Fordítóprogramok
1

PTI szak

2022/23-s tanév II. félév

.

Programozási nyelvek (Programnyelvek)

Magasszintű nyelvek

számítógép függetlenek
hardware funkciókat nem biztosítják
fordítóprogram szükséges, amely
adott operációs rendszer alatt fut és
adott operációs rendszer alá fordít
Vannak: felhasználó orientált nyelvek
(általános probléma könnyen megfogalmazható)
probléma orientált nyelvek
(speciális terület pl folyamatirányítás)

Alacsonyszintű nyelvek

Gépi kód
(utasításai bináris numerikus értékek (kódok),
operandusok, memóriacímek szintén binárisak)
Assembly
(A gépi kód szimbolikus megfelelője)
A Makro-assembly közelít a magasszintű struktúrákhoz

Programozási nyelvek: véges (nemcsak aritmetikai) számítások formális leírásának humán-orientált eszközei

Assembly nyelvek: a magas szintű programozási nyelvek és a gépi szintű nyelvek köztes nyelvei.
Gépi kód helyett mnemonic-okat és előre megírt programrészeket használ.

Gépi nyelvek: a számítógép nyelve, mellyel a feladatot végre tudja hajtani

Fordítás: általában egy humán-orientált forrásnyelven kifejezett algoritmus fordítása hardver-orientált célnyelvre. (Nem mindig: re-esszemblálás – gépi nyelvről assembly szintű nyelvre visszafordítás, többnyire szoftverlopás céljából)

Nyelvi szintek:	magas szintű nyelv	assembly nyelv	gépi nyelv

Tegye az 5-ös regiszter tartalmát a 3-as regiszterbe:
0001 1000 0011 0101 vagy hexadecimálisan : 1835 (IBM370 gépi kód)
LR 3, 5 (IBM assembly nyelven)

Egy sornyi assembly kód normálisan egyetlen gépi nyelvi utasításnak felel meg. Magas szintű programozási nyelv esetén ez nem igaz:

x:= y+z;   (Algol 60)

L 3, Y   tedd az Y tartalmát a 3-as regiszterbe
A 3,Z   add hozzá Z-t a 3-as regiszter tartalmához
ST 3, X   tárold a 3-as regiszter tartalmát X-ben

Egy tipikus magas szintű nyelven (Pascal, C) írt utasítás megfelel körülbelül 10 assembly utasításnak.

A programozási nyelvek komponensei:
adattípusok, objektumok és értékek a rajtuk definiált operációkkal
szabályok, melyek rögzítik a specifikált operációk közötti sorrendi összefüggéseket
szabályok, melyek rögzítik a program statikus szerkezetét
forrásnyelv: amit a fordító inputként elfogad (általában magas szintű)
tárgynyelv : amire fordít a fordító (általában alacsony szintű, néha gépi nyelv)
forráskód: amit le kell fordítani
tárgykód: a fordítás eredménye

A fordítóprogram környezete

A fordítóprogram működési sémaja

$$\begin{tikzcd} & & & \boxed{\text{Adatok}} \arrow[d] & \\ \boxed{\text{Forrásnyelvű program}} \arrow[r] & \boxed{\text{Compiler}} \arrow[r] & \boxed{\text{Tárgynyelvű program}} \arrow[r] & \boxed{\text{Tárgyprogram végrehajtása}} \arrow[r] & \boxed{\text{Eredmények}} \end{tikzcd}$$

↑ Fordítási idő

↑ Futási idő

A fordítási és a futási idő jól elkülönül

Matematikailag: $Q=T(P)$

ahol P - forrásnyelvű program

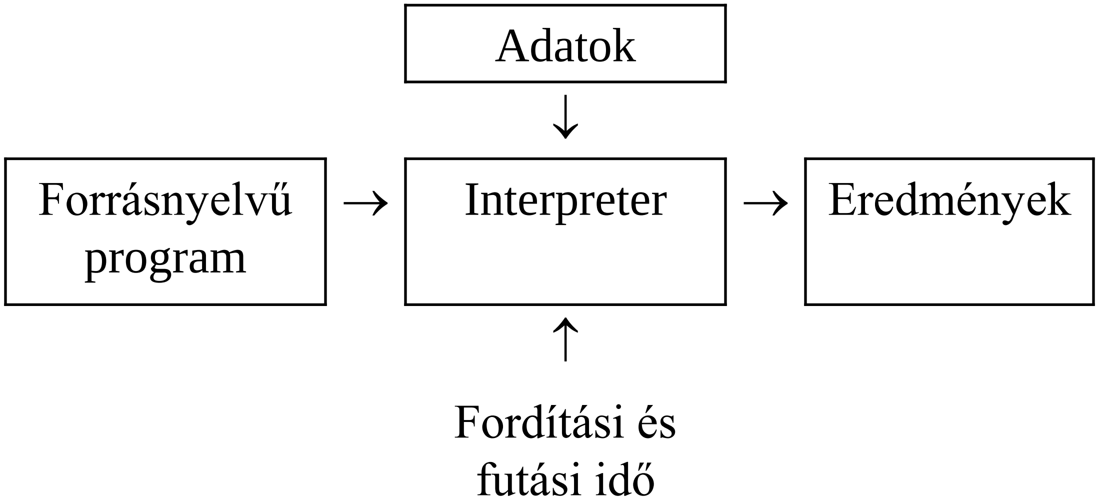
Q - tárgynyelvű program

T - fordítás (Transzláció, transzformáció)

Ha többmenetes a fordító, akkor: $P_{n-1}=T_n(P_n)$, $P_{n-2}=T_{n-1}(P_{n-1})$, … , $P_1=T_2(P_2)$, $Q=T_n(P_1)$

ahol P_i - közbülső programforma

Az interpreter működési sémája



A fordítási és a futási idő nem válik szét
A hardware által vezérelt („bedrótozott”) interpreter neve formulavezérelt számítógép

interpreter: a program előzetes fordítás nélkül hajtódik végre (az aktuális utasítást tárgykód szerkesztése nélkül értelmezi, s utána rögtön végrehajtja). Pld ciklus utasítás esetén lassúbb futást eredményezhet:

110   for I=1 to 100
120 X(I)=Y(I)+X(I)   100 –szor: értelmezi (majd végrehajtja)
130 next I

Interpreter és compiler

(„tiszta”) interpreter: A program előzetes fordítás nélkül hajtódik végre. Minden forrásnyelvi utasítás karakter formáját minden végrehajtás előtt analizálja (valahányszor az végrehajtódik). „okosabb” változatok: csak egyszer analizál minden előforduló forrásnyelvi utasítást, s a már végrehajtott utasításokból fokozatosan felépíti a tárgykódot.

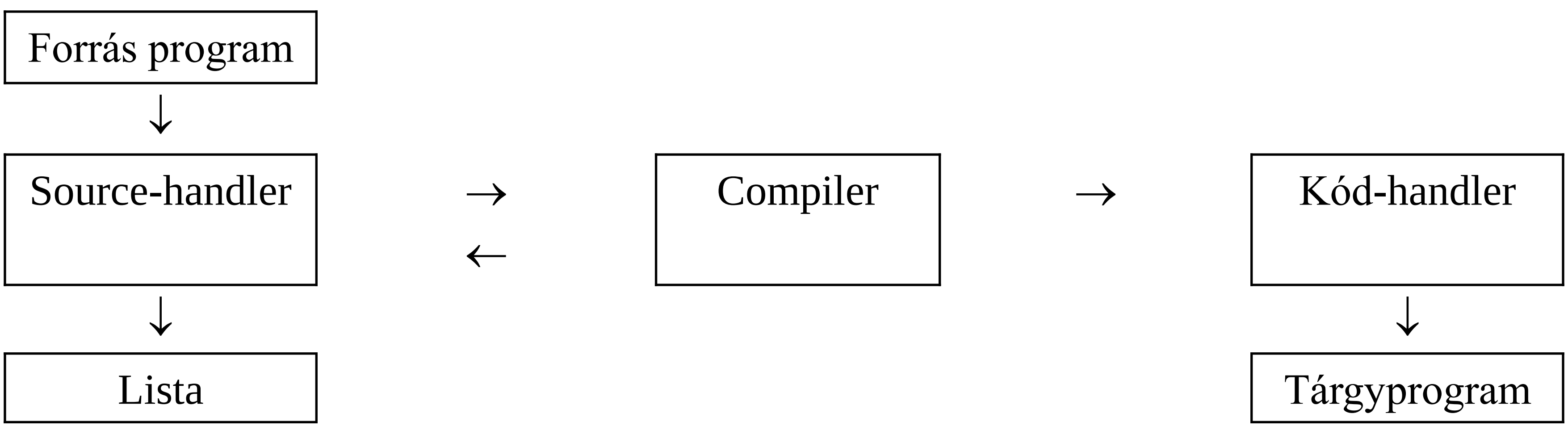
Az interpreter (vagy értelmező) olyan program (ritkábban beépített hardver), ami képes arra, hogy az általa felismert nyelven megfogalmazott utasításokat bemenő adatként kezelje, és a futtató gép saját utasításkészletének megfelelő utasítások sorozatává alakítsa át, majd ezeket az utasítássorozatokat azonnal futtassa is.

Míg egy fordítóprogram a forrásprogramokat utasításonként a futtató gép által végrehajtható (gépi kódú) utasítások sorozatává alakítja át – fordítja – azaz a forrásprogramból a futtatásra kész forma teljes egészében előáll, addig az értelmező a forrásprogramot utasításról utasításra haladva anélkül is végrehajthatja – azonnal – hogy a teljes forrásprogramot beolvasná, s a „klasszikus” típusa az értelmezés során tárgyprogramot nem készít.

A fordítóprogram szerkezete

A forrásprogram általában egy file-ban van.
Compiler(forrásnyelvű program)(tárgyprogram, lista)
A fordítás lépései:

1. Source-handler(forrásnyelvű program, hibák)(karaktersorozat, lista)
   - Input-handler(forrásnyelvű program)(karaktrersorozat)
   - Output-handler(forrásnyelvű program, hibák)(lista)
2. Compiler(karaktersorozat)(tárgykód, hibák)
3. Kód-handler(tárgykód)(tárgyprogram)



A fordítóprogramok felépítése

A fordítóprogramokat különböző fázisokra szokás bontani. Ezek a fázisok általában szekvenciálisan követik egymást, és egy fázis kimenete bemenete lesz az őt követőnek. Bevett gyakorlat, hogy minden fázist különálló modulként implementálnak. Az is megszokott, hogy bizonyos modulokat több fordítóprogram közösen használ. A fázis és a menet különböző fogalmakat jelölnek, ahogy ez majd a későbbiekben ki is fog derülni. Egy menet több fázisból állhat, illetve egy fázis több menetből állhat.

A fordítóprogram fázisai

A fordítóprogramok egyik lehetséges fázisokra bontása az alábbi

1. **Analízis** A forrásprogram (és a közbenső programformák) elemzése különböző szempontok szerint. Az egyes alfázisai mindig ”‘erősebb”’ kimenetet szolgáltatnak, mint a bemenetük volt, azaz például a szintaktikai elemző feltételezheti, hogy a forrásprogram lexikálisan helyes. Az analízist általában a frontend-ben valósítják meg.

1.1 **Strukturális elemzés** A forráskód struktúráját elemzi.
1.1.1 Lexikális elemzés Ez a fázis a forráskódot (vagy a forráskód valamilyen source handlerrel átalakított változatát) elemzi. Próbálja megtalálni benne azokat az összetartozó karaktersorozatokat, amelyek a nyelv egy szimbólumát alkotják (pl. változók, konstansok, kulcsszavak stb.). Ehhez reguláris kifejezéseket használ. A fehérkarakterek és a kommentek általában ignorálva lesznek, mivel ezeknek nincs jelentőségük a programfordítással kapcsolatban. A lexikális elemző minden szimbólumhoz egy előre megadott kódot rendel, valamint egy szimbólumtáblában kiegészítő információt tárol róluk. A lexikális elemző kimenete az így előállt tokensorozat (a kódok sorozata), a szimbólumtábla és egy hibalista, ami az elemzés során bekövetkezett hibákat tartalmazza.
1.1.2 Szintaktikai elemzés Ez a fázis a lexikális elemző kimeneteként előállt tokensorozaton dolgozik, és egy fastruktúrába szervezi (ha tudja). Ez a szintaxisfa. A szintaxis környezetfüggetlen
nyelvtannal írható le, az elemző ez alapján dolgozik. A szintaktikai elemzés kimenete a szintaxisfa és egy hibalista, ami az elemzés során előállt hibákat tartalmazza.
1.2 Szemantikus elemzés Ebben a fázisban a forráskód statikus szemantikájának elemzése történik. A statikus szemantika a nyelv azon része, ami már nemírható le környezetfüggetlen grammatikával. Tipikusan ide tartozik a típusellenőrzés.
2. Szintézis Ebben a fázisban történik a kód előállítása. általában a backend-ben valósítják meg, kivéve a közbensőkód generálását.

2.1 Közbensőkód generálás Ebben a fázisban történik a tárgyprogram előállítása.
2.1.1 Tárgyleképezés Minden objektum méretének és (relatív) címének meghatározása (memory mapping), ugrás értékének kiszámítása (a címke címe ahova ugrik), regiszter allokálás, algebrai azonosságok (x+y=y+x, -(-x)=x, stb).
2.1.2 Kódszelektálás Amennyiben a fordítóprogramot több platform esetén is használjuk (például 32 bites vagy 64 bites architektúra, vagy például különféle processzor típusok esetén), a változatnak megfelelő kód kiválasztása. Egy adott számítógéptípus különféle konfogurációiban vagy egy számítógép másik gépen történő szimulálásakor merül fel.
- regiszterek végső kijelölése
- mely utasításokat kell aktuálisan implementálni

2.1.3 Gépfüggetlen optimalizálás Olyan optimalizációkat hajt végre, amik függeltenek a paltformtól. Ilyen például a kódkiemelés
2.2 összeszerelés A program összeállítása, a különbözőtárgymodulok beszerkesztése.
2.2.1 Belsőcím felbontás. Címke értékének meghatározása, hossz függő utasítások, speciális problémák
2.2.2 Külső cím felbontás. Kereszthivatkozás, könyvtári keresés.
2.2.3 Utasítás bekódolás. Célkód, a bekódolási eljárás.
2.2.4 Gépfüggő optimalizálás. Olyan optimalizációk, amelyek egy bizonyos processzor sajátságait használják ki.

Az analízis feladata

A compiler analizálja a kapott karaktersorozatot és szintetizálva építi fel a tárgykódot.

1. Lexikális elemzés: az input karaktersorozatban a szimbolikus egységek meghatározása (konstansok, változók, kulcsszavak, operátorok felismerése, szóközök, kommentárok kiszűrése). A szimbólumok általában kódoltak (típuskód, cím a szimbólumtáblába)
   Lexikális elemző(karaktersorozat)(szimbólumsorozat, lexikális hibák)
2. Szintaktikus elemző(szimbólumsorozat)(szintaktikusan elemzett program, szintaktikus hibák)
   A programstruktúra felismerése
   Szintaxisfa kialakítása
   Lengyel formák
3. Szemantikus elemző(szintaktikusan elemzett program)(analizált program, szemantikai hibák)
   Szemantikai tulajdonságok vizsgálata (pl A+B esetén deklarált-e A és B, azonosak-e a típusok, van-e értékük)

A szintézis

1. Kódgenerátor(Analizált program)(tárgykód)
   Szintaxisfából, lengyel formából kód generálása
   Gépfüggő, operációs rendszerfüggő assembly vagy gépi kód kimenet
2. Kódoptimalizáló(tárgykód)(tárgykód)
   Azonos programrészek kiemelése alprogramba
   Hurkok ciklusváltozótól független részeinek hurkon kívüli elhelyezése
   Optimális regiszterhasználat

Menetek

Menet: a forrásprogram valamely reprezentációjának (melyet egy fájlt tartalmaz) elejétől a végéig történő feldolgozása és egy másik reprezentáció elkészítése (újabb fájlban). Több fázs i alkothat egy menetet, s több menet alkothat egy fázist.

Minden fordítóprogram hasonlít abban, hogy végrehajtja a fent leírt fázisokat, ráadásul ugyanebben a sorrendben. Amiben különböznek, az az, hogy egyszerre hajtják-e végre őket, vagy több menetben Az előbbi esetben egymenetes, az utóbbiban többmenetes fordítóprogramról beszélünk.

1. Egymenetes fordítás

Az egymenetes fordítás esetén minden fázis végrehajtódik, és csak utána van a fordítórogramnak outputja. Minden fázis egyszerre fut le, megszakítás nélkül.

Az egymenetes fordítás hátránya, hogy mindennek előre definiáltnak kell lennie, azaz programozási nyelvi korlátok szükségesek (mindennek deklaráltnak kell lennie a használat előtt. (pld Pascal)

További hátránya, hogy nincs lehetőség optimalizációra.

Előnye, hogy kisebb a helyigénye a többmenetes fordításénál, mivel nincs szükség közbenső formák tárolására. Ide szokták sorolni azt is, hogy gyorsabb, mint a többmenetes, de vannak, akik ezzel nem értenek egyet.

forrásnyelvű program → Lexikális elemzés / Szintaktikai és szemantikai elemzés / kódgenerálás → tárgynyelvű program

2. **Többmenetes fordítás**

Többmenetes fordításról beszélünk, ha a fázisok több különböző menetben futnak le. Ilyenkor szükség van közbenső programformák tárolására. Ezek az egyes menetek outputjai, és más menetek inputjai. Fontos megjegyezni, hogy nem csak az fordulhat elő, hogy egy menetben több fázis fut le, hanem az is, hogy egy fázis több menetre van osztva. Akárhogy is, a fázisoknak olyan sorrendben kell egymást követniük, ahogy fent szerepelnek. Néhány esetben az egyetlen lehetőség a többmenetes fordítás, mert a nyelv annyira komplex, hogy ezt megköveteli (ilyen nyelv az ALGOL). Többmenetes fordítást kell alkalmazni akkor is, ha a memória korlátozott mennyiségben áll rendelkezésre, mivel ez a módszer kevesebb memóriát követel, mint az egymenetes. A többmenetes fordítás esetén lehetőség van optimalizálásra, viszont hosszabb ideig tart, és nagyobb a helyigénye.

Kétmenetes fordítás (a legjellemzőbb többmenetes fordítás)

Forrásnyelvű program

Első menet

Lexikális elemzés
Szintaktikai és szemantikai elemzés

Közbülső program

Második menet

Kódszelektálás és kódgenerálás

Tárgynyelvű program

Postdefinit címkék kezelése egymenetes fordítás esetén. Mivel a program elején minden címke deklarálva van, a fordítóprogram el tudja készíteni a következő szerkezetű címke táblázatot:

címke neve, címke rövidített neve, jmp (feltétlen ugrás kódja) 00000000

Ha (az egy menetes) fordítás során egy olyan utasításhoz érünk, ami egy címkére hivatkozik, kikeressük a címke táblázatban a hozzá tartozó jmp 00000000 táblaelem címét, s a címke helyett ezen táblaelem címét írjuk be. Ha az (egy menetes) fordítás során egy címkéhez érünk, kikeressük a címke táblázatból a neki megfelelő jmp (feltétlen ugrás kódja) 00000000 táblaelemet, s a 00000000 helyébe beírjuk a címke címét. Ily módon tehát a lefordított programban egy címkére történő (feltételes vagy feltétlen) ugrás úgy történik, hogy először a neki megfelelő címke táblázatbeli " jmp címkecím" utasítás címére ugrunk, majd onnan " jmp címkecím" feltételn ugrás végrehajtásával a kérdéses címke címére ugrunk.
(Az egymenetes fordítás "ára"tehát felesleges ugró utasítások beépítése lesz.)

Front-end és back-end

Front-end fázisai (a fordítóporgram gépfüggetlen része): lexikális elemző, szintaktikus elemző, szemantikus elemző, közbülső kód előállítása (esetleg bizonyos gépfüggetlen optimalizálással)

Back-end fázisai (a fordítóprogram gépfüggő része): géptől függő optimalizálás

A fordítóprogram táblázatai: felépítése és használata a **táblázatkezelő** feladata (a táblázatkezelő vagy része a lexikális elemzőnek, van attól elkülönülve önálló egységként működik).

A forrásprogram adatai : azonosító nevek, konstansok, címkék

azonosító nevek : egyszerű változók, struktúrált változók és tömbök, függvény és eljárásnevek
azonosító nevek táblázata: név, tulajdonság, cím

név:
- *teljes név*
- *rövidített név* (a közbülső programformák rövidített neveket használnak, így könnyebben kezelhető a forrásprogram)

tulajdonság:
- *típus* (egész, valós, logikai, karakter, felsorolás, halmaz, stb.)
- szerkezet (egyszerű változó, tömörített változó, karakterlánc, tömb, struktúrált változó, rekord, mutató)
- *egyéb* (standard függvénynév-e, függvényeljárás-e, formális paraméter –e, a nevet a programozó expliciten deklarálta-e, utasításfüggvény neve-e, entry név-e, külső név-e, más névvel azonosított név-e, az azonosító szerepel-e közös adatmezőben, dimenziószám (skalárnál nulla), dinamikus indexhatárú tömb-e)

cím: a névhez tartozó (bázisrelatív) cím értéke

konstansok táblázata:
konstans forrásprogrambeli alakja
típus (meghatározza azt is, hogy hány bájt kell az ábrázoláshoz)
konstans átkonvertált értéke
(bázisrelatív) cím

címkék táblázata

címke neve
típus (predefinit, postdefinit)
(bázisrelatív) cím
Postdefinit címke esetén: a címek, ahova be kell írni a címke címét

Token: szintaktikai és szemantikai szempontból azonosan kezelt fogalom, amit a nyelvtanban egy szimbólum jelöl.
Egy tokenhez tartozó mintázat (minta): olyan karaktersorozatok, amik egyazon tokent eredményeznek.
Lexéma: egy minta konkrét előfordulása a szövegben
Például:
Token : operátor
Mintázat: +, -, *, /, ↑, XOR, stb
Lexéma: + egy konkrét előfordulása egy szövegben (Pld x2:= (x13+3335)*(y+z) szövegben lexémák:
x2, := , (, x13, +, 3335,), -, (, y, +, z,)
Ha egy tokenhez több minta is tartozik, akkor a tokenhez attribútumokat rendelünk.

Lexémák felismerése:
azonosító eleje: betű; azonosító vége: nem betű és nem számjegy (a következő lexéma első karaktere)
konstans eleje: számjegy; konstans vége: nem számjegy (a következő lexéma első karaktere)
{} kommentár eleje : { , {} kommentár vége: }
(**) kommentár eleje: (* , (**) kommentár vége : *)
értékadás eleje: : , értékadás vége: =
<= eleje: <, <= vége: =
<> eleje: <, <> vége: >
>= eleje: > , >= vége: =
Tokenizálás: a folyamat, melynek során egy karakterfüzért tokenekre osztanak, a lexikális analízis céljából

Példa

DÖMÖL -1 (Dömösi Language) szintaxis

lexémái /tokenei

<azonosító> : : = <betű> | <azonosító><betű>|<azonosító><számjegy>

<konstans> ::= <számjegy>| <konstans><számjegy>

<kommentár-1>::= {<szöveg-1>}

< kommentár-2 >::= (*<szöveg-3>*)

< **:=** > ::= :=

< **<=** > ::= <=

< **<>** > ::= <>

< **>=** > ::= >=

Nem lexémák/tokenek:

<program>::= $ | <utasítás sorozat>$

<utasítás sorozat>::=<lexéma>|<utasítássorozat> <lexéma>

<betű>::= a|…|z|A|…|Z

<számjegy>::= 0|…|9

<szöveg-1 elem>::=<betű>|{|(|*|)| : | =|<|>|szóköz

<szöveg-1>::=<szöveg-1 elem>|<szöveg-1><szöveg-1 elem> (kapcsos végzárójel nem)

<szöveg-2 elem>::=<szöveg-1 elem>| }

<szöveg-2>::= <szöveg_2 elem> | <szöveg-2 ><szöveg-2 elem>

<szöveg-3>= <szöveg-2> \ [<szöveg-2>*) | <szöveg-2> *) <szöveg-2>]

Lexikális elemzés

A lexikális elemzés feladata, hogy a forrásprogramban felismerje az összetartozó szimbólumokat. Ezeket az összetartozószimbólumokat lexémáknak nevezzük. A lexéma a forrásprogram legkisebb, jelentéssel bíró egysége. Lexémák a következők:

- kulcsszavak
- azonosítók
- operátorok
- elhatároló jelek
- zárójelek
- konstansok

Az elemző a forrásprogramban megkeresi a lexémákat, és minden lexémához egy előre definiált kódot rendel. A kimenete ezeknek a kódoknak a sorozata (tokensorozat). Ez már nem értelmezhető emberek számára. A szimbólumokat reguláris kifejezésekkel vagy determinisztikus véges automatákkal írhatjuk le. Ez a legfontosabb oka annak, hogy a lexikális elemzőt különválasztjuk a szintaktikai elemzőtől (hiszen a része is lehetne): ha nem választanánk külön, akkor a szimbólumokat is környezetfüggetlen grammatikával kellene leírnunk, márpedig a reguláris kifejezések kezelése sokkal egyszerűbb. A lexikális elemzők létrehozásához először a szimbolikus egységeket leírjuk reguláris kifejezésekkel, majd megkonstruáljuk az ekvivalens determinisztikus véges automatát.

