LR(1)$ elemzés helyessége

Az $LR(1)$ -elemzés nagy tétele

Egy γ járható prefix hatására az elemző automatája a kezdőállapotból olyan állapotba kerül, amelyhez tartozó kanonikus halmaz éppen a γ járható prefixre érvényes $LR(1)$ elemeket tartalmazza.

- Összefoglalva:

- az $LR(1)$ elemző automatikusan és véges lépésekben generálható a nyelvtan szabályaiból
- minden $LR(1)$ grammatika elemezéséhez használható
- az elemző minden lépést írja elő a fenti tételek miatt

- Probléma:

- túl sok állapota van...

LR elemzések (LALR(1) elemzés)

Fordítóprogramok előadás (A,C,T)

1

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Emlékeztető

- $LR(0)$ elemzés

- léptetés: $[S \rightarrow (S.)S]$
- redukálás: $\{[S \rightarrow (S)S]\}$
- konfliktus: $\{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

Emlékeztető

- $LR(0)$ elemzés

- léptetés: $[S \rightarrow (S.)S]$
- redukálás: $\{[S \rightarrow (S)S]\}$
- konfliktus: $\{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

- $SLR(1)$ elemzés

Az $\{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$ állapotban:

- léptetés: (hatására
- redukálás:) vagy # hatására, mert ezek elemei $FOLLOW_1(S)$ -nek

Emlékeztető

- $LR(0)$ elemzés

- léptetés: $[S \rightarrow (S.)S]$
- redukálás: $\{[S \rightarrow (S)S]\}$
- konfliktus: $\{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$

- $SLR(1)$ elemzés

Az $\{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\}$ állapotban:

- léptetés: (hatására
- redukálás:) vagy # hatására, mert ezek elemei $FOLLOW_1(S)$ -nek

- $LR(1)$ elemzés

Az $\{[S \rightarrow (.S)S, \#], [S \rightarrow ..], [S \rightarrow .(S)S, \#]\}$ állapotban:

- léptetés: (hatására
- redukálás:) hatására

2

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

2

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

$SLR(1)$ és $LR(1)$ állapotok száma

$SLR(1)$ ($LR(0)$) kanonikus halmazok

$$\begin{aligned} I_0 &= closure([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow .], [S \rightarrow .(S)S]\} \\ I_1 &= read(I_0, S) = \{[S' \rightarrow S.\] \} \\ I_2 &= read(I_0, ()) = \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\} \\ I_3 &= read(I_2, S) = \{[S \rightarrow (S)S]\} \\ \text{read}(I_2, ()) &= \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\} = I_2 \\ I_4 &= read(I_3, ()) = \{[S \rightarrow (S).S], [S \rightarrow ..], [S \rightarrow .(S)S]\} \\ I_5 &= read(I_4, S) = \{[S \rightarrow (S)S]\} \end{aligned}$$

$SLR(1)$ és $LR(1)$ állapotok száma

$SLR(1)$ ($LR(0)$) kanonikus halmazok

$$\begin{aligned} I_0 &= closure([S' \rightarrow .S]) = \{[S' \rightarrow .S], [S \rightarrow ..], [S \rightarrow .(S)S, \#]\} \\ I_1 &= read(I_0, S) = \{[S' \rightarrow S., \#]\} \\ I_2 &= read(I_0, ()) = \{[S \rightarrow (.S)S, \#], [S \rightarrow ..], [S \rightarrow .(S)S, \#]\} \\ \text{read}(I_2, ()) &= \{[S \rightarrow (.S)S], [S \rightarrow ..], [S \rightarrow .(S)S]\} = I_2 \\ I_3 &= read(I_2, S) = \{[S \rightarrow (S).S, \#]\} \\ I_4 &= read(I_3, ()) = \{[S \rightarrow (S)S], [S \rightarrow ..], [S \rightarrow .(S)S, \#]\} \end{aligned}$$

$LR(1)$ kanonikus halmazok

$$\begin{aligned} I_0 &= closure([S' \rightarrow .S, \#]) = \{[S' \rightarrow .S, \#], [S \rightarrow .., \#], [S \rightarrow .(S)S, \#]\} \\ I_1 &= read(I_0, S) = \{[S' \rightarrow S., \#]\} \\ I_2 &= read(I_0, ()) = \{[S \rightarrow (.S)S, \#], [S \rightarrow ..], [S \rightarrow .(S)S, \#]\} \\ I_3 &= read(I_2, S) = \{[S \rightarrow (S).S, \#]\} \\ \text{read}(I_2, ()) &= \{[S \rightarrow (.S)S], [S \rightarrow ..], [S \rightarrow .(S)S, \#]\} = I_3 \\ \dots & \end{aligned}$$

Az előreolvasási szimbólumok miatt nő az állapotok száma!

3

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

4

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

SLR(1) és LR(1) állapotok száma

LR(1) kanonikus halmazok

$\mathcal{I}_0 = closure([S' \rightarrow .S, \#]) = \{[S' \rightarrow .S, \#], [S \rightarrow .., \#], [S \rightarrow .(S)S, \#]\}$
 $\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S., \#]\}$
 $\mathcal{I}_2 = read(\mathcal{I}_0, ()) = \{[S \rightarrow .(S)S, \#], [S \rightarrow ..,], [S \rightarrow .(S)S,]\}$
 $\mathcal{I}_3 = read(\mathcal{I}_2, S) = \{[S \rightarrow .(S)S, \#]\}$
 $\mathcal{I}_4 = read(\mathcal{I}_2, ()) = \{[S \rightarrow .(S)S, .], [S \rightarrow ..,], [S \rightarrow .(S)S,]\}$
 $\mathcal{I}_5 = read(\mathcal{I}_3, ()) = \{[S \rightarrow .(S)S, .], [S \rightarrow .., \#], [S \rightarrow .(S)S, \#]\}$
 $\mathcal{I}_6 = read(\mathcal{I}_4, S) = \{[S \rightarrow .(S)S, .]\}$
 $read(\mathcal{I}_4, ()) = \{[S \rightarrow .(S)S, .], [S \rightarrow ..,], [S \rightarrow .(S)S,]\} = \mathcal{I}_4$
 $\mathcal{I}_7 = read(\mathcal{I}_5, S) = \{[S \rightarrow .(S)S, \#]\}$
 $read(\mathcal{I}_5, ()) = \{[S \rightarrow .(S)S, \#], [S \rightarrow ..,], [S \rightarrow .(S)S,]\} = \mathcal{I}_2$
 $\mathcal{I}_8 = read(\mathcal{I}_6, ()) = \{[S \rightarrow .(S)S, .], [S \rightarrow ..,], [S \rightarrow .(S)S,]\}$
 $\mathcal{I}_9 = read(\mathcal{I}_8, S) = \{[S \rightarrow .(S)S, .]\}$
 $read(\mathcal{I}_8, ()) = \{[S \rightarrow .(S)S, .], [S \rightarrow ..,], [S \rightarrow .(S)S,]\} = \mathcal{I}_4$

- SLR(1) (és LR(0)): 6 állapot; LR(1): 10 állapot
- Valódi programnyelveknél:
SLR(1): néhány száz; LR(1): néhány ezer

5

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Egyesíthető LR(1) kanonikus halmazok

Egyes kanonikus halmaz párok csak az előreolvasási szimbólumokban különböznek.

LR(1) kanonikus halmazok

$\mathcal{I}_0 = closure([S' \rightarrow .S, \#]) = \{[S' \rightarrow .S, \#], [S \rightarrow .., \#], [S \rightarrow .(S)S, \#]\}$
 $\mathcal{I}_1 = read(\mathcal{I}_0, S) = \{[S' \rightarrow S., \#]\}$
 $\mathcal{I}_2 = read(\mathcal{I}_0, ()) = \{[S \rightarrow .(S)S, \#], [S \rightarrow ..,], [S \rightarrow .(S)S,]\}$
 $\mathcal{I}_3 = read(\mathcal{I}_2, S) = \{[S \rightarrow .(S)S, \#]\}$
 $\mathcal{I}_4 = read(\mathcal{I}_2, ()) = \{[S \rightarrow .(S)S, .], [S \rightarrow ..,], [S \rightarrow .(S)S,]\}$
 $\mathcal{I}_5 = read(\mathcal{I}_3, ()) = \{[S \rightarrow .(S)S, .], [S \rightarrow .., \#], [S \rightarrow .(S)S, \#]\}$
 $\mathcal{I}_6 = read(\mathcal{I}_4, S) = \{[S \rightarrow .(S)S, .]\}$
 $\mathcal{I}_7 = read(\mathcal{I}_5, S) = \{[S \rightarrow .(S)S, \#]\}$
 $\mathcal{I}_8 = read(\mathcal{I}_6, ()) = \{[S \rightarrow .(S)S, .], [S \rightarrow ..,], [S \rightarrow .(S)S,]\}$
 $\mathcal{I}_9 = read(\mathcal{I}_8, S) = \{[S \rightarrow .(S)S, .]\}$

Ezeket a halmazokat **egyesíthetőnek** nevezük.

6

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

LALR(1) kanonikus halmazok

Az egyesíthető kanonikus halmazokat vonjuk össze!

LALR(1) kanonikus halmazok

$\mathcal{K}_0 = \mathcal{I}_0 = \{[S' \rightarrow .S, \#], [S \rightarrow .., \#], [S \rightarrow .(S)S, \#]\}$
 $\mathcal{K}_1 = \mathcal{I}_1 = \{[S' \rightarrow S., \#]\}$
 $\mathcal{K}_2 = \mathcal{I}_2 \cup \mathcal{I}_4 = \{[S \rightarrow .(S)S, \#], [S \rightarrow ..,], [S \rightarrow .(S)S, .], [S \rightarrow .(S)S,]\}$
 $\mathcal{K}_3 = \mathcal{I}_3 \cup \mathcal{I}_6 = \{[S \rightarrow .(S)S, \#], [S \rightarrow .(S)S, .]\}$
 $\mathcal{K}_4 = \mathcal{I}_5 \cup \mathcal{I}_8 = \{[S \rightarrow .(S)S, \#], [S \rightarrow .., \#], [S \rightarrow .(S)S, .], [S \rightarrow .(S)S, .], [S \rightarrow .(S)S,]\}$
 $\mathcal{K}_5 = \mathcal{I}_7 \cup \mathcal{I}_9 = \{[S \rightarrow .(S)S, \#], [S \rightarrow .(S)S, .]\}$

Az egyesített kanonikus halmazokat **LALR(1) kanonikus halmazoknak** nevezünk.

7

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

LALR(1) elemzés

- Az LALR(1) elemzőnek ugyanannyi állapota van, mint az SLR(1) (és LR(0)) emelzőknek.
- Mivel megjelennek benne az előreolvasási szimbólumok, több nyelvtan lesz elemezhető vele, mint SLR(1) módszerrel.
- De nem minden LR(1) grammátika esetén használható!
- Név: LookAhead LR
(Előreolvasós LR)

8

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Lehetséges problémák az egyesítések miatt

- Előfordulhat-e léptetés-léptetés konfliktus (azaz, hogy két különböző állapotba kellene lépni ugyanazon szimbólum hatására)?
 - Ha \mathcal{I}_m és \mathcal{I}_n halmazok egyesíthetők, akkor $read(\mathcal{I}_m, X)$ és $read(\mathcal{I}_n, X)$ is egyesíthető.
 - Léptetés-léptetés konfliktus nem fordulhat elő!

9

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Lehetséges problémák az egyesítések miatt

- Előfordulhat-e léptetés-léptetés konfliktus (azaz, hogy két különböző állapotba kellene lépni ugyanazon szimbólum hatására)?
 - Ha \mathcal{I}_m és \mathcal{I}_n halmazok egyesíthetők, akkor $read(\mathcal{I}_m, X)$ és $read(\mathcal{I}_n, X)$ is egyesíthető.
 - Léptetés-léptetés konfliktus nem fordulhat elő!
- Előfordulhat-e léptetés-redukálás konfliktus?
 - Tegyük fel, hogy $[A \rightarrow \alpha.a\beta, b]$ és $[B \rightarrow \gamma.., a]$ elemei egy egyesített halmaznak.
 - Nézzük meg azt az LR(1) kanonikus halmazt, amelyikből $[B \rightarrow \gamma.., a]$ -t kaptuk. Ebben benne kell lennie egy $A \rightarrow \alpha.a\beta$ magú elemnek is, csak esetleg más előreolvasási szimbólummal.
 - Viszont ha $[A \rightarrow \alpha.a\beta, b']$ és $[B \rightarrow \gamma.., a]$ elemei egy LR(1) kanonikus halmaznak, akkor már itt léptetés-redukálás konfliktus van, azaz nem LR(1)-es a grammáti!
 - Léptetés-redukálás konfliktus sem fordulhat elő!

9

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Lehetséges problémák az egyesítések miatt

- Redukálás-redukálás konfliktus: előfordulhat!

Példa $LR(1)$, de nem $LALR(1)$ grammatikára

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow aAd \mid bBd \mid aBe \mid bAe \\ A &\rightarrow c \\ B &\rightarrow c \end{aligned}$$

- ac járható prefixre érvényes elemek: $\{[A \rightarrow c., d], [B \rightarrow c., e]\}$
- bc járható prefixre érvényes elemek: $\{[A \rightarrow c., e], [B \rightarrow c., d]\}$

10

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Lehetséges problémák az egyesítések miatt

- Redukálás-redukálás konfliktus: előfordulhat!

Példa $LR(1)$, de nem $LALR(1)$ grammatikára

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow aAd \mid bBd \mid aBe \mid bAe \\ A &\rightarrow c \\ B &\rightarrow c \end{aligned}$$

- ac járható prefixre érvényes elemek: $\{[A \rightarrow c., d], [B \rightarrow c., e]\}$
- bc járható prefixre érvényes elemek: $\{[A \rightarrow c., e], [B \rightarrow c., d]\}$
- az egyesítés után:
 - $\{[A \rightarrow c., d], [B \rightarrow c., e], [A \rightarrow c., e], [B \rightarrow c., d]\}$
 - redukálás-redukálás konfliktus!

10

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

$LALR(1)$ elemzés, $LALR(1)$ grammatika

- az $LALR(1)$ elemző táblázat kitöltése megegyezik az $LR(1)$ táblázat kitöltésével

11

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

$LALR(1)$ elemzés, $LALR(1)$ grammatika

- az $LALR(1)$ elemző táblázat kitöltése megegyezik az $LR(1)$ táblázat kitöltésével
- az elemzés menete is azonos

11

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

$LALR(1)$ elemzés, $LALR(1)$ grammatika

- az $LALR(1)$ elemző táblázat kitöltése megegyezik az $LR(1)$ táblázat kitöltésével
- az elemzés menete is azonos

Definíció: $LALR(1)$ grammatika

Egy grammatika $LALR(1)$ grammatika, ha az $LALR(1)$ elemző táblázata konfliktusmentesen kitölthető.

A programnyelvek többsége leírható $LALR(1)$ nyelvtannal.

11

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

$LALR(1)$ elemző táblázat

$$\begin{aligned} \mathcal{K}_0 &= \{[S' \rightarrow .S, \#], [S \rightarrow .., \#], [S \rightarrow .(S)S, \#]\} \\ \mathcal{K}_1 &= \{[S' \rightarrow S., \#]\} \\ \mathcal{K}_2 &= \{[S \rightarrow (.S)S, \#], [S \rightarrow ..,]\}, [S \rightarrow .(S)S,]], [S \rightarrow .(S)S,)]\} \\ \mathcal{K}_3 &= \{[S \rightarrow (S).S, \#], [S \rightarrow ..,]\}, [S \rightarrow .(S)S,)]\} \\ \mathcal{K}_4 &= \{[S \rightarrow (S).S, \#], [S \rightarrow ..,], [S \rightarrow .(S)S, \#], \\ &\quad [S \rightarrow (S).S,)], [S \rightarrow ..,), [S \rightarrow .(S)S,)]\} \\ \mathcal{K}_5 &= \{[S \rightarrow (S)S, \#], [S \rightarrow (S).S,)]\} \end{aligned}$$

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3		léptetés, 4		
4	léptetés, 2	redukálás, $S \rightarrow \epsilon$	redukálás, $S \rightarrow \epsilon$	5
5		redukálás, $S \rightarrow (S)S$	redukálás, $S \rightarrow (S)S$	

12

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

SLR(1) vs. LALR(1)

SLR(1)	()	#	S
0	léptetés, 2	redukálás, $S \rightarrow \epsilon$	redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$	redukálás, $S \rightarrow \epsilon$	3
3		léptetés, 4		
4	léptetés, 2	redukálás, $S \rightarrow \epsilon$	redukálás, $S \rightarrow \epsilon$	5
5		redukálás, $S \rightarrow (S)S$	redukálás, $S \rightarrow (S)S$	

LALR(1)	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3		léptetés, 4		
4	léptetés, 2	redukálás, $S \rightarrow \epsilon$	redukálás, $S \rightarrow \epsilon$	5
5		redukálás, $S \rightarrow (S)S$	redukálás, $S \rightarrow (S)S$	

13

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

LALR(1) elemző hatékonyabb generálása

- nagyon időigényes létrehozni az összes LR(1) kanonikus halmazt és megkeresni közöttük az összevonhatókat

LALR(1) elemző hatékonyabb generálása

- nagyon időigényes létrehozni az összes LR(1) kanonikus halmazt és megkeresni közöttük az összevonhatókat
- egyszerűbb módszer:
 - az LR(0) kanonikus halmazokból indulunk (de azoknak is csak a lényeges elemeit fogjuk tárolni)
 - az előreolvasási szimbólumokat utólag határozzuk meg

14

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

A kanonikus halmazok törzse

Definíció: LR(0) kanonikus halmazok törzse

Egy LR(0) kanonikus halmaz törzse azokat az elmeket tartalmazza, amelyek nem a lezáras művelet hatására kerültek bele a halmazba. Az \mathcal{I}_0 halmaz esetén ez a $[S' \rightarrow .S]$ elemet, a többi halmaz esetén pedig azokat az elmeket jelenti, amelyekben a pont nem a szabály jobboldalának elején áll.

A kanonikus halmazok törzse

Definíció: LR(0) kanonikus halmazok törzse

Egy LR(0) kanonikus halmaz törzse azokat az elmeket tartalmazza, amelyek nem a lezáras művelet hatására kerültek bele a halmazba. Az \mathcal{I}_0 halmaz esetén ez a $[S' \rightarrow .S]$ elemet, a többi halmaz esetén pedig azokat az elmeket jelenti, amelyekben a pont nem a szabály jobboldalának elején áll.

Példa: helyes zárójelzés

$$\begin{aligned}\mathcal{I}_0 &= \{[S' \rightarrow .S], [S \rightarrow .], [S \rightarrow .(S)S]\} \\ \mathcal{I}_1 &= \{[S' \rightarrow S.\] \\ \mathcal{I}_2 &= \{[S \rightarrow (.S)S], [S \rightarrow .], [S \rightarrow .(S)S]\} \\ \mathcal{I}_3 &= \{[S \rightarrow (S.)S]\} \\ \mathcal{I}_4 &= \{[S \rightarrow (S).S], [S \rightarrow .], [S \rightarrow .(S)S]\} \\ \mathcal{I}_5 &= \{[S \rightarrow (S)S.\]\end{aligned}$$

15

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

A kanonikus halmazok törzse

- Jelölések:

- \mathcal{I}_j^* : az \mathcal{I}_j kanonikus halmaz törzse
- $read^*(\mathcal{I}_j^*, X)$: a $read(\mathcal{I}_j, X)$ törzse

Példa: helyes zárójelzés

$$\begin{aligned}\mathcal{I}_0^* &= \{[S' \rightarrow .S]\} \\ \mathcal{I}_1^* &= \{[S' \rightarrow S.\] \\ \mathcal{I}_2^* &= \{[S \rightarrow (.S)S]\} \\ \mathcal{I}_3^* &= \{[S \rightarrow (S.)S]\} \\ \mathcal{I}_4^* &= \{[S \rightarrow (S).S]\} \\ \mathcal{I}_5^* &= \{[S \rightarrow (S)S.\]\end{aligned}$$

16

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

- $\mathcal{I}_0^* = \{[S' \rightarrow .S]\}$
- ha $[A \rightarrow \alpha.X\beta] \in closure(\mathcal{I}_j^*)$, akkor
 $[A \rightarrow \alpha X.\beta] \in read^*(\mathcal{I}_j^*, X)$

- $\mathcal{I}_0^* = \{[S' \rightarrow .S]\}$
- ha $[A \rightarrow \alpha.X\beta] \in closure(\mathcal{I}_j^*)$, akkor
 $[A \rightarrow \alpha X.\beta] \in read^*(\mathcal{I}_j^*, X)$

Példa

Mivel $[S \rightarrow (.S)S] \in \mathcal{I}_2^*$ és $[S \rightarrow .(S)S] \in closure([S \rightarrow (.S)S])$, ezért $[S \rightarrow (.S)S] \in read(\mathcal{I}_2^*, ())$.

$$\mathcal{I}_0^* = \{[S' \rightarrow .S]\}$$

$$\begin{aligned}\mathcal{I}_0^* &= \{[S' \rightarrow .S]\} \\ \mathcal{I}_1^* &= read^*(\mathcal{I}_0^*, S) = \{[S' \rightarrow S.] \}\end{aligned}$$

$$\begin{aligned}\mathcal{I}_0^* &= \{[S' \rightarrow .S]\} \\ \mathcal{I}_1^* &= read^*(\mathcal{I}_0^*, S) = \{[S' \rightarrow S.] \} \\ \mathcal{I}_2^* &= read^*(\mathcal{I}_0^*, ()) = \{[S \rightarrow (.S)S] \}\end{aligned}$$

$$\begin{aligned}\mathcal{I}_0^* &= \{[S' \rightarrow .S]\} \\ \mathcal{I}_1^* &= read^*(\mathcal{I}_0^*, S) = \{[S' \rightarrow S.] \} \\ \mathcal{I}_2^* &= read^*(\mathcal{I}_0^*, ()) = \{[S \rightarrow (.S)S] \} \\ \mathcal{I}_3^* &= read^*(\mathcal{I}_2^*, S) = \{[S \rightarrow (S.)S] \}\end{aligned}$$

Példa

$$\begin{aligned}\mathcal{I}_0^* &= \{[S' \rightarrow .S]\} \\ \mathcal{I}_1^* &= \text{read}^*(\mathcal{I}_0^*, S) = \{[S' \rightarrow S.] \} \\ \mathcal{I}_2^* &= \text{read}^*(\mathcal{I}_0^*, ()) = \{[S \rightarrow (.S)S] \} \\ \mathcal{I}_3^* &= \text{read}^*(\mathcal{I}_2^*, S) = \{[S \rightarrow (S.)S] \} \\ \text{read}^*(\mathcal{I}_2^*, ()) &= \{[S \rightarrow (.S)S] \} = \mathcal{I}_2^*\end{aligned}$$

18

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

$$\begin{aligned}\mathcal{I}_0^* &= \{[S' \rightarrow .S]\} \\ \mathcal{I}_1^* &= \text{read}^*(\mathcal{I}_0^*, S) = \{[S' \rightarrow S.] \} \\ \mathcal{I}_2^* &= \text{read}^*(\mathcal{I}_0^*, ()) = \{[S \rightarrow (.S)S] \} \\ \mathcal{I}_3^* &= \text{read}^*(\mathcal{I}_2^*, S) = \{[S \rightarrow (S.)S] \} \\ \text{read}^*(\mathcal{I}_2^*, ()) &= \{[S \rightarrow (.S)S] \} = \mathcal{I}_2^* \\ \mathcal{I}_4^* &= \text{read}^*(\mathcal{I}_3^*, ()) = \{[S \rightarrow (S).S] \}\end{aligned}$$

18

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

$$\begin{aligned}\mathcal{I}_0^* &= \{[S' \rightarrow .S]\} \\ \mathcal{I}_1^* &= \text{read}^*(\mathcal{I}_0^*, S) = \{[S' \rightarrow S.] \} \\ \mathcal{I}_2^* &= \text{read}^*(\mathcal{I}_0^*, ()) = \{[S \rightarrow (.S)S] \} \\ \mathcal{I}_3^* &= \text{read}^*(\mathcal{I}_2^*, S) = \{[S \rightarrow (S.)S] \} \\ \text{read}^*(\mathcal{I}_2^*, ()) &= \{[S \rightarrow (.S)S] \} = \mathcal{I}_2^* \\ \mathcal{I}_4^* &= \text{read}^*(\mathcal{I}_3^*, ()) = \{[S \rightarrow (S).S] \} \\ \mathcal{I}_5^* &= \text{read}^*(\mathcal{I}_4^*, S) = \{[S \rightarrow (S)S.] \}\end{aligned}$$

18

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

$$\begin{aligned}\mathcal{I}_0^* &= \{[S' \rightarrow .S]\} \\ \mathcal{I}_1^* &= \text{read}^*(\mathcal{I}_0^*, S) = \{[S' \rightarrow S.] \} \\ \mathcal{I}_2^* &= \text{read}^*(\mathcal{I}_0^*, ()) = \{[S \rightarrow (.S)S] \} \\ \mathcal{I}_3^* &= \text{read}^*(\mathcal{I}_2^*, S) = \{[S \rightarrow (S.)S] \} \\ \text{read}^*(\mathcal{I}_2^*, ()) &= \{[S \rightarrow (.S)S] \} = \mathcal{I}_2^* \\ \mathcal{I}_4^* &= \text{read}^*(\mathcal{I}_3^*, ()) = \{[S \rightarrow (S).S] \} \\ \mathcal{I}_5^* &= \text{read}^*(\mathcal{I}_4^*, S) = \{[S \rightarrow (S)S.] \} \\ \text{read}^*(\mathcal{I}_4^*, ()) &= \{[S \rightarrow (.S)S] \} = \mathcal{I}_2^*\end{aligned}$$

18

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Örökolt és generált előreolvasási szimbólumok

- Mivel $[S' \rightarrow .S, \#] \in \mathcal{I}_0$, ezért $[S' \rightarrow S., \#] \in \text{read}(\mathcal{I}_0, S)$.
 - Ilyenkor a $\#$ szimbólum öröklődik az $[S' \rightarrow .S, \#]$ elemről az $[S' \rightarrow S., \#]$ elemre.

