Alapfogalmak

1. Miről mire fordít a fordítóprogram?
   - Magas szintű programozási nyelvről gépi kódra
1. Miről mire fordít az assembler?
   - Assembly kódról gépi kódra
1. Mi a különbség a fordítóprogram és az interpreter között?
   - A fordítóprogram előállít egy gépi kódú programot, az interpreter futás közben értelmezi és végrehajtja
1. Mi a virtuális gép?
   - A fordítóprogram által előállított bájtkódot (a virtuális gép gépi kódját) értelmező és végrehajtó szoftver.
1. Mi a különbség a fordítási és a futási idő között?
   - Futásidőben a fordítóprogram dolgozik, futásidőben a program fut
1. Mi a feladata az analízisnek (a fordítási folyamat első fele) és milyen rész- feladatokra bontható?
   - A forrásban leírt program belső reprezentációjának előállítása
   - Alfeladatai:
     - Forráskód olvasása
     - Lexikális elemzés
     - Szintaktikus elemzés
     - Szemantikus elemzés
1. Mi a feladata a szintézisnek (a fordítási folyamat második fele) és milyen részfeladatokra bontható?
   - A belső reprezentációból a tárgykód előállítása
   - Részfeladatai:
     - Kódgenerálás
     - Optimalizálás
     - Tárgykód kiírása háttértárra
1. A fordítóprogram mely részei adhatnak hibajelzést?
   - A lexikális elemző, a szintaktikus elemző és a szemantikus elemző
9. Mi a lexikális elemző feladata, bemenete, kimenete?
   - Feladata a lexikális egységek felismerése, lexikális hibák felderítése
   - Bemenete karaktersorozat
   - Kimenete szimbólumsorozat, lexikális hibák
9. Mi a szintaktikus elemző feladata, bemenete, kimenete?
   - Feladata a program szerkezetének felismerése és a szerkezet ellenőrzése
   - Bemenete szimbólumsorozat
   - Kimenete a szintaxisfa, szintaktikus hibák
9. Mi a szemantikus elemző feladata, bemenete, kimenete?

- A forrásprogram környezetfüggő szabályainak ellenőrzése
  - típusellenőrzés
  - láthatóság-ellenőrzés
  - eljárások hívásának ellenőrzése
- Bemenete szintaktikusan elemzett program: szintaxisfa, szimbólumtábla
- Kimenete szemantikusan elemzett program, szemantikus hibák

9. Mi a kódgenerátor feladata, bemenete, kimenete?

- Feladata a forráskóddal ekvivalens tárgykód elkészítése
- Bemenete a szemantikusan elemzett program
- Kimenete tárgykód (assembly vagy gépi kód)

9. Mi a kódoptimalizáló feladata, bemenete, kimenete?

- Feladata a bemenetnél valamilyen szempontból (gyorsabb, kisebb stb.) jobb kód készítése
- Bemenete tárgykód
- Kimenete optimalizált tárgykód

9. Mi a fordítás menetszáma?

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

Lexikális elemző

15. Milyen nyelvtanokkal dolgozik a lexikális elemző?

- Chomsky 3. nyelvosztályba tartozó, reguláris nyelvtanokkal

15. Hogy épülnek fel a reguláris kifejezések?

- Az alapelemekből a konstrukciós és kényelmi műveletekkel állíthatók elő
- Alapelemek: üres halmaz, üres szót tartalmazó halmaz és az egy terminálist tartalmazó halmazok
- Konstrukciós műveletek: konkatenáció, unió és lezárás
- Kényelmi műveletek: pozitív lezárás, opcionális részlet

15. Hogy épülnek fel a véges determinisztikus automaták?

- Elemei: ábécé, állapotok, átmeneti függvény, kezdőállapot, végállapotok halmaza

15. Milyen elv szerint ismeri fel a lexikális elemző a lexikális elemeket?

- Mindig a lehető leghosszabb karaktersorozatot ismeri fel.

15. Mi a szerepe a lexikális elemek sorrendjének?

- Egyfajta „prioritást” adhatunk meg velük a szimbólumok közt, így pl. a kulcsszavakat különválaszthatjuk az egyéb azonosítóktól.

