Programozási nyelvek és módszerek

Algoritmus

Algoritmus

Abu Abdallah Muḥammad ibn Mūsā al-Khwārizmī

- 780 – ?845
- arabul alkotó perzsa tudós
- matematikus

Kidolgozta a matematikai algoritmusok fogalmát

- néhányan a számítástechnika nagyapjának

Az algoritmus szó a nevének félrefordított latin változatából ered

Algoritmus

„Az algoritmus egyértelműen előírt módon és sorrendben végrehajtandó tevékenységek véges sorozata egy feladat megoldására, ami véges idő alatt befejeződik.”

Hétköznapi életben is: egy feladat megoldási lépései teljes részletességgel

- Feladat: Süss egy gyümölcstortát!
  - Hogy írjuk le a feladatot?
  - Mi az input?
  - Mi az output?
- Itt az algoritmus: a recept

Nem minden feladatra adható algoritmus!

Algoritmus

Megadható:

- Szövegesen, természetes nyelven
- Pszeudokóddal
- Grafikusan
  - Folyamatábrával
  - Struktogrammal
  - …
- Automatákkal vagy Turing-gép segítségével
- Táblázattal
- Programozási nyelv segítségével

Algoritmus

Komponensei:

- Szekvencia - parancsok egymásutánja
- Elágazás – valamilyen feltételtől függően alternatívák választása
- Ciklus – utasítások ismételt végrehajtása

Fontos kérdések

- Biztosan megoldja a feladatot?
  - Helyes – specifikációnak megfelel?
  - Mit jelent az, hogy megoldás?
    - Részletesen még jön!

Feladat specifikációja

Specifikáció megadása

- Szövegesen
  - Példa: számítsd ki **a** és **b** legnagyobb közös osztóját!
  - Számos információt „beleértünk”….
  - Néha félreértjük – véletlenül vagy szándékosan
    - „Találkozzunk a Keletinél a metró lépcsőjénél”
      - Pontosabb az, hogy „a Keletinél a metró új kijáratánál”?
    - „A királynőt megölni nem kell félnetek jó lesz”

Feladat specifikációja

Specifikáció megadása

- Formálisan
  - I – a lehetséges bemenetek,
    O – a lehetséges kimenetek
    végesen leírható halmaza
  - $F \subseteq I \times O$ reláció, ahol $i \in I$ és $o \in O$ esetén $(i,o) \in F$ azt jelenti, hogy F az i bemenethez hozzárendeli az o kimenetet.
  - Például:
    - $I = N \times N$
    - $O = N$
    - $F = \{((a,b),c)$, ahol $a, b, c \in N \wedge c \mid a \wedge c \mid b \wedge \forall x : x \mid a \wedge x \mid b \rightarrow c \geq x\}$
    - $c \mid a$ jelentése: a osztható c-vel.

Feladat specifikációja

Szerződés a felhasználó és az implementáló között

- Előfeltétel: állítás, ami leírja azt a feltételt, ami szükséges a feladat helyes működéséhez
- Utófeltétel: állítás, ami leírja azt a feltételt, amit a függvény teljesít a helyes végrehajtás után

Helyesség a specifikáció figyelembevételével

- A program felhasználója teljesíti az előfeltételt
- A program lefut
- A program végeztével az utófeltétel igaz lesz.

Mit kell az implementációnak teljesíteni, ha a felhasználó megsérti az előfeltételt?

Feladat specifikációja

Elő- és utófeltételekkel

- Állapottér: $\mathbb{N}\times\mathbb{N}\times\mathbb{N}$
  $a \quad b \quad c$
- Előfeltétel (Ef)
  - $a=a' \wedge b=b'$
- Utófeltétel (Uf):
  - $\text{Ef} \wedge c \mid a \wedge c \mid b \wedge \forall x : x \mid a \wedge x \mid b \rightarrow c \geq x$

Programozási nyelv

Mi a programozási nyelv?

Egy jelölés …

Eszköz

- a számítógép vezérlésére
- programozók közötti kommunikációra
- algoritmusok leírására
- magas szintű tervezésre
- a feladat bonyolultságának kezelésére

Programozási nyelvek kialakulása

Gépi kód

Assembly nyelv

- Mnemonikok, címkék használata
- Assembler

Magas szintű nyelv

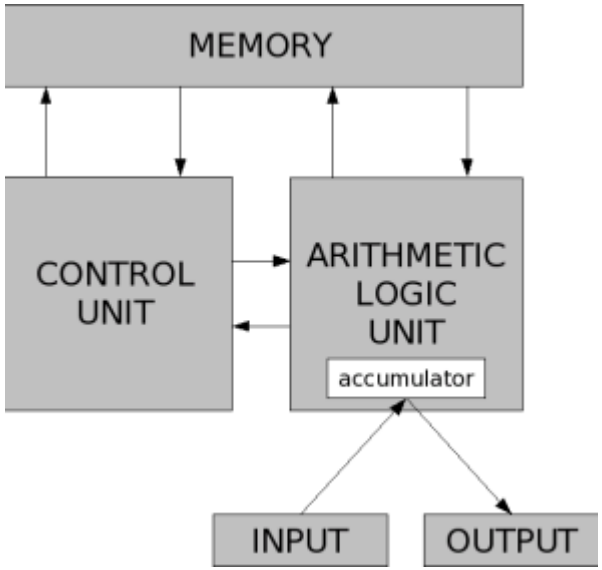
- Utasításkészlete független egy adott számítógép-architektúra utasításkészletétől
  - Végrehajtásuk előtt egy fordítóprogramnak kell fordítani.
    - Assembly kód -> assembler.
    - Közvetlenül gépi kód.

Neumann János (1903 – 1957)

Neumann–elvek a számítógépről:

- Legyen univerzális
  - bármilyen feladat megoldására használható.
- Legyen soros működésű
  - az utasításokat sorban egymás után hajtsa végre
- Legyen teljesen elektronikus működésű
- Az egyes utasítások és adatok leírásához és feldolgozásához a kettes számrendszert használja.
- Legyen belső memóriája
- Tárolt program elven működjön
  - Egy program indításakor a programot alkotó utasítások és a működéshez szükséges adatok is kerüljenek be a belső memóriába
  - Az utasításokat a memóriából kiolvasva, sorban egymás után hajtsa végre a berendezés.

Neumann János – 1946



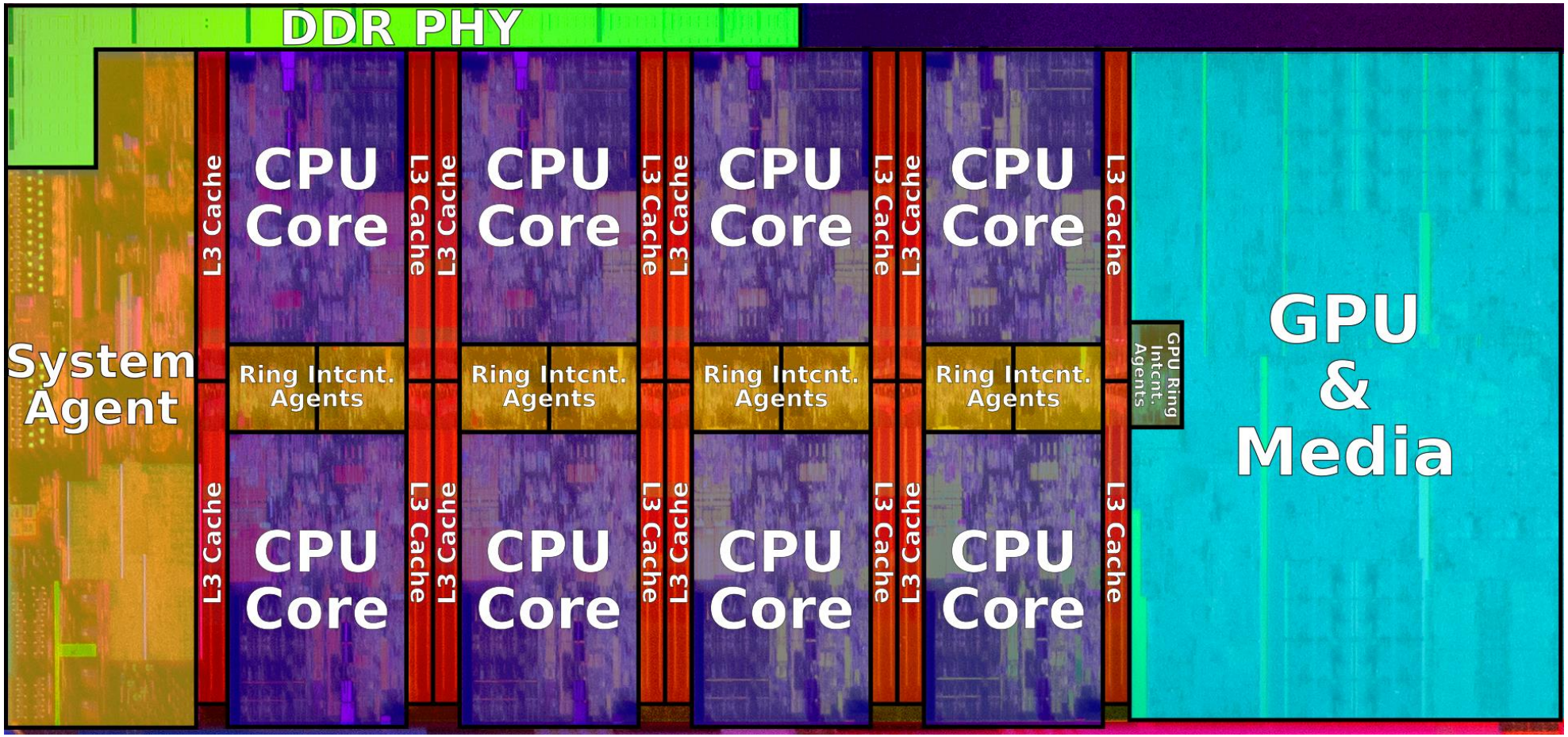
A számítógép a következő egységekből álljon:

- Központi egység:
  - vezérlés, műveletvégzés, adatmozgatás
- Memória:
  - az adatok és a programok tárolása a program végrehajtása közben
- Bemeneti-kimeneti egység:
  - kommunikáció a külvilággal, az adatbevitel- és kivitel megvalósítása

Egyszerűsített példa

HTTPS://GODBOLT.ORG/

CPU – Core i7 9700K



Program végrehajtása

Gépi kód: CPU / Vezérlő által értelmezett utasítások

```
55
48 89 e5
89 7d ec
89 75 e8
82 7d ec 00
79 03
f7 5d ec
83 7d e8 00
79 03
f7 5d e8
83 7d e8 00
7e 18
8b 45 e8
89 45 fc
8b 45 ec
99
f7 7d e8
89 55 e8
8b 45 fc
89 45 ec
eb e2
8b 45 ec
5d
c3
```

```
55
48 89 e5
b0 0a 00 00 00
bf 22 00 00 00
e8 ae ff ff ff
b8 00 00 00 00
5d
c3
0f 1f 44 00 00
```

Ember számára nehezen olvasható.
Számítógép közvetlenül végre tudja hajtani.

Alacsonyszintű nyelv

Assemby – Ember számára is olvasható kód.

```
_Z4lnkoii:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -20(%rbp)
        movl    %esi, -24(%rbp)
        cmpl    $0, -20(%rbp)
        jns     .L2
        negl    -20(%rbp)
.L2:
        cmpl    $0, -24(%rbp)
        jns     .L5
        negl    -24(%rbp)
.L5:
        cmpl    $0, -24(%rbp)
        jle     .L4
        movl    -24(%rbp), %eax
        movl    %eax, -4(%rbp)
        movl    -20(%rbp), %eax
        cltd
        idivl   -24(%rbp)
        movl    %edx, -24(%rbp)
        movl    -4(%rbp), %eax
        movl    %eax, -20(%rbp)
        jmp     .L5
.L4:
        movl    -20(%rbp), %eax
        popq    %rbp
        ret
```

```
main:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    $10, %esi
        movl    $34, %edi
        call    _Z4lnkoii
        movl    $0, %eax
        popq    %rbp
        ret
```

Ember számára olvashatóbb.
Közvetlen fordítható gépi kódra.

Magasszintű nyelv

C++ – Ember számára olvasható kód.

```
int lnko(int a, int b) {
   int tmp;
   if (a<0) a= -a;
   if (b<0) b= -b;
   while(b > 0) {
      tmp = b;
      b = a % b;
      a = tmp;
   }
   return a;
}
```

```
int main()
{
    lnko(34, 10);
    return 0;
}
```

Ezt szoktuk meg.
A nyelv szabályait definiálni kell.
Fordítóprogramok kellenek.