Ezután elkészítjük az automata implementációját. Az implementáció könnyen megoldható feltételes elágaztató utasítással, aminek az egyes ágaiban a feltételek az automata állapotait reprezentálják, de használhatunk keresőtáblázatot is. A véges automata generálására és vázának implementálására léteznek automatikus eszközök. A reguláris nyelvtanhoz vele ekvivalens nemdeterminisztikus véges automatát konstruálhatunk Például Thompson algoritmusával. Mivel nekünk determinisztikus automatára van szükségünk, ezért még egy átalakítást végre kell hajtanunk (a lexikális elemzéshez megfelelne egy nemdeterminisztikus véges állapotú automata is, viszont akkor egy bonyolultabb és kevésbé hatékony, úgynevezett visszalépéses algoritmust kellene alkalmaznunk). Az automata végállapotaihoz különböző szimbólum feldolgozólépéseket is rendelhetünk. Például fehérkarakterek és kommentek eltüntetése.

Lexikális elemzés – lexémák elkülönítése

kulcsszavak
azonosítók
operátorok

Véges automatával történik.

Példa

input: l,d,sp,{,},(,),∗,:,=,<,>,p.
állapotok:

1. kezdőállapot
2. azonosítóban
3. azonosító vége
4. számban (szám belsejében)
5. szám vége
6. {...} kommentár
7. {...} kommentár vége
8. nyitó zárójelet talált
9. (*...*) kommentár
10. *-ot talált (*...*) típusú kommentárban
11. (*...*) vége
12. :-t talált
13. token :=
14. <-t talált
15. token <=
16. token <>
17. >-t talált
18. >=-t talált
19. általános pontoıítás (hibakezelés)
20. általános pontosítás (továbbfejlesztendő blokkok

Az állapotátmenet-függvény táblázatos reprezentációja. Ha az i-edik sor b címkéjű eleme j, akkor ezt úgy kell értelmezni, hogy ha az automata az i-edik állapotban van és b-t olvassa, akkor a j-edik állapotba megy át.

állapot/bemenő jel	betű	szám-jegy	{	}	(	*	)	:	=	<	>	szó-köz	egyéb	$	backup (visszalépés)	inputot olvas
1 kezdőállapot	2	4	6	19	8	19	19	12	19	14	17	1	19	21	nem	igen
2 azonosítóban	2	2	3	3	3	3	3	3	3	3	3	3	3	3	-	igen
3 azonosító vége	1	1	1	1	1	1	1	1	1	1	1	1	1	1	igen	nem
4 számban	5	4	5	5	5	5	5	5	5	5	5	5	5	5	-	igen
5 szám vége	1	1	1	1	1	1	1	1	1	1	1	1	1	1	igen	nem
6{} komm.-ban	6	6	6	7	6	6	6	6	6	6	6	6	6	19	-	igen
7{} komm. vég	1	1	1	1	1	1	1	1	1	1	1	1	1	1	nem	nem
8 (-t talált	20	20	20	20	20	9	20	20	20	20	20	20	20	19	-	igen
9 (**) komm.ban	9	9	9	9	9	10	9	9	9	9	9	9	9	19	-	igen
10 * (**)-ban	9	9	9	9	9	10	11	9	9	9	9	9	9	19	-	igen
11(**)komm.vég	1	1	1	1	1	1	1	1	1	1	1	1	1	1	nem	nem
12 : -t talált	20	20	20	20	20	20	20	20	13	20	20	20	20	19	-	igen
13 := token	1	1	1	1	1	1	1	1	1	1	1	1	1	1	nem	nem
14 < -t talált	20	20	20	20	20	20	20	20	15	20	16	20	20	19	-	igen
15 <= token	1	1	1	1	1	1	1	1	1	1	1	1	1	1	nem	nem
16 <> token	1	1	1	1	1	1	1	1	1	1	1	1	1	1	nem	nem
17 >-t talált	20	20	20	20	20	20	20	20	18	20	20	20	20	19	-	igen
18 >= token	1	1	1	1	1	1	1	1	1	1	1	1	1	1	nem	nem
19 hibakezelő	1	1	1	1	1	1	1	1	1	1	1	1	1	1	nem	nem
20 továbbfejl.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	igen	nem

Baloldali levezetéssé alakítás

Definíció: Tetszőleges $G = (V_N , V_T , S, H)$ környezetfüggetlen nyelvtan esetén egy $(S=)\ P_0 \Rightarrow P_1 \Rightarrow \ldots \Rightarrow P_i \Rightarrow P_{i+1} \Rightarrow \ldots \Rightarrow P_n$ $(=P, P \in V_T^{\ *})$ valódi levezetést (valódi) baloldali levezetésnek hívunk, ha minden $i = 0,\ldots, n-1$ esetén $P_i = Q_i A R_i$, $Q_i \in V_T^{\ *}$, $A \in V_N$, $R_i \in (V_N \cup V_T)^*$, $P_{i+1} = Q_i X R_i$, $X \in (V_N \cup V_T)^*$, és $A \rightarrow X \in H$, azaz egy baloldali levezetésben mindig a legbaloldalibb nemterminálisra alkalmazunk egy helyettesítési szabályt.

Definíció: Tetszőleges $G = (V_N , V_T , S, H)$ környezetfüggetlen nyelvtan esetén egy $(S=)\ P_0 \Rightarrow P_1 \Rightarrow \ldots \Rightarrow P_i \Rightarrow P_{i+1} \Rightarrow \ldots \Rightarrow P_n$ $(=P, P \in V_T^{\ *})$ valódi levezetést (valódi) jobboldali levezetésnek hívunk, ha minden $i = 0,\ldots, n-1$ esetén $P_i = Q_i A R_i$, $R_i \in V_T^{\ *}$, $A \in V_N$, $Q_i \in (V_N \cup V_T)^*$, $P_{i+1} = Q_i X R_i$, $X \in (V_N \cup V_T)^*$, és $A \rightarrow X \in H$, azaz egy jobboldali levezetésben mindig a legjobboldalibb nemterminálisra alkalmazunk egy helyettesítési szabályt.

Tétel: Bármely $G = (V_N, V_T, S, H)$ környezetfüggetlen nyelvtan és $P \in L(G)$ esetén P generálható G –beli baloldali levezetéssel.

Bizonyítás: Legyen $P \in L(G)$. Ekkor létezik egy G -beli

- $(S=)\ P_0 \Rightarrow P_1 \Rightarrow \ldots \Rightarrow P_i \Rightarrow P_{i+1} \Rightarrow \ldots \Rightarrow P_n\ (=P \in V_T^{\ *})$,

levezetés. Tegyük fel, hogy a levezetésünk nem bal oldali. Ekkor $\exists\ i < n$, hogy

$S \Rightarrow^* P_i \Rightarrow P_{i+1} \Rightarrow^* P$, $P_i = Q_i A R_i$, $A \rightarrow X \in H$, $X \in (V_N \cup V_T)^*$, $P_{i+1} = Q_i X R_i$,

$Q_i \notin V_T^{\ *}$.

Ekkor $Q_i = T_i B U_i$ $T_i \in V_T^{\ *}$, $B \in V_N$, $U_i \in (V_N \cup V_T)^*$. Legyen i a legkisebb ilyen index. A levezetésben később biztosan lesz egy olyan $P_j \Rightarrow P_{j+1}$ $(j > i)$ lépés, amikor egy $B \rightarrow Y \in H$ szabályt alkalmazunk ezen B-re. Cseréljük meg a $P_i \Rightarrow P_{i+1}$ lépésben alkalmazott $A \rightarrow X \in H$ helyettesítési szabály és ezen

$P_j \Rightarrow P_{j+1}$ lépésben alkalmazott $B \rightarrow Y \in H$ szabály alkalmazási sorrendjét. Ezzel vagy egy baloldali levezetést kapunk, vagy egy olyan levezetést, melyben az eredeti levezetésben szereplő, fenti tulajdonságú i index legalább eggyel nő. Gondolatmenetünk véges sokszori alkalmazásával a $P \in V_T^{\ *}$ egy baloldali levezetését kapjuk. Ezzel a tétel bizonyítást nyert.

Hasonló módon igazolható az alábbi tétel:

Tétel: Bármely $G = (V_N, V_T, S, H)$ környezetfüggetlen nyelvtan és $P \in L(G)$ esetén P generálható G –beli jobboldali levezetéssel.

Lambdamentessé tétel mint az üres szó lemmánál.

Legyen $G = (V_N , V_T , S, H)$ környezetfüggetlen nyelvtan .

Határozzuk meg a nemterminálisokból álló U halmazt, melynek elemeiből λ levezethető! Ehhez egy halmaz sorozatot fogunk konstruálni, ahol Ui elemei azok a nemterminálisok lesznek, melyekből maximum i lépésben levezethető λ.

Definíció szerint $U_0 = Ø$ s így $(U_0)^* = \{\lambda\}$. Formálisan:

$$U_{i+1} = U_i \cup \{A \mid A \rightarrow \alpha \in H, \alpha \in (U_i)^* \}$$

Ha $U_i = U_{i+1}$, akkor az algoritmus véget ér, és $U = U_{i+1}$. Ha a mondatszimbólum eleme az U halmaznak, akkor felveszünk egy új S' mondatszimbólumot, s az új szabályok közzé bekerülnek az $S' \rightarrow \lambda$ és $S' \rightarrow S$ szabályok, ellenkező esetben nem vesznek részt a nyelvtanban. Az összes lehetséges módon elhagyjuk a régi szabályok jobboldalaiból U elemeit, s bekerülnek ezek a szabályok is az új szabályok közzé mindazok kivételével, ahol jobboldalon λ található.

Normális alakúra hozás

Definíció: Akkor mondjuk, hogy a $G = (V_N, V_T, S, H)$ környezetfüggetlen nyelvtan normális alakú, ha a H-beli szabályokban terminális betű csak $A \rightarrow a \in H$ ($A \in V_N$, $a \in V_T$) alakú szabályokban (azaz ezen szabályok jobboldalán) fordulnak elő.

Tétel: Minden környezetfüggetlen nyelvtanhoz létezik vele ekvivalens normális alakú környezetfüggetlen nyelvtan.

Bizonyítás: Legyen $G = (V_N, V_T, S, H)$ környezetfüggetlen nyelvtan, s minden $a \in V_T$ esetén vezessünk be egy X_a új nemterminálist (mely nem eleme $V_N \cup V_T$ -nek). Vezessük be továbbá minden $a \in V_N \cup V_T$ -re az $f(a)$ jelölést, ahol is $f(a) = a$, ha $a \in V_N$, továbbá $f(a) = X_a$, ha $a \in V_T$. Végül, legyen $f(\lambda) = \lambda$, továbbá tetszőleges $x_1,\ldots,x_k \in V_N \cup V_T$ esetén $f(x_1\ldots x_k)= f(x1)\ldots f(x_k)$. Könnyen látható, hogy az alábbi G' nyelvtan normális alakú és ekvivalens G-vel: $G'= (V'_N, V_T, S, H')$, ahol $V'_N = V_N \cup \{X_a \mid a \in V_T\}$, továbbá
$H'= \{X \rightarrow f(W) \mid X \rightarrow W \in H,\ W \notin V_T\} \cup \{X \rightarrow a \in H \mid a \in V_T\} \cup \{X_a \rightarrow a \mid a \in V_T\})$.

Például: $H = \{D \rightarrow d, P \rightarrow DöMöSi\}$ esetén új nemterminálisok: $X_ö$, X_i, új szabályok:

$H'=\{ P \rightarrow D X_ö M X_ö S X_i , D \rightarrow d, X_ö \rightarrow ö, X_i \rightarrow i\}$

(a $D \rightarrow d$ szabály marad, új szabályok $P \rightarrow D X_ö M X_ö S X_i$, $X_ö \rightarrow ö$, $X_i \rightarrow I$, továbbá a $P \rightarrow DöMöSi$ szabály elmarad).

Balrekurzió kiküszöbölése

Definíció: Akkor mondjuk, hogy a $G = (V_N , V_T , S, H)$ környezetfüggetlen nyelvtan balrekurzió mentes, ha tetszőleges $A \in V_N$ esetén $A \Rightarrow^* BX$, $B \in V_N$, $X \in (V_N \cup V_T)^*$ Esetén $A \neq B$.

Tétel: Minden környezetfüggetlen nyelvtanhoz létezik vele ekvivalens balrekurzió mentes nyelvtan.

Bizonyítás: Legyen $G = (V_N , V_T , S, H)$ környezetfüggetlen nyelvtan, s legyen $V_N=\{A_1,\ldots, A_n\}$ rendezett halmaz, ahol $A_1<\ldots< A_n$ a tekintett rendezésre nézve. (Ez a feltevés nem jelent megszorítást, hisz egy teljesen tetszőleges sorrendet rögzítünk.) Tegyük fel, hogy G normális alakú (ha nem akkor hozzuk normális alakúra), azaz terminális szimbólum csak $A \rightarrow a$ alakú szabályban fordul elő, ahol természetesen A nemterminális, a pedig terminális betű.

Legyen k maximális természetes szám, melyre $1 \le i < k$ esetén minden $A_i \rightarrow A_jX$ alakra $i < j$. Ilyen k van, hisz ha egy $A_i \rightarrow A_jX$ alakra sem teljesül $i < j$, akkor a $k=1$ választás megfelelő.

1.Először, ha létezik olyan $A_k \rightarrow A_jX$ szabály, melyre $k>j$, akkor minden egyes $A_j \rightarrow Y_h$ alakú szabály esetén, ahol $Y_h = A_mY'_h$ és $j < m$ (ha vannak ilyenek, akkor) vezessük be az új $A_k \rightarrow Y_hX$ alakú szabályokat és hagyjuk el a régi $A_k \rightarrow A_jX$ szabályt. Ekkor az új szabályok $A_k \rightarrow A_mY'_h X$ alakúak, ahol $k > j$ és $\mathbf{j < m}$. Ha valamely új $A_k \rightarrow A_mY'_h X$ szabályra $k > m$, akkor ismételjük meg ezt a gondolatmenetet (az $A_k \rightarrow A_jX$ helyett) az $A_k \rightarrow A_mY'_h X$ -re. Véges lépésben olyan új $A_k \rightarrow A_tY''_h$ alakú szabályokhoz jutunk, ahol $k \le t$.

Véges lépésben: minden H- beli $A_k \rightarrow A_jX$ szabály esetén $k \le j$.

2. Ez után ha létezik H-beli $A_k \rightarrow A_kX$ alakú szabály, akkor az összes

$A_k \rightarrow A_kX_1$, …, $A_k \rightarrow A_kX_r$

$A_k \rightarrow Y_1$, …, $A_k \rightarrow Y_s$ $(Y_i = A_uY'_i , u >k)$

szabály helyett vegyük az

$A_k \rightarrow Y_mZ_k$, $m=1,\ldots,s$ valamint a

$Z_k \rightarrow X_t$ és $Z_k \rightarrow X_tZ_k$, $t=1,\ldots, r$

szabályokat.

Ezt az eljárást véges sokszor megismételve olyan nyelvtanhoz jutunk, mely amellett, hogy ekvivalens az eredeti nyelvtannal, minden $A_k \to A_j X$ alakú szabályára $k < j$. Növeljük k értékét 1-el, mindaddig folytassuk a fenti eljárást, míg $k = n+1$ –t el nem érjük. Ezzel a bizonyítást befejeztük.

Magyarázat:

$A_k \to A_k X_1$, …, $A_k \to A_k X_r$

$A_k \to Y_1$, …, $A_k \to Y_s$ $(Y_i = A_u Y'_i$, $u > k)$

szabályok helyett vegyük az

$A_k \to Y_m Z_k$, $m=1,\dots,s$ valamint a

$Z_k \to X_t$ és $Z_k \to X_t Z_k$, $t=1,\dots, r$

szabályokat.

Előbb-utóbb "lecseréljük" a sztring elejéről az A_k –t:

$A_k \Rightarrow A_k X_{i1} \Rightarrow A_k X_{i2} X_{i1} \Rightarrow \dots \Rightarrow A_k X_{i2} \dots X_{i2} X_{i1} \Rightarrow Y_j X_{it} \dots X_{i2} X_{i1}$

Ugyanez elérhető a második szabályhalmazzal:

$A_k \Rightarrow Y_j Z_k \Rightarrow Y_j X_{it} Z_k \Rightarrow \dots \Rightarrow Y_j X_{it} \dots X_{i2} Z_k \Rightarrow Y_j X_{it} \dots X_{i2} X_{i1}$

Láncszabálymentesség (ciklusmentesség): mint a Chomsky-féle normál alaknál. Legyen
$G=(V_N, V_T, S, H)$ egy λ-mentes környezetfüggetlen nyelvtan, s minden $A \in V_N$ esetén képezzük a következő halmazokat:

$U_1(A)=\{ B \in V : A \rightarrow B \in H\}$

$U_{i+1}(A)= U_i(A) \cup \{B\in V : A' \rightarrow B \in H,\ A' \in U_i(A)\}$

$U_1(A)$ valódi részhalmaza $U_2(A)$ -nak, $U_2(A)$ valódi részhalmaza $U_3(A)$ -nak, … ,

Lesz olyan i hogy $U_i(A)= U_{i+1}(A)$. Vezessük be az $U(A)$ jelölést erre az $U_i(A)$ halmazra.

Legyen továbbá minden x terminálisra definíció szerint $U(x)=\{x\}$.

Ekkor $G=(V_N, V_T, S, H')$ az eredeti nyelvtannal ekvivalens nyelvtan lesz, ahol is az új szabályhalmaz ez lesz:

$$H'=\{A \rightarrow X_1\ldots X_k \mid A \rightarrow Y_1\ldots Y_k \in H,\ k>2,\ Y_1 \in U(X_1),\ldots, Y_k \in U(X_k)\} \cup$$
$$\{A \rightarrow x \mid B \rightarrow x,\ B \in U(A),\ x \in V_T\}.$$

Szintaktikai Elemzések:
általános felülről lefelé és alulról felfelé haladó elemzés

Az elemzés alapfeladata: Adjunk olyan algoritmust, amely tetszőleges $G=(V_N, V_T, S, H)$ környezetfüggetlen nyelvtan és $w \in \Sigma^*$ szó esetén eldönti, hogy $w \in L(G)$ teljesül-e!
A felülről-lefelé haladó elemzések (top-down algoritmusok): Az S kezdőszimbólumból kiindulva megpróbálunk felépíteni egy olyan derivációs fát, amelynek a határa w.
Az alulról-felfelé haladó elemzések (bottom-up algoritmusok): A w-ből kiindulva megpróbálunk felépíteni egy olyan derivációs fát, amelynek a gyökere S és a határa w.

Felülről-lefelé haladó elemzések

Definíció: Alternatívák

Egy adott $A \in V_N$ nemterminális lehetséges behelyettesítési szabályainak a jobbolalai.

$A \rightarrow \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_k$

Definíció: Kiterjesztés

Egy nemterminálisnak valamely alternatívájával való helyettesítése a derivációs fában.

Definíció: Illesztés

Annak ellenőrzése, hogy a kiterjesztésnél alkalmazott alternatívában szereplő terminálisok illeszkednek-e az elemzendő szó megfelelő részéhez.

Definíció: Felülről-lefelé haladó elemzés

Minden nemterminálisra lerögzítjük az alternatíváinak egy sorrendjét. Egy nemterminális kiterjesztése esetén az alternatívákat ebben a lerögzített sorrendben vizsgáljuk meg, hogy alkalmasak-e a kiterjesztésre. Ha nem találunk megfelelő alternatívát akkor egy backtrack-et (egy szinttel feljebb történő visszalépést) hajtunk végre.

Algoritmus inputja:

Egy nem balrekurzív $G=(V_N , V_T , S, H)$ környezetfüggetlen nyelvtan és egy $w=a_1a_2\ldots a_n$, $n \geq 0$ input szó.

A w szót n+1. szimbólumként egy # jel zárja le. A # nem tartozik sem V_N -hez, sem V_T-hez.

Algoritmus outputja:

Igen jelzés, és a w szónak egy baloldali levezetése, ha $w \in L(G)$.

Nem jelzés egyébként.

Módszer:

1. Minden $A \in V_N$ esetén rögzítsük le az A alternatíváit $A \rightarrow \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_k$ alakban. Az A i-dik alternatíváját A_i jelöli. (Implementáláskor az (A, i) párt alkalmazzuk A_i jelölésére.)
2. Az elemzés (s, i, α, β) alakú konfigurációk sorozata.
3. A konfigurációk halmazán megadunk egy $\vdash$ átmeneti relációt. A rákövetkező konfiguráció meghatározása az alábbiakban megadott felsorolásból történik.
4. A kezdő konfiguráció $(q,1, \lambda, S)$.

   A befejező konfiguráció: $(t, n+1,\alpha, \lambda)$

   $w \in L(G)$ akkor és csak akkor, ha $(q,1, \lambda, S) \vdash^* (t, n+1, \alpha, \lambda)$

a konfiguráció

(s, i, α, β) **értelmezése:**

s az elemzés állapota.

q - normál

t -elfogadó

b - backtrack

i pointer az input szóban ($1 \le i \le n+1$)

α jobbvégtetejű verem, az elemzés története backtrack-hez és a baloldali levezetéshez. (Passzív verem)

β balvégtetejű verem, a még levezetendő baloldali mondatforma. (Aktív verem)

átmeneti reláció

1. Kiterjesztés:

$(q, i, \alpha, A\beta) \vdash (q, i, \alpha A_1, \gamma_1\beta)$: az aktív szimbólum (az A) egy nemterminális és γ_1 az első alternatívája

2. Input illesztés sikeres: $a=a_i$ mellett $(q, i, \alpha, a\beta) \vdash (q, i+1, \alpha a, \beta)$: az aktív szimbólum egy olyan terminális, mely pont az i-edik betű

3. Sikeres elemzés

$(q, n+1, \alpha, \lambda) \vdash (t, n+1, \alpha, \lambda)$ elértük a befejező konfigurációt

4. Input illesztés sikertelen: $a \neq a_i$: az aktív szimbólum olyan terminális, mely nem illeszkedik az inputra : $(q, i, \alpha, a\beta) \vdash (b, i, \alpha, a\beta)$

5. Backtrack az inputban: b állapotban a passzív verem tetején terminális van.

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

6. Backtrack a kiterjesztésben $(b, i, \alpha A_j, \gamma_j\beta)$ esetén a $\vdash$ jelet követi

I. A-nak van j+1. alternatívája

$(b, i, \alpha A_j, \gamma_j\beta) \vdash (q, i, \alpha A_{j+1}, \gamma_{j+1}\beta)$ vesszük a következő alternatívát

II. i=1, A=S, és S-nek csak j alternatívája van: nincs átmenet semelyik konfigurációba, az elemzett sztring nem eleme a nyelvnek

III Egyébként $(b, i, \alpha A_j, \gamma_j\beta) \vdash (b, i, \alpha, A\beta)$ nincs több alternatívája A-nak, visszatérünk az előző szintre.

Példa. Legyen $G=(V_N, V_T, S, H)$, ahol $H=\{ S \to T + S , S \to T, T \to a , T \to b\}$.

Feladat: $b+a \in L(G)$?

Alternatívák:

$S \to T + S \mid T$

$T \to a \mid b$

azaz S alternatívái : S1 = T+S, S2 = T , T alternatívái : T1 = a, T2=b.

Levezetés:

$(q, 1, \lambda, S) \vdash (q, 1, S1, T + S)$ (S kiterjesztése)

$\vdash (q, 1, S1T1, a + S)$ (T kiterjesztése)

$\vdash (b, 1, S1T1, a + S)$ (sikertelen input illesztés)

$\vdash (q, 1, S1T2, b + S)$ (backtrack a kiterjesztésben I.: T következő alternatíváját vesszük)

$\vdash (q, 2, S1T2b, +S)$ (sikeres input illeszkedés: az első betű b)

$\vdash (q, 3, S1T2b+, S)$ (sikeres input illeszkedés: a 2. betű +)

$\vdash (q, 3, S1T2b + S1, T + S)$ (S kiterjesztése)

$\vdash (q, 3, S1T2b + S1T1, a + S)$ (T kiterjesztése)

$\vdash (q, 4, S1T2b + S1T1a, +S)$ (sikeres input illeszkedés: a 3. betű a)

$\vdash (b, 4, S1T2b + S1T1a, +S)$ (sikertelen input illesztés)

$\vdash (b, 3, S1T2b + S1T1, a + S)$ (backtrack az inputban)

$\vdash (q, 3, S1T2b + S1T2, b + S)$ (backtrack a kiterjesztésben I.: T következő alternatíváját vesszük)

$\vdash (b, 3, S1T2b + S1, T + S)$ (backtrack a kiterjesztésben III.: visszatérünk az előző szintre)

$\vdash (q, 3, S1T2b + S2, T)$ (backtrack a kiterjesztésben I.: S következő alternatíváját vesszük)

$\vdash (q, 3, S1T2b + S2T1, a)$ (T kiterjesztése)

$\vdash (q, 4, S1T2b + S2T1a, \lambda)$ (sikeres input illesztés: a 3. betű a)

$\vdash (t, 4, S1T2b + S2T1a, \lambda)$. (sikeres elemzés, elértünk egy végkonfigurációt)

Következésképpen b + a ϵ L(G).
Baloldali levezetés (mindig balról az első nemterminálist helyettesítjük):
A baloldali levezetésben egymásután alkalmazandó alternatívák
(az αverem S1T2b + S2T1a tartalma alapján a szereplő terminálisok elhagyásával):
S1T2S2T1

Baloldali levezetés:
(S1) S $\rightarrow$ T + S alkalmazásával: S => T+S, (T2) T $\rightarrow$ b alkalmazásával T+S => b+S,
(S2) S $\rightarrow$ T alkalmazásával : b+S => b+T, (T1) T $\rightarrow$ a alkalmazásával b+T => b+a,

azaz a következő baloldali levezetést kapjuk:

S => T+S => b+S => b+T => b+a.