15. Mi a különbség a kulcsszavak és a standard szavak között?

- A kulcsszavak jelentése nem definiálható felül, a standard szavaké igen

15. Mi az előfeldolgozó fázis feladata?

- A direktívák, makrók, beillesztett fájlok kezelése

15. Mutass példát olyan hibára, amelyet a lexikális elemző fel tud ismerni és olyanra is, amelyet nem!

- + Illegális karakter: addr?ess

◦ - Elgépelt kulcsszó: whille

Szintaktikus elemzés alapfogalmai

23. Mikor ciklusmentes egy nyelvtan?

◦ Ha nincs benne$A \overset{+}{\Rightarrow} A$típusú levezetés

23. Mikor redukált egy nyelvtan?

◦ Ha nincsenek benne felesleges nyelvtani jelek

23. Mikor egyértelmű egy nyelvtan?

◦ Ha minden mondathoz pontosan egy szintaxisfa tartozik

23. Mi a különbség a legbal és legjobb levezetés között?

◦ Legbal esetén mindig a legbaloldalibb, legjobb esetén mindig a legjobboldalibb terminálist helyettesítjük

23. Mi a különbség a felülről lefelé és az alulról felfelé elemzés között?

◦ 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.

23. Mi az összefüggés az elemzési irányok és a legbal, illetve legjobb levezetés között?

◦ A felülről lefelé elemzés legbal levezetés, az alulról felfelé elemzés a legjobb levezetés inverze.

23. Milyen alapvető stratégiák használatosak a felülről lefelé elemzésekben?

◦ Backtrack (lassú)

◦ Előreolvasás (LL)

23. Milyen alapvető stratégiák használatosak az alulról felfelé elemzésekben?

◦ Backtrack (lassú)

◦ Precedenciák

◦ Előreolvasás (LR)

LL elemzések

31. Definiáld a $FIRST_k(\alpha)$ halmazt, és röviden magyarázd meg a definíciót!

◦ $FIRST_k(\alpha) = \{x \mid \alpha \overset{*}{\Rightarrow} x\beta \wedge |x| = k\} \cup \{x \mid \alpha \overset{*}{\Rightarrow} x \wedge |x| < k\}$

◦ az α mondatformából levezethető

▪ k-nál hosszabb terminális sorozatok k hosszúságú kezdőszeletei

▪ k-nál nem hosszabb terminális sorozatok

31. Definiáld az LL(k) grammatikákat és röviden magyarázd meg a definíciót!

◦ $S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_1\beta \overset{*}{\Rightarrow} wx$
$S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_2\beta \overset{*}{\Rightarrow} wy$ levezetéspárra $FIRST_k(x) = FIRST_k(y) \Rightarrow \alpha_1 = \alpha_2$

◦ a levezetés tetszőleges pontján a szöveg következő k terminálisa meghatározza az alkalmazandó levezetési szabályt

31. Definiáld a $FOLLOW_k(\alpha)$ halmazt és röviden magyarázd meg a definíciót!

◦ $FOLLOW_k(\alpha) = \left\{x \mid S \overset{*}{\Rightarrow} \beta\alpha\gamma \wedge x \in FIRST_k(\gamma) \setminus \{\varepsilon\}\right\} \cup \left\{\# \mid S \overset{*}{\Rightarrow} \beta\alpha\right\}$

- a levezetésekben az α után előforduló k hosszúságú terminális sorozatok

31. Definiáld az egyszerű LL(1) grammatikát!

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

31. Mi az egyszerű LL(1) grammatikáknak az a tulajdonsága, amire az elemző épül?

- Az azonos nemterminálishoz tartozó szabályok jobboldalai különböző terminálissal kezdődnek.

31. Mit csinál az egyszerű LL(1) elemző, ha a verem tetején az A nemterminális van és a bemenet következő szimbóluma az a terminális?

- ha van $A \rightarrow a\alpha$ szabály: A helyére aα és bejegyzés a szintaxisfába
- különben: hiba

31. Definiáld az ε-mentes LL(1) grammatikát!

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

31. Mi az ε-mentes LL(1) grammatikáknak az a tulajdonsága, amire az elemző épül?

- ε-mentes LL(1) grammatika esetén az egy nemterminálishoz tartozó szabályok jobboldalainak FIRST1 halmazai diszjunktak.

31. Mit csinál az ε-mentes LL(1) elemző, ha a verem tetején az A nemterminális van és a bemenet következő szimbóluma az a terminális?

- ha van $A \rightarrow \alpha$ szabály, amelyre a $\in \mathrm{FIRST}_1(\alpha)$: A helyére α és bejegyzés a szintaxisfába
- különben: hiba

31. Definiáld az LL(1) grammatikát!

- $S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_1\beta \overset{*}{\Rightarrow} wx$
  $S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_2\beta \overset{*}{\Rightarrow} wy$
  levezetéspárra $FIRST_1(x) = FIRST_1(y) \Rightarrow \alpha_1 = \alpha_2$

31. Mi az LL(1) grammatikáknak az a tulajdonsága, amire az elemző épül?

- a levezetés tetszőleges pontján a szöveg következő terminálisa meghatározza az alkalmazandó levezetési szabályt

31. Mit csinál az LL(1) elemző, ha a verem tetején az A nemterminális van és a bemenet következő szimbóluma az a terminális?