Modell

HTTP://WWW.PETERHIGGINSON.CO.UK/LMC/

A „Little Man Computer”

Neumann elvek az oktatásban

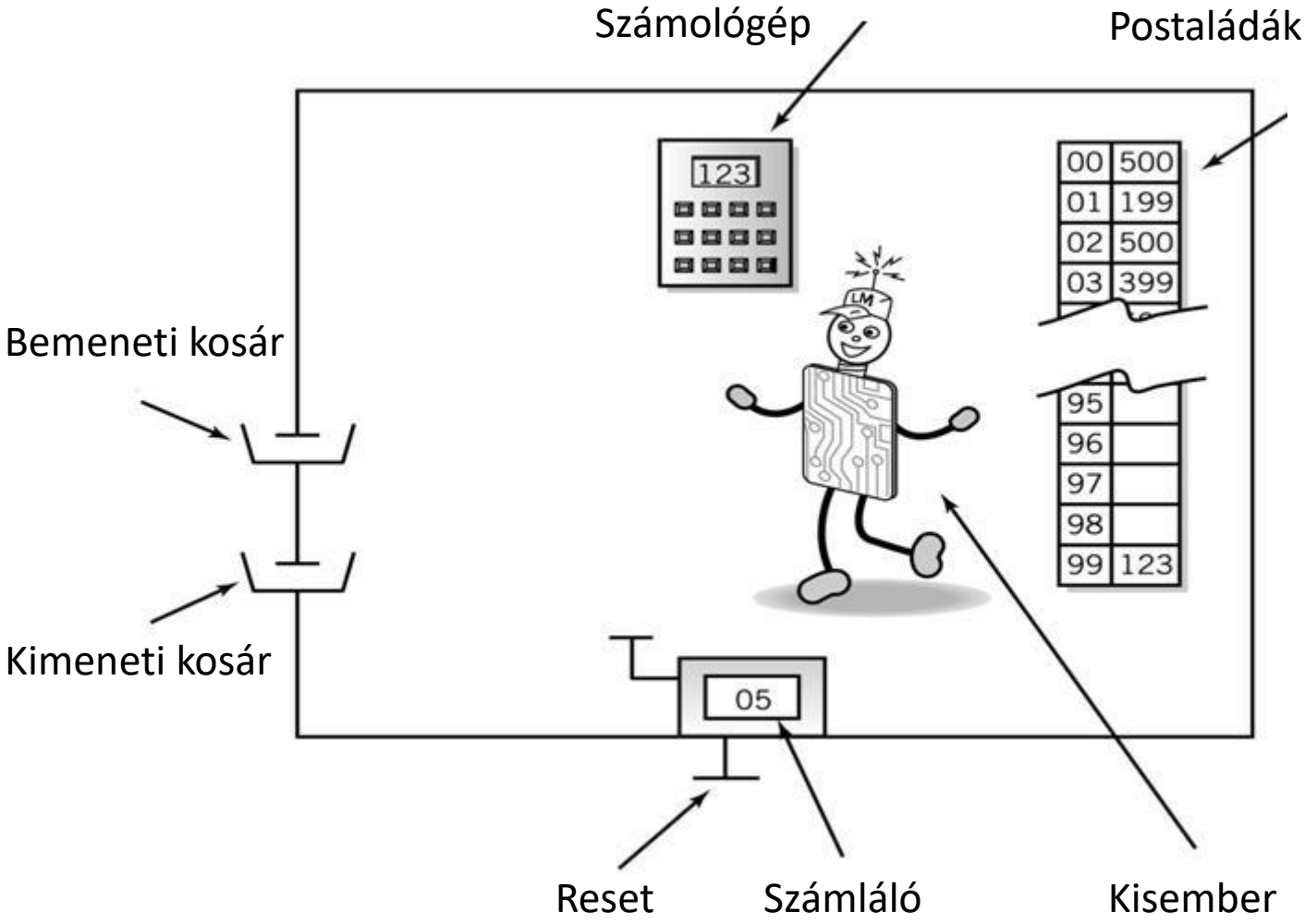
Stuart Madnick, MIT

Szoba, benne:

- Kisember
- 100 postaláda (00-99)
- Számláló (2)
- Bemenő és kimenő kosár
- Számológép

Nem foglalkozunk azzal, hogyan

- töltjük fel a postaládákat
- tegyünk adatot a bemenő kosárba



LMC elemei

Bemeneti és kimeneti kosár:

- Kisember kap/beletesz egy 3 jegyű számot tartalmazó cédulát

Számláló:

- 0 és 99 között tud számolni
- Pedál megnyomására a számlálóban tárolt érték eggyel megnő
- Értékét nullára állíthatjuk egy külső „reset” gombbal

Számológép:

- Háromjegyű decimális számokat tud kezelni.
- Képes
  - Kivonni
  - Összeadni
  - A begépelt vagy a számítás eredményeként kapott értéket eltárolni.

Postaládák

A postaládákban számokat tárolunk

- elérésük a postaláda címe (sorszáma) alapján

A címek folyamatosan egymás után jövő értékek

- 00 .. 99

A tartalom lehet

- Adat vagy
- Utasítás

Cím	Tartalom
00	522
01	123

Utasítás esetén

Műveleti kód

Operandus

- A művelet operandusa: adat vagy az adat címe

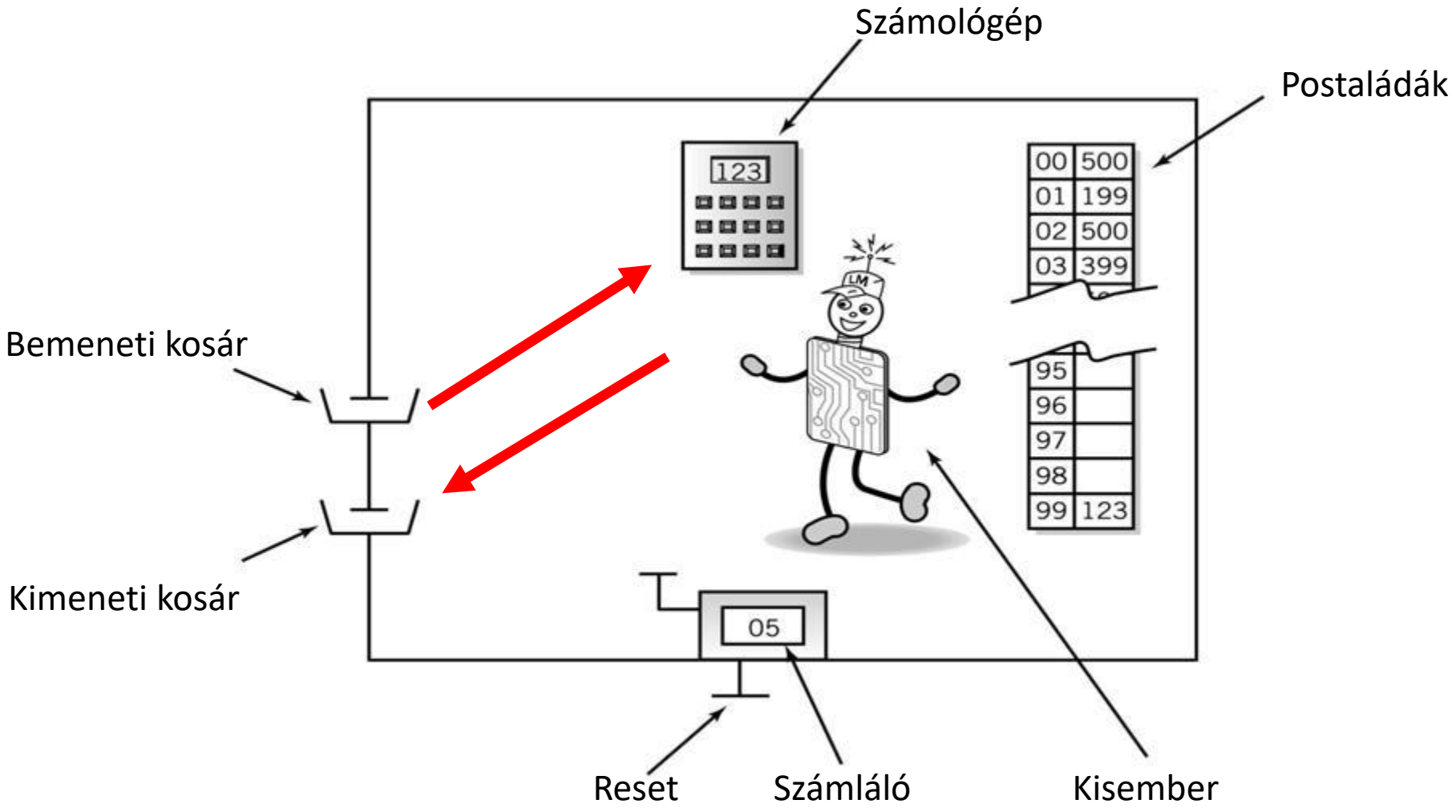
Cím	Tartalom	
	Műv. kód	Operandus

Utasításkészlet

Lehetséges utasításkészlet:

Aritmetikai	1xx	Összeadás
	2xx	Kivonás
Adat mozgató	3xx	Tárolás
	5xx	Betöltés
Input/Output	901	Beolvasás
	902	Kiírás
Vezérlés	000	Leállás

