5.Fordítási algoritmusok története:
A programnyelvek szolgálnak arra, hogy a programozó a megoldandó problémáját a számítógéppel közölje. Fogalmazni a magasszintű nyelveken a legkönyebb, de a gép csak az alacsony szintű gépi kódotfogadja el. **A programnyelvek hierarchiája:** 1, **gépi kód**: 0,1-esek sorozata, numerikus kódok. 2, **assembly-nyelv:** a gépi kód szimbólikus megfelelője, az utasítások és a memória címzése is szimbólumokkal történik. Tartalmazhat makró-utasításokat is. 3, **magasszintű nyelv:** olyan jelölésrendszert használ, ami közel áll a probléma megfogalmazásához: i, felhasználó orientált: álalános problémák, ii, probléma orientált: speciális problémák. A fordítópogram inputja a forrásnyelvű program, outputja a tárgyprogram. **A programnyelvek típusai:** 1**, assembler**: forrásnyelv: assembly-nyelv. tárgynyelv: gépi kód. 2, **compiler**: forrásnyelv: magasszintű nyelv. tárgynyelv: gépi kód, esetleg assembly-nyelv. Ha P forrásnyelvű, Q tárgynyelvű program, T pedig a fordítás transzformációja, akkor: Q = T(P). Ha T = $T_1T_2…T_n$, akkor a $P_{n-1} = T_n(P)$; $P_{n-2} = T_{n-1}(P_{n-1})$; … ; $P_1 = T_2(P_2)$; és $Q = T_1(P_1)$, ahol P_i a program közbülső formája és a fordítóprogram n menetben végzi el a fordítást. 3, **interpreter**: forrásnyelv: magasszintű nyelv. Az interpreter egy olyan program, ami egy gépet valósít meg, ami a magasszintű nyelvet értelmezi (hardware-esen is megvalósítható), az eredmény tehát egy kétszintű gép. Nem készül tárgyprogram, a fordítási és futási idő egybeesik, eléggé lassú. Az első compilereket az 50-es években írták és aritmetikai műveleteket fordítottak gépi kódra. **A compilerek egyéb alkalmazásai: Szilikon-compilerek**: nyomtatottáramkör tervét generálja. **Lekérdező-rendszer**: adatbázis lekérdező utasítás. **Szövegátalakító rendszer**.

6. A Fordítóprogramok szerkezete:
A compiler a tárgyprogramon kívül készít egy listát is, amiben szerepelnek a hibák és egyéb visszajelzések. compiler(forrásprogram)(tárgyprogram, lista). **Ezt finomítjuk tovább:** i, input-**handler(forrásprogram)(karaktersorozat):** A fordítóprogram számára a karaktersorozatok a megfelelõ inputok. Az input-handler feladata ezen sorozatok elkészítése a soremeléseket is levágva. ii, **output-handler(forrásprogram, hibák)(lista):** Az input-handler soraiból készíti el a listát egy fájlban. iii, A tárgykódot fájlba menti. iv, **source-handler**: input-handler + output-handler. A source-handler és a kód-handler elvégzi a perifériákkal kapcsolatos feladatokat, így a compilernek csak a fordítás feladata marad: compiler(karaktersorozat)(tárgykód, hibák). **A compiler finomítása: analízis + szintézis:** ***Analízis*****:** i, **lexikális elemzõ(karaktersorozat)(szimbólumsorozat, lexikális hibák):** szimbólum: típuskód + cím. Kiszûri a kommenteket és a felesleges space-eket. ii, **szintaktikus elemzõ(szimbólumsorozat):** (szintaktikusan elemzett program, szintaktikus hibák). Elemzi a program struktúráját, kimenete speciális pl.: szintaxisfa. iii, **szemantikus elemzõ(szintaktikusan elemzett program):** (analizált program, szemantikus hibák). ***Szintézis*****:** i, **kódgenerátor(analizált program)(tárgykód):** A tárgykód operációs rendszer- és hardware-függõ, rendszerint gépi kód. ii, **kódoptimalizáló(tárgykód)(tárgykód).** A fordítás során elvégzett menetek száma széles skálán mozoghat, ezt a következők határozzák meg: i, rendelkezésre álló memória. ii, a compiler mérete és sebessége. iii, a tárgyprogram mérete és sebessége. iv, hibafelismerési és hibajavítási stratégiák. v, a megírásra szánt idõ

(7. tétel)
A lexikális elemző
Feladata, hogy a karaktersorozatban a szimbolikus egységeket felismerje, azaz meghatározza a szimbólum szövegét és típusát. A szimbolikus egységek precíz definíciója reguláris kifejezésekkel adható meg, ezért a lexikális elemző szimbólumaihoz konstruálhatunk egy-egy velük ekvivalens véges determinisztikus automatát. Ez könnyen implementálható a ’case’ utasítás segítségével. Egy szimbólumot a lehető leghosszabbnak kell tekinteni. Az automata végállapotaihoz szimbólum-feldolgozó funkciók is tartozhatnak. (Pl. egy azonosító szimbólumot a szimbólumtáblába kell írni.) A white-space-eket nem kell továbbadnia a szintaktikus elemzőnek. A lexikális elemzőhöz egy puffert rendelünk, amiben a szimbólum felismerése után a szimbólumot alkotó karakterek vannak (pl. stringek ebbe kerülnek). **Kulcsszavak, standard szavak** Előre definiált jelentésük van, a standard szavaknál ez a jelentés a programban megváltoztatható. Feladat: elkülöníteni a kulcsszavakat a felhasználó által definiált azonosítóktól. Két módszer: i) Minden kulcsszót egy reguláris kifejezéssel írunk le és megadjuk a hozzá tartozó automatát (túl nagy program). ii) A kulcsszavakat táblázatban tároljuk. A karaktersorozatban a szavakat egy általános azonosító-felismerővel határozzuk meg, majd megnézzük, hogy benne van-e a táblázatban. Ha igen, akkor kulcsszó, egyébként azonosító. A standard szavakat ugyanígy határozzuk meg, jelentésüket a szintaktikus elemző dönti el. **Előreolvasás** Mivel a leghosszabb szimbólumot szeretnénk felismerni, ezért egy vagy több karaktert előre kell olvasni. Ekkor a beolvasott karaktereket egy pufferbe tesszük, és minden karakterhez eltároljuk, hogy addig a szöveg értelmezhető-e, és ha igen, mi a típusa. Ha az automatával végállapotba jutunk, akkor a szimbólum egy hatványkitevő szám. Ha nem

vagyunk végállapotba, de nem tudnunk továbblépni, akkor a legutolsó érvényes bejegyzéshez tartozó szöveg lesz a felismert szimbólum. A modern pr.nyelveknél max. 2 karaktert olvasnak előre. **A szimbólumtábla** Egyszerű pr.nyelveknél, amelyekben csak globális változók és deklarációk vannak, a lexikális elemző azonnal felírhatja a megtalált konstans vagy azonosító szimbólumot a szimbólumtáblába, ha az még nincs ott. Blokkstruktúrált pr.nyelveknél a lexikális elemző nem tudja meghatározni az azonosítók típusát. **Direktívák** A direktívák a compilerek működésének vezérlésére szolgálnak. A lexikális elemzőnek fel kell ismernie a direktíva szimbólumait, az értékeket át kell adnia a source-handlernek, majd a direktíva szimbólumait törölni. Pl: a feltételes fordítás direktíváját ki kell értékelni, és csak akkor elemezni, ha a feltétel igaz. **Hibakezelés** Ha egy karaktersorozatnak nem tud szimbólumot megfeleltetni, akkor lexikális hibát talált. Hogy az elemzés ne szakadjon meg, hibaelfedést alkalmazunk. A leggyakoribb hibák: i) nem megengedett karakter (pl. addxress); ebben az esetben vagy nem foglalkozik a már beolvasott résszel (->ress), vagy kihagyja/helyettesíti a karaktert (->address, add+ress) ii) kulcsszóba illegális karakter kerül; ebben az esetben a hibát meg kell jelölni, a szintaktikushiba-javító ki tudja javítani iii) hiányzó karakter iv) különösen sok gondot kell fordítani a stringek és a kommentek befejező szimbólumainak hiányára.

(8. tétel)

A szintaktikus elemzés módszerei.

A gyakorlatban használt programnyelvek szintaxisát nem lehet megadni környezetfüggetlen grammatikákkal, ezért két részre bontjuk a feladatot. Az első részhez tartozik a programok struktúrájának, az utasítások felépítésének leírása, a másodikhoz pedig az olyan megkötések, mint például a típusazonosság egy értékadásnál. Az első részt le tudjuk írni környezetfüggetlen grammatikával (csak ilyenekkel foglalkozunk). Alapfogalmak: **Def mondatforma:** G=(T,N,S,P) grammatika, ha S->*α akkor α mondatforma. **Def mondat**: ha S->*x, akkor x mondat (és egyben mondatforma is). **Def részmondat:** α= α_1 βα_2, β az α egy részmondata, ha ∃A szimbólum: S->* α_1A α_2 és A->* β; β egyszerű részmondata α-nak, ha A-> β. **Def nyél:** Egy mondatforma legbaloldalibb egyszerű részmondatát a mondatforma nyelének nevezzük. PL: G=({i,+,*,(,)},{E,F,T},E,P) ahol P: E->T | E+T ; T->F|T*F ; F->i|(E) Ekkor az E+T*i+T*F mondatformának E+T*i vagy T*i részmondata, T*F egyszerű részmondata, nyele i. Egy mondatforma szintaxisfájának a gyökere a kezdőszimbólum, levelei terminális és nemterminális szimbólumok. Egy mondatforma részmondatának szintaxisfája a mondatforma szintaxisfájának részfája. Egyszerű részmondatának szintaxisfája pedig olyan részfa, amely csak gyökér és levél elemet tartalmaz. Egy szintaxisfában a legbaloldalibb ilyen részfa a mondatforma nyele. Továbbiakban feltesszük: **1**. a grammatika ciklusmentes, vagyis nem tartalmas A->+A helyettesítés sorozatot. **2.** a grammatika redukált, vagyis nincsen benne felesleges nem terminális szimbólum: ∀A-ra: S->* αAβ->αγβ->*xyz, ahol A->*y és |y|>0. **3.** a grammatika ε-mentes, vagyis nincs A->ε szabály; ha van S->ε akkor S nem szerepel egyik szabály jobboldalán sem. A szintaktikus elemzés általános tulajdonságai. A szintaktikus elemzés feladata, hogy eldöntse a program a nyelv egy mondata-e. Ehhez elő kell állítani a szintaxisfát, amelyből csak a gyökérelemet és a leveleket ismerjük. Ha S-ből indulunk ki: felülről-lefelé elemzés. Ha a levelekből indulunk: alulról-felfelé elemzés. Felülről-lefelé elemzés: **Def.** Ha A->α akkor az **xAβ mondatforma legbaloldalibb helyettesítése** xαβ. **Def:** Ha az S->*x levezetésben minden helyettesítés legbaloldalibb akkor a **levezetés is legbaloldalibb**. Pl: i+i*i legbal: E->E+T->T+T->F+T->i+T->i+T*F->i+F*F->i+i*F->i+i*i módszer a kezdőszimbólumból kiindulva és legbaloldalibb helyettesítésekkel próbálunk eljutni a szöveghez. **Tét:** Ha S->*xα->yz, ahol |x|=|y| akkor x=y. Ez alapján, ha a fa építésekor a baloldali terminálisok nem egyeznek meg az elemzendő szöveg bal oldalán álló terminálisokkal, akkor a fa biztosan nem lesz jó, vissza kell építeni. **Alulról-felfelé elemzés**. **Def:** Egy szabály balsarka a jobb oldalán álló első szimbólum balsarkos elemzés: a szövegben balról jobbra haladva meghatározzuk a balsarkokat, majd felülről-lefelé haladva a fa többi részét. A fa elemeit sorba rakjuk inorder bejárással.

