



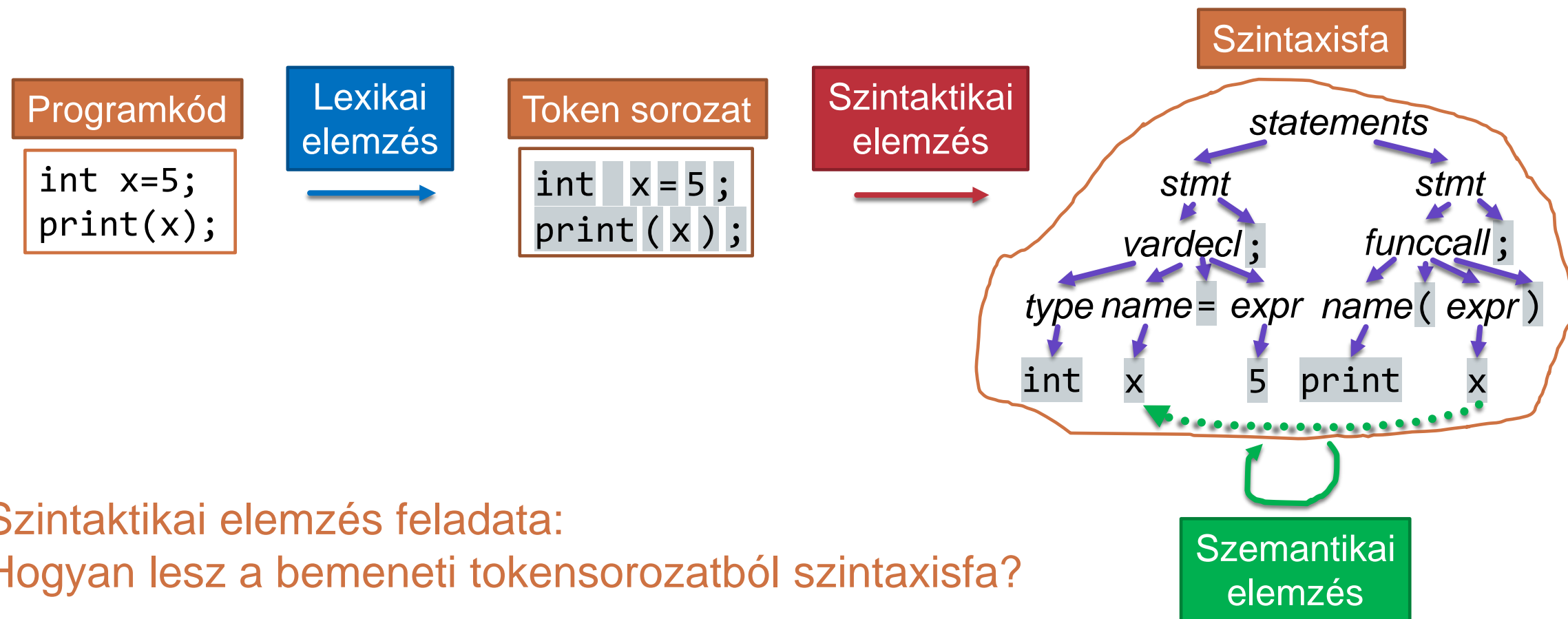
Modellalapú szoftverfejlesztés

IV. előadás

Szintaktikai elemzés

Dr. Simon Balázs

Fordító front-end



A mai előadás: Szintaktikai elemzés

I. Környezetfüggetlen (CF) nyelvtanok

II. Naív elemzési módszerek (BFS, DFS)

III. Balelemzés: $LL(k)$, $LL(*)$

IV. Jobbelemzés: $LR(k)$, LALR

V. Hibakezelés



Környezetfüggetlen (context-free, CF) nyelvtanok

Nemterminális

$A \rightarrow \alpha \mid \beta \mid \dots$

Tetszőleges

terminálisokból és nemterminálisokból
álló szimbólumsorozat

Formális nyelvek jelölései:

- Angol ABC nagybetűi: nemterminális szimbólumok
- Angol ABC kisbetűi, egyéb karakterek: terminálisok
- Görög ABC betűi: nemterminális-terminális sorozat

Levezetési fa:

- Gyökér: mondat-szimbólum (kezdőszimbólum)
- Közbenső csúcs: nemterminális szimbólum
- Levél: terminális szimbólum (token)

CF nyelvtan példa

Nyelvtani szabályok:

$E \rightarrow x|y|z|E \text{ Op } E|(E)$
 $Op \rightarrow +|-|*|/$

Mondat (programkód):

$x+y*z$

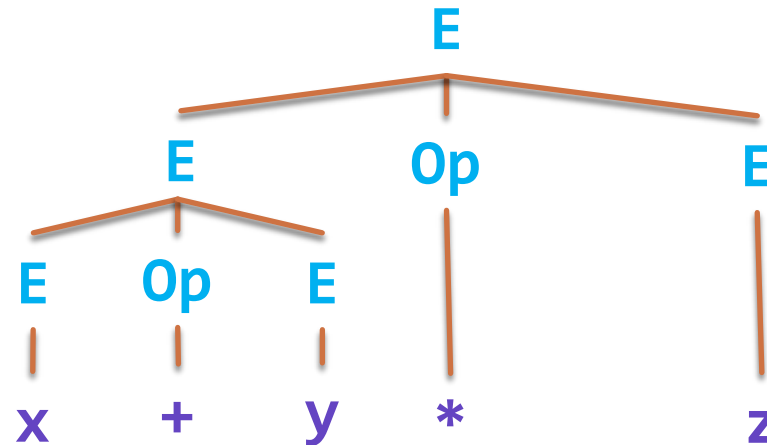
Lexikai elemzés (tokenek/terminálisok):

$x + y * z$

Egy lehetséges levezetés:

$E \xrightarrow{4} E \text{ Op } E \xrightarrow{4} E \text{ Op } E \text{ Op } E \rightarrow$
 $\xrightarrow{2} E \text{ Op } y \text{ Op } E \xrightarrow{6} E + y \text{ Op } E \rightarrow$
 $\xrightarrow{1} x + y \text{ Op } E \xrightarrow{8} x + y * E \rightarrow$
 $\xrightarrow{3} x + y * z$

A levezetéshez tartozó szintaxisfa:



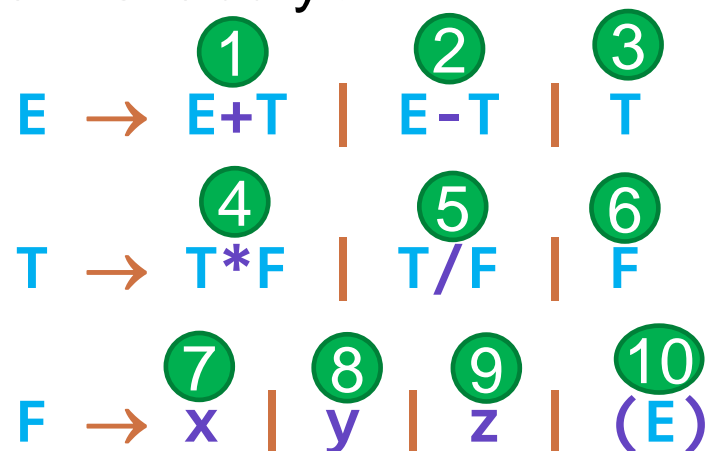
jelentés:
 $(x+y)*z$

Tanulságok

- Többféle levezetés is létezik
 - > a nyelvtan nem egyértelmű
- A levezetési fa struktúrája határozza meg a jelentést
 - > függ az alkalmazott szabályok sorrendjétől
 - > operátorok precedenciája
- Egyértelmű nyelvtan:
 - > a levezetési fa mindig egyértelmű