Input/Output

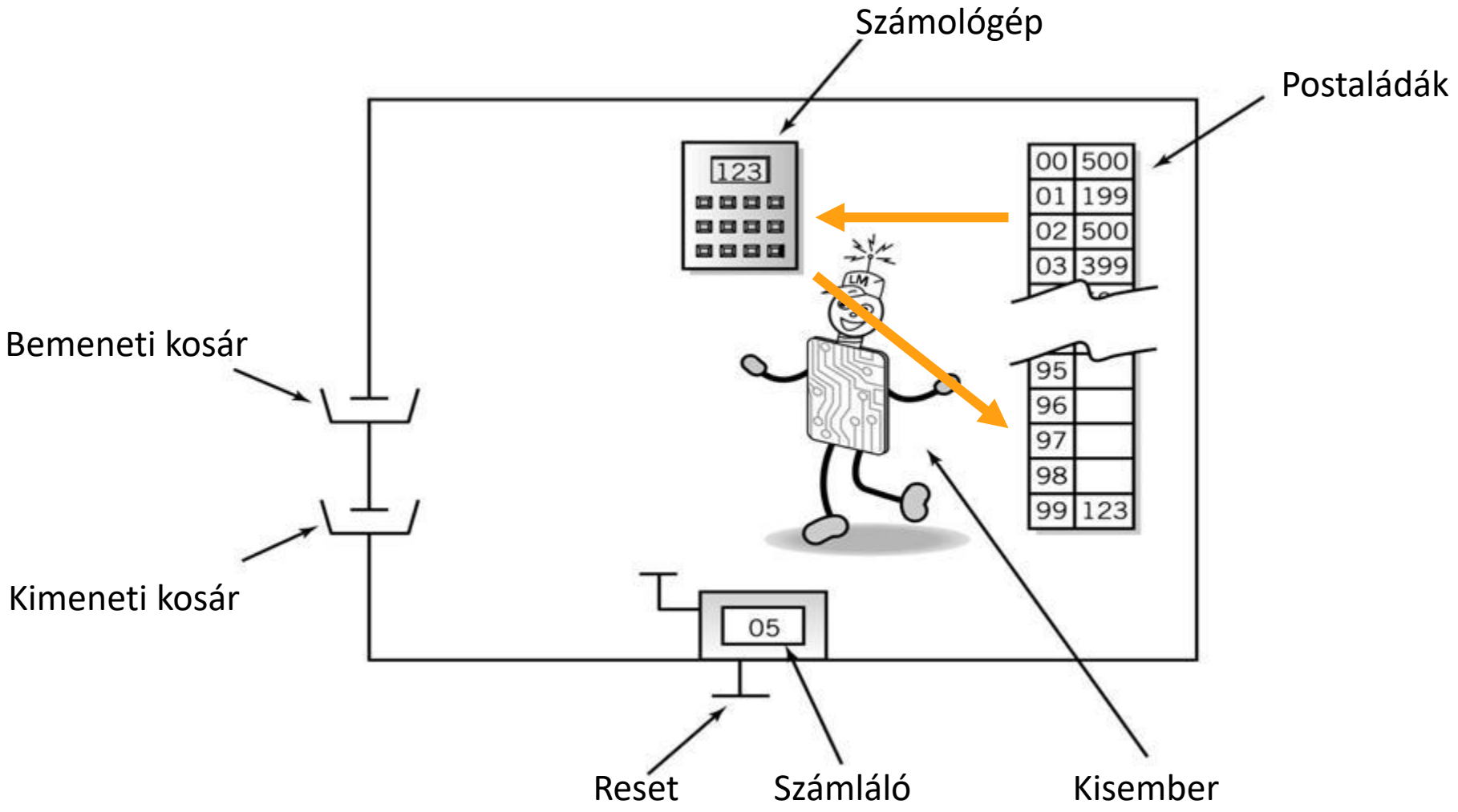


Input/Output

Adatok mozgatása a számológép és a bemeneti/kimeneti kosarak között

	Tartalom	
	Műveleti kód	Operandus (cím)
Beolvasás	9	01
Kiírás	9	02

LMC belső adatmozgatás



Belső adatmozgatás

A postaláda és a számológép között

	Tartalom	
	Műveleti kód	Operandus (cím)
Tárolás	3	xx
Betöltés	5	xx

Adatok tárolása

Az utasításokat és az adatokat tároló postaláda fiókok fizikailag azonosak

- Ezt valós esetekben elkülönítve tároljuk a memóriában

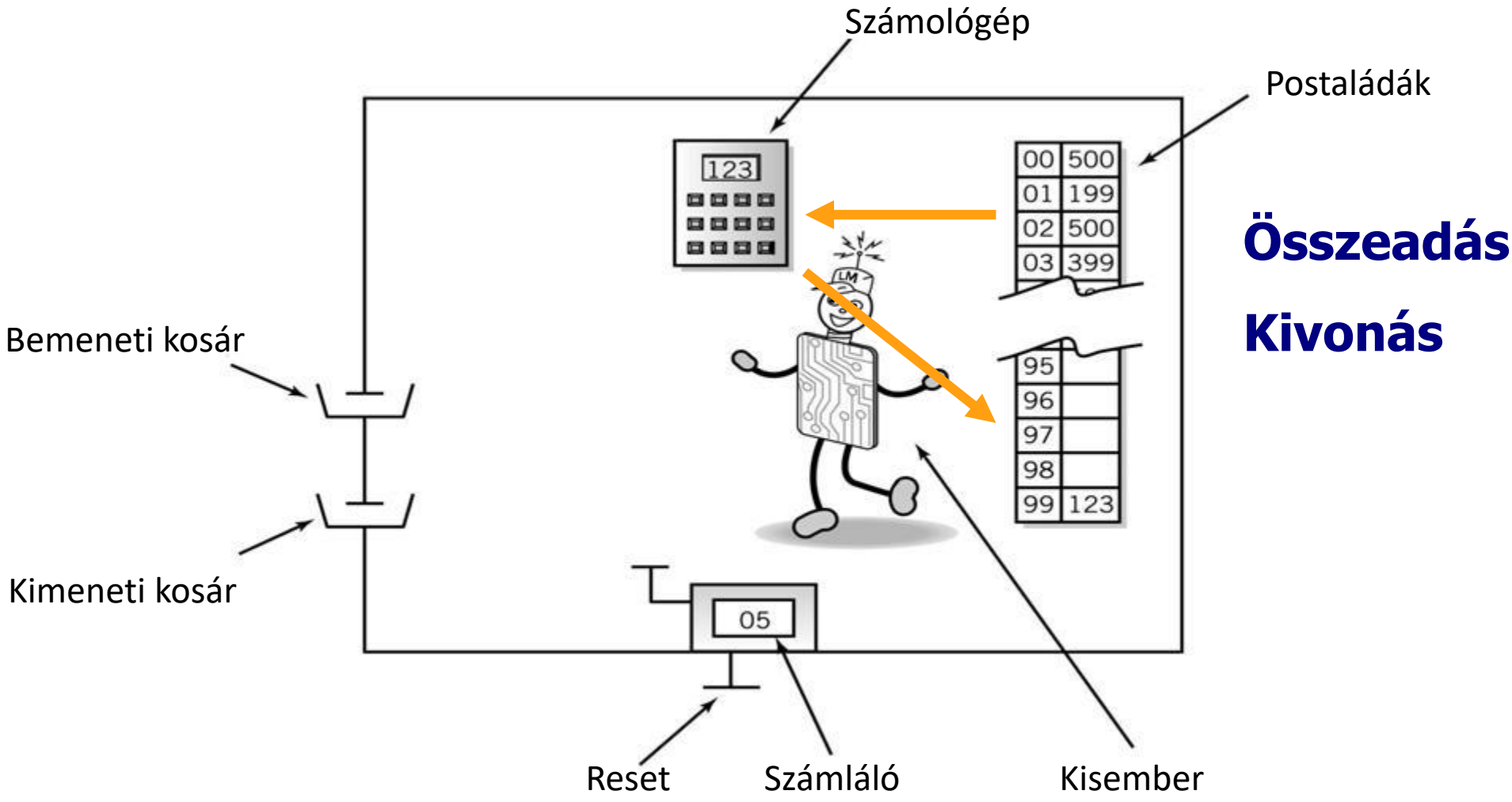
Aritmetikai utasítások

Olvassuk el az operandus által meghatározott levelesládában lévő értéket

Hajtsuk végre a műveletet a számológéppel

	Tartalom	
	Műveleti kód	Operandus (cím)
Összeadás	1	xx
Kivonás	2	xx

LMC Aritmetikai utasítások



Utasítás-végrehajtási ciklus

Az utasítások végrehajtása

- Kikeresés (Fetch)
  - Kisember kikeresi, hogy melyik utasítást kell végrehajtani
- Értelmezi az utasítást (Decode)
  - Az utasítás kódja alapján eldönti, hogy mit kell pontosan csinálni
- Végrehajtás (Execute)
  - Kisember elvégzi a munkát.

A számítógép által egy-egy utasítás végrehajtásakor elvégzett tevékenység-sorozatot utasítás-végrehajtási ciklusnak nevezzük

- Mivel ezek ciklikusan ismétlődnek.

Kikeresés és dekódolás

1. Kisember kiolvassa az utasításszámlálóból annak a postaládának a sorszámát (címét), ami az utasítást tartalmazza
2. Átsétál ahhoz a postaládához, ami megfelel a számláló értékének.
3. Kiveszi belőle a cédulát, elolvassa és dekódolja a számot, ami rajta van
   - A cetlit visszateszi, arra az esetre, ha esetleg később szükség lenne rá, hogy újra elolvassa

Végrehajtás

1. Kisember odamegy a postaládához, aminek a sorszámát az éppen kikeresett utasítás tartalmazza.
2. Kiveszi belőle a cédulát és elolvassa a rajta lévő számot
   - nem felejti el a cédulát visszatenni, mert később is szükség lehet rá
3. A kiolvasott utasítás kódjának megfelelően elvégzi a teendőt az adattal
4. Odasétál a számlálóhoz és továbblöki, így folytathatja a következő utasítás kikeresésével.

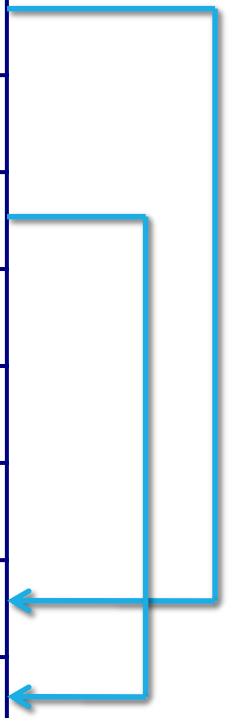
Példaprogram

Adjuk meg két szám különbségét

- Olvassuk be az első számot
- Tároljuk el
- Olvassuk be a második számot
- Tároljuk el
- Vonjuk ki belőle az elsőt
- Írjuk ki az eredményt
- Fejezzük be a munkát

LMC gépi kód

Postaláda	Kód
00	901
01	307
02	901
03	308
04	207
05	902
06	000
07	000
08	000



LMC gépi kód

Postaláda	Kód	Utasítás leírása
00	901	Első szám beolvasása
01	307	Szám tárolása
02	901	Második szám beolvasása
03	308	Szám tárolása
04	207	Az első kivonása a 2.-ból
05	902	Eredmény kiírása
06	000	Stop
07	000	Adatrekesz
08	000	Adatrekesz

Számítsuk ki két szám különbségének abszolút értékét!

1. Olvassuk be az első számot
2. Tároljuk el
3. Olvassuk be a másik számot
4. Tároljuk el
5. Vonjuk ki belőle az elsőt
6. Ha pozitív az eredmény, folytassuk a 9.–nél
7. (Különben) töltsük be az első számot
8. Vonjuk ki belőle a másodikat
9. Írjuk ki az eredményt
10. Fejezzük be a munkát

Program vezérlés

Elágazás utasítások

- Az utasítások egymásutánjából való kiugrás lehetősége
- Megváltoztatja a címet az utasítás számlálóban

Leállás

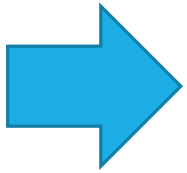
	Tartalom	
	Műv. kód	Operandus (cím)
Ugrás az xx számú postafiókra	6	xx
Ha a számológép tartalma 0, ugrás az xx számú postafiókra	7	xx
Ha a számológép tartalma >0, ugrás az xx számú postafiókra	8	xx

LMC utasításkészlete

Hozzáadás	1xx
Kivonás	2xx
Tárolás	3xx
Betöltés	5xx
Ugrás	6xx
Ugrás 0-nál	7xx
Ugrás pozitívnál	8xx
Input	901
Output	902
Stop	000

Számítsuk ki két szám különbségének abszolút értékét!

Postaláda	Kód
00	901
01	307
02	901
03	308
04	207
05	902
06	000
07	000
08	000



Postaláda	Kód
00	901
01	310
02	901
03	311
04	210
05	808
06	510
07	211
08	902
09	000
10	000
11	000

Számítsuk ki két szám különbségének abszolút értékét!

00	901	Első szám beolvasása
01	310	Szám tárolása
02	901	Második szám beolvasása
03	311	Szám tárolása
04	210	Első számot kivonjuk
05	808	Ha pozitív, ugorj a 08-ra (a kiírásra)
06	510	(különben) töltsd be az elsőt
07	211	Vond ki belőle a másodikat
08	902	Írd ki az eredményt
09	000	Stop
10	000	Az első adat helye
11	000	A második adat helye

Nevezzük el az utasításkódokat!

Hozzáadás	1xx	ADD xx
Kivonás	2xx	SUB xx
Tárolás	3xx	STO xx
Betöltés	5xx	LDA xx
Ugrás	6xx	BR xx
Ugrás 0-nál	7xx	BRZ xx
Ugrás pozitívnál	8xx	BRP xx
Input	901	INP
Output	902	OUT
Stop (Halt)	000	HLT
Adattároló		DAT

Számítsuk ki két szám különbségének abszolút értékét!

00	INP	901	Első szám beolvasása
01	STO 10	310	Szám tárolása
02	INP	901	Második szám beolvasása
03	STO 11	311	Szám tárolása
04	SUB 10	210	Első számot kivonjuk
05	BRP 08	808	Ha pozitív, ugorj a 08-ra (a kiírásra)
06	LDA 10	510	(különben) töltsd be az elsőt
07	SUB 11	211	Vond ki belőle a másodikat
08	OUT	902	Írd ki az eredményt
09	HLT	000	Stop
10	DAT 00	000	Az első adat helye
11	DAT 00	000	A második adat helye

Adjunk nevet a változóknak és az utasításoknak!

	00	INP	901	Első szám beolvasása
	01	STO ELS	310	Szám tárolása
	02	INP	901	Második szám beolvasása
	03	STO MAS	311	Szám tárolása
	04	SUB ELS	210	Első számot kivonjuk
	05	BRP KI	808	Ha pozitív, ugorj a 08-ra (a kiírásra)
	06	LDA ELS	510	(különben) töltsd be az elsőt
	07	SUB MAS	211	Vond ki belőle a másodikat
KI	08	OUT	902	Írd ki az eredményt
	09	HLT	000	Stop
ELS	10	DAT 00	000	Az első adat helye
MAS	11	DAT 00	000	A második adat helye

Példa

Mit csinál?

```
     INP
     STO ELS
     INP
     ADD ELS
     OUT
     HLT
ELS  DAT 00
```

Másik példa

És ez mit csinál?

```
      INP
LOOP  SUB  ONE
      OUT
      BRZ  QUIT
      BR   LOOP
QUIT  HLT
ONE   DAT  1
```

Assembly nyelv

Jellemzői:

- CPU specifikus
- 1 - 1 megfelelés az assembly nyelvű utasítás és a gépi nyelvű utasítás között
  - makró assembly később alakult ki
- Mnemonikok reprezentálják az utasításokat
- Változók deklarálhatók
- Utasítások címkézhetők

Assembler lefordítja

- inputja az assembly nyelvű program
- outputja a gépi nyelvű

Végeztünk?

Assembly

Láthattuk, hogy bár az assembly mindenre jó, mégsem célszerű.

Szembetűnően hiányzik:

- Átláthatóság
  - Absztrakció hiánya
- Hordozhatóság
  - CPU, környezet specifikus
- Újrafelhasználhatóság
  - Magasabb szintű, nemcsak hívható alprogramok
- …

Magasabb szintű nyelvek

Magasabb szintű nyelvek

Láttuk, hogy a gépi kód egyértelmű, utasításokat adunk a CPU-nak

- Az Assembly ennek az olvasmányosabb változata
  - További funkciókkal
- A magasabb szintű nyelvek nyelvi elemei nem feleltethetők meg egy-egy módon ennek, fordítás vagy értelmezés szükséges
- **Szabályok kellenek, amik leírják az adott nyelvet:**
  - Mik a nyelvi elemek
  - Hogyan kell használni
  - Mi az értelmezése ezeknek

Szintaktika és szemantika

Avram Noam Chomsky (1928-)

Amerikai nyelvész,

A generatív nyelvtan elméletének megalkotója

Chomsky – Generatív grammatikák

Chomsky nyelvészként a természetes nyelveket vizsgálta

- Leírást adni a természetes nyelvek szabályaira
- Számos elmélet, cikk szerzője

Alapgondolat tömören

- Gyerek – véges számú mondatot hall
  - Képes tetszőleges nyelvtanilag helyes mondat megalkotására
  - Kell legyen egy szabályrendszer minden nyelvben!

A mesterséges nyelvek esetén nagyobb a siker …

Generatív grammatika – Példa

Egyszerű példa, illusztrációra – kis nyelvtani emlékeztető.

- „ A haszontalan kisfiú ledobta a tányért.”

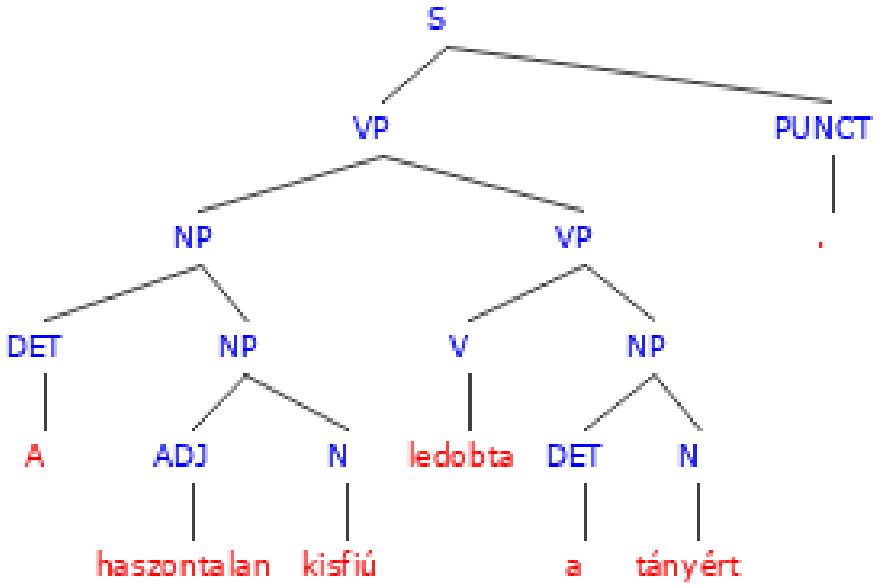
Ez egy mondat, ami felbontható

- Van alanyi és állítmányi részre
  - Az alanyi rész névelő, jelző és alany sorozata lehet
  - Az állítmányi rész az állítmány és a tárgy egymásutánja, stb.