9. A teljes vissalépéses elemzés:

Rendezzük a helyettesítési szabályokat sorba, így beszélhetünk egy A nemterminális szimbólum n. helyettesítési szabályáról. **1,** Az S szimbólum helyettesítésére az elsõ olyan szabályt alkalmazzuk, amelyiknek a baloldalán az S áll. **2,** Az így létrehozott mondatformában a legbaloldalibb nemterminálist a nemterminális elsõ helyettesítési szabályával helyettesítsük. A létrehozott új mondatformákra ezt a mûveletet ismételjük addig, amíg lehetséges. **3,** Két eset van, amikor a 2. pont mûvelete nem alkalmazható tovább: i, a mondatforma baloldalán álló terminálisok nem egyeznek meg az elemezendõ szöveg prefixével. Ekkor a levezetés sikertelen. ii, nincs több nemterminális a mondatformában. Ekkor ennek a mondatnak és az elemezendõ szövegnek az azonossága az elemzés végét jelenti, az elemzés sikeres. **4,** Az elsõ esetben, illetve a második esetben, ha a terminálisok nem egyeznek meg, lépjünk egy helyettesítést vissza, és

az A-ra alkalmazott helyettesítési szabály helyett alkalmazzuk az A következõ helyettesítési szabályát, ha létezik. Ezután folytassuk az elemzést a 2. pontban leírtakkal. **5,** a Ha az A nemterminális szimbólumnak nincs már több helyettesítési sza\-bálya, akkor lépjünk vissza az A helyettesítését megelõzõ lépésre, és folytassuk az elemzést a 4. pont második felében leírt mûvelettel, amikor is a következõ helyettesítési szabályt alkalmazzuk. **6,** Ha egy ilyen visszalépéskor az S szimbólumhoz jutunk vissza, és nincs már az S-nek további helyettesítési szabálya, akkor az elemzést befejeztük, az elemezendõ szöveg nem mondata a grammatika által definiált nyelvnek. **Definíció 1:** A **balrekúrzív szimbólum**, ha $A \rightarrow^{+} A\alpha$. **Definició 2:** A **közvetlen balrekúrzív szimbólum**, ha $A \rightarrow A\alpha$. **Definíció 3:** G **balrekúrzív grammatika**, ha tartalmaz legalább egy balrekúrzív szimbólumot. A közvetlen balrekurzió egy új nemterminális szimbólum bevezetésével kiküszöbölhetõ. A továbbiakban nem balrekúrzív grammatikákkal foglalkozunk. **Algoritmus:** $A \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_k$ helyett $A_1 \rightarrow \alpha_1|\ A_2 \rightarrow \alpha_2| \ \ldots \ | \ \alpha_k$. Az elemzés egy állapota: (S, i, α, β), ahol i, S: q = normál állapot, b = visszalépés, t =az elemzés vége. ii, i: az inputszövegre mutató pointer, a szöveg végén #. iii, α: a vizsgált mondatforma története (verem), $\alpha \in T$ unio N_i , N_i az indexekkel ellátott nemterminális szimbólumok halmaza. iv, β: a vizsgált mondatforma (verem), $\beta \in T$ unio N unio #. Kezdőállapot: (q, 1, ϵ, S#). Végállapot: (t, n+1, α, ϵ). **Átmenetek: 1, szintaxisfa építése az elsõ szabállyal**: Ha $A_1 \rightarrow \gamma_1$, akkor (q, i, α, Aβ) → (q, i, αA_1, $\gamma_1\beta$). **2,** input szimbólum olvasása: Ha c_i = a, akkor (q, i, α, aβ) → (q, i+1, αa, β). **3, input szimbólum nem azonos a vizsgált szimbólummal**: Ha $c_i \neq a$, akkor (q, i, α, aβ) → (b, i, α, aβ). **4,** visszalépés az input szövegben: (b, i, αa, β) → (b, i-1, α, aβ). **5,** a **következõ helyettesítési szabály keresése**: (b, i, αA_j, $\gamma_j\beta$) → ***i,*** ha i = 1, A = S és S-nek csak j helyettesítése van: leállás: (b, 1, αS_j, $\gamma_j\beta$) → (t, 1, αS_j, $\gamma_j\beta$). ***ii,*** ha van $A_{j+1} \rightarrow \gamma_{j+1}$ szabály: (b, i, αA_j, $\gamma_j\beta$) → (q, i, αA_{j+1}, $\gamma_{j+1}\beta$). ***iii,*** egyébként , azaz ha A minden szabályát már felhasználtuk (A ≠ S): (b, i, αA_j, $\gamma_j\beta$) → (b, i, α, Aβ). **6, sikeres befejezés**: (q, n+1, α, #) → (t, n+1, α, ϵ). Ekkor α tartalmazza a szintaxisfa leírását. $\alpha = X_1X_2 \ldots X_n$ és legyen h(X_i) = { ϵ, ha $X_i \in T$; $A_k \rightarrow \alpha_k$, ha $X_i \in N$, és $X_i = A_k$. A $h(X_1)$, $h(X_2)$, …, $h(X_m)$ helyettesítési szabályokból álló sorozat a szintaxisfa felépítését adja. Az algoritmus lassú és nem adja meg a hiba helyét, viszont minden balrekurzió-mentes, környezetfüggetlen grammatikára alkalmazható.

(10. tétel)

Korlátozott visszalépéses keresés.

Tfh: a teljes visszalépéses elemzéssel eljutunk egy S->*xAβ levezetéshez, majd alkalmazzuk az $A->\alpha_\varepsilon$ szabályt és S->xAβ->$x\alpha_2\beta$->xyβ. Ha xy prefixe az elemzendő szövegnek, de később visszalépésre kényszerülünk és eljutunk az A->α_k szabályig, ami helyett A->α_{k+1}-et kellene alkalmaznunk, akkor ezt már nem tesszük meg. Tehát ha egy prefixet már megtaláltunk, akkor abból nem engedünk. A korlátozott visszalépéssel kevesebb nyelvet tudunk elemezni. **Egy elemző (Knuth): egy elemző számítógép:** regiszter (true, false); verem; i pointer-regiszter; leáll, ha OK-t vagy ERROR-t kap. **Utasítássor:** címke, mnemonik, kétcímű operandus. (a) címkezóna: nemterminális szimbólum vagy üres. (b) mnemonik zóna: a szabályok jobb oldalán szereplő szimbólumok egyesével. (c) operandusok: AT, AF. A mnemonik zóna írt utasítások alapján: (1) az utasításkód az 'a' terminális szimbólum: elágazás: if c_i=a then i=i+1; execute AT; else execute AF; (2) az utasításkód az 'A' nemterminális szimbólum: (i) PUSH i; (ii) call A; (iii) if reg=FALSE then POP i; execute AF; else execute AT; execute AF/AT az operandusok tartalmától függ: (1) ha üres, akkor a következő sort hajtjuk végre (2) ha TRUE/FALSE, akkor visszatérünk a hívó procedurába és a regiszterbe true/false-t teszünk. (3) ha egy A nemterminális szimbólum van benne: ugrunk az A: címkéjű utasításra (4) OK/ERROR: végre az elemzésnek. **Az elemző felépítése**: **címke:** szabályok baloldala (csak egyszer) **mnenonik:** szabályok jobboldala (egyenként). AT: szabály utolsó betűjeléhez TRUE (S-nél OK), AF: utolsó szabály összes betűjéhez FALSE (S-nél ERROR). A helyettesítési szabályokat érdemes hosszuk szerinti csökkenő sorrendben felírni.

(11. tétel)

Egyszerű és ε-mentes LL(1) grammatika és elemzés.

LL(k) grammatika: k szimbólum előreolvasásával a következő lépés egyértelműen meghatározható. **Def:** A G grammatika **egyszerű LL(1) grammatika**, ha ε-mentes és minden A nemterminálisok szimbólumra a szabályok a különböző terminális szimbólummal kezdődnek. **Módszer:** Az elemzés állapota: (x,β,y), ahol x a még nem elemzett szöveg; β egy verem amiben az aktuális mondatforma van (alján #); y egy lista, amiben az alkalmazott szabályok sorszáma szerepel, ez alapján fogjuk felépíteni a szintaxisfát. **Elemző táblázat:** sorok: T∪N∪#; verem tetején szimbólum, oszlopok: T∪#; előre olvasott szimbólum. Elemei: M(x,a)={ pop, ha x=a; accept, ha x=a=#; (aα,i), ha x→aα az i. szabály; ERROR egyébként. Kezdőállapot: (x#, S#, C). Végállapot: (#,#,y). Egyszerűsítés: szabályalkalmazás és POP összevonása. **Def: FIRST(α)**={a|α→*aβ}, azaz azok a terminális szimbólumok, amik az α-ból levezethető részmondatok bal oldalán állnak. **Def:** G egy **ε-mentes LL(1) grammatika**, ha minden A

nemterminális szimbólumra és k>1-re $A\rightarrow\alpha_1|\alpha_2|\ldots|\alpha_k$ esetén $FIRST(\alpha_i)\cap FIRST(\alpha_j)=\varnothing$, ha i≠j. **Tét:** Ha G egyszerű LL(1) grammatika, akkor G egy ε-mentes LL(1) grammatika. **Tét:** Ha G egy ε-mentes LL(1) grammatika, akkor megoldható vele ekvivalens G' egyszerű LL(1) grammatika. Biz: Greibach-normál formulára hozzuk. Ha ebben van $A\rightarrow a\alpha|a\beta$ akkor átalakítjuk: $A\rightarrow aC$, $C\rightarrow\alpha|\beta$ majd erre újra Greibach-normál forma és így tovább.Δ Ugyanaz a módszer, mint az egyszerű LL(1)-nél, de most azt a szabályt alkalmazzuk, amelynek a FIRST halmazában van az előreolvasott karakter.

(12. tétel)

LL(k) és erős LL(k) grammatika és elemzés.

Def: $\mathbf{FRIST_k(\alpha)}=\{x|\alpha\rightarrow^*x,|x|<k\ k\geq0$ vagy $\alpha\rightarrow^*x\beta,|x|=k\}$. Ha $\alpha\rightarrow^*\varepsilon$ akkor $\varepsilon\in FIRST_k(\alpha)$. **Def:** G egy LL(k) grammatika, ha (i) $S\rightarrow^*NA\beta\rightarrow N\alpha_1\beta\rightarrow^*wx$ (ii) $S\rightarrow^*NA\beta\rightarrow N\alpha_2\beta\rightarrow^*wy$ és $FIRST_k(x)=FIRST_k(y)$ esetén $\alpha_1=\alpha_2$. Eszerint, ha a már elemzett N utáni k terminális szimbólumot ismerjük, akkor az A-ra alkalmazandó szabály egyértelmű. Ha egy grammatika $LL(k_0)$ grammatika, akkor LL(k) grammatika is $\forall k>k_0$-ra. Az LL(k) grammatikák nem feltétlenül ε-mentesek, de: **Tét:** ha G egy nem ε-mentes LL(k) grammatika, akkor létezik olyan ε-mentes LL(k+1) grammatika, amelyik az L(G) nyelvet generálja. **Tét:** Ha G egy ε-mentes LL(k+1) grammatika (k>0), akkor van olyan LL(k) nem ε-mentes grammatika, amelyik az L(G) nyelvet generálja. Nem minden környezetfüggetlen grammatika LL(k) grammatika! **Tét:** A G grammatika akkor és csak akkor LL(k) grammatika, ha $\forall S\rightarrow^*wA\beta$ és $A\rightarrow\gamma|\delta$ ($\gamma\neq\delta$) esetén $FIRST_k(\gamma\beta)\cap FIRST_k(\delta\beta)=\varnothing$. Biz: Indirekt, definíció alapján. Ez a tétel ε-mentes grammatikákra azt mondja ki, hogy egy G grammatika akkor és csak akkor LL(k), ha $A\rightarrow\alpha_1|\ldots|\alpha_n$ esetén $FIRST_1(\alpha_1)$, $FIRST_1(\alpha_2)$,…, $FIRST_1(\alpha_n)$ páronként diszjunktak. Ez nem ε-mentes grammatikákra nem igaz. **Def:** $\mathbf{FOLLOW_k(\beta)=}\{x|S\rightarrow^*\alpha\beta\gamma$ és $x\in FIRST_k(\gamma)\}$. **Tét:** A G grammatika akkor és csak akkor LL(1) grammatika, ha $\forall A$ nemterminális szimbólumra $A\rightarrow\gamma|\delta$ esetén $FIRST_1(\gamma FOLLOW_1(A))\cap FIRST_2(\delta FOLLOW_1(A))=\varnothing$. Biz: Indirekt, definíció alapján. Eszerint a tétel szerint egy G grammatika akkor és csak akkor LL(1) grammatika, ha $\forall$A-ra $A\rightarrow\alpha_1|\ldots|\alpha_n$ esetén $FIRST_1(\alpha_1)$, $FIRST_1(\alpha_2)$,…, $FIRST_1(\alpha_n)$ páronként diszjunktak és ha $\alpha_i\rightarrow^*\varepsilon$, akkor $FIRST_1(\alpha_j)\cap FOLLOW_1(A)=\varnothing$ ($1\leq j\leq n$, $j\neq i$). A tétel nem általánosítható tetszőleges k-ra, ez csak egyszerűbb nyelvosztályokra igaz. **Def:** A G grammatika **erős LL(k) grammatika**, ha (i) $S\rightarrow wA\beta\rightarrow w\alpha_1\beta\rightarrow wx$ és $S\rightarrow^*vA\mu\rightarrow v\alpha_2\mu\rightarrow^*vy$ és $FIRST_k(x)=FIRST_k(y)$ esetén $\alpha_1=\alpha_2$. **Tét:** A G grammatika akkor és csak akkor erős LL(k) grammatika, ha $\forall A\rightarrow\gamma|\delta$ ($\gamma\neq\delta$) esetén $FIRST_k(\gamma FOLLOW_k(A))\cap FIRST_k(\delta FOLLOW_k(A))=\varnothing$. **Tét:** A G grammatika akkor és csak akkor erős LL(1) grammatika, ha LL(1) grammatika k>1-re van olyan grammatika, amelyik LL(k), de nem erős LL(k). Pl: LL(2): $S\rightarrow aAaa|bAba$ és $A\rightarrow b|\varepsilon$.