Egyértelmű nyelvtan

Nyelvtani szabályok:



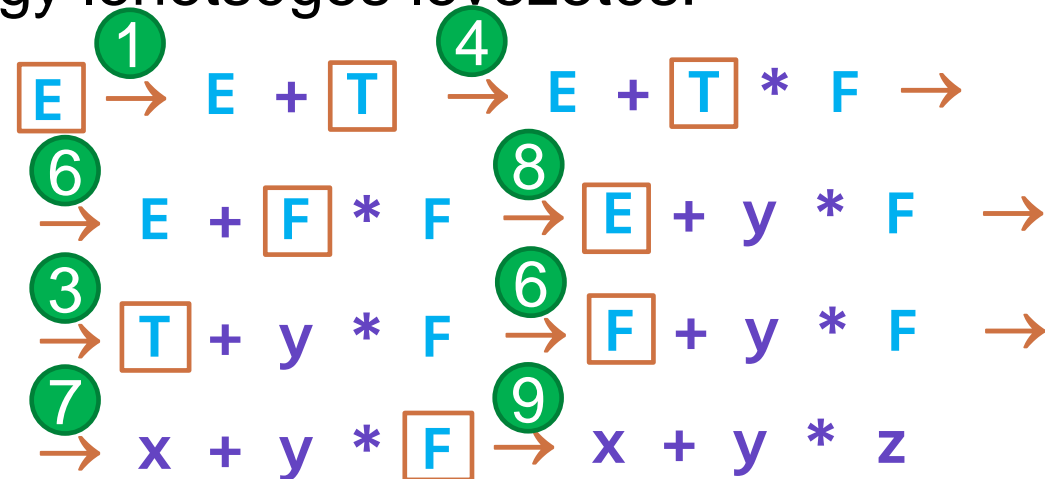
Mondat (programkód):

$x+y*z$

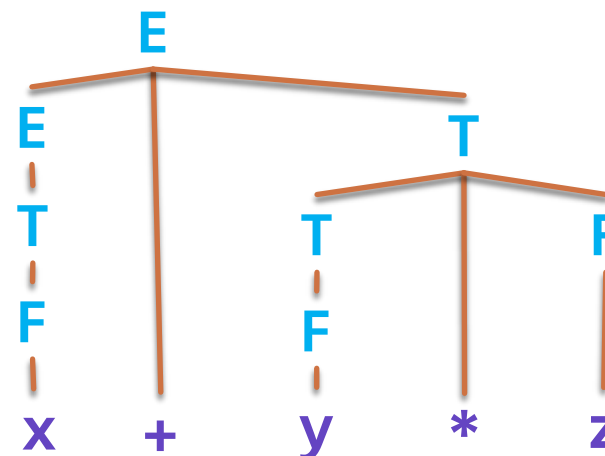
Lexikai elemzés (tokenek/terminálisok):

$x \ + \ y \ * \ z$

Egy lehetséges levezetés:



A levezetéshez tartozó szintaxisfa:



jelentés:
 $x+(y*z)$

Kérdések

- Hogyan lehet az elemzést és a szintaxisfa felépítését automatizálni?
 - > többféle algoritmus (elemző) létezik
- Hogyan lehet eldönteni, hogy egy nyelvtan egyértelmű-e?
 - > elemzőtől is függ, hogy az elemző készítésekor vagy elemzés közben vesszük-e észre
- Egyértelművé lehet-e tenni egy nyelvtant?
 - > általában igen, a levezetési szabályok átírásával
 - > vigyázni kell, mert a struktúra változhat

A mai előadás: Szintaktikai elemzés

I. Környezetfüggetlen (CF) nyelvtanok

II. Naív elemzési módszerek (BFS, DFS)

III. Balelemzés: $LL(k)$, $LL(*)$

IV. Jobbelemzés: $LR(k)$, LALR

V. Hibakezelés

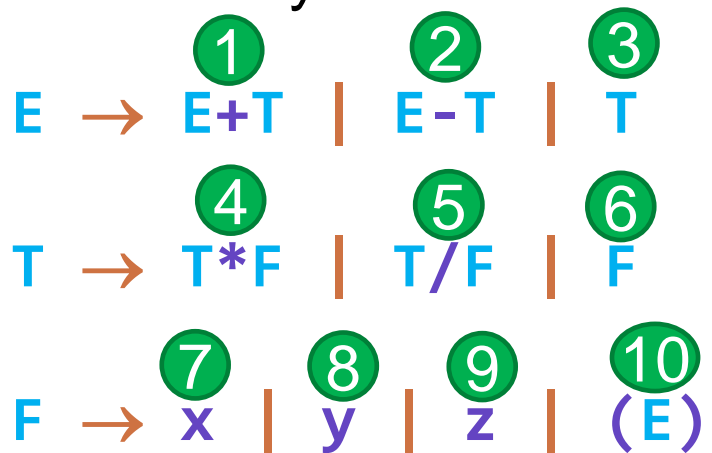


Naív elemzés

- Alapötlet:
 - > gráfkereső algoritmusok (BFS, DFS)
 - > visszalépéses keresés (backtracking)
- Kiindulás:
 - > mondatzimbólum
- Gráf csúcsai:
 - > olyan szimbólumsorozatok, amelyek egy vagy több lépésben levezethetők a mondatzimbólumból
- Gráf élei:
 - > α és β között akkor van él, ha β közvetlenül (egy lépésben) levezethető α -ból

Példa

Nyelvtani szabályok:



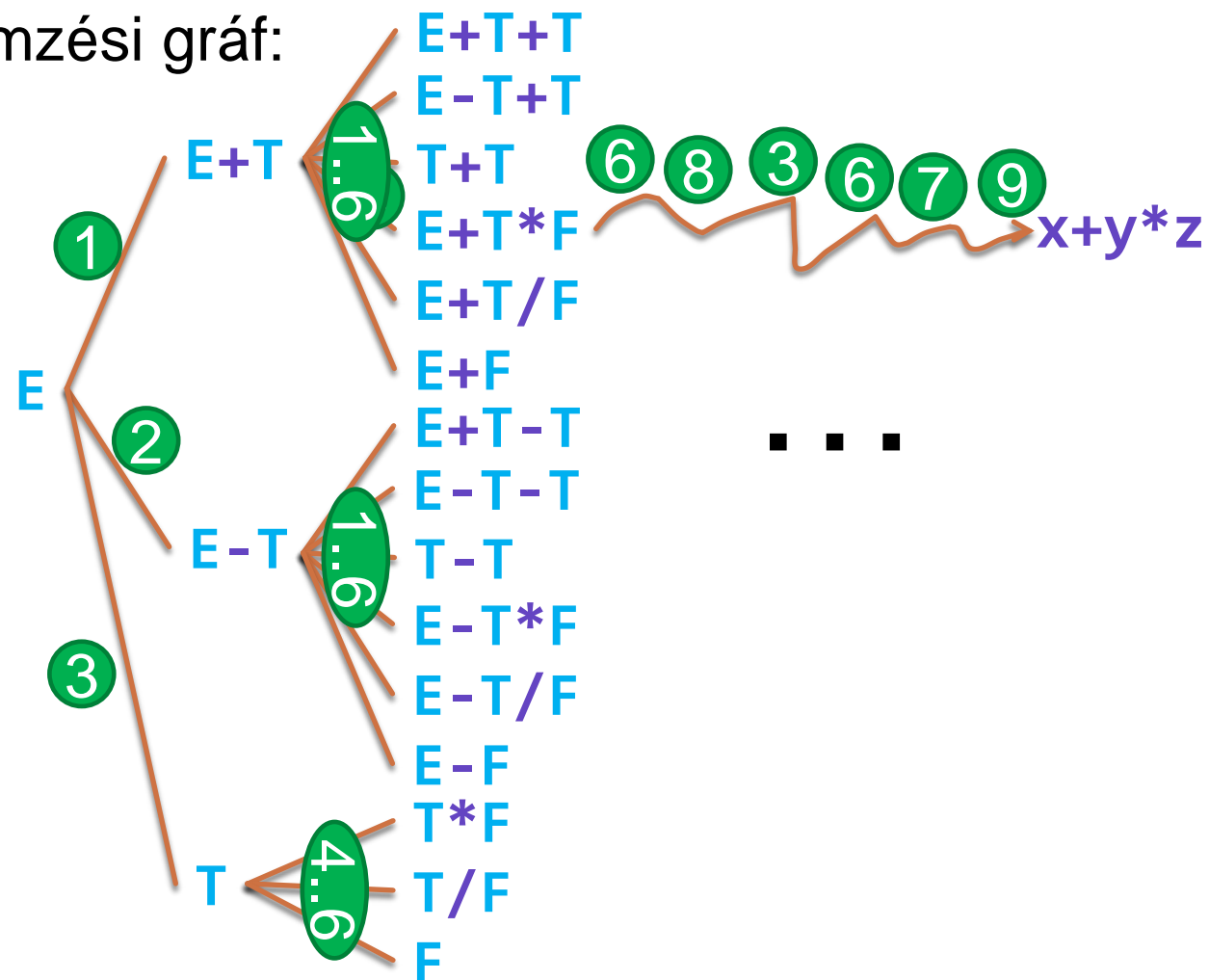
Mondat (programkód):

$$x + y * z$$

Lexikai elemzés (tokenek/terminálisok):

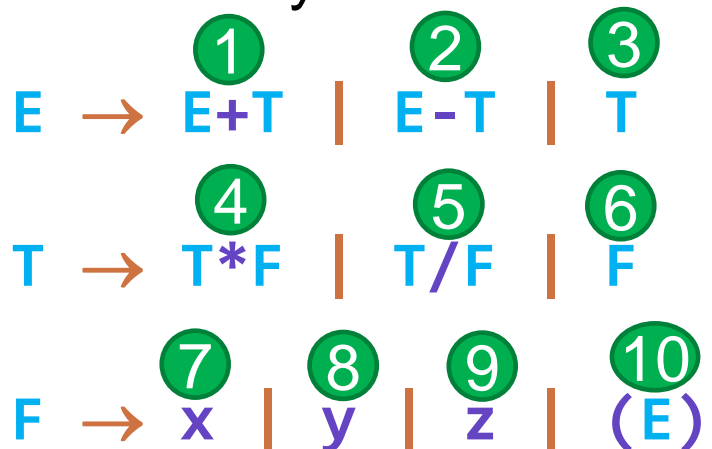
x + y * z

Elemzési gráf:



Példa: Szélességi keresés (BFS)

Nyelvtani szabályok:



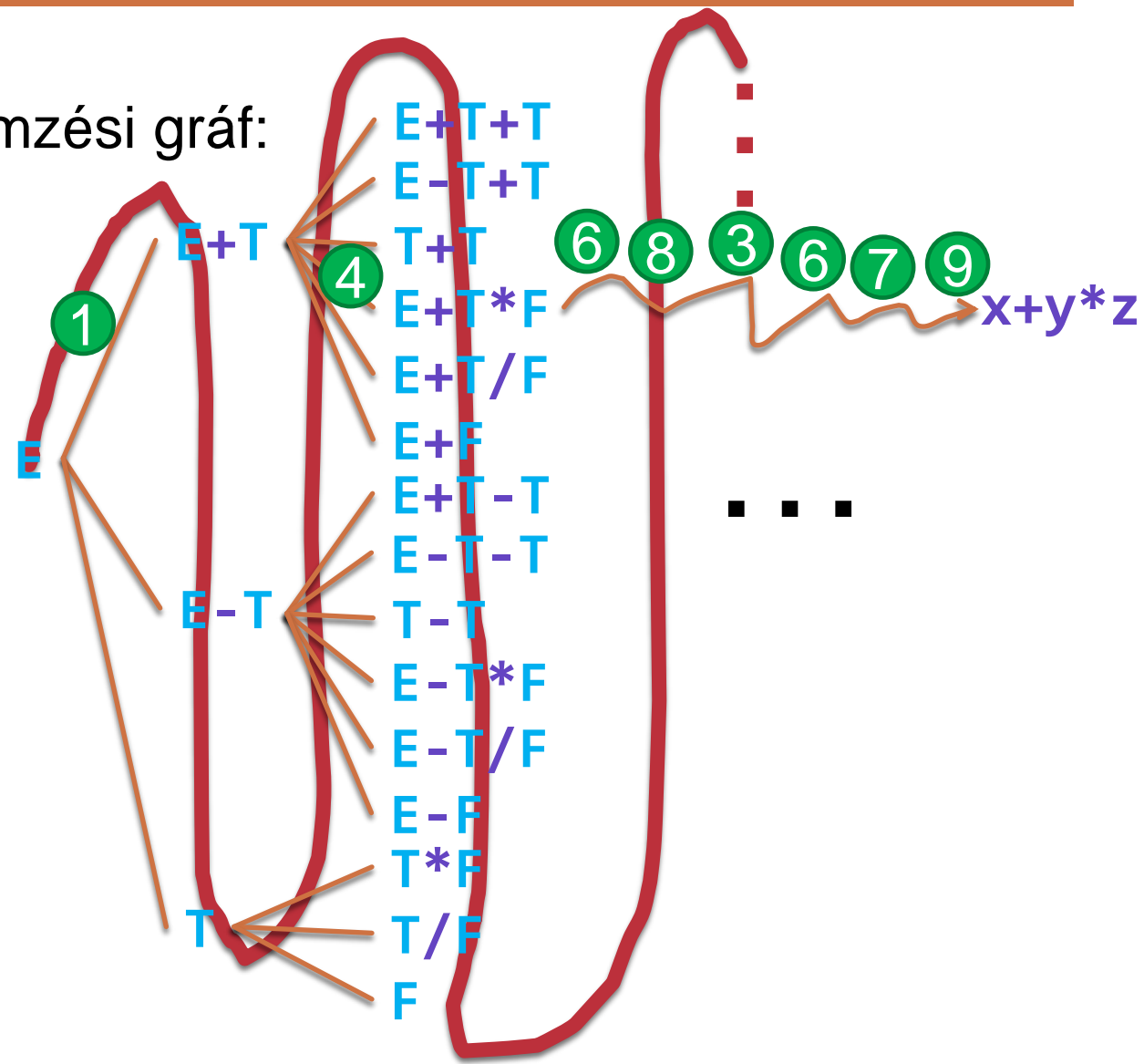
Mondat (programkód):

$x+y*z$

Lexikai elemzés (tokenek/terminálisok):