A struktúra leírásakor „grammatikai jeleket” is használhatunk a mondatrészek, illetve szófajok jelzésére

- ezeket < > zárójelek közé tesszük

Elemzési fa



ADJ	melléknév (adjective)
DET	névelő (determiner)
N	főnév (noun)
NP	névszói csoport (noun phrase)
PRON	névmás (pronoun)
PUNCT	írásjel (punctuation)
S	mondat (sentence)
V	ige (verb)
VP	igei csoport (verbal phrase)

https://www.webforditas.hu/elemzo

Példa egyszerűsített szabályokra

< S >	→ < VP > < PUNCT >
< VP >	→ < VP > < NP >
< VP >	→ < V > < NP >
< NP >	→ < DET > < NP>
< NP >	→ < DET > < N >
< NP >	→ < ADJ > < N >
< DET >	→ a
< ADJ >	→ haszontalan
< N >	→ kisfiú
< N >	→ tányér(t)
< V >	→ ledobta
< PUNCT >	→ .

ADJ	melléknév (adjective)
DET	névelő (determiner)
N	főnév (noun)
NP	névszói csoport (noun phrase)
PRON	névmás (pronoun)
PUNCT	írásjel (punctuation)
S	mondat (sentence)
V	ige (verb)
VP	igei csoport (verbal phrase)

Legbaloldalibb levezetés

Ha adott a fenti módon egy szabályhalmaz, akkor azt használhatjuk mondatok képzésére a következő módon:

- A < S > elnevezésű bal oldalon lévő grammatikai elemből kiindulva behelyettesíthetjük a szabály jobb oldalán álló jelsorozatot
- Majd ennek első elemét vesszük sorra, s írjuk át olyan szabály felhasználásával, ahol megfelelő részt bal oldalon találjuk.
- Ezt az eljárást folytatjuk, amíg még találunk mondatrészt a levezetett jelsorozatban, azaz, amíg át nem térünk a grammatikai egységekről a nyelv szavaira.
- A levezetés során az egyes grammatikai elemeket tetszés szerinti sorrendben helyettesíthetjük.

A következőkben a fenti grammatikai szabályok segítségével oly módon jutunk el az előző példamondatunkhoz, hogy a levezetés során mindig a lehetséges legbaloldalibb helyettesítést választjuk.

Legbaloldalibb levezetés

< S > → < VP > < PUNCT >

< VP > < PUNCT > → < NP > < VP > < PUNCT >

< NP > < VP > < PUNCT > → < DET > < NP > < VP > < PUNCT >

< DET > < NP > < VP > < PUNCT > → A < NP > < VP > < PUNCT >

A < NP > < VP > < PUNCT > → A < ADJ > < N > < VP > < PUNCT >

A < ADJ > < N > < VP > < PUNCT > → A haszontalan < N > < VP > < PUNCT >

A haszontalan < N > < VP > < PUNCT > → A haszontalan kisfiú < VP > < PUNCT >

A haszontalan kisfiú < VP > < PUNCT > → A haszontalan kisfiú < V > < NP > < PUNCT >

A haszontalan kisfiú < V > < NP > < PUNCT > → A haszontalan kisfiú ledobta < NP > < PUNCT >

A haszontalan kisfiú ledobta < NP > < PUNCT > → A haszontalan kisfiú ledobta < DET > < N > < PUNCT >

A haszontalan kisfiú ledobta < DET > < N > < PUNCT > → A haszontalan kisfiú ledobta a < N > < PUNCT >

A haszontalan kisfiú ledobta a < N > < PUNCT > → A haszontalan kisfiú ledobta a tányért < PUNCT >

A haszontalan kisfiú ledobta a tányért < PUNCT > → A haszontalan kisfiú ledobta a tányért .

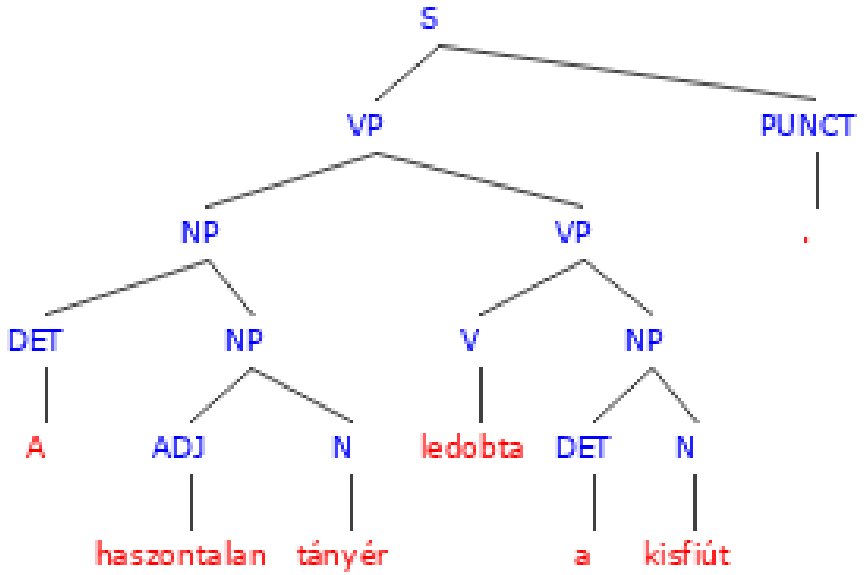
Legbaloldalibb levezetés

A szabályok szerint lehetséges még:

- A haszontalan tányér ledobta a kisfiút.
- A haszontalan kisfiú ledobta a kisfiút.
- A haszontalan tányér ledobta a tányért.

Nem triviális.

- A továbbiakban szorítkozzunk a mesterséges nyelvekre.



Mesterséges nyelvek

Magas szintű programozási nyelv esetén szükség van a forrásprogram „fordítására”, vagy „értelmezésére”

- Assemblyre
- Gépi kódra

Ehhez szükséges egy

- Fordítóprogram (compiler), vagy
- Értelmezőprogram (interpreter)

A programok végrehajtása

Az interpreter egy utasítás lefordítása után azonnal végrehajtja azt

A fordítóprogram átalakítja a programot egy vele ekvivalens formára

- ez lehet a számítógép által (majdnem) közvetlenül végrehajtható forma, vagy
- lehet egy másik programozási nyelv

Fordítás

A lefordítandó program a forrásprogram.

- Source program / sorce code

A fordítás eredményeként kapott program a tárgyprogram.

- Object program / object code

Az az idő, amikor az utasítások fordítása folyik, a fordítási idő.

- Compilation time

Az az idő, amikor az utasítások végrehajtása folyik, a végrehajtási idő.

- Execution time / runtime

Szabályok

Szabályok kellenek, amik leírják

- Mik a nyelvben használható jelek (karakterek)
- Milyen nyelvi elemek vannak (formailag helyes karaktersorozatok)
- Jelentés szempontjából mik a helyes karaktersorozatok

A forráskódot író és az azt fordítóprogram ugyanazt kell, hogy értse a leírt kódon

- A gép a szabályok szerint értelmezett parancsainkat teljesíti, nem a kívánságainkat.

Általános fogalmak

A nyelv szintaxisa azoknak a szabályoknak az összessége, amelyek az adott nyelven írható összes lehetséges, formailag helyes programot (jelsorozatot) definiálják.

- Reguláris kifejezések, BNF forma

Az adott nyelv programjainak jelentését leíró szabályok összessége a nyelv szemantikája.

Szintaxis és szemantika

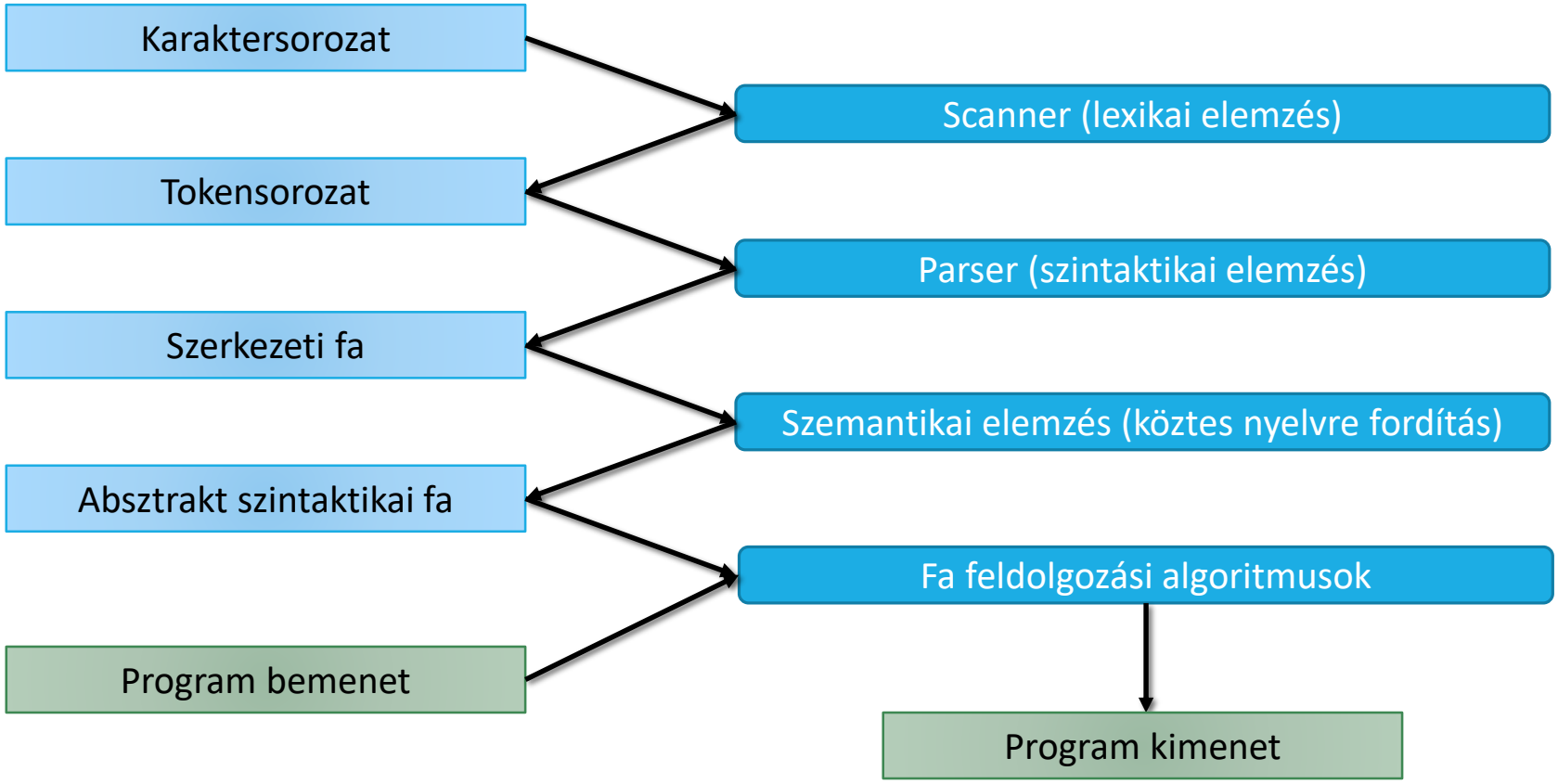
Példa:

- DD / DD / DDDD
- 02 / 11 / 2019

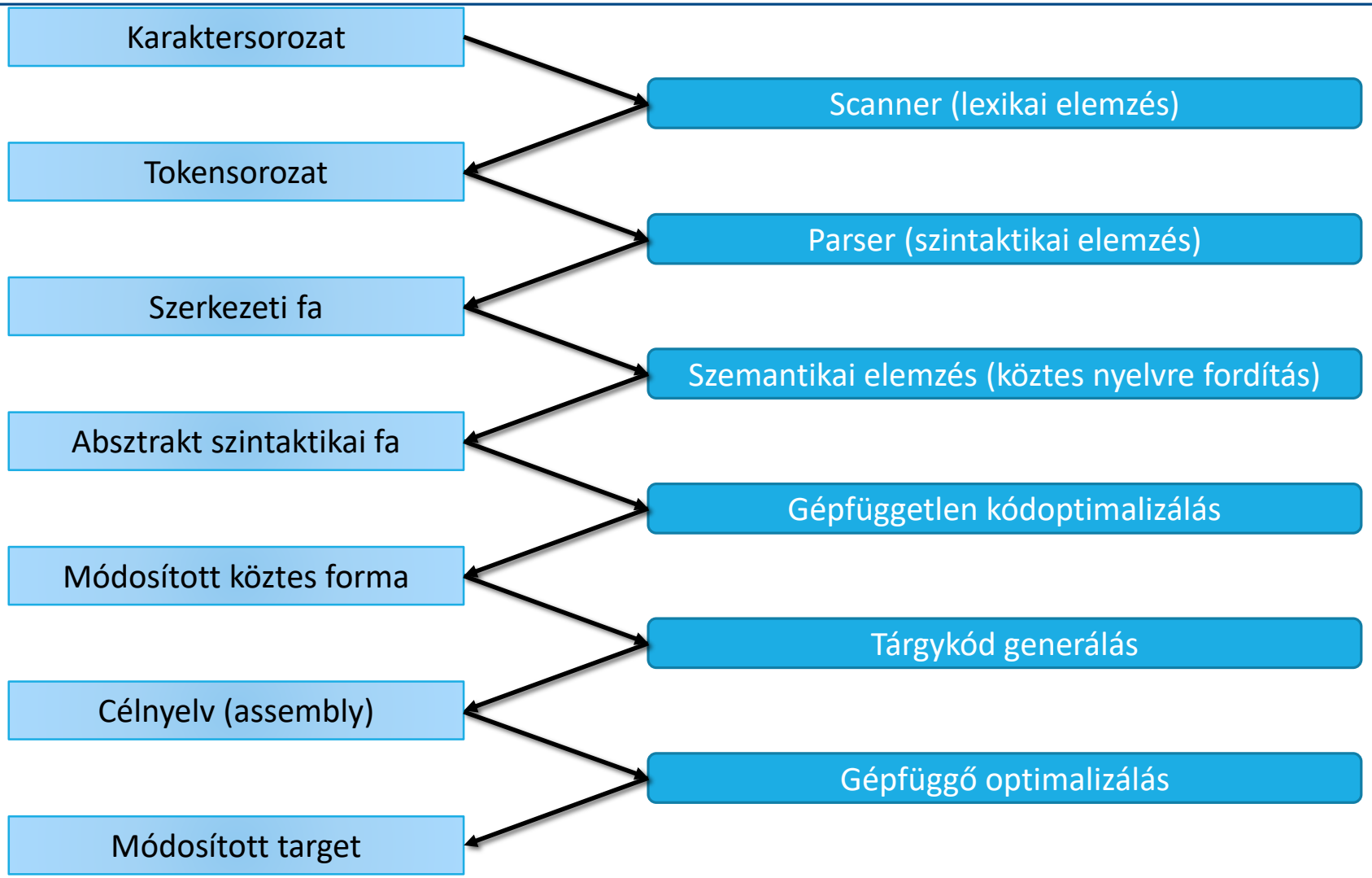
2019. február 11. vagy 2019. november 2.?