Alulról felfelé haladó elemzések

Definíció: Redukálás

egy szabály jobboldalát a baloldalával helyettesítjük (míg a felülről lefelé haladó elemzésnél a baloldalt helyettesítettük a jobbal)

Definíció: Léptetés

eggyel tovább haladunk az input szóban, azaz olvasunk belőle egy szimbólumot.

Definíció: Alulról felfelé haladó elemzés

Az elemzés megkezdése előtt megsorszámozzuk a szabályokat. Ebben a sorrendben vizsgáljuk meg, hogy az input szó eddig beolvasott részéből redukálásokkal keletkezett mondatforma valamely szuffixe (végszelete)

alkalmas-e a redukálásra. Ha nem találunk megfelelő alternatívát akkor egy backtrack-et (egy szinttel feljebb történő visszalépést) hajtunk végre.

Algoritmus inputja: Egy nem ciklikus $G=(V_N, V_T, S, H)$ környezetfüggetlen nyelvtan és egy $w=a_1a_2\ldots a_n$, $n \geq 1$ input szó.

A w szót n+1. szimbólumként egy # jel zárja le. A # nem tartozik sem V_N -hez, sem V_T -hez.

Algoritmus outputja: Igen jelzés, és a w szónak egy jobboldali levezetése, ha $w \in L(G)$.

Nem jelzés egyébként.

Módszer:

1. Sorszámozzuk be H elemeit, azaz minden $A \rightarrow \gamma \in H$ esetén rögzítsük le az $A \rightarrow \gamma$ sorszámát.
2. Az elemzés (s, i, α, β) alakú konfigurációk sorozata.
3. A konfigurációk halmazán megadunk egy $\vdash$ átmeneti relációt. A rákövetkező konfiguráció meghatározása az alábbiakban megadott felsorolásból történik.
4. A kezdő konfiguráció $(q,1, \lambda, \lambda)$.

   A befejező konfiguráció: $(t, n+1, S, \beta)$

   $w \in L(G)$ akkor és csak akkor, ha $(q,1, \lambda, \lambda) \vdash^* (t, n+1, S, \beta)$

a konfiguráció

(s, i, α, β) **értelmezése:**

s az elemzés állapota.

q - normál

t -elfogadó

b - backtrack

i pointer az input szóban ($1 \leq i \leq n+1$)

α jobbvégtetejű verem, az input szó eddig beolvasott részéből redukálásokkal keletkezett mondatforma (passzív verem).

β balvégtetejű verem, tartalmazza az elemzés történetét, azaz a redukálások és shiftelések történetét (aktív verem).

átmeneti reláció

1. Redukálás: $(q, i, \alpha\gamma, \beta) \vdash (q, i, \alpha A, j\beta)$ ha az α verem teteje redukálható, ahol is j az első olyan szabály sorszáma, mellyel ez megtehető.

2. Léptetés:

ha már nem lehet redukálni, valamint az elemzett szó i-edik betűje $a=a_i$ továbbá $i< n+1$,

akkor $(q, i, \alpha, \beta) \vdash (q, i+1, \alpha a, s\beta)$, ahol az s szimbólum a léptetés műveletét jelöli

3. Sikeres befejezés:

$(q, n+1, S, \beta) \vdash (t, n+1, S, \beta)$, az algoritmus leáll “igen” jelzéssel

4. Átmenet backtrack állapotba: $i = n+1$ és $\alpha \neq S$ esetén

$(q, n+1, \alpha, \beta) \vdash (b, n+1, \alpha, \beta)$

5. Visszalépés végrehajtása (egymást kizáró esetek):

I. $(b,i, \alpha A, j\beta) \vdash (q,i, \alpha' B, k\beta)$, ha a j-edik szabály $A \rightarrow \gamma$ alakú úgy, hogy létezik az a legkisebb olyan $k > j$, hogy a k-adik szabály $B \rightarrow \delta$ alakú, továbbá $\alpha\gamma = \alpha'\delta$.

II. $(b,i, \alpha A, j\beta) \vdash (q,i+1, \alpha\gamma a_i , s\beta)$, ha $i < n+1$, a j-edik szabály $A \rightarrow \gamma$ alakú úgy, hogy $\alpha\gamma$ nem redukálható más szabállyal, azaz nem létezik olyan $k > j$, hogy a k-adik szabály $B \rightarrow \delta$ alakú, továbbá $\alpha\gamma = \alpha'\delta$.

III. $(b,n+1, \alpha A, j\beta) \vdash (b,n+1, \alpha \gamma, \beta)$, ha a j-edik szabály $A \rightarrow \gamma$ alakú (továbbá nincs olyan $k > j$, mely eleget tenne az I. –beli feltételeknek és már léptetni sem tudunk).

IV. $(b, i, \alpha a, s\beta) \vdash (b, i-1, \alpha, \beta)$, ha $i > 1$.

V. $(b, 1, \alpha a, s\beta)$ esetén, azaz ha a fenti feltételek egyike sem teljesül, akkor az elemzett szó nem eleme a nyelvnek.

Példa. Legyen $G=(V_N, V_T, S, H)$, ahol $H=\{ S \rightarrow S + T , S \rightarrow T, T \rightarrow a , T \rightarrow b\}$.

Feladat: b+a ϵ L(G)?

Szabályok sorszámozása: 1. $S \rightarrow S + T$, 2. $S \rightarrow T$, 3. $T \rightarrow a$, 4. $T \rightarrow b$.

Levezetés:

(q, 1, λ, λ) ⊦(q, 2, b, s) (léptetés: a pointer 1-el nő, az elemzett sztring 1. betűje az α verem tetejére kerül)

⊦(q, 2, T, 4s) (redukálás: az első a 4. $T \rightarrow b$ szabály, mellyel tudunk redukálni)

⊦ (q, 2,S, 24s) (redukálás: az első a 2. $S \rightarrow T$ szabály, mellyel tudunk redukálni)

⊦ (q, 3,S+, s24s) (léptetés: a pointer 1-el nő, az elemzett sztring 2. betűje az α verem tetejére kerül)

⊦ (q, 4,S + a, ss24s) (léptetés: a pointer 1-el nő, az elemzett sztring 3. betűje az α verem tetejére kerül)

⊦ (q, 4,S + T, 3ss24s) (redukálás: az első a 3. $T \rightarrow a$ szabály, mellyel tudunk redukálni)

⊦ (q, 4,S, 13ss24s) (redukálás: az első az 1. $S \rightarrow T + S$ szabály, mellyel tudunk redukálni)

⊦ (t, 4,S, 13ss24s) (elfogadás, mert elértünk egy végkonfigurációt, hisz a pointer értéke 1-el nagyobb mint a b+a szó hossza, továbbá az α verem tartalma S.)

Tehát b + a ϵ L(G). Továbbá a β verem tartalmát képező 13ss24s sztringből ből törölve az s-eket, az 1324 szabály sorszám sorozatot kapjuk, vagyis azon szabályok sorszámainak sorozatát, melyek b + a jobb oldali levezetését adják.

(1: $S \rightarrow S + T$) S => S+T, (3. $T \rightarrow a$) S+T => S+a, (2. $S \rightarrow T$) S+a => T+a, (4. $T \rightarrow b$) T+a => b+a, azaz egy jobboldali levezetést kapunk:

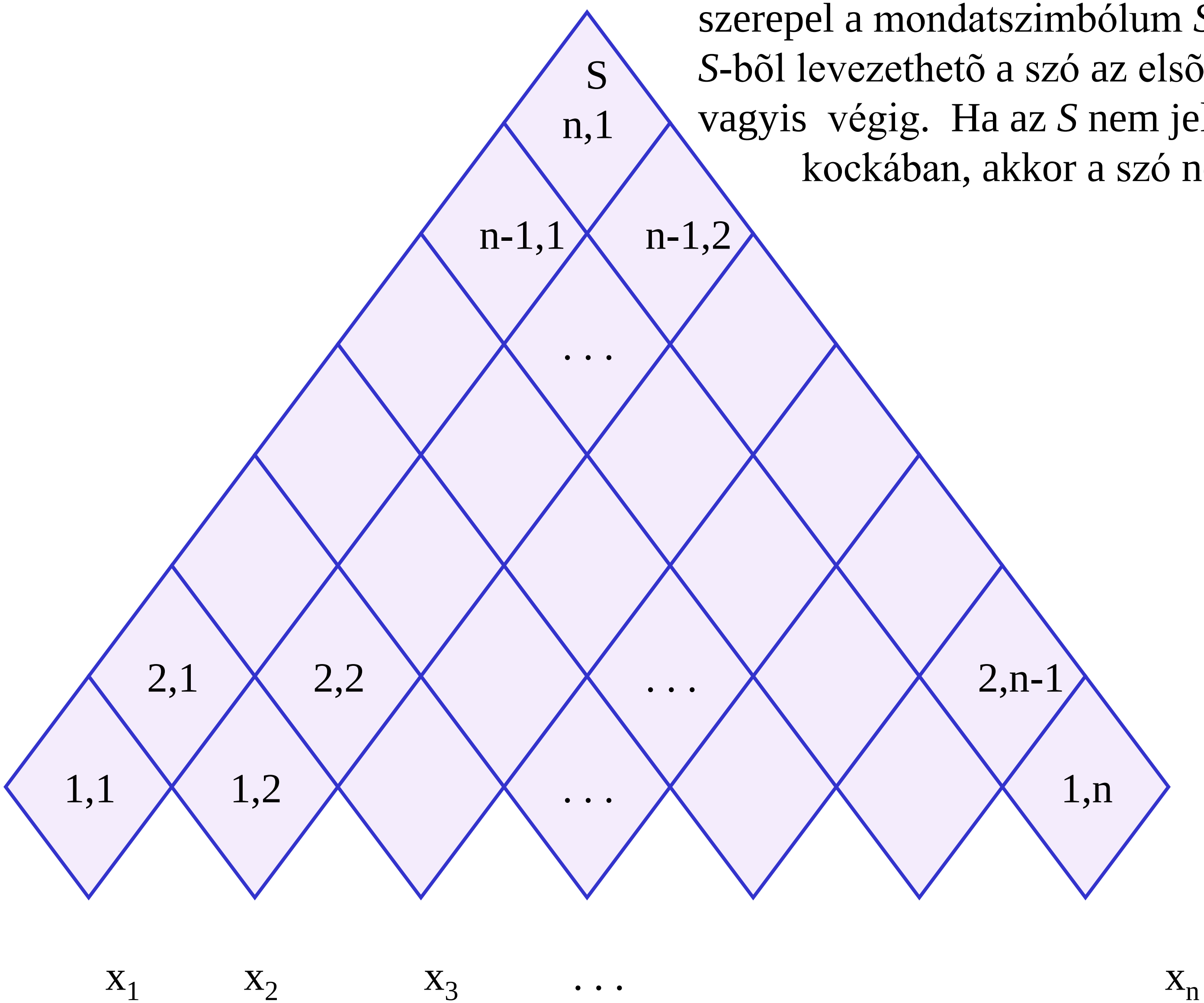
S => S+T => S+a => T+a => b+a.

A CYK algoritmus (Cocke-Younger-Kasami)

Az algoritmussal tetszőleges Chomsky féle normál alakban megadott nyelvtan és tetszőleges terminális sztring esetén eldönthető (polinomiális időben), hogy a sztring eleme-e a nyelvtan által generált nyelvnek.

Az algoritmus egy alulról felfele történõ elemzést valósít meg. Ahhoz, hogy működjön az kell, hogy a $G=(V_N, V_T, S, H)$ nyelvtan Chomsky normál alakban (CNF) legyen, azaz a nyelvtanban csak $A \to BC$ ($A, B, C \in V_N$), illetve $A \to a$ ($A \in V_N$, $a \in V_T$) alakú szabályok vannak . Ha egy $n > 0$ hosszú ($x_1, \ldots, x_n \in V_T$) szót szeretnénk elemezni, akkor egy n x n-es alsó háromszög mátrix alakú táblázatot fogunk kitölteni a következõ módon. A sorokat alulról felfele számozzuk, az oszlopokat balról jobbra. Az i. sor j. kockájába akkor kerül egy $A \in V_N$ nemterminálisa a nyelvtannak, ha az A-ból levezethetõ az elemzendõ input szó azon darabkája, ami a j. betûnél kezdõdik és i hosszan tart, vagyis levezethetõ az részszó.

Ha kitöltjük a táblázatot és a legfelsõ mezõben szerepel a mondatszimbólum *S* az azt jelenti, hogy *S*-bõl levezethetõ a szó az elsõ betûtõl az *n*.-ig, vagyis végig. Ha az *S* nem jelenik meg a legfelsõ kockában, akkor a szó nem eleme a nyelvnek.



A táblázat kitöltése:

Az elsõ sor egyértelmû: azok a nemterminálisok kerülnek a *k*. mezõbe, akik egy lépésben a *k*. terminálist generálják egy alakú szabállyal.

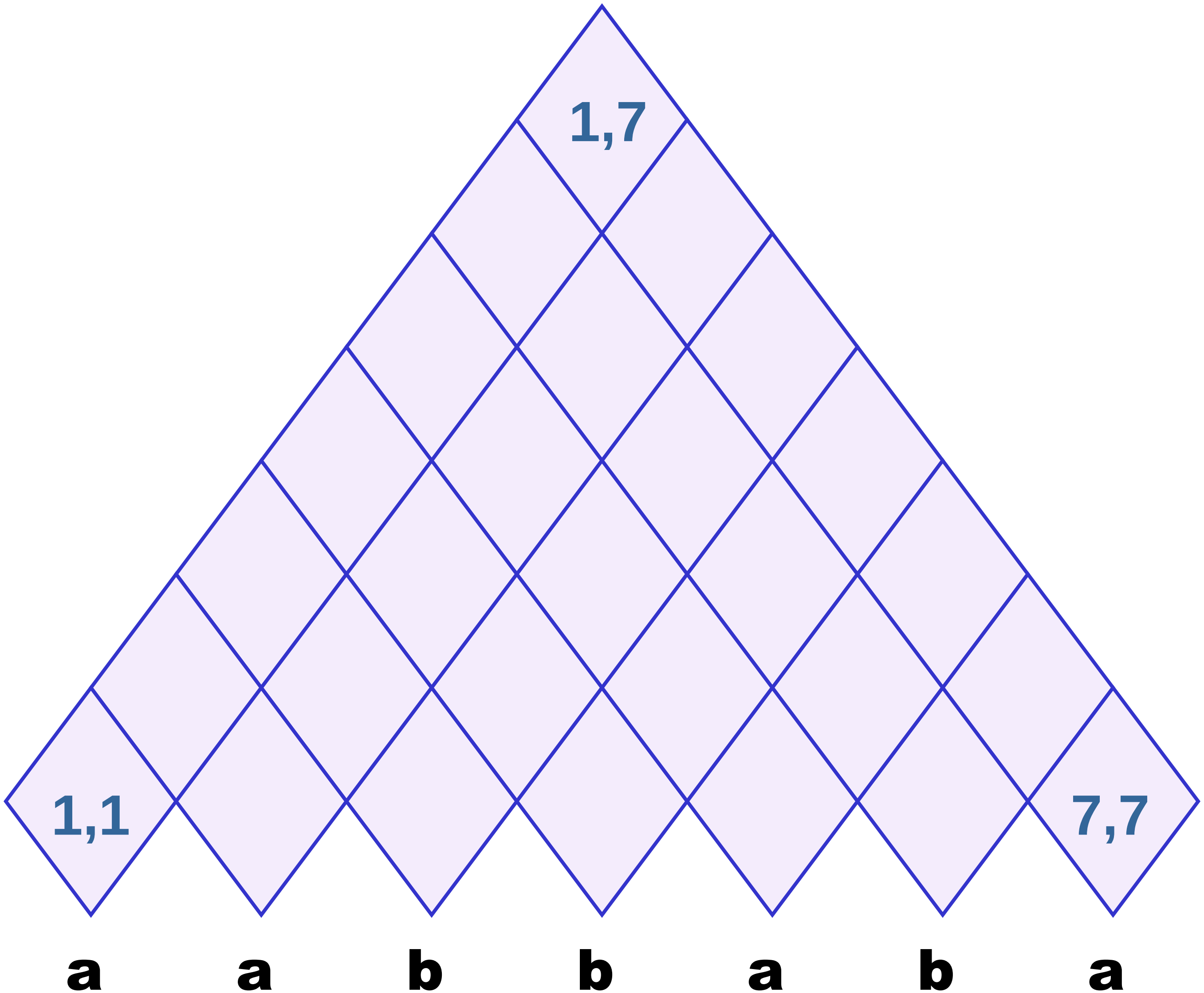
Késõbbi sorok: egy *A* akkor lesz az *i*. sor *j*. oszlopában, ha belõle levezethetõ az szó. Mivel csak alakú szabályok vannak ezért ez csak úgy lehet, hogy a *B* megcsinálja -t (az elejét, valameddig), a *C* pedig -et (a maradékot). De ezt már le lehet ellenõrizni, mert ezek az információk a táblázat már kitöltött részében benne vannak. Tehát egy *A*-t akkor írunk be az *i*. sor *j*. kockájába, ha van olyan szabály, hogy *B* benne van a *j*. oszlop *k*. sorában valami *k*-ra, a *C* meg benne van a *k*+1. oszlop *i*-*k*. sorában.

Ha nem csak arra vagyunk kíváncsiak, hogy generálni lehet-e a szót, hanem arra is, hogy hogyan, akkor nem csak a megfelelõ nemterminálist írjuk be a táblázatba, hanem ellátjuk két indexszel is: az elsõ mutatja, hogy milyen felbontásban generálja a *BC* sorozat a szórészletet (azaz, hogy a *B* hány darab betû generál, ez a fenti jelölésekkel a *k*), a második meg annak a szabálynak a száma, amit használunk (vagyis az szabály sorszáma, a szabályokat még az elején megszámoztuk, hogy lehessen rájuk hivatkozni). Az elsõ index tulajdonképpen azt mutatja, hogy az így beírt *A* oszlopában hányadik sorban kell keresnünk a *B*-t, a szabály száma meg azt mutatja, hogy mit is kell keresnünk. Így a levezetési fa felépíthetõ. Ha ezen visszakeresés során elágazást tapasztalunk (azaz van olyan kocka, ahol két ugyanolyan, de más indexû nemterminális áll), akkor a szó nem egyértelmûen áll elõ. Ekkor a visszakeresõs eljárás mindkét levezetési fát megadja.

Példa a

C-Y-K-algoritmus

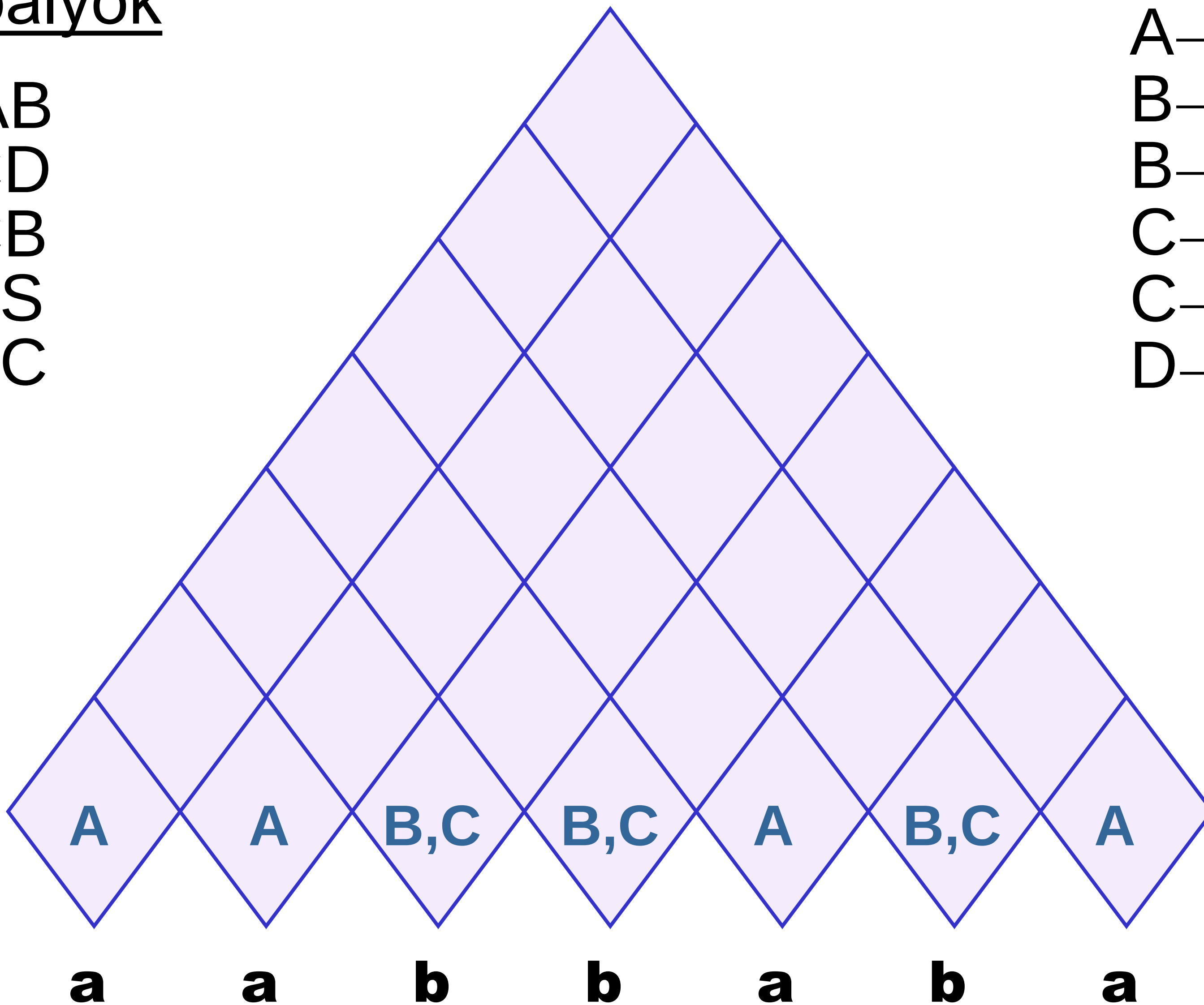
alkalmazására


1,7
1,1
7,7
a
a
b
b
a
b
a

Szabályok

$S \rightarrow AB$
$S \rightarrow CD$
$S \rightarrow CB$
$S \rightarrow SS$
$A \rightarrow BC$

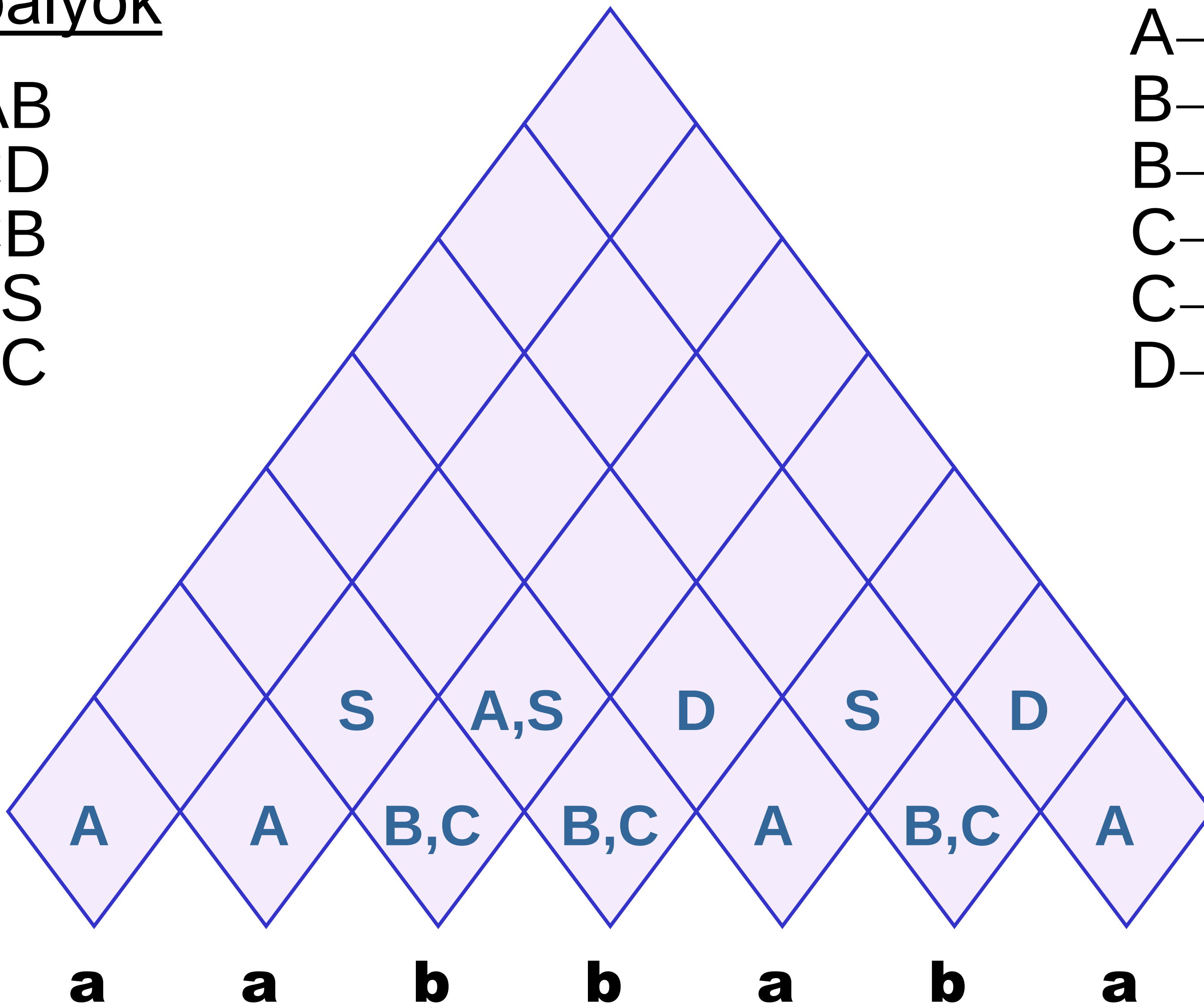
$A \rightarrow a$
$B \rightarrow SC$
$B \rightarrow b$
$C \rightarrow DD$
$C \rightarrow b$
$D \rightarrow BA$