19

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Örökolt és generált előreolvasási szimbólumok

- Mivel $[S' \rightarrow .S, \#] \in \mathcal{I}_0$, ezért $[S' \rightarrow S., \#] \in \text{read}(\mathcal{I}_0, S)$.
 - Ilyenkor a $\#$ szimbólum öröklődik az $[S' \rightarrow .S, \#]$ elemről az $[S' \rightarrow S., \#]$ elemre.
- Mivel $[S \rightarrow (.S)S, \#] \in \mathcal{I}_2$, ezért $[S \rightarrow (S.)S,] \in \mathcal{I}_2$, és így $[S \rightarrow (S)S,] \in \text{read}(\mathcal{I}_2, ())$.
 - Ilyenkor a $)$ szimbólum spontán generálható a $[S \rightarrow (S)S,]$ elemhez.

19

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Előreolvasási szimbólumok meghatározása

- Legyen \emptyset egy olyan szimbólum, ami nem szerepel a grammatikában!
- Ha $[A \rightarrow \alpha.\beta] \in \mathcal{I}_j^*$, akkor határozzuk meg $closure([A \rightarrow \alpha.\beta, \emptyset])$ elemeit!

20

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Előreolvasási szimbólumok meghatározása

- Legyen \emptyset egy olyan szimbólum, ami nem szerepel a grammatikában!
- Ha $[A \rightarrow \alpha.\beta] \in \mathcal{I}_j^*$, akkor határozzuk meg $closure([A \rightarrow \alpha.\beta, \emptyset])$ elemeit!
 - Ha $[B \rightarrow \gamma.X\delta, a] \in closure([A \rightarrow \alpha.\beta, \emptyset])$ és $a \neq \emptyset$, akkor a $read^*(\mathcal{I}_j^*, X)$ halmaz $[B \rightarrow \gamma X.\delta]$ eleméhez az a spontán generálható.

20

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Előreolvasási szimbólumok meghatározása

- Legyen \emptyset egy olyan szimbólum, ami nem szerepel a grammatikában!
- Ha $[A \rightarrow \alpha.\beta] \in \mathcal{I}_j^*$, akkor határozzuk meg $closure([A \rightarrow \alpha.\beta, \emptyset])$ elemeit!
 - Ha $[B \rightarrow \gamma.X\delta, a] \in closure([A \rightarrow \alpha.\beta, \emptyset])$ és $a \neq \emptyset$, akkor a $read^*(\mathcal{I}_j^*, X)$ halmaz $[B \rightarrow \gamma X.\delta]$ eleméhez az a spontán generálható.
 - Ha $[B \rightarrow \gamma.X\delta, \emptyset] \in closure([A \rightarrow \alpha.\beta, \emptyset])$, akkor az előreolvasási szimbólumok öröklődnek az \mathcal{I}_j^* halmaz $[A \rightarrow \alpha.\beta]$ elemétől a $read^*(\mathcal{I}_j^*, X)$ halmaz $[B \rightarrow \gamma X.\delta]$ eleméhez.

20

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

Törzs	Elem	Spontán generálható	Honnan örökölt?
\mathcal{I}_0^*	$[S' \rightarrow .S]$		
\mathcal{I}_1^*	$[S' \rightarrow S.]$		
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$		
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$		
\mathcal{I}_4^*	$[S \rightarrow (S).S]$		
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$		

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Példa

$$closure([S' \rightarrow .S, \emptyset]) = \\ = \{[S' \rightarrow .S, \emptyset], [S \rightarrow .., \emptyset], [S \rightarrow .(S)S, \emptyset]\}$$

Törzs	Elem	Spontán generálható	Honnan örökölt?
\mathcal{I}_0^*	$[S' \rightarrow .S]$		
\mathcal{I}_1^*	$[S' \rightarrow S.]$	\mathcal{I}_0^* -beli $[S' \rightarrow .S]$	
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$	\mathcal{I}_0^* -beli $[S' \rightarrow .S]$	
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$		
\mathcal{I}_4^*	$[S \rightarrow (S).S]$		
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$		

22

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

$$closure([S \rightarrow (.S)S, \emptyset]) = \\ = \{[S \rightarrow (.S)S, \emptyset], [S \rightarrow ..], [S \rightarrow .(S)S,]\}$$

Törzs	Elem	Spontán generálható	Honnan örökölt?
\mathcal{I}_0^*	$[S' \rightarrow .S]$		
\mathcal{I}_1^*	$[S' \rightarrow S.]$		\mathcal{I}_0^* -beli $[S' \rightarrow .S]$
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$		\mathcal{I}_0^* -beli $[S' \rightarrow .S]$
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$		\mathcal{I}_2^* -beli $[S \rightarrow (.S)S]$
\mathcal{I}_4^*	$[S \rightarrow (S).S]$		
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$		

23

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

$$\text{closure}([S \rightarrow (S.)S], \emptyset) = \\ = \{[S \rightarrow (S.)S, \emptyset]\}$$

Törzs	Elem	Spontán generálható	Honnan örökölt?
\mathcal{I}_0^*	$[S' \rightarrow .S]$		
\mathcal{I}_1^*	$[S' \rightarrow S.]$		\mathcal{I}_0^* -beli $[S' \rightarrow .S]$
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$)	\mathcal{I}_0^* -beli $[S' \rightarrow .S]$
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$		\mathcal{I}_2^* -beli $[S \rightarrow (.S)S]$
\mathcal{I}_4^*	$[S \rightarrow (S).S]$		\mathcal{I}_3^* -beli $[S \rightarrow (S.)S]$
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$		

24

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Példa

$$\text{closure}([S \rightarrow (S).S], \emptyset) = \\ = \{[S \rightarrow (S).S, \emptyset], [S \rightarrow ., \emptyset], [S \rightarrow .(S)S, \emptyset]\}$$

Törzs	Elem	Spontán generálható	Honnan örökölt?
\mathcal{I}_0^*	$[S' \rightarrow .S]$		
\mathcal{I}_1^*	$[S' \rightarrow S.]$		\mathcal{I}_0^* -beli $[S' \rightarrow .S]$
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$)	\mathcal{I}_0^* -beli $[S' \rightarrow .S]$ \mathcal{I}_4^* -beli $[S \rightarrow (S).S]$
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$		\mathcal{I}_2^* -beli $[S \rightarrow (.S)S]$
\mathcal{I}_4^*	$[S \rightarrow (S).S]$		\mathcal{I}_3^* -beli $[S \rightarrow (S.)S]$
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$		\mathcal{I}_4^* -beli $[S \rightarrow (S).S]$

25

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Az előreolvasási szimbólumok meghatározása

- 1. menet:
 - az $[S' \rightarrow .S]$ elemhez felvesszük a # előreolvasási szimbólumot
 - minden elemhez felvesszük a spontán generálódó szimbólumait
- további menetek:
 - az öröklési szabályok szerint továbbterjesztjük a szimbólumokat

26

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Példa

Törzs	Elem	1. menet
\mathcal{I}_0^*	$[S' \rightarrow .S]$	#
\mathcal{I}_1^*	$[S' \rightarrow S.]$	
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$)
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$	
\mathcal{I}_4^*	$[S \rightarrow (S).S]$	
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$	

27

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Példa

Törzs	Elem	1. menet	2. menet
\mathcal{I}_0^*	$[S' \rightarrow .S]$	#	#
\mathcal{I}_1^*	$[S' \rightarrow S.]$		#
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$)), #
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$)
\mathcal{I}_4^*	$[S \rightarrow (S).S]$		
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$		

28

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Példa

Törzs	Elem	1. menet	2. menet	3. menet
\mathcal{I}_0^*	$[S' \rightarrow .S]$	#	#	#
\mathcal{I}_1^*	$[S' \rightarrow S.]$		#	#
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$)), #), #
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$)), #
\mathcal{I}_4^*	$[S \rightarrow (S).S]$)
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$			

29

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Törzs	Elem	Spontán generálható	Honnan örökölt?
\mathcal{I}_0^*	$[S' \rightarrow .S]$		
\mathcal{I}_1^*	$[S' \rightarrow S.]$		\mathcal{I}_0^* -beli $[S' \rightarrow .S]$
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$)	\mathcal{I}_0^* -beli $[S' \rightarrow .S]$, \mathcal{I}_4^* -beli $[S \rightarrow (S).S]$
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$		\mathcal{I}_2^* -beli $[S \rightarrow (.S)S]$
\mathcal{I}_4^*	$[S \rightarrow (S).S]$		\mathcal{I}_3^* -beli $[S \rightarrow (S.)S]$
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$		\mathcal{I}_4^* -beli $[S \rightarrow (S).S]$

Példa

Törzs	Elem	1. menet	2. menet	3. menet	4. menet
\mathcal{I}_0^*	$[S' \rightarrow .S]$	#	#	#	#
\mathcal{I}_1^*	$[S' \rightarrow S.]$		#	#	#
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$)), #), #), #
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$)), #), #	
\mathcal{I}_4^*	$[S \rightarrow (S).S]$)), #	
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$)

Törzs	Elem	Spontán generálható	Honnan örököl?
\mathcal{I}_0^*	$[S' \rightarrow .S]$		
\mathcal{I}_1^*	$[S' \rightarrow S.]$		\mathcal{I}_0^* -beli $[S' \rightarrow .S]$
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$)	\mathcal{I}_0^* -beli $[S' \rightarrow .S]$, \mathcal{I}_4^* -beli $[S \rightarrow (S.)S]$
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$		\mathcal{I}_2^* -beli $[S \rightarrow (.S)S]$
\mathcal{I}_4^*	$[S \rightarrow (S).S]$		\mathcal{I}_3^* -beli $[S \rightarrow (S.)S]$
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$		\mathcal{I}_4^* -beli $[S \rightarrow (S).S]$

Példa

Törzs	Elem	1. menet	2. menet	3. menet	4. menet	5. menet
\mathcal{I}_0^*	$[S' \rightarrow .S]$	#	#	#	#	#
\mathcal{I}_1^*	$[S' \rightarrow S.]$		#	#	#	#
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$)), #), #), #), #
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$)), #), #), #
\mathcal{I}_4^*	$[S \rightarrow (S).S]$)), #), #), #
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$))

Példa

Törzs	Elem	Előreolvasási szimbólumok
\mathcal{I}_0^*	$[S' \rightarrow .S]$	#
\mathcal{I}_1^*	$[S' \rightarrow S.]$	#
\mathcal{I}_2^*	$[S \rightarrow (.S)S]$), #
\mathcal{I}_3^*	$[S \rightarrow (S.)S]$), #
\mathcal{I}_4^*	$[S \rightarrow (S).S]$), #
\mathcal{I}_5^*	$[S \rightarrow (S)S.]$), #

$$\mathcal{K}_0^* = \{[S' \rightarrow .S, \#]\}$$

$$\mathcal{K}_1^* = \{[S' \rightarrow S., \#]\}$$

$$\mathcal{K}_2^* = \{[S \rightarrow (.S)S,], [S \rightarrow (.S)S, \#]\}$$

$$\mathcal{K}_3^* = \{[S \rightarrow (S.)S,], [S \rightarrow (S.)S, \#]\}$$

$$\mathcal{K}_4^* = \{[S \rightarrow (S).S,], [S \rightarrow (S).S, \#]\}$$

$$\mathcal{K}_5^* = \{[S \rightarrow (S)S.,][S \rightarrow (S)S., \#]\}$$

Így az egyesített kanonikus halmazok törzsét kaptuk meg!

LR(1) kanonikus halmazok törzse

Definíció: LR(1) kanonikus halmazok törzse

Egy (egyesített) LR(1) kanonikus halmaz törzse azokat az elmeket tartalmazza, amelyek nem a lezárást művelet hatására kerültek bele a halmazba.

Az \mathcal{I}_0 halmaz esetén ez a $[S' \rightarrow .S, \#]$ elemet, a többi halmaz esetén pedig azokat az elmeket jelenti, amelyekben a pont nem a szabály jobboldalának elején áll.

LR(1) kanonikus halmazok törzse

Definíció: LR(1) kanonikus halmazok törzse

Egy (egyesített) LR(1) kanonikus halmaz törzse azokat az elmeket tartalmazza, amelyek nem a lezárást művelet hatására kerültek bele a halmazba.

Az \mathcal{I}_0 halmaz esetén ez a $[S' \rightarrow .S, \#]$ elemet, a többi halmaz esetén pedig azokat az elmeket jelenti, amelyekben a pont nem a szabály jobboldalának elején áll.

- Cél: az egyesített kanonikus halmazok törzsei segítségével kitölteni az LALR(1) elemző táblázatot

Az elemző táblázat kitöltése

- $A \rightarrow \alpha$ redukció felismerése:

Az elemző táblázat kitöltése

- $A \rightarrow \alpha$ redukció felismerése:

- ha $\alpha \neq \epsilon$, akkor $[A \rightarrow \alpha, a]$ eleme a törzsnek
 \Rightarrow az a szimbólumhoz kell írni a redukciót

34

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Az elemző táblázat kitöltése

- $A \rightarrow \alpha$ redukció felismerése:

- ha $\alpha \neq \epsilon$, akkor $[A \rightarrow \alpha, a]$ eleme a törzsnek
 \Rightarrow az a szimbólumhoz kell írni a redukciót
- ha $\alpha = \epsilon$, akkor az olyan a szimbólumokhoz tartozik a redukció, amelyekre
 - $[B \rightarrow \beta, C\gamma, b]$ eleme az adott törzsnek,
 - $C \Rightarrow^* A\delta$, és
 - $a \in FIRST_1(\delta\gamma b)$.

34

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Az elemző táblázat kitöltése

- $A \rightarrow \alpha$ redukció felismerése:

- ha $\alpha \neq \epsilon$, akkor $[A \rightarrow \alpha, a]$ eleme a törzsnek
 \Rightarrow az a szimbólumhoz kell írni a redukciót
- ha $\alpha = \epsilon$, akkor az olyan a szimbólumokhoz tartozik a redukció, amelyekre
 - $[B \rightarrow \beta, C\gamma, b]$ eleme az adott törzsnek,
 - $C \Rightarrow^* A\delta$, és
 - $a \in FIRST_1(\delta\gamma b)$.
- az a szimbólumhoz a j állapotban léptetést kell előírni, ha
 - $read(\mathcal{K}_j^*, a) = \mathcal{K}_k^*$

34

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0				
1				
2				
3				
4				
5				

35

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2			
1				
2				
3				
4				
5				

$$read(\mathcal{K}_0^*, ()) = \mathcal{K}_2^*$$

36

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, S → ε	
1				
2				
3				
4				
5				

$$[S' \rightarrow .S, \#] = \mathcal{K}_2^* \text{ és } S \Rightarrow^* S \text{ és } \# \in FIRST_1(\#)$$

37

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1				
2				
3				
4				
5				

$$read(\mathcal{K}_0^*, S) = \mathcal{K}_1^*$$

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2				
3				
4				
5				

$$[S' \rightarrow S., \#] \in \mathcal{K}_1^*$$

38

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2			
3				
4				
5				

$$read(\mathcal{K}_2^*, ()) = \mathcal{K}_2^*$$

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		
3				
4				
5				

$$[S \rightarrow (\cdot S)S, \cdot] \in \mathcal{K}_2^* \text{ és } S \Rightarrow^* S \text{ és }) \in FIRST_1(\)S)$$

40

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3				
4				
5				

$$read(\mathcal{K}_2^*, S) = \mathcal{K}_3^*$$

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3		léptetés, 4		
4				
5				

$$read(\mathcal{K}_3^*,)) = \mathcal{K}_4^*$$

42

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

43

Fordítóprogramok előadás (A,C,T) LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3		léptetés, 4		
4	léptetés, 2			
5				

$$read(\mathcal{K}_4^*, ()) = \mathcal{K}_2^*$$

44

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3		léptetés, 4		
4	léptetés, 2	redukálás, $S \rightarrow \epsilon$		
5				

$$[S \rightarrow (S).S, ()] \in \mathcal{K}_4^* \text{ és } S \Rightarrow^* S \text{ és }) \in FIRST_1()$$

45

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3		léptetés, 4		
4	léptetés, 2	redukálás, $S \rightarrow \epsilon$	redukálás, $S \rightarrow \epsilon$	
5				

$$[S \rightarrow (S).S, \#] \in \mathcal{K}_4^* \text{ és } S \Rightarrow^* S \text{ és }) \in FIRST_1(\#)$$

46

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3		léptetés, 4		
4	léptetés, 2	redukálás, $S \rightarrow \epsilon$	redukálás, $S \rightarrow \epsilon$	5
5				

$$read(\mathcal{K}_4^*, S) = \mathcal{K}_5^*$$

47

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3		léptetés, 4		
4	léptetés, 2	redukálás, $S \rightarrow \epsilon$	redukálás, $S \rightarrow \epsilon$	5
5		redukálás, $S \rightarrow (S)S$		

$$[S \rightarrow (S)S., ()] \in \mathcal{K}_5^*$$

48

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

Példa

	()	#	S
0	léptetés, 2		redukálás, $S \rightarrow \epsilon$	1
1			OK	
2	léptetés, 2	redukálás, $S \rightarrow \epsilon$		3
3		léptetés, 4		
4	léptetés, 2	redukálás, $S \rightarrow \epsilon$	redukálás, $S \rightarrow \epsilon$	5
5		redukálás, $S \rightarrow (S)S$	redukálás, $S \rightarrow (S)S$	

$$[S \rightarrow (S)S., \#] \in \mathcal{K}_5^*$$

49

Fordítóprogramok előadás (A,C,T)

LR elemzések (LALR(1) elemzés)

- *Hibajelzés*: a táblázatok üres cellái hibarutinokra hivatkoznak
 - a hibaüzeneteket az adott nyelvtan és nyelv alapján kell kitalálni
 - nem automatizálható

- *Hibajelzés*: a táblázatok üres cellái hibarutinokra hivatkoznak
 - a hibaüzeneteket az adott nyelvtan és nyelv alapján kell kitalálni
 - nem automatizálható
- *Hibaelfedés*: az elemző szinkronizálja a vermet az inputtal és folytatja az elemzést

- ❶ Vezessünk be egy speciális terminális szimbólumot: *error*
- ❷ A „fontos” szabályokhoz (pl. kifejezés, utasítás, blokk) adjunk hozzá egy *hibaalternatívát*:

$$\text{Statement} \rightarrow \text{Assignment} ; | \dots | \text{error} ;$$

$$\text{Block} \rightarrow \text{begin StatementList end} | \text{begin error end}$$
- ❸ Az így kiegészített grammatikához készítsük el az elemzőt!

- ❶ Hiba detektálása esetén meghívja a megfelelő hibarutint.
- ❷ A verem tetejéről addig törl, amíg olyan állapotba nem kerül, ahol lehet az *error* szimbólummal lépni.
- ❸ A verembe lépteti az *error* szimbólumot.
- ❹ Az bemeneten addig ugorja át a soron következő terminálisokat, amíg a hibaalternatíva építését folytatni nem tudja.

Az if-then-else probléma

Fordítóprogramok előadás (A, C, T szakirány)

Az if-then-else probléma

C/C++ if utasítás

```
if(b1) if(b2) x++; else y++;
```

Az if-then-else probléma

C/C++ if utasítás

```
if(b1) if(b2) x++; else y++;
```

Mit jelent?