A szintaxis befolyásolja a programok megbízhatóságát

Interpretálás



Fordítás


Karaktersorozat
Scanner (lexikai elemzés)
Tokensorozat
Parser (szintaktikai elemzés)
Szerkezeti fa
Szemantikai elemzés (köztes nyelvre fordítás)
Absztrakt szintaktikai fa
Gépfüggetlen kódoptimalizálás
Módosított köztes forma
Tárgykód generálás
Célnyelv (assembly)
Gépfüggő optimalizálás
Módosított target

Nyelv leírása szabályokkal

Nyelv leírása a fordítónak

Formális nyelv

- Grammatika – generálja a nyelvet
  - A szabályok alapján a nyelv szerinti „mondatok” hozhatók létre
- Automaták – felismerik
  - A generált „mondatok” elemzése, ellenőrzése, feldolgozása

Formális nyelvek – definíciók

Σ: az **ábécé** tetszőleges szimbólumok véges, nem üres halmaza

A **szó** az ábécé elemeiből képezett $a_1a_2...a_k$ alakú sorozat, ahol $k \geq 0$, $a_1, a_2, ..., a_k \in \Sigma$

A **Σ*** a Σ ábécé elemeiből képezhető összes szavak halmaza:

- $\Sigma^* = \{a_1a_2...a_k \mid k \geq 0, a_1, a_2, ..., a_k \in \Sigma\}$
- Ha $k=0$, akkor a szó üres szó, jele ε vagy λ

Σ^+ a nem üres szavak halmaza:

- $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$

Példa

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

$$\Sigma^+ = \Sigma^* - \lambda$$

Grammatikák – definíciók

Generatív nyelvtan (G)

$G=(N, \Sigma, S, P)$, ahol

N a nemterminálisok (grammatikai jelek) ábécéje

Σ a terminálisok abc-je, $N \cap \Sigma = \varnothing$

$S \in N$ a kezdőszimbólum

P: $\alpha \rightarrow \beta$ alakú átírási szabályok véges halmaza

α a szabály baloldala, β a szabály jobboldala

$\alpha, \beta \in (N \cup \Sigma)^*$

α-ban van legalább egy nemterminális szimbólum

Magyarázat

Nemterminális

- Nem „célnyelvi” szavak, hanem a levezetési szabályok még behelyettesíthető szimbólumai
  - < ADJ > < NP > < PUNCT>

Terminális

- „Célnyelvi” szavak, nem helyettesíthetők be tovább
  - Kisfiú tárnyért

Kezdőszimbólum

- A startszimbólum, ezzel kezdjük meg a levezetést

Grammatikák – definíciók

Közvetlen levezetés (közvetlen levezetési reláció, $\Rightarrow G$)

Legyen $\gamma, \delta \in (N \cup \Sigma)^*$

$\gamma \Rightarrow G\ \delta$, ha van olyan $\alpha \rightarrow \beta \in P$ szabály és

vannak olyan $\alpha', \beta' \in (N \cup \Sigma)^*$ szavak, hogy

$\gamma = \alpha'\alpha\beta'$ és $\delta = \alpha'\beta\beta'$

- $\Rightarrow G^n$ - n lépéses levezetés
- $\Rightarrow G^+$ - legalább egy lépéses levezetés
- $\Rightarrow G^*$ - levezethetőség (esetleg nulla lépésben is)

Grammatikák – definíciók

G grammatika által generált nyelv L(G)

- Legyen G=(N, Σ, S, P) generatív nyelvtan.
- L(G)={u$\in\Sigma^*$ | S $\Rightarrow_G^*$ u}
- L(G) = a G generatív grammatika által generált nyelv
  - összes lehetséges szó

Ekvivalens nyelvtanok

- Legyenek G=(N, Σ, S, P) és G'=(N', Σ', S', P') generatív nyelvtanok.
  - G ekvivalens G'-vel, ha L(G)=L'(G').
  - Ekvivalens két nyelvtan, ha ugyanazt a nyelvet generálja

Chomsky-féle nyelvosztályok

0. típusú nyelvtan (kifejezés struktúrájú)

- ha semmilyen korlátozás nincs
- Turing gép tudja feldolgozni

1. típusú nyelvtan (környezetfüggő)

- ha P-ben minden szabály $\alpha A\beta \rightarrow \alpha\delta\beta$ alakú
  - Itt $\delta \neq \lambda$, kivéve esetleg az $S \rightarrow \lambda$ szabályt, mely esetben $\alpha=\beta=\delta=\lambda$, de ekkor S nem szerepelhet egyetlen szabálynak sem a jobboldalán.
  - $(\alpha,\beta,\gamma,\delta \in (N \cup \Sigma)^*)$
- Turing gép tudja feldolgozni
- *Környezettfüggőség: a behelyettesítésnél figyelembe kell venni a behelyettesítendő szimbólum környezetében (szomszédságában) levő további szimbólumokat*

Chomsky nyelvosztályok (folyt.)

2. típusú nyelvtan (környezetfüggetlen)

- ha P-ben minden szabály $A \rightarrow \alpha$ alakú
- ($A \in N$, $\alpha \in (N \cup \Sigma)^*$)
- Nemdeterminisztikus veremautomata tudja feldolgozni
- *Környezettfüggőség: a behelyettesítésnél nincs meghatározva megkötés a behelyettesítendő szimbólum környezetében levő további szimbólumokra*

3. típusú nyelvtan (reguláris, jobblineáris)

- ha P-ben minden szabály $A \rightarrow xB$, vagy $A \rightarrow x$ alakú.
- ($A, B \in N$, $x \in \Sigma^*$)
- Véges determinisztikus automata tudja feldolgozni

Chomsky nyelvosztályok

Nyelv típusa

Egy $L \subseteq \Sigma^*$ nyelv i típusú (i=0,1,2,3),
ha van olyan i típusú nyelvtan, ami éppen L-et generálja.

$$\mathcal{L}_0 \supset \mathcal{L}_1 \supset \mathcal{L}_2 \supset \mathcal{L}_3$$

Példák

Reguláris grammatika: $G = (\{S\}, \{a,b\}, S, P)$

P: $S \rightarrow abS$

$S \rightarrow a$

$S \Rightarrow a$

$S \Rightarrow abS \Rightarrow aba$

$S \Rightarrow abS \Rightarrow ababS \Rightarrow ababa = (ab)^2a$

$S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS \Rightarrow abababa = (ab)^3a$

$S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS \Rightarrow abababababS \Rightarrow abababababa = (ab)^4a$

$L(G) = \{(ab)^n a \mid \text{ahol } n \geq 0\}$

$L(G) = (ab)^*a$

Példák

Környezetfüggetlen grammatika: $G = (\{S\}, \{a,b\}, S, P)$

P: $S \rightarrow aSb$

$S \rightarrow \lambda$

$S \Rightarrow \lambda$

$S \Rightarrow aSb \Rightarrow ab$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb = a^2b^2$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb = a^3b^3$

....

$L(G) = \{a^n b^n \mid \text{ahol } n \geq 0\}$

Példák és problémák

Reguláris kifejezések

Definíció: Egy S abc feletti reguláris kifejezések:

1. S minden eleme.
2. λ (vagy e)
3. $\emptyset$
4. Ha a és b reguláris kifejezések, akkor (ab) is az.
5. Ha a és b reguláris kifejezések, akkor $(a \cup b)$ is az.
6. Ha a reguláris kifejezés, akkor a^* is az.

Minden reguláris kifejezés reprezentál egy nyelvet.
Példa reguláris kifejezésre és nyelvre:

$(a^* bc)\ ^*ab$ $(\{a\}^*\{b\}\{c\}\ )^*\{a\}\{b\}$

Reguláris kifejezések

Minták megadása – „hagyományos” jelölések a metakarakterekre:

- Karakter megfeleltetés
  - . (pont) Bármilyen karakter
    - Pl. .a. ⇔ ‘pap’, ‘rak’, ‘lap’, ‘kap’, stb.
  - [karakterek] A szögletes zárójelek között felsorolt karakterek valamelyikével megegyező karakter:
    - Pl. [abc] ⇔ ‘a’ vagy ‘b’ vagy ‘c’
    - A - (mínusz) jellel tartományt is megadhatunk.
    - Például [0-9] megfelel bármely számjegynek vagy [a-zA-Z] bármely kis vagy nagybetűnek.
  - [^karakterek] az előző operátor tagadása
    - Pl. [^a-d] : bármely karakter, ami nem az ‘a’, ‘b’, ‘c’ vagy ‘d’

Reguláris kifejezések

- \d rövidítés: számjegyek 0-9
- \w rövidítés: betűk, számjegyek, ‘_’
- \s rövidítés: whitespace (space,tab,CR,LF)
- \D, \W, \S rövidítések: a fentiek negációi

Többszörözés - ismétlő operátorok a minták után:

- ? : 0 vagy 1 egyezés - pl. \d? ⇔ ∅, vagy egy számjegy
- * : 0 vagy több egyezés - pl. ab*a ⇔ aa, aba, abba, …
- + : 1 vagy több egyezés - pl. [0-9]+ ⇔ 1, 45, 460, ...
- {n} : n egyezés (n >= 1) - pl. .{3} ⇔ aaa, 888, ...
- {n,m} : n és m közötti egyezés (n>=1, n<=m) –
  pl. a{2,4} ⇔ ‘aaaa’, ‘aaa’ vagy ‘aa’
- {n,} : legalább n egyezés (n>=1) -pl. xa{2,}. ⇔ xaaf, xaaa, ...
- {,m}: legfeljebb m egyezés

Reguláris kifejezések

Diszjunkció

- „|” (pipe) szimbólum két minta között: „vagy”

  pl. a|A ⇔ ‘a’, ‘A’

- lehet sorozatban több is:

  pl. a|b|c ugyanaz, mint [a-c]

- a „|” -jel precedenciája a legalacsonyabb az operátorok között. Ezért:

  te|i ⇔ ‘te’, ‘i’

  t(e|i) ⇔ ‘te’, ‘ti’

Reguláris kifejezések

Horgonyok

- ^ : az elejére illeszkedik
  pl. ^M : <‘M’ a kezdete>
- $: a végére illeszkedik:
  pl. \.$: <pont a végén>
- \b : szóhatárra illeszkedik (\w és \W karakterek közötti pozícióra, vagy \w és sztring kezdete/vége közé)
  pl. c\b : ‘abc’-re illeszkedik ‘abc def’-ben
- \B: \b negáltja (karakter „szó” közepén)

Speciális karakterek csak ‘\’ segítségével használhatók

- Pl. \. vagy \-

Reguláris kifejezések

Csoportok

- A zárójel operátorokkal tetszőleges számú csoport kijelölhető a mintában:

  pl. a((bla)ko(k))(a)t
- A csoportokra alkalmazhatók az operátorok:

  pl. a(lm|kn)a

Használat

Bármilyen keresésnél, formai ellenőrzésnél:

- Google-ben, ha nem tudjuk biztosan, vagy több írásmód is létezik:
  - "Handel", "Händel” vagy "Haendel” ?
  - H(ä|ae?)ndel
- IP-cím formátum:
  - <3-jegyű_szám>.<3-jegyű_szám>.<3-jegyű_szám>.<3-jegyű_szám>
  - RegKif: \b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b

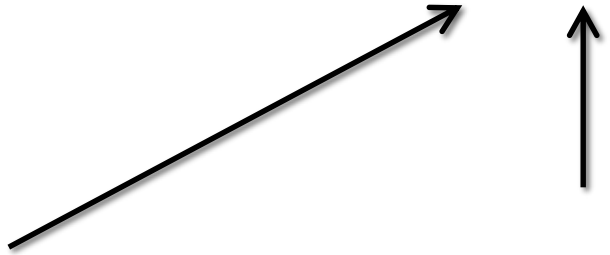
Fordítóprogramokban a lexikális elemző (lexer)

- Eldönthető, hogy érvényes szimbólumokból állnak-e a szavak
- változónév: [a-zA-Z][a-zA-Z0-9]*
- egész szám: (\+|\-)?[0-9]+
- valós szám: [0-9]+\.[0-9]*

Context-Free Grammatika

$G=(N, \Sigma, S, P)$

ha P-ben minden szabály $A \rightarrow \alpha$ alakú.