13. LL(1)-es elemzés táblázattal:

Nem alakítható át minden grammatika LL(1)-es grammatikára. Nem ismert olyan algoritmus, amely tetszõleges nyelvhez LL(1)-es grammatikát ad. Bizonyos esetekben egy grammatikából egyszerû átalakításokkal LL(1)-es grammatikát kaphatunk (belrekurzió mentesítés + faktorizáció). **Tétel:** Ha G egy LL(k) típusú grammatika, akkor nem lehet balrekurzív grammatika. Bizonyítás: $A_0\rightarrow A_1\alpha\rightarrow\ldots\rightarrow A_n\alpha_n\rightarrow A_0\alpha$ és $A_j\rightarrow\alpha_{j1}|\alpha_{j2}$. Ekkor $A_j\rightarrow\alpha_{j1}\rightarrow^*A_j\beta\rightarrow\alpha_{j2}\beta\rightarrow x$ => $FIRST_k(\alpha_{j1})\cap FIRST_k(\alpha_{j2})\neq 0$. Minden környezetfüggetlen grammatikát lehet balrekurziómentesíteni. **QED. Definíció 1: A → γ helyettesítési szabály**: $LEFT_1(A,\gamma)=\{FIRST_1(x)\mid S\rightarrow^*\omega A\beta\rightarrow\omega\gamma\beta\rightarrow^*\omega x\}$. **Definíció 2:** Az A **nemterminális szimbólum belfaktorizált**, ha minden $A\rightarrow\gamma|\delta$ esetén: $LEFT_1(A,\gamma)\cap LEFT_1(A,\delta)=0$. **Definíció 3:** Egy grammatika **balfaktorizált**, ha minden nem terminális szimbóluma balfaktorizált. **Tétel 2:** Egy grammatika balfaktoriztált <=> a grammatika LL(1)-es Bizonyítás: Ez az **LL(k) definíciójának átfogalmazása**. Közvetlen balfaktorizáció: $A\rightarrow Bc|Bd$ helyett $A\rightarrow BB'$, $B'\rightarrow c|d$. Közvetett balfaktorizáció: több szabály összevonásával érjük el. **Elemzés táblázattal:** M(X,a)={ pop, ha X=a; accept , ha X = # = a; (α,i), ha X→α az i. szabály ($a\in FIRST_1(\alpha)$) vagy ($\epsilon\in FIRST_1(\alpha)$ és $\alpha\in FOLLOW_1(X)$); error, egyébként. A táblázat kitöltéséhez meg kell határozni a $FIRST_1(\alpha)$ és a $FOLLOW_1(A)$ halmazokat. **1.módszer: $FIRST_1(X)$ meghatározása (X∈{N unio T})**: 1, $FIRST_1(X)=0$. 2, Ha X∈T, akkor $FIRST_1(X)=FIRST_1(X)$ unio {X}. 3, Ha X → ϵ, akkor $FIRST_1(X)=FIRST_1(X)$ unio{ϵ}. 4, Ha $X\rightarrow Y_1\ldots Y_m$, $Y_1\ldots Y_k\rightarrow^*\epsilon$ ($1\leq k\leq m$) és a∈$FIRST_1(Y_{k+1})$, akkor $FIRST_1(X)=FIRST_1(X)$unio {a}. Ha k=m, akkor $FIRST_1(X)=FIRST_1(X)$ unio { ϵ }. **$FIRST_1(\alpha)$ meghatározása ($\alpha=X_1\ldots X_n$)**: 1, $FIRST_1(\alpha)=FIRST_1(X_1)\backslash\{\epsilon\}$. 2, Ha ϵ∈$FIRST_1(X_1)$, akkor $FIRST_1(\alpha)=FIRST_1(\alpha)$ unio $FIRST_1(X_2)\backslash\{\epsilon\}$. 3, Ha $B\rightarrow\alpha A\beta(|\beta|>0)$, $\beta\rightarrow^*\epsilon$, akkor $FOLLOW_1(A)=FOLLOW_1(A)$ unio $FOLLOW_1(B)$. 4, Ha $B\rightarrow\alpha A$, akkor $FOLLOW_1(A)=FOLLOW_1(A)$ unio $FOLLOW_1(B)$. **2. módszer:** Relációkat határoz meg a grammatika szimbólumai között, ezeket a relációkat táblázatokba(mátrixokba) helyezi és a

mátrixműveletek alkalmazásával jut el a $FIRST_1(A)$ és a $FOLLOW_1(A)$ halmazokhoz. $N_\epsilon = \{A \mid A \rightarrow * \epsilon\}$. **F:** Ha $A \rightarrow X_1X_2\ldots X_n$, akkor AFX_1. Ha $X_1 \in N_\epsilon$, akkor AFX_2 Ha $X_1, X_2 \in N_\epsilon$, akkor AFX_3, és 0gy tovább. F^+ (**Warshall-mátrix**): $FIRST_1(A) = \{a \mid AF^+a\}$ unio $\{ \epsilon \mid A \in N_\epsilon \}$. **B:** Ha $A \rightarrow X_1X_2\ldots X_n$, akkor X_iBX_{i+1} $(1 \leq i \leq n-1)$. Ha $X_{i+1} \in N_\epsilon$, akkor X_iBX_{i+2} $(1 \leq i \leq n-2)$. Ha $X_{i+1}, X_{i+2} \in N_\epsilon$, akkor X_iBX_{i+3} $(1 \leq i \leq n-3)$, és így tovább. **L:**Ha $A \rightarrow X_1X_2\ldots X_n$, akkor X_nLA. Ha $X_n \in N_\epsilon$, akkor $X_{n-1}LA$. Ha $X_{n-1}X_n \in N_\epsilon$, akkor $X_{n-2}LA$, és így tovább. Ekkor $FOLLOW_1(A) = \{ a \mid A(L^*BF^*)a \}$. Kell egy $S' \rightarrow S\#$ szabály is.

(14. tétel)

A rekurzív leszállás módszere.

A grammatika nemterminális szimbólumaihoz procedurákat rendelünk és a rekurzív procedurahívásokon keresztül a programnyelv valósítja meg a blabla. **Módszer:** szimbólumok vizsgálata: accept:

```
Procedure accept (szimbólum);#
 Begin if akt-szimbólum=szimbólum#
        then következő-szimbólum#
        else (error…)#
end;
```

∀ nemterminális szimbólumhoz:

```
Procedure A;#
 Begin T(A) end;#
```

Ahol T(A):# (1) $A \rightarrow a$ accept(a) (2) $A \rightarrow B$ esetén B procedure hívás (3) $A \rightarrow x_1\ldots x_n$ esetén begin $T(x_1);\ldots;T(x_n)$;end; (4) Ha az $A \rightarrow \alpha_1|\ldots|\alpha_n$ szabály ε-mentes, akkor T(A) legyen: case akt-szimbólum of#

```
 First(α1): T(α1);#
 …#
 First(αn): T(αn);#
```

(5) Ha ez ε-t is megengedjük, akkor kiegészül egy follow(A):-val. Így az LL(1) grammatika szabályaiból közvetlenül megírhatjuk az elemző programot. Példa: program→program azonosító; blokk. # blokk→változódeklaráció, proc. deklaráció, utasítások # utasítások→begin utasítás {;utasítás} end # utasítás→utasítások | értékadó-utasítások | read-ut | write-utasítás | if-utasítás | while-utasítás # értékadó-utasítás→azonosító:=kifejezés# read-ut→read…# write-ut→write…# if-ut→if…# while-ut→while…

(15. tétel)

Operátor precedencia grammatika és elemzés

Alulról-felfelé elemzések. A terminális szimbólumokból kiindulva halad a kezdőszimbólum felé. Az elemzések léptetés-redukálás típusúak: először redukciót akar végrehajtani, az azt vizsgálja, hogy a verem tetején és közvetlenül alatta levő szimbólumokból alkotott α mondatforma megfelel-e valamely szabály jobb oldalának; ha a redukció nem lehetséges, akkor léptet, vagyis a verem következő elemét is hozzácsapja α-hoz és próbál redukálni. **A visszalépéses elemzés.** Az elemzésredukciókkal és léptetésekkel eljut a vizsgált szimbólumsorozat végére. Ha a szintaxisfa léptetése is eljut az S-ig, akkor jó, 1bként vissza kell lépni. Visszalépésekkel az utoljára végrehajtott redukciókig kell visszamenni és meg kell próbálni egy másik szabály szerint redukálni. Ha nincs ilyen, akkor léptetni kell. A grammatikáról fel kell tenni, hogy ciklus($A \rightarrow^+ A$) és ε-mentes ($A \rightarrow \varepsilon$). **Operátor-precedencia** relációkat csak terminálisok között értelmezünk ($S_1<S_2$, $S_1>S_2$, $S_1=S_2$). Segítségükkel meghatározható a mondatforma nyele. Nem engedjük meg az olyan szabályokat, amelyeknek jobboldalán két nemterminális áll egymás mellett. **Def:** Egy grammatika **operátor-grammatika**, ha nem tartalmaz $A \rightarrow \alpha BC\beta$ és $A \rightarrow \varepsilon$ szabályt. Az általa generált nyelv egy **operátor-nyelv**. **Def: Operátor-precedencia relációk meghatározása**: 1. $a = b \Leftrightarrow \exists\ A \rightarrow \alpha ab\beta$ vagy $A \rightarrow \alpha aBb\beta$ szabály, 2. $a < b \Leftrightarrow \exists\ A \rightarrow \alpha aB\beta$ szabály: $B \rightarrow^+ b\gamma$ vagy $B \rightarrow^+ Cb\gamma$, 3. $a > b \Leftrightarrow \exists\ A \rightarrow \alpha Bb\beta$ szabály: $B \rightarrow^+ \gamma a$ vagy $B \rightarrow^+ \gamma aC$. Az = könnyen kiolvasható a szabályokból, a többihez: **Def:** B nemterminális levezetéseibő származó lehetséges **baloldali terminálisok**: **LEFTOP(B)** = $\{ b \mid B \rightarrow^+ b\gamma$ vagy $B \rightarrow^+ Cb\gamma \}$. Ha $A \rightarrow \alpha aB\beta$, akkor $\forall\ b \in$ LEFTOP(B): $a < b$. Hasonlóan: **RIGHTOP(B)** = $\{ b \mid B \rightarrow^+ \gamma a$ vagy $B \rightarrow^+ \gamma aC \}$. Lehetséges, hogy két terminális között több reláció is fennáll. **Def:** Ha két terminális között nincs reláció, vagy egy reláció van, akkor a grammatika **operátorprecedencia grammatika**. **Elemzők.** Cél a mondatforma nyelvének meghatározása, amiben most a nemterminálisok semmilyen szerepet nem játszanak. **Def:** Egy mondatforma **prím-részmondata** azaz **egyszerű részmondata**, amely legalább egy terminálist tartalmaz. Így most a legbaloldalibb prím-részmondatot keressük. A terminálisok közé beírjuk a relációkat és a legbaloldalibb < és > közé zárt rész a keresett „nyél”. **Tétel:** Legye $[A_\varepsilon]$ $(1 \leq \varepsilon \leq n+1)$ egy nemterminális, vagy ε és legyen $a_o = a_{n+1} = \#$. Ha $[A_1]\ a_1\ [A_2]\ a_2\ \ldots\ [A_n]\ a_n\ [A_{n+1}]$ egy operátor-