- ha van $A \rightarrow \alpha$ szabály, amelyre a $\in \mathrm{FIRST}_1(\alpha\mathrm{FOLLOW}_1(A))$: A helyére α és bejegyzés a szintaxisfába
- különben: hiba

31. Milyen komponensei vannak az LL(1) elemzőknek?

- elemző táblázat
- verem

31. Hogyan épülnek fel a rekurzív leszállásos elemzőben a nemterminális szimbólumokhoz rendelt eljárások?

◦
```
void A() {
       if(aktualis_szimbolum ∈ FIRST1(α1FOLLOW1(A))) {
          // alfa_1 programja
       }
       ...
       else if(aktualis_szimbolum ∈ FIRST1(αnFOLLOW1(A))) {
          // alfa_n programja
       } else {
           hiba(...);
       }
   }
```

LR elemzések

45. Mit jelentenek a léptetés és redukálás műveletek?

- Léptetés: a bemenet következő szimbólumát a verem tetejére helyezzük
- Redukálás: A verem tetején levő szabály-jobboldalt helyettesítjük a megfelelő nemterminális szimbólummal

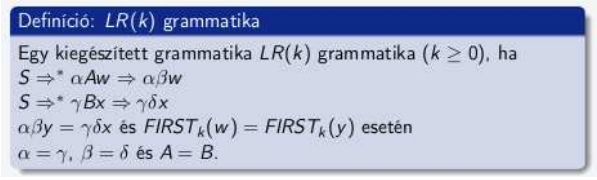
45. Mi a kiegészített grammatika és miért van rá szükség?

- 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 mindig kiegészített nyelvtanokat fogunk használni

45. Mi a nyél szerepe az alulról felfelé elemzésekben?

- Mindig a nyelet (legbaloldali egyszerű részmondat) keressük, hogy aszerint tudjunk redukálni

45. Mondd ki az LR(k) grammatika definícióját és magyarázd meg!

- 

- A grammatika k szimbólum előreolvasásával egyértelműen meghatározza a nyelet. Azaz, ha van két levezetésem, ami ugyanúgy kezdődik balról olvasva, és ugyanazt a k szimbólumot kapom előreolvasással bármelyik ponton, akkor egyértelmű lesz, hogy melyik szabály szerint kell redukálnom.

45. Hogyan határozza meg az LR(0) elemző véges determinisztikus automatája, hogy léptetni vagy redukálni kell?

- Léptetés: nem elfogadó állapotban, Redukálás: elfogadó állapotokban.

45. Hogy néz ki egy LR(0)-elem és mi a jelentése?

- [A->$\alpha_1.\alpha_2$]
- jelentése: Az aktuális állapotban még az A-> $\alpha_1\alpha_2$ szabály elemzésében α_1-et már olvastuk és hátra van még α_2

45. Milyen műveletek segítségével állítjuk elő a kanonikus halmazokat és mi ezeknek a szerepe?

- A kanonikus halmazokat rekurzívan állítjuk elő a closure és read műveletekkel
- closure (X): az X LR(0) elemből az összes következő lehetséges LR(0) elemet előállítja egy

halmazba, amerre mehet az elemzés, ez állítja elő lényegében az adott X elemhez tartozó „kanonikus halmazt”. Pl.: closure([A->α_1.Bα_2]) = {[A->α_1.Bα_2] ,[B->.b]}

- read (I,X): a „következő” kanonikus halmazt állítja elő, az I egy LR(0) elemének „továbbléptetésével”. Pl.: J:=read(I,X), ha I-nek eleme [A->α_1.Bα_2] , akkor J-nek részhalmaza lesz closure([A->α_1B.α_2])

45. Mi köze van az LR(0) kanonikus halmazoknak az LR(0) elemző véges determinisztikus automatájához?

- Minden kanonikus halmaz megfelel egy-egy vda állapotnak (bijekció, homomorf reprezentáció)

45. Hogyan határozzuk meg az LR(0) elemző automatájában az átmeneteket?

- A kanonikus halmazok közötti read művelet által olvasott elem adja meg az átmeneti feltételt

45. Hogyan határozzuk meg az LR(0) elemző automatájában a végállapotokat?

- Amennyiben van a kanonikus halmazban olyan elem, aminek a szabály jobboldalának végén van a „.” , akkor végállapot.

