Szintaktikai elemzés

Simon Balázs

BME IIT, 2011.

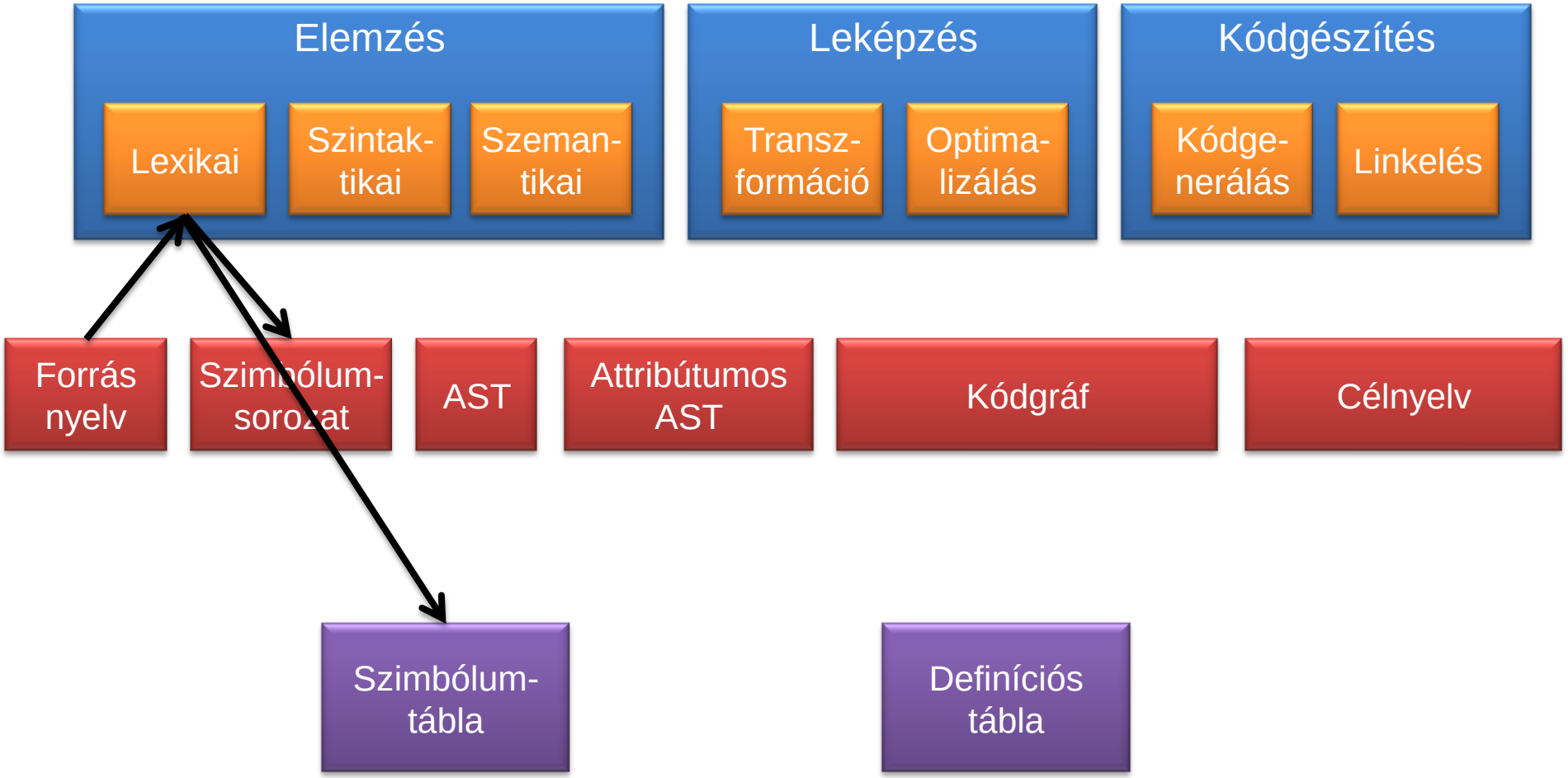
forrás: http://www.info.uni-karlsruhe.de/lehre/2007WS/uebau1/

Tartalom

- Lexikai elemzés
- Szintaktikai elemzés
  - balelemzés
  - jobbelemzés
  - hibakezelés

Lexikai elemzés

Lexikai elemzés



Lexikai elemzés

- A forráskód feldarabolása jelentéssel bíró egységekre: szimbólumokra
- Felesleges karakterek figyelmen elhagyása:
  - white-space
  - kommentek
- Megvalósítás: véges automata
- Nagyfokú tömörítés: byte-ok helyett szimbólumok
- A fordítás külön szakasza:
  - a mai programnyelvek így lettek megtervezve
  - a véges automata hatékonyabb, mint a veremautomata

Kivételek

- Fortran 77
  - `READ 5,ggg,…`
  - itt a `ggg` formázást jelent, nem azonosítót
- C
  - `#pragma …`
  - tartalma bármi lehet
- Egyes programnyelvekben a karakterláncok

Bemenet kezelése

- Karakterenként:
  - túl sok rendszerhívás, túl lassú
  - csak billentyűzetről való olvasáskor célszerű
- Soronként:
  - generált kód esetén lehet egyetlen sor az egész program
- Pufferelés
- Ma:
  - az egész fájl a virtuális memóriában
  - sok fájl esetén nagy memóriaigény

Szimbólumok felismerése

- Véges automata:
  - karakterek:
    - az automata által elfogadott karaktersorozatok lehetséges karakterei
  - állapotok 
  - kezdőállapot 
  - elfogadó állapotok 
    - amelyekben megállva az automata elfogadja a beolvasott karaktersorozatot
  - átmenetek 

    - állapotok közötti váltást adják meg az adott karakter hatására

Véges automata

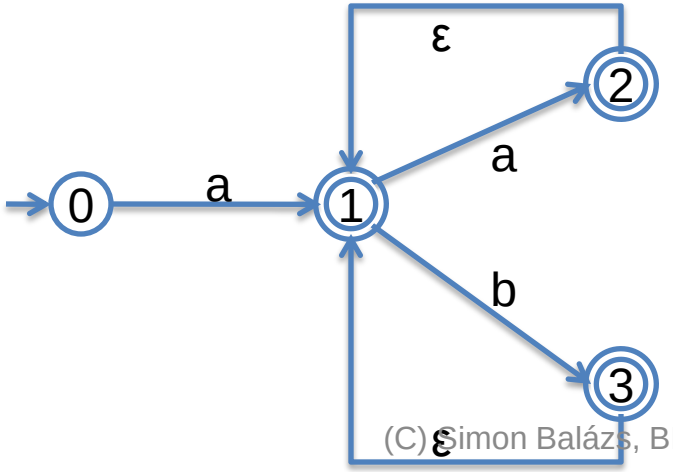
- Teljesen specifikált automata:
  - egy állapotból minden karakterre van átmenet definiálva
- Determinisztikus automata:
  - egy állapotból egy karakter hatására legfeljebb egy átmenet lehetséges
- Minimálautomata:
  - a legkevesebb állapottal rendelkező teljesen specifikált és determinisztikus automata
  - egyértelmű:
    adott nyelv esetén pontosan egy ilyen létezik