precedencia egy mondatformája, akkor a **legbaloldalibb prím részmondata** az a $[A_i]\ a_i \ldots [A_j]\ a_j\ [A_{j+1}]$ sorozat, amelyre $a_{i-1} < a_i = \ldots = a_j > a_{j+1}$. A # jelek azért kellenek, hogy az utolsó lépésben is legyen nyelünk. Semmilyen információnk nincs a nemterminálisokról, ezért két lépésre bontjuk a megoldást: 1. alulról-lefelé haladva, „dummy" nemterminális szimbólumokat bevezetve megalkotjuk a szintaxisfa vázát, 2. felülről-lefelé haladva meghatározzuk a konkrét helyettesítési szabályokat. **Az operátorprecedencia függvény:** A precedencia táblázat tárolásához $O(u^2)$ hely kell. Legyen f és g két fv., amelyekre: 1. $a = b \Rightarrow f(a) = g(b)$ 2. $a < b \Rightarrow f(a) < g(b)$ 3. $a > b \Rightarrow f(a) > g(b)$. A fv. értékek tárolásához csak O(u) hely kell. Algoritmus: 1. Ha $a \in T \cup \{\#\}$, akkor a-hoz rendeljünk f(a)-t és g(a)-t. 2. Csoportosítsunk: ha a = b, akkor f(a) és g(b) legyen egy csoport. 3. A csoportok legyenek egy gráf pontjai, ha a < b, akkor mutasson él g(b) csoportjából f(a) csoportjába, ha a > b, akkor fordítva. 4. Ha a gráfban van kör, akkor f és g nem létezik! Ha körmentes, akkor f(a) ill. g(a) legyen a csoportjukból kiinduló leghosszabb úthossza. **Hibakezelés:** • Ha a vizsgált két terminális között nincs reláció, akkor a precedencia táblázatban üres hely van. Írjunk ide egy hibarutin hívását jelző karaktert és paraméterezzük a hiba típusától függően. • Mivel csak a terminálisokat vizsgáljuk ezért egy $A \rightarrow aBc$ szabály alapján az ac szöveget is redukálni fogja, így például elfogadja a # ( ) # szöveget. Ez kiszűrhető, ha megadjuk az egymás melletti lehetséges operátorokat és az elemzést avval kezdjük, hogy ezt leellenőrizzük. • Ha nincs olyan szabály amelyet alkalmazni tudnánk (a második menetben), akkor hibajavítási algoritmussal próbáljuk meg átalakítani a mondatformát, hogy egy szabály alkalmazható legyen.

(16. tétel)

Egyszerű precedencia grammatika és elemzés

A precedencia reláció fogalmát kiterjesztjük a nemterminális szimbólumokra is. Egy menetben futnak az elemzők. A mondatformák nyelét keressük. **Def:** X és Y a **grammatika tetszőleges szimbóluma**: 1. $X = Y \Leftrightarrow \exists\ A \rightarrow \alpha XY\beta$ szabály, 2. $X < Y \Leftrightarrow \exists\ A \rightarrow \alpha XB\beta$ szabály és $B \rightarrow^+ Y\gamma$, 3. $X > a \Leftrightarrow \exists\ A \rightarrow \alpha BY\beta$ szabály és $B \rightarrow^+ \gamma X$, $Y \rightarrow^* a\delta$. Mivel a legbaloldalibb levezetés inverzét alkalmazzuk és a > reláció a nyél jobboldalát határozza meg, a > reláció jobb oldalain csak terminális állhat. Az = relációban álló szimbólumokat könnyű meghatározni. $LEFT(A) = \{ X \mid A \rightarrow^+ X\alpha \}$, $RIGHT(A) = \{ X \mid A \rightarrow^+ \alpha X \}$, ($\neq FIRST_1(A)$). Vezessük be az L és R relációkat: ha $A \rightarrow X\alpha$, akkor ALX; ha $A \rightarrow \alpha X$, akkor ARX; Warshall algoritmussal: L^+, L*, R^+. Így $Left(A) = \{ X \mid AL^+X \}$ és $RIGHT(A) = \{ X \mid AR^+X \}$. Ezek felhasználásával: • $X < Y \Leftrightarrow \exists\ A \rightarrow \alpha XB\beta$ és BL^+Y, • $X > a \Leftrightarrow \exists\ A \rightarrow \alpha BC\beta$ és BR^+X és CL^*a. **Tétel:** $X = Y \Leftrightarrow XY$ szöveg része egy mondatforma nyelének. Biz: Nyél definícióját felhasználva. **Tétel:** $X > a \Leftrightarrow \exists\ \alpha\beta Xa\gamma$ mondatforma, amelynek βX a nyele. **Tétel:** $X < Y \Leftrightarrow \exists\ \alpha XY\beta\gamma$ mondatforma, amelynek $Y\beta$ a nyele. Ezek alapján az = relációval a mondatnyél belseje, a < és > relációkkal pedig a kezdő és végpontja kapható meg. **Def:** Egy ε-mentes G grammatikát **egyszerű precedencia-grammatikának** nevezünk, ha bármely két szimbóluma között legfeljebb egy precedencia-reláció van és a szabályok között nincs két azonos jobboldalú. **Elemzők:** A mondat elejére és végére # : $\# < X$ és $X > \#\ \forall X$-re. **Tétel:** $X_0 = X_{n+1} = \#$; az $X_1X_2 \ldots X_n$ **mondatforma nyele** az a legbaloldalibb $X_i \ldots X_j$ sorozat, amelyre $X_{i-1} < X_i = \ldots = X_j > X_{j+1}$. **Precedencia-függvény:** f és g fv-ek: $X = Y \Rightarrow f(x) = g(Y)$, $X < Y \Rightarrow f(X) < g(Y)$, $X > Y \Rightarrow f(X) > g(Y)$. Irányított gráf konstruálása. Pontjai: $f(X_i)$, $g(X_j)$, ha $X_i < X_j$ vagy $X_i = X_j$, akkor mutasson él $g(X_j)$-ből $f(X_i)$-be és ha $X_i > X_j$ vagy $X_i = X_j$, akkor mutasson él $f(X_i)$-ből $g(X_j)$-be. **Tétel:** Ha egy egyszerű precedencia-grammatikához konstruált irányított gráf olyan kört tartalmaz, amelynek van nem = reláció miatt összekötött két pontja, akkor az egyszerű precedencia-függvények nem léteznek. Biz: Indirekt. **Tétel:** $f(X_i)$ és $g(X_j)$ értékek a gráfban az adott csúcsból elérhető csúcsok száma. A fv. értékek meghatározása: n szimbólum esetén. 2m x 2n-es mátrix: $f(X_1)$, …, $f(X_n)$, $g(X_1)$, …, $g(X_n)$. (!*2x2-es mátrixok köv. soronként*!) $\mathbb{B}$:= [0 $\mathbb{G}$] [$\mathbb{L}^T$ 0], ahol $\mathbb{G}[i,j] = 1$, ha $X_i > X_j$, $\mathbb{L}[i,j] = 1$, ha $X_i < X_j$. $\mathbb{C}$:= [0 $\mathbb{E}$] [$\mathbb{E}^T$ 0], ahol $\mathbb{E}[i,j] = 1$, ha $X_i = X_j$. Warshall algoritmussal: $(C*B)^+$, ha ennek a főátlójában van 1-es, akkor a pr. fv-ek nem léteznek: $\mathbb{M}$:= [$\mathbb{I}$ $\mathbb{N}$] [$\mathbb{K}^T$ $\mathbb{I}$], ahol $\mathbb{N}[i,j] = 1$, ha $X_i > X_j$ vagy $X_i = X_j$, $\mathbb{K}[i,j] = 1$, ha $X_i < X_j$ vagy $X_i = X_j$. Warshall-algoritmussal: $\mathbb{M}^+$ és ekkor $\mathbb{M}^+[i,j] = 1$, ha a gráf i. pontjából út vezet a j. pontjába. **Hibakezelés:** Két típusú hiba: szimbólumpár-hiba, redukció-hiba. Két ellenőrzés: • **érvényes prefix ellenőrzés:** a verem tetején lévő szimbólumok a legfelső < relációtól kezdve, vajon egy szabály prefixét alkotják-e, • **baloldali vermelhetőség ellenőrzése:** redukciónál kell végrehajtani, mert ekkor a nyél szimbólumai helyett a szabály bal oldalán álló nemterminális kerül a verem tetejére és ellenőrizni kell, hogy van-e reláció a nyél alatti szimbólum és a bekerült új szimbólum között.

(17. tétel)

LR(0) elemek, az LR(0)-elemző

LR(0) grammatikák és elemzések: Az LR(k) elemzés visszalépés nélküli, léptetés-redukálás típusú elemzés. Első lépésben egy elemző-táblázatot állítunk elő, majd az ezt felhasználó elemző programot készítjük el. **Def:** A G = (T,