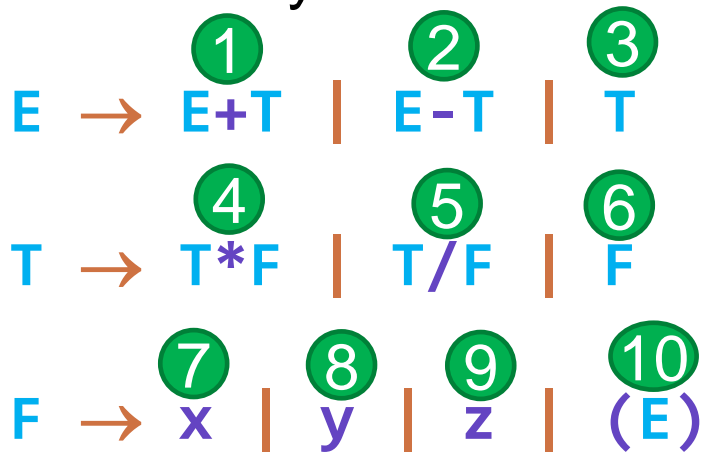
x **$+$** **y** **$*$** **z**

Elemzési gráf:



Példa: Mélységi keresés (DFS)

Nyelvtani szabályok:



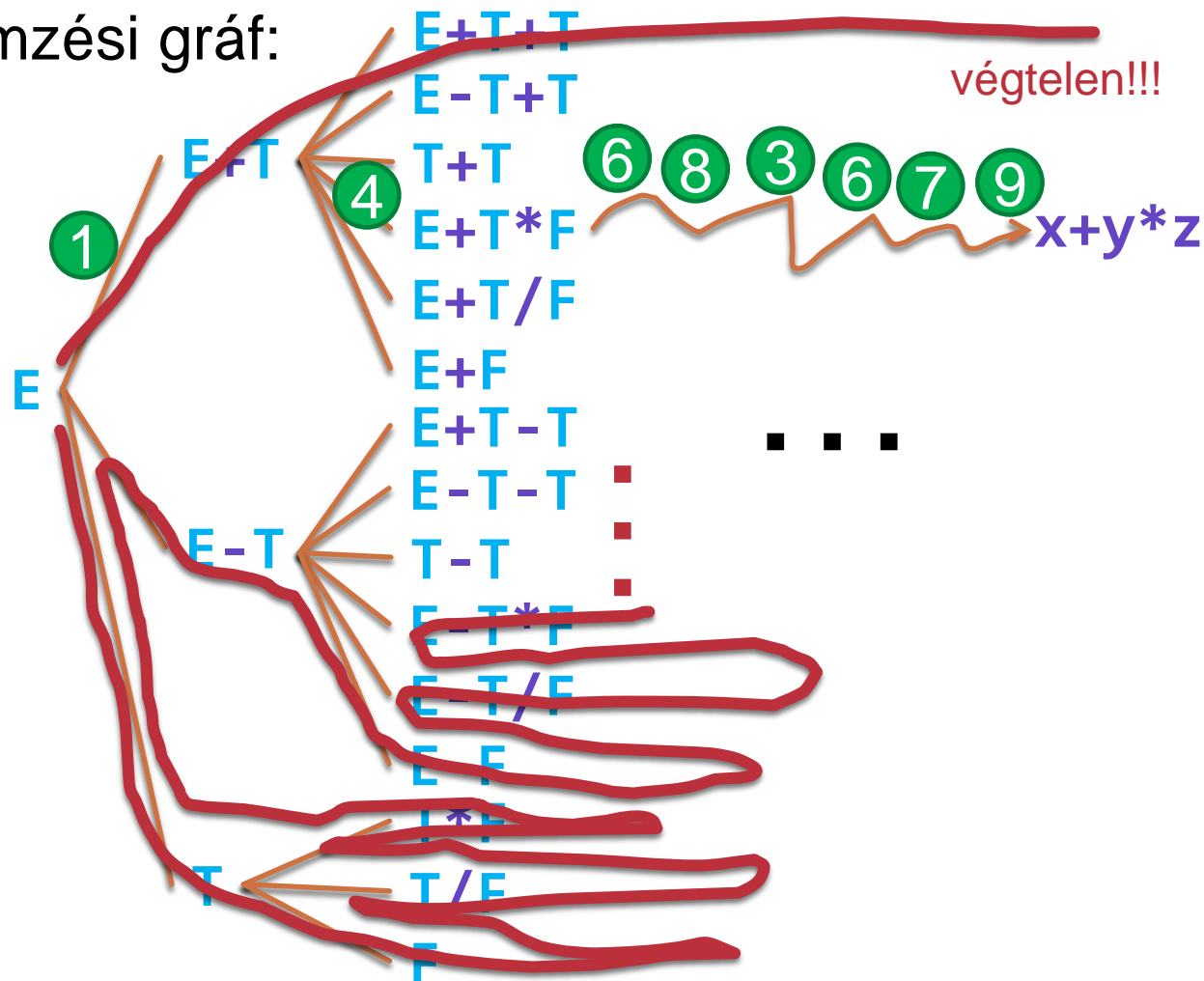
Mondat (programkód):

$$x + y * z$$

Lexikai elemzés (tokenek/terminálisok):

x + y * z

Elemzési gráf:



Tanulságok

- Keresési gráf \neq levezetési fa
 - > a levezetési fa rekonstruálható az élek (alkalmazott szabályok sorszámai) alapján
- Probléma: nagy keresési tér, sok elágazás
 - > BFS: akár exponenciális memória használat, akár exponenciális futási idő
 - > DFS: kisebb memóriaigény, de akár végtelen futási idő
 - probléma: balrekurzió ($E \rightarrow E+T \rightarrow E+E+T \rightarrow E+E+E+T \rightarrow \dots$)
 - megoldás: később...
- Célszerű a keresési teret szűkíteni
- Gyakorlatban ritkán használjuk

A mai előadás: Szintaktikai elemzés

I. Környezetfüggetlen (CF) nyelvtanok

II. Naív elemzési módszerek (BFS, DFS)

III. Balelemzés: $LL(k)$, $LL(*)$, $ALL(*)$

IV. Jobbelemzés: $LR(k)$, LALR

V. Hibakezelés



Balelemzés: LL(k)

- Visszalépéses keresés helyett előrejelzés (predikció)
- Alapötlet:
 - > L: balról jobbra haladunk (left-to-right)
 - > L: mindig a legbaloldalibb nemterminális kibontása (leftmost derivation)
 - > alternatívák közötti döntés: k db terminális (token) előreolvasása (lookahead) alapján
- Automatikus hibajelzés: az elkészült és az előrejelzett terminálisok ellenőrzése
 - > felesleges illetve hiányzó tokenek felismerése
- Probléma: balrekurzió

Balrekurzió példa

Nyelvtani szabályok:

$S \rightarrow Sa \mid b$

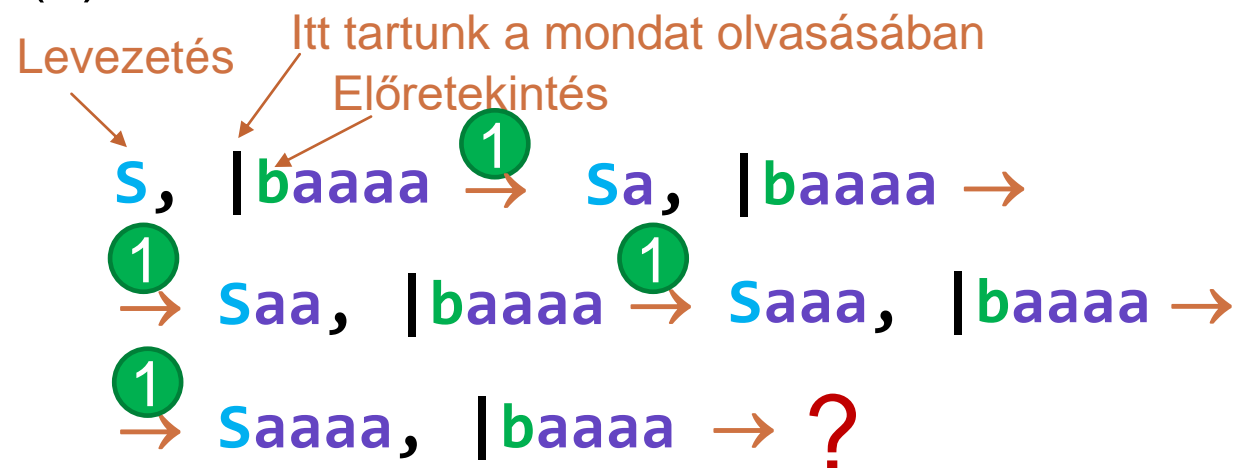
Mondat (programkód):

baaaa

Lexikai elemzés (tokenek/terminálisok):

b a a a a

LL(1) elemzés:



Itt kellene abbahagyni a 2-es szabállyal,
de az előretekintés alapján ezt nem tudjuk!

Balrekurzió feloldása: a nyelvtan átalakításával

Nyelvtani szabályok:

$S \rightarrow b\hat{S}$ (1)
 $\hat{S} \rightarrow a\hat{S} \mid \epsilon$ (2) (3)

Mondat (programkód):
baaaa

Lexikai elemzés (tokenek/terminálisok):

b a a a a

LL(1) elemzés:

Levezetés Itt tartunk a mondat olvasásában
Előretekintés

$S, \mid baaaa \xrightarrow{(1)} b\hat{S}, b \mid aaaa \rightarrow$
 $\xrightarrow{(2)} ba\hat{S}, ba \mid aaa \xrightarrow{(2)} baa\hat{S}, baa \mid aa \rightarrow$
 $\xrightarrow{(2)} baaa\hat{S}, baaa \mid a \rightarrow$
 $\xrightarrow{(2)} baaaa\hat{S}, baaaa \mid \epsilon \rightarrow$
 $\xrightarrow{(3)} baaaa, baaaa \mid \epsilon$

Balrekurzió

- 1. megoldás: nyelvtan átalakítása
 - > hátrány: megváltozik a struktúra!
- 2. megoldás: szabályok között prioritási sorrend
 - > ANTLR4: megengedett a közvetlen balrekurzió (de a közvetett nem!)

1. megoldás: nyelvtan átalakítása

$E \rightarrow E+T \mid E-T \mid T$
 $T \rightarrow T * F \mid T / F \mid F$
 $F \rightarrow x \mid y \mid z \mid (E)$

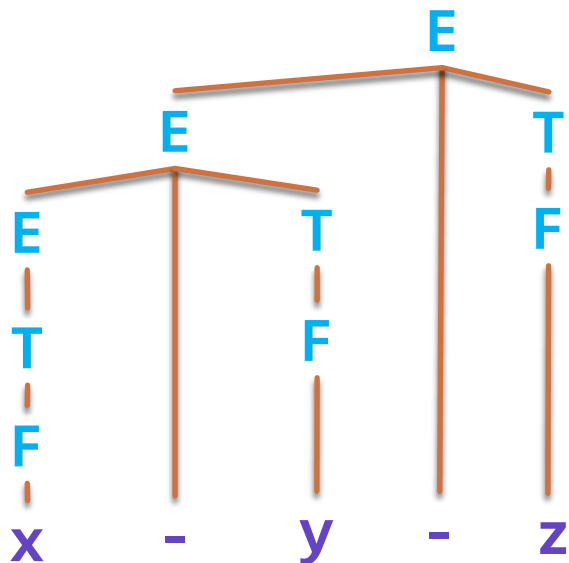


Kérdés: miért nem az alábbi módon alakítjuk át?

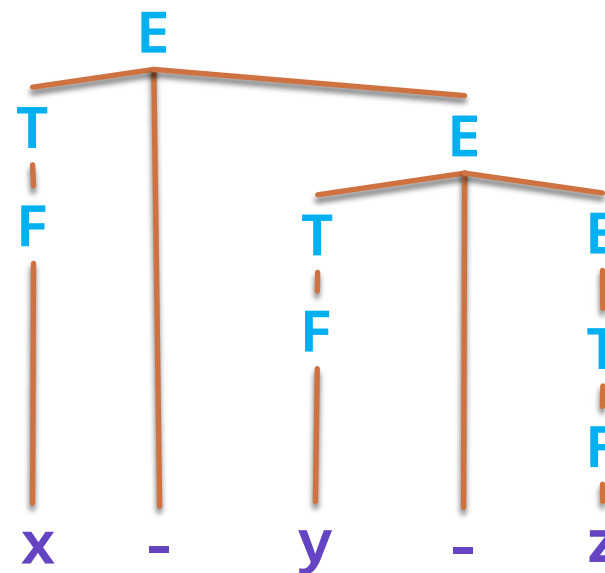
$E \rightarrow T+E \mid T-E \mid T$
 $T \rightarrow F * T \mid F / T \mid F$
 $F \rightarrow x \mid y \mid z \mid (E)$

Válasz: fontos az operátorok asszociativitása!

jelentés:
 $(x-y)-z$



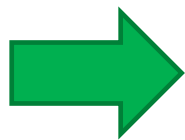
vs.



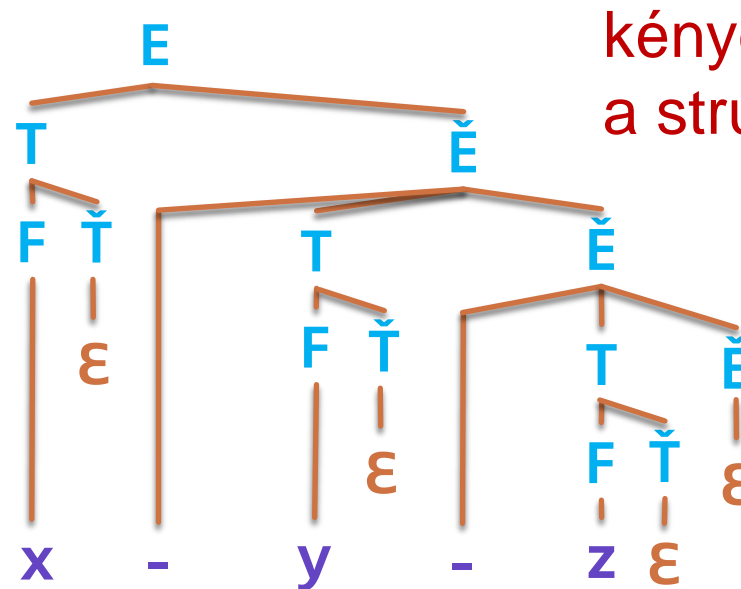
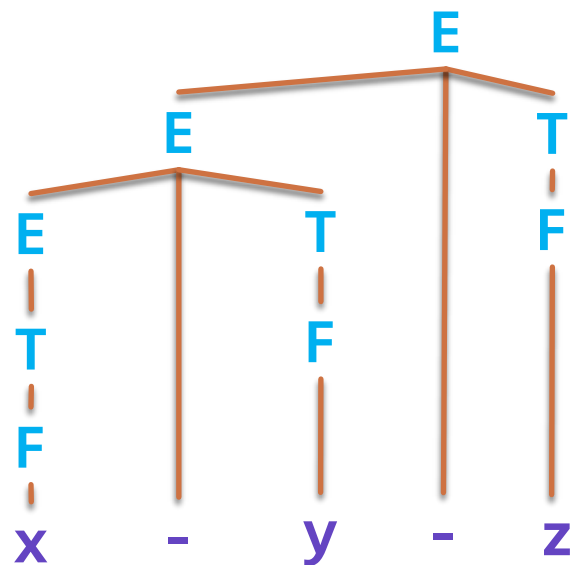
jelentés:
 $x-(y-z)$