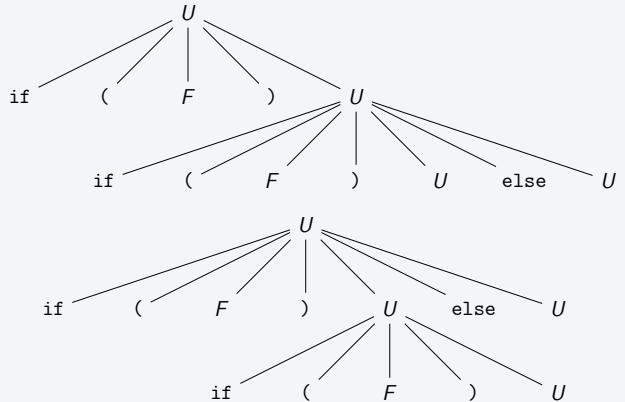
Egyik lehetőség

```
if(b1)
  if(b2)
    x++;
  else
    y++;
```

Másik lehetőség

```
if(b1)
  if(b2)
    x++;
  else
    y++;
```

Nyelvtan: $U \rightarrow \dots | if(F)U | if(F)U\ else U | \dots$



Nyelvtan: $U \rightarrow \dots | if(F)U | if(F)U\ else U | \dots$

Rekurzív leszállás

$U \rightarrow \dots | if(F)U | if(F)U\ else U | \dots$

```
void U()
{
  if( aktualis == if_szimbolum )
  {
    elfogad( if_szimbolum );
    elfogad( nyitozarojel );
    F();
    elfogad( csukozarojel );
    U();
    if( aktualis == else_szimbolum )
    {
      elfogad( else_szimbolum );
      U();
    }
  }
  else if( ... )
  ...
}
```

- A $if(F)if(S)U\ else U$ részmondathoz több szintaxisfa is tartozik.
- **Nem egyértelmű a nyelvtan!**
- Problémát okoz mindegyik elemző esetén.
- A gyakorlatban:

„Az else ág az őt közvetlenül megelőző if utasításhoz tartozik.”

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
    }  
    else if( ... )  
    ...  
}
```

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
    }  
    else if( ... )  
    ...  
}
```

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
    }  
    else if( ... )  
    ...  
}
```

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
    }  
    else if( ... )  
    ...  
}
```

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
    }  
    else if( ... )  
    ...  
}
```

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
    }  
    else if( ... )  
    ...  
}
```

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
        else if( ... )  
        ...  
    }  
}
```

12 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
        else if( ... )  
        ...  
    }  
}
```

13 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
        else if( ... )  
        ...  
    }  
}
```

14 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
        else if( ... )  
        ...  
    }  
}
```

15 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
        else if( ... )  
        ...  
    }  
}
```

16 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;  
void U()  
{  
    if( aktualis == if_szimbolum )  
    {  
        elfogad( if_szimbolum );  
        elfogad( nyitozarojel );  
        F();  
        elfogad( csukozarojel );  
        U();  
        if( aktualis == else_szimbolum )  
        {  
            elfogad( else_szimbolum );  
            U();  
        }  
        else if( ... )  
        ...  
    }  
}
```

17 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;
void U()
{
    if( aktualis == if_szimbolum )
    {
        elfogad( if_szimbolum );
        elfogad( nyitozarojel );
        F();
        elfogad( csukozarojel );
        U();
        if( aktualis == else_szimbolum )
        {
            elfogad( else_szimbolum );
            U();
        }
    }
    else if( ... )
    ...
}
```

18 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;
void U()
{
    if( aktualis == if_szimbolum )
    {
        elfogad( if_szimbolum );
        elfogad( nyitozarojel );
        F();
        elfogad( csukozarojel );
        U();
        if( aktualis == else_szimbolum )
        {
            elfogad( else_szimbolum );
            U();
        }
    }
    else if( ... )
    ...
}
```

19 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

Rekurzív leszállás

```
if(b1) if(b2) x++; else y++;
void U()
{
    if( aktualis == if_szimbolum )
    {
        elfogad( if_szimbolum );
        elfogad( nyitozarojel );
        F();
        elfogad( csukozarojel );
        U();
        if( aktualis == else_szimbolum )
        {
            elfogad( else_szimbolum );
            U();
        }
    }
    else if( ... )
    ...
}
```

20 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

LALR(1) elemzés

- Az LALR(1) elemzésnél konfliktushoz vezet az *if* utasítás előbb látott nyelvtana.

21 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

LALR(1) elemzés

- Az LALR(1) elemzésnél konfliktushoz vezet az *if* utasítás előbb látott nyelvtana.
- Az egyik kanonikus halmaz:
 $\mathcal{K}_i = \{[U \rightarrow \text{if } (F) U., \text{else}], [U \rightarrow \text{if } (F) U., \#], [U \rightarrow \text{if } (F) U. \text{ else } U., \text{else}], [U \rightarrow \text{if } (F) U. \text{ else } U., \#]\}$
- Ez egy léptetés-redukálás konfliktus.

LALR(1) elemzés

- Az LALR(1) elemzésnél konfliktushoz vezet az *if* utasítás előbb látott nyelvtana.
- Az egyik kanonikus halmaz:
 $\mathcal{K}_i = \{[U \rightarrow \text{if } (F) U., \text{else}], [U \rightarrow \text{if } (F) U., \#], [U \rightarrow \text{if } (F) U. \text{ else } U., \text{else}], [U \rightarrow \text{if } (F) U. \text{ else } U., \#]\}$
- Ez egy léptetés-redukálás konfliktus.
- Feloldása: léptetni kell!
 - Így az *else* az ót közvetlenül megelőző *if* utasításhoz fog tartozni.

21 Fordítóprogramok előadás (A, C, T szakirány) Az if-then-else probléma

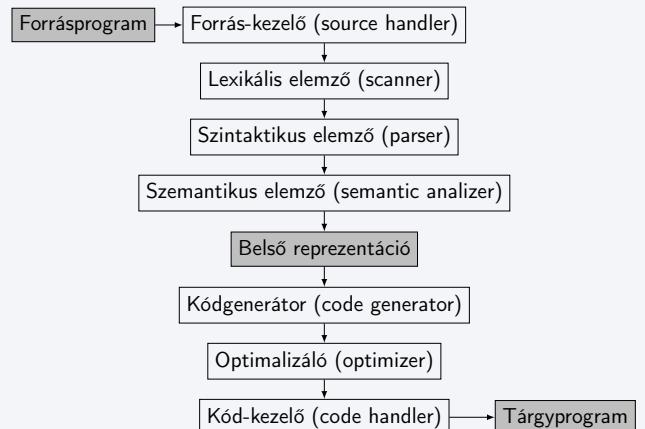
- Megoldás: az *if* utasítás végét jelző kulcsszó bevezetése.

```
    if(b1)
    {
if b1 then          if(b2)
                  {
if b2 then          x++;
                  {
else
                  y := y+1;
end if;
end if;           else
                  {
                  y++;
}
}
```

Szimbólumtábla-kezelés

Fordítóprogramok előadás (A, C, T szakirány)

Emlékeztető: a fordítás lépései



Információáramlás

Programszöveg

$x = y + 2;$

- Lexikális elemző: lexikális elemek sorozatát állítja elő

Lexikális elemek

változó operátor változó operátor számkonstans utasításvég

- Szintaktikus elemző: felépíti a szintaxisfát

Információáramlás

Szemantikus elemző:

- deklarálva van-e az x és y változó?
- x és $y + 2$ kifejezések típusa azonos-e?
- stb.

- Kódgenerálás: utasítások generálása a tárgyprogram nyelvén, amelyek megvalósítják az értékkedést.

Információáramlás

Szemantikus elemző:

- deklarálva van-e az x és y változó?
- x és $y + 2$ kifejezések típusa azonos-e?
- stb.

- Kódgenerálás: utasítások generálása a tárgyprogram nyelvén, amelyek megvalósítják az értékkedést.

A szemantikus elemzéshez és a kódgeneráláshoz nem elég a lexikális elemek sorozata és a szintaxis!

Információáramlás

A szemantikus elemzés és a kódgenerálás számára szükséges kiegészítő információk:

- konstansok értéke
- változók típusa
- függvények, operátorok szignatúrája
- a tárgykódba már bekerült változók, függvények címe
- részkifejezések típusa, hozzárendelt tárgykód
- stb.

Információáramlás

- A szemantikus elemzés és a kódgenerálás számára szükséges kiegészítő információk:
 - konstansok értéke
 - változók típusa
 - függvények, operátorok szignatúrája
 - a tárgykódba már bekerült változók, függvények címe
 - részkifejezések típusa, hozzárendelt tárgykód
 - stb.
- Ezeket az információkat két módon tároljuk:
 - **szimbólumtábla**
 - szemantikus értékek

Szimbólumtáblában tárolt információk

- **szimbólum neve**
- **szimbólum attribútumai:**
 - definíció adatai
 - típus
 - tárgyprogram-beli cím
 - definíció helye a forrásprogramban
 - szimbólumra hivatkozások a forrásprogramban

Szimbólum neve

- a szemantikus elemző és a kódgenerátor a szimbólumokat a nevükkel azonosítja
- a nevek általában változó hosszúságúak lehetnek
 - érdemes dinamikus memoriában tárolni
 - a tábla csak egy mutatót tartalmaz
 - pl. C++ `std :: string` választás esetén pont ez történik...

Definíció adatai

- **változó**
 - típus
 - módosító kulcsszavak: `const`, `static` ...
 - címe a tárgyprogramban (függ a változó tárolási módjától)

Definíció adatai

- **változó**
 - típus
 - módosító kulcsszavak: `const`, `static` ...
 - címe a tárgyprogramban (függ a változó tárolási módjától)
- **függvény, eljárás, operátor**
 - paraméterek típusa
 - visszatérési típus
 - módosítók
 - címe a tárgyprogramban

Definíció adatai

- **típus (típusleíró, típusdeszkriptor)**
 - egyszerű típusok: méret
 - rekord: mezők nevei és típusleírói
 - tömb: elem típusleírása, index típusleírása, méret
 - intervallum-típus: elem típusleírása, minimum, maximum
 - unio-típus: a lehetséges típusok leírói, méret

Tárgyprogram-beli cím

- A változók címét a definiáláskor határozza meg a fordító.
 - globális és statikus változók:
 - az adatszegmens kezdetétől számított relatív cím
 - az utoljára elhelyezett változó utáni szabad helyre
 - lokális változók (alprogramban vagy belső blokkban deklarálvá):
 - a veremben kap helyet
 - az aktuális veremkereten belüli relatív cím
 - az ebben a blokkban utoljára deklarált változó után
 - dinamikusan foglalt változók
 - a heap memoriában kapnak helyet
 - címük csak a program futásakor derül ki
 - mutatókkal hivatkozunk rájuk, csak azonak van „neve”
 - deklarált mutatók az előző két kategóriába esnek

Tárgyprogram-beli cím

- Az alprogramok tárgyprogram-beli címét is definiáláskor határozza meg a fordító.
 - kódszegmensen belüli relatív cím
 - az utoljára definiált alprogram utáni szabad helyre

Hivatkozások a szimbólumra

- deklaráció, definíció és hivatkozások sorainak száma a forrásprogramban
- hasznos lehet:
 - jó hibaüzenetek generálásához
 - kereszthivatkozás-lista létrehozásához
„Hol melyik változót használjuk?”)

(Túl) egyszerű példa

Forrásprogram

```

1. double d;
2. double fv( int i )
3. {
4.   int j = i*i;
5.   return d*j;
6. }
```

Név	Fajta	Típus	Param.	Def.	Hivatk.

(Túl) egyszerű példa

Forrásprogram

```

1. double d;
2. double fv( int i )
3. {
4.   int j = i*i;
5.   return d*j;
6. }
```

Név	Fajta	Típus	Param.	Def.	Hivatk.
"d"	változó	double		1	

(Túl) egyszerű példa

Forrásprogram

```

1. double d;
2. double fv( int i )
3. {
4.   int j = i*i;
5.   return d*j;
6. }
```

Név	Fajta	Típus	Param.	Def.	Hivatk.
"d"	változó	double		1	
"fv"	függvény	double	int	2	
"i"	paraméter	int		2	

(Túl) egyszerű példa

Forrásprogram

```
1. double d;  
2. double fv( int i )  
3. {  
4.     int j = i*i;  
5.     return d*j;  
6. }
```

Név	Fajta	Típus	Param.	Def.	Hivatk.
"d"	változó	double		1	
"fv"	függvény	double	int	2	
"i"	paraméter	int		2	4
"j"	változó	int		4	

(Túl) egyszerű példa

Forrásprogram

```
1. double d;  
2. double fv( int i )  
3. {  
4.     int j = i*i;  
5.     return d*j;  
6. }
```

Név	Fajta	Típus	Param.	Def.	Hivatk.
"d"	változó	double		1	5
"fv"	függvény	double	int	2	
"i"	paraméter	int		2	4
"j"	változó	int		4	5

A szimbólumtábla műveletei

- keresés
 - szimbólum használatakor
- beszúrás
 - új szimbólum megjelenésekor
 - tartalmaz egy keresést is: „Volt-e már deklárálva?”

Ezek hatékonysága nagy mértékben befolyásolja a fordítóprogram sebességét!

Hatókör, láthatóság, élettartam

- hatókör: „Ahhol a deklaráció érvényben van.”

Hatókör, láthatóság, élettartam

- hatókör: „Ahhol a deklaráció érvényben van.”
- láthatóság: „Ahhol hivatkozni lehet rá a nevével.”
 - része a hatókörnek
 - az elfedés miatt lehet kisebb, mint a hatókör

Hatókör, láthatóság, élettartam

- hatókör: „Ahhol a deklaráció érvényben van.”
- láthatóság: „Ahhol hivatkozni lehet rá a nevével.”
 - része a hatókörnek
 - az elfedés miatt lehet kisebb, mint a hatókör
- élettartam: „Amíg memóriaterület van hozzárendelve.”

Példa: hatókör

```
int x = 2;
cout << x << endl;
{
    cout << x << endl;
    int x = 3;
    cout << x << endl;
}
cout << x << endl;
```

Példa: láthatóság

```
int x = 2;
cout << x << endl;
{
    cout << x << endl;
    int x = 3;
    cout << x << endl;
}
cout << x << endl;
```

Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

Hatókör és láthatóság kezelése szimbólumtáblával

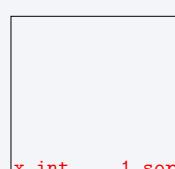
- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

1. int x = 2;
2. cout << x << endl;
3. {
4. cout << x << endl;
5. int x = 3;
6. cout << x << endl;
7. }
8. cout << x << endl;

Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

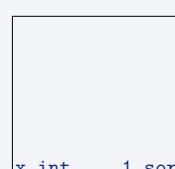
```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

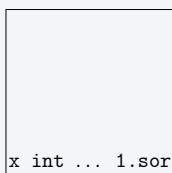
```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

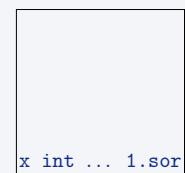
```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

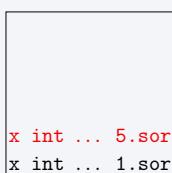
```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

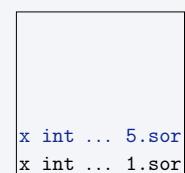
```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

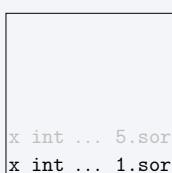
```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

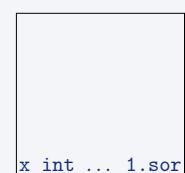
```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



Hatókör és láthatóság kezelése szimbólumtáblával

- A szimbólumokat egy verembe tesszük.
- Keresés:
 - a verem tetejéről indul
 - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

```
1. int x = 2;
2. cout << x << endl;
3. {
4.   cout << x << endl;
5.   int x = 3;
6.   cout << x << endl;
7. }
8. cout << x << endl;
```



Verem szimbólumtábla

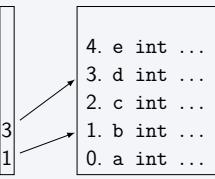
- „Meddig kell a szimbólumokat törölni a blokk végén?”
 - Számon kell tartani a blokk kezdetét a szimbólumtábla vermében!

Verem szimbólumtábla

- „Meddig kell a szimbólumokat törölni a blokk végén?”
 - Számon kell tartani a blokk kezdetét a szimbólumtábla vermében!
- Ötlet: **blokk-index vektor**
 - ez is egy verem
 - blokk kezdetén: *set*
 - az új blokk első szimbólumára mutató pointert teszünk a blokk-index vektorba
 - blokk végén: *reset*
 - a blokk-index vektor tetején lévő pointer mutatja, hogy meddig kell törölni az elemeket
 - a szimbólumok törlése után a blokk-index vektor legfelső elemét is törölni kell
- Név: **verem szimbólumtábla**

Példa

```
int a
{
    int b;
    int c;
    {
        int d;
        int e;
        ...
    }
}
```



Hatókonyság-növelés

- verem-szimbólumtábla:
 - keresés: lineáris
 - beszúrás (mivel tartalmaz egy keresést is): lineáris

Hatókonyság-növelés

- verem-szimbólumtábla:
 - keresés: lineáris
 - beszúrás (mivel tartalmaz egy keresést is): lineáris
- hatékonyabb adatszerkezetek:
 - kiegyensúlyozott fa:**
keresés és beszúrás is logaritmikus
 - hash-tábla:**
keresés és beszúrás is konstans műveletigényű
(jól megválasztott hash-függvény esetén)

Faszerkezetű szimbólumtábla

- Minden blokkhoz egy fa tartozik.
 - A szimbólumtábla sorai a fa csúcsaiban helyezkednek el.
- A fákat egy veremben tároljuk.
 - Új blokk kezdetén egy új (üres) fát teszünk a verembe.
 - A blokk szimbólumait ebbe a fába láncoljuk be.
 - Időnként kiegyensúlyozzuk a fát.
(Ha a nyelvben van külön deklarációs része a blokkoknak, akkor elég ennek a végén kiegyensúlyozni.)
 - A blokk végén a teljes fát törölni kell a veremből.
- Keresés:
 - A verem tetején lévő fában kezdjük.
 - Továbblépünk az előző fára, ha addig nem volt találat.
 - Az első előfordulásig keresünk.

Hash-szerkezetű szimbólumtábla

- szimbólumtábla sorai: veremben (mint a verem-szimbólumtábla esetén)
 - van blokk-index vektor is
- hash-függvény: a szimbólum nevét egy hash-értékre képezi le
- hash-tábla: a hash értékekhez hozzárendeli a legutóbbi ilyen hash-kódú szimbólum pozícióját a veremben
 - ha még nem volt ilyen kódú szimbólum: nullptr vagy speciális érték
 - ha több ilyen kódú szimbólum is van: láncolni kell őket a legutóbbitól a régebbiek felé

Keresés a hash-szerkezetű szimbólumtáblában

- a hash-függvénnel meghatározzuk a keresendő szimbólum hash-értékét
- a hash-táblából megkapjuk az ilyen kódú szimbólumok láncolt listáját
- végigmegyünk a listán az első találatig

Beszúrás a hash-szerkezetű szimbólumtáblába

- a szimbólumot (és attribútumait) a verem tetejére helyezzük
- a hash-függvénnel meghatározzuk a beszúrandó szimbólum hash-értékét
- a hash táblában a kapott hash értékhez bejegyezzük a verem-beli pozíciót
- ha volt már ilyen hash-kódú szimbólum, akkor az új szimbólumhoz beláncoljuk

Törlés a hash-szerkezetű szimbólumtáblából

- blokk végén a blokk-index vektor tetején lévő elem mutatja, hogy meddig kell törlni a szimbólumokat
- egy szimbólum törlése:
 - előállítjuk a hash-értékét
 - a hash-táblába a kapott hash-értékhez beírjuk a törlendő elemhez láncolt következő elem pozícióját (ha nincs hozzá láncolva elem, akkor a hash táblába az üres lista jelzés kerül)
 - eltávolítjuk az elemet a verem tetejéről
- a blokk összes szimbólumának törlése után a blokk-index vektor legfelső elemét is töröljük

Minősített nevek kezelése

```
namespace a
{
    int x = 1;
    namespace b
    {
        int x = 2;
        void print()
        {
            cout << a::x << x << endl;
        }
    }
}
```

Minősített nevek kezelése

```
namespace a
{
    int x = 1;
    namespace b
    {
        int x = 2;
        void print()
        {
            cout << a::x << x << endl;
        }
    }
}
```

A (külső) x és b szimbólumokhoz fel kell jegyezni a szimbólumtáblába, hogy az a névtérhez tartoznak.

Minősített nevek kezelése

```
namespace a
{
    int x = 1;
    namespace b
    {
        int x = 2;
        void print()
        {
            cout << a::x << x << endl;
        }
    }
}
```

A belső `x` és `print` szimbólumokhoz fel kell jegyezni a szimbólumtáblába, hogy a `b` névtérhez tartoznak.

Minősített nevek kezelése

```
namespace a
{
    int x = 1;
    namespace b
    {
        int x = 2;
        void print()
        {
            cout << a::x << x << endl;
        }
    }
}
```

A nem minősített nevű szimbólumok keresése nem változik.

Minősített nevek kezelése

```
namespace a
{
    int x = 1;
    namespace b
    {
        int x = 2;
        void print()
        {
            cout << a::x << x << endl;
        }
    }
}
```

Az `a :: x` szimbólum keresésekor olyan `x`-et kell keresni a szimbólumtáblában, amihez fel van jegyezve, hogy az a névtérben van.

Szimbólumok importálása

```
namespace a
{
    int x = 1;
}
using namespace a;
int main()
{
    cout << x << endl;
    return 0;
}
```

- A névterek szimbólumait a veremből nem törölni kell, hanem feljegyezni egy másik tárterületre.
- A `using` direktíva használatakor az importált névtér szimbólumait be kell másolni a verembe (vagy legalább hivatkozást tenni a verembe erre a névtérre).

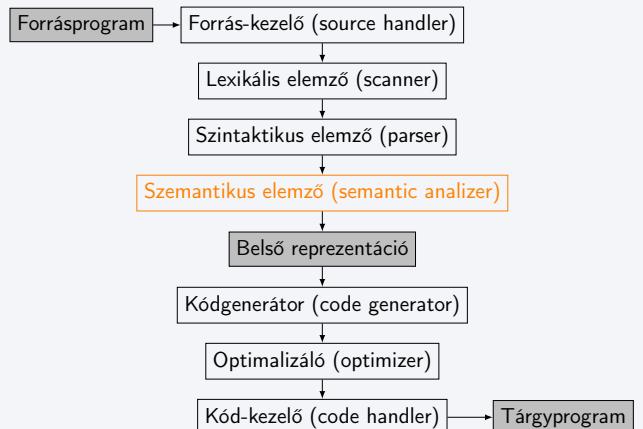
Osztályok

- A rekordokhoz hasonló: típusleírót kell készíteni hozzá.
- Láthatóság szabályozása: a mezőkhöz és tagfüggvényekhez fel kell jegyezni, hogy milyen a láthatóságuk (`public`, `protected`, `private`).
- Az osztályok névteret is alkotnak: (statikus) adattagjai minősített névvel is elérhetők, ilyenkor az előbb látott technikát lehet használni.

A szemantikus elemzés feladatai

Fordítóprogramok előadás (A, C, T szakirány)

A szemantikus elemzés helye



1 Fordítóprogramok előadás (A, C, T szakirány)

A szemantikus elemzés feladatai

2 Fordítóprogramok előadás (A, C, T szakirány)

A szemantikus elemzés feladatai

A szemantikus elemzés feladatai

A szemantikus elemzés jellemzően a környezetfüggő ellenőrzéseket valósítja meg.

- deklarációk kezelése
 - változók
 - függvények, eljárások, operátorok
 - típusok
- láthatósági szabályok
- aritmetikai ellenőrzések
- a program szintaxisának környezetfüggő részei
- típusellenőrzés
- stb.

A szemantikus elemzés erősen függ a konkrét programozási nyelvtől!

Deklarációk és láthatósági szabályok

Többszörös deklaráció

