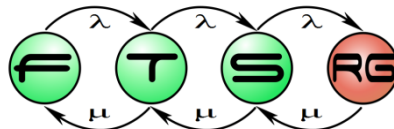


Szöveges domain-specifikus nyelvek



Összegzés

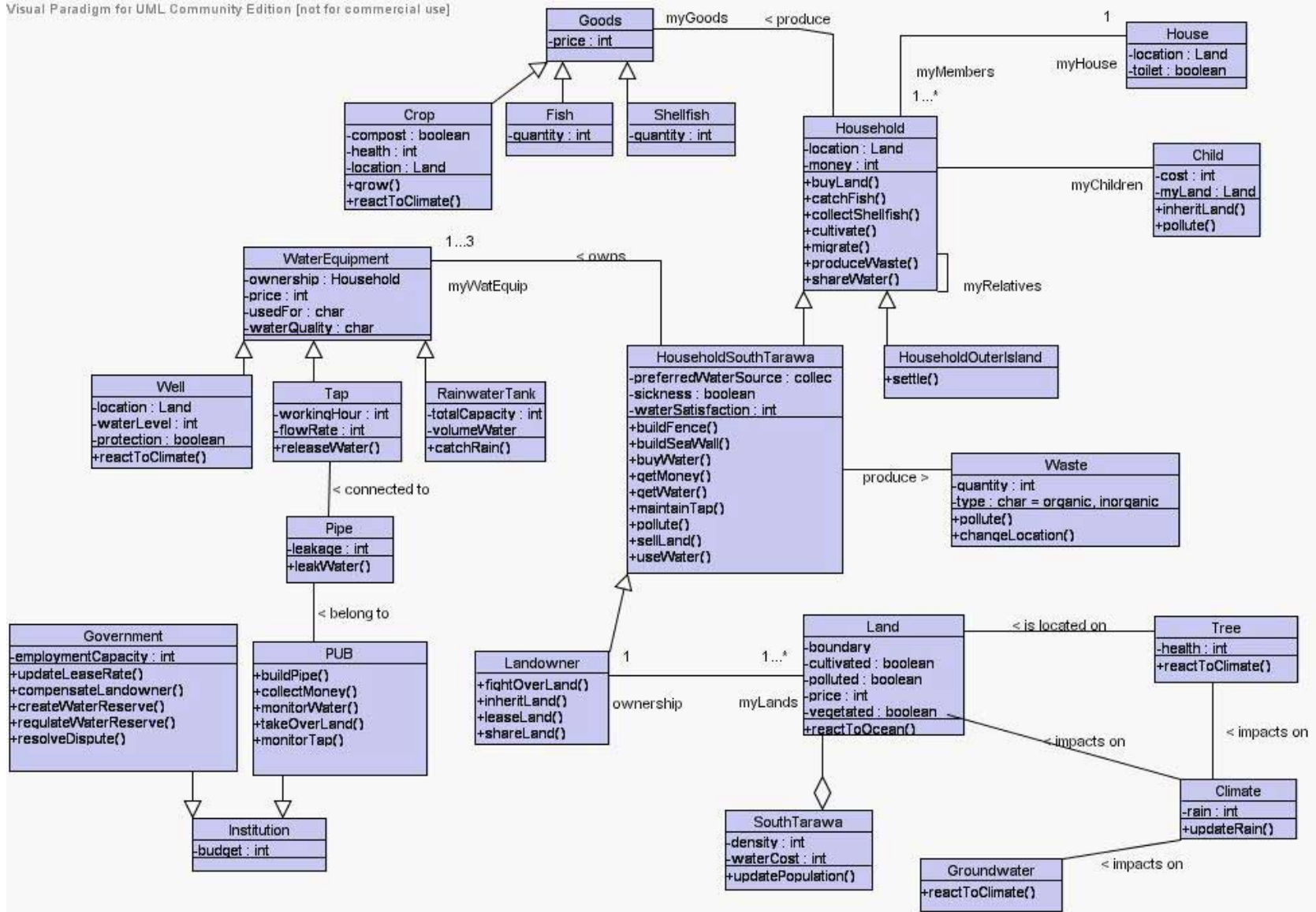
- Eddig
 - Metamodellek
 - Grafikus szerkesztők

Grafikus szerkesztők

- Átlátható, olvasható
- Nehézkes szerkesztés
- Nagyobb (20+ elem) modellek kezelése

Nagyobb modell

Visual Paradigm for UML Community Edition [not for commercial use]