Szabályok

$S \rightarrow AB$
$S \rightarrow CD$
$S \rightarrow CB$
$S \rightarrow SS$
$A \rightarrow BC$

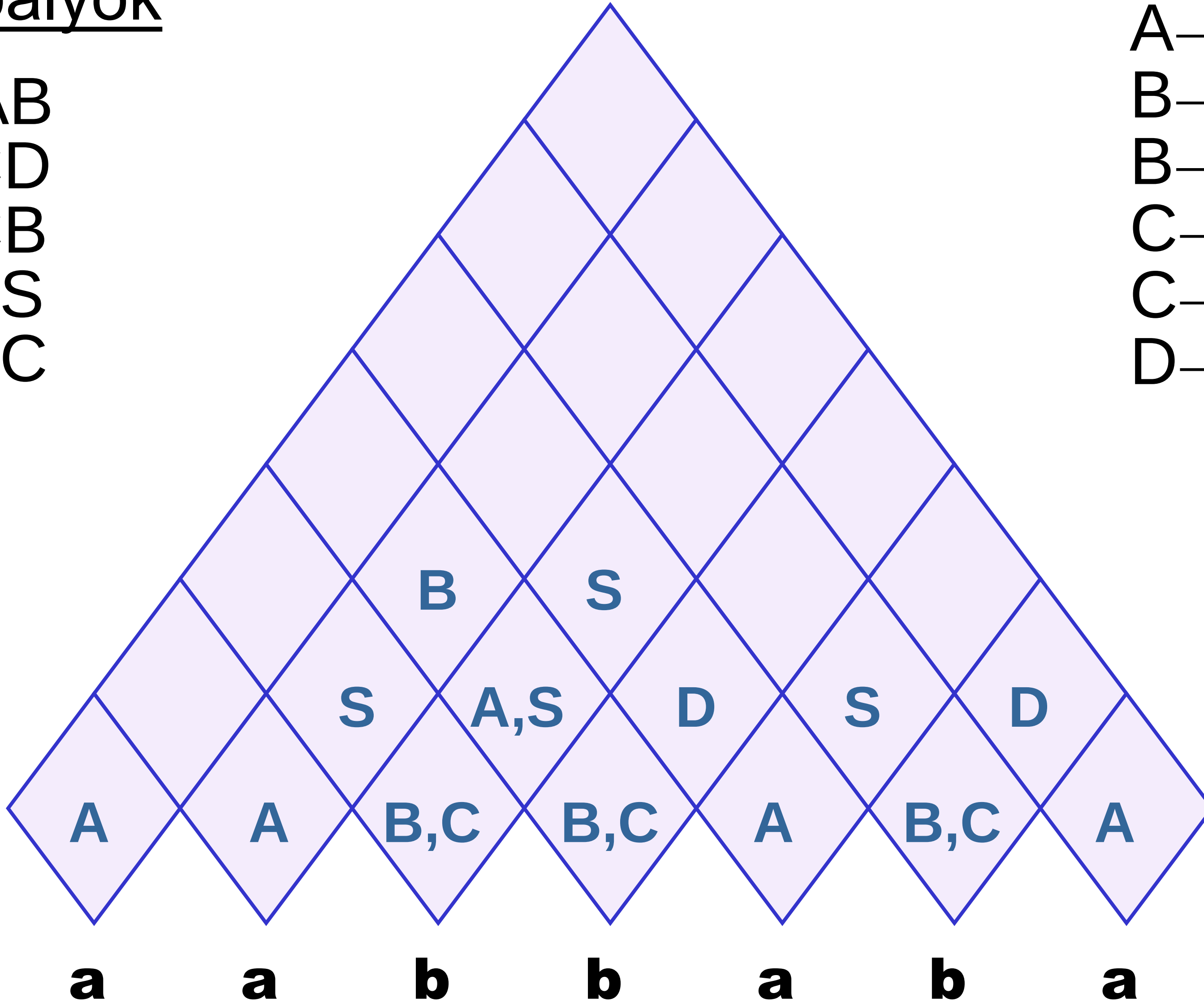
$A \rightarrow a$
$B \rightarrow SC$
$B \rightarrow b$
$C \rightarrow DD$
$C \rightarrow b$
$D \rightarrow BA$



Szabályok

S→AB
S→CD
S→CB
S→SS
A→BC

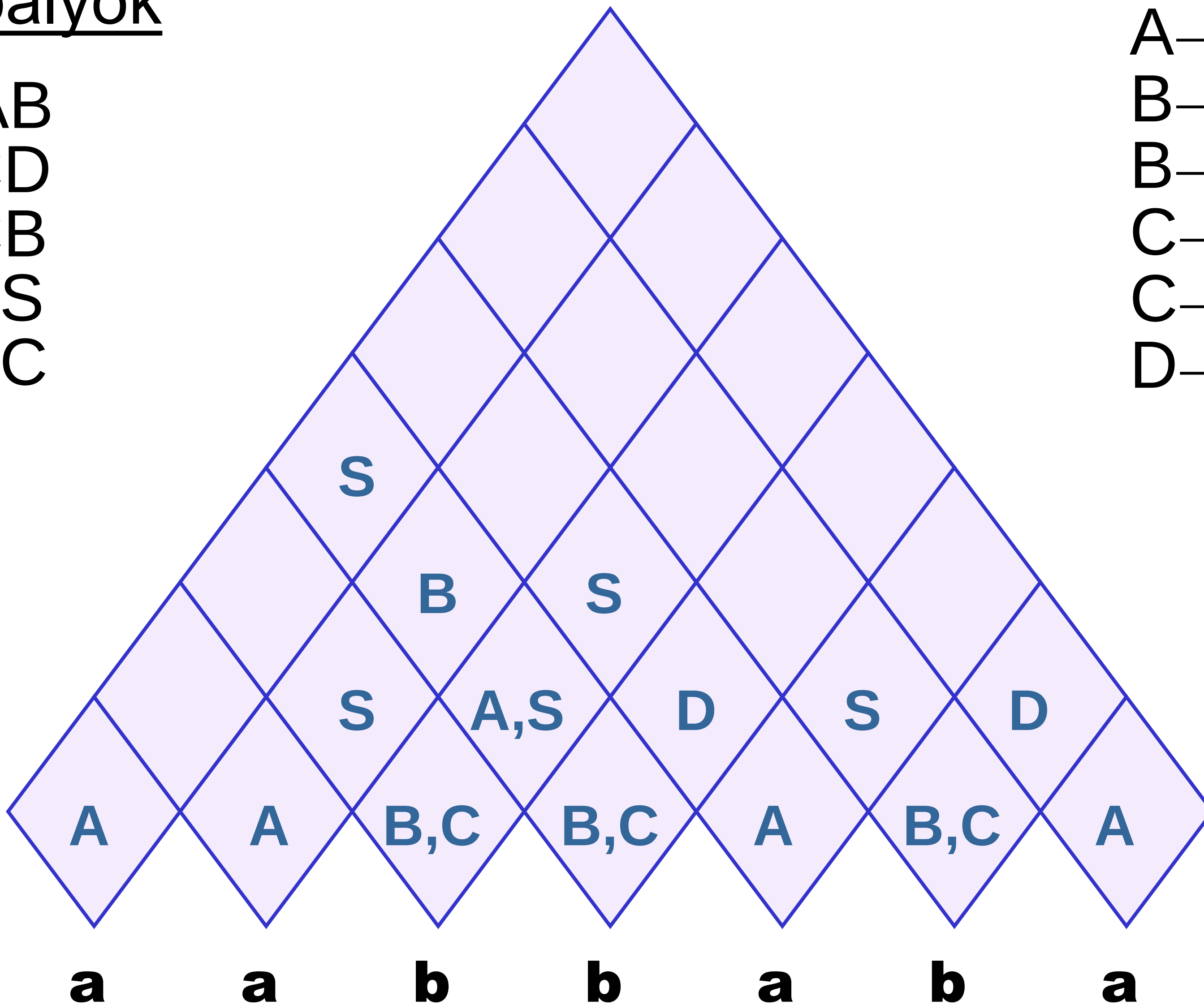
A→a
B→SC
B→b
C→DD
C→b
D→BA



Szabályok

$S \rightarrow AB$
$S \rightarrow CD$
$S \rightarrow CB$
$S \rightarrow SS$
$A \rightarrow BC$

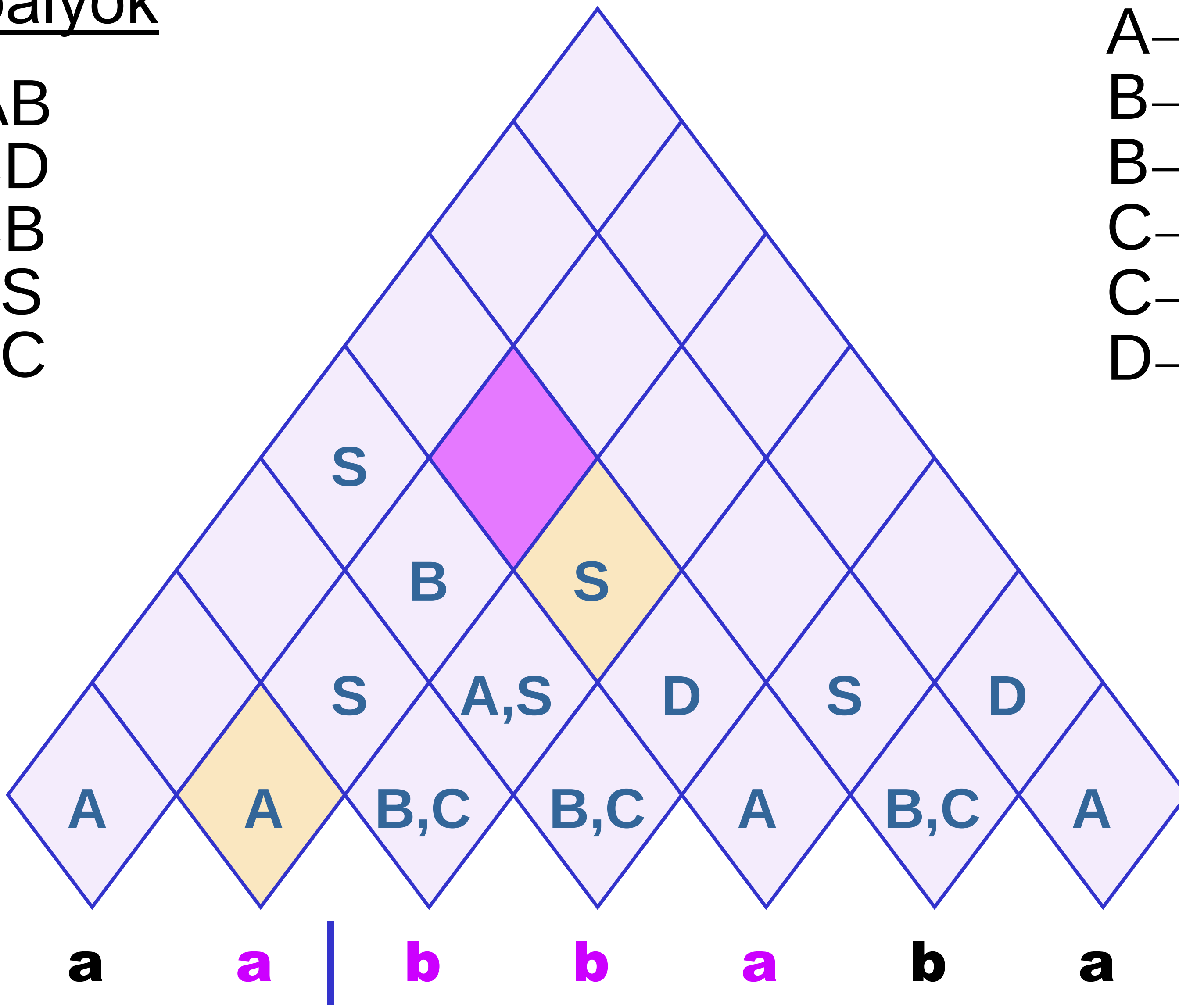
$A \rightarrow a$
$B \rightarrow SC$
$B \rightarrow b$
$C \rightarrow DD$
$C \rightarrow b$
$D \rightarrow BA$



Szabályok

$S \rightarrow AB$
$S \rightarrow CD$
$S \rightarrow CB$
$S \rightarrow SS$
$A \rightarrow BC$

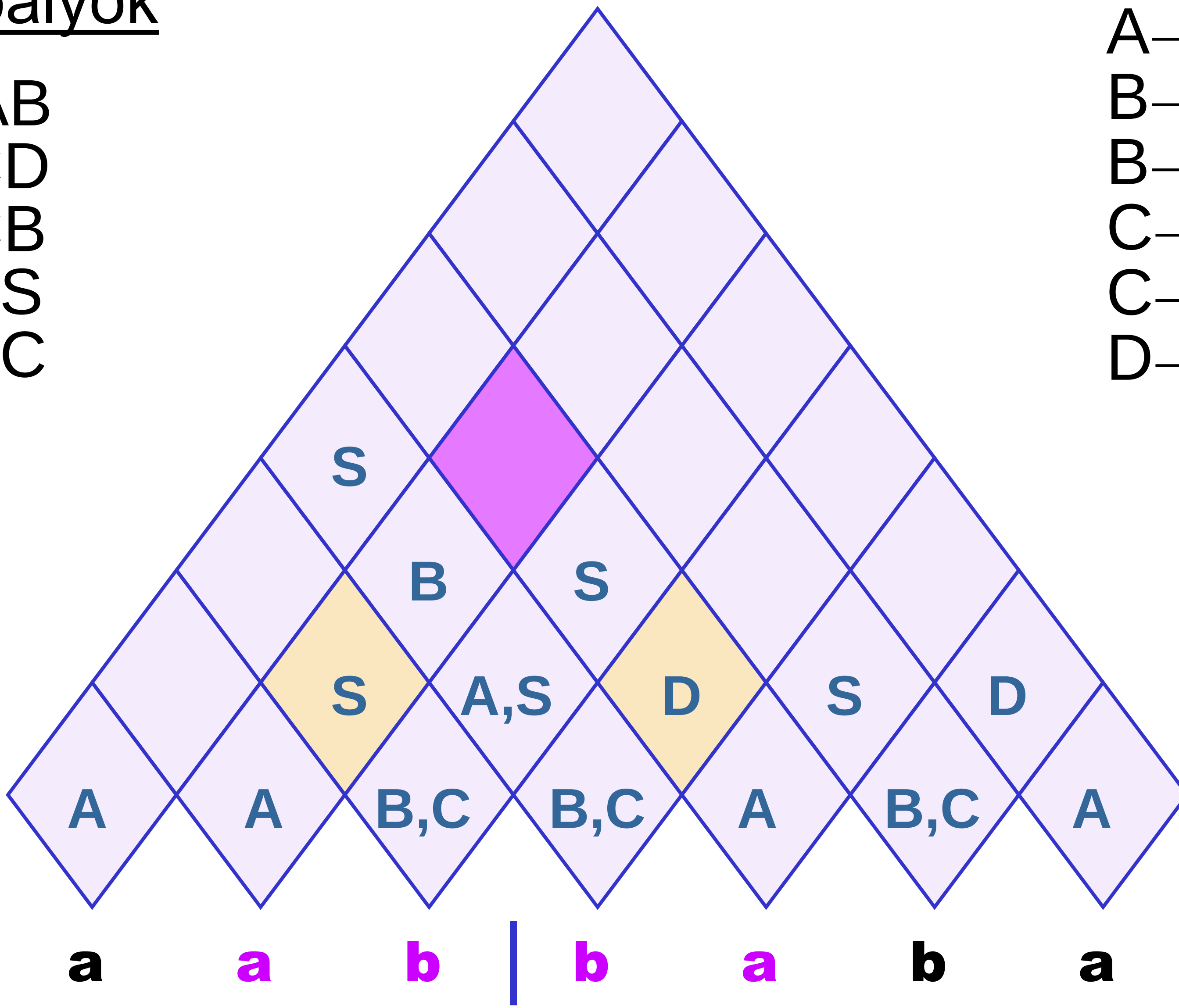
$A \rightarrow a$
$B \rightarrow SC$
$B \rightarrow b$
$C \rightarrow DD$
$C \rightarrow b$
$D \rightarrow BA$