```
int x = 1;
cout << x;
int x = 2;
cout << x;
```

Elfedés

```
int x = 1;
cout << x;
{
    int x = 2;
    cout << x;
}
```

Adatelrejtés

```
class sajat
{
    public:
        int x;
    private:
        int y;
}
...
sajat s;
cout << s.x;
cout << s.y;
```

3 Fordítóprogramok előadás (A, C, T szakirány)

A szemantikus elemzés feladatai

4 Fordítóprogramok előadás (A, C, T szakirány)

A szemantikus elemzés feladatai

Deklarációk és láthatósági szabályok

- Jól megválasztott szimbólumtáblával megoldható:
 - verem szerkezetű szimbólumtábla (+ keresőfa vagy hash-tábla)
 - minden blokkhoz egy új szint a veremben
 - a keresés a verem tetejéről indul
- Lásd az előző előadás anyagát!

Aritmetikai ellenőrzések

- pl. a nullával osztás eseténként kiszűrhető:

Konstanssal osztás

```
a = b / 0;
// Itt lehet hibát vagy
// figyelmeztetést adni!
```

Változóval osztás

```
cin >> c;
a = b / c;
// Itt nem...
```

- hasonlóan kiszűrhető konstansok esetén a túl- vagy alulcsordulás

5 Fordítóprogramok előadás (A, C, T szakirány)

A szemantikus elemzés feladatai

6 Fordítóprogramok előadás (A, C, T szakirány)

A szemantikus elemzés feladatai

A szintaxis környezetfüggő részei

Példa: Ada eljárásdefiníciók

```
procedure eljarasom is
  ...
end eljarasom
```

A szintaxis környezetfüggő részei

Példa: Ada eljárásdefiníciók

```
procedure eljarasom is
  ...
end eljarasom
```

A környezetfüggetlen nyelvtan nem tudja leírni a szabályt:

EljDef → procedure Azonosito is Program end Azonosito

A szintaxis környezetfüggő részei

Példa: Ada eljárásdefiníciók

```
procedure eljarasom is
  ...
end eljarasom
```

A környezetfüggetlen nyelvtan nem tudja leírni a szabályt:

EljDef → procedure Azonosito is Program end Azonosito

Ez megengedi ezt a programszöveget is:

Hibás eljárásdefiníció

```
procedure egyik is
  ...
end masik
```

Ezt a hibát a szemantikus elemzésnek kell megtalálnia.

Típusellenőrzés

- első közelítésben a kifejezések típusozhatóságának ellenőrzése

s.length() + 1

```
s          :: string
s.length() :: int
1          :: int
s.length() + 1 :: int
```

s + 1

```
s      :: string
1      :: int
s + 1 :: hiba
```

Típusellenőrzés

- első közelítésben a kifejezések típusozhatóságának ellenőrzése

s.length() + 1

```
s          :: string
s.length() :: int
1          :: int
s.length() + 1 :: int
```

s + 1

```
s      :: string
1      :: int
s + 1 :: hiba
```

- mára nagyon kifejező típusrendszerek léteznek

- sablon (generikus) típusok
- altípusok
- öröklődés
- minél több információ tárolása a típusban
(pl. string hossza, gráf párossága stb.)

A típusok szerepe

- időnként a típushibás utasításokhoz nem lehet értelmesen kódot generálni

- pl. különböző méretű rekordok közötti értékadás

- a típusellenőrzés számos elírást felderít

```
string s;
if( s = 'hello' )
// ...
```

Statikus vs. dinamikus típusozás

- **Statikus:** a kifejezésekhez *fordítási időben* a szemantikus elemzés rendel típust
 - az ellenőrzések fordítási időben történnek
 - futás közben csak az értékeket kell tárolni
 - futás közben „nem történhet baj”
 - előny: biztonságosabb
 - pl.: Ada, C++, Haskell ...
- **Dinamikus:** a típusellenőrzés *futási időben* történik
 - futás közben az értékek mellett típusinformációt is kell tárolni
 - minden utasítás végrehajtása előtt ellenőrizni kell a típusokat
 - típushiba esetén futási idejű hiba keletkezik
 - előny: hajlékonyabb
 - pl.: Lisp, Erlang ...

10 Fordítóprogramok előadás (A, C, T szakirány) A szemantikus elemzés feladatai

Statikus és dinamikus típusozás

Bizonyos feladatokhoz használni kell a dinamikus típusellenőrzés technikáit:

- objektumorientált nyelvekben a *dinamikus kötés*
- Java *instanceof* operátora

Típusellenőrzés vs. típuslevezetés

```
C++  
int fac( int n )  
{  
    if( n == 0 )  
        return 1;  
    else  
        return n * fac(n-1);  
}
```

```
Haskell  
fac n =  
    if n == 0  
    then 1  
    else n * fac (n-1)
```

12 Fordítóprogramok előadás (A, C, T szakirány) A szemantikus elemzés feladatai

Típusellenőrzés vs. típuslevezetés

• Típusellenőrzés:

- minden típus a deklarációban adott
- a kifejezések egyszerű szabályok alapján típusozhatók
- egyszerűbb fordítóprogram, gyorsabb fordítás
- kényelmetlenebb a programozónak

13 Fordítóprogramok előadás (A, C, T szakirány) A szemantikus elemzés feladatai

Típusellenőrzés vs. típuslevezetés

• Típusellenőrzés:

- minden típus a deklarációban adott
- a kifejezések egyszerű szabályok alapján típusozhatók
- egyszerűbb fordítóprogram, gyorsabb fordítás
- kényelmetlenebb a programozónak

• Típuslevezetés, típuskikövetkeztetés:

- a változók, függvények típusait (általában) nem kell megadni
- a típusokat fordítóprogram „találja ki” a definíciójuk, használatuk alapján
- bonyolultabb fordítóprogram, lassabb fordítás
- kényelmesebb a programozónak

Automatikus típuskonverziók

```
void f1( double d ) {}  
void f2( int i ) {}  
int main()  
{  
    int i;  
    double d;  
    f1(i); // ez megy  
    f2(i);  
    f1(d);  
    f2(d); // ez problémás ...  
    return 0;  
}
```

Warning

warning: passing 'double' for argument 1
to 'void f2(int)',

13 Fordítóprogramok előadás (A, C, T szakirány) A szemantikus elemzés feladatai

14 Fordítóprogramok előadás (A, C, T szakirány) A szemantikus elemzés feladatai

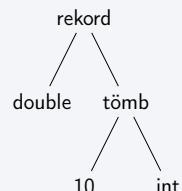
- C++ példa:
 - *int ~> double* automatikusan konvertálódik
 - *double ~> int* figyelmeztetést vált ki (elvesztjük a törrészt)
- ha nincs automatikus típuskonverzió:
 - ki kell írni: `f2((int)d);`
 - kényelmetlenebb, de biztonságosabb
- ha nagyon sok automatikus típuskonverzió van:
 - időnként meglepő eredmény születhet
 - kényelmesebb, de veszélyes lehet

- alaptípusok
 - pl. `int, double, char ...`
- összetett típusok
 - tömb
 - rekord (osztály ...)
 - unió

- alaptípusok
 - pl. `int, double, char ...`
- összetett típusok
 - tömb
 - rekord (osztály ...)
 - unió

Összetett típusok fa-struktúrája:

```
struct T
{
    double d;
    int t[10];
};
```



Mikor tekintsünk két típust ekvivalénsnek?

• szerkezeti (strukturális) ekvivalencia:

„Két típus egyenlő, ha az őket leíró fa azonos.”

- bonyolultabb az ellenőrzés
- nem minden fejezi ki a programozói szándékot
 - `struct Ember { string nev; int eletkor; };`
 - `struct Uzenet { string szoveg; int azonosito; };`

• név szerinti ekvivalencia:

„Két típus egyenlő, ha a nevük azonos.”

- egyszerűbb az ellenőrzés (`==`)
- időnként kényelmetlen lehet a programozónak

- altípus: általában valamilyen megszorítást tesz az alaptípusra

- pl. a természetes szám típus altípusa az egész szám típusnak

• származtatott típus:

- öröklődéssel jön létre az alaptípusból
- a specializáció sokféle lehet:
 - új adattag
 - új metódus
 - meglévő metódus felüldefiniálása

Upcast: Altípus vagy származtatott típus mindenhol használható, ahol az alaptípus megengedett.

Downcast: Az alaptípus általában nem használható a származtatott vagy altípus helyett.

Az öröklődési vagy altípus kapcsolatok a szokásos módon ábrázolhatók fával (többszörös öröklődés esetén gráffal):

**Upcast**