Reguláris kifejezés

- Karakterek
- Zárójelek: csoportosítás
- Vagy: | (függőleges vonal)
- Ismétlés:
  - Nulla vagy egy: ? (kérdőjel)
  - Nulla vagy több: * (csillag)
  - Egy vagy több: + (plusz)
- Példa: a(b|c)*d+
  - helyes: ad, add, abd, abcbbbccdddd
  - helytelen: a, dddd, abcdbb

Reguláris kifejezésből véges automata

- Példa: a(a|b)*
- Bevezetve az alábbi állapotokat:
  - $_0a_1(a_2|b_3)^*$
- Az átmenetek:
  - (0,a)->1, (1,a)->2, (1,b)->3, (2,ε)->1, (3,ε)->1
- Kezdőállapot: 0
- Elfogadó állapotok: 1, 2, 3



Lexer definiálása reguláris kifejezésekkel

- White-space
- Kommentek
- Kulcsszavak (pl. if, class, private, stb.)
- Számok
- Azonosítók
- stb.

- Mindegyikre létrejön egy véges automata, amely az adott szimbólumot el tudja fogadni

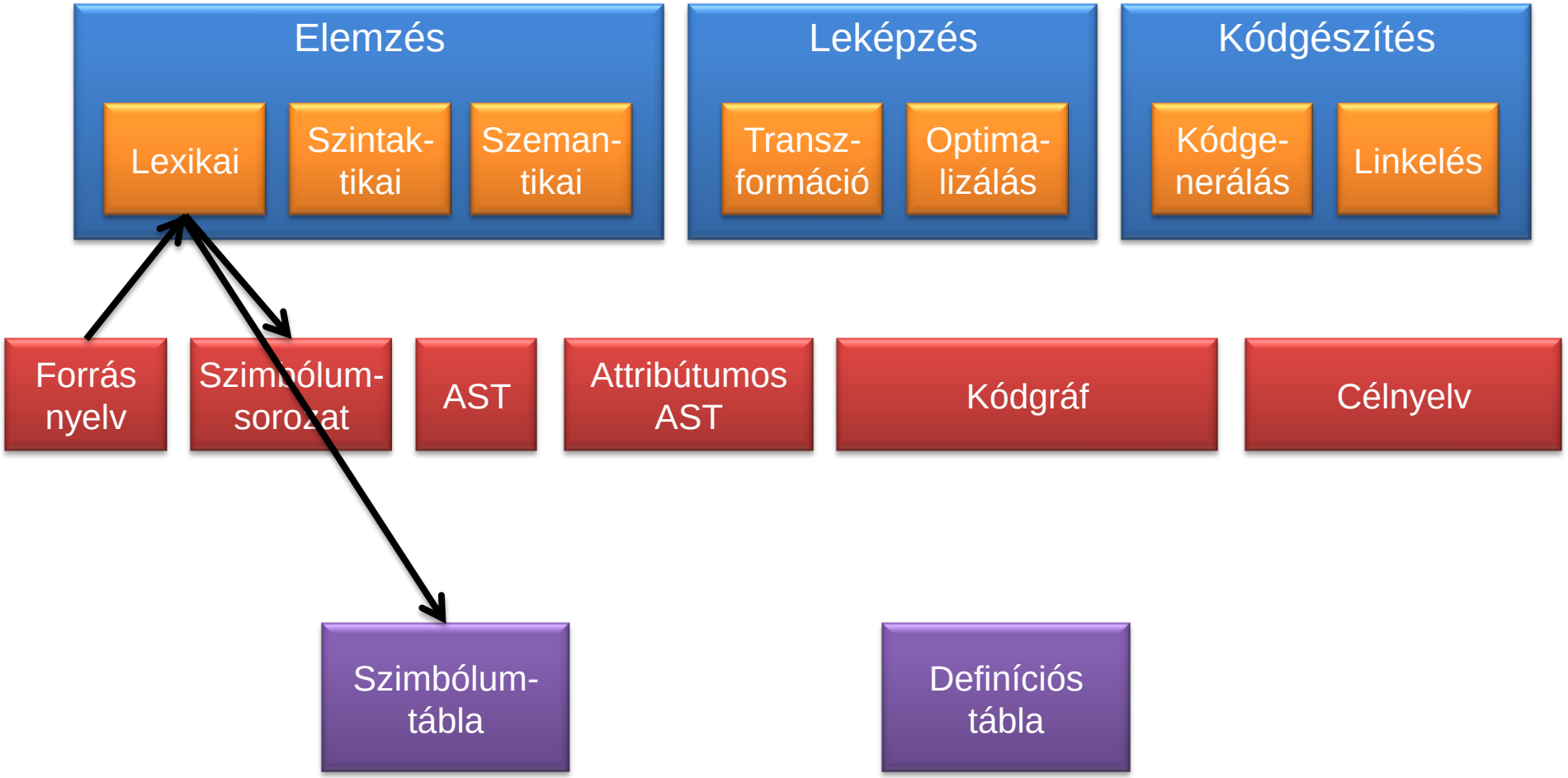
Lexer generátorok

- A lexer automatikusan generálható
- Ehhez definiálni kell a véges automatákat
- Eredmény: táblázat vagy programkód
- Lexer generátorok:
  - Lex, Flex, stb.
  - mi JFlex-et fogunk használni