45. Mondd ki az LR(0)-elemzés nagy tételét!

- Az LR(0) elemzés nagy tétele szerint egy γ járható prefix által kiváltott állapotban az elemző automata állapotához tartozó kanonikus halmaz éppen a γ járható prefixre érvényes LR(0) elemeket tartalmazza

45. Milyen konfliktusok lehetnek az LR(0) elemző táblázatban?

- Ha nem LR(0) a nyelv, de teljesül az LR(k): léptetés/redukálás konfliktus (ekkor az automatában végállapotból is vezet ki átmenet, a táblázatban az akció oszlopba „léptetés, redukálás” kerülne, ha tudna)

45. Milyen esetben ír elő redukciót az SLR(1) elemzés?

- Abban az esetben ír elő redukciót az SLR(1), ha verem tetején levő elemek sorozata (nyél) pont egy olyan A nemterminális-ból vezethető le, aminek a $FOLLOW_1$(A) halmaza tartalmazza az előreolvasott szimbólumot
- másképp: az adott állapot kanonikus halmazában van egy végigolvasott LR(0) elem (pl.: [A->$\alpha_1\alpha_2$.]) és az adott szabályhoz tartozó nemterminális (pl itt: A) $FOLLOW_1$ halmaza tartalmazza az előreolvasott szimbólumot

45. Hogy néz ki egy LR(1)-elem és mi a jelentése?

- [A->α_1.α_2, a], ahol A->α_1.α_2 az elem magja és az *a* nemterminális (vagy #) az előreolvasási szimbóluma
- jelentése: Az aktuális állapotban még az A-> $\alpha_1\alpha_2$ szabály elemzésében α_1-et már olvastuk és hátra van még α_2, valamint, ha elolvastuk a szabályt, akkor *a* nemterminálisnak kell következnie

45. Milyen esetben ír elő redukciót az LR(1) elemzés?

- Abban az esetben ír elő redukciót az SLR(1), ha az adott állapot kanonikus halmazában van egy végigolvasott LR(1) elem (pl.: [A-> $\alpha_1\alpha_2$., a]) és az előreolvasott szimbólum megegyezik az elem előreolvasási szimbólumával (pl itt: a)

45. Miért van általában lényegesen több állapota az LR(1) elemzőknek, mint az LR(0) (illetve SLR(1)) elemzőknek?

- Amiatt, mert az LR(0) globális FOLLOW halmazokkal dolgozik, és az LR(0) elemeknek nincs

előreolvasási szimbólumuk. AZ LR(1) azzal, hogy elemenkénti előreolvasási szimbólumokkal dolgozik, nagyságrendi állapotnövekedést hoz be.

45. Mikor nevezünk két LR(1) kanonikus halmazt összevonhatónak?

• Abban az esetben vonható össze két LR(1) kanonikus halmaz ha minden eleme páronként ugyanazzal a maggal rendelkezik, de előreolvasási szimbólumukban különbözik.

45. Hogyan kapjuk meg az LALR(1) kanonikus halmazokat?

• Vagy az összevonható LR(1) halmazok összevonásával

• Vagy az LR(0) halmazokból generált törzshalmazok előállításával majd az örökölt és spontán generált előolvasási szimbólumok meghatározásával

45. Milyen fajta konfliktus keletkezhet a halmazok összevonása miatt az LALR(1) elemző készítése során?

• redukálás/redukálás konfliktus

45. Milyen lépésekből áll az LR elemzők hibaelfedő tevékenysége?

• Lényeg: szinkronizálni kell a vermet az inputtal és folytatni az elemzést.

• Megvalósítás: **hibaalternatívák** (új szabály) a „fontos” szabályokhoz, **error** szimbólum (új szimbólum) bevezetésével

• lépések:

1. hiba detektálása esetén hibakezelő rutin meghívása
2. a verem tetejéről addig töröl, amíg olyan állapotba nem kerül, ahol már lehet az **error** szimbólummal lépni
3. a verembe lépteti az **error**-t
4. a bemeten addig ugorja át a soron következő terminálisokat, amíg a hibaalternatíva építését folytatni nem tudja

Az if − then − else probléma

66. Mi az if − then − else probléma?

◦ A if (F) if (S) U else U részmondathoz több szintaxisfa is tartozik.

66. Hogyan kell értelmezni a gyakorlatban az egymásba ágyazott elágazásokat, ha az az if − then − else probléma miatt nem egyértelmű?