Forrás: <http://epress.anu.edu.au/cs/html/ch12s04.html>

Grafikus szerkesztők

- Átlátható, olvasható
- Nehézkes szerkesztés
- Nagyobb (20+ elem) modellek kezelése

Helyettük: Szöveges nyelvek

Szerkesztők szöveges nyelvekhez

- Első körben
 - Ablak, amibe szöveget írunk
 - +támogató funkciók

Támogatás szintjei

- Egyszerű szövegszerkesztő
 - Példa: Jegyzetömb
 - Alapműveletek (vágólapkezelés, keresés/csere, stb.)
 - Implementálva
- Programozói szövegszerkesztő
 - Példa: Notepad++, Emacs, Vim, ...
 - Nyelvspecifikus szolgáltatások (forráskód színezés, ...)
- Integrált szövegszerkesztő
 - Példa: Eclipse JDT, Visual Studio
 - Összetett szolgáltatások (projekt támogatás, autocomplete, dokumentáció, ...)

JDT szolgáltatások

Szerkesztő
forráskód
színezéssel

Projektstruktúra
kezelése és
megjelenítése

Fájl áttekintő
nézete

Hibák
összegyűjtése

The screenshot displays the Eclipse IDE interface with the following components:

- Project Explorer (Left):** Shows a project structure for 'org.eclipse.viatra2.gtasm.staticcheck.solver'. It includes folders like 'src', 'tests', 'variables', and 'META-INF', along with source files such as 'ConstraintSolver.java', 'ConstraintSolverState.java', 'CSPSolverEngine.java', and 'IntegerSet.java'.
- Editor (Center):** Displays the source code of 'ConstraintSolver.java'. The code is color-coded, and a callout box points to the 'Fájl áttekintő nézete' (File overview view) feature.
- Task List (Right):** Shows a list of tasks or methods, including 'variableNumber', 'vr', 'buildRelationConstraint', and 'buildTypeConstraint'. A callout box points to the 'Fájl áttekintő nézete' (File overview view) feature.
- Error Log (Bottom):** Shows a warning message: 'Javadoc: Missing comment for public declaration CSPEnabledSets.java'. A callout box points to the 'Hibák összegyűjtése' (Error collection) feature.

JDT szolgáltatások: Hibajelzés

```
340     }
341
342     private Constraint buildTypeConstraint(TypeListConstraint constraintSource,
343     IASMTypedValue value = constraintSource.getValue();
344     CSPIntegerVariable variable = getIntegerVariable(value);
345     Collection<ASMTType> types = constraintSource.getTypes();
346     ArrayList<IntegerSet> modelElements = new ArrayList<IntegerSet>();
347     ArrayList<Integer> nativeTypes = new ArrayList<Integer>();
348     int modelElementFound = 0;
349     for (ASMTType type : types) {
350         if (ty_pe instanceof ModelElementType) {
351
352
353
354
355     } else
356
357     }
358
359     if (modelElements.add(modelElementCode);
360     Constraint = null;
361     if (constraintSource.getTypes().contains(type)) {
362         intConstraint = new IntConstantDomainConstraint(solver, variable,
363         nativeTypes.toArray(new Integer[nativeTypes.size()]));
364         if (modelElementFound > 1) {
365             setConstraint = new OrConstraint(solver);
366             CSPSetVariable set = getSetVariable(value);
367             setConstraint.add(set);
368         }
369     }
370     return setConstraint;
371 }
```

ty_pe cannot be resolved to a variable

5 quick fixes available:

- Create local variable 'ty_pe'
- Create field 'ty_pe'
- Create parameter 'ty_pe'
- Create constant 'ty_pe'
- Change to 'type'

JDT szolgáltatások: Dokumentáció

The screenshot displays an IDE window titled `*ConstraintSolver.java` with the following code snippet:

```
340 }
341
342 private Constraint buildTypeConstraint(TypeListConstraint constraintSource,
343     IASMTypedValue value = constraintSource.getValue();
344     CSPIntegerVariable variable = getIntegerVariable(value);
345     Collection<ASMTType> types = constraintSource.getTypes();
346     ArrayList<IntegerSet> modelElements = new ArrayList<IntegerSet>();
347     ArrayList<Integer> nativeTypes = new ArrayList<Integer>();
348     int modelElementFound = 0;
349     for (ASMTType type : types) {
350         if (type instanceof ModelElementType) {
351             modelElementFound++;
352             List<Integer> modelCode = tr
353                 .extractModelElementCode((ModelElementType) type);
354             modelElementFound++;
355         } else {
356             nativeTypes.add(type);
357         }
358     }
359     if (modelElementFound > 0) {
360         Constraint intConstr = new OrConstraint(solver);
361         if (constraintSource instanceof TypeListConstraint) {
362             intConstraint = new OrConstraint(solver);
363             nativeTypes.add(type);
364         }
365         if (modelElementFound > 0) {
366             setConstraint = new OrConstraint(solver);
367             CSPSetVariable set = getSetVariable(value);
```