Szimbólumtábla

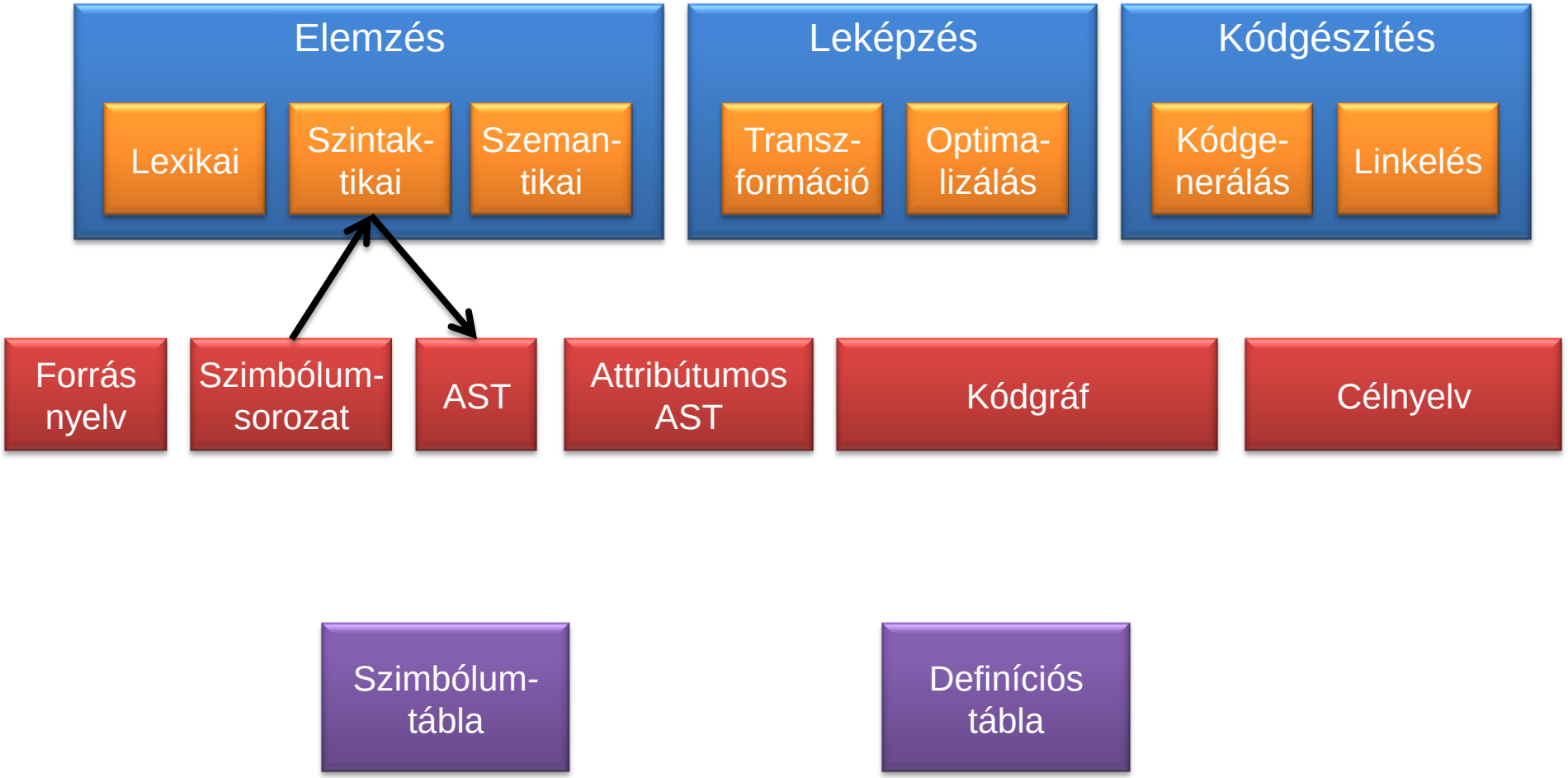
- Azonosítók (identifier) és konstans értékek (literal) leképzése a fordító belső reprezentációjára
- Egy táblabejegyzés:
  - szimbólum (token) és érték (value)
  - valamint sor- és oszlopindex a forráskódban (hibajelzéshez)
- Lexer építi fel
- Felépítés után végig változatlan marad a fordítás során

Lexikai elemzés



Szintaktikai elemzés

Szintaktikai elemzés



Szintaktikai elemzés

- Előállítja a programkód fastruktúráját
  - Concrete Syntax Tree (CST)
  - Abstract Syntax Tree (AST)
- Bemenete: szimbólumsorozat
- Kontextusfüggetlen (CF = context-free) nyelvtan alapján dolgozik
  - ezt könnyű elemezni
  - viszonylag gyorsan
  - elég nagy kifejezőerő
  - tipikus elemző automaták: LL, LR, LALR

CF nyelvtan

- Elméletben:
  - $A \rightarrow \alpha|\beta|\ldots$
  - ahol A nemterminális
  - α, β, … pedig tetszőleges terminálisokból és nemterminálisokból álló karaktersorozat
- Gyakorlatban:
  - BNF (Bachus-Naur forma)
  - EBNF (kiterjesztett BNF)

BNF

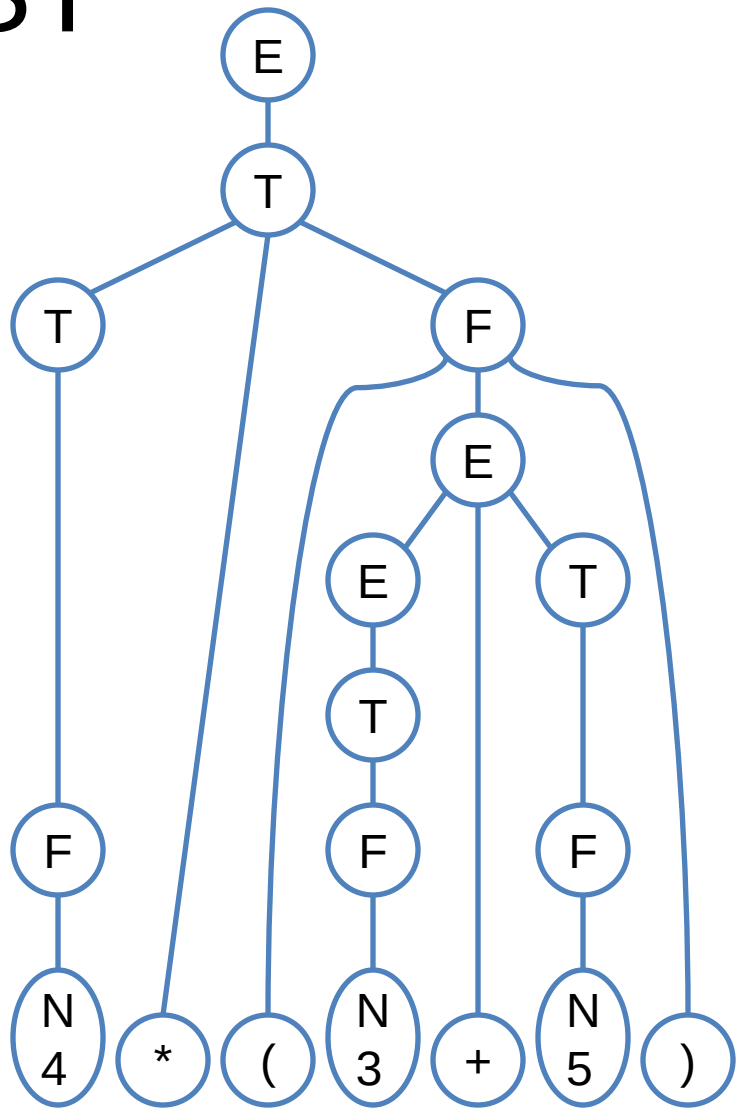
- Nemterminális: <…> között
- Terminális: egyszerű karakterek
- Nyíl helyett: ::=
- Példa:
  - <Expression> ::= <Expression> + <Term> | <Term>
  - <Term> ::= <Term> * <Factor> | <Factor>
  - <Factor> ::= <Number> | (<Expression>)