◦ Az else ág az őt közvetlenül megelőző if ághoz tartozik

66. Hogyan oldják meg az if − then − else problémát az LR elemzők?

◦ A léptetés-redukálás konfliktust léptetéssel oldják fel

66. Mire kell figyelni programozási nyelvek tervezésekor, ha el akarjuk kerülni az if − then − else problémát?

◦ Az elágazás végét kulcsszóval jelölni kell

Szimbólumtábla

71. Milyen információkat tárolunk a szimbólumtáblában a szimbólumokról (fajtájuktól függetlenül)?

◦ szimbólum neve

◦ szimbólum attribútumai

▪ definíció adatai

- típus
- tárgyprogram-beli cím
- definíció helye a forrásprogramban
- szimbólumra hivatkozások a forrásprogramban

71. Milyen információkat tárolunk a szimbólumtáblában a változókról?
- típus
- módosító kulcsszavak (static, const stb.)
- cím a tárgyprogramban

71. Milyen információkat tárolunk a szimbólumtáblában a függvényekről?
- Szignatúra
  - paraméterek típusa
  - visszatérési típus
- módosítók
- cím a tárgyprogramban

71. Milyen információkat tárolunk a szimbólumtáblában a típusokról?
- Egyszerű típusokról: méret
- rekord: mezők nevei és típusleírói
- tömb: elem típusleírója, index típusleírója, méret
- intervallum: elem típusleírója, minimum, maximum
- unió: a lehetséges típusok leírói, méret

71. Milyen információkat tárolunk a szimbólumtáblában az osztályokról?
- Attribútumok nevei, láthatóságai és típusleírói
- névtér

71. Mi a szimbólumtábla két alapvető művelete és mikor használja ezeket a fordítóprogram?
- Keresés a szimbólum használatakor
- Beszúrás új szimbólum megjelenésekor (keres is)

71. Mi a változó hatóköre?
- Ahol a deklarációja érvényben van

71. Mi a változó láthatósága?
- Ahol a nevével hivatkozni lehet rá (a hatókör részhalmaza)

71. Mi a változó élettartama?
- Amíg memóriaterület van hozzárendelve

71. Hogyan kezeljük változó hatókörét és láthatóságát szimbólumtáblával?
- A szimbólumokat egy verembe tesszük.
- Keresés:
  - a verem tetejéről indul
  - az első találatnál megáll
- Blokk végén a hozzá tartozó szimbólumokat töröljük.

71. Milyen szerkezetű szimbólumtáblákat ismersz?
- Verem

◦ fa

◦ hash

71. Miben tér el a névterek és blokkok kezelése a szimbólumtáblában?

◦ 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)

Szemantikus elemzés

83. Miért nem a szintaktikus elemző végzi el a szemantikus elemzés feladatait?

• Mert a környezetfüggetlen nyelvtan kezelése (szintaktikus elemző, L2-es nyelvtan) jóval egyszerűbb feladat, mint az L1-es környezetfüggőé. Amit pedig meg lehet oldani egyszerű eszközökkel, azt érdemes azzal megoldani.

83. Mi a különbség a statikus és a dinamikus típusozás között?

• Statikus típusozásnál fordítási időben kalkulálódik és ellenőrződik a típus

• Dinamikusnál futási időben derül ki és ellenőrződik minden utasításnál a típus.

83. Mi a különbség a típusellenőrzés és a típuslevezetés között?

• A típusellenőrzésnél minden típus a deklarációkban adott, a kifejezések egyszerű szabályok alapján típusozhatóak (pl.: C++, Java)

• A típuslevezetésnél a fordítóprogram „találja ki” a kifejezések típusát definíciójuk, használatuk alapján, jóval bonyolultabb algoritmusokkal (pl.: Haskell)

83. Mi a fordítóprogram teendője típuskonverzió esetén?

• Típuskonverzió (cast) esetén esetén a következőket kell tenni:

  • típusellenőrzésnél át kell írni a kifejezés típusát

  • ha szükséges, akkor a tárgykódba generálni kell a típuskonverzió utasításait (pl.: int és double reprezentációja különbözik!)

83. Mik az akciószimbólumok?

• Egy **fordítási grammatika** szabályaiban az akciószimbólumokkal jelöljük, hogy milyen elemzési tevékenységekre van szükség. Jel.: @Tevékenység Pl.: @Ellenőrzés, @Feljegyzés

• Ezek a tevékenységek a *szemantikus rutinok*.

83. Mik az attribútumok?

• A szemantikus rutinok információ átadási mechanizmusában használt struktúrák, melyeket a szintaktikus elemző szimbólumaihoz rendelünk. Jelölés: A.x, y <=> Az A szimbólumhoz az x és y attribútumokat rendeljük

83. Hogyan kapnak értéket az attribútumok?

• Az attribútumok a szemantikus rutinokban kapnak értéket.