```

void f( Kutya k )
{ ... }

int main()
{
    Vizsla v;
    f( v ); // gond nélkül megy
}
  
```

Upcast

```

void f( Kutya k )
{ ... }

int main()
{
    Vizsla v;
    f( v ); // gond nélkül megy
}
  
```

Típusellenőrzés: a fában felfelé kell keresni, hogy van-e olyan ősosztály (alaptípus), ami használható az adott helyen.

Szemantikus elemzés (attribútum fordítási grammatikák)

Fordítóprogramok előadás (A, C, T szakirány)

A szemantikus elemzés elmélete

- a nyelvtan szabályait kiegészítjük a szemantikus elemzés tevékenységeivel
⇒ [fordítási grammatikák](#)

A szemantikus elemzés elmélete

- a nyelvtan szabályait kiegészítjük a szemantikus elemzés tevékenységeivel
⇒ [fordítási grammatikák](#)
- a nyelvtan szimbólumaihoz szemantikus típusokat (attribútumokat) rendelünk
⇒ [attribútum fordítási grammatikák](#)

A szemantikus elemzés elmélete

- a nyelvtan szabályait kiegészítjük a szemantikus elemzés tevékenységeivel
⇒ [fordítási grammatikák](#)
- a nyelvtan szimbólumaihoz szemantikus típusokat (attribútumokat) rendelünk
⇒ [attribútum fordítási grammatikák](#)
- megvizsgáljuk, hogy mikor lehet kiértékelni az attribútumértékeket
⇒ [jól definiált attribútum fordítási grammatikák](#)

A szemantikus elemzés elmélete

- a nyelvtan szabályait kiegészítjük a szemantikus elemzés tevékenységeivel
⇒ [fordítási grammatikák](#)
- a nyelvtan szimbólumaihoz szemantikus típusokat (attribútumokat) rendelünk
⇒ [attribútum fordítási grammatikák](#)
- megvizsgáljuk, hogy mikor lehet kiértékelni az attribútumértékeket
⇒ [jól definiált attribútum fordítási grammatikák](#)
- megszorításokat teszünk a grammikára, hogy a kiértékelés egyszerűbb legyen
⇒ [particionált attribútum fordítási grammatikák](#)
⇒ [rendezett attribútum fordítási grammatikák](#)

A szemantikus elemzés elmélete

- a nyelvtan szabályait kiegészítjük a szemantikus elemzés tevékenységeivel
⇒ [fordítási grammatikák](#)
- a nyelvtan szimbólumaihoz szemantikus típusokat (attribútumokat) rendelünk
⇒ [attribútum fordítási grammatikák](#)
- megvizsgáljuk, hogy mikor lehet kiértékelni az attribútumértékeket
⇒ [jól definiált attribútum fordítási grammatikák](#)
- megszorításokat teszünk a grammikára, hogy a kiértékelés egyszerűbb legyen
⇒ [particionált attribútum fordítási grammatikák](#)
⇒ [rendezett attribútum fordítási grammatikák](#)
- megvizsgáljuk, hogyan illeszthetők ezek a módszerek a tanult szintaktikus elemzőkhöz

- A grammatika szabályaiban jelöljük, hogy milyen szemantikus elemzési tevékenységekre van szükség.
 - Ezeket hívjuk **akciósziimbólumoknak**.
 - A nyelvtanban @ jelrel kezdődnek.
 - Az akciósziimbólumok által jelölt szemantikus tevékenységeket **szemantikus rutinoknak** nevezzük.

- A grammatika szabályaiban jelöljük, hogy milyen szemantikus elemzési tevékenységekre van szükség.
 - Ezeket hívjuk **akciósziimbólumoknak**.
 - A nyelvtanban @ jelrel kezdődnek.
 - Az akciósziimbólumok által jelölt szemantikus tevékenységeket **szemantikus rutinoknak** nevezzük.
- Az ilyen módon kiegészített grammatikát **fordítási grammatikának** nevezzük.

Példa I.

Értékadó utasítás

$\text{Értékadás} \rightarrow \text{Változó} := \text{Kifejezés } @\text{Ellenőrzés}$

Az $@\text{Ellenőrzés}$ akciósziimbólum a következő tevékenységet jelöli:

- ellenőrizni kell, hogy a változó típusa és a kifejezés típusa megfelelnek-e egymásnak
(egyenlők-e, vagy a kifejezés típusa konvertálható-e a változó típusára)
- ellenőrizni kell, hogy a változónak szabad-e értéket adni
(pl. nem konstans változó-e stb.)

Példa II.

Változódeklaráció

$\text{Deklaráció} \rightarrow \text{Típus } \text{Változó } @\text{Feljegyzés}$

A $@\text{Feljegyzés}$ akciósziimbólum a következő tevékenységet jelöli:

- ellenőrizni kell, hogy a deklaráció nem ütközik-e egy korábbival
(használni kell a szimbólumtáblát)
- a változó adatait be kell tenni a szimbólumtáblába

Attribútumok

- Hol találják a szemantikus rutinok a szükséges információkat?
Hova írják a kiszámolt eredményt?
 - a szemantikus rutinok önállóan is gondoskodhatnak a paraméterátadásukról
(például külön vermet kezelhetnek)
 - jobb megoldás: a szemantikus információkat a szintaktikus elemző szimbólumaihoz csatolva tároljuk

Attribútumok

- Hol találják a szemantikus rutinok a szükséges információkat?
Hova írják a kiszámolt eredményt?
 - a szemantikus rutinok önállóan is gondoskodhatnak a paraméterátadásukról
(például külön vermet kezelhetnek)
 - jobb megoldás: a szemantikus információkat a szintaktikus elemző szimbólumaihoz csatolva tároljuk
- A szimbólumokhoz **attribútumokat** rendelünk.
Ezek jelzik, hogy a szimbólumhoz milyen szemantikus értékek (attribútumértékek) kapcsolódnak.
- Jelölés: $A.x, y$
Az A szimbólumhoz az x és y attribútumokat rendeljük.

Példa I.

Kifejezés

$Kifejezés_0.t \rightarrow Kifejezés_1.t \pm Kifejezés_2.t @TípusEllenőrzés$
 $Kifejezés.t \rightarrow \underline{\text{konstans}.t} @\text{KonstansKifejezés}$

- A Kifejezés szimbólum különböző előfordulásait indexeléssel különböztetjük meg.

Példa I.

Kifejezés

$Kifejezés_0.t \rightarrow Kifejezés_1.t \pm Kifejezés_2.t @TípusEllenőrzés$
 $Kifejezés.t \rightarrow \underline{\text{konstans}.t} @\text{KonstansKifejezés}$

- A Kifejezés szimbólum különböző előfordulásait indexeléssel különböztetjük meg.
- A $@TípusEllenőrzés$ által jelzett szemantikus rutin:
 - $Kifejezés_0.t := \text{int}$

7

Fordítóprogramok előadás (A, C, T szakirány)

Szemantikus elemzés (attribútum fordítási grammatikák)

7

Fordítóprogramok előadás (A, C, T szakirány)

Szemantikus elemzés (attribútum fordítási grammatikák)

Példa I.

Kifejezés

$Kifejezés_0.t \rightarrow Kifejezés_1.t \pm Kifejezés_2.t @TípusEllenőrzés$
 $Kifejezés.t \rightarrow \underline{\text{konstans}.t} @\text{KonstansKifejezés}$

- A Kifejezés szimbólum különböző előfordulásait indexeléssel különböztetjük meg.
- A $@TípusEllenőrzés$ által jelzett szemantikus rutin:
 - $Kifejezés_0.t := \text{int}$
- A $@\text{KonstansKifejezés}$ által jelzett szemantikus rutin:
 - $Kifejezés.t := \underline{\text{konstans}.t}$

Példa I.

Kifejezés

$Kifejezés_0.t \rightarrow Kifejezés_1.t \pm Kifejezés_2.t @TípusEllenőrzés$
 $Kifejezés.t \rightarrow \underline{\text{konstans}.t} @\text{KonstansKifejezés}$

- A Kifejezés szimbólum különböző előfordulásait indexeléssel különböztetjük meg.
- A $@TípusEllenőrzés$ által jelzett szemantikus rutin:
 - $Kifejezés_0.t := \text{int}$
- A $@\text{KonstansKifejezés}$ által jelzett szemantikus rutin:
 - $Kifejezés.t := \underline{\text{konstans}.t}$
- Szemantikus ellenőrzések:
 $Kifejezés_1.t = \text{int}$ és $Kifejezés_2.t = \text{int}$ teljesülnek-e?

7

Fordítóprogramok előadás (A, C, T szakirány)

Szemantikus elemzés (attribútum fordítási grammatikák)

7

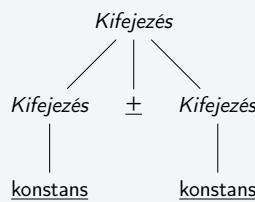
Fordítóprogramok előadás (A, C, T szakirány)

Szemantikus elemzés (attribútum fordítási grammatikák)

A szemantikus információ terjedése I.

- Példaszöveg: $1 + 2$

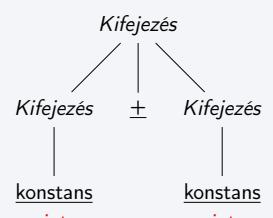
- Kezdetben a konstansok típusai ismertek.
- $Kifejezés.t := \underline{\text{konstans}.t}$
- $Kifejezés_0.t := \text{int}$
- Ellenőrzések:
 $Kifejezés_1.t = \text{int} ?$
 $Kifejezés_2.t = \text{int} ?$



A szemantikus információ terjedése I.

- Példaszöveg: $1 + 2$

- Kezdetben a konstansok típusai ismertek.
- $Kifejezés.t := \underline{\text{konstans}.t}$
- $Kifejezés_0.t := \text{int}$
- Ellenőrzések:
 $Kifejezés_1.t = \text{int} ?$
 $Kifejezés_2.t = \text{int} ?$



8

Fordítóprogramok előadás (A, C, T szakirány)

Szemantikus elemzés (attribútum fordítási grammatikák)

9

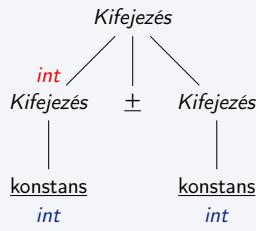
Fordítóprogramok előadás (A, C, T szakirány)

Szemantikus elemzés (attribútum fordítási grammatikák)

A szemantikus információ terjedése I.

- Példaszöveg: 1 + 2

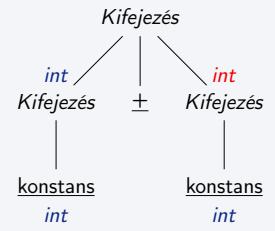
- Kezdetben a konstansok típusai ismertek.
- Kifejezés.t := konstans.t*
- Kifejezés₀.t := int*
- Ellenőrzések:
Kifejezés₁.t = int ?
Kifejezés₂.t = int ?



A szemantikus információ terjedése I.

- Példaszöveg: 1 + 2

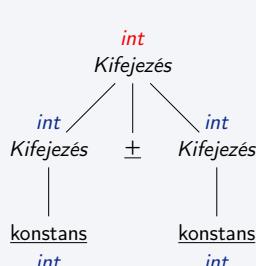
- Kezdetben a konstansok típusai ismertek.
- Kifejezés.t := konstans.t*
- Kifejezés₀.t := int*
- Ellenőrzések:
Kifejezés₁.t = int ?
Kifejezés₂.t = int ?



A szemantikus információ terjedése I.

- Példaszöveg: 1 + 2

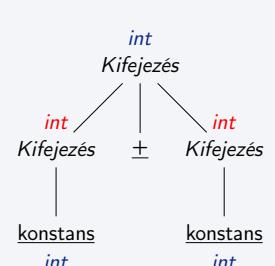
- Kezdetben a konstansok típusai ismertek.
- Kifejezés.t := konstans.t*
- Kifejezés₀.t := int*
- Ellenőrzések:
Kifejezés₁.t = int ?
Kifejezés₂.t = int ?



A szemantikus információ terjedése I.

- Példaszöveg: 1 + 2

- Kezdetben a konstansok típusai ismertek.
- Kifejezés.t := konstans.t*
- Kifejezés₀.t := int*
- Ellenőrzések:
Kifejezés₁.t = int ?
Kifejezés₂.t = int ?



Példa II.

Deklarációs lista

Deklaráció → típusnév.t Változólista.t @Beállít-1
 Változólista.t → változó.t Folytatás.t @Beállít-2
 Folytatás₀.t → vége | vessző változó.t Folytatás₁.t @Beállít-3

Példa II.

Deklarációs lista

Deklaráció → típusnév.t Változólista.t @Beállít-1
 Változólista.t → változó.t Folytatás.t @Beállít-2
 Folytatás₀.t → vége | vessző változó.t Folytatás₁.t @Beállít-3

- Kezdetben a típusnév.t attribútum értéke ismert egyedül.

Példa II.

Deklarációs lista

Deklaráció → típusnév.t Változólista.t @Beállít-1
 Változólista.t → változó.t Folytatás.t @Beállít-2
 Folytatás₀.t → vége | vessző változó.t Folytatás₁.t @Beállít-3

- Kezdetben a típusnév.t attribútum értéke ismert egyedül.
- @Beállít-1
 - Változólista.t := típusnév.t

Példa II.

Deklarációs lista

Deklaráció → típusnév.t Változólista.t @Beállít-1
 Változólista.t → változó.t Folytatás.t @Beállít-2
 Folytatás₀.t → vége | vessző változó.t Folytatás₁.t @Beállít-3

- Kezdetben a típusnév.t attribútum értéke ismert egyedül.
- @Beállít-1
 - Változólista.t := típusnév.t
- @Beállít-2
 - változó.t := Változólista.t és bejegyzés a szimbólumtáblába
 - Folytatás.t := Változólista.t

Példa II.

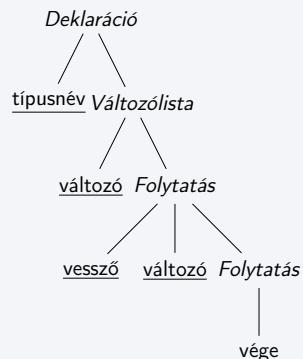
Deklarációs lista

Deklaráció → típusnév.t Változólista.t @Beállít-1
 Változólista.t → változó.t Folytatás.t @Beállít-2
 Folytatás₀.t → vége | vessző változó.t Folytatás₁.t @Beállít-3

- Kezdetben a típusnév.t attribútum értéke ismert egyedül.
- @Beállít-1
 - Változólista.t := típusnév.t
- @Beállít-2
 - változó.t := Változólista.t és bejegyzés a szimbólumtáblába
 - Folytatás.t := Változólista.t
- @Beállít-3
 - változó.t := Folytatás₀.t és bejegyzés a szimbólumtáblába
 - Folytatás₁.t := Folytatás₀.t

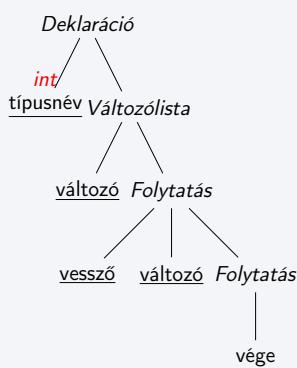
A szemantikus információ terjedése II.

- Példaszöveg: int a,b;



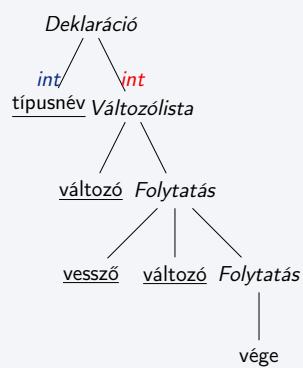
A szemantikus információ terjedése II.

- Példaszöveg: int a,b;
- Kezdetben a típusnév.t attribútum értéke ismert.
- Változólista.t := típusnév.t
- változó.t := Változólista.t
- Folytatás.t := Változólista.t
- változó.t := Folytatás₀.t
- Folytatás₁.t := Folytatás₀.t

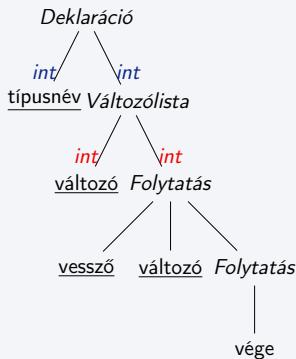


A szemantikus információ terjedése II.

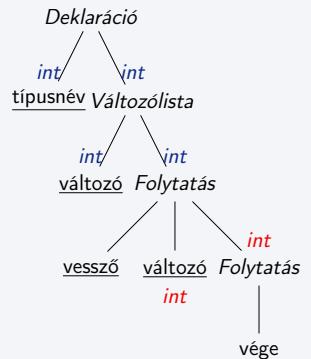
- Példaszöveg: int a,b;
- Kezdetben a típusnév.t attribútum értéke ismert.
- Változólista.t := típusnév.t
- változó.t := Változólista.t
- Folytatás.t := Változólista.t
- változó.t := Folytatás₀.t
- Folytatás₁.t := Folytatás₀.t



- Példaszöveg: int a,b;
 - Kezdetben a típusnév.t attribútum értéke ismert.
 - Változólista.t := típusnév.t
 - változó.t := Változólista.t
 - Folytatás.t := Változólista.t
 - változó.t := Folytatásq.t
 - Folytatás1.t := Folytatásq.t



- Példaszöveg: int a,b;
 - Kezdetben a típusnév.t attribútum értéke ismert.
 - Változólista.t := típusnév.t
 - változó.t := Változólista.t
 - Folytatás.t := Változólista.t
 - változó.t := Folytatás0.t
 - Folytatás1.t := Folytatás0.t



Attribútum fajták

- Szintetizált attribútum:
A helyettesítési szabály *bal oldalán* áll abban a szabályban, amelyikhez az őt kiszámoló szemantikus rutin tartozik.
 - például:
szabály: $Kifejezés_0.t \rightarrow Kifejezés_1.t \pm Kifejezés_2.t$
semantikus rutin: $Kifejezés_0.t := \text{int}$
 - Az információt a szintaxisfában *aljáról felfelé* közvetíti.

Attribútum fajták

- **Örökítő attribútum:**
A helyettesítési szabály jobb oldalán áll abban a szabályban, amelyikhez az őt kiszámoló szemantikus rutin tartozik.
 - például:
szabály: Változólista.t → változó.t *Folytatás.t*
semantikus rutin: változó.t := Változólista.t
 - Az információt a szintaxisfában *felülről lefelé* közvetíti.

Attribútum fajták

- **Kitüntetett szintetizált attribútum:**
Olyan attribútumok, amelyek terminális szimbólumokhoz tartoznak és kiszámításukhoz nem használunk fel más attribútumokat.
 - például:
szabály: *Kifejezés*. $t \rightarrow \text{konstans}.$ *t*
 - Az információt általában a lexikális elemző szolgáltatja

Attribútum fordítási grammatika (ATG)

- Egészítsük ki egy fordítási grammatika szimbólumait *attribútumokkal*, a szabályokat *feltételekkel*, valamint rendeljünk *szemantikus rutinokat* az akciószimbólumokhoz a következő szabályok betartásával:

Attribútum fordítási gramatika (ATG)

- Egészítük ki egy fordítási gramatika szimbólumait *attribútumokkal*, a szabályokat *feltételekkel*, valamint rendeljünk *szemantikus rutinokat* az akciósimbólumokhoz a következő szabályok betartásával:
 - Egy adott szabályhoz tartozó feltételek csak a szabályban előforduló attribútumoktól függhetnek.
(Ha egy feltétel nem teljesül, akkor szemantikus hibát kell jelezni!)

Attribútum fordítási gramatika (ATG)

- Egészítük ki egy fordítási gramatika szimbólumait *attribútumokkal*, a szabályokat *feltételekkel*, valamint rendeljünk *szemantikus rutinokat* az akciósimbólumokhoz a következő szabályok betartásával:
 - Egy adott szabályhoz tartozó feltételek csak a szabályban előforduló attribútumoktól függhetnek.
(Ha egy feltétel nem teljesül, akkor szemantikus hibát kell jelezni!)
 - A szemantikus rutinok csak annak a szabálynak az attribútumait használhatják és számíthatják ki, amelyikhez az őket reprezentáló akciósimbólum tartozik.

23 Fordítóprogramok előadás (A, C, T szakirány)

Szemantikus elemzés (attribútum fordítási gramatikák)

23 Fordítóprogramok előadás (A, C, T szakirány) Szemantikus elemzés (attribútum fordítási gramatikák)

Attribútum fordítási gramatika (ATG)

- Egészítük ki egy fordítási gramatika szimbólumait *attribútumokkal*, a szabályokat *feltételekkel*, valamint rendeljünk *szemantikus rutinokat* az akciósimbólumokhoz a következő szabályok betartásával:
 - Egy adott szabályhoz tartozó feltételek csak a szabályban előforduló attribútumoktól függhetnek.
(Ha egy feltétel nem teljesül, akkor szemantikus hibát kell jelezni!)
 - A szemantikus rutinok csak annak a szabálynak az attribútumait használhatják és számíthatják ki, amelyikhez az őket reprezentáló akciósimbólum tartozik.
 - Minden szintaxisfában minden attribútumértéket pontosan egy szemantikus rutin határozhat meg.

Így **attribútum fordítási gramatikát** kapunk.

Kiszámíthatóság

- Nem minden attribútum fordítási gramatikában lehet kiszámolni az összes attribútum értékét.
 - Például ha $X.a$ kiszámításához szükség van $Y.b$ -re és $Y.b$ kiszámításához szükség van $X.a$ -ra...

Definíció: Jól definiált attribútum fordítási gramatika

Olyan attribútum fordítási gramatika, amelyre igaz, hogy a gramatika által definiált nyelv mondataihoz tartozó minden szintaxisfában minden attribútum értéke egyértelműen kiszámítható.

23 Fordítóprogramok előadás (A, C, T szakirány)

Szemantikus elemzés (attribútum fordítási gramatikák)

24 Fordítóprogramok előadás (A, C, T szakirány) Szemantikus elemzés (attribútum fordítási gramatikák)

Direkt függőségek

Definíció: Direkt attribútumfüggőségek

Ha az $Y.b$ attribútumot kiszámoló szemantikus rutin használja az $X.a$ attribútumot, akkor $(X.a, Y.b)$ egy **direkt attribútumfüggőség**. Ezek a függőségek a *függőségi gráfban* ábrázolhatók.

Direkt függőségek

Definíció: Direkt attribútumfüggőségek

Ha az $Y.b$ attribútumot kiszámoló szemantikus rutin használja az $X.a$ attribútumot, akkor $(X.a, Y.b)$ egy **direkt attribútumfüggőség**. Ezek a függőségek a *függőségi gráfban* ábrázolhatók.

Jól definiált attribútum fordítási gramatikákhoz tartozó szintaxisfák függőségi gráfjában biztosan *nincsenek körök!*

25 Fordítóprogramok előadás (A, C, T szakirány)

Szemantikus elemzés (attribútum fordítási gramatikák)

25 Fordítóprogramok előadás (A, C, T szakirány) Szemantikus elemzés (attribútum fordítási gramatikák)

Egy attribútumkiértékelő algoritmus

Nemdeterminisztikus algoritmus:

- a szintaxisfa minden attribútumához egy folyamatot rendelünk
- a folyamat figyeli, hogy az adott attribútum kiértékeléséhez szükséges összes attribútum értékét meghatározta-e már a többi folyamat
- ha igen, akkor a folyamat kiszámítja az adott attribútum értékét

Egy attribútumkiértékelő algoritmus

Nemdeterminisztikus algoritmus:

- a szintaxisfa minden attribútumához egy folyamatot rendelünk
- a folyamat figyeli, hogy az adott attribútum kiértékeléséhez szükséges összes attribútum értékét meghatározta-e már a többi folyamat
- ha igen, akkor a folyamat kiszámítja az adott attribútum értékét

Ha az attribútum fordítási grammaтика jól definiált, akkor ez a program biztosan terminál és kiszámítja az összes attribútumértéket.

De ennél hatékonyabb algoritmust szeretnénk...

Particionált ATG

- A kiértékelés sorrendjét általában nem lehet az attribútum-fordítási grammaтикаból meghatározni. (Függet a konkrét szintaxisfától.)

Particionált ATG

- A kiértékelés sorrendjét általában nem lehet az attribútum-fordítási grammaтикаból meghatározni. (Függet a konkrét szintaxisfától.)
- **Particionált attribútum fordítási grammaтика:** minden X szimbólum attribútumai szétoszthatók az $A_1^X, A_2^X, \dots, A_{m_X}^X$ diszjunkt halmazokba (*partíciókba*) úgy, hogy
 - $A_{m_X}^X, A_{m_X-2}^X, \dots$ halmazokban csak szintetizált attribútumok
 - $A_{m_X-1}^X, A_{m_X-3}^X, \dots$ halmazokban csak örökolt attribútumok vannak, és
 - minden szintaxisfában az X attribútumai a halmazok növekvő sorrendjében meghatározhatók.
- Ha ismerjük a particiókat, akkor csak az attribútum fordítási grammaтика szabályai alapján tudunk olyan programot írni, ami a megfelelő sorrendben kiszámítja a szintaxisfában lévő összes attribútumot.

Rendezett ATG

- A particiókat általában nem könnyű meghatározni.

Rendezett ATG

- A particiókat általában nem könnyű meghatározni.
- **Rendezett attribútum fordítási grammaтика:** egyszerű algoritmussal meghatározhatók a particiók.
 - a halmazokat fordított sorrendben határozzuk meg
 - felváltva szintetizált és örökolt attribútumokat teszünk a halmazokba (utolsóba szintetizált, utolsó előtérbe örökolt, stb.)
 - mindegyikbe azokat az attribútumokat tessük, amikre csak olyan attribútumok kiszámításához van szükség, amelyeket már beosztottunk valamelyik halmazba
- Ha ezzel az algoritmussal megfelelő particiókat kapunk, akkor hívjuk a nyelvtant *rendezett attribútum fordítási grammaтикаnak*.

- Olyan attribútum fordítási grammaika, amelyben kizárolag szintetizált attribútumok vannak.
- A szemantikus információ a szintaxisfában a levelektől a gyökér felé terjed.
- Jól illeszthető az alulról felfelé elemzésekhez!

2
int

- léptetés
- kitüntetett szintetizált attribútum: konstans típusa

$$\begin{array}{c} \textit{int} \\ E \\ | \\ 2 \\ \textit{int} \end{array}$$

- redukció $E.t \rightarrow \underline{\text{konstans}}.t$ szabály alapján
- szemantikus rutin: $E.t := \underline{\text{konstans}}.t$

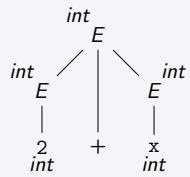
$$\begin{array}{c} \textit{int} \\ E \\ | \\ 2 \\ \textit{int} \\ + \end{array}$$

- léptetés

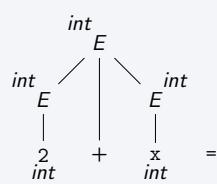
$$\begin{array}{ccc} \textit{int} & & \textit{int} \\ E & & E \\ | & & | \\ 2 & + & x \\ \textit{int} & \textit{int} & \textit{int} \end{array}$$

- léptetés
- kitüntetett szintetizált attribútum: változó típusa

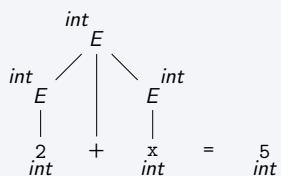
- redukció $E.t \rightarrow \underline{\text{változó}}.t$ szabály alapján
- szemantikus rutin: $E.t := \underline{\text{változó}}.t$



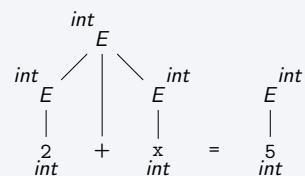
- redukció $E_0.t \rightarrow E_1.t \pm E_2.t$ szabály alapján
- szemantikus rutin: $E_0.t := \text{int}$
- ellenőrzés: $E_1.t = \text{int}$ és $E_2.t = \text{int}$?



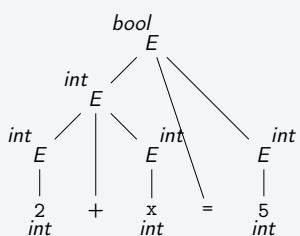
- léptetés



- léptetés
- kitüntetett szintetizált attribútum: konstans típusa



- redukció $E.t \rightarrow \underline{\text{konstans}.t}$ szabály alapján
- szemantikus rutin: $E.t := \underline{\text{konstans}.t}$



- redukció $E_0.t \rightarrow E_1.t \equiv E_2.t$ szabály alapján
- szemantikus rutin: $E_0.t := \text{bool}$
- ellenőrzés: $E_1.t = E_2.t$?

- redukció esetén a szabály jobb oldalának attribútumértékei már ismertek
(hiszen nincsenek örökölt attribútumok)
- a szabályhoz rendelt szemantikus rutin feladata a baloldalon álló szimbólum szintetizált attribútumainak meghatározása

Örökolt attribútumok és az alulról felfelé elemzés

- esetenként a nyelvtan átalakítása segíthet kiküszöbölni az örökolt attribútumokat
- szimbólumtáblában vagy más globális változóban tárolható az örökítendő attribútumérték
- eltárolható az a részfa, ahol örökolt attribútumok vannak; mikor meghatározható az értékük, akkor be kell járni a részfát és pótolni a hiányzó attribútumokat; ez a részfa gyökeréhez tartozó szabály szemantikus rutinjában implementálható

L-attribútum fordítási grammatikák

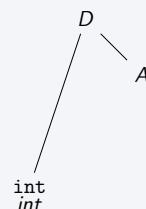
- Olyan attribútum fordítási grammatika, amelyben minden $A \rightarrow X_1 X_2 \dots X_n$ szabályban az attribútumértékek az alábbi sorrendben meghatározhatók:
 - A örökolt attribútumai
 - X_1 örökolt attribútumai
 - X_1 szintetizált attribútumai
 - X_2 örökolt attribútumai
 - X_2 szintetizált attribútumai
 - ...
 - X_n örökolt attribútumai
 - X_n szintetizált attribútumai
 - A szintetizált attribútumai
- Jól illeszkedik a felülről lefelé elemzésekhez.

L-ATG és a felülről lefelé elemzés

D

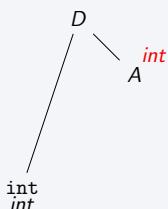
- D -ből indulva felülről lefelé levezetünk egy deklarációt
- D -nek most nincs örökolt attribútuma

L-ATG és a felülről lefelé elemzés



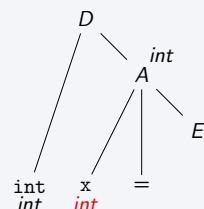
- $D \rightarrow \underline{\text{típusnév}.d} A.d$ szabály alkalmazása
- kitüntetett szintetizált attribútum: a típusnév által azonosított típus

L-ATG és a felülről lefelé elemzés

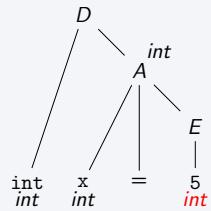


- szemantikus rutin: $A.d := \underline{\text{típusnév}.d}$
- ez egy örökolt attribútum

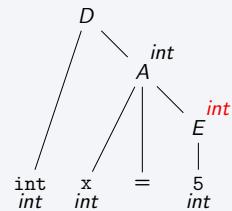
L-ATG és a felülről lefelé elemzés



- $A.d \rightarrow \underline{\text{változó}.d} = E.t$ szabály alkalmazása
- szemantikus rutin: $\underline{\text{változó}.d} := A.d$ (örökolt)



- $E.t \rightarrow \text{konstans}.t$ szabály alkalmazása
- kitüntetett szintetizált attribútum: konstans típusa



- szemantikus rutin: $E.t := \text{konstans}.t$ (szintetizált)
- ellenőrzés: $E.t = A.d$?

- a $A \rightarrow X_1 \dots X_n$ szabályhoz tartozó eljárás
 - formális paraméterei legyenek az A örökölt attribútumai
 - visszatérési értéke az A szintetizált attribútumai
- az eljárások végrehajtása épp az L-ATG attribútumkiértékelési sorrendjét adja

Gyakorlatias assembly bevezető

Fordítóprogramok előadás (A, C, T szakirány)

Mi az assembly?

- programozási nyelvek egy csoportja
- gépközeli:
 - az adott processzor utasításai használhatóak
 - általában nincsenek programkonstrukciók, típusok, osztályok stb.
- a futtatható programban pontosan azok az utasítások lesznek, amit a programba írunk
 - lehet optimalizálni
 - lehet olyan trükköket használni, amit a magasabb szintű nyelvek nem engednek meg

Sokfélé assembly van...

- különféle architektúrák (processzorok) utasításkészlete különbözik
 - van olyan architektúra, ahol gépi utasítás van külön egy bináris keresőfa kiegysúlyozására
 - Intel architektúrán ilyen nincs...
- egy architektúrán belül is lehet különböző szintaxisú assembly nyelvet definiálni
 - NASM assembly: `mov eax,0`
 - GAS assembly: `movl $0, %eax`

Mit fogunk mi használni?

- Intel x86 architektúra (386, 486, Pentium, ...)
 - mindenki számára könnyen elérhető
- NASM assembly
 - tiszta, könnyen érthető a szintaxisa

Assembly programok fordítása

- az assembly programok fordítóprogramja az *assembler*
- a magas szintű nyelvek fordítóprogramjaihoz képest:
 - az elemzés része egyszerűbb
 - általában nincsenek típusok, bonyolult szintaktikus elemek stb.
 - a kódgenerálás bonyolultabb
 - az assembler gépi kódra fordít
 - a többi fordítóprogram általában assemblyre vagy egy másik magasszintű nyelvre

A NASM fordítóprogramja

A fordítás lépései

```
$ ls  
io.c programom.asm  
$ nasm -f elf -o programom.o programom.asm  
$ ls  
io.c programom.asm programom.o  
$ gcc -o programom programom.o io.c  
$ ls  
io.c programom programom.asm programom.o
```

- nasm: fordítás (asszembblálás)
- gcc: szerkesztés (linkelés)
(a C fordítóprogramját használjuk)
- io.o: egy object file, amit C-ben írtam és az ebben lévő függvényeket meghívjuk a majd programunkból

Az asszembolás

```
$ nasm -f elf -o programom.o programom.asm
```

- **-f elf:** megadjuk az object fájl formátumát (elf)
- **-o programom.o:** az eredményként keletkező object fájl neve
- egyéb kapcsolók: nasm -h

regiszterek

- kis méretű tárolóhelyek a processzoron
- külön nevük van: eax, ebx, ecx, ...

memória

- itt tárolhatók a program által használt adatok
- itt tárolódik maga a program kódja is
- címzéssel lehet hivatkozni rá: [tömb+4]

Regiszterek adattároláshoz, számoláshoz

- eax: „*accumulator*” - elsősorban aritmetikai számításokhoz
- ebx: „*base*” - ebben szokás tömbök, rekordok kezdőcímét tárolni
- ecx: „*counter*” - számlálókat szokás tárolni benne (pl. for jellegű ciklusokhoz)
- edx: „*data*” - egyéb adatok tárolása; aritmetikai számításokhoz segédregiszter
- esi: „*source index*” - sztringmásolásnál a forrás címe
- edi: „*destination index*” - sztringmásolásnál a cél címe

Ezek egymással felcserélhetők, de célszerű a megadott feladatokra használni őket, ha lehet.

Regiszterek a program vermének kezeléséhez

- esp: „*stack pointer*” - a program verménék a tetejét tartja nyilván
- ebp: „*base pointer*” - az aktuális alprogramhoz tartozó verem-részt tartja nyilván

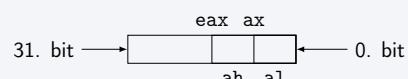
Ezeket csak a veremkezeléshez érdemes használni. Meggondolatlan elállításuk hibához vezethet!

Adminisztratív regiszterek

- eip: „*instruction pointer*” - a következő végrehajtandó utasítás címe
- eflags: *jelzőbitek* - pl. összehasonlítások eredményeinek tárolása

Ezeket közvetlenül nem tudjuk olvasni és írni.

A regiszterek felépítése



- eax 32 bitból áll; két része van:
 - a „felső” 16 bitnek nincs külön neve
 - az „alsó” 16 bit: ax; ennek részei:
 - a „felső” bájt: ah
 - az „alsó” bájt: al
- ugyanígy épülnek fel: ebx, ecx, edx
- esi, edi, esp és ebp regiszterekben csak az alsó 16 bitnek van külön neve: si, di, sp, bp

Az adatterület megadása

- a programban helyet foglalunk az adatainak:
 - ha csak helyet akarunk foglalni nekik: .bss szakasz
 - ha kezdeti értéket is akarunk adni: .data szakasz

Adatterület megadása

```
section .bss  
A: resb 4      ; négy bájt lefoglalása  
B: resb 2      ; két bájt lefoglalása  
  
section .data  
C: db 1,2,3,4  ; négyeszer 1 bájt  
           ; az 1, 2, 3, 4 értékekkel  
D: dd 42       ; egyszer 4 bájt a 42 értékkel
```

Az adatterület megadása

Adatterület megadása