EBNF

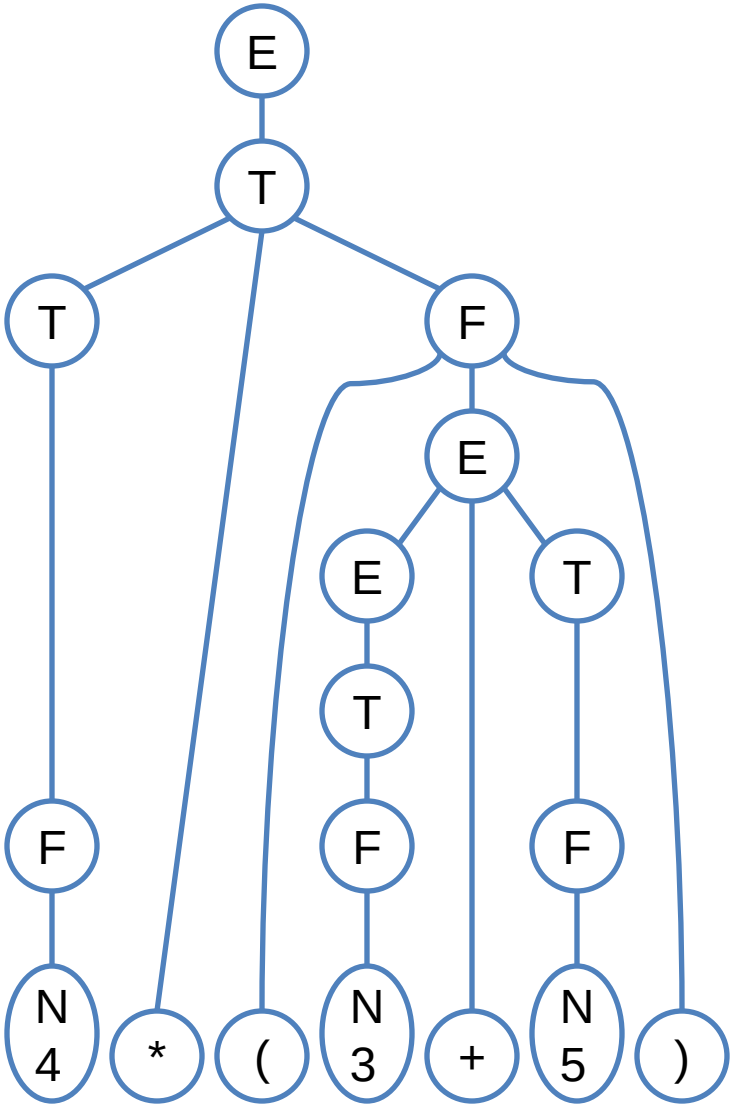
- Olyan, mint a BNF, bizonyos változtatásokkal
- Nemterminális: kacsacsőrök elhagyva
- Terminális: aposztrófok között
- Vagy: |
- Szabály vége: .
- Csoportosítás: (…)
- Opcionális elem: […]
- Ismétlés nullaszor vagy többször: *
- Ismétlés legalább egyszer: +
- Példa:
  - Expression ::= Term (‘+’ Term)* .
  - Term ::= Factor (‘*’ Factor)* .
  - Factor ::= Number | ‘(‘ Expression ‘)’ .

Példa: CST

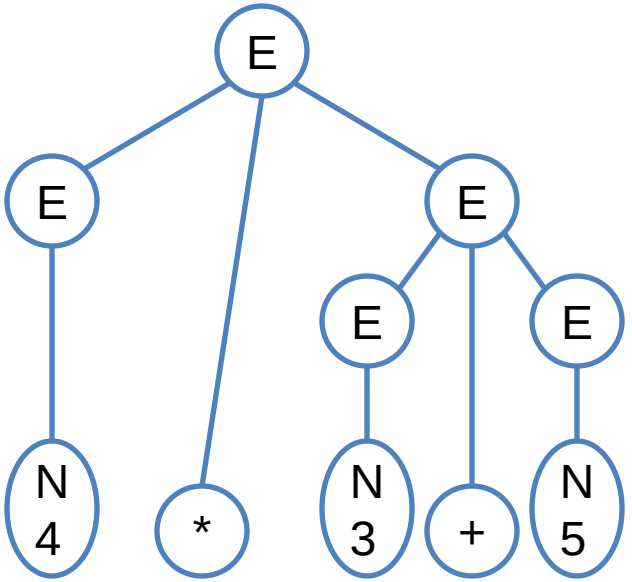
- BNF:
  - <E> ::= <E> + <T> | <T>
  - <T> ::= <T> * <F> | <F>
  - <F> ::= <N> | (<E>)
- EBNF:
  - E ::= T (‘+’ T)* .
  - T ::= F (‘*’ F)* .
  - F ::= N | ‘(‘ E ‘)’ .
- Kifejezés: 4*(3+5)