83. Mi a szintetizált attribútum?

• A szintetizált attribútum a helyettesítési szabály bal oldalán áll abban a szabályban, amelyikhez az őt kiszámoló szemantikus rutin tartozik. Az információt **alulról felfelé** közvetíti.

• Pl:

szabály: **Kifejezés0.t** → Kifejezés1.t + Kifejezés2.t

szemantikus rutin: **Kifejezés0.t** := int

83. Mi a kitüntetett szintetizált attribútum?

- Olyan attribútumok, amelyek terminális szimbólumokhoz tartoznak és kiszámításukhoz nem használunk fel más attribútumokat
- Pl: Kifejezés.t → konstans.t
- Az információt általában a lexikális elemző szolgáltatja

83. Mi az örökölt attribútum?

- A szintetizált attribútum a helyettesítési szabály jobb oldalán áll abban a szabályban, amelyikhez az őt kiszámoló szemantikus rutin tartozik. Az információt **alulról felfelé** közvetíti.
- Pl:

szabály: Változólista.t → **változó.t** Folytatás.t

szemantikus rutin: **változó.t** := Változólista.t

83. Mivel egészítjük ki a nyelvtan szabályait attribútum fordítási grammatikák esetében?

- A nyelvtan szabályait attribútumokkal, és az attribútumokra vonatkozó logikai feltételekkel, valamint az attribútumokat kalkuláló szemantikus rutinnal egészítjük ki. Ha logikai feltétel nem teljesül, a szemantikus fordító hibát kell, hogy dobjon. Minden szintaxisfában minden attribútumértéket pontosan egy szemantikus rutin határozhat meg.

83. Mi a direkt attribútumfüggőség?

- Ha az Y.b attribútumot kiszámoló szemantikus rutin használja az X.a attribútumot, akkor (X.a, Y.b) egy direkt attribútumfüggőség. Ezek a függőségek függőségi gráfban ábrázolhatóak.

83. Mi a jól definiált ATG?

- Jól definiált attribútum fordítási grammatika olyan attribútumfordítási grammatika amelyre igaz, hogy a grammatika által definiált nyelv mondataihoz tartozó minden szintaxisfában minden attribútum értéke egyértelműen kiszámítható.
- Mj.: Jól definiált ATG-hez tartozó szintaxisfák függőségi gráfjaiban nincsenek körök!

83. Mi az S − AT G? Milyen elemzésekhez illeszkedik?

- AZ S-ATG kizárólag szintetizált attribútumokat használó ATG. Alulról felfele elemzésekhez illeszkedik.

83. Mi az L − AT G? Milyen elemzésekhez illeszkedik?

- Olyan ATG, amelyben minden $A \rightarrow X_1X_2...X_n$ szabályban az attribútumértékek az alábbi sorrendben meghatározhatóak:
- A örökölt attribútumai,
- X_1 örökölt attribútumai, X_1 szintetizált attribútumai
- X_2 örökölt attribútumai, X_2 szintetizált attribútumai
- …
- X_n örökölt attribútumai, X_n szintetizált attribútumai

Assembly

97. Mi az assembly?

◦ Gépközeli programozási nyelv, amelyben

- az adott processzor utasításai használhatók
- általában nincsenek programkonstrukciók, típusok, osztályok
- a futtatható programban pontosan ugyanazok az utasítások lesznek, amiket a programba írunk

97. Mi az assembler?

- Az assembly programok fordítója

97. Milyen főbb regisztereket ismersz (általános célú, veremkezeléshez, adminisztratív célra)?

- EAX: accumulator (számításokhoz)
- EBX: base (kezdőcímek)
- ECX: counter (számláló)
- EDX: data (egyéb adatok, segédregiszter)
- ESI: source index (string másolásnál a forrás címe)
- EDI: destination index (string másolásnál a cél címe)
- ESP: stack pointer (veremmutató)
- EBP: base pointer (aktuális alprogram veremrésze)
- EIP: instruction pointer
- EFLAGS: jelzőbitek

97. Mi köze van egymáshoz az EAX, AX, AL, AH regisztereknek?

- A 32 bites EAX regiszter alsó 16 bitje az AX, az AX felső 8 bitje az AH, alsó 8 bitje az AL.