N, S, P) grammatikához tartozó G' **kiegészített gram**.: G' = (T, N ∪ {S'}, S', P ∪ {S' → S}). A szabályokat sorszámozzuk, az S' → S a 0. **Def:** Egy G' kiegészitett gram. **LR(k) gram**. (k≥0), ha S' →* αAw → αβw, S' →* γBx → γδx = αβy és $FIRST_k(w) = FIRST_k(y)$ esetén α = γ, A = B és x = y. Van olyan környezetfüggetlen gram., amely nem LR(k) gram. Egyetlen k-ra sem. **Tétel:** ∀ LL(k) gram. LR(k) gram., de ∃ LR(k) gram. olyan, hogy nem LL(k') gram. egyetlen k'-re sem. **Tétel:** ∀ LR(k) (k>1) gram.-hoz ∃ vele ekvivalens LR(1) gram. Ez nagyon fontos, mert csak az LR(1) gram.-ákkal kell foglalkozni. **Az LR(0) elemzés:** **Def:** Legyen az αβx mondatforma nyele β. Ekkor az αβ prefixeit az αβx **járható prefixeinek** nevezzük. Egy gram. járható prefixeinek halmaza nem feltétlenül véges. A járható prefixekből képzett halmazokhoz hozzárendelhetőek egy véges determinisztikus automata állapotai. Az automata egy állapota az állapothoz tartozó járható prefixeket ismeri fel. Az automata állapotai két diszjunkt csoportra oszthatók, a nem végállapothoz a léptetés, a végállapotokhoz a redukció műveletét rendeljük hozzá. A léptetés művelete állapotátmenetet eredményez és egy terminális vagy nem terminális szimbólum olvasását jelenti. A redukció állapot azt mutatja, hogy egy járható prefixben eljutottunk a nyél utolsó szimbólumáig, a nyelet redukálni kell. **Az automata leírható két táblázattal:** a sorokhoz az állapotok sorszámát rendeljük, az első táblázatba az kerül, hogy léptetni kell(S), vagy redukálni (ekkor az alkalmazott szabály sorszáma kerül be), a 0. szabály szerinti redukció legyen 'accept'. A másik táblázat azt tartalmazza, hogy léptetés esetén milyen szimbólumhoz, milyen állapotot rendelünk. Ha megadtuk az automatát, akkor az LR(0) elemző: van egy vermünk amiben szimbólumpárok vannak (szimbólum, automata állapot). Ha léptettünk, akkor a beolvasott új szimbólum és az új állapot kerül a verembe. Ha redukálunk A → α szerint, akkor |α| darab sort törlünk és egy új sort írunk bele, mintha 'A'-t olvastunk volna. 'accept' = ok, ha nincs megadva állapot az olvasott szimbólumhoz, akkor hiba. **LR(0) elemek:** Az elemzés alapvető feladata a nyelek meghatározása. Az automata először a nyél előtt levő szimbólumokat vizsgálja, majd a nyél prefixein keresztül jut el a nyél jobboldali végpontjához. Módszert adunk az automata állapotainak meghatározásához. **Def:** „ **.** " egy metaszimbólum. Ha A → αβ egy szabály, akkor a gram. **LR(0)-eleme** legyen [A → α **.** β] (A → ε : [A → **.**]). Értelmezése. A pont előtti részt már elemeztük. Pl.: E → E + T : [E → **.** E + T], [E → E **.** + T], [E → E + **.** T], [E → E + T **.**]. **Def:** egy G **gram. [A → α . β] LR(0)-eleme érvényes a γα járható prefixre nézve**, ha S' →* γAx → γαβx. Egy grammatika elemei LR(0) elemei véges sokan vannak. Csoportosítsuk őket: LR(0)-kanonikus halmazok. **Def:** Legyen J halmaz egy gram. egy **LR(0)-elem halmaza**. Ekkor **closure(J)**: 1. a J halmaz elemei legyenek closure(J)-nek is elemei, 2. ha [A → α **.** Bβ] ∈ closure(J) és β → γ egy szabály, akkor [β → **.** γ] ∈ closure(J), 3. closure(J)-t addig kell így bővíteni, amíg lehet. **Köv:** Az összes várható inputot leíró LR(0)-elemet tartalmazza. **Def: read(J,X)**: 1. ha [A → α **.** Xβ] ∈ J, akkor closure([A → αX **.** β]) ⊂ read(J,X), 2. read(J,X)-et addig kell így bővíteni, amíg lehet. Ha J a γ járható prefixekre nézve érvényes LR(0) elemeket tartalmazza, akkor a read(J,X) a γX-re nézve érvényes LR(0)-elemek halmaza. A **kanonikus halmazok meghatározása**: • kezdjük az S' → S szabállyal, J_0 := closure([S' → S]); • képezzük read (J_0, X)-et valamilyen X ∈ T ∪ N-re; ha a kapott halmaz nem üres és még nincs ilyen halmaz, akkor legyen ez a következő kanonikus halmaz; • ismételjük meg ezt az összes X ∈ T ∪ N-re; • az egész műveletet ismételjük meg az összes halmazra, amíg új halmazt kapunk. Mivel az LR(0)-elemek száma véges, az algoritmus is véges. **Az automata megkonstuálása:** ℂ = {J_0, J_1, …, J_m}, az állapotok halmaza: {0, 1, …, m}. • Ha egy halmazban olyan LR(0)-elemek vannak, amelyeknek a jobboldali végén van a pont: redukálás; • ha J_j = read(J_i,X), akkor az i állapot léptetése: X szimbólum hatására a j állapotba kerül. **LR(0)-elemzés nagy tétele:** Egy γ járható prefix érvényes elemeinek halmaza az a kanonikus elemhalmaz, amelyik az automatának ahhoz az állapotához tartozik, amelyikbe az automata a kezdőállapotból a γ hatására került.

(18. tétel)

Az SLR(1) elemző táblázata és az elemző működése:

Ha egy állapotban nem egyértelmű, hogy léptetni kell-e, vagy redukálni, akkor inadekvált állapotban vagyunk. Ezt megszüntethetjük, ha egy szimbólumot előre olvasunk. Az automatát itt is táblázattal írjuk le, de az actions táblát bővíteni kell: az oszlopokhoz a terminálisokat rendeljük, így tudjuk, hogy melyik input szimbólum esetén kell léptetni, ill. redukálni. A két táblázatot összevonjuk, a léptetéseket (s állapotsorszám), a redukciókat (r szabály) alakban adjuk meg. Az action táblázat értékeit kanonikus halmazokból számítjuk ki: 1. ha [A → α **.** aβ] ∈ J_i és read(J_i, a) = J_j, akkor action[i,a] = s_j, 2. ha [A → α **.**] ∈ J_i, A ≠ S' és a ∈ $FOLLOW_1$(A), akkor action[i,a] = rl, ahol A → α a l. szabály, 3. ha [S' → S_2] ∈ J_i, akkor action[i,#] = accept, 4. a goto táblázat(csak nem terminálisok vannak) ha read(J_i, a) = J_j, akkor goto[i,A] = j, 5. az üres helyekre error kerül, 6. a kezdeti állapot legyen az, amelyikben az, amelyikben az [S' → **.** S] van. **Def:** Ha ezzel az eljárással a táblázat ∀ elemére max egy elemet adunk, akkor a táblázat kitöltése konfliktusmentes és ekkor a gram. **SLR(1) gram.** Az elemzés az LR(0)-hoz

hasonlóan történik: (α,x) kettős, ahol α a verem(párokat tartalmaz), x a szöveg kezdetben: (#0, x#), x az elemzendő szöveg. Egy állapot: (# $0x_1s_1\ x_2s_2\ldots\ x_ks_k,\ a_ia_{i+1}\ldots a_n$ #). • ha action[s_k, a_i] = s_j, akkor (# $0x_1s_1\ \ldots\ x_ks_ka_ij,\ a_{i+1}\ldots a_n$ #), • ha action[s_k, a_i] = rl és az l. szabály az $A \rightarrow \alpha$ és $|\alpha| = m$, akkor (# $0x_1s_1\ \ldots\ x_{k-m}s_{k-m}A_S,\ a_i\ldots a_n$ #), ahol s = goto[s_{k-m}A], • ha action[s_k, a_i] = accept, akkor siker • ha action[s_k, a_i] = error, akkor szintaktikus hiba.

(19. tétel)

LR(1) elemek és az LR(1) elemző

Az SLR(1)-nél az elemek inadekváltságát úgy oldottuk fel, hogy léptetésre az LR(0)-elemekben a pont utáni szimbólumok, redukció esetén pedig az LR(0) elem bal oldalán álló A nemterminális szimbólumra a FOLLOW(A) halmazok diszjunktságát vizsgáltuk. A FOLLOW halmazok azonban globálisak, függetlenek az automata állapotától. **Gondolat:** bővebb gr.osztályt kapunk, az inadekváltságot előreolvasással oldjuk fel. **Def.:** Ha a gr. egy osztálya $A \rightarrow \alpha\beta$, akkor a **gr. egy LR(1)-eleme:** [$A \rightarrow \alpha.\beta$,a] ($a \in T \cup \{\#\}$), ahol az α.β az elem magja, és 'a' az előreolvasási szimbólum. Az előreolvasási szimbólumnak csak akkor van szerepe, ha az LR(1) elem redukciót ír elő, azaz [$A \rightarrow \alpha.$,a] alakú. **Def.:** egy **gr. [$A \rightarrow \alpha.\beta$,a] LR(1) eleme érvényes a γα járható prefixre nézve**, ha $S' \rightarrow^* \gamma Ax \rightarrow \gamma\alpha\beta x$ és az 'a' az 'x' első szimbóluma, vagy ha x=ε, akkor x=#. Az LR(1)-elemzőhöz szükségünk van a kanonikus halmazokra, vagyis a closure és a read függvényekre. **Def.:** ha létezik a gr. egy **LR(1) elemhalmaza**, akkor **closure(J):** i) J ∀ eleme eleme closure(J)-nek is; ii) ha [$A \rightarrow \alpha.B\beta$,a] ∈ closure(J) és $B \rightarrow \gamma$ egy szabály, akkor legyen [$B \rightarrow .\beta$,b] ∈ closure(J) ∀b∈ FIRST(βa)-ra; iii) a ii) pont szerint addig bővítjük, amíg tudjuk. Eszerint, ha δ α járható prefixre nézve az [$A \rightarrow \alpha.B\beta$,a] egy érvényes LR(1)-elem, akkor ugyanerre a prefixre a [$B \rightarrow .\gamma$,b] is egy érvényes LR(1)-elem lesz. **Def.:** ha létezik a gr. egy **LR(1) halmaza**, akkor **read(J,X):** i) ha [$A \rightarrow \alpha.X\beta$,a]∈J, akkor a closure([$A \rightarrow \alpha X.\beta$,a]) ∀ eleme legyen a read(J,X) halmaz eleme; ii) az i) pont szerint addig bővítjük, amíg tudjuk. Rövidítés: [$A \rightarrow \alpha.X\beta$,a/b] két elem. A kanonikus halmazokat ugyanúgy határozzuk meg, mint LR(0)-nál. **A táblázatok kitöltése:** $C=\{J_0,J_1,\ldots,J_m\}$. i) ha [$A \rightarrow \alpha.a\beta$,b] ∈ J_i és read(J_i,a)=J_j , akkor action[i,a]=S_j . ii) ha [$A \rightarrow \alpha.$,a] eleme J_i és $A \neq S'$, akkor action[i,a]=rl, ahol az $A \rightarrow \alpha$ az l-edik szabály. iii) ha [$S' \rightarrow S.$, #] ∈ J_i , akkor action[i,#]=accept. A maradék úgy, mint SLR(1)-nél. **Tétel:** a G' kiegészített grammatika akkor és csak akkor LR(1) gr., ha a hozzá készített kanonikus elemző táblázatok kitöltése konfliktusmentes.

(20. Tétel)

LALR(1) – elemző

Az SLR(1) elemző állapotszámánál az LR(1) elemző állapotszáma lényegesen magasabb. Cél az állapotok számának csökkentése úgy, hogy az elemezhető nyelvek halmaza az LR(1) nyelvekhez viszonyítva lényegesen ne csökkenjen. Az LR(1) elemek kanonikus halmazai között vannak olyanok, amelyekben ugyanolyan magu LR(1) elemek vannak. Egyesítsük ezeket, így kapjuk LALR(1)-et, ahol a halmazok száma az LR(0) kanonikus halmazok számával egyezik meg. A read fv. Az egyesített halmazokra nem okozhat problémát, mert csak a magtól függ. Léptetés/redukálás probléma nem fordulhat elő, mert az eredeti grammatika nem lett volna LR(1). Redukálás/redukálás probléma viszont előfordulhat. Ha az LR(1) elemek kanonikus halmazai: $I_0,I_1, \ldots I_m$, akkor az egyesített halmazok: $R_1, \ldots, R_n$ ($n \leq m$). Ezeket használva töltjük ki a táblázatot, pont úgy, mint az LR(1)-nél. **Definíció**: Ha a G kiegészített grammatikára az LALR(1) elemző táblázatok kitöltése konfliktus mentes, akkor G egy **LALR(1) grammatika**. Az LALR(1) elemző táblázatok kitöltése nagyon munkaigényes, mert az összes LR(1)-elemet meg kell határozni. *Gyorsítás:* a nem I_0 LR(1) kanonikus halmazokban csak azokban az LR(1)-elemekben kezdődik a mag jobb oldala ponttal, amelyek már egy, a kanonikus halmazban levő LR(1)-elemből a *closure* fv-nyel származnak. Így a kanonikus halmazokat nem kell összes elemük felsorolásával megadni. **Definíció:** I_0 **kanonikus halmaz törzse** legyen az [$S' \rightarrow .S$, #] LR(1)-elem; nem I_0 kanonikus halmaz törzsébe pedig azok az LR(1)-elemei tartoznak, amelyekben a mag jobboldala ben ponttal kezdődik. Egy kanonikus halmazt tehát a törzsével is magadhatunk, hiszen abból az összes többi LR(1)-elem előállítható. A törzsből a léptetés és a redukció műveletek is meghatározhatók, így a táblázatok felépíthetők. **A törzs meghatározásának problémája:** Ha az LR(0)-elemek kanonikus halmazaira is definiáljuk a törzs fogalmát, akkor láthatjuk, hogy ezek a törzsek megegyeznek az LR(1) kanonikus halmazaiból egyesítéssel létrehozott halmazok törzseivel(a magok azonosak). Ha a törzseket(LR(0)) kiegészítjük előreolvasási szimbólumokkal, akkor az egyesített LR(1) halmazok törzseit fogjuk megkapni. Az LR(0)-elemekre, ha ismerjük I_j-t, akkor a read(I_j,x) könnyen meghatározható. LR(1)-nél ez már bonyolúltabb, mert ahhoz az előreolvasási szimbólumot is meg kell határozni, vagyis, hogy milyen a-ra lesz [$A \rightarrow X.\alpha$, a] ∈ READ(R_j , x). Ha $\mu\delta \neq \varepsilon$ és a∈ $FIRST_1(\mu\delta)$, akkor biztosan [$A \rightarrow X.\alpha$, a] ∈ READ(R_j, x)-hez spontán generálható, ([$B \rightarrow \gamma.C\delta$, b] ∈ R_j , $C \rightarrow^* A\mu$ és $A \rightarrow X\alpha$) hiszen a 'b' szimbolumnak semmilyen szerepe nincs az új előreolvasási szimbólum meghatározásában. Ha $\mu\delta = \varepsilon$, akkor [$A \rightarrow X.\alpha$, b] lesz a READ(R_j, x) eleme, azaz ebben az elemben a 'b' lesz az előreolvasási szimbólum, ekkor azt mondjuk,