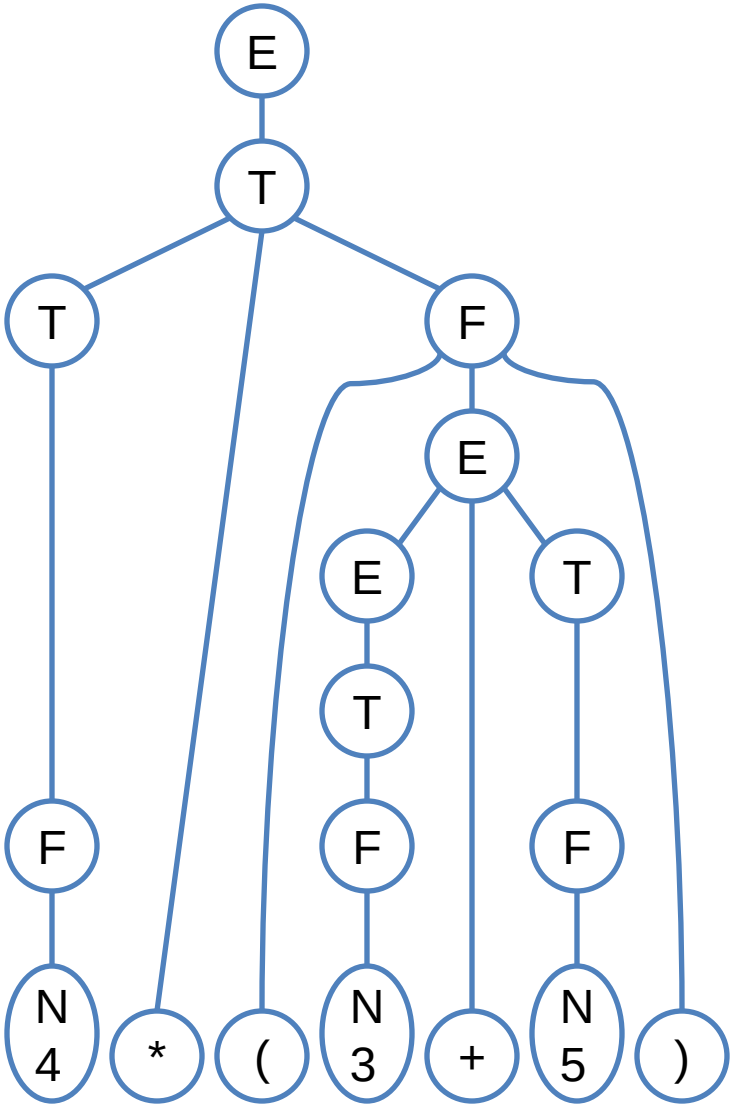
Példa: CST → AST



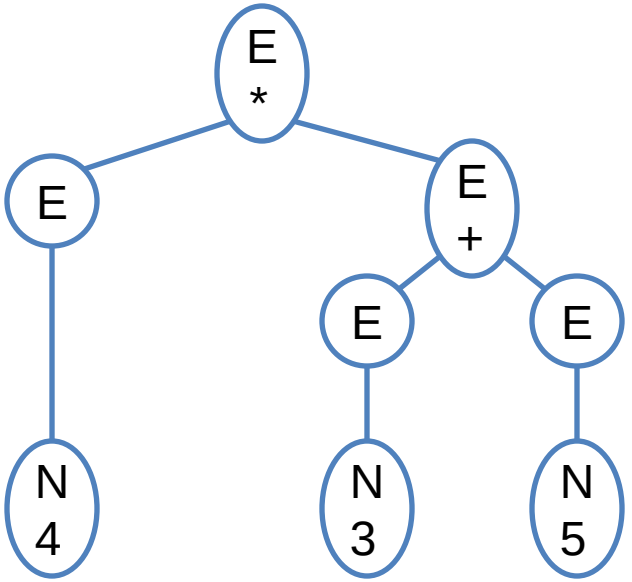
- Csak a jelentés szempontjából érdekes elemek megtartása:
- E → E+E | E*E | N



Példa: CST → AST



- Csak a jelentés szempontjából érdekes elemek megtartása:
- E → E+E | E*E | N



CST és AST

- Konkrét szintaxis:
  (adatstruktúra: szimbólumsorozat)
  - Explicit struktúrainformáció: (…), begin…end, stb.
  - Láncszabályok (A→B)
  - Választási szabályok (A→B|C|…)
  - Kulcsszavak
- Absztrakt szintaxis:
  (adatstruktúra: AST fa)
  - Zárójelezést és struktúrát a fahierarchia adja
  - Láncszabályok feleslegesek, ha nincs jelentéstartalmuk
    - kivétel: A→azonosító (a szemantikai analízis miatt szükséges)
  - Kulcsszavak általában feleslegesek, így hiányoznak (pl. class, begin, stb.)

Absztrakt szintaxis

- Kérdés: függ-e a programnyelvtől?
  - programstruktúrát a szemantikus analízis fel tudja dolgozni
  - utána már csak a függvényhívás az érdekes, ott is csak a paraméterátadás
  - vezérlés: mindenhol ugyanolyan
    - kivétel esetleg: számláló cikusok
  - kivételkezelés: minden modern programnyelv ugyanolyan
  - értékadás, kifejezések, operátorok: ugyanolyanok
- Következmény:
  - a további feldolgozás (transzformáció, optimalizálás, kódgenerálás) független a forrásnyelvtől
  - jó példák: UNCOL, .NET

Elemzés CF nyelvtanok segítségével

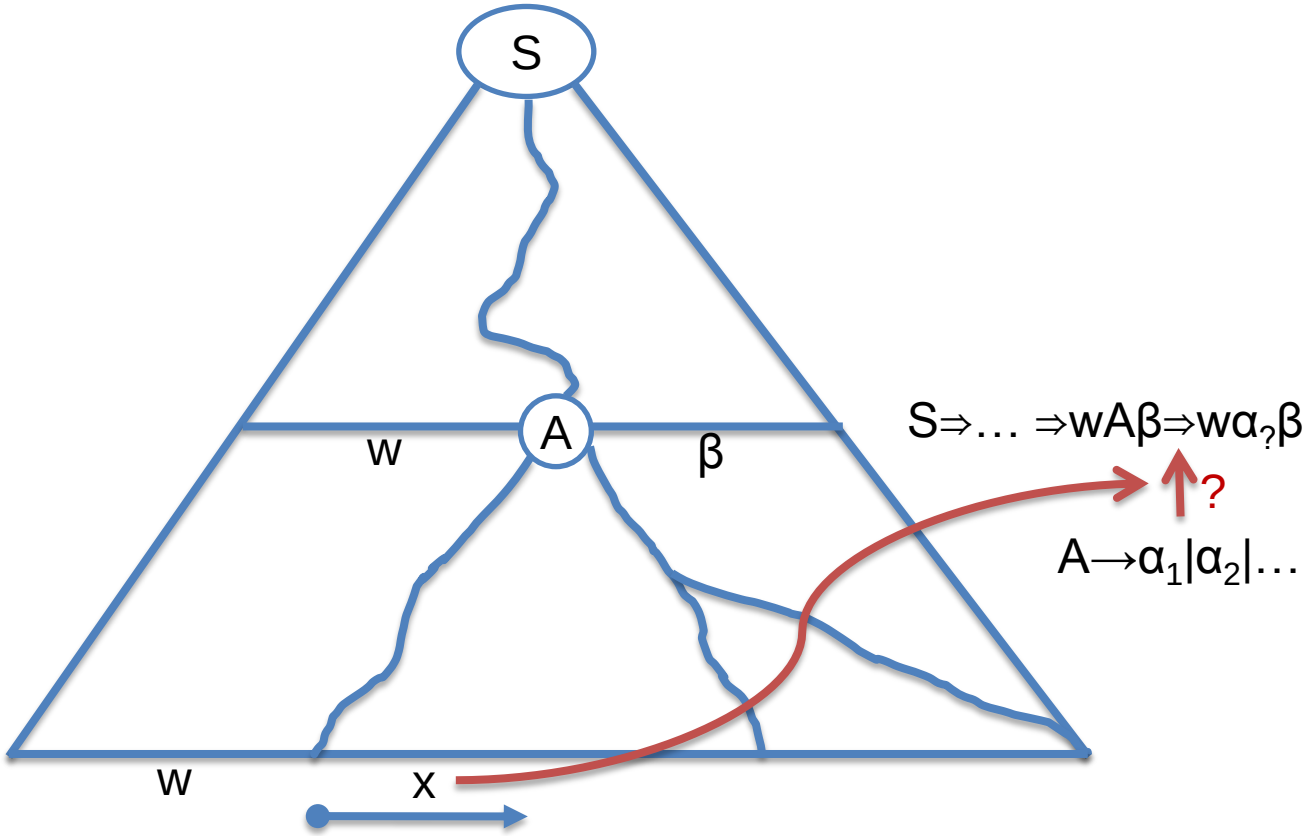
- Elemzés:
  - bemenet:
    - terminálisokból álló karakterlánc
  - kimenet:
    - a bemenet mondat-e
    - ha igen, akkor a levezetési fa megadása
- Elemzők:
  - balelemzés: LL(k)
  - jobbelemzés: LR(k), LALR(k)
  - egyéb: pl. precedenciaelemzés