Grammatikai jel

Grammatikai jelek és terminálisok sorozata

$(A \in N,\ \alpha \in (N \cup \Sigma)^*)$

A grammatika által generált nyelv

G grammatika

S kezdőszimbólum

$$L(G) = \{w : \quad S \overset{*}{\Rightarrow} w, \quad w \in \Sigma^*\}$$

Terminálisok sorozata vagy a λ

Context-Free nyelv

Egy L nyelv környezetfüggetlen (CF) ha van egy G context-free grammatika, ahol

$$L = L(G)$$

Levezetési fa

Definíció: Egy G CF grammatika szerinti levezetés egy fával ábrázolható

- A gyökere a kezdőszimbólum
- A levelek balról jobbra a levezetett szó betűi
- További csomópontjai nemterminálisok
- Az elágazások egy-egy alkalmazott szabálynak felelnek meg

Ez a levezetési fa.

Példa

Context-free grammatika: G

$$S \to aSa \mid bSb \mid \lambda$$

Példa levezetések:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$$

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaba$$

$$L(G) = \{ww^R : \quad w \in \{a,b\}^*\}$$

Páros hosszú palindromák

Másik példa

Context-free grammatika : G

$$S \to aSb \mid SS \mid \lambda$$

Példa levezetések:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow ab$$

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSb \Rightarrow abab$$

$$L(G) = \{w \ : n_a(w) = n_b(w),$$

$$\text{és } \ n_a(v) \geq n_b(v)$$

$$\text{a } \ w \ \textit{bármely} \ v \ \textit{prefixében}\}$$

Leírja a jó zárójelezést: () ((( ))) (( ))

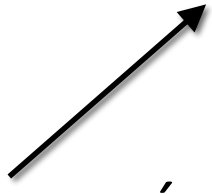
$a = ($, b $=)$

A matematikai kifejezések grammatikája

$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

Példa sorozatok:

$$(a + a) * a + (a + a * (a + a))$$



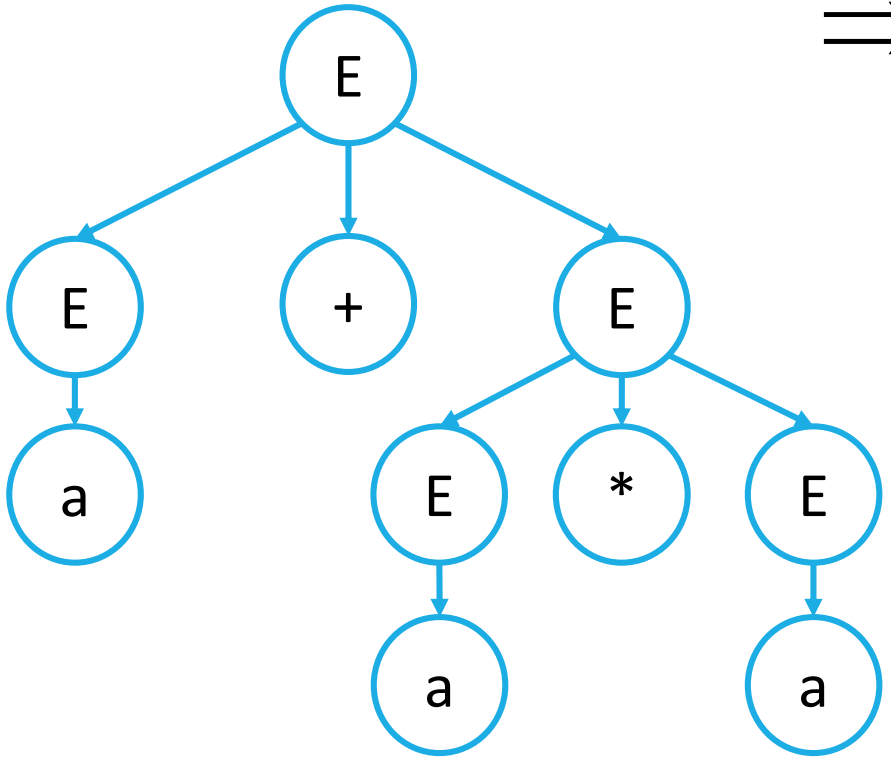
Tetszőleges szám

$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E$$

$$\Rightarrow a + a * E \Rightarrow a + a * a$$

$$a + a * a$$



egy legbaloldalibb levezetése

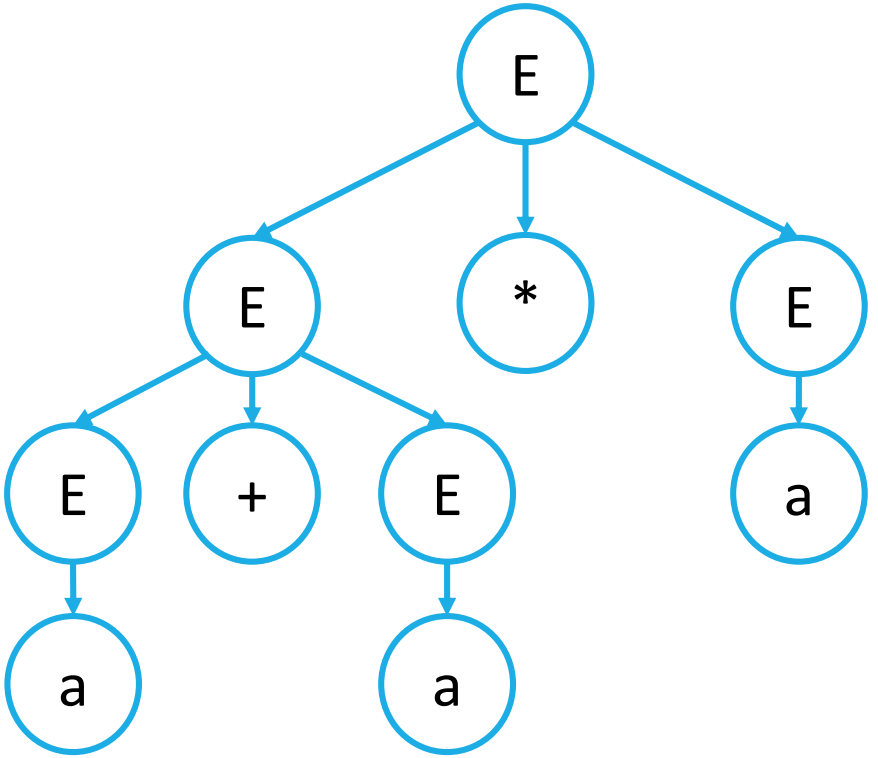
$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E$$

$$\Rightarrow a + a * E \Rightarrow a + a * a$$

$$a + a * a$$

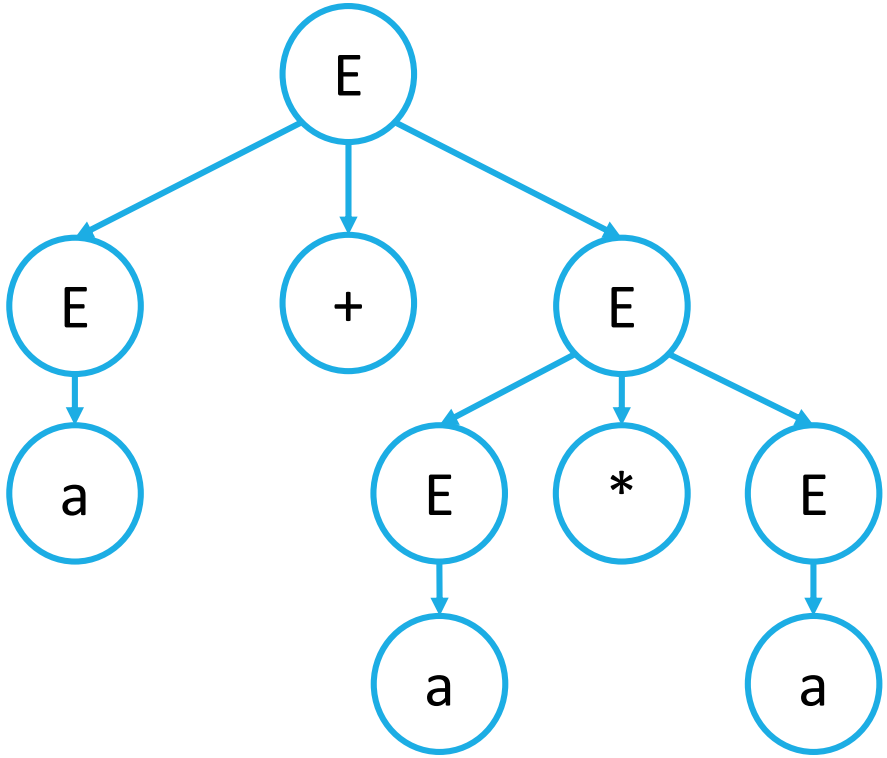
Egy másik legbaloldalibb levezetése

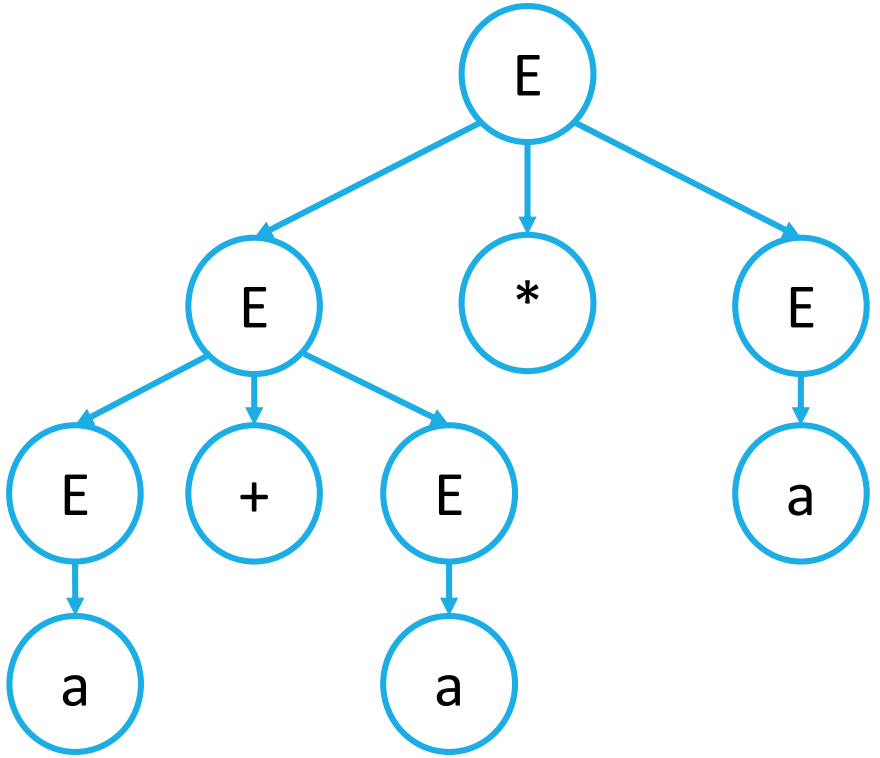


$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

$$a + a * a$$

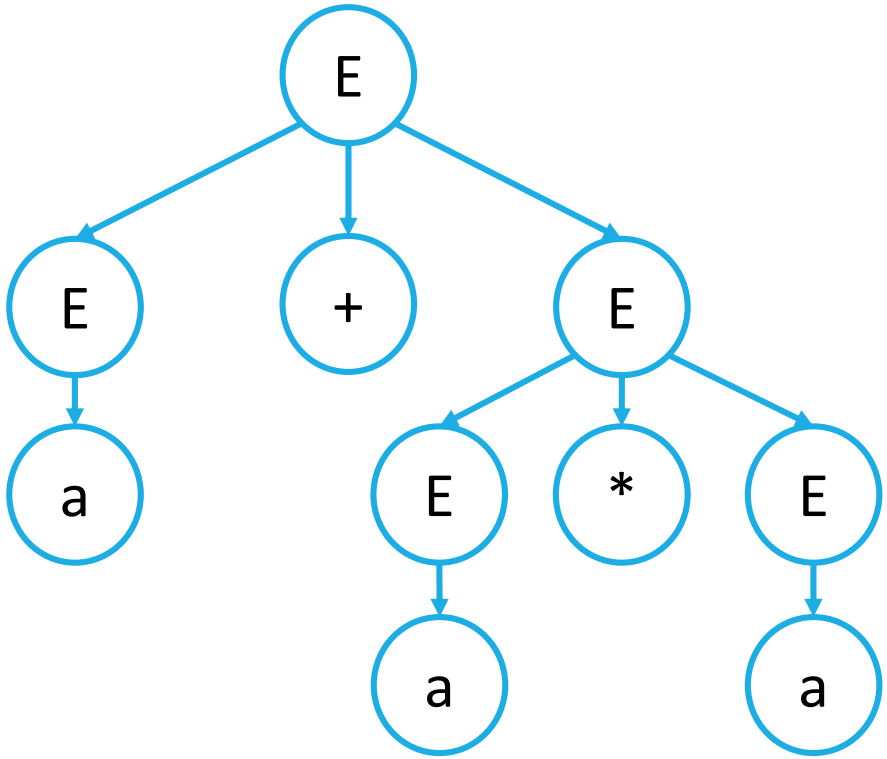
Két levezetési fa!