hogy a ’b’ R_j-ből öröklődik a READ(R_j, x)-be. Ha adott az I_j törzse, akkor minden X-re a READ(R_j, X)-hez az előreolvasási szimbólumok meghatározása: **i,** minden [B → γ.δ]ЄI_j-re határozzuk meg az R_j´ = CLOSURE([B → γ, @]) halmazt, ahol ’@’ egy dummy szimbólum. **ii,** Ha [A → α.Xβ, a]ЄR_j´ és a ≠ @, akkor [A → αX.β, a] Є READ(R_j, X) és az ’a’ spontán generálható. **iii,** Ha [A → α.Xβ, @]Є R_j´, akkor [A → αβ, @] Є READ(R_j,X) és a @ öröklődik a READ(R_j,X)-be. Ha már ismerjük a spontán generált és az öröklődő előreolvasási szimbólumokat, akkor az egyesített halmazok törzséhez meg tudjuk határozni az összes előreolvasási szimbólumot: egy táblázat tartalmazza az LR(0) elemek halmazainak a törzsét, az elemekhez tartozó spontán generált előreolvasási szimbólumokat és az R_0-hoz definiált #-ot. Ezután az előreolvasási szimbólumokat az öröklődések alapján átvisszük az összes lehetséges elemhez.

(21.tétel)

Az if-then-else LL(1) és SLR(1) elemzése

LL(1): utasítás→ …| if utasítás |… ; if utasítás→if kifejezés then utasítás | if kifejezés then utasítás else utasítás. Ezekkel a szabályokkal a grammatika nem egyértelmű→faktorizáció: if utasítás→if kifejezés then utasítás if vége; if vége→ε | else utasítás. Még mindig nem LL(1)-es a gr. (A→γ | δ : $FIRST_1(\gamma FOLLOW_1(A)) \cap FIRST_1(\delta FOLLOW_1(A)) = Ø$ –nak nem tesz eleget). A probléma az egymásba skatulyázott if-ekkel van. A rekurzív leszállásnál az else ahhoz az if-hez tartozik, amelyikhez az if utasítás procedúrája hozzárendeli:

```
*procedure if-utasítás;#
   Begin#
      Accept(if-szimbólum);#
      Kifejezés;#
      Accept(then-szimbólum);#
      Utasítás;#
      if akt-szimbólum=else-szimbólum then#
         Begin#
            Accept(else-szimbólum);#
            Utasítás;#
         End;#
   End;#*
```

SLR(1): i:=if expr then, e:=else, a:=nem if utasítás. Ekkor a grammatika:

S’→S

S→iS | iSeS | a

Az LR(0)-elemek kanonikus halmazai:

J_0={[S’→S.],[S→.iSeS],[S→.iS],[S→.a]}#

J_1={[S’→S.]}#

J_2={[S→i.SeS],[S→i.S],[S→.iSeS],[S→.iS],[S→.a]}#

J_3={[S→a.]}#

J_4={[S→iS.eS],[S→iS.]}#

J_5={[S→iSe.S],[S→.iSeS],[S→.iS],[S→.a]}#

J_6={[s→iSeS.]}#

J_4 miatt a 4. állapotban az ’e’ szimbólumra léptetés/redukálás konfliktus lép fel, mivel $FOLLOW_1(S)$={e,#}. Feloldás: ha a veremben ’iS’ van és az ’e’ következik, akkor hajtsunk végre léptetést.

(22. tétel)

Hibakezelési módszerek az LL(1) és\\ SLR(1) elemzésekben}

LL(1): Hibaelfedési eljárás a rekurzív leszállás módszeréhez. Alapprobléma: ha a szintaktikus elemző hibát detektált, akkor hogyan lehet az elemző programot és a szimbólumsorozatot újból szinkronizálni. Arra törekszünk, hogy minden procedúrahívás és procedúrából való kilépés szinkronizált legyen. A kihagyott szimbólumok száma a procedúrában meghívott procedúrák számával egyenlő. Minden szimbólumhoz meg tudjuk határozni a $FIRST_1(A) terminálisok halmazát, legyen ez 'starters'. SKIPTO(starters): az input sorozat szimbólumait átúgorja, amíg a starters halmazba tartozót nem talál. Hasonlóan a $FOLLOW_1(A)$ halmaz legyen 'followers'.

*Okosítás: SKIPTO(starters+followers)#

Okosítás: followers := followers ∪ $FOLLOW_1(B)$, ahol B az A-t hívó procedúra.#

```
procedure A(followers);#
  begin#
    if not(akt_szimbólum in starters) then#
      begin#
        error(...);#
        skipto(starters + followers);#
      end;#
    if (akt_szimbólum in starters) then#
      begin#
        T(A)#
        if not (akt_szimbólum in followers) then#
          begin#
            error(...)#
            skipto(followers)#
          end;#
      end;#
  end;#*
```

SLR(1): Hiba a táblázatból sosem származhat, az \emph{action} táblázatban (az operátor - precedenciához hasonlóan) hibajelzéseket tehetünk és különböző típusú hibákhoz különböző tevékenységeket rendelhetünk. Egy egyszerű hibaelfedési módszer: vezessünk be egy error terminálist, majd válasszuk ki a legfontosabb nemterminálisokat (pl.: program, blokk, utasítás) és az ezekre vonatkozó szabályokat egészítsük ki egy $A \rightarrow$error alternatívával. Ez alapján készítsük el a táblázatokat. Ha az elemző a táblázat alapján hibát talál, akkor a veremből (elempárokat) törölve visszalép ahhoz a legközelebbi állapothoz, amelyikből az error egy másik állapotba vezet. Ekkor az error szimbólumot a verembe teszi, az állapotot pedig a táblázat alapján határozza meg. Ha $\alpha \neq \varepsilon$, akkor az inputban is előrelép addig, amíg az input pointer az első szimbólumra nem mutat.

(23.tétel)

Szemantikus elemzés veremmel}

Szemantikus elemzés a szemantikus fa alapján történik. A szintaxis fa pontjaihoz olyan attribútumokat rendelünk, amelyek leírják az adott pont tulajdonságát. Ezeknek az attribútumoknak a meghatározása és konzisztenciájuk vizsgálata a szemantikus elemzés feladata. A vizsgált tulajdonságok (i) változók deklarációja, hatásköre, láthatósága (ii) többszörös deklaráció, deklaráció hiánya (iii) operátorok és operandusok közötti kompatibilitás (iv) procedúrák, tömbök paraméterei közötti kompatibilitás. Az első kettő vizsgálat megoldható egy jól választott szimbólum-tábla kezeléssel. Akciószimbólumok: Ha a szintaxisfa felépítésekor egy adott szabály esetében szemantikus elemzést is kell végezni, akkor a szabályokba 'akciószimbólumokat' írhatunk. '@Prog' esetén a 'Prog' szemantikus rutint kell végrehajtani. Ha a G grammatikát akciószimbólumokkal egészítjük ki, akkor TG fordítási grammatikát kapunk. Probléma, hogy a szemantikus rutinoknak nincs paraméterük. Mivel a szintaktikus elemzők vermet használnak, ezért a rutinok egymás közti paraméterátadásaira is vermet fogunk használni. A **szemantikus verem** lehet: **akcióvezérelt** a rutinok kezelik a verem push és pop műveleteit (vannak hátrányai) **elemzővezérelt**: a szemantikus verem kezelése a szintaktikus verem kezelé\-sével párhúzamos, az elemző egyszerre hajt bennük végre műveleteket. Elemzővezérelt szemantikus verem: **Alulról felfelé elemzés (LR(1)):** Ha $A \rightarrow \alpha$ szerinti redukció következik, akkor az α szimbólumai a szintaktikai veremben vannak, a szemantikai veremben pedig a szimbólumokhoz tartozó bejegyzések vannak. Ezek a bejegyzések csak akkor használhatóak egy rutin paramétereiként, ha kikerülnek a szemantikai veremből, vagyis redukciónál. Tehát alulról-felfelé elemzésnél rutin csak helyettesítési szabályhoz adható meg (egyhez csak egy), végrehajtani pedig az adott szabály szerinti redukciónál lehet. A szabály jobb oldalának szimbólumaihoz tartozó attribútumok címzése: a pop művelet ne változtassa meg a verem tartalmát, csak az SP veremmutatót mozgassa. Így a redukáláskor végrehajtandó pop műveletek után, de a push művelet előtt az attribútumokat SP+1, SP+2,… módon tudjuk címezni és nem kell a kiolvasott értékeket külön tárolni. **Felülről lefelé elemzés (LL(1)):** Ha $A \rightarrow \alpha$ szabályt alkalmazunk, akkor egy pop művelettel az A szimbólumot kivesszük a veremből, majd az α szimbólumait a verembe tesszük push-sal. Ha a két vermet párhúzamosan kezelnénk, akkor $A \rightarrow \alpha$-nál elveszne az A-hoz tartozó szemantikus elem, amire később még szükség lehet. Amikor a szintaktikai verem tetején álló szimbólum megeggyezik a következő input szimbólummal, akkor is pop-ot kellene végrehajtani, de akkor is elvesztenénk az attribútumot. Megoldás: egy $A \rightarrow X_1, X_2, \ldots, X_n$ szabály esetén egyik veremre se hajtsuk végre a pop műveletet. A szintaktikai veremben az A helyére írjunk egy EOP kódot, a szemantikus verembe pedig tegyük bele A-t. Ezután már

végrehajthatjuk a push műveleteket az X_i-kre, de az akciószimbólumokat csak a szintaktikai verembe tegyük be (a másikba felesleges). A szemantikus verem kezelésére három pointert használunk: $A \rightarrow X_1, X_2, \ldots, X_n$ szabály esetén left az A, right az X_n, top pedig az X_1 attribútumait tartalmazó mezőkre mutat. Az EOP szimbólum beírásakor mentsük el a pillanatnyi állapotot is EOP(left,right,top) formában. Ha a szintaktikai verem tetején egy EOP van, akkor az $A \rightarrow X_1, X_2, \ldots, X_n$ feldolgozása megtörtént, a szemantikus veremből ki kell törölni a top és right közötti attribútumokat.