Szintaktikai elemzés
Balelemzés: LL(k)

Balelemzés: LL(k)

- Bemenet: karakterlánc
- Kiindulás: mondatszimbólum (fentről lefelé)
- Egy lépés:
  - mindig a legbaloldalibb nemterminálist bontjuk fel valamelyik leképzési szabály alapján
  - az előtte lévő terminálisoknak egyezniük kell a bemeneti karaktersorozat elejével:
    - úgy is fel lehet fogni, hogy eddig elolvastuk a bemenetet
  - ha több leképzési szabály van, dönteni kell, melyiket alkalmazzuk:
    - k db terminálist még előre olvasunk a bemenetből, ez alapján döntünk
    - a döntésnek egyértelműnek kell lennie
  - közben feljegyezzük a felhasznált levezetési szabályok sorszámait, ebből a fa rekonstruálható

Balelemzés LL(k)



Balelemzés LL(k)

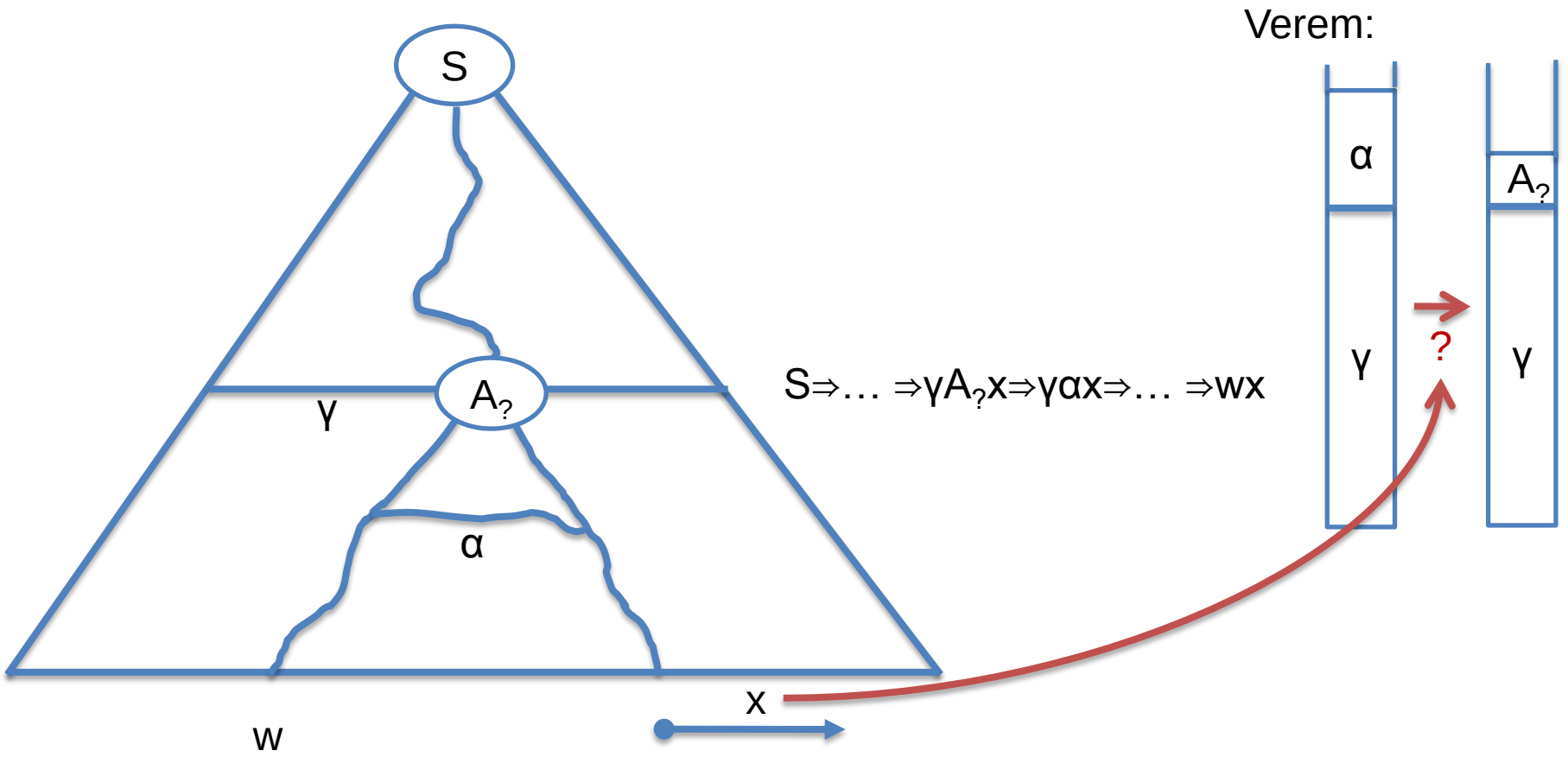
- Gyakorlatban: k=1
  - különben túl sok kombináció
- Balrekurzív nyelvtanok nem balelemezhetők:
  - pl. S→Sa|ε
  - az első szabályt annyiszor kell alkalmazni, ahány ‘a’ van a mondatban
  - a mondat tetszőleges hosszú lehet, azonban csak k db karaktert látunk előre
  - DE: a balrekurzió megszüntethető (azonban a nyelvtan általában bonyolultabb lesz és elrontja a szemantikát):
    - pl. A→Aα|β helyett: A→βA’ és A→αA’|ε
- Nagyobb k esetén bizonyos mértékig átalakítható a nyelvtan, hogy a k csökkenjen
- Vannak azonban olyan nyelvek, amelyeknél ez nem működik:
  - pl. $a^i(b|b^kc)^i$
  - itt a k nem csökkenthető