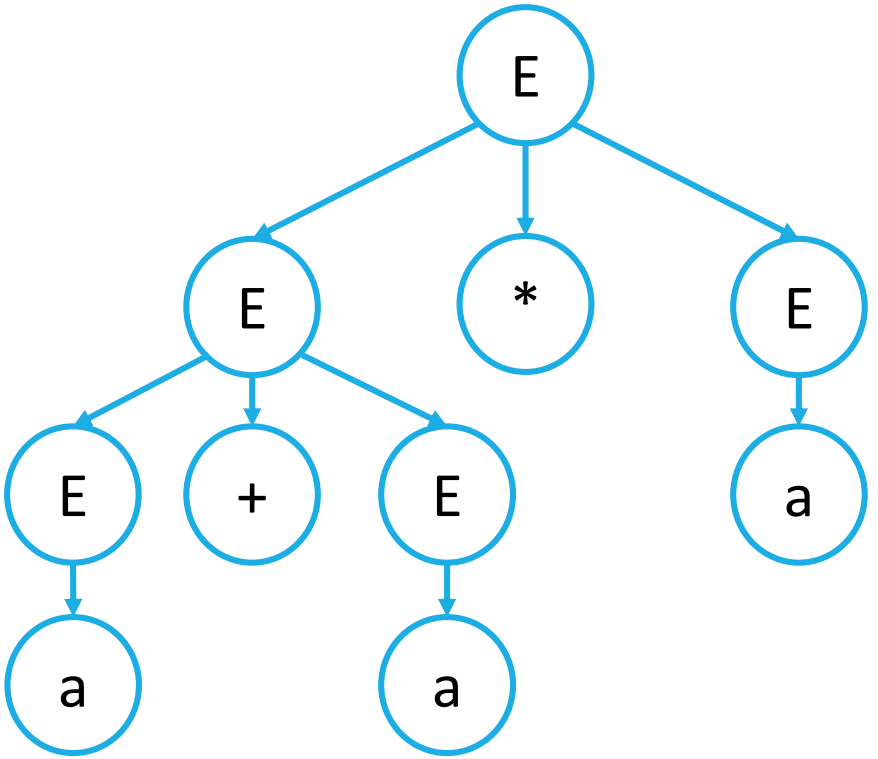




Legyen a = 2

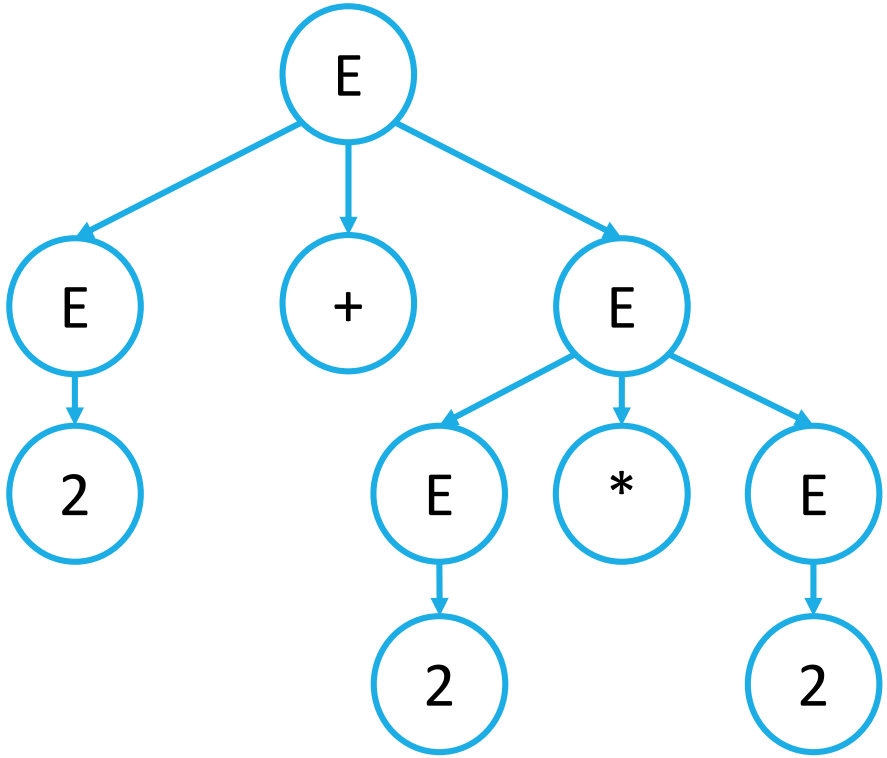
$$a + a * a = 2 + 2 * 2$$





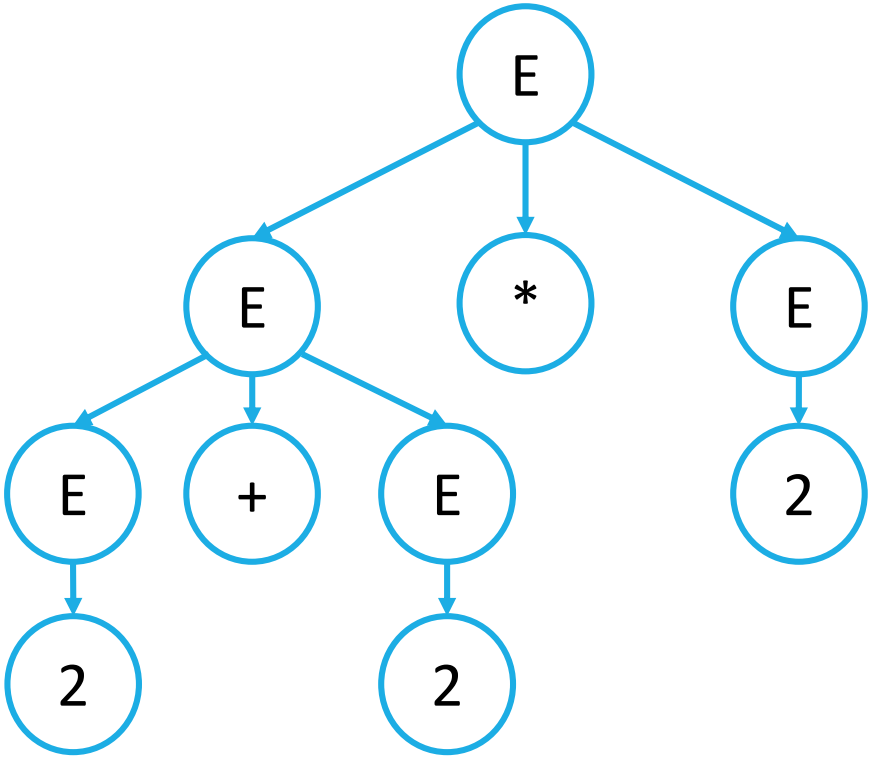
Legyen a = 2

$2 + 2 * 2 = 6$



Számítsuk ki a kifejezés eredményét a fát használva

$2 + 2 * 2 = 8$



Többértelmű grammatika

Egy G környezetfüggetlen grammatika **többértelmű**

ha van olyan $w \in L(G)$ amelyiknek:

- két különböző levezetési fája van, vagy
- két különböző legbaloldalibb levezetése van

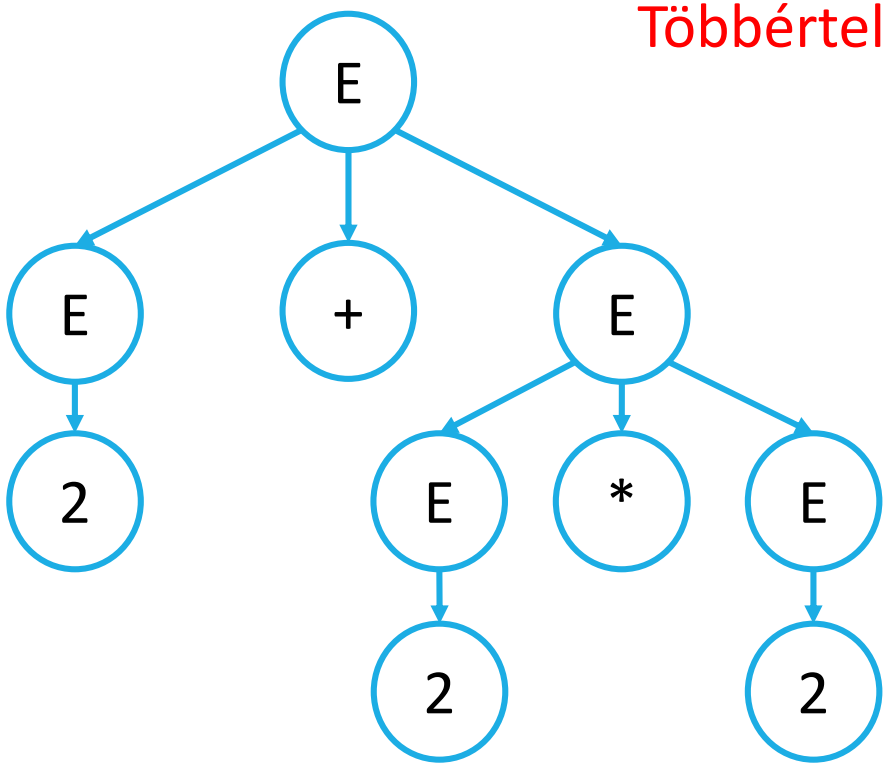
Két különböző levezetési fa problémákat okozhat

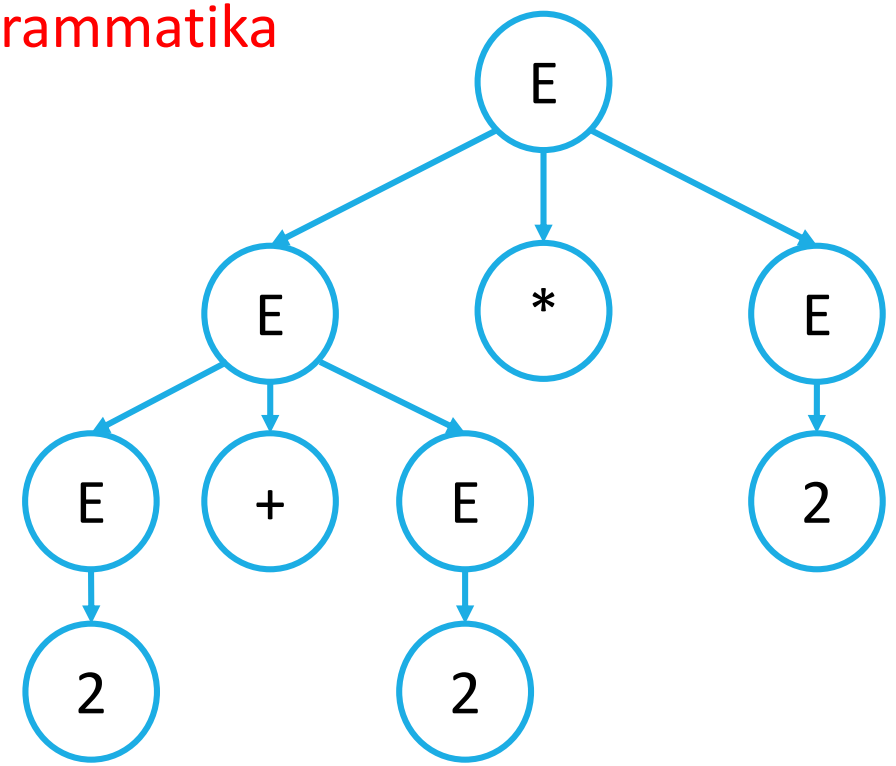
- kifejezések kiértékelésénél
- Általában - programozási nyelvek fordítóprogramjainál!