(24. tétel)

Attribútum fordítási grammatikák

A rutinok közötti **információátadást** most úgy oldjuk meg, hogy a gr. nyelvtani- és akciószimbólumaihoz attribútumokat rendelünk és megadjuk, hogy ezeket hogyan kell használni a szemantikus műveletek végrehajtásában. Legyen az akciószimbólumok halmaza: @φ, A: attribútumok véges halmaza, $\forall x$: $A(x) \subset A$, $\forall @s$: $A(@s) \subset A$, V: attribútumértékek halmaza, R: szemantikus szabályok halmaza, ha $\forall p \in P$szabályra: $R(p) \subset R$, R(p) elemei szemantikus fv-ek, $R(p): A(p) \rightarrow V$, ahol A(p) a p-hez tartozó összes attribútum előfordulás halmaza. Az X szimbólum ($X \in T \cup N \cup @$ φ) attr. értékeinek egyértelműnek kell lenniük, vagyis $\forall x \in L(ATG)$: az x-hez tartozó szintaxisfa minden x pontjában minden attr. értéket max. 1 szemantikus fv. határozhatja meg. C: logikai feltételek halmaza, $\forall p \in P$: $C(p) \in C$. C(p) egy logikai állítást mond ki a p szabályhoz tartozó A(p) attr.-okra; ha aza állítás a p szabály feldolgozásakor nem teljesül, akkor a szemantikus elemző hibát jelez. **DEF.:** az **ATG**=(TG, A, V, R, C) ötöst attr. **fordítási grammatikának** nevezzük. Az X szimbólum egy attr.-át **szintetizált**nak nevezzük, ha értékét egy szemantikus fv. abban az esetben határozza meg, amikor az X egy szabály bal oldalán áll. Ha a jobb oldalán áll, akkor **örökölt**. Tehát az információt egy szintaxisfában a szintetizált attr.-ok alulról felfelé, az örökölt attr.-ok pedig felülről lefelé és szinten belül továbbítják. A terminális szimbólumokhoz tartozhatnak **kitüntetett szintetizált attr.-ok**, melyeknek értékét egy konstans fv. (külső eljárás, pl. lexikális elemző) adja meg. Jelölés: X.a az X-hez tartozó 'a' attr. $AF(p)=\{X.a \mid X.a \leftarrow f(...) \in R(p)\}$, vagyis az R(p) fv.-ei által meghatározott attr.-ok halmaza. Ekkor az X.a szintetizált attr., ha létezik $p: X \rightarrow \alpha$: $X.a \in AF(p)$, és X.a örökölt, ha létezik $q: A \rightarrow \alpha X \beta$: $X.a \in AF(q)$. @s akciószimbólumra: @s.a örökölt, ha a @s-t tartalmazó p szabályra: $@s.a \in AF(p)$, és szintetizált, ha értékét az s rutin határozza meg. Szintetizált: $X\uparrow a$, örökölt: $X\downarrow a$. Létezhet X.a és Y.a, de ez semmilyen összefüggésre nem utal. **Tétel:** egy ATG grammatika X ($X \in T \cup N \cup @$ φ) szimbólumainak szintetizált és örökölt attr.-ai: φ(X) és μ(X). Ekkor: $\varphi(X) \cap \mu(X) = \emptyset$. **DEF.:** egy **ATG teljes**, ha $\forall$: i)$p: A \rightarrow \alpha$: $\varphi(A) \subseteq AF(p)$, ii) $p: A \rightarrow \alpha @s,b$: $\varphi(@s) \subseteq AF(p)$, iii) $p: A \rightarrow \alpha X,b$: $\mu(X) \subseteq AF(p)$ ($X \in T \cup N \cup @$ φ), iv) X szimbólumra: $A(X) = \varphi(X) \cup \mu(X)$. Egy szintaxsfa pontjaiban szeretnénk meghatározni az attr. értékeket. Először csak a levelekhez tartozó kitüntetett szintaktikus attr.-ok értékei ismertek. A többit a szemantikus elemzés határozza meg. A meghatározás sorrendje közömbös, de az kell, hogy egy szemantikus fv. alkalmazása esetén az argumentumok értéke már ismert legyen. **DEF.:** egy **ATG jól definiált**, ha minden szintaxisfa minden pontjában minden attr. értéke egyértelműen kiszámítható. Minden jól definiált ATG teljes, de fordítva nem igaz. Jelölés: (X.a, Y.b) X.a attr. értékére szükség van Y.b értékének meghatározásához. **DEF.:** a $p: X_o \rightarrow X_1 \ldots X_n$ szabályhoz tartozó **direkt attr. függőségek: DP(p)**=$\{(X_i.a, X_j.b) \mid X_j.b = f(\ldots, X_i.a, \ldots) \in R(p)\ 0 \le i,j \le n\}$. A direkt attr. függőségek egy adott szintaxisfára egy függőségi gráfot generálnak: csúcsok=attr.-ok, élek=direkt függőségek. **DEF.:** egy **ATG lokálisan aciklikus**, ha $\forall p \in P$-re: DP(p) körmentes. **DEF.: DT(x)** az x mondat levezetésében felhasznált összes p szabályhoz tartozó DP(p) **függőségek halmaza**. **Tétel:** egy teljes ATG jóldefiniált, ha a nyelv minden x mondatára a DT(x) gráf körmentes. Minden jól definiált ATG-hez megadható egy nemdeterminisztikus algoritmus, amely tetszőleges mondat szintaxisfájának minden pontjában meghatározza az attr.-ok értékét: minden attr. érték kiszámításához rendeljünk egy processzt, ami akkor indul el, ha a kiszámításához szükséges összes attr. érték már ismert, és ha befejeződik, akkor az általa meghatározott értékekkel további processzek indulhatnak el. Az algoritmus a kitüntetett szintetizált attr.-ok felhasználásával kezdődik.

(25. tétel)

Rendezett attribútum fordítási grammatikák

A nyelv mondatain értelmezett nem feltétlenül véges attribútum függő\-ségeket a szabályokhoz és a szimbólumokhoz kapcsolódó, az attribútomok közötti véges darabszámú függőségekre képezzük le. DS(X) legyen parciális rendezés $\forall X$-re, ez határozza meg az attribútumok kiszámítási sorrendjét pl.: $(X.a, X.b) \in DS(X)$. **Def:** Az A(X) egy $A_1(X)$, $A_2(X)$, …, $A_{m(x)}(X)$ particionálását **megengedett particionálás**nak nevezzük, ha (a) $A_{m(x)}, A_{m(x)-2}, A_{m(x)-4} \ldots \subseteq \varphi(X)$, (b) $A_{m(x)-1}, A_{m(x)-3}, A_{m(x)-5} \ldots \subseteq \mu(X)$. **Def:** Egy **ATG partícionált**, ha lokálisan aciklikus és minden X-hez létezik egy olyan $A_1(X), A_2(X), \ldots, A_n(X)$ megengedett partícionálás, hogy az X attribútumai a partíciók sorrendjében

meghatározhatók. **Köv:** Ha egy ATG partícionált, akkor jóldefiniált $A_i(X)$et a DP(p) direkt függőségek figyelembevételével kell meghatározni (DP(p) a p-hez tartozó szem. fv.-ek halmaza) Mivel DP(p) tranzitív, ezért képezzük NDP(p) normalizált tranzitív lezárását: $NDP(p) = DP^+(p) - \{(X.a, X.b) \mid X.a, X.b \in AF(p)\}$ (AF(p) a szemantikus fv.-ekben a baloldalak halmaza). Legyen IDP(p) a p szabályra vonatkozó, IDS(X) az X attribútumai között fennálló indukált függőségek: **Def:** Egy **ATG indukált függőségei:** (i) $\forall p \in$ P-re IDP(p) = NDP(p), (ii) $\forall X$: $IDS(X) = \{(X.a, X.b) \mid \exists q: (X.a, X.b) \in IDP^+(q)\}$, (iii) $\forall p: X_0 \rightarrow X_1 \ldots X_n$: $IDP(p) = IDP(p) \cup IDS(X_0) \cup \ldots \cup IDS(X_n)$, (iv) A (ii) és (iii) pontot ismételjük amíg van változás. **Tét:** Ha egy ATG partícionált akkor $\forall p$ és $\forall X$: IDP(p) és IDS(X) gráfja körmentes. Ha $(X.a, X.b) \in IDS(X)$, akkor $X.a \in A(X_i)$ és $X.b \in A(X_j)$ és $i \leq j$. A probléma az, hogy egy partícionálás további függőségeket indukál, hiszen két különböző partíció beli attr.-ra a paríciók automatikusan relációt írnak elő. **Def:** Legyen $A_1(X)$, $A_2(X)$, …, $A_{m(x)}(X)$ megengedett partícionálás és $p: X_0 \rightarrow X_1 \ldots X_n \in DP(p)$ kiterjesztett attr függőség. **EDP(p):** $EDP(p) = IDP(p) \cup \{(X_i.a, X_i.b) \mid X_i.a \in A_j(X_i), X_i.b \in A_k(X_i)\}$, ahol $0 \leq i \leq n$, $j < k$. **Tét:** Egy ATG partícionált $\Leftrightarrow$ $tp \in P$: EDP(p) gráf körmentes. **Def:** Egy **ATG rendezett ATG**, ha a következő, az A partícionálását végző algoritmussal partícionált grammatikát kapunk: $\forall X$: a partíciók legyenek $A_1(X)$, $A_2(X)$, …, $A_{m(x)}(X)$, ahol $A_i(X) = T_{m-i+1}(X) - T_{m-i-1}(X)$, (i=1..m(x)), ahol: $T_{-1}(X) = T_0(X) = 0$, m(X) az a legkisebb olyan index, amelyre $T_{n-1}(X) \cup T_n(X) = A(X)$ és $\forall k>0$: $T_{2k-1}(X) = \{X.a \in \varphi(X) \mid$ ha $(X.a, X.b) \in IDS(X)$, akkor $X.b \in T_j(X), j \leq 2k-1\}$, és : $T_{2k}(X) = \{X.a \in \mu(X) \mid$ ha $(X.a, X.b) \in IDS(X)$, akkor $X.b \in T_j(X), j \leq 2k\}$. **Köv:** Minden rendezett ATG partícionált! **Eredmény:** Ha egy rendezett ATG-hez meghatározzuk $A_i(X)$-eket, akkor $DS(X) = \{(X.a, X.b) \mid X.a \in A_i(X), X.b \in A_j(X)\}$, $1 \leq i, j \leq m(x)$, $i \leq j\}$. Az attribútumok meghatározásának sorrendje független a nyelv mondataitól, csak a grammatikától függ!

(26. tétel)

A bejárási út és az elemző program

Az attribútum értékeket meghatározó program: $\forall p \in P$: VP(p) a p-szabályhoz tartozó szintaxisfa egy olyan bejárása, amely a szabályhoz tatozó attribútumokat értékeli ki, ezt nevezzük látogatási sorozatnak. Tegyük fel, hogy már minden x-hez meghatároztuk az $A_1(x)$,…, $A_{m(x)}(x)$ partíciókat. Legyen $p: x_0 \rightarrow x_1 \ldots x_n$. **VP(p) meghatározása:** ha $x_i.a \in A_k(x_i)$ és $x_i.a$ nem$\in AF(p)$, azaz az attribútum értékeket nem a p-szabályban határozzuk meg, akkor az $x_i.a$ attribútumot jelöljük $c_{k,i}$-vel. Ezek a $c_{k,i}$ attribútumok az α_0 örökölt attr.-ai, azaz $A_{m(x0)-1}(x_0)$, $A_{m(x0)-3}(x_0)$ elemei, és az x_i $(1 \leq i \leq n)$ szimbólumok szintetizált attr.-ai, azaz $A_{m(xi)}(x_i)$, $A_{m(xi)-2}(x_i)$ elemei. Mivel minden VP(p)-nek a C(p) logikai feltétel kiértékelésével kell befejeződnie, ha $A_{m(xi)}(x_i)=\emptyset$, akkor ez jelentse a C(p) kiértékelését, ha x_i a p szabály bal oldalán áll. Először meghatározzuk az összes összefüggőséget, de EDP(p) mellett DP(p)-t is figyelembe vesszük. Az attr. függőségek egy parciális rendezést adnak, ezt kiegészítjük teljes rendezéssé (új relációkat vezetünk be, a tranzitivitás miatt feleslegessé váló relációkat elhagyjuk és figyelembe vesszük, hogy C(p)-vel kell befejezni a bejárást). A **teljes rendezés** definiálja a VP(p) látogatási sorozatot: ha $(x_i.a, x_j.b)$ eleme a rendezésnek, akkor először az $x_i.a$ pontba menjünk, ha ez a pont egy AF(p)-beli attr., akkor határozzuk meg az értékét a hozzá tartozó szemantikus fv-nyel; ha a ponthoz a $c_{k,0}$ jel tartozik, akkor rendeljük hozzá azt a programot, amelyik átlép a szintaxisfában a pont őséhez; ha a ponthoz a $c_{k,i}$ $(i \neq 0)$ jel tartozik, akkor pedig a leszármazottjához.