A tooltip is shown over the `extractModelElementCode` call on line 353. The tooltip content is:

- Method:** `org.eclipse.viatra2.gtasm.staticcheck.variables.TypeRepository.extractModelElementCode(ModelElement type)`
- Description:** Extract a model element code set from the model element type
- Parameters:** `type` the model element type
- Returns:** the extracted set

The right-hand side of the IDE shows the Outline and Task List panels. The Outline panel lists various methods and classes, including `variableNumber : int`, `vr : VariableRepository`, and several `buildRelationConstraint` and `buildTypeConstraint` methods. The Task List panel is currently empty.

JDT szolgáltatások: Occurrence marker

```
341
342 private Constraint buildTypeConstraint(TypeListConstraint constraintSource,
343   IASMTypedValue value = constraintSource.getValue();
344   CSPIntegerVariable variable = getIntegerVariable(value);
345   Collection<ASMType> types = constraintSource.getTypes();
346   ArrayList<IntegerSet> modelElements = new ArrayList<IntegerSet>();
347   ArrayList<Integer> nativeTypes = new ArrayList<Integer>();
348   int modelElementFound = 0;
349   for (ASMType type : types) {
350     if (type instanceof ModelElementType) {
351       modelElementFound++;
352       List<Integer> modelCode = tr
353         .extractModelElementCode((ModelElementType) type);
354       modelElements.add(new IntegerSet(setDomainSize, modelCode));
355     } else {
356       nativeTypes.add(typeMapping.get(type));
357     }
358   }
359   if (modelElementFound > 0) nativeTypes.add(modelElementCode);
360   Constraint intConstraint = null, setConstraint = null;
361   if (constraintSource.isPositive()) {
362     intConstraint = new IntConstantDomainConstraint(solver, variable,
363       nativeTypes.toArray(new Integer[nativeTypes.size()]));
364     if (modelElementFound > 1) {
365       setConstraint = new OrConstraint(solver);
366       CSPSetVariable set = getSetVariable(value);
```

JDT szolgáltatások: Content assist

The screenshot shows the Eclipse IDE with the following components:

- Left Panel (Project Explorer):** Shows a project structure with folders 'tpmvisitors' and 'variables'. Under 'variables', there is a 'types' folder containing several Java files, including 'ModelElementType.java' which is selected.
- Editor:** Displays the source code of 'ConstraintSolver.java'. The code includes a private method 'buildTypeConstraint' with a loop over 'types'. The word 'model' is typed at the end of line 352, triggering a content assist popup.
- Content Assist Popup:** A list of suggestions is shown, including:
 - modelElementFound : int
 - modelElements : ArrayList<org.eclipse.viatra2.gtasm.staticcheck.solver.Int
 - modelElementCode : Integer - ConstraintSolver
 - ModelElementType - org.eclipse.viatra2.gtasm.staticcheck.variables.types** (highlighted)
 - ModelChecker - org.eclipse.viatra2.modelChecker.impl
 - ModelCheckerPropertyProvider - org.eclipse.viatra2.modelChecker
 - ModelCopy - org.eclipse.viatra2.copier
 - ModelCopyException - org.eclipse.viatra2.copier
 - ModelInterpreter - org.eclipse.viatra2.interpreters
 - ModelInterpreterFactory - org.eclipse.viatra2.interpreters
 - ModelMBean - javax.management.modelmbean
 - ModelMBeanAttributeInfo - javax.management.modelmbean
 - ModelMBeanConstructorInfo - javax.management.modelmbean
- Bottom Panel (Help):** A tooltip for 'ModelElementType' is visible, stating: 'A class representing a model element type. A model element type is member of a class hierarchy, and lattice genes are used to represent this hierarchy (latticeelement interface). Author: Zoltan Ujhelyi'.

JDT szolgáltatások

- Struktúra áttekintéséhez nézetek
- Navigációs linkek
- Varázslók
- Refactoring
- Inkrementális builder
- Futtatás/debug támogatás
- Bővíthetőség!