Szabályok

S→AB
S→CD
S→CB
S→SS
A→BC

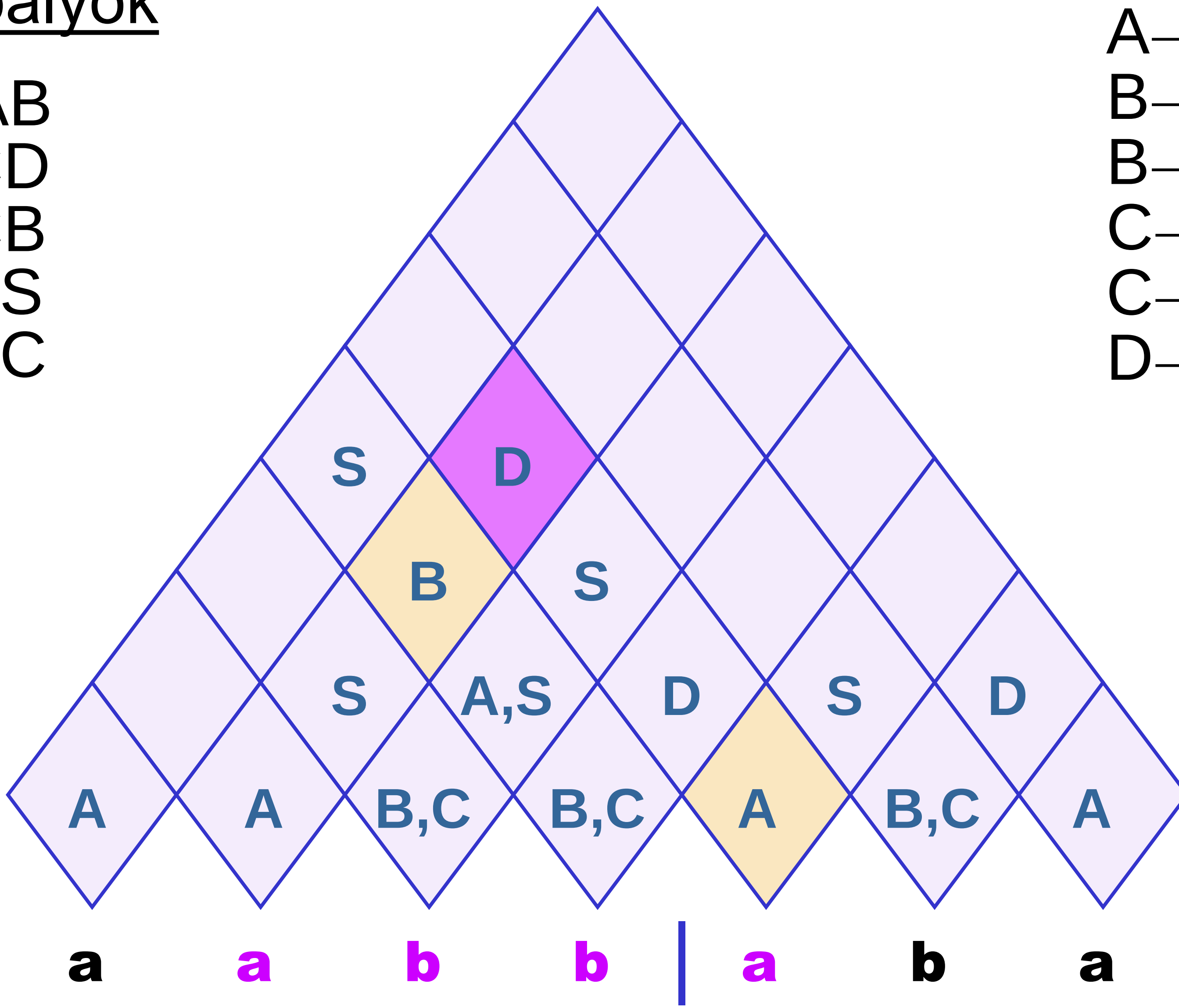
A→a
B→SC
B→b
C→DD
C→b
D→BA



Szabályok

S→AB
S→CD
S→CB
S→SS
A→BC

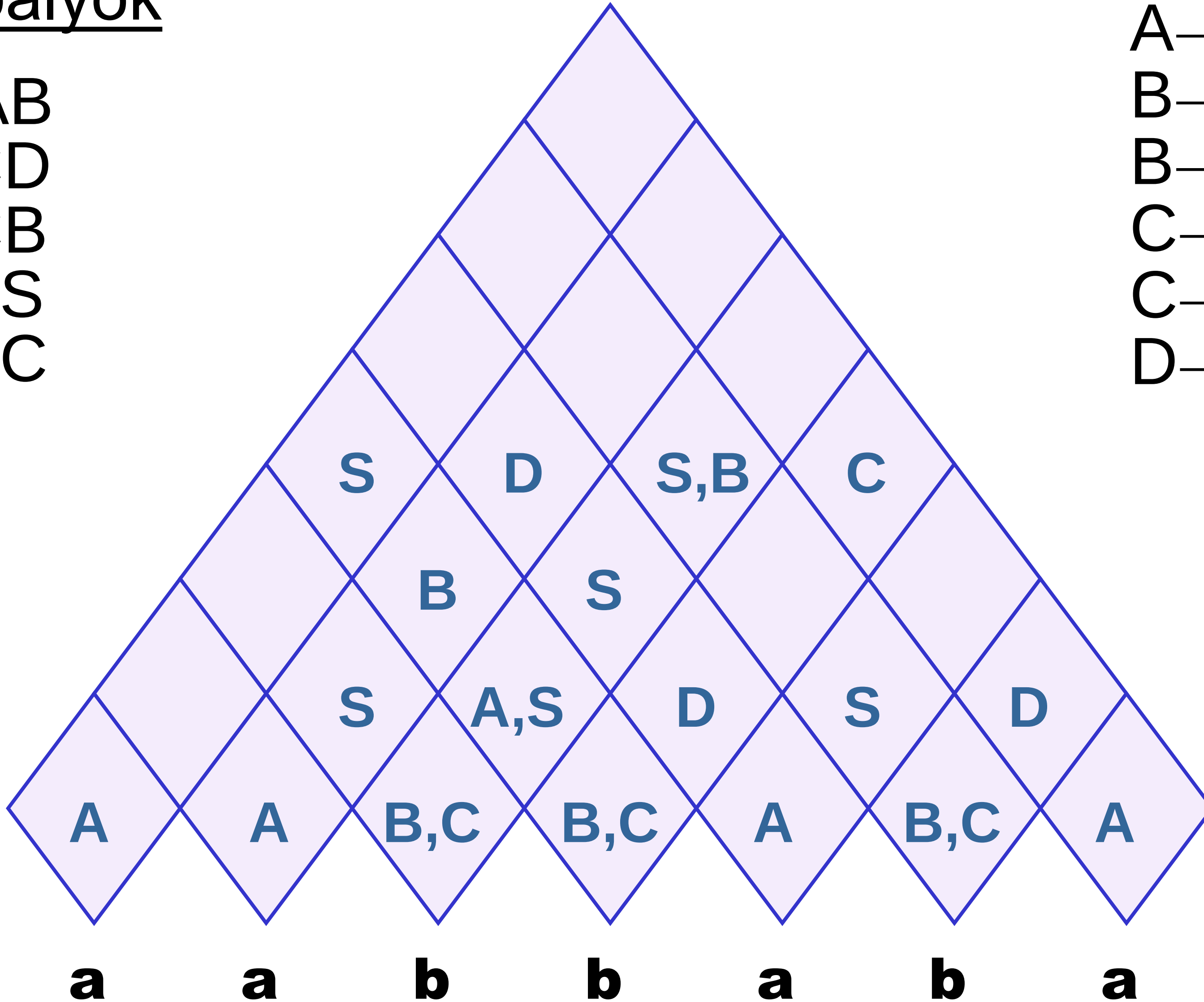
A→a
B→SC
B→b
C→DD
C→b
D→BA



Szabályok

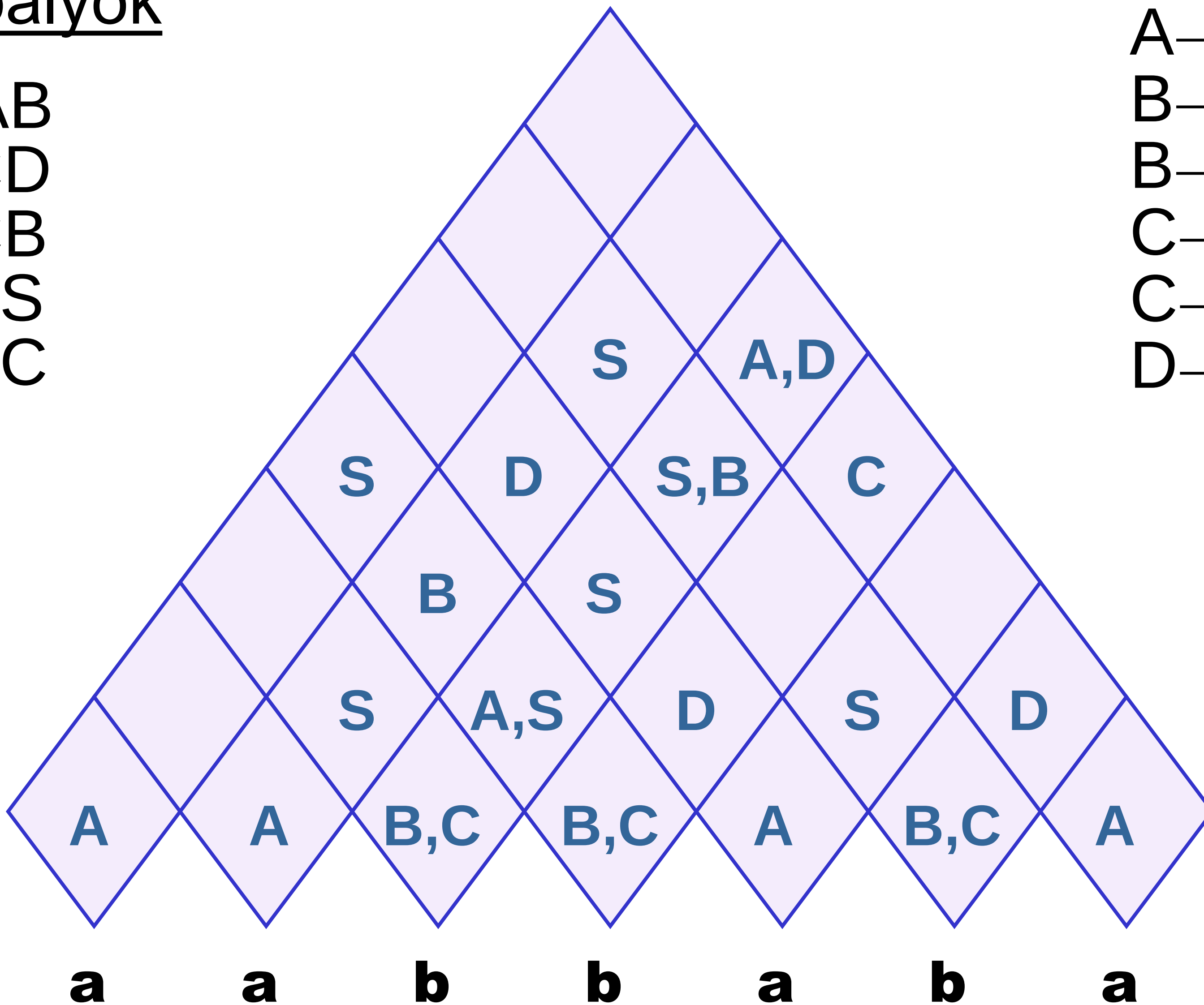
S→AB
S→CD
S→CB
S→SS
A→BC

A→a
B→SC
B→b
C→DD
C→b
D→BA



Szabályok

S→AB
S→CD
S→CB
S→SS
A→BC

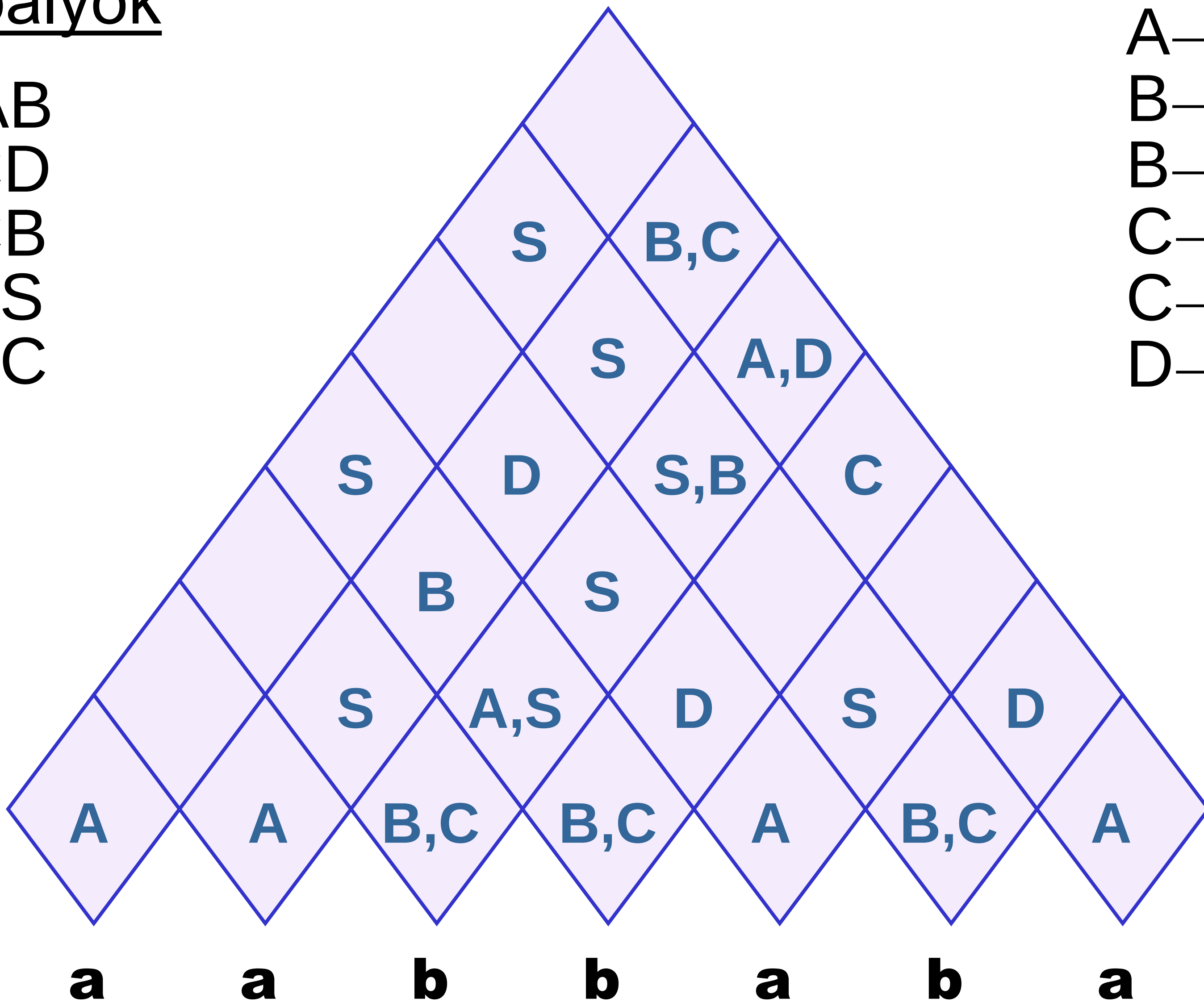


A→a
B→SC
B→b
C→DD
C→b
D→BA

Szabályok

S→AB
S→CD
S→CB
S→SS
A→BC

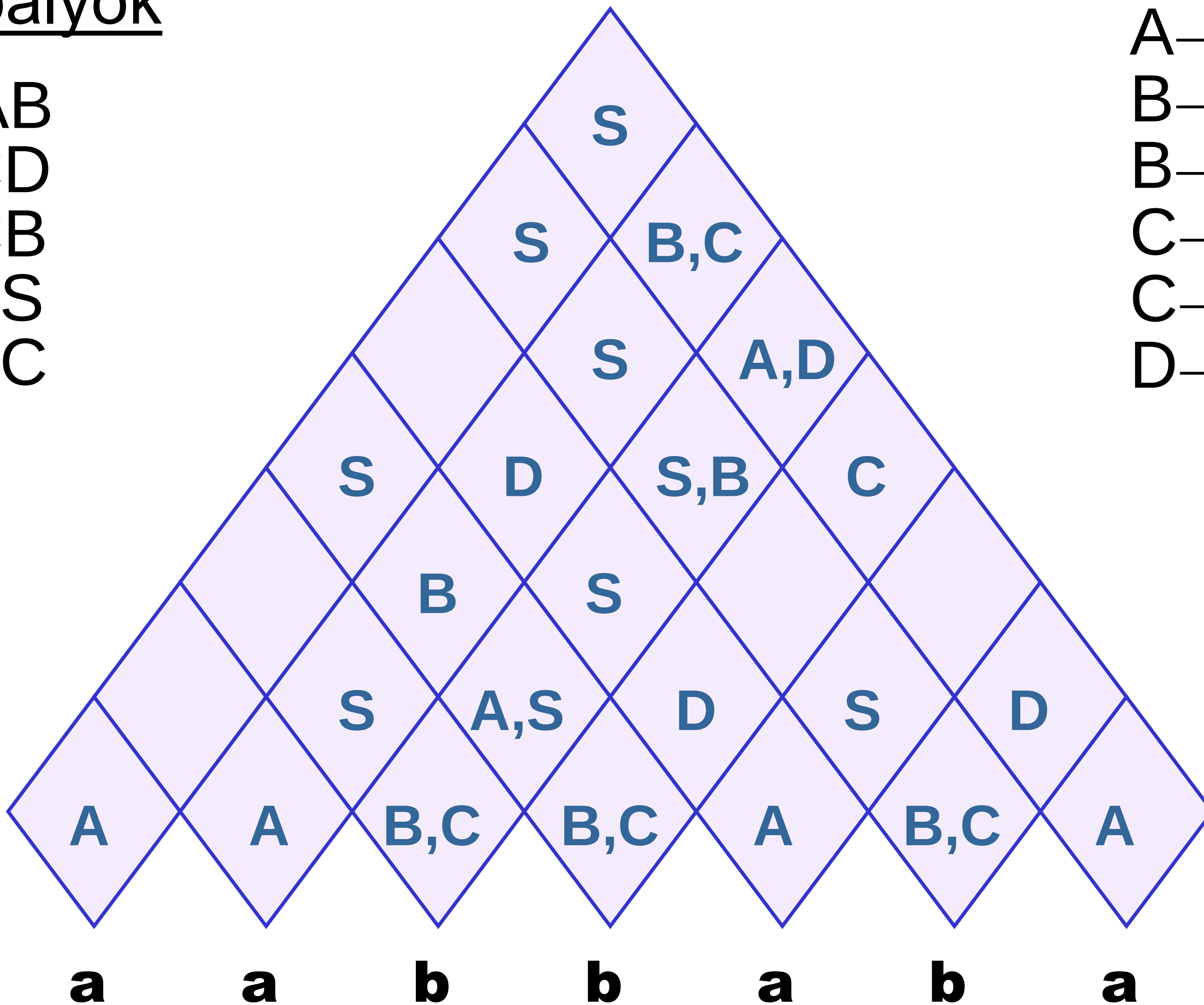
A→a
B→SC
B→b
C→DD
C→b
D→BA

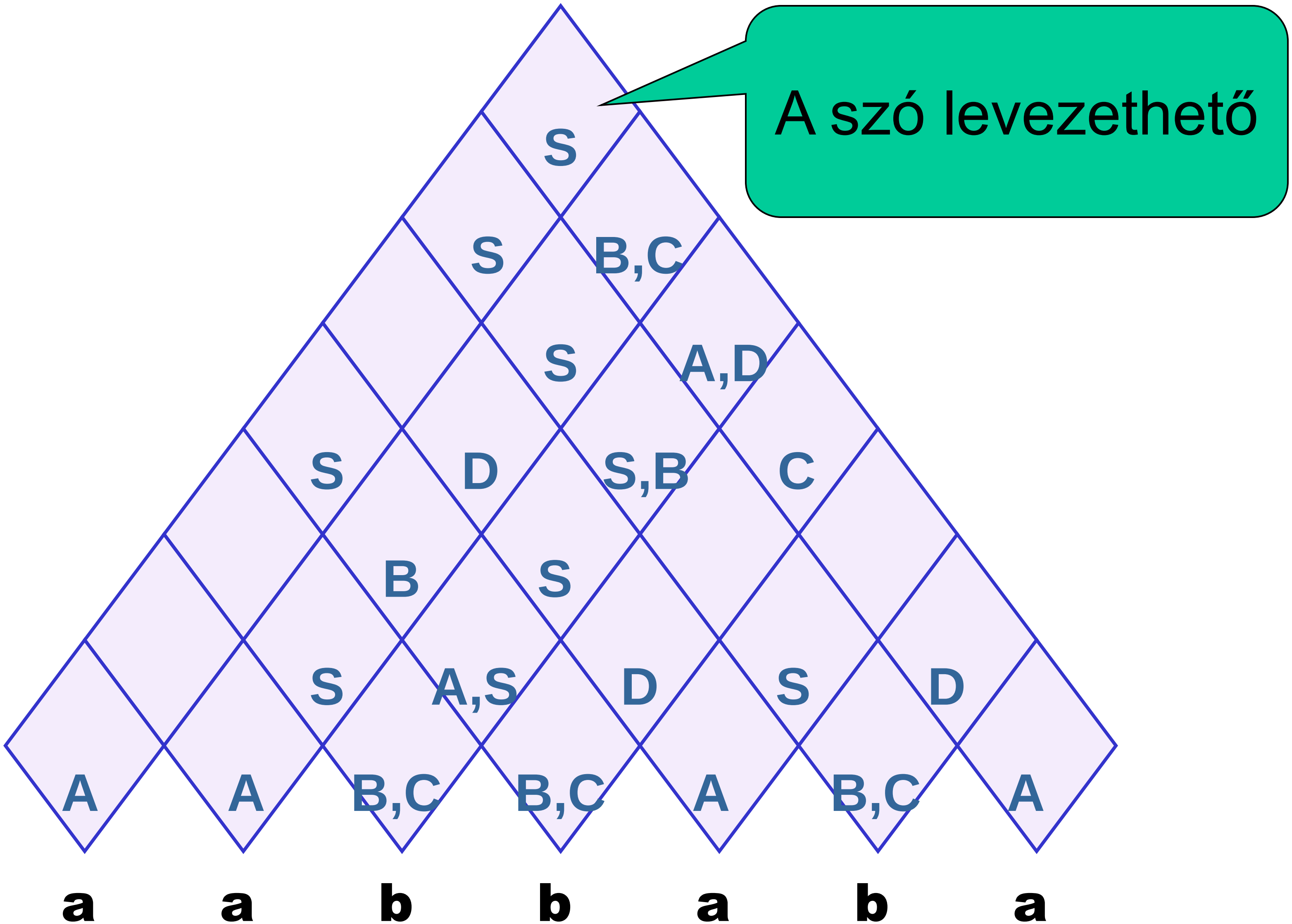


Szabályok

S→AB
S→CD
S→CB
S→SS
A→BC

A→a
B→SC
B→b
C→DD
C→b
D→BA




A szó levezethető
S
S
B,C
S
A,D
S
D
S,B
C
B
S
S
A,S
D
S
D
A
A
B,C
B,C
A
B,C
A
a
a
b
b
a
b
a

LL(k) és LR(k) elemzés

LL(k) elemzés: visszaléptetés nélküli kiterjesztés-illesztés típusú elemzés.

LR(k) elemzés: visszaléptetés nélküli léptetés-redukálás típusú elemzés.

Legyen $First_k(\alpha)$ ($k \geq 0$, $\alpha \in (V_N \cup V_T)^*$) az α-ból levezethető szimbólumsorozatok k hosszuságú kezdő terminális sorozatainak halmaza, azaz
$First_k(\alpha) = \{x \mid \alpha \Rightarrow^* x\beta \text{ és } |x| = k\} \cup \{x \mid \alpha \Rightarrow^* x \text{ és } |x| < k\}$
($x \in V^*_T$, $\beta \in (V_N \cup V_T)^*$) .

Tehát az $First_k(x)$ halmaz az x első k darab szimbólumát, $|x| < k$ esetén pedig a teljes x-t tartalmazza. Ha $\alpha \Rightarrow^* \lambda$, akkor természetesen $\lambda, \in First_k(\alpha)$.

A G nyelvtan LL(k) nyelvtan ($k \geq 0$), ha bármely két
$S \Rightarrow^* wA\beta \Rightarrow^* w\alpha_1\beta \Rightarrow^* wx$,
$S \Rightarrow^* wA\beta \Rightarrow^* w\alpha_2\beta \Rightarrow^* wy$
($A \in V_N$, $x, y,w \in V^*_T$, $\alpha_1, \alpha_2, \beta \in (V_N \cup V_T)^*$) levezetésre
$First_k (x) = First_k (y)$ esetén $\alpha_1 = \alpha_2$.
A fenti értelmezés szerint, ha egy nyelvtan LL(k) nyelvtan, akkor a már elemzett w utáni k darab terminális szimbólum az A-ra alkalmazható helyettesítési szabályt egyértelműen meghatározza .
Az értelmezésből az is látható, hogy ha egy nyelvtan $LL(k_0)$ nyelvtan, akkor minden $k > k_0$-ra is LL(k) nyelvtan. Ha LL(k) nyelvtanról beszélünk, akkor k alatt mindig azt a legkisebb k-t értjük, amelyre az értelmezésben megadott tulajdonság teljesül.

Egyszerű LL(1) grammatika:

(a) Minden helyettesítési szabály jobboldala terminális betűvel kezdőődik;

(b) Minden A nemterminális minden egymástól különböző $A \rightarrow \beta_1$, $A \rightarrow \beta_2$ helyettesítési szabályára (alternatívájára β_1 és β_2 első karaktere különbözik

Például: $S \rightarrow aS \mid bAc$
$A \rightarrow bAc \mid d$

Módszer: Először besorszámozzuk a szabályokat. A sorrend tetszőleges, de az eljárás alatt nem változik)

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

Legyen: $T[X, a]$ =- $(a\beta, (i))$, ha $X \rightarrow a\beta$ az i-edik helyettesítési szabály ,
- pop, ha X = a ,
- accept, ha X = # és a = # ,
- hiba egyébként .

Utána elkészítjük az elemző táblázatot

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

Megjegyzés: Van olyan környezetfüggetlen nyelvtan, mely semmilyen k-ra sem LL(k) nyelvtan.

Tétel. A G nyelvtan akkor és csak akkor LL(k) nyelvtan, ha minden $S \Rightarrow^* wA\beta$, és $A \rightarrow \gamma \mid \delta$ ($\gamma \neq \delta$, $w \in V_T^*$, $A \in V_N^*$, $\beta, \gamma, \delta \in (V_N \cup V_T)^*$) esetén $First_k(\gamma\beta) \cap First_k(\delta\beta) = \emptyset$.

$FOLLOW_k(\beta)= \{x \mid S \Rightarrow^* \alpha\beta\gamma$ és $x \in Firstk(\gamma)\}$,
és ha
$\lambda \in FOLLOW_k(\beta)$, akkor legyen $FOLLOW_k(\beta) = FOLLOW_k(\beta)\backslash\{\lambda\}\cup\{\#\}$
($\alpha, \beta, \gamma \in \in (V_N \cup V_T)^*$), $x \in V_T^*$).

A $FOLLOW_k(A)$ halmazt megkapjuk, ha vesszük az összes A-t tartalmazó mondatformából az A-t közvetlenül követő terminálisok közül a First k –t.¨Azaz:
$FOLLOW_k(A) = \{FIRST_k(\beta) : S \Rightarrow^+ \alpha A\beta\}$

A gyakorlatban LL(1) elemzőket szokás használni.

Tétel. A G nyelvtan akkor és csak akkor LL(1) nyelvtan, ha minden A nemterminális szimbólum $A \rightarrow \gamma \mid \delta$ helyettesítési szabályaira
$First_1(\gamma FOLLOW_1(A)) \cap First_1(\delta FOLLOW_1(A)) = \emptyset$.

A G nyelvtant **erős LL(k) grammatikának** hívjuk, ha tetszőleges $A \in V_N$ és $A \rightarrow \beta$, $A \rightarrow \gamma$ különböző szabályokra $FIRST_k(\beta FOLLOW_k(A)) \cap FIRST_k(\gamma FOLLOW_k(A)) = \emptyset$.
(k=1 esetén tehát minden LL(k) nyelvtan egyben erős LL(k) nyelvtan is.)

Az LL(1) elemzést egy T táblázat kitöltésével kezdjük az alábbiak szerint:

Legyen # tetszőleges szimbólum, mely nem eleme $V_N \cup V_T$ -nek

Minden $X \in V_N \cup V_T \cup \{\#\}$ és $a \in V_T \cup \{\#\}$ párra

Legyen:

T[X, a] =

- $(\beta, (i))$, ha $X \to \beta$ az i-edik helyettesítési szabály ,
  $a \in FIRST(\beta)$ vagy
  $(\lambda \in FIRST(\beta)$ és $a \in FOLLOW(X))$,
- pop, ha X = a ,
- elfogad, ha X = # és a = # ,
- hiba egyébként .

Kezdő konfiguráció: $(w\#, S\#, \lambda)$, ahol is w az elemzendő terminális sztring.

Sikeres elemzés eredménye: egy (#,#, z) végkonfiguráció. Átmeneti reláció:

Ha a még nem elemzett szöveg az ay#, és a verem tetején az X szimbólum áll,

az átmeneti reláció értéke a következő: $(ay\#, X\alpha\#, v) \vdash$

- $(ay\#, \beta\alpha\#, v(i))$, ha $T[X, a] = (\beta, (i))$,
- $(y\#, \alpha\#, v)$, ha T[X, a] = pop ,
- O.K., ha T[X, a] = elfogad ,
- HIBA, ha T[X, a] = hiba .

O.K. Esetén a $z = i_1 i_2 \ldots i_m$, ahol is i_1 az első, i_2 a második,..., i_m az utolsó alkalmazandó szabály sorszámát jelöli egy baloldali levezetésben.

$$(aabbdcc\#, S\#, \lambda) \overset{(aS,1)}{\vdash} (aabbdcc\#, aS\#, 1) \overset{pop}{\vdash} (abbdcc\#, S\#, 1) \overset{(aS,1)}{\vdash} (abbdcc\#, aS\#, 11)$$

$$\overset{pop}{\vdash} (bbdcc\#, S\#, 11) \overset{(bAc,\ 2,)}{\vdash} (bbdcc\#, bAc\#, 112) \overset{pop}{\vdash} (bdcc\#, Ac\#, 112) \overset{(bAc,3)}{\vdash} (bdcc\#, bAcc\#,1123)$$

$$\overset{pop}{\vdash} (dcc\#, Acc\#, 1123) \overset{(d,4)}{\vdash} (dcc\#, dcc\#, 11234) \overset{pop}{\vdash} (cc\#, cc\#, 11234) \overset{pop}{\vdash} (c\#, c\#, 11234)$$

$$\overset{pop}{\vdash} (\#,\#,11234) \overset{accept}{\vdash} (\lambda, \lambda, 11234)$$

$$S \overset{1}{=>} aS \overset{1}{=>} aaS \overset{2}{=>} aabAc \overset{3}{=>} aabbAcc \overset{4}{=>} (aabbdcc)$$

$First_k(\alpha) = \{x \mid \alpha \Rightarrow^* x\beta \text{ és } |x| = k\} \cup \{x \mid \alpha \Rightarrow^* x \text{ és } |x| < k\}$
$(x \in V^*_T ,\ \beta \in (V_N \cup V_T)^*)$, azaz
$First_1(\alpha) = \{a \mid \alpha \Rightarrow^* a\beta\ \} \cup \{\lambda \mid \alpha \Rightarrow^* \lambda\ \}$
$(a \in V_T ,\ \beta \in (V_N \cup V_T)^*)$
vagyis λ-mentes esetben $First_1(\alpha) = \{a \mid \alpha \Rightarrow^* a\beta \text{ és } a \in V_T\}$
$(a \in V_T ,\ \beta \in (V_N \cup V_T)^*)$.
λ-mentes LL(1) grammatika:
(a) λ-mentes
(b) minden egymástól különböző, ugyanazon nemterminálist
helyettesítő $A \rightarrow \beta_1$, $A \rightarrow \beta_2$ helyettesítési szabályára (alternatívájára)
$First_1(\beta_1) \cap First_1(\beta_2) = \emptyset$.