```
section .bss  
A: resb 4      ; négy bájt lefoglalása  
B: resb 2      ; két bájt lefoglalása  
  
section .data  
C: db 1,2,3,4  ; négyeszer 1 bájt  
           ; az 1, 2, 3, 4 értékekkel  
D: dd 42       ; egyszer 4 bájt a 42 értékkel
```

A	A+1	A+2	A+3	B	B+1		
?	?	?	?	?	?		
C	C+1	C+2	C+3	D	D+1	D+2	D+3
1	2	3	4	42	0	0	0

Adatterület megadása

- resb: „reserve byte” - egy bájt
- resw: „reserve word” - két bájt (egy gépi szó)
- resd: „reserve double word” - négy bájt (egy gépi duplaszó)
- db: „define byte” - egy bájt kezdőértékkel
- dw: „define word” - két bájt kezdőértékkel
- dd: „define double word” - négy bájt kezdőértékkel

Memória hivatkozás

C	C+1	C+2	C+3	D	D+1	D+2	D+3
1	2	3	4	42	0	0	0

- byte [C]: 1 bájt a C címtől
értéke: 1
- byte [C+1]: 1 bájt a C+1 címtől
értéke: 2
- word [C+1]: 2 bájt a C+1 címtől
értéke: $256^1 \cdot 3 + 256^0 \cdot 2 = 770$
- dword [D]: 4 bájt a D címtől
értéke: $256^3 \cdot 0 + 256^2 \cdot 0 + 256^1 \cdot 0 + 256^0 \cdot 42 = 42$

Adatmozgatás: mov

- mov eax,ebx
ebx értékét eax-be tölti
- mov eax,dword [D]
a D címtől kezdődő 4 bájt értékét eax-be tölti
(a dword itt el is hagyható, mert eax-ból kiderül, hogy 4 bájtot kell mozogni)
- mov dword [D],eax
eax értékét a D-től kezdődő 4 bájtba tölti
- mov al,bh
bh értékét al-be tölti
- mov byte [C],al
al értékét a C című bájtba tölti
- mov byte [C],byte [D]
Hibás! Nincs memoriából memóriába mozgatás.

Aritmetikai műveletek: inc, dec, add, sub

- inc eax
az eax értékét megnöveli eggyel
(lehet memória hivatkozás is az operandus)
- dec word [C]
a C címtől kezdődő 2 bájt értékét csökkenti eggyel
(lehet regiszter is az operandus)
- add eax,dword [D]
eax-hez adj a D címtől kezdődő 4 bájt értékét
- sub edx,ecx
levonja edx-ból ecx-et
- Az add és a sub utasításokra ugyanaz a szabály, mint a mov-ra:
legfeljebb az egyik operandus lehet memória hivatkozás.

Aritmetikai műveletek: mul, div

mul ebx

Megszorozza eax értékét ebx értékével.

Az eredmény:

- edx-be kerülnek a nagy helyértékű bajtok
- eax-be az alacsony helyértékűek

div ebx

Elosztja a 256^4 edx+eax értéket ebx értékével.

Az eredmény:

- eax-be kerül a hányados
- edx-be a maradék

- Ha előjeles egész számokkal dolgozunk, akkor az imul és idiv utasításokat kell használni.

Logikai műveletek: and, or, xor, not

and al, bl

Bitenkénti „és” művelet.

Bitenkénti "és"

Ha előtte al= 12 = 0000 1100₂ és bl= 6 = 0000 0110₂, akkor utána al= 0000 0100₂ = 4.

or eax,dword [C]

Bitenkénti „vagy” művelet.

xor word [D],bx

Bitenkénti „kizáró vagy” művelet.

not bl

Bitenkénti „nem” művelet.

Bitenkénti "nem"

Ha előtte bl= 9 = 0000 1001₂, akkor utána bl= 1111 0110₂ = 246.

Igaz, hamis értékek

A magasszintű programozási nyelvek *logikai* (bool) típusának egy lehetséges megvalósítása:

- 1 bájton tároljuk
- hamis érték: 0
- igaz érték: 1
- logikai és: and utasítás
- logikai vagy: or utasítás
- tagadás: not és and 1

Ugró utasítás: jmp

Végtelen ciklus ugró utasítással

eleje: inc ecx
jmp eleje

Utasítások átugrása

```
mov ecx,0  
jmp vege  
add ecx,ebx  
inc ecx  
vege: mov edx,ecx
```

Feltételesek ugró utasítások

- cmp: „compare” - két érték összehasonlítása
- feltételesek ugró utasítások: „ugorj a megadott címkére, ha ...”

Ha eax=0, ugorj az A címkére!

```
cmp eax,0  
je A
```

Ha eax≠0, ugorj az A címkére!

```
cmp eax,0  
jne A
```

Ha eax<ebx, ugorj az A címkére!

```
cmp eax,ebx  
jb A
```

Ha eax>ebx, ugorj az A címkére!

```
cmp eax,ebx  
ja A
```

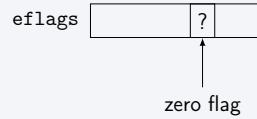
Feltételesek ugró utasítások

- je: „equal” - ugorj, ha egyenlő
- jne: „not equal” - ugorj, ha nem egyenlő
- jb: „below” - ugorj, ha kisebb
≡ jnae: „not above or equal” - nem nagyobb egyenlő
- ja: „above” - ugorj, ha nagyobb
≡ jnbe: „not below or equal” - nem kisebb egyenlő
- jnb: „not below” - nem kisebb
≡ jae: „above or equal” - nagyobb egyenlő
- jna: „not above” - nem nagyobb
≡ jbe: „below or equal” - kisebb egyenlő
- Ha előjeles egészszámokkal számolunk:
jl („less”), jg („greater”), jnl, jng, jle, jge, ...

Feltételes ugró utasítások működése

```
mov eax,10
cmp eax,10
je A
eax [ ] ?
```

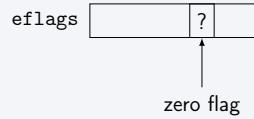
- Az eax regiszter értéke 10 lesz.
- Mivel eax=10, az összehasonlítás beállítja a zero flag-et 1-re.
- A je akkor ugrik, ha a zero flag=1.



Feltételes ugró utasítások működése

```
mov eax,10
cmp eax,10
je A
eax [ ] 10
```

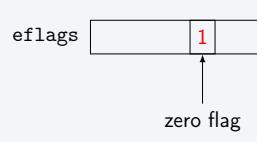
- Az eax regiszter értéke 10 lesz.
- Mivel eax=10, az összehasonlítás beállítja a zero flag-et 1-re.
- A je akkor ugrik, ha a zero flag=1.



Feltételes ugró utasítások működése

```
mov eax,10
cmp eax,10
je A
eax [ ] 10
```

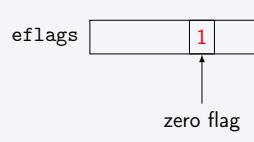
- Az eax regiszter értéke 10 lesz.
- Mivel eax=10, az összehasonlítás beállítja a zero flag-et 1-re.
- A je akkor ugrik, ha a zero flag=1.



Feltételes ugró utasítások működése

```
mov eax,10
cmp eax,10
je A
eax [ ] 10
```

- Az eax regiszter értéke 10 lesz.
- Mivel eax=10, az összehasonlítás beállítja a zero flag-et 1-re.
- A je akkor ugrik, ha a zero flag=1.



Feltételes elágazás

Ha eax<10, akkor eax:=eax+1.

```
cmp eax,10
jnb vege
inc eax
vege:
```

Ha eax<10, akkor eax:=eax+1, különben eax:=eax-1.

```
cmp eax,10
jnb hamis_ag
inc eax
jmp vege
hamis_ag: dec eax
vege:
```

Ciklus

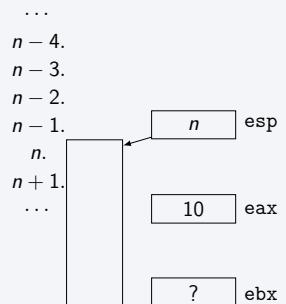
Amíg eax<10, eax:=eax+1.

```
eleje:    cmp eax,10
          jnb vege
          inc eax
          jmp eleje
vege:
```

- Minden futó programhoz hozzá van rendelve egy saját memóriaterület, a program verme.
- Ebben lehet tárolni a lokális változókat.
- Ebben adjuk át a paramétereket alprogramhívás esetén.
- Ebbe kerül bele a visszatérési cím, azaz, hogy hova kell visszatérni az alprogram végén.

```
push eax
pop ebx
```

- eax értéke bekerül a verembe (4 bájt)
- a verem tetején lévő (4 bájtos) érték bemásolódik ebx-be, a veremmutató visszaáll



- ```
push eax
pop ebx
```
- eax értéke bekerül a verembe (4 bájt)
  - a verem tetején lévő (4 bájtos) érték bemásolódik ebx-be, a veremmutató visszaáll
- 
- The diagram shows a vertical stack frame with memory addresses on the left: ..., n-4, n-3, n-2, n-1, n, n+1, ... . At address n-4, there is a box containing the number 10. An arrow points from this box to another box at address n-1 containing the number n-4, labeled 'esp'. At address n+1, there is a box containing the number 10, labeled 'eax'. At the bottom of the stack frame, there is a question mark in a box, labeled 'ebx'.

```
push eax
pop ebx
```

- eax értéke bekerül a verembe (4 bájt)
  - a verem tetején lévő (4 bájtos) érték bemásolódik ebx-be, a veremmutató visszaáll
- 
- The diagram shows a vertical stack frame with memory addresses on the left: ..., n-4, n-3, n-2, n-1, n, n+1, ... . At address n-4, there is a box containing the number 10. An arrow points from this box to another box at address n-1 containing the number n, labeled 'esp'. At address n+1, there is a box containing the number 10, labeled 'eax'. At the bottom of the stack frame, there is a question mark in a box, labeled 'ebx'.

- Csak 2 vagy 4 bájtot lehet a verembe tenni (vagy kivenni).
  - Tehát pl. **push ah** hibás!
- A verem tetején lévő adat pop művelet esetén nem törlődik a memóriából, csak lemosolódik, és a veremmutató megváltozik.
- Ha csak a veremmutató visszaállítása a cél, akkor lehet
 

```
pop ebx
 helyett
 add esp,4
```

 utasítást használni.
  - Ez visszaállítja a veremmutatót, de nem másolja seholra a verem tetején lévő értéket.

**A kiir eljárás hívása**

```
push eax ; Betesszük a verembe a paramétert.
call kiir ; Meghívjuk az eljárást.
 ; Az eljárás a veremből használhatja
 ; az átadott paramétert.
add esp,4 ; Visszaállítjuk a veremmutatót
 ; a push előtti állapotba.
```

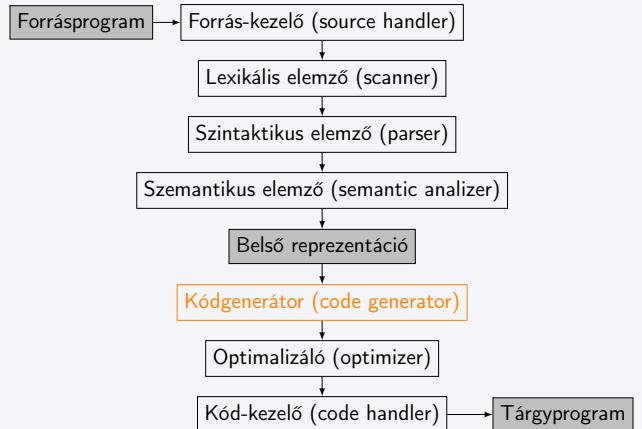
- Szabályozni kell, hogy a *hívó* és a *hívott* kód részlet pontosan hogyan kommunikál egymással:
  - Milyen sorrendben kerülnek a verembe a paraméterek?
  - Kiszedi-e az alprogram a paramétereket a veremből, vagy ez a hívó kód részlet dolga?
  - Hol kapja meg a hívó a függvények visszatérési értékét?
  - Melyik regisztereket változtathatja meg az alprogram?
- A C nyelv hívási konvenciói:
  - A paramétereket **fordított sorrendben** kell a verembe tenni, azaz az első paraméter lesz legfelül.
  - Az alprogram **bennt hagyja a paramétereket** a veremben.
  - A visszatérési érték az **eax** regiszterbe kerül.
  - Az alprogram csak az **eax, edx, ecx** regisztereket módosíthatja. (*Intel ABI* konvenció.)

## Kódgenerálás I. (kifejezések és vezérlési szerkezetek)

Fordítóprogramok előadás (A,C,T szakirány)

2008. őszi félév

## A kódgenerálás helye a fordítási folyamatban



## A kódgenerálás feladata

- a szintaktikusan és szemantikusan elemzett programot tárgykóddá alakítja
- a valóságban általában szorosan összekapcsolódik a szemantikus elemzéssel
- bonyolultabb esetekben:
  - ① a forrásprogram egyszerű utasításokká alakítása
  - ② gépfüggetlen kódoptimalizálás
  - ③ az egyszerű utasítások továbbfordítása assemblyre vagy gépi kódra
  - ④ gépfüggő kódoptimalizálás

## Ebben az előadásban...

- Intel 8086-os architektúrára,
- NASM assembly nyelvre fogjuk fordítani
- az imperatív programozási nyelvek leggyakrabban előforduló konstrukcióit.

## Értékadások fordítása

- alakja:  
*assignment* → *variable assignment operator expression*
- generálandó kód:

### Értékadást megalosító kód

- ① a kifejezést az eax regiszterbe kiértékelő kód
- ② mov [Változó],eax

- megjegyzések:
  - a kifejezések kiértékelésével később foglalkozunk
  - Változó az értékadás bal oldalán szereplő változó címkéje
  - bonyolultabb adatszerkezetek lemásolását külön eljárás végzi (*másoló konstruktör*)

## Összetett típusokra vontakozó értékadások

- bizonyos típusokra (pl. rekordok, tömbök) könnyen lehet alapértelmezett másoló eljárást készíteni:
  - mezőkénti / elemenkénti másolás
- láncolt adatszerkezetek másolási stratégiái:
  - rekurzívan minden lemásolunk
  - csak a legfelső szinten történik másolás, a mutatók által hivatkozott memóriaterületek közösek lesznek
  - megadjuk a lehetőséget a programozónak, hogy maga írja meg a másoló konstruktort

## Egy ágú elágazás fordítása

- alakja: *statement → if condition then program end*

### Egy ágú elágazás kódja

- a feltételt az al regiszterbe kiértékelő kód
- cmp al,1
- jne near Vége
- a then-ág programjának kódja
- Vége:

- a feltételek kiértékelésével később foglalkozunk
- itt a *logikai igaz* értéket az 1 reprezentálja
- jne Vége utasítás: maximum 128 bájt távolságra tud ugrani; az then-ág ennél hosszabb is lehet ⇒ jne *near* Vége
- a programban több elágazás is lehet
- ⇒ minden esetben **egyedi címkéket** kell generálni!

## Ha a feltételes ugrás csak rövid lehet

- ha a feltételes ugrásból kihagyjuk a *near-t*, akkor csak rövidet tud ugrani
- a jmp (feltétel nélküli) ugrás viszont alapértelmezetten hosszú ugrás
- így egy alternatív megoldás:

### Egy ágú elágazás kódja - másik megoldás

- a feltételt az al regiszterbe kiértékelő kód
- cmp al,1
- je Then
- jmp Vége
- Then: a then-ág programjának kódja
- Vége:

- ez a trükk a többi programkonstrukció esetén is alkalmazható

## Címkék generálása

- Egyedi címkékre van szükség:

- elágazások
- ciklusok
- változó- és alprogramdefiníciók fordításakor.

- Egy lehetséges megoldás:

- Lab1, Lab2, Lab3, ...
- Egy számlálót tartunk fent, amit minden alkalommal inkrementálunk.
- Új címke: a számláló értékét kell a végére koncatenálni:

```
stringstream ss;
ss << "Lab" << szamlalo++;
string cimkenev = ss.str();
```

## Több ágú elágazás alakja

```
statement →
if condition1 then program1
elseif condition2 then program2
...
elseif conditionn then programn
else programn+1 end
```

## Több ágú elágazás kódja

- az 1. feltétel kiértékelése az al regiszterbe
- cmp al,1
- jne near Feltétel\_2
- az 1. ág programjának kódja
- jmp Vége
- ...
- Feltétel\_n: az n-edik feltétel kiértékelése az al regiszterbe
- cmp al,1
- jne near Else
- az n-edik ág programjának kódja
- jmp Vége
- Else: az else ág programjának kódja
- Vége:

## A switch-case utasítás fordítása

- alakja:  
*statement → switch variable  
case value<sub>1</sub> : program<sub>1</sub>  
...  
case value<sub>n</sub> : program<sub>n</sub>*
- a generálandó kód hasonló egy több ágú elágazáshoz
- mivel itt a feltételekre megszorítások vannak, lehet hatékonyabb a kiértékelés
  - csak *variable == value* alakúak a feltételek, ahol a *value* konstans érték
- a switch-case másként működik az egyes nyelvekben:
  - Ada stílus: csak egy ág hajtódk végre
  - C stílus: az első teljesülő ágtól kezdve az összes végrehajtódk (hacsak nem használunk break utasítást az ágak végén)

## A switch-case utasítás fordítása (Ada stílus)

```
❶ cmp [Változó],Érték_1
❷ jne near Feltétel_2
❸ első ág programjának kódja
❹ jmp Vége
❺ Feltétel_2: cmp [Változó],Érték_2
❻ ...
❼ Feltétel_n: cmp [Változó],Érték_n
➋ jne near Vége
⩿ n-edik ág programjának kódja
⩾ Vége:
```

## A switch-case utasítás fordítása (C stílus)

```
❶ cmp [Változó],Érték_1
❷ je near Program_1
❸ cmp [Változó],Érték_2
❹ je near Program_2
❺ ...
❻ cmp [Változó],Érték_n
❼ je near Program_n
⩿ jmp Vége
⩾ Program_1: az 1. ág programjának kódja
⩾ ...
⩾ Program_n: az n-edik ág programjának kódja
⩾ Vége:
```

## Elöl tesztelő ciklus fordítása

- alakja: *statement* → *while condition program end*
- generálandó kód:

### Elöl tesztelő ciklus kódja

```
❶ Eleje: a ciklusfeltétel kiértékelése az al regiszterbe
❷ cmp al,1
❸ jne near Vége
❹ a ciklusmag programjának kódja
❺ jmp Eleje
❻ Vége:
```

## Hátul tesztelő ciklus fordítása

- alakja: *statement* → *loop program while condition*
- generálandó kód:

### Hátul tesztelő ciklus kódja

```
❶ Eleje: a ciklusmag programjának kódja
❷ a ciklusfeltétel kiértékelése az al regiszterbe
❸ cmp al,1
❹ je near Eleje
```

## For ciklus fordítása

- alakja:  
*statement* → *for variable from value<sub>1</sub> to value<sub>2</sub> program end*
- hasonlítható a while ciklusok fordításához
  - hiszen minden for ciklus átalakítható while ciklussá

### For ciklus kódja

```
❶ a „from” érték kiszámítása a [Változó] memóriahezre
❷ Eleje: a „to” érték kiszámítása az eax regiszterbe
❸ cmp [Változó],eax
❹ ja near Vége
❺ a ciklusmag kódja
❻ inc [Változó]
❼ jmp Eleje
⩿ Vége:
```

## Ciklusváltozó tárolása regiszterben

- hatékonyabb lehet a kód, ha a ciklusváltozót regiszterben tároljuk
- van processzor-szintű támogatás is erre: loop utasítás

### Példa a loop utasításra

```
mov ecx,10 ; 10-szer fogjuk végrehajtani
Eleje:
; ide kerül a ciklusmag
loop Eleje ; csökkenti ecx-et,
; ha még pozitív: visszaugrik,
; ha nulla lett: továbblép
```

- Vigyázat:** lehet, hogy a ciklusmag elállítja az ecx regiszert!
  - vagy gondoskoni kell róla, hogy ne állítsa el
  - vagy körül kell venni a ciklusmagot:

**Az ecx regiszter eleme**

```
push ecx
; ciklusmag
pop ecx
```

- sokféle változó van: statikus, lokális, dinamikusan allokat...
- most a **statikus** változókkal foglalkozunk
  - globális változók
  - kifejezetten statikusnak deklarált változók  
(pl. C++ static kulcsszó)
- a statikus változó a .data vagy .bss szakaszban kapnak helyet
- a többi változóval a következő előadáson foglalkozunk

- kezdőérték nélkül: int x;

**Kezdőérték nélküli változódefinició fordítása**

```
section .bss
; a korábban definiált változók...
Lab12: resd 1 ; 1 x 4 bájtnyi terület
```

- kezdőértékkel: int x=5;

**Kezdőértékkel adott változódefinició fordítása**

```
section .data
; a korábban definiált változók...
Lab12: dd 5 ; 4 bájton tárolva az 5-ös érték
```

Minden változóhoz **új címkét** kell generálni és **fel kell jegyezni** a szimbólumtáblába a változó attribútumai közé!

**1. próbálkozás (*kifejezés<sub>1</sub>+kifejezés<sub>2</sub>*)**

```
; a 2. kifejezés kiértékelése eax-be
mov ebx,eax
; az 1. kifejezés kiértékelése eax-be
add eax,ebx
```

**Ez hibás!** Ha a részkifejezésekben is használjuk ebx-et, elállítjuk az értékét...

**Megoldás:**

**2. próbálkozás (*kifejezés<sub>1</sub>+kifejezés<sub>2</sub>*)**

```
; a 2. kifejezés kiértékelése eax-be
push eax
; az 1. kifejezés kiértékelése eax-be
pop ebx
add eax,ebx
```

- A példákban az eax regiszterbe értékeljük ki a kifejezéseket.
- 1. eset: egyetlen konstans értékből álló kifejezés (pl. 25)

**Konstans kiértékelése**

```
mov eax,25
```

- 2. eset: egyetlen változóból álló kifejezés (pl. x)

**Változó kiértékelése**

```
mov eax,[X] ; ahol X a változó címkeje
```

- 3. eset: összetett kifejezés
  - (pl. fibonacci(25) + factorial(x))
  - általános megoldás: függvényhívásokat teszünk a kódba
  - a beépített függvényekhez (+,-,\*,/,&&,||,!,...) lehet hatékonyabban is

- cél: az al regiszterbe kiértékelni a logikai kifejezés eredményét
- 1. eset: <, >, =, ... operátorok

***kifejezés<sub>1</sub> < kifejezés<sub>2</sub>* kiértékelése**

```
; a 2. kifejezés kiértékelése az eax regiszterbe
push eax
; az 1. kifejezés kiértékelése az eax regiszterbe
pop ebx
cmp eax,ebx
jb Kisebb
mov al,0 ; hamis
jmp Vége
Kisebb:
mov al,1 ; igaz
Vége:
```

## Logikai kifejezések fordítása

- 2. eset: *és*, *vagy*, *nem*, *kizárvagy*, ... műveletek (hasonlóan a +, -, ... kiértékeléséhez)

### *kifejezés<sub>1</sub>* és *kifejezés<sub>2</sub>* kiértékelése

```
; a 2. kifejezés kiértékelése az al regiszterbe
push ax ; nem lehet 1 bájtot a verembe tenni!
; az 1. kifejezés kiértékelése az al regiszterbe
pop bx ; bx-nek a bl részében van,
 ; ami nekünk fontos
and al,bl
```

- de ha az 1. hamis, akkor már nem is kell kiértékelni a 2-dikat...

25

Fordítóprogramok előadás (A,C,T szakirány)

Kódgenerálás I. (kifejezések és vezérlési szerkezetek)

## Rövidzás logikai operátorok

- Ha *kifejezés<sub>1</sub>* && *kifejezés<sub>2</sub>* kifejezésben az első hamis, akkor a másodikat garantáltan nem értékeli ki.
  - fontos lehet: (a != 0) && (c == b / a)

### *kifejezés<sub>1</sub>* és *kifejezés<sub>2</sub>* kiértékelése