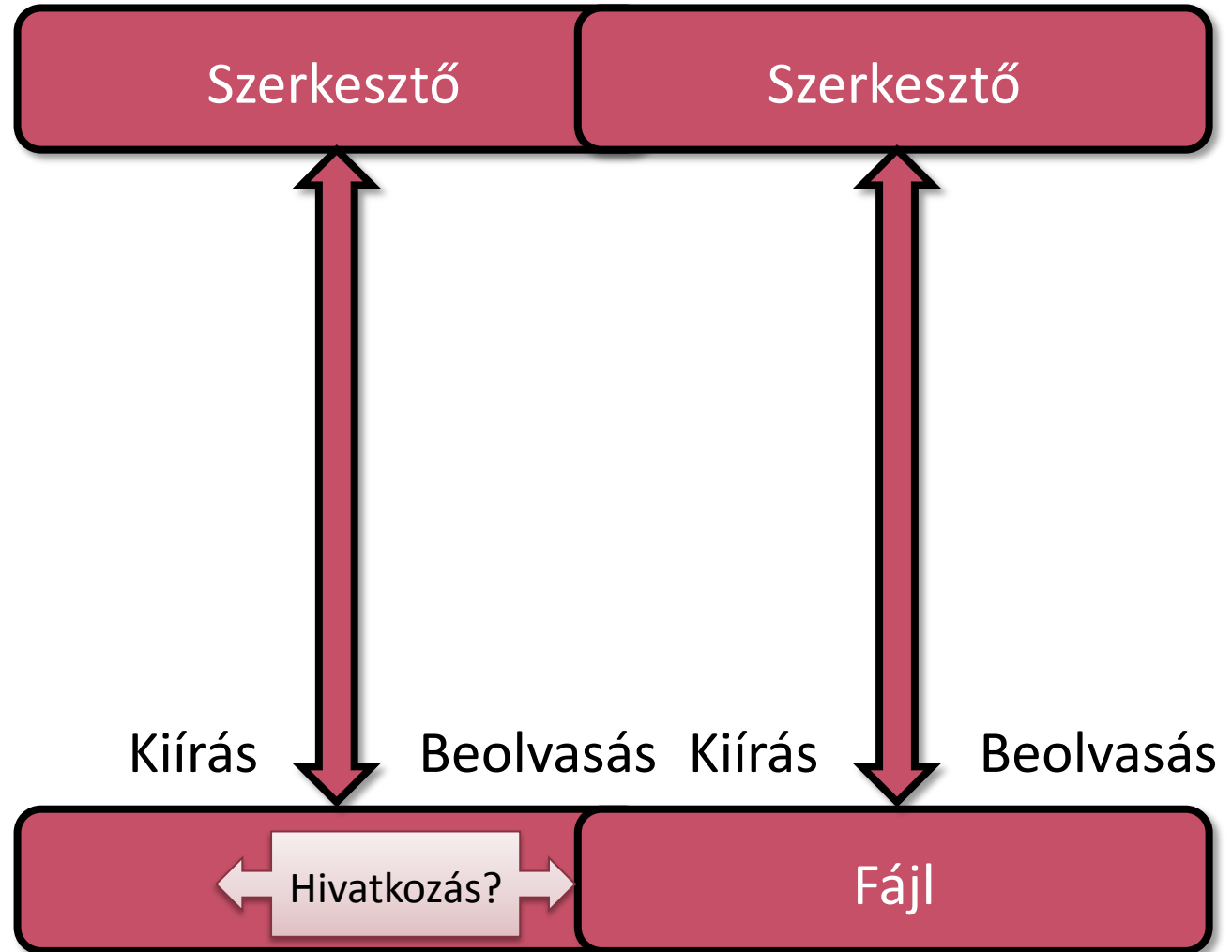
Összegzés

- Integrált szerkesztők szöveges nyelvekhez
 - Sokféle hasznos funkcionális
- Ebben a blokkban
 - Szerkesztők
 - Beolvasás, kiírás
 - Dokumentum modell
 - (Részben) később
 - Builder
 - Projektek