97. Milyen aritmetikai utasításokat ismersz assemblyben?

- INC (++)
- DEC (–)
- ADD (+)
- SUB (-)
- MUL (*)
- DIV (/)

97. Mutasd be a logikai értékek egy lehetséges ábrázolását és a műveleteik megvalósítását assemblyben!

- AL regiszterben 8 biten; legyen 0 (00000000) a hamis, 1 (00000001) az igaz. Így a bitenkénti műveletek megfelelnek a logikaiaknak.
- AND
- OR
- XOR
- NOT

97. Milyen feltételes ugró utasításokat ismersz?

- Je, jne: equal
- jb, jnb: below
- ja, jna: above
- jl, jnl, jle: less (előjeles)
- jg, jng, jge: greater (előjeles)

97. Hogyan kapják meg a feltételes ugró utasítások a CMP utasítás eredményét?

◦ Az EFLAGS regiszter megfelelő bitjein

97. Milyen veremkezelő utasításokat ismersz assemblyben, és hogyan működnek ezek?

◦ PUSH: betesz a verembe (a forrásból a verem tetejére kerül a 2 vagy 4 bájtos változó, ESP nő 2 vagy 4 bájttal

◦ POP: kivesz a veremből (a verem tetejéről a célba kerül a 2 vagy 4 bájtos változó, ESP csökken 2 vagy 4 bájttal

97. Melyik utasításokkal lehet alprogramot hívni és alprogramból visszatérni assemblyben?

◦ CALL

◦ RET

Kódgenerálás

107. Hogyan generálunk kódot egyszerű típusok értékadásához?

• Kifejezést az eax-be kiszámító kód

• Mov [Változó], eax

107. Hogyan generálunk kódot egy ágú elágazáshoz?

• *Feltételt az **al**-be kiszámító kód*

· `cmp al, 1`

· `jne near Vége`

• *Then ág kódja*

• `Vége:`

107. Hogyan generálunk kódot több ágú elágazáshoz?

• *1. Feltételt az **al**-be kiszámító kód*

· `cmp al, 1`

· `jne near Feltétel2`

• *első feltétel kódja*

· `jmp Vége`

· `...`

· `Feltételn:`

• *n. Feltételt az **al**-be kiszámító kód*

· `cmp al, 1`

· `jne near Else`

• *n-edik feltétel kódja*

· `jmp Vége`

· `Else: az else ág kódja`

· `Vége:`

107. Hogyan generálunk kódot elöltesztelő ciklushoz?

· `Eleje:`

• Ciklusfeltételt az al-be kalkuláló kód

- `cmp al,1`
- `jne near Vége`
- ciklusmag kódja
- `jmp Eleje`
- `Vége:`

107. Hogyan generálunk kódot hátultesztelő ciklushoz?

- `Eleje:`
- ciklusmag kódja
- Ciklusfeltételt az al-be kalkuláló kód
- `cmp al,1`
- `je near Eleje`
- `Vége:`

107. Hogyan generálunk kódot for ciklushoz?

- Egy for (int i=0; i<15; i++){ ciklusmag } kódhoz:
  - `mov ecx, 15`
  - `Eleje:`
  - `push ecx`
  - ciklusmag kódja
  - `pop ecx`
  - `loop Eleje`

107. Hogyan generáljuk kezdőérték nélküli statikus változó definíciójának assembly kódját?

- Változónév: resx y , a .bss szegmensben
- x és y az adott változó típusának reprezentációjának méretétől függ, pl. int esetén 4 byte, ami pl. így foglalható le:
- `section .bss`
- `Változó: resb 4`

107. Hogyan generáljuk kezdőértékkel rendelkező statikus változó definíciójának assembly kódját?

- `section .data`
- `Változó: dd 5`
- Ennek jelentése: lefoglalok a Változó névre egy duplaszót, és 5 a kezdőértéke

107. Hogyan generáljuk aritmetikai kifejezés kiértékelésének assembly kódját? (konstans, változó, beépített függvény)

- *ökölszabály: a végeredménynek mindig az eax regiszterbe kell kerülnie.*
- Egyszerű kifejezésnél (konstans, változó) ez egyszerű:
  - `mov eax, 25`
  - `mov eax, [Változó]`
- Összetett infix operátoros kifejezésnél, pl. a + b
  - *b kiértékelése eax-be*

- push eax
- *a kiértékelése eax-be*
- pop ebx
- add eax, ebx

107. Mutasd meg a különbséget a mindkét részkifejezést kiértékelő és a rövid- záras logikai operátorok assembly kódja között!