```
; az 1. kifejezés kiértékelése az al regiszterbe
cmp al,0
je Vége
push ax
; a 2. kifejezés kiértékelése az al regiszterbe
mov bl,al
pop ax
and al,bl
Vége:
```

26

Fordítóprogramok előadás (A,C,T szakirány)

Kódgenerálás I. (kifejezések és vezérlési szerkezetek)

## Kódgenerálás és *S* – ATG

- Emlékeztető: *S*-attribútum fordítási grammatika
  - csak szintetizált („alulról felfelé” terjedő) attribútumok
  - jól illeszkedik az alulról felfelé elemzésekhez (*bisonc++*)

27

Fordítóprogramok előadás (A,C,T szakirány)

Kódgenerálás I. (kifejezések és vezérlési szerkezetek)

## Kódgenerálás és *S* – ATG

- Emlékeztető: *S*-attribútum fordítási grammatica
  - csak szintetizált („alulról felfelé” terjedő) attribútumok
  - jól illeszkedik az alulról felfelé elemzésekhez (*bisonc++*)
- Az eddig látott konstrukciók kényelmesen beilleszthetők a szemantikus elemzésbe:
  - a szimbólumoknak egy attribútuma lesz a hozzájuk generált kód
  - az összetettebb konstrukciók kódjához (eddig) csak kombinálni kellett a részeihez generált kódokat
    - pl. az elágazás kódja az egyes ágak kódja kiegészítve néhány összehasonlítással és ugrással

27

Fordítóprogramok előadás (A,C,T szakirány)

Kódgenerálás I. (kifejezések és vezérlési szerkezetek)

## Kódgenerálás és *S* – ATG

- Emlékeztető: *S*-attribútum fordítási grammatica
  - csak szintetizált („alulról felfelé” terjedő) attribútumok
  - jól illeszkedik az alulról felfelé elemzésekhez (*bisonc++*)
- Az eddig látott konstrukciók kényelmesen beilleszthetők a szemantikus elemzésbe:
  - a szimbólumoknak egy attribútuma lesz a hozzájuk generált kód
  - az összetettebb konstrukciók kódjához (eddig) csak kombinálni kellett a részeihez generált kódokat
    - pl. az elágazás kódja az egyes ágak kódja kiegészítve néhány összehasonlítással és ugrással
- Bonyolultabb a kódgenerálás a *nem strukturált* konstrukciók esetén.
  - goto, break, kivételkezelés

27

Fordítóprogramok előadás (A,C,T szakirány)

Kódgenerálás I. (kifejezések és vezérlési szerkezetek)

## A goto utasítás fordítása

### Forrásprogram

```
Lab: x++;
 ...
 goto Lab;
```

Vigyázni kell a következőkre:

- A felhasználó címkeje nehogy egybe essen egy generált címkével.
  - vagy generálni kell a felhasználó címkeje helyett is egy újat, és feljegyezni a szimbólumtáblába
  - vagy olyan címkéket generálni, amit a felhasználó nem írhat a forrásprogramba
- Ha pl. push ecx ... pop ecx utasításokkal körülvett ciklusmagból történik a kiugrás, helyre kell állítani a vermet...
- Még bonyolultabb, ha alprogramkból is ki lehet ugrani...

### Generálandó kód

```
Lab: inc [X]
 ...
 jmp Lab
```

28

Fordítóprogramok előadás (A,C,T szakirány)

Kódgenerálás I. (kifejezések és vezérlési szerkezetek)

## A break utasítás fordítása

- A break utasítás a legbelso ciklusból / elágazásból ugrik ki.
- Példa:

### Forrásszöveg

```
while(b)
{
 x++;
 break;
}
```

### Generálandó kód

```
Eleje: mov al,[B]
 cmp al,1
 jne Vége
 inc [X]
 jmp Vége
 jmp Eleje
```

Vége:

## A break utasítás fordítása

- Alulról felfelé elemzéskor...
  - a ciklusmag kódját kell először generálni (a break kódját is)
  - a ciklus kódját később

## A break utasítás fordítása

- Alulról felfelé elemzéskor...
  - a ciklusmag kódját kell először generálni (a break kódját is)
  - a ciklus kódját később
- Probléma:
  - a Vége címkét a ciklus feldolgozásakor generáljuk,
  - pedig szükség van rá a break kódjában is!
  - Azaz ez a címke egy örökölt attribútum...

## A break utasítás fordítása

- Alulról felfelé elemzéskor...
  - a ciklusmag kódját kell először generálni (a break kódját is)
  - a ciklus kódját később
- Probléma:
  - a Vége címkét a ciklus feldolgozásakor generáljuk,
  - pedig szükség van rá a break kódjában is!
  - Azaz ez a címke egy örökölt attribútum...
- Megoldás lehet:
  - kihagyni a generált kódban a címke helyét
  - megjegyezni, hogy volt-e a ciklusmagban break (ez szintetizált attribútum!)
  - ha volt, akkor a ciklus generálásakor kitöltsük a hiányzó címkét

## A break és az L – ATG-k

- Emlékeztető: L-attribútum fordítási grammaika
  - az attribútumok először felülről lefelé, majd a szabályban balról jobbra, végül felfelé terjednek
  - jól illeszkedik az LL elemzésekhez (pl. rekurzív leszállás)
- A break fordításához szükséges örökölt attribútum nem okoz gondot L – ATG esetén
  - ciklus szimbólumnál lefelé haladva generáljuk a Vége címkét
  - a ciklusmag kódjának generálásakor a címke már rendelkezésre áll
  - a ciklusmag feldolgozása után felfelé haladva a szintaxisában elérhető a ciklus kódja

## Példa: rekurzív leszállás és a break

```
Kod ciklus_utasitas()
{
 Cimke eleje = cimkegenerator.ujcimke();
 Cimke vege = cimkegenerator.ujcimke();
 Kod ciklusmag_kodja;
 ...
 if(aktualis_token == break_token)
 ciklusmag_kodja = break_utasitas(vege);
 ...
 return ...;
}

Kod break_utasitas(Cimke c)
{
 elfogad(break_token);
 return new Kod("jmp " + c);
}
```

(Kicsit csaltunk: a ciklus eljárásában közvetlenül nem jelenne meg a break, hanem a ciklusmag egy utasítássorozat, ami utasításokból áll és egy utasítás lehet break is...)

## Kódgenerálás II. (alprogramok, memóriakezelés)

Fordítóprogramok előadás (A,C,T szakirány)

2008. őszi félév

## Alprogramok fordítása

- függvény, eljárás, metódus
- paraméterátadási formák
  - érték szerint, hivatkozás szerint...
- tagfüggvények

## Függvény, eljárás

- **függvény:**
  - csak „bemenő” paraméterek
  - visszatérési érték
  - nincs mellékhatása
- **eljárás:**
  - „ki- és bemenő” paraméterek
  - jellemzően nincs visszatérési érték
- *C, C++, Java, ...*: nincs ilyen megkülönböztetés

### Példa: függvény

```
int duplaja(int n)
{
 return 2*n;
}
```

### Példa: eljárás

```
void duplaz(int & n)
{
 n *= 2;
}
```

## Alprogramok írása assemblyben

- paraméter nélküli alprogramok
- paraméterátadás
- lokális változók

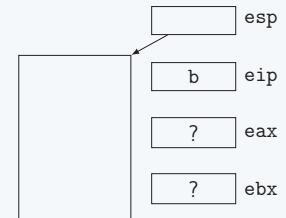
## A call és ret utasítások

- **call Címke**
  - az eip regiszter tartalmát a verembe teszi
    - ez a call utáni utasítás címe
    - visszatérési címnek nevezik
  - átadja a vezérlést a Címke címkéhez
    - mint egy ugró utasítás
- **ret**
  - kiveszi a verem legfelső négy bájtját és az eip regiszterbe teszi
    - mint egy pop utasítás
    - a program a veremben talált címnél folytatódik

## Példa: paraméter nélküli alprogram

```
a: ...
b: call nulláz
c: ...

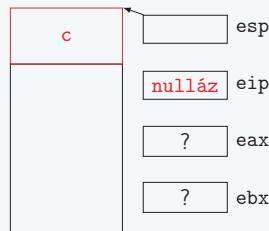
nulláz: mov eax,0
d: mov ebx,0
e: ret
```



## Példa: paraméter nélküli alprogram

```
a: ...
b: call nulláz
c: ...

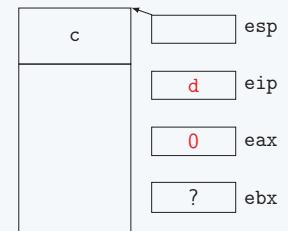
nulláz: mov eax,0
d: mov ebx,0
e: ret
```



## Példa: paraméter nélküli alprogram

```
a: ...
b: call nulláz
c: ...

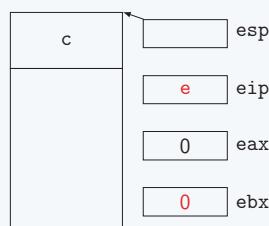
nulláz: mov eax,0
d: mov ebx,0
e: ret
```



## Példa: paraméter nélküli alprogram

```
a: ...
b: call nulláz
c: ...

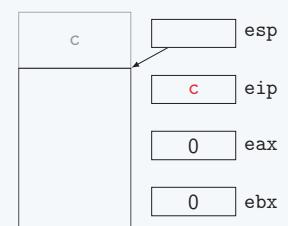
nulláz: mov eax,0
d: mov ebx,0
e: ret
```



## Példa: paraméter nélküli alprogram

```
a: ...
b: call nulláz
c: ...

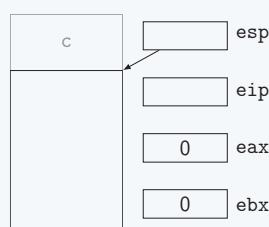
nulláz: mov eax,0
d: mov ebx,0
e: ret
```



## Példa: paraméter nélküli alprogram

```
a: ...
b: call nulláz
c: ...

nulláz: mov eax,0
d: mov ebx,0
e: ret
```



## Paraméterek átadása

- a paramétereket a verembe kell tenni a call utasítás előtt
- C stílusú paraméterátadás: fordított sorrendben tessük a verembe
  - az utolsó kerül legalulra
  - az első a verem tetejére
- az ejárásból való visszatérés után a hívó állítja vissza a vermet
- visszatérési érték: az eax regiszterbe kerül

## Példa: paraméterátadás (1. változat)

```

a: ...
b: push dword 5
c: call duplája
d: add esp,4
e: ...

duplája: mov eax,[esp+4]
f: add eax,[esp+4]
g: ret

```

## Példa: paraméterátadás (1. változat)

```

a: ...
b: push dword 5
c: call duplája
d: add esp,4
e: ...

duplája: mov eax,[esp+4]
f: add eax,[esp+4]
g: ret

```

## Példa: paraméterátadás (1. változat)

```

a: ...
b: push dword 5
c: call duplája
d: add esp,4
e: ...

duplája: mov eax,[esp+4]
f: add eax,[esp+4]
g: ret

```

## Példa: paraméterátadás (1. változat)

```

a: ...
b: push dword 5
c: call duplája
d: add esp,4
e: ...

duplája: mov eax,[esp+4]
f: add eax,[esp+4]
g: ret

```

## Példa: paraméterátadás (1. változat)

```

a: ...
b: push dword 5
c: call duplája
d: add esp,4
e: ...

duplája: mov eax,[esp+4]
f: add eax,[esp+4]
g: ret

```

## Példa: paraméterátadás (1. változat)

```

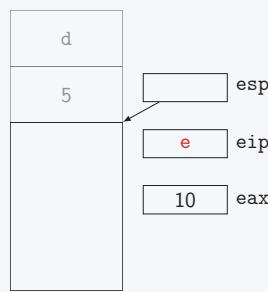
a: ...
b: push dword 5
c: call duplája
d: add esp,4
e: ...

duplája: mov eax,[esp+4]
f: add eax,[esp+4]
g: ret

```

## Példa: paraméterátadás (1. változat)

```
a: ...
b: push dword 5
c: call duplája
d: add esp,4
e: ...
duplája: mov eax,[esp+4]
f: add eax,[esp+4]
g: ret
```



## Paraméterátadás – 1. változat

- hivatkozás a paraméterekre (ha mindegyik 4 bájtos):
   
1.: [esp+4] 2.: [esp+8] ...

## Paraméterátadás – 1. változat

- hivatkozás a paraméterekre (ha mindegyik 4 bájtos):
   
1.: [esp+4] 2.: [esp+8] ...
- probléma:** Időnként használni akarjuk a vermet az alprogram belsejében.  
Például:
  - egyes regiszterek elmentése
  - lokális változóknak helyfoglalás
- Ilyenkor változik a paraméterekre való hivatkozás módja!

### Példa: veremhasználat alprogram belsejében

```
alprogram: ; itt [esp+4] az első paraméter
 push ecx
 ; itt már [esp+8]
 pop ecx
 ; itt mégint [esp+4]
 ret
```

## Paraméterátadás – 2. változat

- Megoldási ötlet:** Használjuk az ebp regisztert az esp helyett:
  - az alprogram elején ebp megkapja esp értékét
  - közben esp akár hogyan változhat
  - a paramétereket minden ugyanúgy érjük el: [ebp+4], [ebp+8], ...

### Példa: ebp használata a paraméterek eléréséhez

```
alprogram: mov ebp,esp
 ; az első paraméter: [ebp+4]
 push ecx
 ; ugyanúgy [ebp+4]
 pop ecx
 ; továbbra is [ebp+4]
 ret
```

## Paraméterátadás – 2. változat

- probléma:** Alprogramhívás egy alprogram belsejében: elállítja ebp értékét!

### Alprogramhívás alprogramban

```
egyik: mov ebp,esp
 ; első paraméter: [ebp+4]
 call masik ; ez elállítja ebp-t
 ; itt ebp-t már nem tudjuk használni
 ret

masik: mov ebp,esp ; itt állítjuk el ebp-t
 ; ...
 ret
```

## Paraméterátadás – 3. (végső) változat

- Megoldás:** Mentsük el ebp értékét az alprogram elején a verembe és állítsuk vissza a végén!

### Alprogramhívás alprogramban

```
egyik: push ebp
 mov ebp,esp
 ; első paraméter: [ebp+4]
 call masik ; ez megőrzi ebp értékét
 ; itt is [ebp+8] az első paraméter
 pop ebp
 ret

masik: push ebp ; itt mentjük el ebp-t
 mov ebp,esp
 ; ...
 pop ebp ; és itt állítjuk vissza
 ret
```

## Paraméterátadás – 3. (végső) változat

- Hivatkozás a paraméterekre: [ebp+8], [ebp+12], ...
- [ebp]: az ebp elmentett értéke
- [ebp+4]: a visszatérési cím
- Akár rekurzív is lehet az alprogram...

## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

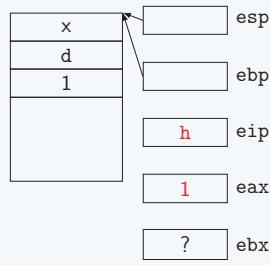
```

## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

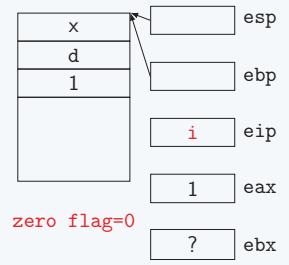


## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

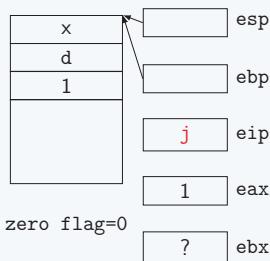


## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

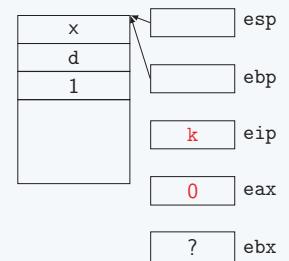


## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

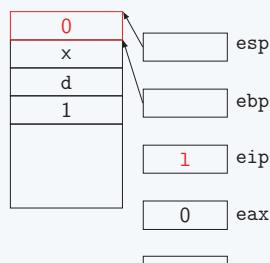


## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

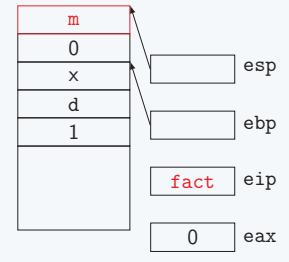


## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

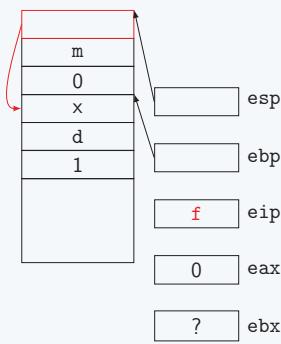


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

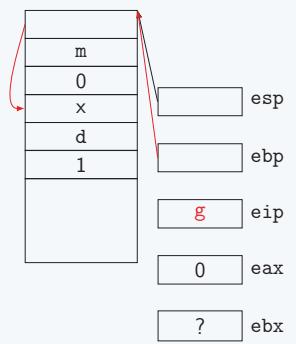


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

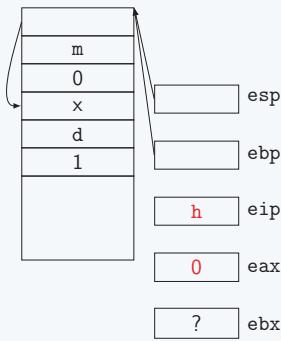


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

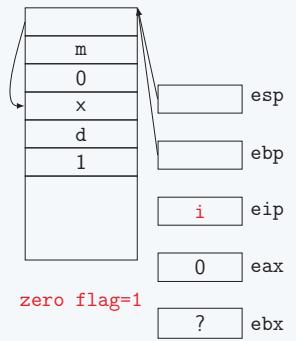


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

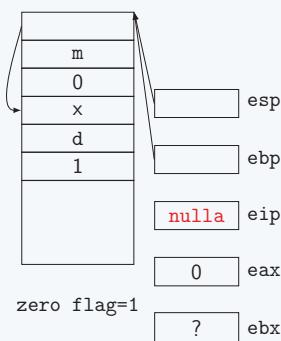


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

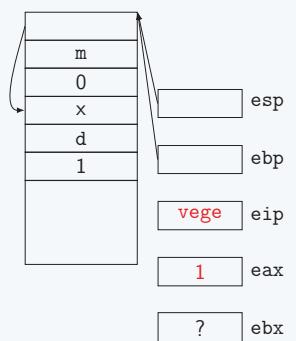


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
ret

```

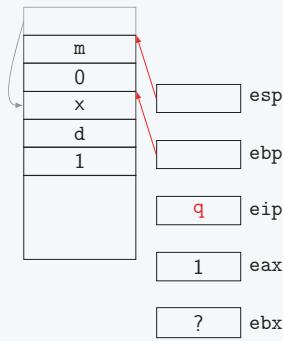


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
q: ret

```

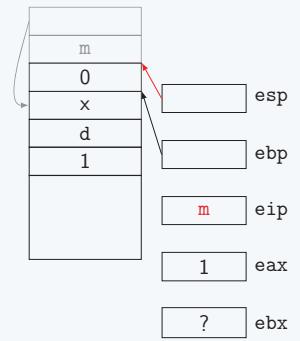


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
q: ret

```

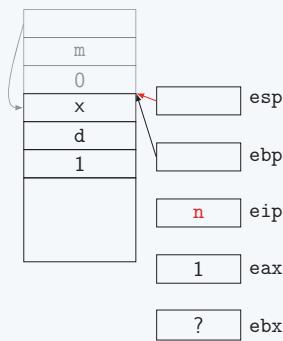


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
q: ret

```

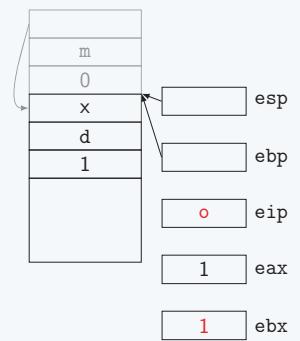


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
q: ret

```

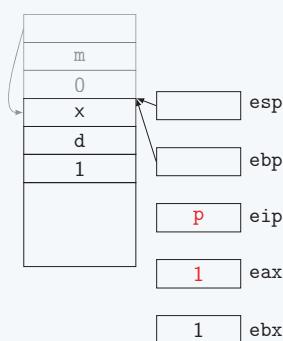


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
q: ret

```

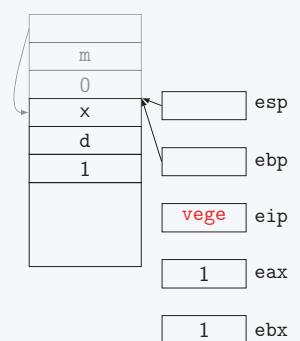


### Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
q: ret

```

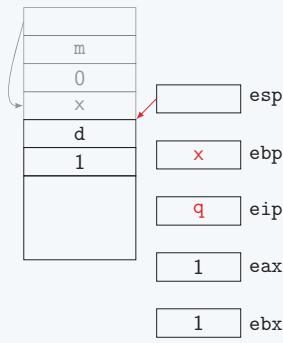


## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
q: ret

```

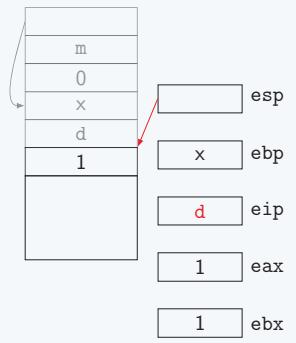


## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
q: ret

```

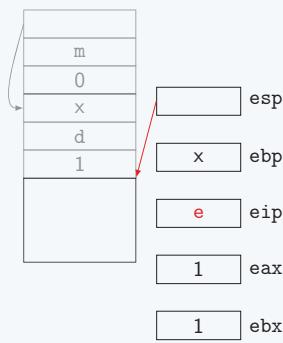


## Paraméterátadás (3. változat)

```

a: ...
b: push dword 1
c: call fact
d: add esp,4
e: ...
fact: push ebp
f: mov ebp,esp
g: mov eax,[ebp+8]
h: cmp eax,0
i: je nulla
j: dec eax
k: push eax
l: call fact
m: add esp,4
n: mov ebx,[ebp+8]
o: mul ebx
p: jmp vege
nulla: mov eax,1
vege: pop ebp
q: ret

```



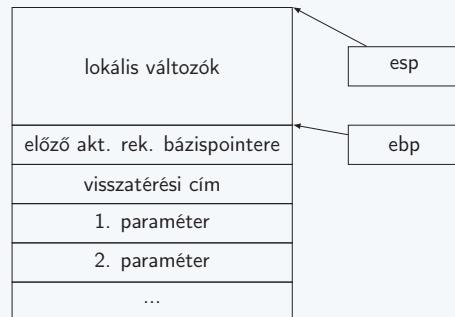
## Paraméterátadás – megjegyzések

- minden éppen futó alprogram-példányhoz tartozik egy résza veremből
  - aktivációs rekord / verem-keret (stack frame)
- az ebp regiszter segítségével érhetők el az aktuális aktivációs rekord részei bázis pointer
- az ebp-ból indulva egy láncolt listára vannak felfűzve az aktivációs rekordok
  - ez teszi lehetővé az eggyel korábbi aktivációs rekordhoz való visszatérést az alprogram végén

## Lokális változók

- a lokális változók a verem tetejére kerülnek
- hivatkozás a lokális változóra (ha mindenky 4 bájtos):
  - [ebp-4], [ebp-8], ...
- az esp a lokális változók területe „fölé” mutat
  - lehet veremműveleteket és alprogram-hívásokat végrehajtani a lokális változók felülírása nélkül

## Az aktivációs rekord felépítése



## Alprogramok sémája