Szöveges editorok Eclipse alatt

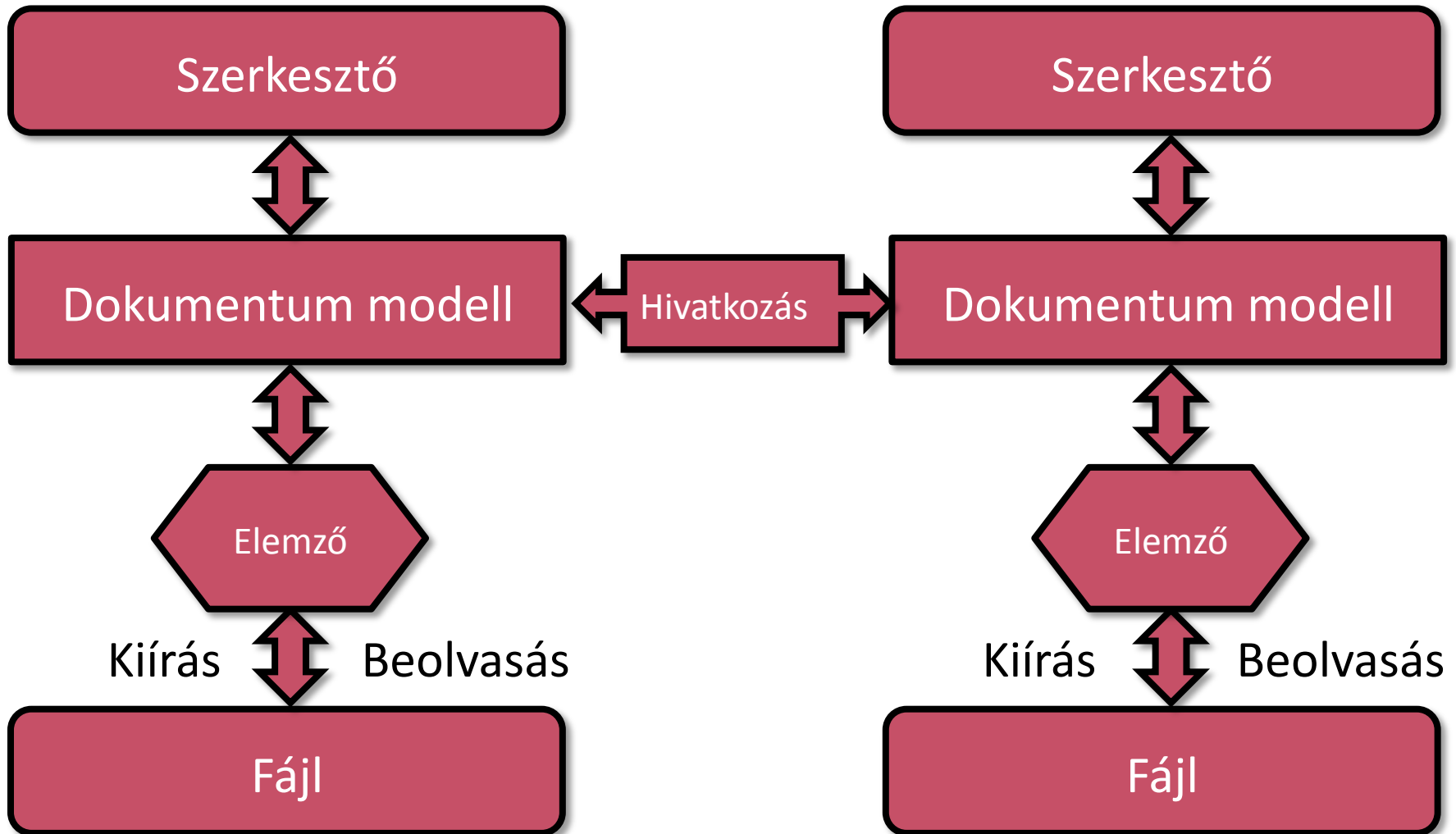
Architektúra

- Egyszerű szövegszerkesztő



Architektúra

- Dokumentum modell (MVC architektúra)



Dokumentum modell

- Objektum reprezentáció a forrásfájlhoz
- Mire használjuk?
 - Hivatkozások követése
 - Fájlon belül
 - Fájlok között
 - Editor szolgáltatások erre épülnek
 - Content assist
 - Outline
 - ...

Dokumentum modell

- Felépítés
 - Szöveg tördelése
 - Absztrakt szintaxis
 - Szoros kapcsolat nyers szöveggel
 - Azonosítható a szövegszakaszok
- Előállítás
 - Elemzőkkel

Dokumentum modell

■ Részletesség

○ Teljes részletesség

- 1:1 megfeleltetés modell és fájl tartalma között
- Megvalósítás
 - AST
 - AST hivatkozáskövetéssel

○ Áttekintő szint

- A modell kevésbé részletes, mint a fájl
- Cél: több fájl tartalmának együttes áttekintése

Példa: JDT dokumentum modell

- Java Object Model

- Osztályok
- Metódusok
- Attribútumok
- Java projektekre

- AST

- Teljes részletesség
- Lusta felépítés

AST
(file1.java)

AST
(file2.java)