Szintaktikai elemzés
Jobbelemzés: LR(k)

Jobbelemzés: LR(k)

- Bemenet: karakterlánc
- Kiindulás: karakterlánc (lentről felfelé)
- Lépések (shift-reduce algoritmus):
  - a karakterlánc szimbólumait egyenként elkezdjük egy verembe pakolni (shift)
  - minden lépésben megvizsgáljuk a verem tetejét, hogy ott kialakult-e valamilyen szabály jobb oldala
  - amennyiben igen, lecseréljük (reduce) a verem tetejét a szabály bal oldalán álló nemterminálisra
  - minden lépésben döntés:
    - shift vagy reduce műveletet alkalmazzunk-e
    - reduce esetén melyik szabály alapján
    - ehhez k db terminálist még előre olvasunk a bemenetből, ez alapján döntünk

Jobbelemzés: LR(k)



Jobbelemzés: LR(k)

- Minden determinisztikus CF nyelvtan jobbelemezhető
- Mi a probléma a következő nyelvtannal?
  - Statement ::= IfStatement | Expression
  - Expression ::= IDENTIFIER
  - IfStatement ::=
    IF Expression THEN Statement |
    IF Expression THEN Statement ELSE Statement

Hogyan elemzi a következő kódot?

```
IF Cond1 THEN
   IF Cond2 THEN DoSomething1
   ELSE DoSomething2
```

Hogyan elemzi a következő kódot?

```
IF Cond1 THEN
   IF Cond2 THEN DoSomething1
ELSE DoSomething2
```

shift-reduce conflict!

Jobbelemzés: LR(k)

- Mi a megoldás?
- Át kell alakítani a nyelvtant:
  - Statement ::= IfStatement | Expression
  - Expression ::= IDENTIFIER
  - IfStatement ::=
    IF Expression THEN Statement |
    IF Expression THEN IfThenElseStatement
    ELSE Statement
  - IfThenElseStatement ::=
    IF Expression THEN IfThenElseStatement
    ELSE IfThenElseStatement |
    Expression

Jobbelemzés: LR(k)

- Előny:
  - nagyon erős elemző
  - minden determinisztikus CF nyelvtan elemezhető
  - (az LL(k) alkalmazhatósága eléggé korlátozott)
- Hátrány: túlságosan nagy a táblázat még kis k értékre is
- Gyakorlatban:
  - LR(1) elemző használata LR(0) állapotaira korlátozva (különböző előretekintésű, de azonos veremtartalmú állapotok összevonása): LALR(1)
- Gyakorlatban is használható eszköz, pl.:
  - Beaver: LALR(1) elemzőt generál

Tételek

- **1. tétel:** Minden LR(k) k>1 nyelvtanból készíthető LR(1) nyelvtan, amely ugyanazt a nyelvet generálja.
- **2. tétel:** Minden LL(k) nyelvtan egyben LR(k) nyelvtan is.
- **3. tétel:** Van olyan LR(k) nyelvtan, amely semmilyen k'-re sem LL(k') elemezhető.
- **4. tétel:** Eldönthető az a probléma, hogy egy LR(k) nyelvtanhoz létezik-e olyan k', hogy a nyelvtan LL(k') elemezhető.
- **5. tétel:** Eldönthetetlen az a probléma, hogy egy nyelvhez létezik-e olyan nyelvtan, amely LL(1) elemezhető.
- **6. tétel:** Eldönthetetlen az a probléma, hogy egy nyelvhez létezik-e olyan nyelvtan, amely LL(k) vagy LR(k) elemezhető.

Szintaktikai elemzés Hibakezelés

Hibakezelés

- Cél:
  - az elemzésnek helyes eredményt kell szolgáltatnia
  - vagy hiba esetén minél több hibát kell megtalálnia
  - a hibából eredő következmények nem mindig javíthatók
- Reakció:
  - hibaüzenet: mindig szükséges
  - javítás: amennyiben lehetséges
  - visszaállás: belső állapot konzisztensre állítása és újabb hibák keresése
  - leállás: túl sok hiba, kevés erőforrás (pl. túl nagy táblázat)

Hibakezelés

- Hiba:
  - nem a programozó által kívánt kód
- Hibatünet:
  - a hiba látható hatása: a nyelv szabályainak megsértése
  - az elemző által jelzett hibahely
- Válasz:
  - megpróbálni kideríteni a hiba okát
  - hibajelzés küldése, javítás vagy visszaállás

Példa

- Értékadás:
  - x := (a+b*c;
- Hiba (valószínűleg): ‘)’ hiányzik a ‘b’ után
- Hibajelenség: ‘)’ hiányzik a ‘;’ előtt
- A hibajelenség pozíciója nem feltétlenül egyezik meg a hiba helyével!

Hibák osztályozása

- Anomáliák:
  - megjegyzés: nem megfelelő programstílus
  - figyelmeztetés: lehetséges hiba (pl. nem használt változó)
- Hibák:
  - egyszerű hiba: javítható, kód generálható
  - fatális hiba: nem generálható kód
  - leállási hiba: a fordító feladja (pl. erőforráshiány)

Hibajelzés

- Jelezni kell:
  - pozíció (fájl neve, sor, oszlop)
  - hiba osztálya
  - hibaüzenet
- Belsőleg: a hibaüzenet számmal kódolva
- A hibák általában nem abban a sorrendben jönnek elő, mint ahogyan a forráskódban megjelennek:
  - össze kell őket gyűjteni és rendezni a pozíció alapján

Fordítás fázisaiban előforduló hibák

- Szimbólumhiba:
  - hibás karakter, karakterlánc vége vagy komment vége hiányzik, túl korai fájlvége
- Szintaktikai hiba:
  - a kód nem felel meg a megadott nyelvtannak
- Szemantikai hiba:
  - hiba a statikus szemantikában (pl. típushiba)
- Szemantikai hiba, amely először az optimalizálás során észlelhető (pl. kiindexelés)
- Erőforráshiány:
  - bármelyik fázisban jelentkezhet