$FIRST_1(x)$, $x \in V_N \cup V_T$ kiszámítása:

1. Minden $a \in V_T$ és $i \geq 0$ esetén $H_i(a) = \{a\}$,
2. Legyen minden $A \in V_N$ esetén $H_0(A) = \{a \in V_T \mid A \rightarrow a\alpha \in H\}$
3. Ha minden $A \in V_N$ esetén $H_0(A), H_1(A), \ldots, H_i(A)$ mind ismertek, akkor $H_{i+1}(A) = H_i(A) \cup \{a \in H_i(X) \mid A \rightarrow X\alpha \in H, X \in V_N \cup V_T\}$
4. Ha minden $A \in V_N$ esetén $H_i(A) = H_{i+1}(A)$ akkor minden $A \in V_N$ esetén $FIRST_1(A) = H_i(A)$ és kész vagyunk, különben $i+1 \rightarrow i$ és ugrás 3-ra

Legyen például
$S \rightarrow ABC$
$A \rightarrow a \mid Bbc \mid Ccd$
$B \rightarrow bBb \mid cCc$
$C \rightarrow dDd \mid Dd$
$D \rightarrow e$

	S	A	B	C	D
H_0		a	b,c	d	e
H_1	a	a,b,c,d	b,c	d,e	e
H_2	a,b,c,d	a,b,c,d,e	b,c	d,e	e
H_3	a,b,c,d,e	a,b,c,d,e	b,c	d,e	e
H_4	a,b,c,d,e	a,b,c,d,e	b,c	d,e	e
$FIRST_1$	a,b,c,d,e	a,b,c,d,e	b,c	d,e	e

Később kell majd ez is:
$FIRST_1(\beta)$, $\beta \in (V_N \cup V_T)^*$ kiszámítása: 1. Legyen először $FIRST_1(\beta) = \emptyset$
2. ha $\beta = \lambda$ akkor $FIRST_1(\beta) = \{\lambda\}$ és készen vagyunk
3. ha β soron következő (először az első) betűje egy $a \in V_T$ terminális akkor $FIRST_1(\beta) = FIRST_1(\beta) \cup \{a\}$ és készen vagyunk
4. ha β soron következő (először az első) betűje egy $A \in V_N$ nemterminális akkor $FIRST_1(\beta) = FIRST_1(\beta) \cup FIRST_1(A)$ és
4.1 ugrás 3-ra ha $\lambda \in FIRST_1(A)$ és A nem az utolsó betűje β-nak
4.2 készen vagyunk ha λ nem eleme $FIRST_1(A)$-nak vagy A utolsó betűje β-nak

$S \to ABC$
$A \to a \mid Bbc \mid Ccd$
$B \to bBb \mid cCc$
$C \to dDd \mid Dd$
$D \to e$

$First_1$	S	A	B	C	D	a	b	c	d	e
S		1	2	2	3	2	3	3	3	4
A			1	1	2	1	2	2	2	3
B							1	1		
C					1				1	2
D										1
a						1				
b							1			
c								1		
d									1	
e										1

Alternatívák $First_1$-jei:

S: S-nek csak egy alternatívája van: $First_1(ABC)=First_1(A)=\{a,b,c,d,e\}$

A: $First_1(a)=a$, $First_1(Bbc)=First_1(B)=\{b,c\}$,$First_1Ccd)=First_1(C)=\{d,e\}$: $\{a\}$, $\{b,c\}$, $\{d,e\}$ diszjunktak

B: $First_1(bBb)=First_1(b)=\{b\}$, $First_1(cCc)=First_1(c)=\{c\}$: $\{b\}$, $\{c\}$ diszjunktak

C: $First_1(dDd)=First_1(d)=\{d\}$, $First_1(Dd)=First_1(D)=\{e\}$: $\{d\}$, $\{e\}$ diszjunktak.

D: D-nek csak egy alternatívája van: $First_1(e)=\{e\}$.

$S \to (1)ABC$
$A \to (2)\ a \mid (3)Bbc \mid (4)\ Ccd$
$B \to (5)bBb \mid (6)cCc$
$C \to (7)dDd \mid (8)Dd$
$D \to (9)\ e$

	a	**b**	**c**	**d**	**e**	**#**
S	(ABC,1)	(ABC,1)	(ABC,1)	(ABC,1)	(ABC,1)	
A	(a,2)	(Bbc,3)	(Bbc,3)	(Ccd,4)	(Ccd,4)	
B		(bBb,5)	(cCc,6)			
C				(dDd,7)	(Dd,8)	
D					(e,9)	
a	pop					
b		pop				
c			**pop**			
d				pop		
e					pop	
#						accept

S: S-nek csak egy alternatívája van, $First_1(ABC)= First_1(A)=\{a,b,c,d,e\}$
A: $First_1(a)=a$, $First_1(Bbc)=First_1(B)=\{b,c\}$, $First_1(Ccd)=First_1(C)=\{d,e\}$: {a}, {b,c}, {d,e} diszjunktak
B: $First_1(bBb)=First_1(b)=\{b\}$, $First_1(cCc)=First_1(c)=\{c\}$: {b}, {c} diszjunktak
C: $First_1(dDd)=First_1(d)=\{d\}$, $First_1(Dd)=First_1(D)=\{e\}$: {d}, {e} diszjunktak.
D: D-nek csak egy alternatívája van.

	a	b	c	d	e	#
S	(ABC,1)	(ABC,1)	(ABC,1)	(ABC,1)	(ABC,1)	
A	(a,2)	(Bbc,3)	(Bbc,3)	(Ccd,4)	(Ccd,4)	
B		(bBb,5)	(cCc,6)			
C				(dDd,7)	(Dd,8)	
D					(e,9)	
a	pop					
b		pop				
c			pop			
d				pop		
e					pop	
#						accept

(acedcded#, S#, λ) $\overset{(ABC,1)}{\vdash}$ (acedcded#, ABC#, 1)) $\overset{(a,2)}{\vdash}$ (acedcded#, aBC#, 12)) $\overset{pop}{\vdash}$ (cedcded#, BC#, 12)

$\overset{(cCc,6)}{\vdash}$ (cedcded#, cCcC#,126) $\overset{pop}{\vdash}$ (edcded#, CcC#, 126) $\overset{(Dd,8)}{\vdash}$ (edcded#, DdcC#, 1268) $\overset{(e,9)}{\vdash}$

(edcded#, edcC#, 12689) $\overset{pop}{\vdash}$ (dcded#, dcC#, 12689) $\overset{pop}{\vdash}$ (cded#, cC#, 12689)) $\overset{pop}{\vdash}$ (ded#,C#,12689) $\overset{(dDd,7)}{\vdash}$

(ded#, dDd#,126897)) $\overset{pop}{\vdash}$ (ed#,Dd,126897)) $\overset{(e,9)}{\vdash}$ (ed#,ed#,1268979) $\overset{pop}{\vdash}$ (d#,d#,1268979) $\overset{pop}{\vdash}$

(#,#,1268979) $\overset{accept}{\vdash}$ (λ, λ,1268979).

$S \overset{1}{\Rightarrow} ABC \overset{2}{\Rightarrow} aBC \overset{6}{\Rightarrow} acCcC \overset{8}{\Rightarrow} acDdcC \overset{9}{\Rightarrow} acedcC$

$\overset{7}{\Rightarrow} acedcdDd \overset{9}{\Rightarrow} acedcded$

1268979

$S \rightarrow (1)ABC$
$A \rightarrow (2)\ a \mid (3)Bbc \mid (4)\ Ccd$
$B \rightarrow (5)bBb \mid (6)cCc$
$C \rightarrow (7)dDd \mid (8)Dd$
$D \rightarrow (9)\ e$

Tétel. A G nyelvtan akkor és csak akkor LL(1) nyelvtan, ha minden A nemterminális szimbólum $A \rightarrow \gamma \mid \delta$ helyettesítési szabályaira
$First_1(\gamma FOLLOW_1(A)) \cap First_1(\delta FOLLOW_1(A)) = \emptyset$.

$FOLLOW_k(\beta)= \{x \mid S \Rightarrow^* \alpha\beta\gamma$ és $x \in Firstk(\gamma)\}$,
és ha
$\lambda \in FOLLOW_k(\beta)$, akkor legyen $FOLLOW_k(\beta) <= FOLLOW_k(\beta) \setminus \{\lambda\} \cup \{\#\}$
$(\alpha, \beta, \gamma \in (V_N \cup V_T)^*)$, $x \in V_T^*)$.
A $FOLLOW_1(A)$ tehát azokat a terminálisokat tartalmazza, melyek az $S \Rightarrow^* \alpha A\gamma \Rightarrow^* \alpha Aw$ levezetésben közvetlenül A mögött állnak.

$FOLLOW_1(x),\ x \in V_N \cup V_T$ kiszámítása:

1. Minden $a \in V_T$ és $i \geq 0$ esetén $H'_i(a) = \{a\}$,
2. Legyen $H'_0(S) = \{\lambda\}$ és minden $A \in V_N \setminus \{S\}$ esetén $H'_0(A) = \emptyset$
3. Ha minden $A \in V_N$ esetén $H'_0(A), H'_1(A), \ldots, H'_i(A)$ mind ismertek,
   akkor $H'_{i+1}(A) = H'_i(A) \cup \{x \in V_T \cup \{\lambda\} \mid x \in FIRST_1(\beta H'_i(B)) \mid B \to \alpha A \beta \in H,\ \alpha,\beta \in (V_N \cup V_T)^*\}$
   vagyis $H_{i+1}(A) = H'_i(A) \cup \{FIRST_1(\beta H'_i(B)) \mid B \to \alpha A \beta \in H,\ \alpha,\beta \in (V_N \cup V_T)^*\}$
   3.1 $\{FIRST_1(\beta H'_i(B)) \mid B \to \alpha A \beta \in H,\ \alpha,\beta \in (V_N \cup V_T)^*\}$halmaz elemeinek kiszámítása:
   $FIRST_1(\beta H'_i(B)) = FIRST_1(\beta) \cup H'_i(B)$ ha $\lambda \in FIRST_1(\beta)$ ($FIRST_1(\beta)$ kiszámítását ld előbb)
   $FIRST_1(\beta H'_i(B)) = FIRST_1(\beta)$ ha λ nem eleme $FIRST_1(\beta)$-nek
4. Ha minden $A \in V_N$ esetén $H'_i(A) = H'_{i+1}(A)$ akkor minden $A \in V_N$ esetén
   $FOLLOW_1(A) = H'_i(A)$ és kész vagyunk, különben $i+1 \to i$ és ugrás 3-ra

Legyen például
$S \to ABC$
$A \to a \mid Bbc \mid Ccd$
$B \to bBb \mid cCc$
$C \to dDd \mid Dd$
$D \to e$

	S	A	B	C	D
$FIRST_1$	a,b,c,d,e	a,b,c,d,e	b,c	d,e	e

	S	A	B	C	D
H'_0	#				
H'_1	#	b,c	d,e,b	# ,c	d
H'_2	#	b,c	d,e,b	# ,c	d
$FOLLOW_1$	#	b,c	d,e,b	# ,c	d

Pld $H_1'(C)$:
λ az $S \to ABC$
és $\lambda \in H'_0(S)$

$FIRST_1(x)$, $x \in V_N \cup V_T$ kiszámítása **ismét**:

1. Minden $a \in V_T$ és $i \geq 0$ esetén $H_i(a) = \{a\}$,
2. Legyen minden $A \in V_N$ esetén $H_0(A) = \{a \in V_T \mid A \to a\alpha \in H\}$
3. Ha minden $A \in V_N$ esetén $H_0(A), H_1(A), \ldots, H_i(A)$ mind ismertek, akkor $H_{i+1}(A) = H_i(A) \cup \{a \in H_i(X) \mid A \to X\alpha \in H, X \in V_N \cup V_T\}$
4. Ha minden $A \in V_N$ esetén $H_i(A) = H_{i+1}(A)$ akkor minden $A \in V_N$ esetén $FIRST_1(A) = H_i(A)$ és kész vagyunk, különben $i+1 \to i$ és ugrás 3-ra

Legyen például
$S \to TE'$,
$E' \to +TE' \mid \lambda$,
$T \to FT'$
$T' \to *FT' \mid \lambda$,
$F \to (S) \mid i$
nyelvtanhoz tartozó LL(1) elemző

	S	**E'**	**T**	**T'**	**F**
H_0		+, λ		*, λ	(,i
H_1		+, λ	(,i	*, λ	(,i
H_2	(,i	+, λ	(,i	*, λ	(,i
H_3	(,i	+, λ	(,i	*, λ	(,i
$FIRST_1$	**(,i**	**+, λ**	**(,i**	***, λ**	**(,i**

$FIRST_1(\beta)$, $\beta \in (V_N \cup V_T)^*$ kiszámítása **ismét**: 1. Legyen először $FIRST_1(\beta) = \{\emptyset\}$

2. ha $\beta = \lambda$ akkor $FIRST_1(\beta) = \{\lambda\}$ és készen vagyunk
3. ha β soron következő (először az első) betűje egy $a \in V_T$ terminális akkor $FIRST_1(\beta) = FIRST_1(\beta) \cup \{a\}$ és készen vagyunk
4. ha β soron következő (először az első) betűje egy $A \in V_N$ nemterminális akkor $FIRST_1(\beta) = FIRST_1(\beta) \cup FIRST_1(A)$ és

4.1 ugrás 3-ra ha $\lambda \in FIRST_1(A)$ és A nem az utolsó betűje β-nak
4.2 készen vagyunk ha λ nem eleme $FIRST_1(A)$-nak vagy A utolsó betűje β-nak

FOLLOW$_1$ (x), $x \in V_N \cup V_T$ kiszámítása **ismét:**

1. Minden $a \in V_T$ és $i \geq 0$ esetén $H'_i(a) = \{a\}$,
2. Legyen $H'_0(S)=\{\lambda\}$ és minden $A \in V_N \setminus \{S\}$ esetén $H'_0(A)=\emptyset$
3. Ha minden $A \in V_N$ esetén $H'_0(A)$, $H'_1(A)$, …, $H'_i(A)$ mind ismertek,
   akkor $H'_{i+1}(A)= H'_i(A) \cup \{x \in V_T \cup \{\lambda\} \mid x \in FIRST_1(\beta H'_i(B)) \mid B \to \alpha A \beta \in H,\ \alpha,\beta \in (V_N \cup V_T)^*\}$

vagyis $H_{i+1}(A)= H'_i(A) \cup \{FIRST_1(\beta H'_i(B)) \mid B \to \alpha A \beta \in H,\ \alpha,\beta \in (V_N \cup V_T)^*\}$

3.1 $\{FIRST_1(\beta H'_i(B)) \mid B \to \alpha A \beta \in H,\ \alpha,\beta \in (V_N \cup V_T)^*\}$ halmaz elemeinek kiszámítása:

$FIRST_1(\beta H'_i(B)) = FIRST_1(\beta) \cup H'_i(B)$ ha $\lambda \in FIRST_1(\beta)$ ($FIRST_1(\beta)$ kiszámítását ld előző lap)

$FIRST_1(\beta H'_i(B)) = FIRST_1(\beta)$ ha λ nem eleme $FIRST_1(\beta)$-nek

4. Ha minden $A \in V_N$ esetén $H'_i(A) = H'_{i+1}(A)$ akkor minden $A \in V_N$ esetén

$FOLLOW_1(A) = H'_i(A)$ és kész vagyunk, különben $i+1 \to i$ és ugrás 3-ra

Legyen például
$S \to TE'$,
$E' \to +TE' \mid \lambda$,
$T \to FT'$
$T' \to *FT' \mid \lambda$,
$F \to (S) \mid i$

	S	E'	T	T'	F
FIRST$_1$	(,i	+, λ	(,i	*, λ	(,i

	S	E'	T	T'	F
H'_0	#				
H'_1	#,)	#	#,+		*,#
H'_2	#,)	#,)	#,+,)	#,+	*,#,
H'_3	#,)	#,)	#,+,)	#,+,)	*,#,+,)
H'_4	#,)	#,)	#,+,)	#,+,)	*,#,+,)
FOLLOW$_1$	#,)	#,)	#,+,)	#,+,)	*,#,+,)

Pld $H_1'(E')$: β értéke az
$S \to TE'$
Szabályban λ
és $\lambda \in H'_0(S)$

Fordítóprogram
FORD01

Készítsük el az S → TE', E' → +TE', E' → λ, T → FT', T' → *FT', T' → λ, F → (S), F → i nyelvtanhoz tartozó LL(1) elemző táblázatot!

Először besorszámozzuk: S →(1)TE', E' → (2) +TE' | (3) λ, T →(4) FT', T' → (5) *FT' | (6) λ, F → (7) (S) | (8) i

	First	Follow
S	{(,i}	{#,)}
E'	{λ,+}	{#,)}
T	{(,i}	{#,+,)}
T'	{λ,*}	{#,+,)}
F	{(,i}	{#,+,*,)}

$\mathrm{FOLLOW}_k(\beta) = \{x \mid S \Rightarrow^* \alpha\beta\gamma$ és $x \in \mathrm{First}k(\gamma)\}$,
és ha
$\lambda \in \mathrm{FOLLOW}_k(\beta)$, akkor legyen $\mathrm{FOLLOW}_k(\beta) = \mathrm{FOLLOW}_k(\beta) \setminus \{\lambda\} \cup \{\#\}$ $(\alpha, \beta, \gamma \in \in (V_N \cup V_T)^*)$, $x \in V_T{}^*)$.
A $\mathrm{FOLLOW}_1(A)$ tehát azokat a terminálisokat tartalmazza, melyek az $S \Rightarrow^* \alpha A\gamma \Rightarrow^* \alpha Aw$ levezetésben közvetlenül A mögött állnak.

Készítsük el az elemző táblázatot! Mint látjuk, nincs ütközés, így a nyelvtan LL(1).

	(	)	*	+	i	#
S	(TE', 1)				(TE', 1)	
E'		(λ, 3)		(+TE' ,2)		(λ, 3)
T	(FT' ,4)				(FT', 4)	
T'		(λ,6)	(*FT' ,5)	(λ,6)		(λ,6)
F	((S),7)				(i,8)	

i+i *i∈ L(G) ?

a táblázat:

	(	)	*	+	i	#
S	(TE', 1)				(TE', 1)	
E'		(λ, 3)		(+TE' ,2)		(λ, 3)
T	(FT' ,4)				(FT', 4)	
T'		(λ,6)	(*FT' ,5)	(λ,6)		(λ,6)
F	((S),7)				(i,8)	

a grammatika:

S →(1)TE', E' → (2) +TE' | (3) λ,
T →(4) FT',T' → (5) *FT' | (6) λ,
F → (7) (S) | (8) i

i+i *i∈ L(G) ?

LL(1) elemzés:

(TE',1) (FT',4) (i,8) pop (λ,6)
(i+i*i #, S#, λ)) ⊢ (i+i*i#,TE'#,1)) ⊢ (i+i*i#,FT'E'#,14)) ⊢(i+i*i#,iT'E'#,148)) ⊢(+i*i#,T'E'#,148)) ⊢

(+TE',2) pop (FT',4) (i,8) pop
⊢(+i*i#, E'#,1486) ⊢(+i*i#, +TE'#,14862) ⊢(i*i#,TE'#,14862) ⊢(i*i#,FT'E'#,148624) ⊢ (i*i#,iT'E'#,1486248) ⊢

(*FT',5) pop (i,8) po
⊢ (*i#,T'E'#,1486248) ⊢ (*i#,*FT'E'#,14862485) ⊢(i#,FT'E'#,14862485) ⊢ (i#,iT'E'#,148624858) ⊢

(λ,6) (λ,3) accept
(#,T'E'#,148624858) ⊢ (#,E'#,1486248586) ⊢ (#,#,14862485863) ⊢ OK

baloldali levezetés:

1 4 8 6 2 4 8 5 8 6
S=>TE'=> FT'E'=> iT'E'=> iE'=> i+TE'=> i+FT'E'=>i+iT'E'=> i+i *FT'E'=> i+i*iT'E'=>
3
i+i*iE'=> i+i*i

LR(k) elemzés

A modern elemzők LR(k) elemzők, ugyanis az LL(k) elemzők komolyabb programnyelvek esetén nem alkalmazhatóak. Minden k és minden LR(k) nyelvtan esetén érvényes, hogy van vele ekvivalens LR(1) nyelvtan.
LR(k) elemzés: visszaléptetés nélküli léptetés-redukálás típusú elemzés.

Egy $G=(V_N,V_T,S,H)$ nyelvtanhoz tartozó kiegészített nyelvtanon értjük a $G'=(V_N \cup \{S'\}, V_T, S',H \cup \{S' \to S\})$ nyelvtant.

Sorszámozzuk meg a helyettesítési szabályokat, az $S' \to S$ szabály legyen a nulladik szabály. Így, ha redukáláskor a nulladik szabályt kell alkalmazni, akkor ez az elemzés végét, és az elemzett szöveg szintaktikus helyességét fogja jelenteni. Megjegyezzük, hogy ha az eredeti S kezdőszimbólum nem szerepel egyik helyettesítési szabály jobb oldalán sem, akkor az $S' \to S$ kiegészítésre nincs is szükség. Az általánosság kedvéért azonban az LR(k) tulajdonságot csak kiegészített nyelvtanokra értelmezzük.

Egy G' kiegészített nyelvtan LR(k) nyelvtan ($k \geq 0$), ha a következő tulajdonságú bármely két
$S' \Rightarrow^* \alpha Aw \Rightarrow \alpha\beta w$,
$S' \Rightarrow^* \gamma Bx \Rightarrow \gamma\delta x = \alpha\beta y$
($A,B \in V_N$, $x, y,w \in V_T^*$, $\alpha, \beta, \gamma, \delta \in (V_N \cup V_T)^*$) levezetésre
$\mathrm{FIRTS}_k(w) = \mathrm{FIRST}_k(y)$ esetén $\alpha = \gamma$, $A = B$ és $x = y$.
(Ekkor $S' \Rightarrow^* \gamma Bx \Rightarrow \gamma\delta x = \alpha\beta y$ -ből $S' \Rightarrow^* \alpha Ax \Rightarrow \alpha\delta x = \alpha\beta y = \alpha\beta x$, $\mathrm{FIRTS}_k(w) = \mathrm{FIRST}_k(x)$ miatt $\delta=\beta$)

Tétel. Minden LL(k) nyelvtan LR(k) nyelvtan, de létezik olyan LR(k) nyelvtan, amelyik nem LL(k') nyelvtan egyetlen $k' \geq 0$ esetén sem.

Tétel. Minden LR(k) ($k > 1$) nyelvtanhoz létezik vele ekvivalens LR(1) nyelvtan.

LR(0) elemzés

Az elemzéshez fel kell építeni egy elemző táblázatot. A táblázat felépítéséhez el kell készíteni egy determinisztikus véges automatát. A táblázat tulajdonképpen az automata állapotátmenet táblázata. Az automata felépítésének megértéséhez a következő fogalmakra van szükségünk: LR(0) elem, closure függvény, read függvény. Legyen $A \rightarrow \alpha\beta$ a grammatika egy szabálya. Ekkor a grammatika egy LR(0) eleme $A \rightarrow \alpha.\beta$ ($A \in V_N$, $\alpha, \beta \in (V_N \cup V_T)^*$). A pont egy metaszimbólum, azaz nem tartozik a grammatikához. Egy LR(0) elem lényegében egy pontozott szabály. Ha egy szabály jobb oldalán n darab szimbólum áll, akkor ehhez a szabályhoz n + 1 LR(0) elem képezhető. A pontot a későbbiekben úgy értelmezhetjük, hogy a pont előtti részt már elemeztük, és a pont utáni rész elemzése még hátra van.

Legyen I egy grammatika LR(0) elemeinek halmaza (azaz a pontozott szabályok halmaza). Ekkor closure(I) a következő elemeket tartalmazza:

1. I minden eleme eleme closure(I)-nek is
2. Ha $A \rightarrow \alpha.B\beta \in$ closure(I) és $B \rightarrow \gamma \in H$, akkor $B \rightarrow .\gamma \in$ closure(I) .
3. Az előző szabályt addig alkalmazzuk, amíg closure(I) változik.

A closure függvényt egy grammatika LR(0) elemeinek I halmazán értelmezzük. Jelentése: ha $A \rightarrow \alpha.B\beta \in$ closure(I), akkor α-t már elemeztük, és ekkor closure(I) az összes várható inputot leíró LR(0) elemek halmaza, mivel ekkor $B\beta$-ból levezethető szimbólumsorozatok várhatóak inputként, és pontosan ezek a halmaz elemei.

A read(I,X) függvény definíciójában legyen I ismét egy grammatika LR(0) elemeinek halmaza. Ekkor read(I,X) elemei:

1. ha $A \to \alpha.X\beta \in I$, akkor closure($\{A \to \alpha X.\beta\}$) minden eleme legyen eleme read(I,X) nek
2. az előző szabállyal addig bővítjük a halmazt, amíg lehet.