- kfif1 ÉS kif2
- mindkettőt kiértékelve:
  - *kif2 kiértékelése al-be*
  - push ax
  - *kif1 kiértékelése al-be*
  - pop bx
  - and ax, bx
- rövidzárral:
  - ***kif1 kiértékelése al-be***
  - ***cmp al, 1***
  - ***jne Vége***
  - *push ax*
  - *kif2 kiértékelése al-be*
  - *pop bx*
  - *and al, bl*
  - *Vége:*

107. Hogyan generáljuk a goto utasítás assembly kódját?

- Megfelelő Label létrehozása, vigyázni kell, hogy ne essen egybe másik címkével pl. ciklus címkével
- goto-nál jmp Label
- **figyelni kell**, hogy ha pl ciklusból ugrunk ki, akkor az ecx-et visszaállítani, alprogramok esetén még bonyolultabb lehet a helyzet.

107. Miért nehéz a break utasítás kódgenerálását megoldani S − AT G használata esetén?

- Az S-ATG csak szintetizált attribútumokat használ, a break-hez pedig örökölt attribútumra van szükség, mert pl. ciklus esetén:
- először generáljuk a ciklusmag (pl.: *Utasítások*) kódját (a break utasítással együtt)
- a ciklus kódját pedig később (pl.: „*ciklus → WHILE feltétel DO Utasítások DONE”* szabálynál)
- DE a break utasítás kódjához már szükség lenne a ciklus kódjából a Vége címkére
- megoldás lehet: feljegyezni, hogy volt-e break (ez szintetizált, pl. Utasítások.containsBreak), és a címke helyét kihagyni, majd a ciklus kód generálásakor kitölteni

107. Mit csinál a call és a ret utasítás?

- call Címke

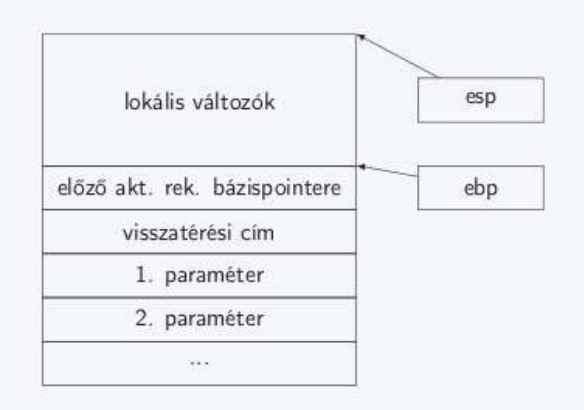
    - eip-t verembe (ez a call utáni utasítás címe, a *visszatérési cím*)
    - Átadja a vezérlést a Címke címkéhez (mint egy ugró utasítás)
- ret
    - kiveszi verem tetején levő címet és az eip-be teszi (~ pop eip)
    - a program ennél a címnél folytatja a működését

107. Hogyan adjuk át assemblyben az alprogramok paramétereit és hol lesz a lefutás után a visszatérési érték (C stílus esetén)?

- A call hívása előtt a paramétereket a verembe rakjuk, fordított sorrendben (utolsó alulra, első legfelülre)
- visszatérési érték az eax-be
- a hívó állítja vissza a vermet (C-stílusban az alprogram írónak kényelmesebb)

107. Hogyan épül fel az aktivációs rekord?

- Az aktivációs rekord minden épp futó alprogram példányhoz készül és a veremben foglal helyet. Ez a stackframe más néven. Szerkezete:



107. Mi a bázismutató és melyik regisztert szoktuk erre a célra felhasználni?

- Általában a verem (relatív) címzésére jó, az esp (a verem tényleges címét tartalmazza) elállítása nélkül lehet vele dolgozni így, az EBP regisztert használjuk erre

107. Hol tároljuk alprogramok lokális változóit?

- A verem tetejére kerülnek

107. Mi a különbség az érték és a hivatkozás szerinti paraméterátadás assembly kódja között?

- Érték szerinti átadásnál a verembe az értéket másoljuk, és a verem értékét módosítja az alprogram → ekkor nem hat vissza az átadott változó értékére
- Hivatkozás szerinti átadás esetén a változó címét másoljuk, és a hivatkozás már a veremben levő cím alapján kell, hogy történjen, pl.:
    - mov eax, [ebp – p] ; cím másolása eax-be
    - mov eax, [eax] ; érték másolása eax-be

107. Milyen csoportokba oszthatók a változók tárolásuk szerint és a memória mely részeiben tároljuk az egyes csoportokba tartozó változókat?

- A változókat lehet dinamikusan és statikusan tárolni
- Statikus:

    - előre ismert számú változó, ismert mérettel, global v. static változók
    - .data vagy .bss szekciókban kell foglalni helyet nekik
- Dinamikus:
    - blokk-szerkezethez kötődő, lokális változók: verem
    - tetszőleges élettartamú változók: heap memória