1. megoldás: nyelvtan átalakítása helyesen

$E \rightarrow E+T \mid E-T \mid T$
 $T \rightarrow T * F \mid T / F \mid F$
 $F \rightarrow x \mid y \mid z \mid (E)$



$E \rightarrow T\check{E}$
 $\check{E} \rightarrow +T\check{E} \mid -T\check{E} \mid \epsilon$
 $T \rightarrow F\check{T}$
 $\check{T} \rightarrow *F\check{T} \mid /F\check{T} \mid \epsilon$
 $F \rightarrow x \mid y \mid z \mid (E)$



kényelmetlenség:
a struktúra más

LL(1) elemzés

Nyelvtani szabályok:

$$\begin{aligned} E &\rightarrow T\check{E} \\ \check{E} &\rightarrow +T\check{E} \mid -T\check{E} \mid \epsilon \\ T &\rightarrow F\check{T} \\ \check{T} &\rightarrow *F\check{T} \mid /F\check{T} \mid \epsilon \\ F &\rightarrow x \mid y \mid z \mid (E) \end{aligned}$$

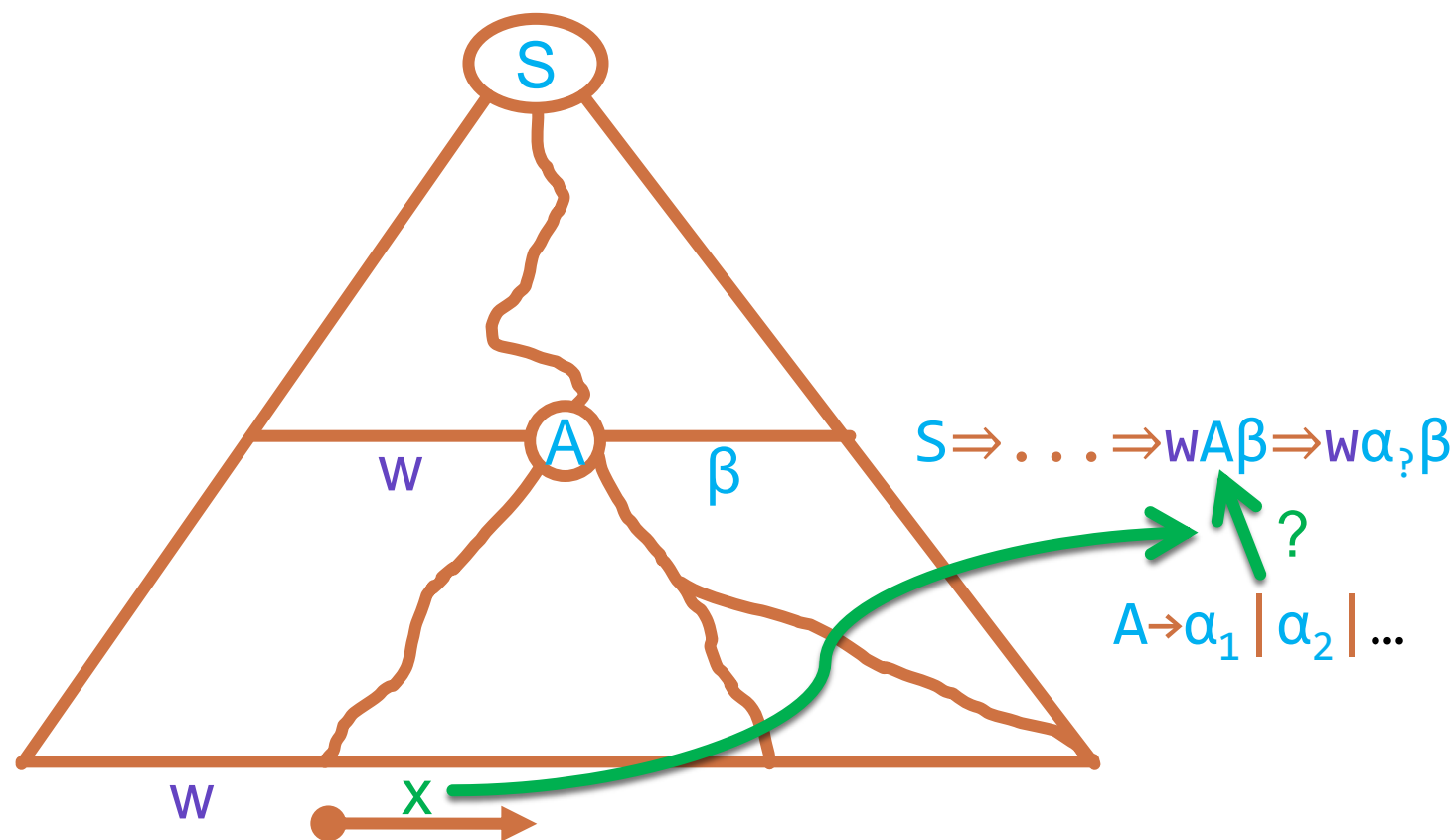
Mondat (programkód): $x+y*z$

Lexikai elemzés (tokenek): $x \ + \ y \ * \ z$

LL(1) elemzés:

$$\begin{aligned} E, \mid x+y*z &\xrightarrow{1} T\check{E}, \mid x+y*z \rightarrow \\ &\xrightarrow{5} F\check{T}\check{E}, \mid x+y*z \xrightarrow{9} x\check{T}\check{E}, x \mid +y*z \rightarrow \\ &\xrightarrow{8} x\check{E}, x \mid +y*z \xrightarrow{2} x+T\check{E}, x+ \mid y*z \rightarrow \\ &\xrightarrow{5} x+F\check{T}\check{E}, x+ \mid y*z \xrightarrow{10} x+y\check{T}\check{E}, x+y \mid *z \rightarrow \\ &\xrightarrow{6} x+y*F\check{T}\check{E}, x+y* \mid z \rightarrow \\ &\xrightarrow{11} x+y*z\check{T}\check{E}, x+y*z \mid \epsilon \rightarrow \\ &\xrightarrow{8} x+y*z\check{E}, x+y*z \mid \epsilon \rightarrow \\ &\xrightarrow{4} x+y*z, x+y*z \mid \epsilon \end{aligned}$$

LL(k) szemléltetése



Az A és a β által generált karaktereket is figyelembe kell venni a k hosszú x számításánál!