A fenti függvények segítségével előállítunk egy olyan determinisztikus véges automatát, ami a grammatika által generált nyelv mondatformáinak nyelét ismeri fel. Ha van egy ilyen automatánk, akkor megtalálhatjuk a nyeleket, amiket redukálnunk kell az egyes lépésekben. Az automata állapotainak előállítása előtt a grammatikához hozzáveszünk egy S′ nemterminálist, ami eddig még nem szerepelt a nyelvben, és a grammatika kezdőszimbóluma S′ lesz, továbbá a helyettesítési szabályokhoz hozzávesszük az $S' \to S$ szabályt. Ezután az automata állapotainak előállítása a következő leírás alapján történik:

• Legyen az első előállított halmaz I_0. Az I_0-t számítsuk a következőképpen:
I_0 = closure($\{S' \to S\}$)
• Ezután a nyelvtan egy X szimbólumára (terminális és nemterminális) számítsuk ki a read(I_0,X) halmazt. Ha olyan halmazt kapunk, ami nem üres, és nem egyezik meg I_0-lal, akkkor legyen ez a halmaz I_1. Ezt a számítást végezzük el a nyelvtan minden X szimbólumára, és ha az előzőektől különböző, nem üres halmazt kapunk, akkor legyen az a következő halmaz a sorozatban (indexe eggyel nagyobb, mint az előzőé).
• Legyen az alőző lépésben előállított halmazaink száma m. Ekkor minden I_k ($k = 1, . . . ,m$) -ra hajtsuk végre az előző lépést addig, amíg kapunk új halmazt, ami az előzőektől különbözik, és nem üres.

A fent előállított halmazokat a G LR(0)-kanonikus halmazainak nevezzük. A kanonikus halmazok segítségével elkészíthetjük az elemzőt. Legyen a kanonikus halmazok száma n. Ekkor az előállított automata állapotainak halmaza legyen {1, . . . , n}, és az i-edik állapotnak az i-edik halmazt feleltetjük meg. Két féle állapota van az automatának:

- Redukáló állapot Azokban az állapotokban, amelyeknek megfelelő halmaz csak olyan pontozott szabályokat tartalmaz, amiknek a végén van a pont, redukálást kell végrehajtanunk. Ezek az automata végállapotai.
- Léptető állapot A többi állapotban léptetést kell végrehajtanunk. Ha az i és j állapot esetén I_j = read(I_i,X), akkor az automata az X szimbólum hatására az i állapotból a j állapotba megy át.

Mostmár felépíthetjük az elemző táblázatot az automata alapján. A táblázat sorai az állapotokkal lesznek felcímkézve (azaz a táblázat n soros). Az első oszlop felirata action, a többi oszlop címkéje pedig egy szimbólum (minden terminálishoz és nemterminálishoz tartozik egy oszlop).

Az action oszlop tartalma s, r vagy accept. Az s szimbólum azt jelenti, hogy léptetésre van szükség. Ekkor a táblázat megfelelő oszlopában egy állapot sorszáma szerepel: annak az állpotnak a sorszáma, amibe az aktuális állapotból és az aktuális szibólum hatására az automata kerül. Ha az action oszlopban r áll, akkor redukcióra van szükség. Ekkor az r után egy szám is áll, ami azt mutatja meg, hogy hányadik szabály alapján redukálunk. Az action oszlop accept tartalma azt jelenti, hogy elfogadó állapotban vagyunk, azaz az elemzés sikeres.

A táblázatot a fentiek alapján kell kitölteni. Az üresen maradó helyekre az error szöveg kerül. Az action oszlopban egyetlen hely sem marad majd üresen, mivel vagy léptetünk, vagy redukálunk, olyan nincs, hogy nem csinálunk semmit.

Az elemző működésének állapotát egy kettőssel írhatjuk le. A kettős első eleme egy verem, ami párokat tartalmaz: a pár első eleme egy szimbólum, második pedig egy állapot sorszáma. A verem kezdeti értéke: #0. A kettős második eleme az inputszó eddig még fel nem dolgozott része. Kezdőértéke w#, ahol w = a1a2 . . . al az inputszó. A kettős kezdeti tartalma tehát: (#0,w#). Az elemzést itt is állapot átmenetekkel adjuk meg. A végrehajtandó műveletet mindig a verem tetején lévő i_k állapotsorszám határozza meg, és a végrehajtandó műveletet a táblázat action oszlopának i_k-adik sorából olvashatjuk ki. A lehetőségek:

- Ha action[i_k] = s, akkor léptetés következik. Ekkor az input szó következő szimbóluma és az új állapot sorszáma ($i_j = M(i_k, a)$) a verembe kerül. Az állapotátmenet tehát $(\#0 \ldots Y_k i_k, ay\#) \vdash (\#0 \ldots Y_k i_k a i_j , y\#)$.
- Ha action[i_k] = r_j, akkor redukcióra van szükség, és ehhez a j-edik szabályt kell használni. Legyen ez a szabály $A \rightarrow \alpha$. Ekkor a verem annyi sorát kell törölni, ahány szimbólumból α áll. Ezután meghatározzuk, hogy i_k állapotból melyik állapotba megy át az elemző, és ezt az állapotot A-val együtt a verembe írjuk.
- Ha az action oszlop tartalma accept, akkor sikeres az elemzés.
- Ha az oszlop tartalma error, akkor az elemző szintaktikai hibát észlelt.

Az elemzést addig folytatjuk, amíg van lehetséges átmenet.

Példa: $S \rightarrow aAd$, $A \rightarrow bA \mid c$

1. Új S' nemterminális szimbólumot és új $S' \rightarrow S$ szabályt veszünk fel (kiegészített nyelvtan)
2. Besorszámozzuk 0-tól a szabályokat: (0) $S' \rightarrow S$, (1) $S \rightarrow aAd$, (2) $A \rightarrow bA$, (3) $A \rightarrow c$
3. LR(0) kanonikus elemei és az elemző automata meghatározása:

I_0 = closure([$S' \rightarrow .S$])= ([$S' \rightarrow .S$], [$S \rightarrow .aAd$])

I_1 =read(I_0 ,S) = closure([$S' \rightarrow S.$])= ([$S' \rightarrow S.$])

I_2 = read(I_0 ,a) = closure([$S \rightarrow a.Ad$])= ([$S \rightarrow a.Ad$], [$A \rightarrow .bA$],[$A \rightarrow .c$])

I_3 = read(I_2 ,A)= closure ([$S \rightarrow aA.d$])=([$S \rightarrow aA.d$])

I_4 = read(I_2 ,b)= closure ([$A \rightarrow b.A$])=([([$A \rightarrow b.A$], [$A \rightarrow .bA$],[$A \rightarrow .c$])

I_5 = read(I_2 ,c)= closure ([$A \rightarrow c.$])=([$A \rightarrow c.$])

I_6 = read(I_3 ,d)= closure ([$S \rightarrow aAd.$])=([$S \rightarrow aAd.$])

I_7 = read(I_4 ,A)= closure ([$A \rightarrow bA.$])=([$A \rightarrow bA.$])

I_8 = read(I_4 ,b)= closure ([$A \rightarrow b.A$])=
([([$A \rightarrow b.A$], [$A \rightarrow .bA$],[$A \rightarrow .c$])= I_4

I_9 = read(I_4 ,c)= closure ([$A \rightarrow .c$])=
([$A \rightarrow c.$])= I_5

álla-pot	action	GOTO					
		S	A	a	b	c	d
0	s	1		2			
1	accept						
2	s		3		4	5	
3	s						6
4	s		7		4	5	
5	r3						
6	r1						
7	r2						

Táblakitöltés: pld 1. sor action: accept
mert I_1 tartalma ([$S' \rightarrow S.$])
6. sor action: r1 (redukció) mert I_6 tartalma egy
végpontos állapot: I_6 = ([$S \rightarrow aAd.$])

4.sor A.oszlop=> 7. mert I_7 = read(I_4 ,A)
action: s (léptetés, shift)

álla-pot	ac-tion	G	O	T	O		
		S	A	a	b	c	d
0	s	1		2			
1	accept						
2	s		3		4	5	
3	s						6
4	s		7		4	5	
5	r3						
6	r1						
7	r2						

(0) S’→S, (1) S → aAd, (2) A → bA, (3) A → c

r3: I_5 =([A →c.]) c redukálva A-ra
r1: I_6 = ([S →aAd.]) aAd redukálva S-re
r2: I_7 = ([A →bA.]} bA redukálva A-ra
(a redukciók sorrendje tetszőleges)

LR(0) elemzés példa: aabbcd ∈ L(G) ?

(s,2) (s,4) (s,4)
(#0,abbcd#) ⊢ (#0a2,bbcd#) ⊢ (#0a2b4,bcd#) ⊢
(s,5) r3
(#0a2b4b4,cd#) ⊢ (#0a2b4b4c5,d#) ⊢ (#0a2b4b4A7,d#)

r2 r2 (s,6)
⊢ (#0a2b4A7,d#) ⊢ (#0a2A3,d#) ⊢ (#0a2A3d6,#)

r1 accept
⊢ (#0S1,#) ⊢ (#0,#)
Jobboldali levezetés tehát kódolva: r0r1r2r2r3(r0 : S’ => S)
(a redukciók az elemzés fordított sorrendjében)
r0 r1 r2 r2 r3
S’ => S=> aAd=> abAd => abbAd => abbcd

másik LR(0) elemzés példa: aa ∈ L(G) ?
(s,2) reject
(#0,aaa#) ⊢ (#0a2,aa#) ⊢ (2. sor s. oszlop üres és
Redukálni se lehet mert az action értéke s (léptetés) a
2. sor a. oszlopban.

LR(1) elemzés

A $w \in (V_N \cup V_T)^*$ sztringet a $G=(V_N,V_T,S,H)$ nyelvtan mondatformájának hívjuk, ha $S \Rightarrow^* w$.

Legyen a $G=(V_N,V_T,S,H)$ nyelvtannak $\alpha = \alpha_1\beta\alpha_2$ egy mondatformája (α, $\alpha 1$, $\alpha 2$, $\beta \in (V_N \cup V_T)^*$).
A β-t az α egy részmondatának nevezzük, ha van olyan $A \in V_N$ szimbólum, amelyre $S \Rightarrow^* \alpha_1 A \alpha_2$ és $A \Rightarrow^* \beta$. Az α-nak β egy egyszerű részmondata, ha a fentiekben az $A \rightarrow \beta \in H$ teljesül. Egy mondatforma legbaloldalibb egyszerű részmondatát a mondatforma nyelének nevezzük.

Az LR(k) nyelvtanokra az a jellemző, hogy az $\alpha\beta w$ mondatformában a w első szimbólumától kezdve előreolvasva k darab szimbólumot, egyértelműen meghatározható, hogy valóban β a nyél, és az, hogy ha az elemzett szó eleme a nyelvnek, akkor az $\alpha\beta w$ mondatformát az $A \rightarrow \beta$ szabállyal kell redukálni, azaz az $\alpha\beta w$ mondatforma az $\alpha A w$ mondatformára redukálható.

Legyen az $\alpha\beta x$ ($\alpha, \beta \in (V_N \cup V_T)^*$, $x \in V_T^*$) mondatforma nyele β. Ekkor az $\alpha\beta$ jelsorozat prefixeit az $\alpha\beta x$ járható prefixeinek nevezzük. Az értelmezés szerint a járható prefixek a mondatforma nyele utáni szimbólumokat nem tartalmazhatják. Így, mivel az alulról-felfelé elemzésben a feladat a mondatforma nyelének a meghatározása, ez a feladat visszavezethető a mondatforma leghosszabb járható prefixének meghatározására.

Ha a G' nyelvtan egy helyettesitési szabálya $A \to \alpha\beta$, akkor a nyelvtan LR(1)-elemén értjük az $[A \to \alpha.\beta, a]$, $(a \in T \cup \{\#\})$, kifejezést, ahol az $A \to \alpha.\beta$ -t az LR(1)-elem magjának és a –t az LR(1)-elem előreolvasási szimbólumának nevezzük.

Az előreolvasási szimbólumnak csak akkor van szerepe, ha az LR(1)-elem redukciót ír elő, azaz $[A \to \alpha., a]$ alakú. Ez azt jelenti, hogy redukciót majd csak abban az esetben szabad végrehajtani, ha az α-t, azaz a mondat nyelét az a szimbólum követi.

Egy G' nyelvtan $[A \to \alpha.\beta, a]$ LR(1)-elemét a $\gamma\alpha$ járható prefixre nézve érvényesnek hívjuk, ha $S' \Rightarrow^* \gamma Ax \Rightarrow \gamma\alpha\beta x$ $(\gamma \in (V_N \cup V_T)^*$, $x \in V_T^*$, továbbá vagy a az x első szimbóluma, vagy pedig ha $x = \lambda$, akkor $a = \#$.

Legyen a $\mathscr{H}$ halmaz egy nyelvtan egy LR(1)-elemhalmaza. Ekkor a closure($\mathscr{H}$) halmaz a következő LR(1)-elemeket tartalmazza:

1. a $\mathscr{H}$ halmaz minden eleme legyen eleme a closure($\mathscr{H}$) halmaznak is,
2. ha $[A \to \alpha.B\beta, a] \in$ closure($\mathscr{H}$) és $B \to \gamma$ a nyelvtan egy helyettesítési szabálya, akkor legyen $[B \to .\gamma, b] \in$ closure($\mathscr{H}$) minden $b \in FIRST_1(\beta a)$-ra,
3. a closure($\mathscr{H}$) halmazt a 2. pontban leirt művelettel addig kell bőviteni, ameddig az lehetséges.

Legyen a $\mathscr{H}$ halmaz egy nyelvtan egy LR(1)- elemhalmaza. Ekkor a read($\mathscr{H}$,X) ($X \in (V_N \cup V_T)$) halmaz a következő LR(1)- elemeket tartalmazza:
1. ha $[A \to \alpha.X\beta, a] \in \mathscr{H}$, akkor a closure($[A \to \alpha X.\beta, a]$) minden eleme legyen a read($\mathscr{H}$,X) halmaz eleme,
2. a read($\mathscr{H}$,X) halmazt az 1. művelettel addig kell bővíteni, ameddig az lehetséges.

Szemléletesen, a read($\mathscr{H}$,X) függvény a $\mathscr{H}$ halmaz elemeiben az X szimbólumot olvassa, a "pont" jel az eredmény halmaz elemeiben már az X jobboldalán van. Ha $\mathscr{H}$ a γ járható prefixekre nézve érvényes LR(1)-elemeket tartalmazza, akkor a read($\mathscr{H}$,X) a γX-re nézve érvényes LR(1)-elemek halmaza lesz.

A $\mathscr{H}_0, \mathscr{H}_1, \ldots, \mathscr{H}_m$ LR(1)-elemek kanonikus halmazai a következők:

- Legyen $\mathscr{H}_0 = \text{closure}([S' \to .S,\#])$,
- Ezután képezzük egy X szimbólumra a read($\mathscr{H}_0$, X) halmazt. Ha az így kapott halmaz nem üres, és nem egyezik meg a $\mathscr{H}_0$ kanonikus halmazzal, akkor legyen ez a következő kanonikus halmaz, azaz $\mathscr{H}_1$.

Ismételjük meg ezt a műveletet az összes lehetséges X terminális és nemterminális szimbólumra úgy, hogy ha olyan nem üres halmazt kapunk, amelyik nem egyezik meg egyik korabbi kanonikus halmazzal sem, akkor ez a halmaz legyen egy új kanonikus halmaz, es indexe legyen 1-gyel nagyobb, mint az eddigi maximális index.

- Ezután ismételjük meg ezt a műveletet a már korábban előállított összes kanonikus halmazra és a nyelvtan minden szimbólumára, egészen addig, amig csak új kanonikus halmazt kapunk.

Az így letrehozott $\mathscr{H}_0, \mathscr{H}_1, \ldots, \mathscr{H}_m$ halmazokat nevezzük a G nyelvtan LR(1)-kanonikus halmazainak.

Mivel egy nyelvtanra az LR(1)-elemek darabszáma véges, az LR(1)-kanonikus halmazok létrehozása biztosan véges lépésben befejeződik.

Ha egy G′ kiegészített nyelvtanhoz meghatároztuk az LR(1)-elemek $\mathscr{H}_0, \mathscr{H}_1, \ldots, \mathscr{H}_m$ kanonikus halmazait, akkor egy automata k állapotához rendeljük hozzá a $\mathscr{H}_k$ halmazt. Az automata állapotai és az LR(1)-elemek kanonikus halmazai közötti kapcsolatot a következő, az LR(1)-elemzes nagy tételének is nevezett állítás mondja ki:

A következő tétel azt mondja ki, hogy a járható prefixeket felismerő automata felépíthető a kanonikus halmazok ismeretében:

Tétel. Egy γ járható prefixre érvényes LR(1)-elemek halmaza az a $\mathscr{H}_1$ kanonikus elemhalmaz, amelyik az elemző véges determinisztikus automatájának ahhoz a k állapotához tartozik, amelyikbe az automata a kezdőállapotból a γ hatására kerül.

A járható prefixeket felismerő determinisztikus véges automata leírható egy táblázattal, ezt LR(1) elemző táblázatnak nevezzük. A táblázat sorait az automata állapotaihoz rendeljük hozzá. Az elemző táblázat két részből áll. Az első neve az action táblázat. Mivel az elemezendő szöveg szimbóluma határozza meg az elvégzendő műveletet, az action táblázatot oszlopokra bontjuk, és az oszlopokhoz a terminális szimbólumokat rendeljük.

Az action táblázat azt tartalmazza, hogy az adott állapotban, ha az oszlophoz tartozó terminális szimbólum a bemenő jel, léptetést vagy redukciót kell-e végrehajtani. A léptetés műveletét jelöljük sj-vel, ahol s a léptetést, j a léptetés utáni állapotot jelenti. A redukció jele legyen r_i, ahol i az alkalmazott helyettesítési szabály sorszáma. Mivel a nulladik szabály szerinti redukció azt jelenti, hogy elemzés befejeződött és az elemzett szöveg szintaktikusan helyes, jelöljük ezt a táblázatban az elfogad szóval.

A második rész a goto táblázat. Ebbe az az információ kerül, hogy a nemterminális szimbólumok hatására az automata egy adott állapotból melyik állapotba megy át. (A terminális szimbólumok állapot-átmeneteit az action táblázat sj bejegyzései tartalmazzák.)
Az automata állapotainak halmaza legyen a $\{0, 1, \ldots, m\}$ halmaz, az elemző táblázatok i-edik sorát a $\mathscr{H}_i$ LR(1)-elemeiből töltjük ki.

Az action táblázat i-edik sora:
- ha $[A \rightarrow \alpha.a\beta, b] \in \mathscr{H}_i$ és $\text{read}(\mathscr{H}_i, a) = \mathscr{H}_j$, akkor legyen action[i, a] = sj
- ha $[A \rightarrow \alpha., a] \in \mathscr{H}_i$ és $A \neq S'$, akkor legyen action[i, a] = rt, ahol az $A \rightarrow \alpha$ a nyelvtan t-edik szabálya,
- ha $[S' \rightarrow S.,\#] \in \mathscr{H}_i$, akkor legyen action[i,#] = elfogad.

A goto táblázat kitöltésének módszere:
- ha $\text{read}(\mathscr{H}_i, A) = \mathscr{H}_j$, akkor legyen goto[i,A] = j.

Mindkét táblázatban az üresen maradt helyeket a hiba szöveggel töltsük ki.

Tétel. A G′ kiegeszitett nyelvtan akkor es csak akkor LR(1) nyelvtan, ha a nyelvtanhoz készített kanonikus elemző táblázatok kitöltése konfliktusmentes.

Az LR(1) elemző működése a következőképpen adható meg :
Az elemző verme egy ”dupla verem”, azaz egy push vagy pop művelettel két információt írunk vagy olvasunk. A verem szimbólumpárokat tartalmaz, a párok első elemében egy terminális vagy nemterminális szimbólumot tárolunk, a második elemben pedig az automata állapotának sorszámát. A verem kezdeti tartalma legyen #0.

Az elemző állapotát egy kettőssel írjuk le, a kettős első eleme legyen a verem tartalma, a második elem pedig a bemenő szimbólumsorozat még nem elemzett része. Az elemző kezdőallapota tehát (#0, z#), ahol z az elemezendő szimbólumsorozat. Az elemzés sikeresen befejeződik, azaz az elemző a végállapotba kerül, ha a verem tartalma ismét #0, és az elemzéssel az elemezendő szimbólumsorozat végére értünk.

Tegyük fel, hogy az elemző pillanatnyi állapota a $(\#0 \ldots Y_k i_k, ay\#)$ kettőssel írható le. Ekkor az elemző következő lépését az action$[i_k, a]$ adat határozza meg.
Az állapotátmenetek a következők:

• Ha action$[i_k, a]$ = st, azaz az automata egy léptetést hajt végre, akkor a bemenet soron következő a szimbóluma és az új állapot i_t sorszáma kerüljön a verembe, azaz
$(\#0 \ldots Y_k i_k, ay\#) \to (\#0 \ldots Y_k i_k a i_t\ y\#)$.

• Ha action$[i_k, a]$ = rt, akkor a t-edik szabály, az $A \to \alpha$ szabály szerint kell redukálni. Először töröljük a verem $|\alpha|$ darab sorát, azaz $2|\alpha|$ elemét. Ezután határozzuk meg a goto táblázatból, hogy az automata a törlés után a verem tetejére kerülő állapotból az A hatására melyik állapotba kerül, majd az A szimbólumot és a meghatározott állapotsorszámot írjuk be a verembe.
$(\#0 \ldots Y_{k-r} i_{k-r} Y_{k-r+1} i_{k-r+1} \ldots Y_k i_k, y\#) \to (\#0 \ldots Y_{k-r}\ i_{k-r} A i_t, y\#)$,
ahol $|\alpha| = r$, és goto$[i_{k-r}, A] = i_t$.

• Ha action$[i_k, a]$ = elfogad, akkor az elemzés a veremből való törlés után befejeződik, az elemző az elemzett szöveget elfogadja.

• Ha action$[i_k, a]$ = hiba, akkor az elemzés befejeződik, az elemző az elemzett szövegben az a szimbólumnál egy szintaktikai hibat detektált.

Az LR(1) elemzőt gyakran kanonikus LR(1) elemzőnek is nevezik.

Az elemző algoritmus bemenő paramétere az xay elemezendő szöveg és a T elemző táblázat. Az s′ változó az elemző működését jelzi, működés közben az s′ értéke elemez, az elemzés befejezésekor O.K. vagy HIBA. Az elemző az automatának a verem tetején levő xk állapota és az a aktuális szimbólum alapján a action táblázatból meghatározzaaz elvégzendő műveletet.

Az algoritmus végeredménye ennek megfelelően az O.K. vagy HIBA jelzés, és kimenetként mindkét esetben megjelenik az elemző állapota is. szintaktikai hiba esetén az elemző állapot második elemének első szimbóluma a hiba helyét adja meg.

Jelölés: LR(1) elemek felsorolásának rövidebb leírása kedvéért **[A→ α. β , a/b] jelentése:**
[A→ α. β , a] és [A→ α. β , b] LR(1) elemek

Példa: Legyen adva egy nyelvtan:S → AA, A →aA,A →b ,
vesszük a kiegészített nyelvtanát és besorszámozzuk: (0) S' → S,(1) S → AA, (2)A →aA,(3)A →b
[S' → .S,#] egy LR(1) elem.
Erre closure([S' →.S,#])={[S' →.S,#], [S →.AA,#], [A →.aA,a/b], A →.b,a/b]}.