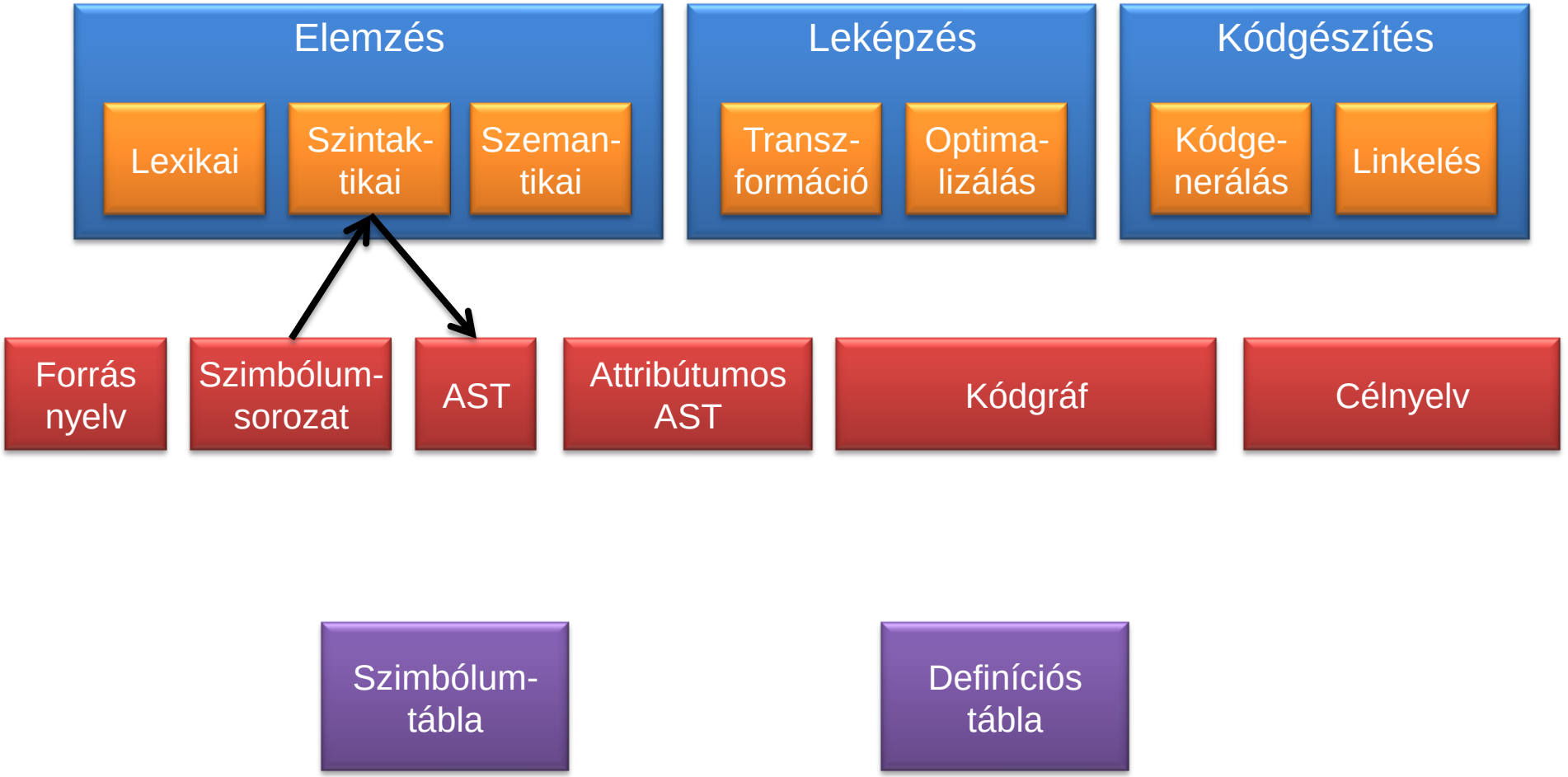
Szintaktikai hiba javítása

- Legkevesebb művelettel:
  - beszúrás
  - törlés
  - csere (törlés és beszúrás)

Szemantikai hiba javítása

- Hiba esetén ‘Unknown’ típus alkalmazása
  - Az ezen végzett művelet típusa is ‘Unknown’
  - Hibajelzés csak ilyen típussal való visszatéréskor
- Fatális hibák forrásai:
  - Nincs definíció: a változó típusa ‘Unknown’ lesz (vagy típuskövetkeztetéssel számítódik)
  - Ismeretlen operáció: a visszatérési érték típusa ‘Unknown’ lesz
  - Nem kompatibilis operandusok vagy egyéb konzisztenciafeltétel sérülése: hibajelzés, az eredmény típusa ‘Unknown’, az elemzés folytatása

Szintaktikai elemzés



XText

XText

- Egyszerű szöveges DSL-ek készítése
- Teljes programnyelvek megvalósítása
- Eclipse támogatás
  - A nyelvtan megírásához
  - A nyelvtan által leírt nyelvhez

Nyelvtant leíró fájl

```
grammar mypackage.MyLanguage
  with org.eclipse.xtext.common.Terminals

generate mymodel "http://www.example.org/mypackage/MyLanguage"
```

- A nyelv egyértelmű neve:
  - mypackage.MyLanguage
- A nyelvtani szabályok őse (tipikusan a következő):
  - org.eclipse.xtext.common.Terminals
- AST generálása EMF modellként:
  - generate kulcsszó
  - utána a package neve: mymodel
  - majd a névtér URI: http://www.example.org/mypackage/MyLanguage
- Ezután jönnek a nyelvtani szabályok

Nyelvtani szabályok

- Felépítés:
  - Nemterminális, kettőspont, jobb oldal, pontosvessző
  - Több jobb oldal esetén: pipe jel az elválasztó
- Jobb oldal:
  - Nemterminális: nagybetűs azonosító
  - Terminális: szöveg aposztrófok között
- Példák:

```
TypeDeclaration :
  InterfaceDeclaration |
  ClassDeclaration;

ClassDeclaration : 'public' 'class' Identifier '{' ClassBody '}';
```

Számosság

- Számosság:
  - ? – nulla vagy egy
  - * – nulla vagy több
  - + – egy vagy több
- Példák:

```
TypeDeclarations:
  ClassDeclarations*;

ClassDeclaration :
   'public' 'class' Identifier ('extends' Identifier)?
   '{'
       ClassBody
   '}';
```

EMF modell property-k

- Értékadás:

```
name=...                setName(…)
items+=...              getItems().add(…)
condition?=...          setCondition(true)
```

- Példa:

```
TypeDeclarations:
  (classes+=ClassDeclarations)*;

ClassDeclaration :
  'public' 'class' name=Identifier
     ('extends' superClassName=Identifier)?
  '{'
      ClassBody
  '}';
```

Hivatkozások

- Ez már a szemantikai ellenőrzés része
- Az ID típusú property-vel rendelkező nemterminálisokra lehet hivatkozni
- Az ID típus beépített
- Jelölés: hivatkozott nemterminális szögletes zárójelben
- Példa:

```
ClassDeclaration :
   'public' 'class' name=ID
      ('extends' superClass=[ClassDeclaration])?
   '{'
       ClassBody
   '}';
```