(29.tétel)

Prológus, epilógus, típusdeklarációk, változódeklarációk

A kódgeneráló egy olyan assembly nyelvű programot késszít, amelyben egy adat-, egy verem-és egy kódszegmens van. Az adatszegmensbe kerrülnek a program globális változói, a veremszegmensbe az alprogramok aktivációs rekordjai, a kódszegmensbe a lefordított utasítások. A lefordítandó forrásnyelvű program: <program> → *program* <id> ↑v @Prologue (↓v) <block> @Epilogue, ahol @Prologue(↓v) az assembly program bevezető sorait generálja (v a neve), @Eplilogue pedig a végét. A .DATA és a .CODE direktívák bárhol és többször is előfordulhatnak, az alprogramoktól függően. Most csak egy alprogrammal foglalkozunk. **Típusdeklaráció:** <típusdeklaráció> → *type* <id> ↑n = <típusdefiníció> ↑d @TypeDecl (↓n,d) ahol @TypeDecl végzi a szimbólumtábla kezelését: az n névhez egy bejegyzést készít és ehhez kapcsolja a d típusdeszkriptort. Fontos probléma a típusok ekvivalenciája és kompatibilitása. Két típus ekvivalenciájának vizsgálata a tipusdeszkriptorokra mutató pointerek azonosságának vizsgálata. Egy kifejezésben egy operátor operandusainak kell kompatibilisnek lenniök. **Konstans deklaráció:** pl.: constant real pi := 3.14;. <konstansdeklaráció> → *constant* <típus> ↑t <id> ↑n := <kifejezés> ↑s,v @ConstDecl (↓n,t,s,v) ahol t: type, n: name, v: value, s: az érték típusa, @ConstDecl ellenőrzi t és s azonosságát és n,t,v,-vel a szimbólumtáblába ír (n EQU v; @ConstDecl). **Változódeklarációk:** A változók lehetnek statikusak (egész, valós), vagy dinamikusak (stringek, dinamikus tömbök). A dinamikus változók fordításakor csak a változó deszkriptora

számára kell helyet foglalni. **(a) statikus, predefinit típusú egyszerű változók** *<egyszerű_változó_deklaráció> → <típus> ↑t,m <id> ↑n @VarAlloc (↓n,m,↑q) @VarDecl (↓n,t,m,q)# <típus> → real ↑t,m | integer ↑t,m | boolean ↑t,m | character ↑t (<darabszám> ↑m)#* ahol t: type, m: méret byte-okban, n: name, VarDecl: a változót a szimbólum táblába írja, @VarAlloc: helyet allokál a változónak ha globális változó, akkor az adatszegmensben n DB m DUP(?); @VarAlloc, ha lokális változó, akkor egy aktivációs rekordban n EQU WORD PTR[BP]-k ; @VarAlloc. **(b) felsorolás típusú változók:** Intenzív szimbólumtábla kezelést igényel, mert az azonosító lista elemeit is fel kell írni a szimbólumtáblába. **(c) intervallum típusú változók:** Az intervallum határait kell tárolni. <intervallum_változó_deklaráció> → <id> ↑n: <integer_exp> ↑s @Lowerbound(↓s) . . <integer_exp> ↑r @Upperbound(↓r) @InterVarAlloc (↓n,s,r,↑q) @InterVarDecl (↓n,t,s,r,q). **(d) tömbök:** Sorfolytonosan szoktuk tárolni, n-dimenziós tömb memóriacíme loc, i-edik dimenzió határai L(i) és U(i), így az (i_1, …, i_n) indexű tömbelem címe: loc + $\sum^{n}_{j=1}$ (i_j – L(j)) P(j), ahol P(j) = $\Pi^{n}_{k=j+1}$ (U(k) – L(k) + 1) P(n) = m, (m a tömb mérete byteban), legyen RC = $\sum^{n}_{j=1}$ L(j) P(j), így a cím: loc – RC + $\sum^{n}_{j=1}$ i_j P(j). A tömb descriptora: n:dimenzió , L(i), U(i), P(i), RC, loc. *<tömb_deklaráció> → *array* @InitDim(↑i) <id> ↑n [<sublist>↓i↑j] *of* <típus> ↑t,m @ArrayAlloc(↓n,t,m,j,↑q) @ArrayDecl(↓n,tm,jiq)# <sublist> ↓i↑j → <sublist> ↓i↑j | <sublist> ↓i↑j @echo(↓j↑i), sublist↓i↑j# <sublist> ↓i↑j → @Dimension(↓i,↑j) <integer_exp> ↑s @Bounds(↓s,j) | <integer_exp>↑r @LowerBound(↓r,j) <int_exp> ↑s @UpperBound#* ahol @InitDim(↑i) i változót deklarál a dimenziónak, @Dimension(↓i,↑j) j←i+1, és @Bounds(↓s,j) L(j)←1, U(j)←s. **(e) rekordok:** Az elemek offset címeinek meghatározása a fő feladat. *<rekord_deklaráció> → <id> ↑n: *record* @InitOffset(↑r) <elem_lista>↓r↑s @RecAlloc(↓n,s,↑q) @RecDecl *end record*# <elem_lista>↓r↑s → <elem>↓r↑s | <elem>↓r↑s @echo(↓s,↑r), <elem_lista>↓r↑s# <elem>↓r↑s → <idn>↑e : <típus>↑t,m @RecElemDecl(↓e,t,m,r,↑s)#*

(30. tétel)

Kifejezések, logikai kifejezések

Kifejezések: A kifejezéseket a következő példán tanulmányozzuk:

*(1) E→TE',#
(2) E'→+TE' | ε,#
(3) T→FT',#
(4) T'→*FT' | ε,#
(5) F→(E) | i #*.

Csak integerekkel (két byte-os) dolgozunk. A kif eredményét az AX regiszterben tároljuk, a részeredményeket a run-time veremben. Az utolsó szabályban szereplő 'i' esetei: (1) 'i' egy egyszerű változó. F→(i)↑n @SearchVar(↓n,↑t,q) StLoadAX(↓q), ahol t:type, n:azonosító, q: a deklarációban meghatározott címe, @SearchVar: megkeresi az n szimbólumot a szimbólum-táblában, ellenőrzi a láthatóságot és visszaadja a típusát és a címét, @StLoadAX: a változó értékét az AX regiszterbe töltő utasítást generálja: MOV,AX,q; globális változókra: MOV AX,n; lokális vált: MOV AX, WORDPTR[BP]-k; . (2) 'i' egy konstans deklarációjában szereplő név MOV AX,n. (3) egy konstans literál, akkor legyen a 'q' értéke az 'i'-t alkotó literál. (4) 'i' egy tömb elem: *F→i↑n @InitDim(↑i) @SearchVar(↓n,↑t,d,q) [<sublist>↓i,d↑j] @GenLabel(↑r) @ArrayRef(↓t,d,q,j,r)# <sublist> ↓i,d,↑j → <subscript> ↓i,d↑j | <subscript>↓i,d↑j @echo(↓i,↑j) <sublist>↓i,d,↑j# <subscript>↓i,d,↑j → @Dimension(↓i,↑j) <integer_exp> ↑s @CallIndex(↓j,s,d)#* ahol @InitDim(↑i) i változót deklarál a dimenziónak, @Dimension(↓i,↑j) j←i+1, @SearchVar: visszadja a descriptort (d-ben), @CallIndex(↓j,s,d): olyan utasításokat generál, amelyek az AX-ben lévő 's' értéket összeszorozzák a tömb descriptorában lévő P(j) értékkel:

```
*MOV BX,P(j);#
IMUL BX;#
PUSH AX;#*
```

, @ArrayRef: ellenőrzi a dimenziószámot és kódot generál:

```
*   LEA BX,loc;#
    SUB BX,RC;#
    MOV CX,j#
L_r: POP AX;#
    ADD BX,AX;#
    LOOP L_r;#
    MOV AX,[BX]#*
```

@GenLabel: generálja az L_r címkét a LOOP-nak. (5) A statikus rekord egy elemének címe: rekord eleje+offset. Az 'E' kifejezést tovább vizsgálva, legyen:
*@StPushAX: PUSH AX;#
@StPopBX: POP BX;#
@StAddBX: ADD AX,BX;#
@StImulBX: IMUL BX;#
A kifejezést leíró grammatika:#
E→TE'#
E' → + @StPushAX T @StPopBX @StADDBX E' | ε#
T→FT'#
T' → * @StPushAX F @StPopBX @StImulBX T' | ε#
F → (E) | i↑n @ … (i-től függ)#*
Logikai kifejezések: FALSE: 0, TRUE: nem 0; + és * helyett or és and van, így kell @StAndBX, @StOrBX és @Negate: NEG AX. Megadható, hogy olyan kódot generáljon a fordításkor, amelyik optimalizálja a kiértékelést, ha lehet
*E → @GenLabel(↑s) D @StJTrue(↓s) or E @StLabel(↓s) | D#
D → @GenLabel(↑r) C @StJFalse$(↓r) and D @StLabel(↓r) | C#
C → not N@Negate | N#
N → (E) | i #*
ahol @GenLabel$(↑r) létrehoz egy cimkét, @StLabel(↓r) kiad egy „L_r:" sort, és @StFalse(↓r): CMP AX,0; JNZ B+5 ?????; JMP L_r. (*a ?-jelek lábjegyzetéhez az volt írva, hogy mi a pina ☺*)

(31. tétel)
Utasítások
Az értékadó utasítás: a legtöbb magasszintű programnyelvben nincs lehetőség egy változó címének elérésére, a szemantikus elemzőnek kell el-döntenie, hogy adott esetben a változó címére, vagy értékére hivatkozunk-e. <értékadó_utasítás> → @SET$(↑r) <változó> ↓r↑t := @RESET (↓r) <kifejezés> ↑s @STORE(↓r,t,s), ahol r: a változó címét, vagy értékét kell-e meghatározni, @STORE: ellenőrzi t és s típusát és típuskonvertál, ha kell. **Az if és case utasítás**: *<if_utasítás> → if <kifejezés> then <utasítás$_1$> <if_tail># <if_tail> → else <utasítás$_2$> endif | endif#*. Az endif bevezetésével megszűnik a nem-egyértelműség! kódgen: *<if_utasítás> → if <kifejezés> @GenLabel(↑r) @StJFalse(↓r) then utasítás <if_tail># <if_tail> → else @GenLabel(↑s) @StJMP (↓s) @StLabel(↓r) <utasítás> endif @StLabel(↓s) | endif @StLabel(↓r)#*. A case fordítása lényegében megegyezik az if fordításával, de gondoskodni kell arról, hogy az egyes elágazások feltételeinek fordításakor rendelkezésre álljon a case utáni kifejezés (@SaveExp, @LoadExp, @ResetExp). **Ciklusutasítások**: **feltétel nélküli ciklus :** <ciklusutasítás> → loop @GenLabel (↑r) @StLabel(↓r) <utasítások> endloop; @StJMP(↓r). **while-ciklus :** *<while_utasítás> → <while_head> ↑r,s do <utasítás> <while_tail> ↓r,s# <while_head>↑r,s → while @GenLabel(↑r) @StLabel$(↓r) <kifejezés> @GenLabel(↑s) @StJFalse(↓s) <while_tail>↓r,s @StJMP(↓r) @StLabel(↓s)#*. **for-ciklus :** (*nem írtak ide semmit*). **GOTO utasítás:** A goto fordítása a már ismert predefinit/postdefinit hivatkozások fordításához hasonló. A fordítás csak az őt deklaráló blokkban lehet. **Az exit és a return utasítások**: exit(break): a vezérlés a ciklus utáni első utasításra adódik. Probléma: Hol van az exitet tartalmazó legbelső ciklus vége?. Megoldás: Minden ciklushoz rendelünk egy logikai attr.-ot (e) Minden ciklus kezdődjön @ResetExit(↑e) (jelentése e := FALSE) és @GenLabel(↑f) rutinokkal ill. végződjön @CoudExit(↓e,f)-el ami ha e=TRUE, akkor @StLabel (↓f). Legyen: <exit_utasítás>↓e,f → exit @SetExit(↑e) @StJMP(↓f). **return:** az alprogram végrehajtása befejeződik, fordítása az exit-hez hasonló. **Kivételkezelés:** Ha kivétel lép fel, akkor a kivételkezelő handler kezdőcímére kell a vezérlést adni és biztosítani kell, hogy a vezérlés a kivételt kiváltó utasítást követő utasításra visszatérhessen. **Átviteli vektor:** a handlerek címeit tartalmazza, a kivételek sorszámának sorrendjében. A forrásnyelvű programot úgy osztjuk tartományokra, hogy egy tartomány összes utasítására ugyanaz a kivételkezelés legyen érvényes. Ekkor egy tartomány-térkép segítségével a program minden pontján egyértelműen megadhatjuk, hogy az i-edik tartományban a j-edik kivételhez a handler$_{ij}$ tartozik.