$\mathcal{H}_0$ =closure ([S' →.S,#])={[S' →.S,#],={[S →.AA,#], [A →.aA,a/b], A →.b,a/b]}.
$\mathcal{H}_1$ =read($\mathcal{H}_0$, S)=closure ([S' →S.,#])= {[S' →S.,#]}
$\mathcal{H}_2$ =read($\mathcal{H}_0$, A)=closure ([S →A.A,#])={ [S →A.A,#]), [A →.aA,#], [A →.b,#]}
$\mathcal{H}_3$ =read($\mathcal{H}_0$, a)=closure ([A →a.A,a/b])= {[A→a.A,a/b], [A→.aA,a/b], [A→.b,a/b]}
$\mathcal{H}_4$ =read($\mathcal{H}_0$, b)=closure ([A →b.,a/b])= {[A →b.,a/b]}
$\mathcal{H}_5$ =read($\mathcal{H}_2$, A)=closure ([S →AA.,#])= {[S →AA.,#]}
$\mathcal{H}_6$ =read($\mathcal{H}_2$, a)=closure ([A →a.A,#])= {[A→a.A,#], [A→.aA,#], [A→.b,#]}
$\mathcal{H}_7$ =read($\mathcal{H}_2$, b)=closure ([A →b.,#])= {[A →b.,#]}
$\mathcal{H}_8$ =read($\mathcal{H}_3$, A)=closure ([A →aA.,a/b])= {[A →aA.,a/b}}
read($\mathcal{H}_3$, a)= $\mathcal{H}_3$
read($\mathcal{H}_3$, b)= $\mathcal{H}_4$
$\mathcal{H}_9$ =read($\mathcal{H}_6$, A)=closure ([A →aA.,#])= {[A →aA.,#]}
read($\mathcal{H}_6$, a)= $\mathcal{H}_6$
Kapjuk: $\mathcal{H}_1$ =read($\mathcal{H}_0$, S), $\mathcal{H}_2$ =read($\mathcal{H}_0$, A), $\mathcal{H}_3$ =read($\mathcal{H}_0$, a), $\mathcal{H}_4$ =read($\mathcal{H}_0$, b), $\mathcal{H}_5$ =read($\mathcal{H}_2$, A)
$\mathcal{H}_6$ =read($\mathcal{H}_2$, a), $\mathcal{H}_7$ =read($\mathcal{H}_2$, b), $\mathcal{H}_8$ =read($\mathcal{H}_3$, A), $\mathcal{H}_9$ =read($\mathcal{H}_2$, A)

$\mathscr{H}_1$ =read($\mathscr{H}_0$, S), $\mathscr{H}_2$ =read($\mathscr{H}_0$, A), $\mathscr{H}_3$ =read($\mathscr{H}_0$, a), $\mathscr{H}_4$ =read($\mathscr{H}_0$, b), $\mathscr{H}_5$ =read($\mathscr{H}_2$, A)
$\mathscr{H}_6$ =read($\mathscr{H}_2$, a), $\mathscr{H}_7$ =read($\mathscr{H}_2$, b), $\mathscr{H}_8$ =read($\mathscr{H}_3$, A), $\mathscr{H}_9$ =read($\mathscr{H}_2$, A)

←-------action--------→ ← ----goto---→

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

például : 2 és b kereszteződésénél s7 mert read($\mathscr{H}_2$, b)= $\mathscr{H}_7$

3 és A kereszteződésénél 8 mert read($\mathscr{H}_3$, A)=b

8 és a kereszteződésénél r2 mert pont végű LR(1) eleme van, ami a 2. szabály és ez az RL(1) elem a előrenézésű

Léptetés például. 3b nél s4 van, emiatt a b-t átteszi az első verembe és 4-et ír utána

Redukció például: 8b-nél r2 van, a második szabály A →aA, a3A8 törlődik az első veremből, helyébe A kerül, s mivel törlés után 0 az állapot és 0A-nál 2 van, A után 2 íródik

például egy LR(1) elemzés:

s3 **s4** r3 **r2** s7 r3 r1

(#0,abb#) ⊢ (#0a3,bb#) ⊢(#0a3b4,b#) ⊢(#0a3A8,b#) ⊢ (#0A2,b#) ⊢(#0A2b7,#) ⊢ (#0A2A5,#) ⊢

⊢ (#0S1,#) ⊢ accept

Emlékeztetőül, a szabályok sorszáma: (0) S’ → S,(1) S → AA, (2)A →aA,(3)A →b

Ekkor az ememzés egy abb startszó esetén (l.d. előző oldal):

s3 **s4** r3 **r2** s7 r3

(#0,abb#) ⊢ (#0a3,bb#) ⊢(#0a3b4,b#) ⊢(#0a3A8,b#) ⊢ (#0A2,b#) ⊢(#0A2b7,#) ⊢

r1

⊢ (#0A2A5,#) ⊢ (#0S1,#) ⊢ accept

Jobboldali levezetés tehát kódolva: r0r1r3r2r3 (r0 : S’ => S)
(a redukciók az elemzés fordított sorrendjében szerepelnek)

r0 r1 r3 r2 r3

S’ => S=> AA=> Ab => aAb => abb

Fordítóprogram készítési technikák

Bootstrapping (Cipőkanalazás) – kettőnél több részből is állhat

A fordítóprogram készítését párhuzamosítjuk önerőből történő felemelkedéssel.

I. projekt

1. Minimális fordító (például minimális C fordító) assembly nyelven. A minimális funkcionalitású fordítót assembly nyelven készítjük el, és a célnyelvnek csak azt a részhalmazát fedjük le, amire a teljes fordító elkészítéséhez szükségünk van. (A minimális fordító csak a compiler író operációkat támogatja.)
2. ASSEMBLER. Az előző szakaszban elkészített kódot a célplatform assemblerével lefordítjuk.
3. Tárgymodul (példánkban minimális C). Az előző szakasz eredménye a fordítóprogram tárgymodulja a célplatformon.
4. SZERKESZTÉS. A szükséges tárgymodulokat összeszerkesztjük.
5. Minimális fordító. A végeredmény egy végrehajtható minimális fordító (példánkban minimális C fordító) , ami a célplatformra fordít.

II. projekt

Ezután, vagy párhuzamosan, esetleg több iterációban, a következő lépéseket hajtjuk végre:

1. Teljes fordító (példánkbanl teljes C fordító) készítése minimális nyelven (példánkban minimális C nyelven). Megírjuk a teljes fordítóprogramot. A megíráshoz csak azt a résznyelvet használjuk, amit a fent elkészített minimális compiler képes lefordítani.
2. FORDÍTÁS. A fordítást a minimális fordítóval (példánkban minimális C fordítóval) végezzük.
3. Teljes fordító tárgymodulja. (Példánkban teljes C fordító tárgymodul.) Az előző szakasz eredménye.
4. SZERKESZTÉS. A tárgymodulok összeszerkesztése.
5. Végrehajtható fordító (példánkban végrehajtható C fordító). A végeredmény egy olyan compiler, ami a célplatformon végrehajtható, és a célnyelv teljes egészét lefedi.

Cross comliper (Keresztfordító)

Tegyük fel, hogy van egy VAX C fordítóprogramunk. Feladat : ennek felhasználásával egy MacIntosh C fordítóprogram készítése.

1. Forráskód. A MAC C compiler forráskódját elkészítjük Vax C nyelven.
2. VAX C fordító. A VAX platformra fordító C compilerrel lefordítjuk a forráskódot.
3. Tárgymodul VAX-on. Az előző szakasz eredménye az, hogy elkészült a MAC C fordító tárgymodulja, ami VAX-on futtatható .
4. Szerkesztő. A VAX szerkesztőjével összeszerkesztjük a szükséges tárgymodulokat.
5. Végrehajtható Mac C fordító, ami VAX-on fut. Az előző szakasz eredményeképp előállt egy végrehajtható fordítóprogram, ami VAX platformon fut, és MAC platformra fordít. Ezt nevezzük cross-compiler-nek.

Cross compiler segítségével végrehajtható Mac-on futó C compiler előállítása Vax-on:

1. Forráskód. A MAC C fordító (ami VAX platformon fut) forráskódjával indulunk el. (Ez ugyanaz a forráskód, mint amit az első részben használtunk.)
2. Fordítás. A forráskód lefordításához a VAX platformon futó MAC C compilert használjuk.
3. MAC C fordító tárgymodul: az előző lépés eredménye a C fordító tárgymodulja MAC-en.
4. Szerkesztő a **MacIntosh-on**. (Innen kell csak hogy legyen Mac gépünk.) Az összeszerkesztéshez a MAC platform szerkesztőjét használjuk.
5. Végrehajtható C compiler MAC-en. A végeredmény az, amit várunk.

Átirányítható fordítóprogramok
Olyan fordítóprogram, ami több forrásnyelvről képes fordítani.

A modell egyik legnagyobb előnye az, hogy a front-end függetlenné tehető a célnyelvtől, ugyanis mindegy, hogy a back-end milyen kódot generál a köztes kódból, a front-end mindig a köztes nyelvre fordít. Ez az előny az, amit az átirányítható fordítóprogramok kapcsán kihasználhatunk.

Front-end back-end illesztés
A front-end a köztes nyelvre fordít, amit a back-end tud használni, azaz abból nem kell újat írnunk.
Tehát ha egy új nyelv fordítására szeretnénk alkalmassá tenni a fordítóprogramot, csak a lexikális, szintaktikai és szemantikai elemzéssel kell törődnünk, amely feladatokhoz léteznek automatizált módszerek. (például Front-end VAX c, új Back-end írása MacIntosh-ra).

Köztes kód interpretálása
Sok esetben a fordítóprogram gépfüggetlen része virtuális számítógép absztrakt nyelvére fordít. (VDL, Pascal-p kód). Ilyenkor a legtöbbször a lefordított programot a virtuális gépet szimuláló interpreterrel használjuk.

A másik lehetőség az, hogy a front-end outputjaként előálló köztes kódhoz interpretert készítünk. Az interpretert minden platfromra lefordítjuk. Ezután szintén csak front-endet kell készítenünk, ha egy új nyelvet szeretnék fordítani.

Automatikus compiler generátorok

Az automatizált eszközök a következő kategóriákba sorolhatók:

1. Lexikális elemző generátor. Ilyen eszköz a Lex és a Flex .
2. Szintaktkus elemző generátor. Szintaktikus elemzőt generál a nyelv valamilyen formális leírása alapján.
3. Szintaxis vezérelt compiler generátor. Például Yacc és Bison.
4. Attribútum kiértékelő. Ezekre az eszközökre a szemantikus elemzés miatt van szükség.
5. Automatikus kódgenerátor. A kódot generálja, és általában mintahelyettesítést használ (Yacc, HLP).

Közbülső programformák
A program fordítása során a fordító átalakítja a forráskódot egy belső formátumra, amit egyszerűbb kezelni. Ilyen belső adatszerkezet például a szintaxisfa, ami a szintaktikus elemző kimenete. A szintaxisfa a forráskód alapján épül fel. Vannak ezen kívül olyan fordítók, amik a kifejezések kiértékelését külön menetben végzik el. A kifejezéseket a kiértékeléshez egyértelmű alakra kell hozni, mivel a legtöbb esetben nem egyértelműen vannak megadva.

1. Rutishauser módszer

Teljesen zárójelezett kifejezések szintaxisfájának felépítésére szolgáló módszer. Legyen S az a kifejezés, aminek a szintaxisfáját fel szeretnénk építeni. Legyen S szimbólumaina száma n (itt szimbólum alatt a zárójeleket, operátorokat és operandusokat értjük). Legyen
$S' = \perp \| S \| \perp$, ahol $\|$ a konkatenáció jele. Így az S' 0-dik és az n + 1-edik szimbóluma $\perp$.
Ezután S' i-edik szimbólumához hozzárendeljük az n(i) számot (i = 0, ..., n + 1).
A hozzárendelés algoritmusa a következő:

Algoritmus. A Rutishauser szám hozzárendelése

1: $i \leftarrow 0$ és $n(i) \leftarrow 0$

2: $i \leftarrow i + 1$

3: **if** S’

i =⊥ **then**

4: goto 11

5: **end if**

6: if S’i = (vagy S’i egy operandus **then**

7: $n(i) \leftarrow n(i-1) + 1$ és goto 2

8: **else**

9: $n(i) \leftarrow n(i-1) - 1$

10: **end if**

11: $u(i) \leftarrow 0$

S’j−1	**S’j**	**S’j+1**	**S’j+2**	**S’j+3**
α	**x**	**Ω**	**y**	**β**
k − 1	**k**	**k – 1**	**k**	**k – 1**

1. táblázat. Jelölések a Rutishauser módszerben

A Rutishauser módszer azzal ezdődik, hogy az n(i) értékeket kiszámítjuk. Ezután megkeressük az első olyan j-t, amire n(j) maximális. Ekkor az 1.táblázatban látható jelöléseket alkalmazzuk. Az ábrán látható, hogy az S' j-edik eleméhez rendelt érték k (azaz k a maximális érték). Tudjuk, hogy ekkor x és y operandusok, egy operátor, α és β pedig vagy (és), vagy két $\perp$. Ekkor $x\ \mathbf{\Omega}\ y$ -t végrehajtjuk. Legyen A a végrehajtás eredménye, azaz $A \leftarrow x\ \mathbf{\Omega}\ y$. Ha α és β $\perp$, akkor A lesz a teljes kifejezés eredménye, azaz a szintaxisfa gyökere. Ha nem, akkor az S' kifejezésben az $x\ \mathbf{\Omega}\ y$ részt A-val helyettesítjük, A-hoz a $k - 1$ értéket rendeljük, és folytatjuk ugyanezt a műveletsort addig, amíg eljutunk odáig, hogy már csak egy szimbólumból áll a kifejezés, akkor az A lesz a gyökér.

2. Lengyel forma

A kifejezéseknek három alakja lehetséges: prefix (az operátor az operandusai előtt áll), infix (az operátor az operandusai között áll) és postfix (az operator követi az operandusait). A postfix alakot Lukasiewicz javasolta, és sok fordítóprogram erre a fordított lengyel formába (RPN -be) – azaz postfix alakra - alakítja át a kifejezéseket, mielőtt fordítana.

A postfix alak előnyei:

- az operátor közvetlenül az operandusait követi, így mindig eldönthető, hogy mik az operandusok
- az operátorok a végrehajtás sorrendjében követik egymást
- a postfix alak ekvivalens a teljesen zárójelezett alakkal abban az értelemben, hogy mindkettő egyértelmű.

Például "5 + ((1 + 2) * 4) − 3" RPN-re átírva: 5 1 2 + 4 * + 3 -

A kifejezés balról jobbra értékelődik ki a felírás sorrendjében:

Input	Művelet	Veremtartalom	Megjegyzés
5	PUSH	5	
1	PUSH	1 5	
2	PUSH	2 1 5	
+	ADD	3 5	(= 1+2)
4	PUSH	4 3 5	
*	MUL	12 5	(=3*4)
+	ADD	17	(=5+12)
3	PUSH	3 17	
-	SUB	14	(=17-3)
	EREDMÉNY	(14)	

$(a + b) * c^d - e * f$ postfix alakja: ab+cd↑*ef*-

Input	Művelet	Veremtartalom	Megjegyzés
a	PUSH	a	
b	PUSH	b a	
+	ADD	A	(=a+b)
c	PUSH	c A	
d	PUSH	d c A	
↑	POW	B A	(=$\mathbf{c^d}$)
*	MUL	C	(=A*B)
e	PUSH	e C	
f	PUSH	f e C	
*	MUL	D C	(=e*f)
-	SUB	E	(=C-D)
	EREDMÉNY	(E)	

Egy postfix alakú kifejezéshez egyszerűen elkészíthető a hozzá tartozó szintaxisfa. Az algoritmus leírása az alábbi.

Algoritmus. Egy postfix alakú S kifejezéshez elkészíti a szintaxisfát. Az átalakítás során egy V vermet használunk. (Az indexelés S esetén 1-től kezdődik, továbbá feltételezzük, hogy minden operátor kétoperandusú). Az APPLY függvény egy operátort alkalmaz két operandusra, a PUSH egy szimbólumot helyez el a verem tetején, a POP pedig kiveszi a veremből a legfelső elemet. Az algoritmus végén V egyelemű, és V -ben a szintaxisfa gyökere van.

i ← 1 {Ezzel indexeljük S-t}
while i ≤ S hossza **do**
if Si operátor **then**
T1 ← POP() {A jobb oldali operandus}
T2 ← POP() {A bal oldali operandus}
A ← APPLY (T2, Si, T1)
PUSH(A)
else
PUSH(Si)
end if
i ← i + 1
end while

ÁBRÁZOLÁS

A felépített szintaxisfát ábrázolni is kell a számítógépen. Ennek két módja az ábrázolás négyesekkel és hármasokkal.

1. Műveletek ábrázolása négyesekkel

Négyelemű rekordokkal ábrázoljuk a szintaxisfát. Minden rekor első eleme egy operátor, majd azt követi az operátor két operandusa. Az utolsó elem egy mutató, ami az adott művelet eredményére mutat. Például az

$$\mathbf{(a + b) * c^d - e * f}$$

kifejezés esetén a következő négyesekkel ábrázolhatjuk a szintaxisfát:

(+, a, b, p), (pow, c, d, q), (∗, p, q, r), (∗, e, f, s) és (−, r, s, t)

Mivel p-re és q-ra a harmadik lépés után már nincs szükség, ezek újra felhasználhatók (ekkor s-re és t-re nincs is szükség).

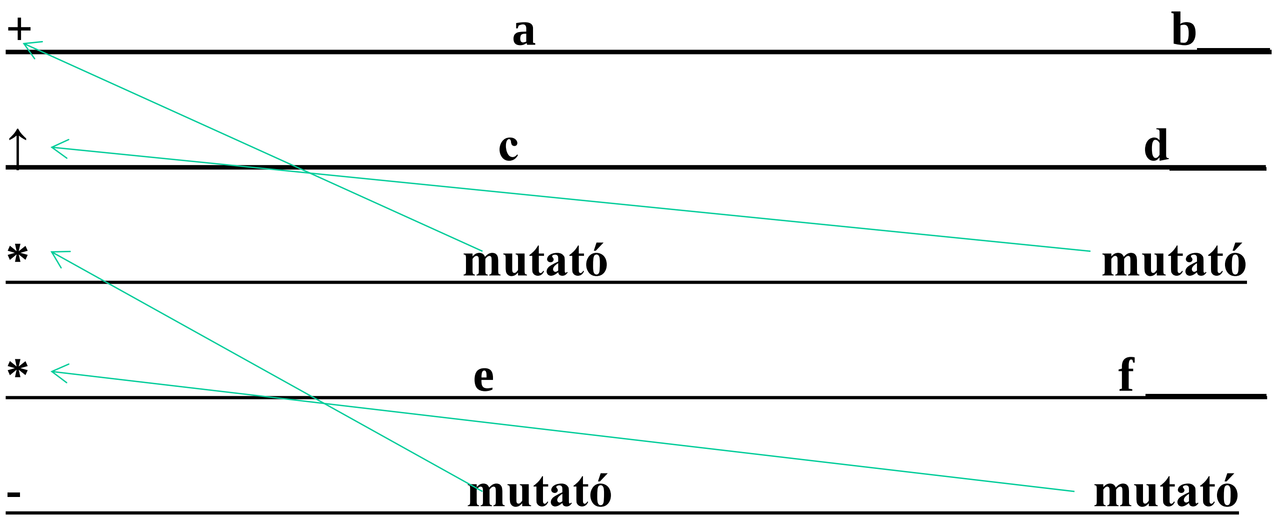
1. Műveletek ábrázolása hármasokkal

A hármasokkal történő ábrázolás csak annyiban tér el a négyesektől, hogy a negyedik elemet elhagyjuk, s helyette mutatókat alkalmazunk a megfelelő módon.

Műveletek hármasokkal. Az előző formula hármasokkal ábrázolva az alábbi ábrákon látható. azaz:

cím1	+	a	b
cím2	↑	c	d
cím3	*	mutató cím1-re	mutató cím2 -re
cím4	*	e	f
cím5	-	mutató cím3 ra	mutató cím4 -re

azaz



Példa hármasokkal történő ábrázolásra

Szemantikadefiníciós módszerek (áttekintés)

A szintaxis alapján felírt modellek jelentése:

* Műveleti (operációs) szemantika: „Programozóknak”
– Megadja, mi történik a végrehajtás (számítások) során
– Egyszerű elemekre épít: pl. állapotok, akciók

*Axiomatikus szemantika: „Helyességbizonyításhoz”
– Állítás nyelv + axiómakészlet + következtetési szabályok
– Pl. automatikus tételbizonyító rendszerekhez

* Denotációs szemantika: „Fordítóprogramokhoz”
– Szintaxis által meghatározott leképzés egy ismert doménre
Ismert matematikai domén, pl. számítási szekvencia, vezérlési gráf,
állapothalmaz, ... és ezeken definiált műveletek (összefűzés, unió, ...)

Szemantikadefiníciós módszerek

1. Verbális szemantika

Miért nem igazán jó?

- A leíró eszköz nem pontosan definiált
- Egy élő nyelv nem szemantika definiálására készült, túl hosszan és írható le a kívánt szemantika
- A szemantika szöveges megadásához a nyelv programjait jelsoroz formális szintaxis (pld.BNF azaz Bacchus-Naur forma) mellett is.

Egy nyelv precíz definíciója tisztán szöveges leírással nem adható m

2. Szemantika megadása absztrakt program alapján

2.1 Fordítóprogramos szemantika : adott egy COMP fordítóprogram

A nyelv minden mondatára egy TP = COMP (SP) függvényértéket de

Minden $X \rightarrow Y_1 \cdots Y_k$ helyettesítéshez egy fordítási idő alatti $f_X(Y_1 \cdots Y_k)$ szemantikus akciót rendelünk.

Miért nem igazán jó?

- a célnyelvnek kielégítően definiáltnak kell lennie
- csak a forrás és a célnyelv kapcsolatáról ad információt, elfedi a forrás

2.2 Operációs szemantika (elsőként : a LISP nyelv definíciója)

S programozási nyelv értelmező orientált (operációs) szemantika megadá

Lényege egy INT interpreter, mely a nyelv minden P programjához és a

Egy algoritmust definiál: érték(P(D)) = INT (P, D)

Ugyanazt a kimenő értéket adja, mintha a P rpogramot a D adatlistára haj

Minden $X \rightarrow Y_1 \cdots Y_k$ előállítási szabályhoz egy transzformáció tartozik

Egy adott llapotból egy másik állapotba viszi. (VDL, LISP)

2.3 Matematikai szemantika

a.) Axiomatikus módszer

Axioma: P{Q}R : ha P igaz a Q program változóira a Q program végre

végrehajtása normálisan befejeződik, akkor R igaz lesz a program változ

Végrehajtásra megtörtént.

Helyettesítési szabályok típusai: $\dfrac{A,\ B}{C}$ ha A és B igaz, akkor C i

$\dfrac{A,\ B \vdash C}{D}$ D-re következtetünk,

bizonyítható B -ből

b.) Fixpontos szemantika: a jelentést egy f: Input → Output függvény legkisebb fixpontja adja. (x az f függvény fixpontja, ha f(x) = x.)

Adott egy D (adat) tartomány és rajta egy $\sqsubseteq$ parciális rendezés. $x \sqsubseteq y$ azt jelöli, hogy y legalább annyi információt nyújt (legalább annyira jól definiált) mint x.

$x, y \in D_1$ és f . D1 → D2 esetén . $x \sqsubseteq y$ -ból $f(x) \sqsubseteq f(y)$ –nak kell következnie, azaz Azaz az f függvénynek monotonnak kell lennie.

3. Weingarten –féle kétszintes szemantika:

1. szint: nyelvi elemek szintaktikus reprezentációi
2. szint: szemantikai akciók

Metaszabályok: a két szint közötti átmenetet definiálják

4. Attribútumos szemantika : egy $G= (V_N, V_T, S, H)$ környezetfüggetlen nyelvtan minden X nemterminálisának egy A(X) attribútum felel meg, mely az X –nek egy specifikus környezetfüggő tulajdonságát reprezentálja és értékek egy specifikált halmazát veszi fel.

X.a : az a attribútum eleme A(X) –nek.

Egy L(G) –beli levezetési fa minden csúcsa valamely X nemterminálisra attribútumok értékeinek egy halmazával van kapcsolatban. Ezen értékek
$R(p) = \{ X_0 .a \leftarrow f(X1.b,\ldots,Xn.c)\}$
Attibútum szabályok segítségével adottak valamely
$P: X_0 \rightarrow X_1 \ldots X_n$ helyettesítési szabály esetén.
Minden szabály definiál egy $X_i .a$ attribútumot ugyanazon levezetési szabály
Nemterminálisainak X1.b,…,Xn.c attribútumaiban kifejezve.

Egy B(Xi.b,…,Xj.c) feltétel a p-ben előforduló attribútumaira vonatkozóan ugyancsak megadható. B specifikálja a környezeti feltételeket úgy, hogy akkor kell kitölteni, ha
Egy szintaktikusan korrekt mondatforma korrekt a statikus szemantikára vonatkozóan,
Azaz lefordítható.

Erre a feltételre mint konzisztens Boole-attribútumra hivatkozunk, melyet egy levezetési szabály baloldalával asszociálunk.

Emlékeztető: Az

MUNKAANYAG, NEM RÉSZE A TANANYAGNAK!!!

Adott γ járható prefix esetén a γ-ra érvényes LR(k) elemek halmazát jelöljük $\mathscr{V}_k(\gamma)$ –val.
$\mathscr{V}_k(\gamma)$ meghatározása:
Input: G grammatika és $\gamma = X_1 X_2 \ldots X_n$ szó, ahol $X_1, X_2, \ldots, X_n \in V_N \cup V_T$
Output: $\mathscr{V}_k(\gamma)$ halmaz
Módszer: kiszámoljuk sorra a $\mathscr{V}_k(\lambda)$, $\mathscr{V}_k(X_1)$, $\mathscr{V}_k(X_1X_2)$,.., $\mathscr{V}_k(X_1X_2X_n) = \mathscr{V}_k(\gamma)$ halmazokat

$\mathscr{V}_k(\lambda)$ meghatározása:

1. Inicializálás: Minden $S \rightarrow \alpha \in H$ esetén legyen $[S \rightarrow .\ \alpha\ ,\ \lambda] \in {}_k(\lambda)$
2. Lezárás: amíg bővíthető a $\mathscr{V}_k(\lambda)$, a $\mathscr{V}_k(\lambda)$ minden $[A \rightarrow\ .\ B\beta\ ,\ u]$ alakú elemére és $B \rightarrow \delta \in H$ szabályra legyen $[B \rightarrow .\ \delta\ ,\ v] \in \mathscr{V}_k(\lambda)$ minden $v \in FIRST_k\ (\beta u)$ esetén.

$\mathscr{V}_k(X_1X_2X_n)$ meghatározása: tegyük fel, hogy $\mathscr{V}_k(X_1X_2X_{n-1})$ már meghatározásra került.
(1) Léptetés: minden $[A \rightarrow \alpha\ .\ X_i\beta\ ,\ u]$ alakú $\mathscr{V}_k(X_1X_2X_{i-1})$ –beli elem esetén legyen $[A \rightarrow \alpha\ .\ X_i\beta\ ,\ u] \in \mathscr{V}_k(X_1X_2X_i)$
(2) Lezárás: mindaddig, amíg $\mathscr{V}_k(X_1X_2X_i)$ bővíthető, minden $[A \rightarrow \alpha\ .\ X_i\beta\ ,\ u]$ alakú elemére és $B \rightarrow \delta \in H$ szabályra legyen legyen $[B \rightarrow .\ \delta\ ,\ v] \in \mathscr{V}_k(X_1X_2X_i)$ minden $v \in FIRST_k\ (\beta u)$ esetén.