$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

$$a + a * a$$

Többértelmű a grammatika

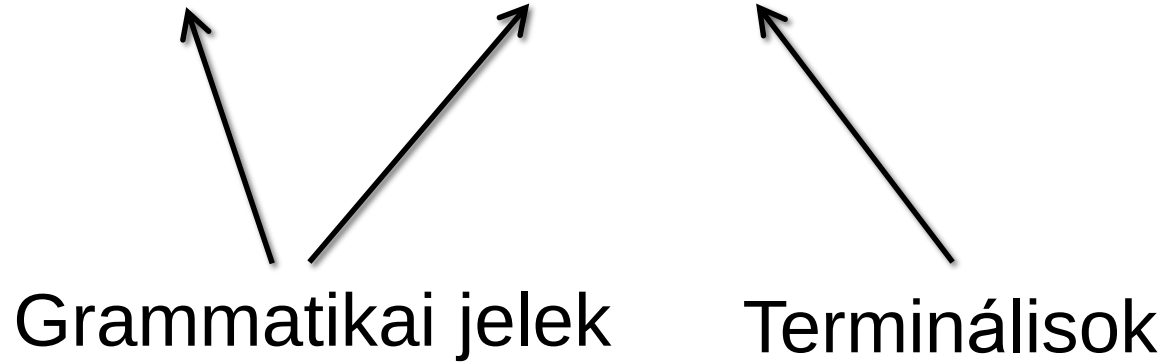




Egy másik többértelmű grammatika

IF_STMT → **if** EXPR **then** STMT

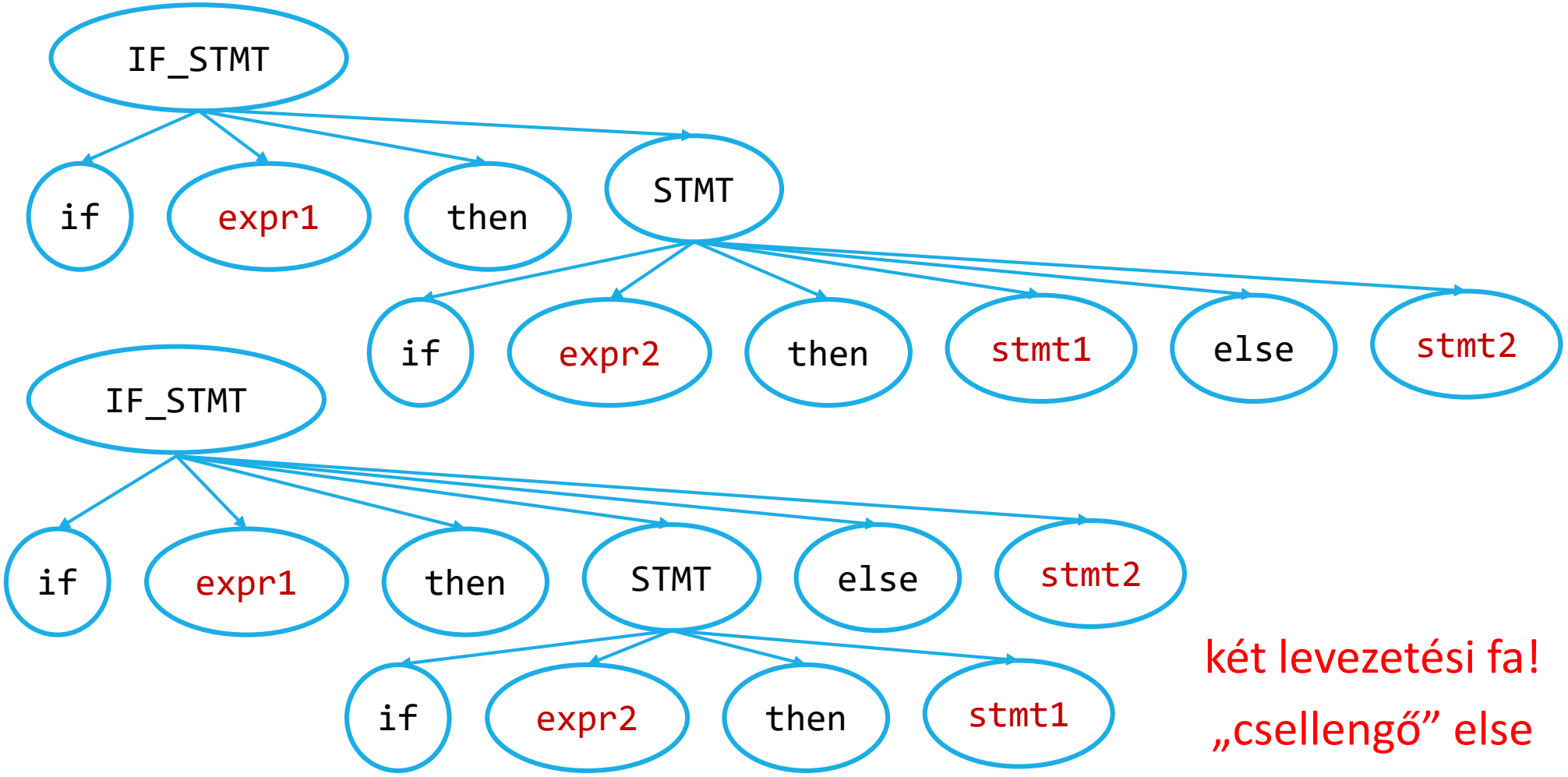
| **if** EXPR **then** STMT **else** STMT



A programozási nyelvek grammatikájának jellemző része

Egy másik többértelmű grammatika

```
if expr1 then if expr2 then stmt1 else stmt2
```



két levezetési fa!

„csellengő” else

Többértelműség

Általában a többértelműség rossz, és szeretnénk megszüntetni…

- Néha sikerül találni egyértelmű (nem többértelmű) grammatikát a nyelvhez
- De általánosságban ezt nehéz elérni...

Egy sikeres példa

Többértelmű grammatika:

$$E \to E + E$$
$$E \to E * E$$
$$E \to (E)$$
$$E \to a$$

Ekvivalens

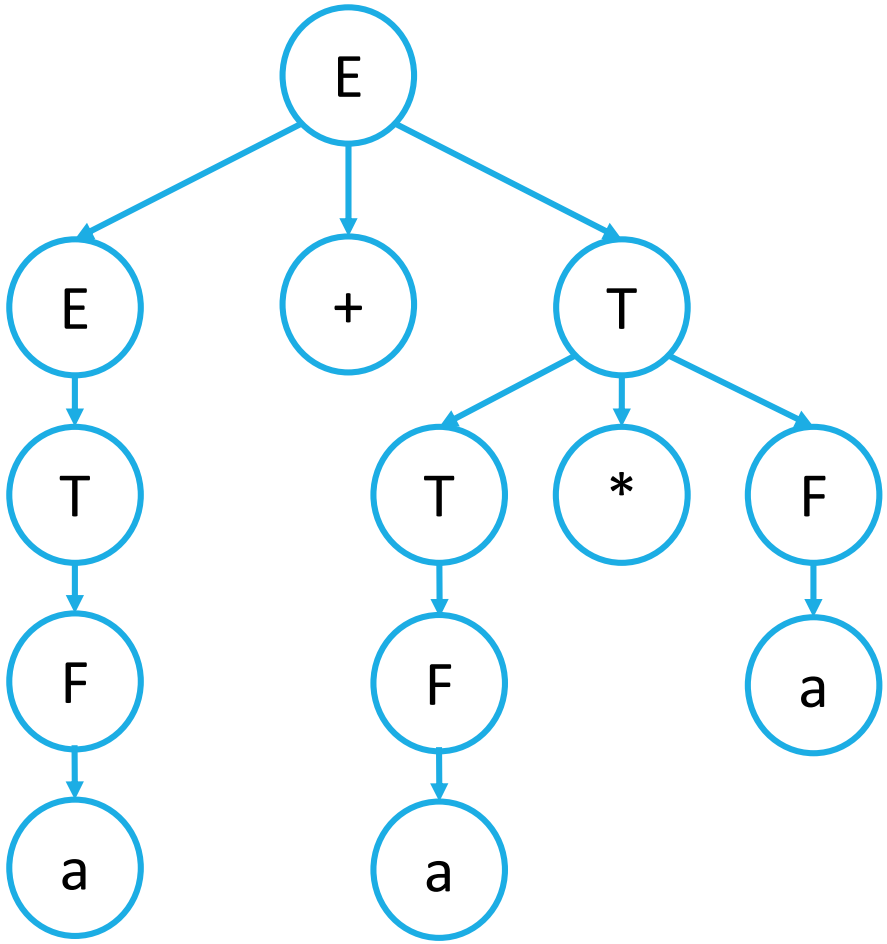
Nem-többértelmű grammatika

Ugyanazt a nyelvet generálja!

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid a$$

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+T*F$$

$$\Rightarrow a+F*F \Rightarrow a+a*F \Rightarrow a+a*a$$



$$E \rightarrow E+T \mid T$$

$$T \rightarrow T*F \mid F$$

$$F \rightarrow (E) \mid a$$

$$a+a*a$$

Egyértelmű a levezetési fa

Programozási nyelv megadása

Programozási nyelv megadása

Formális szintaxis leírás

- ezt használja a fordítóprogram a nyelvi helyesség ellenőrzésére

Formális szemantika leírás

- a jelentés ellentmondásmentességének ellenőrzésére

John Backus (1924-2007)

FORTRAN tervezője

"Munkám túlnyomó része egyszerűen a lustaságomból eredt. Amikor a rakéták röppályáját kiszámító programokon dolgoztam, hozzáfogtam egy programozási rendszer kidolgozásához, amelynek elsődleges feladata a munka megkönnyítése, leegyszerűsítése volt”

Backus-Naur Form (BNF)

Nyelv a programozási nyelvek definiálására

- meta-nyelv, eredetileg 59-60-ból, az Algol és Fortran definiálására
- változatok ma is programozási nyelvek megadására

BNF formában adott szintaxishoz léteznek eszközök:

- automatikusan generálnak egy elemzőt
  - a szintaktikus helyesség ellenőrzésére
  - levezetési fa készítésére
  - adott esetben a program lefordítására

Backus-Naur Form (BNF)

Példa

```
<loop statement> ::= <while loop> | <for loop>

<while loop> ::= while ( <condition> ) <statement>

<for loop> ::= for ( <expression> ; <expression>;
    <expression> ) <statement>

<assignment statement> ::=
    <variable> = <expression>
```

BNF Linkek

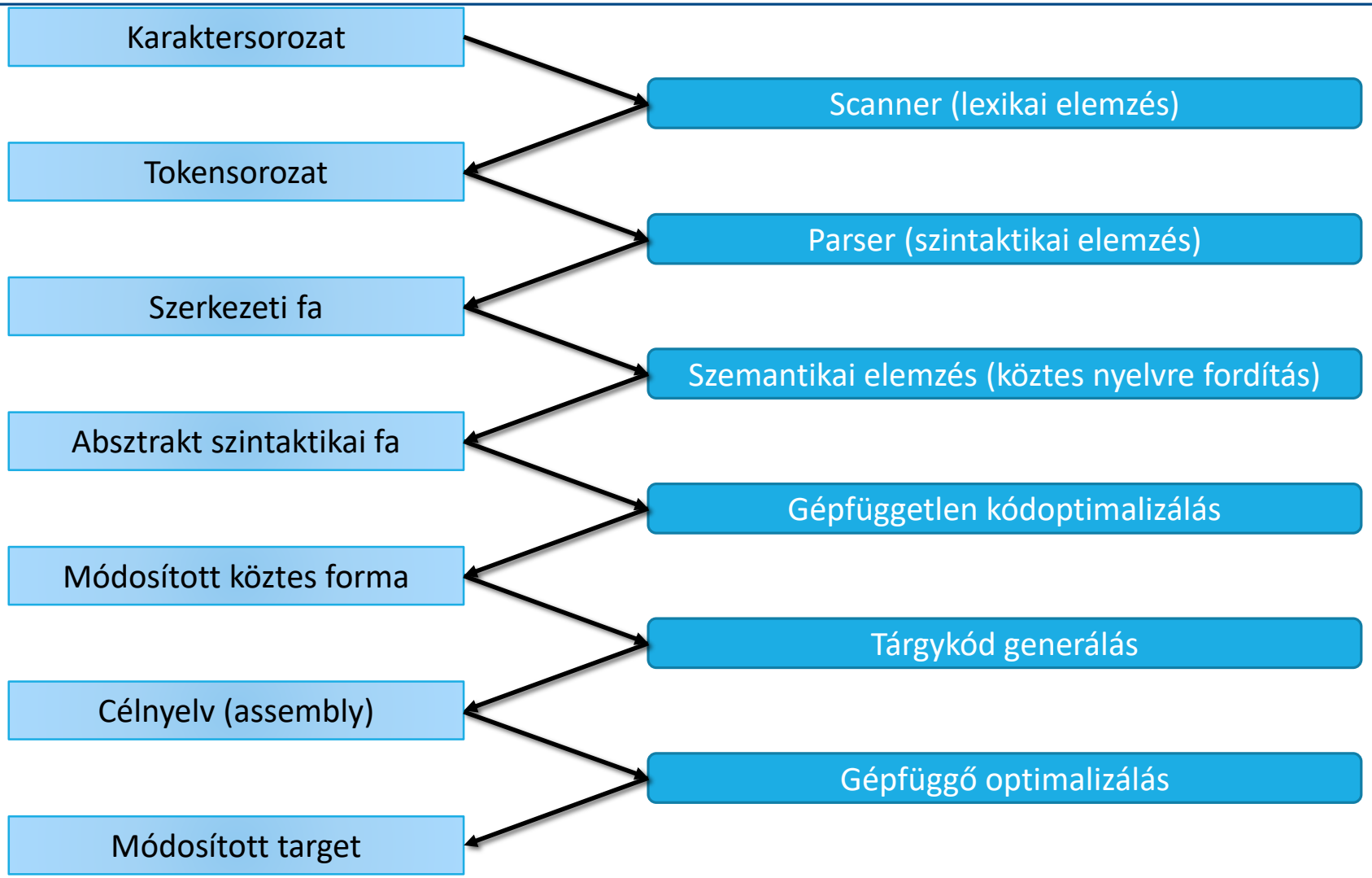
Java BNF

- https://docs.oracle.com/javase/specs/jls/se13/html/jls-19.html

Python BNF

- https://docs.python.org/3/reference/grammar.html

Fordítás


Karaktersorozat
Scanner (lexikai elemzés)
Tokensorozat
Parser (szintaktikai elemzés)
Szerkezeti fa
Szemantikai elemzés (köztes nyelvre fordítás)
Absztrakt szintaktikai fa
Gépfüggetlen kódoptimalizálás
Módosított köztes forma
Tárgykód generálás
Célnyelv (assembly)
Gépfüggő optimalizálás
Módosított target

Fordítóprogram

3. Lexikális elemzés
   - megadható reguláris nyelvtannal
2. Szintaktikus elemzés
   - megadható környezetfüggetlen nyelvtannal
1. Szemantikus elemzés
   - környezetfüggő
     - egy változó használatának helyessége függhet a deklarációjától, azaz a környezettől

A három lépés szétválasztásának oka
   - Egyszerű feladathoz ne használjunk bonyolult eszközöket.

Programszerkezet

Fordítási egység vagy modul

- A nyelvnek az az egysége, ami a fordítóprogram egyszeri lefuttatásával, a program többi részétől elkülönülten lefordítható
- Több modulból álló programok esetében a fordítás és a végrehajtás között: a program összeszerkesztése

Lexikai elemek

JÖVŐ HÉTEN