Java Object Model

Dokumentum modell előállítása

- Nyelvspecifikus elemző segítségével
 - Beolvasás
 - Részekre bontás
 - Hivatkozás feloldás
 - AST/Dokumentum modell építés
- Hibakezelés!
- Később részletesebben

Jelölők

- Jelölők (markerek)
 - Platform támogatás információk hozzárendelésére
- Fajtái
 - Error marker
 - Bookmark
 - Task marker
 - ...
 - Egyedi jelölők

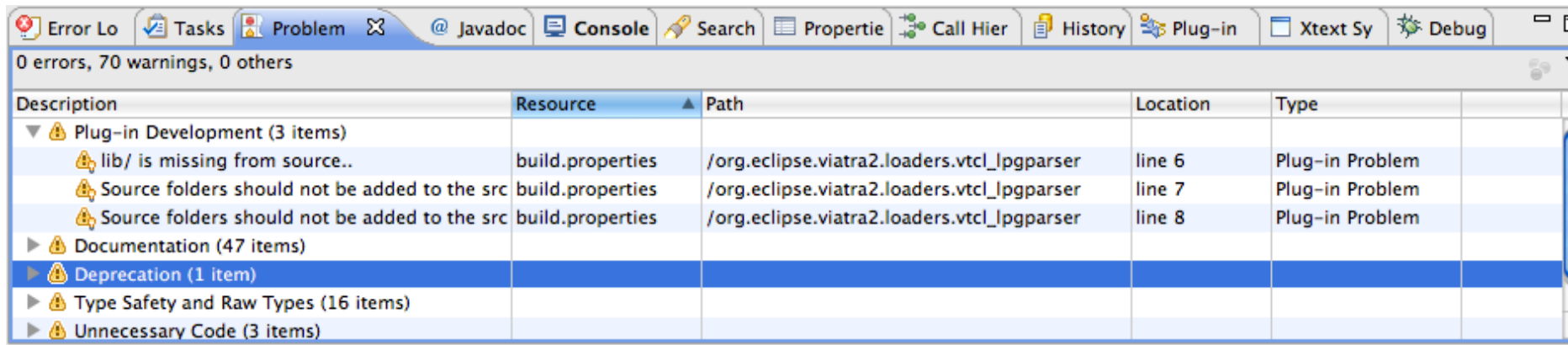
Jelölők

- **Visszajelzés**
 - Elemzők
 - Analízis
 - Felhasználói
- **Szabad adatszerkezet**
 - Kulcs-érték párok
 - Megjelenítés nincs definiálva!
 - Nézetek
 - Problems view
 - Bookmarks
 - Tasks
 - ...

Jelölők

- Fájlokhoz rendelve
 - Ill. workspace erőforrásokhoz
- Tipikus értékek
 - Sorszám/terület

Problem markers



The screenshot shows the Eclipse IDE's Problem view. The toolbar at the top includes icons for Error Log, Tasks, Problem, Javadoc, Console, Search, Properties, Call Hierarchy, History, Plug-in, Xtext System, and Debug. Below the toolbar, a summary bar indicates "0 errors, 70 warnings, 0 others". The main table lists the following items:

Description	Resource	Path	Location	Type
▼ ⚠ Plug-in Development (3 items)				
⚠ lib/ is missing from source..	build.properties	/org.eclipse.viatra2.loaders.vtcl_lpgparser	line 6	Plug-in Problem
⚠ Source folders should not be added to the src	build.properties	/org.eclipse.viatra2.loaders.vtcl_lpgparser	line 7	Plug-in Problem
⚠ Source folders should not be added to the src	build.properties	/org.eclipse.viatra2.loaders.vtcl_lpgparser	line 8	Plug-in Problem
▶ ⚠ Documentation (47 items)				
▶ ⚠ Deprecation (1 item)				
▶ ⚠ Type Safety and Raw Types (16 items)				
▶ ⚠ Unnecessary Code (3 items)				

Dokumentum modell felhasználása

■ Content Assist

- Milyen elemek vannak már definiálva?
- Modellspecifikus szűrés

■ Outline készítés

- Egyszerű szűrése a dokumentum modellnek

■ Hivatkozások követése

- Occurrence marking
- Navigáció

Dokumentum modell felhasználása

- Refactoring
 - Modell szintjén célszerű elvégezni
 - Könnyebb értelmezni a változtatásokat
 - Modell támogatja a visszaírást
- Pretty printing
 - Modell kiírása
 - Tördelés