LL(k) elemzés előnyei

- Egyszerű
 - > manuálisan is könnyen programozható (pl. Roslyn – C# fordító): Recursive Descent Parser
 - > könnyen generálható is az elemző: táblázat alapján működik
- Gyors
- Kis memóriaigény
- Könnyű hibadetektálás és hibajelzés
 - > egyszerűbb hibák javíthatók, és az elemzés folytatható
 - > pl. egy felesleges token átugrása, egy hiányzó token beszúrása

Példa (1/3): LL(k) elemző programozása manuálisan

Nyelvtani szabályok:

$E \rightarrow T\check{E}$
 $\check{E} \rightarrow +T\check{E} \mid -T\check{E} \mid \epsilon$
 $T \rightarrow F\check{T}$
 $\check{T} \rightarrow *F\check{T} \mid /F\check{T} \mid \epsilon$
 $F \rightarrow x \mid y \mid z \mid (E)$

C# kód:



```
E ParseE()  
{  
    var t = ParseT();  
    var ehat = ParseEHat();  
    return new E(t, ehat);  
}
```

Példa (2/3): LL(k) elemző programozása manuálisan

C# kód:

Nyelvtani szabályok:

$E \rightarrow T\check{E}$
 $\check{E} \rightarrow +T\check{E} \mid -T\check{E} \mid \epsilon$
 $T \rightarrow F\check{T}$
 $\check{T} \rightarrow *F\check{T} \mid /F\check{T} \mid \epsilon$
 $F \rightarrow x \mid y \mid z \mid (E)$

```
EHat? ParseEHat() {  
    var la1 = LA(1);  Előrettekintés  
    switch (la1)  
    {  
        case "+":  
            Match("+");  Továbblépés  
            var t1 = ParseT();  
            var ehat1 = ParseEHat();  
            return new EHat(t1, ehat1);  
        case "*":  
            Match("*");  
            var t2 = ParseT();  
            var ehat2 = ParseEHat();  
            return new EHat(t2, ehat2);  
        default:  
            return null;  
    }  
}
```

Példa (3/3): LL(k) elemző programozása manuálisan

C# kód:

Nyelvtani szabályok:

$E \rightarrow T\check{E}$

$\check{E} \rightarrow +T\check{E} \mid -T\check{E} \mid \epsilon$

$T \rightarrow F\check{T}$

$\check{T} \rightarrow *F\check{T} \mid /F\check{T} \mid \epsilon$

$F \rightarrow x \mid y \mid z \mid (E)$

```
F? ParseF()
{
    var la1 = LA(1);
    switch (la1)
    {
        case "x": Match("x"); return new F("x");
        case "y": Match("y"); return new F("y");
        case "z": Match("z"); return new F("z");
        case "(":
            Match("(");
            var e = ParseE();
            Match(")"); ← Hibajelzés, ha más jön!
            return new F(e);
        default:
            Unexpected(la1); ← Hibajelzés, ha
                             nem várt token jön!
            return null;
    }
}
```

LL(*) elemzés

- Alapötlet:

- > fix **k** előretekintés helyett egy állapotgéppel becsüljük meg, melyik alternatíva fog nyerni
- > az állapotgép tetszőlegesen sok tokenet előre olvashat

- Előny: erősebb, mint az LL(k)

- Gyakorlatban:

- > ANTLR3 - LL(*): <http://wwwantlr.org/papers/LL-star-PLDI11.pdf>
- > ANTLR4 - ALL(*) (Adaptive LL): <http://wwwantlr.org/papers/allstar-techreport.pdf>
 - közvetlen balrekurziót is tud kezelni, alternatívák között precedencia

A mai előadás: Szintaktikai elemzés

I. Környezetfüggetlen (CF) nyelvtanok

II. Naív elemzési módszerek (BFS, DFS)

III. Balelemzés: $LL(k)$, $LL(*)$

IV. Jobbelemzés: $LR(k)$, LALR

V. Hibakezelés



Jobbelemzés: LR(k)

- Visszalépéses keresés helyett előrejelzés (predikció)
- Alapötlet:
 - > L: balról jobbra haladunk (left-to-right)
 - > R: mindig a levezetés jobb oldalát (verem tetejét) alakítjuk át (rightmost derivation in reverse)
 - > shift-reduce művelet közötti döntés: k db terminális (token) előreolvasása (lookahead) alapján
 - shift: a következő terminális a verem tetejére
 - reduce: a verem tetején egy potenciális jobb oldal lecserélése a szabály bal oldalára
- Automatikus hibajelzés: az első hibát képes pontosan megmutatni
 - > Potenciális továbbelemzéshez backtracking szükséges
- Balrekurzió nem okoz problémát

LR(1) elemzés

Nyelvtani szabályok:

$E \rightarrow E+T \mid E-T \mid T$
 $T \rightarrow T*F \mid T/F \mid F$
 $F \rightarrow x \mid y \mid z \mid (E)$

Mondat (programkód):

$x+y*z$

Lexikai elemzés (tokenek):

$x \ + \ y \ * \ z$

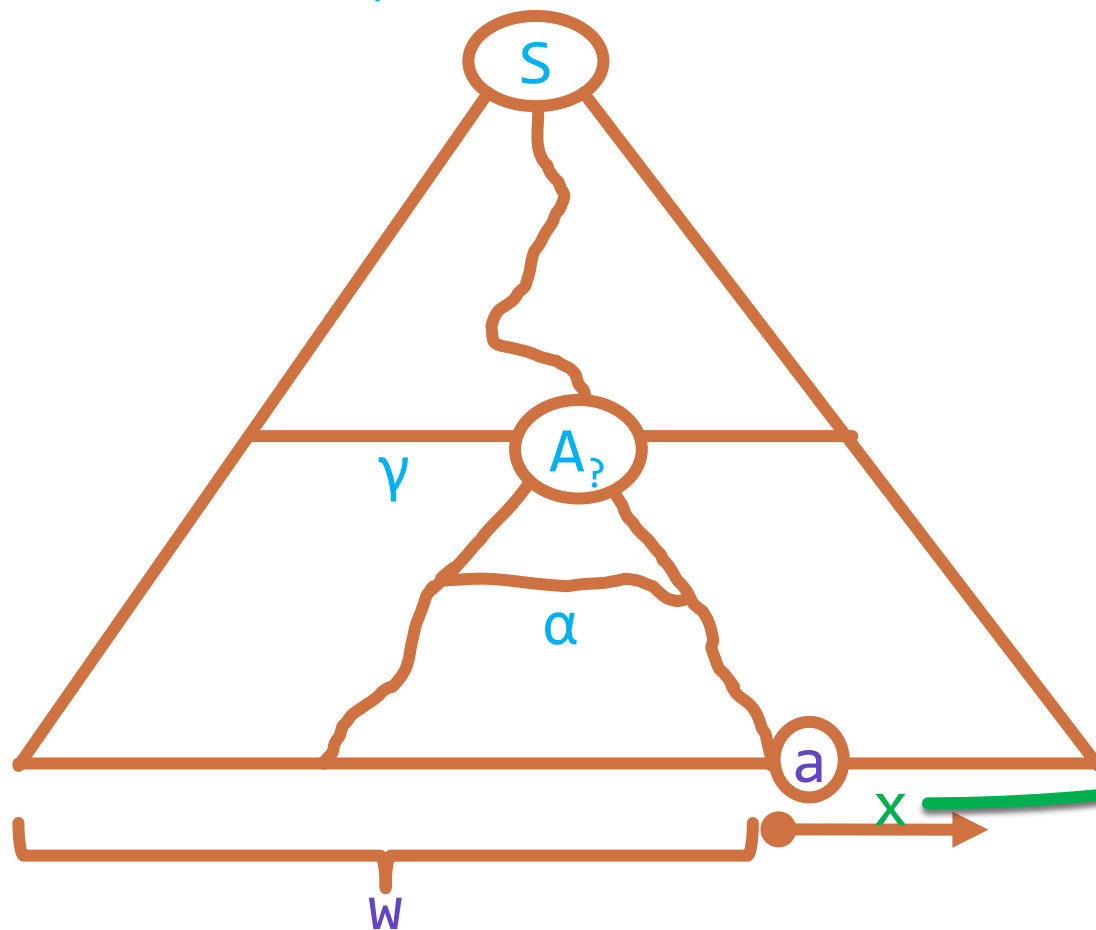
LR(1) elemzés:

Verem
Itt tartunk a mondat olvasásában
Előretekintés

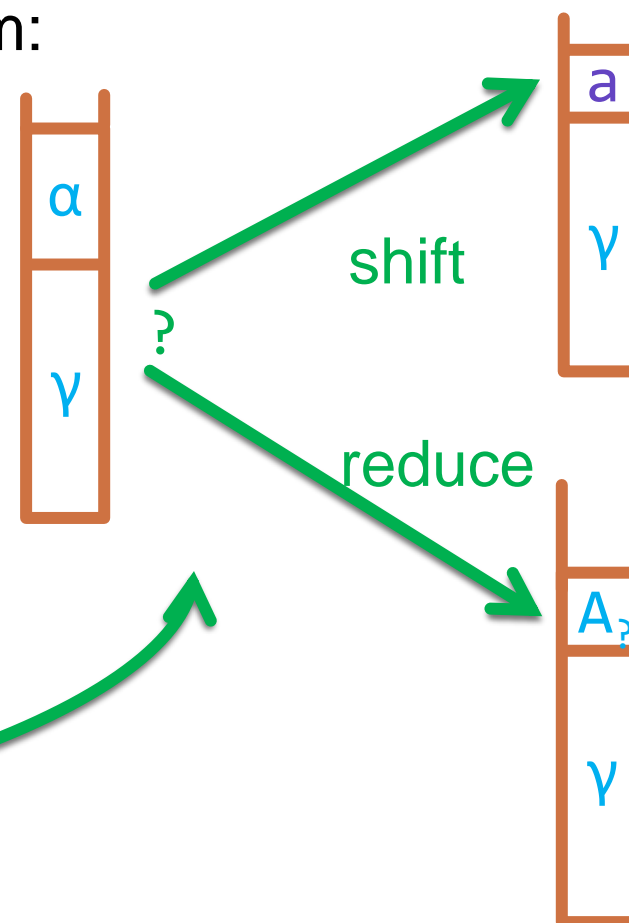
$\epsilon, \mid x+y*z \xrightarrow{S} x, x \mid +y*z \xrightarrow{7} F, x \mid +y*z \rightarrow$
 $\xrightarrow{6} T, x \mid +y*z \xrightarrow{3} E, x \mid +y*z \xrightarrow{S} E+, x+ \mid y*z \rightarrow$
 $\xrightarrow{S} E+y, x+y \mid *z \xrightarrow{8} E+F, x+y \mid *z \rightarrow$
 $\xrightarrow{6} E+T, x+y \mid *z \xrightarrow{S} E+T*, x+y* \mid z \rightarrow$
 $\xrightarrow{S} E+T*z, x+y*z \mid \epsilon \xrightarrow{9} E+T*F, x+y*z \mid \epsilon \rightarrow$
 $\xrightarrow{4} E+T, x+y*z \mid \epsilon \xrightarrow{1} E, x+y*z \mid \epsilon$

LR(k) szemléltetése

$S \Rightarrow \dots \Rightarrow \gamma A_? x \Rightarrow \gamma \alpha x \Rightarrow \dots \Rightarrow wx$



Verem:



Shift-Reduce konfliktus: Dangling Else

Mi a probléma az alábbi nyelvtannal?

Statement \rightarrow IfStatement | Expression

Expression \rightarrow IDENTIFIER

IfStatement \rightarrow

① IF Expression THEN Statement |

② IF Expression THEN Statement ELSE Statement

Megoldási lehetőségek:

1. Nyelvtan átalakítása
2. Shift v. reduce preferálása
3. Szabályok között precedencia

Hogyan elemzi a következő kódot?

IF Cond1 THEN

IF Cond2 THEN DoSomething1

② ELSE DoSomething2

Shift?

Hogyan elemzi a következő kódot?

IF Cond1 THEN

IF Cond2 THEN DoSomething1

ELSE DoSomething2

Reduce?

①

Shift-Reduce konfliktus feloldása: nyelvtan átalakításával

Statement \rightarrow IfStatement | Expression

Expression \rightarrow IDENTIFIER

IfStatement \rightarrow

IF Expression THEN Statement |

IF Expression THEN Statement ELSE Statement



Statement \rightarrow IfStatement | Expression

Expression \rightarrow IDENTIFIER

IfStatement \rightarrow

IF Expression THEN Statement |

IF Expression THEN IfThenElseStatement ELSE Statement

IfThenElseStatement \rightarrow

IF Expression THEN IfThenElseStatement ELSE IfThenElseStatement |
Expression

Itt mindig a legközelebbi IF-hez
kötjük az ELSE-t

LR(k) elemzés

■ Előnyök:

- > Erősebb, mint az LL(k)
- > Determinisztikus
- > Gyors: lineáris időben működik
- > Balrekurzió nem okoz gondot
- > Automatikusan generálható: táblázat alapján működik

■ Hátrányok:

- > A táblázat nagyon nagy, már LR(1) esetén is
- > Shift-reduce konfliktusokat fel kell oldani

LALR elemzés

- LALR = Look-Ahead LR parser
- Egyszerűsített jobbelemező
 - > LR(1) elemzést végez: erősebb, mint az LR(0)
 - > LR(0) állapotterével: kisebb állapotter, mint LR(1) esetén
- Automatikusan generálható
 - > Pl. Yacc, Bison

A mai előadás: Szintaktikai elemzés

I. Környezetfüggetlen (CF) nyelvtanok

II. Naív elemzési módszerek (BFS, DFS)

III. Balelemzés: $LL(k)$, $LL(*)$

IV. Jobbelemzés: $LR(k)$, LALR

V. Hibakezelés



Hibakezelés

- Szintaktikai elemzés célja:
 - > helyes eredmény vagy minél több szintaktikai hiba megtalálása
- Hiba esetén:
 - > hibaüzenet, visszaállítás egy konzisztens állapotra, majd az elemzés folytatása
- Jelezni kell:
 - > pozíció (fájlnev, sor, oszlop)
 - > hiba osztálya (hiba, figyelmeztetés, megjegyzés, stb.)
 - > hibaüzenet

Hiba helye vs. Hibajelenség pozíciója

$x = (a+b*c;$

Hibajelenség:
) hiányzik a ; előtt

Hiba (valószínűleg):
) hiányzik a * előtt

A hibajelenség pozíciója nem feltétlenül egyezik meg a hiba helyével!

Fordítás különböző fázisaiban előforduló hibák

- Lexikai hiba:
 - > pl. hibás karakter, karakterlánc vége vagy komment vége hiányzik, túl korai fájlvége
- Szintaktikai hiba:
 - > a kód nem felel meg a megadott nyelvtannak
 - > javítás: minél kevesebb művelettel (token beszúrása, törlése, cseréje)
- Szemantikai hiba:
 - > hiba a statikus szemantikában (pl. típushiba)
 - > van olyan, amely csak az optimalizálás során észlelhető (pl. kiindexelés)
- Erőforráshiány:
 - > bármelyik fázisban jelentkezhet



Köszönöm a figyelmet!