```

alprogram: push ebp
 mov ebp,esp
 sub esp,'a lokális változók mérete'
 ; az alprogram törzse
vége: add esp,'a lokális változók mérete'
 pop ebp
 ret

```

- egyedi címkeket kell generálni
- a lokális változók összmérete a törzs szintetizált attribútuma lehet

- összesíteni kell az alprogramban használt lokális változókat
  - ha minden az elején kell definiálni, akkor könnyű
  - fel kell venni őket a szimbólumtáblába
  - mindenkihez ki kell számolni, hogy hol fog elhelyezkedni: [ebp-?]
- az összméretükre szükség lesz az alprogram
 

```
sub esp,'a lokális változók mérete'
```

 soránál

- kiegészítjük a kifejezéskiértékelés fordítását egy új esettel: „ha a kifejezés egyetlen lokális változó...”
- a szimbólumtáblából kiolvassuk a pozícióját: *p*
- a generálandó kód:

## Hivatkozás lokális változóra

```
mov eax,[ebp-p]
```

## A 'return kifejezés' utasítás kódja

```
; a kifejezés kiértékelése eax-be
jmp vége
```

- itt is tudni kell az alprogram végét jelző címkét (hasonló problémák, mint a break esetén)

## Alprogramok sémája

```

; utolsó paraméter kiértékelése eax-be
push eax
;
; ...
; 1. paraméter kiértékelése eax-be
push eax
call alprogram
add esp,'a paraméterek összhossza'

```

- érték szerint:
  - a paraméterértékeket másoljuk a verembe
  - ha az alprogram módosítja, az nem hat az átadott változóra
- hivatkozás szerint
  - az átadandó változóra mutató pointert kell a verembe tenni
  - az alprogramban a lokális változó kiértékelése is módosul:
 

```
mov eax,[ebp+p]
 mov eax,[eax]
```

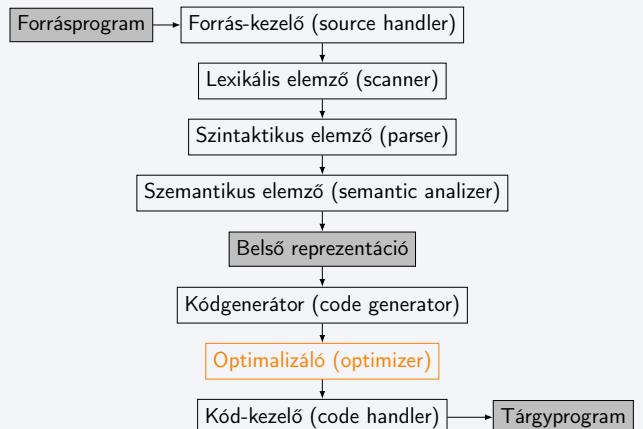
- statikus memóriakezelés:
  - a .data vagy .bss szakaszban
  - globális vagy statikusnak deklarált változók
  - előre ismerni kell a változók méretét, darabszámát
- dinamikus memóriakezelés
  - blokk-szerkezethez kötődő, lokális változók: *verem*
  - tetszőleges élettartamú változók: *heap memória*

- két alapvető művelet: *allokálás* és *felszabadítás*
- az operációs rendszer vagy egy programkönyvtár végzi
- a szabad és foglalt területeket nyilván kell tartani
  - allokáláskor valamilyen stratégia szerint egy szabad területet kell lefoglalni
  - a felszabadított memóriát hozzá kell tenni a szabad területhez
- assemblyból lehet hívni a C malloc és free függvényeit

## Kódoptimalizálás

Fordítóprogramok előadás (A,C,T szakirány)

## A fordítóprogramok szerkezete



1 Fordítóprogramok előadás (A,C,T szakirány) Kódoptimalizálás

2 Fordítóprogramok előadás (A,C,T szakirány) Kódoptimalizálás

## A szintézis menete „valójában”

- ① Optimalizálási lépések végrehajtása az eredeti programon (vagy annak egyszerűsített változatán).
- ② Kódgenerálás.
- ③ Gépfüggő optimalizálás végrehajtása a generált kódon.

## A kódoptimalizálás célja

- Hatékonyabb program létrehozása:
  - nagyobb sebesség
  - kisebb méret
- **Ezek a célok időnként ellentmondanak egymásnak!**

3 Fordítóprogramok előadás (A,C,T szakirány) Kódoptimalizálás

4 Fordítóprogramok előadás (A,C,T szakirány) Kódoptimalizálás

## Követelmény a kódoptimalizálással szemben

- „Az optimalizált programnak ugyanúgy kell működnie, mint az eredetinek.” - Ez sok minden jelenthet!
  - ugyanarra a bemenetre ugyanazt a kimenetet adja?
  - eseményvezérelt környezetben ugyanúgy viselkedik?
  - párhuzamos környezetben ugyanúgy viselkedik?
  - stb.
- Az adott nyelv szemantikája (jelentése) dönti el, hogy egy optimalizálási lépés megengedhető-e.

## Kódoptimalizálási lépések osztályozása

- Mit optimalizálunk?
  - az eredeti programot (vagy annak egyszerűsített változatát):  
itt még vannak ciklusok, elágazások, kifejezéskiértékelés stb.
  - a tárgyprogramot: jellemzően assembly kódot
- Mi az átalakítás hatóköre?
  - lokális: kis programrészletek átalakítása
  - globális: a teljes program szerkezetét kell vizsgálni
- Mit használ ki az átalakítás?
  - általános optimalizálási stratégiák: algoritmusok javítása
  - gépfüggő optimalizálás: az adott architektúra sajátosságait használja ki

5 Fordítóprogramok előadás (A,C,T szakirány) Kódoptimalizálás

6 Fordítóprogramok előadás (A,C,T szakirány) Kódoptimalizálás

## Lokális optimalizálás: alapblokk fogalma

### Definíció: alapblokk

Egy programban egymást követő utasítások sorozatát alapblokknak nevezzük, ha

- az első utasítás kivételével egyik utasítására sem lehet távolról átadni a vezérlést  
(assembly programokban: ahová a `jmp`, `call`, `ret` utasítások „ugranak”; magas szintű nyelvekben: eljárások, ciklusok eleje, elágazások ágainak első utasítása, `goto` utasítások célpontjai)
- az utolsó utasítás kivételével nincs benne vezérlés-átadó utasítás  
(assembly programban: `jmp`, `call`, `ret` magas szintű nyelvekben: elágazás vége, ciklus vége, eljárás vége, `goto`)
- az utasítás-sorozat nem bővíthető a fenti két szabály megsértése nélkül

## Alapblokk szerepe az optimalizálásban

### A definíció következményei:

- egy alapblokknak pontosan egy belépési pontja van (az első utasítás)
- az utasításai szekvenciálisan hajtódnak végre
- ha rá kerül a vezérlés, akkor az összes utasítása pontosan egyszer hajtódnak végre

### • Lokális optimalizálás: egy alapblokkon belüli átalakítások

## Alapblokok meghatározása

- Jelöljük meg:
  - a program első utasítását
  - azokat az utasításokat, amelyekre távolról át lehet adni a vezérlést
  - a vezérlés-átadó utasításokat követő utasításokat
- minden megjelölt utasításhoz tartozik egy alapblokk, ami a következő megjelölt utasításig (vagy az utolsó utasításig) tart.

## Alapblokok: példa

```
main: mov eax,[Cimke1]
 cmp eax,[Cimke2]
 jz igaz
 dec dword [Cimke1]
 inc dword [Cimke2]
 jmp vege
igaz: inc dword [Cimke1]
 dec dword [Cimke2]
vege: ret
```

## Tömörítés

- Cél: minél kevesebb konstans és konstans értékű változó legyen!
- konstansok összevonása: a fordítási időben kiértékelhető kifejezések kiszámítása

### Eredeti kód

```
a := 1 + b + 3 + 4;
```

### Optimalizált kód

```
a := 8 + b;
```

- konstans továbbterjesztése: a fordítási időben kiszámítható értékű változók helyettesítése az értékükkel

### Eredeti kód

```
a := 6;
b := a / 2;
c := b + 5;
```

### Optimalizált kód

```
a := 6;
b := 3;
c := 8;
```

## Azonos kifejezések többszöri kiszámításának elkerülése

### Eredeti kód

```
x := 20 - (a * b);
y := (a * b) ^ 2;
```

### Optimalizált kód

```
t := a * b;
x := 20 - t;
y := t ^ 2;
```

- Ez csak látszólag növeli a program méretét!

- az `a*b` kifejezés kiértékelése is több assembly utasítás
- a `t` változó lehet egy regiszter is

- Megvalósítás a gyakorlatban:

- az utasításokból egy címkézett, irányított körmentes gráfot építünk,
- ebből generálható az optimalizált kód

## Változó továbbterjesztése

## Eredeti kód

```
x := a + b;
y := x;
z := y;
```

## Optimalizált kód

```
x := a + b;
y := x;
z := x;
```

- Ha az y változóra a továbbiakban már nincs szükség, akkor  $y := x$  törölhető!
- (De ez a törlés már globális optimalizálás...)
- Ez is megoldható az előzőleg említett gráfos módszerrel.

## Ablakoptimalizálás

- Ez egy módszer a lokális optimalizálás egyes fajtához.

## • Ablak:

- egyszerre csak egy néhány utasításnyi részt vizsgálunk a kódóból
- a vizsgált részt előre megadott mintákkal hasonlítjuk össze
- ha illeszkedik, akkor a mintához megadott szabály szerint átalakítjuk
- ezt az „ablakot” végigcsúsztatjuk a programon

## • Az átalakítások megadása:

- $\{minta \rightarrow helyettesítés\}$  szabályhalmazzal
- a mintában lehet paramétereket is használni

## Ablakoptimalizálás: példa

- ablak mérete: 1 utasítás
- szabályhalmaz:
  $\{\text{mov } reg, 0 \rightarrow \text{xor } reg, reg$   
 $\text{add } reg, 0 \rightarrow ; \text{ elhagyható}\}$

## Eredeti kód

```
add eax,0
mov ebx,eax
mov ecx,0
```

## Optimalizált kód

## Ablakoptimalizálás: példa

- ablak mérete: 1 utasítás
- szabályhalmaz:
  $\{\text{mov } reg, 0 \rightarrow \text{xor } reg, reg$   
 $\text{add } reg, 0 \rightarrow ; \text{ elhagyható}\}$

## Eredeti kód

```
add eax,0
mov ebx,eax
mov ecx,0
```

## Optimalizált kód

; elhagyott utasítás

## Ablakoptimalizálás: példa

- ablak mérete: 1 utasítás
- szabályhalmaz:
  $\{\text{mov } reg, 0 \rightarrow \text{xor } reg, reg$   
 $\text{add } reg, 0 \rightarrow ; \text{ elhagyható}\}$

## Eredeti kód

```
add eax,0
mov ebx,eax
mov ecx,0
```

## Optimalizált kód

; elhagyott utasítás

```
mov ebx,eax
```

## Ablakoptimalizálás: példa

- ablak mérete: 1 utasítás
- szabályhalmaz:
  $\{\text{mov } reg, 0 \rightarrow \text{xor } reg, reg$   
 $\text{add } reg, 0 \rightarrow ; \text{ elhagyható}\}$

## Eredeti kód

```
add eax,0
mov ebx,eax
mov ecx,0
```

## Optimalizált kód

; elhagyott utasítás

```
mov ebx,eax
xor ecx,ecx
```

## Tipikus egyszerűsítések ablakoptimalizáláshoz

- felesleges műveletek törlése: nulla hozzáadása vagy kivonása
- egyszerűsítések: nullával szorzás helyett a regiszter törlése
- nulla mozgatása helyett a regiszter törlése
- regiszterbe töltés és ugyanoda visszaírás esetén a visszaírás elhagyható
- utasításmétlések törlése: ha lehetséges, az ismétlések törlése

## Globális optimalizálás

- a teljes program szerkezetét meg kell vizsgálni
- ennek módszere az [adatáram-analízis](#):
  - Mely változók értékeit számolja ki egy adott alapblokk?
  - Mely változók értékeit melyik alapblokk használja fel?
- lehetővé teszi:
  - az azonos kifejezések többszöri kiszámításának kiküszöbölését akkor is, ha különböző alapblokokban szerepelnek
  - konstansok és változók továbbterjesztését alapblokkok között is
  - elágazások, ciklusok optimalizálását

## Kódkiemelés

**Eredeti kód**

```
if(x < 10)
{
 a = 0;
 b++;
}
else
{
 b--;
 a = 0;
}
```

**Optimalizált kód**

```
a = 0;
if(x < 10)
{
 b++;
}
else
{
 b--;
}
```

## Kódcsülyesztés

**Eredeti kód**

```
if(x < 10)
{
 x = 0;
 b++;
}
else
{
 b--;
 x = 0;
}
```

**Optimalizált kód**

```
if(x < 10)
{
 b++;
}
else
{
 b--;
}
x = 0;
```

Mi lenne, ha itt *kódkiemelést* alkalmaznánk?

## Ciklusok kifejtése

**Eredeti kód**

```
for(int i=0; i<4; ++i)
{
 a += t[i];
}
```

**Optimalizált kód**

```
a += t[0];
a += t[1];
a += t[2];
a += t[3];
```

Mérlegelní kell, hogy a méret és a sebesség mennyire fontos...

## Parciális kifejtés

**Eredeti kód**

```
for(int i=0; i<4; ++i)
{
 a += t[i];
}
```

**Optimalizált kód**

```
for(int i=0; i<4; i+=2)
{
 a += t[i];
 a += t[i+1];
}
```

## Ciklusok összevonása

### Eredeti kód

```
for(int i=0; i<4; ++i)
{
 t1[i] = 0;
}
for(int i=0; i<4; ++i)
{
 t2[i] = 1;
}
```

### Optimalizált kód

```
for(int i=0; i<4; i++)
{
 t1[i] = 0;
 t2[i] = 1;
}
```

## Frekvenciaredukálás

- költséges utasítások „átköltözöttetése” ritkábban végrehajtódó alapblokkba
- példa:
  - ciklusinvariánsnak nevezük azokat a kifejezéseket, amelyeknek a ciklus minden lefutásakor azonos az értékük
  - a ciklusinvariánsok (esetenként) kiemelhetők a ciklusból

### Eredeti kód

```
cin >> a;
cin >> h;
while(h > 0)
{
 cout << h*h*sin(a);
 cin >> h;
}
```

### Optimalizált kód

```
cin >> a;
cin >> h;
double s = sin(a);
while(h > 0)
{
 cout << h*h*s;
 cin >> h;
}
```

## Erős redukció

- a ciklusban lévő költséges művelet (legtöbbször szorzás) kiváltása kevésbé költségessel

### Eredeti kód

```
for(int i=a; i<b; i+=c)
{
 cout << 3*i;
}
```

### Optimalizált kód

```
int t1 = 3*a;
int t2 = 3*c;
for(int i=a; i<b; i+=c)
{
 cout << t1;
 t1 += t2;
}
```

## Funkcionális programozási nyelvek fordítása

Fordítóprogramok előadás (A,C,T szakirány)

## Funkcionális programozási nyelvek

### • programok futása:

- funkcionális eset: függvények kiszámítása
- imperatív eset: állapotváltozások sorozata
- tiszta függvény: olyan függvény, amelynek nincs mellékhatása
- tiszta funkcionális nyelv: csak tiszta függvények írhatók benne
  - akár imperatív nyelveken is lehet „tiszta funkcionális” stílusban programozni
  - az imperatív nyelvekben megszokott változók nem léteznek
- példák: Haskell, Clean, Erlang, OCaml, F#, Lisp, Scheme, ...

## Funkcionális nyelvek sajátosságai

- (általában) nagyon kifinomult a típusrendszerük
  - többalakúság (polimorf típusrendszer)
  - típusellenőrzés helyett típusvezetés
  - típusosztályok
- a függvények „teljes jogú tagjai a nyelvnek” („first class citizens”):
  - függvényeket is lehet paraméterként átadni
  - függvény is lehet visszatérési érték
- lusta (lazy) / szigorú (strict) kiértékelés:
  - lusta: csak akkor értékel ki egy kifejezést, amikor arra valóban szükség van
  - szigorú: minden paramétert kiértékel még a függvényhívás előtt (az imperatív nyelvek szigorúak)

## Funkcionális nyelvek fordítása

- lexikális / szintaktikus / szemantikus elemzés eredménye egy transzformált program, amely
  - típusozott
  - néhány egyszerű nyelvi konstrukcióból épül fel
- ezen a transzformált programon optimalizációs lépéseket hajtunk végre
- kódgenerálás: általában C-re
  - így nem kell újraírni a C fordítóban meglévő sok kifinomult imperatív jellegű optimalizációt
- a végrehajtáshoz szükséges egy futtatókörnyezet is

## A margó szabály

- az utasítások végét és a blokkszerkezetet a sorok behúzása definíálja

### Margó szabálytal

```
f x y = a + b where
 a =
 x * y
 b = x - y
 g x = 2 * x
```

### Explicit jelöléssel

```
f x y = a + b where {
 a =
 x * y;
 b = x - y;
 g x = 2 * x;
```

## A margó szabály megvalósítása

- a lexikális elemző feladata
- minden lexikális elemhez megjegyezzük, hogy hányadik oszlopban van az első karaktere (a fehér szóközök is figyelembe kell venni)
- egy verembe beletesszük a legelső utasítás behúzását
- bizonyos kulcsszavak után (Haskellben pl. where, of) beszűrünk egy „{” tokenet és a következő token behúzását a verembe tesszük
- minden sortörésnél meg kell vizsgálni az új sor behúzást:
  - ha egyenlő a verem tetején lévő értékkel, akkor beszűrünk egy „;”-t
  - ha nagyobb a behúzás, akkor nincs teendő
  - ha kisebb a behúzás, akkor beszűrünk egy „;”-t, majd addig szűrünk be egy-egy „}”-t és veszünk ki egy-egy értéket a veremből, amíg a verem tetején nagyobb érték van az aktuális sor behúzásánál

## Polimorf típusok

- A típusok lehetnek „több alakúak”:

- pl. egészek listája, karakterek listája, listák listája...

```
[1,2,3] :: [Int]
['x'] :: [Char]
[[5], [1,2]] :: [[Int]]
[] :: [a]
[][] :: [[b]]
```

- A példában az a és b típusváltozók.

- Típuslevezetéskor nem a típusok *egyenlőségét* kell vizsgálni, hanem az *egyesíthetőségét*.

- Egyesíthetőség (kb.): A típusváltozók helyettesítésével azonos alakúra hozható-e a két típus?

7

Fordítóprogramok előadás (A,C,T szakirány)

Funkcionális programozási nyelvek fordítása

## Példák típuslevezetésre

- A [[‘x’, ‘y’], []] kifejezés típusát szeretnénk levezetni.

‘x’ :: Char  
‘y’ :: Char

- Char és Char egyesíthetők és az eredmény Char, ezért:

['x', 'y'] :: [Char]

- Az üres lista típusa:

[] :: [a]

- [Char] és [a] egyesíthetők az a → Char helyettesítéssel, ezért:  
[['x', 'y'], []] :: [[Char]]

- A [[‘x’, ‘y’], ‘z’]] kifejezés esetén egyesíteni kellene a [Char] és Char típusokat, ami nem lehetséges. ⇒ Típushiba!

8

Fordítóprogramok előadás (A,C,T szakirány)

Funkcionális programozási nyelvek fordítása

## Szigorú és lusta kiértékelés

- C-ben kiértékelődik minden paraméter:

```
int f(int x, int y)
{
 return x + 1;
}
int a = 4321;
int b = f(a/2, a*515341)
```

- Haskellben csak az első paraméter értékelődik ki:

```
f x y = x + 1
a = 4321
b = f (a/2) (a*515341)
```

- Nem csak hatékonysági kérdés!

- f( 5, a/0 ) futási idejű hiba C-ben
- f 5 (a/0) hiba nélkül lefut, és 6-ot ad Haskellben

9

Fordítóprogramok előadás (A,C,T szakirány)

Funkcionális programozási nyelvek fordítása

## Függvények mint paraméterek vagy visszatérési értékek

- A map egy lista minden elemére végrehajtja az f függvényt:

map f [] = []
map f (első:többi) = (f első) : (map f többi)

- Az addToAll egy olyan függvényt ad eredményül, ami egy lista minden elemét növeli n-nel:

addToList n = map (n+)

- A függvények tehát adatokként kezelhetők, azaz teljes jogú tagjai a nyelvnek.

- Ez C-ben elég nehézkesen, függvénypointerekkel oldható csak meg.

10

Fordítóprogramok előadás (A,C,T szakirány)

Funkcionális programozási nyelvek fordítása

## A lusta kiértékelés és a függvények teljes jogú tagságának megvalósítása

- Az f (a/2) (a\*515341) hívásban „kiértékeletlenül” kell átadni a paramétereket, hogy az f függvény dönthesse el, melyikre van ténylegesen szüksége.

```
f x y = x + 1
```

- A map függvény első paraméterében egy függvényt kell átadnunk, ami szintén egy „kiértékeletlen” kifejezés.
- Az addToList függvény eredménye egy függvény, amit még alkalmaznunk kell egy listára, hogy kiértékelhető legyen.

**Hogyan lehet kiértékeletlen kifejezéseket kezelni?**

## A „kiértékeletlen kifejezések” kezelése

- Készítünk egy objektumot, amely tartalmaz

- egy pointert a függvényre
- további pointereket a függvény meglévő paramétereire

- Az ilyen objektumok (closure) átadhatók függvényeknek, visszaadhatók visszatérési értékként.

- Ha minden paraméter rendelkezésre áll egy closure-ben és szükséges válik a kiértékelése, akkor elvégezhető a művelet.

11

Fordítóprogramok előadás (A,C,T szakirány)

Funkcionális programozási nyelvek fordítása

12

Fordítóprogramok előadás (A,C,T szakirány)

Funkcionális programozási nyelvek fordítása

- Egy closure-ben a paraméterek is és a függvény is lehet további kiértékeletlen kifejezés.
- Ezért egy teljes kifejezés ábrázolható egy fával, általánosabb esetben egy gráffal.
- Kódgenerálás a modern lusta kiértékelésű funkcionális nyelveken írt programokhoz:
  - a programban lévő main függvényből felépítünk egy gráfot
  - a programban lévő függvénydefiníciókból egy-egy gráfátíró szabályt készítünk
- A programok lefutása:
  - a futtatórendszer minden lépésben kiválaszt egy megfelelő részt a gráfból
  - átírja valamelyik szabály alapján
  - ezt addig ismételgeti, amíg
    - a gráf egy adottá egyszerűsödött (ez a végeredmény)
    - vagy már nem egyszerűsíthető tovább