Szöveges nyelvek szerkesztése

Szöveges nyelvek feldolgozása

- Természetes nyelvek
- XML feldolgozás
- Nyelvtan alapú feldolgozás
 - Kézzel kódolt elemző
 - Reguláris kifejezések
 - Generált elemző

Természetes nyelvek feldolgozása

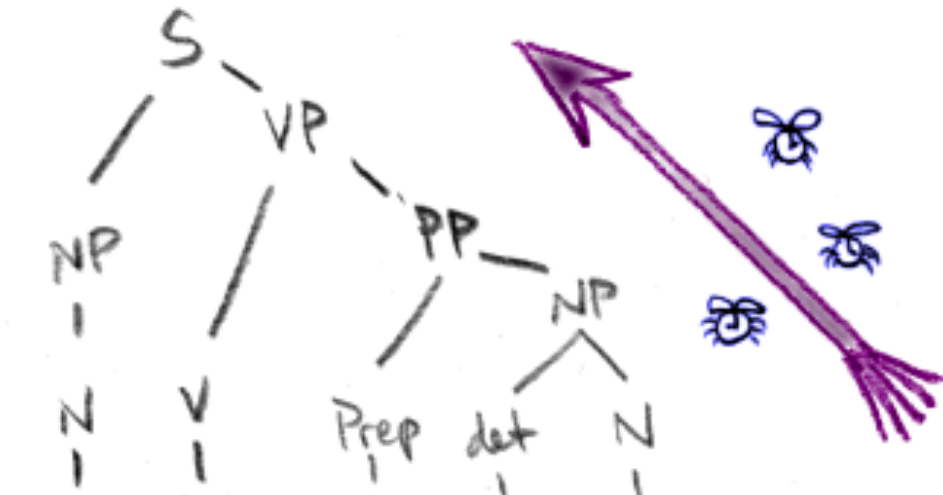
- Általánosan “nehéz”
 - Kontextus-információk
 - Nem (jól) definiált szemantika

Környezet

We gave the monkeys₁ the bananas₂
... because they₁ were hungry.
... because they₂ were ripe.

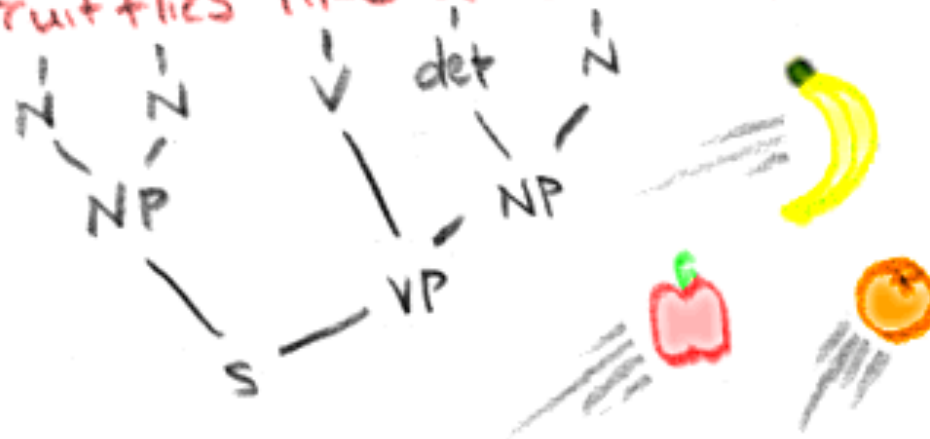


Jelentések



Time flies like an arrow.

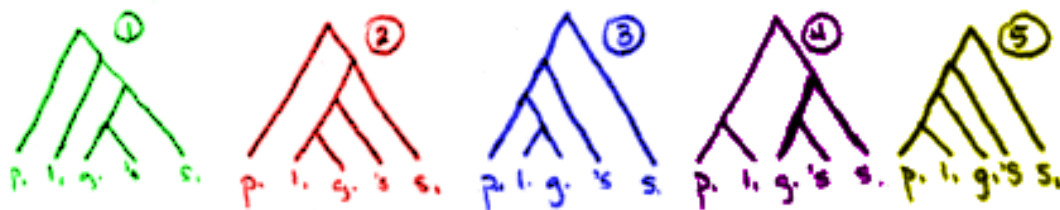
Fruit flies like a banana.



Egyértelműség



pretty little girl's school



XML feldolgozás

- XML: szabványos szöveges szintaxis
- Sémák definiálhatóak
 - Kb. metamodel
- Jó tooling
 - Többféle beolvasási technika
 - Validáció

Problémák az XML használatával

- Nehéz írni
 - Lezáró címkék
 - Entitások (pl. > „)
- Nehéz olvasni
 - Szigorú fastruktúra
 - Hivatkozásfeloldás nem automatikus

Reguláris kifejezések

- Mintaillesztés karakterláncokban
 - Jó támogatás
 - Legtöbb programozási nyelven
 - Nagyjából közös szintaxis
 - Illeszkedés számítás/visszaadás
- Felhasználható szöveg elemzésére?

Some people, when confronted with a problem, think “I know, I’ll use regular expressions.” Now they have two problems.

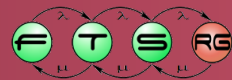
Jamie Zawinski

<http://regex.info/blog/2006-09-15/247>

RegExp: Email validáció

```
(?:(?:[a-zA-Z0-9_\-\.\!*\`\{\}|\(\)|\+]|(?:[^\s@"]|"(?:[^\s"]|"(\s|\\[^\s"])*"?)*)@(?:(?:[a-zA-Z0-9\-\.\!*\`\{\}|\(\)|\+]|(?:[^\s@"]|"(?:[^\s"]|"(\s|\\[^\s"])*"?)*)\.)+(?:[a-zA-Z0-9\-\.\!*\`\{\}|\(\)|\+]|(?:[^\s@"]|"(?:[^\s"]|"(\s|\\[^\s"])*"?)*)$
```

This regular expression will only validate addresses that have had any comments stripped with whitespace



További probléma

- Kimenet egyetlen logikai változó
 - Illeszkedik-e vagy sem
- Mi hiányzik?

Hibajelzés!

Nyelvek és nyelvtanok

Formális nyelvek gyorstalpaló

Formális nyelv

■ Fogalmak

- Nyelv: lehetséges mondatok halmaza
- Mondat: szimbólumok sorozata
- Ábécé: lehetséges szimbólumok halmaza

Példa: Természetes számok

- Ábécé: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Egyjegyű számok nyelve:
 - «0», «1», «2», «3», «4», «5», «6», «7», «8», «9»
- 15-nél kisebb páros számok nyelve:
 - «0», «2», «4», «6», «8», «10», «12», «14»
- Pozitív páros számok nyelve:
 - «0», «2», «4», «6», «8», «10», «12», «14», «16», «18», «20», «22», «24», «26», «28», ...

Nyelvek megadása

- Elemek felsorolása
 - Csak véges elemszám esetén működik
 - Ld. páros számok nyelve
- Véges algoritmus felhasználása
 - Lépések, melynek a kimenetei a nyelv mondatai

Példa: Nevek felsorolása

- Nevek felsorolása:
 - Neveket vesszővel válasszuk el;
 - Kivéve az utolsót, ott és legyen közben
 - Pl.: Zoltán, István és Dániel
 - Ismétlődés megengedett
 - Pl.: Zoltán, Zoltán és Dániel

Példa: Nevek felsorolása

■ Algoritmus

- ① Dániel, István, Zoltán nevek
- ② Egy név érvényes mondat
- ③ Egy mondat, amelyet vessző és egy név követ, értelmes mondat
- ④ Ha a befejezés előtt a mondat “, «név»” alakú, cseréljük ki “ és «név»” formájúra

«név» nem fog szerepelni a szövegben!

Probléma megoldás

■ Kétféle szimbólum

- Megjelenhet a szövegben (ábécéhez tartozik)
 - Terminális szimbólum
- Nem jelenhet meg a szövegben
 - Nem-terminális szimbólum

■ Nem-terminálisok

- Cserélhetőek más szimbólumokkal (→)
- Algoritmus vége
 - Csak terminális szimbólum van
 - Nem cserélhetünk

Módosított algoritmus

- Ábécé
 - Dániel, István, Zoltán, és, “”, sorvége
- Nem-terminális szimbólumok
 - «Név», «Mondat»

① Dániel → «Név»; István → «Név»;
Zoltán → «Név»

② «Mondat» → «Név»

③ «Mondat» → «Mondat», «Név»

④ «Mondat» végén , «Név» → és «Név»

⑤ Kiindulás: «Mondat»

Bonyolult
elemzési szabály

Nyelvek osztályozása

Nyelvek osztályozása

- Elemző bonyolult
 - Nem is lehetséges minden nyelvtanhoz
- Nyelvtanok kategorizálhatóak
 - Chomsky-féle nyelvosztályok

Rekurzívan felsorolható nyelvek (Type 0)

- Korlátozás nélküli szabályok
- Általános elemzés lehetetlen
 - Eldönthetlenségi problémák
 - Speciális esetek működhetnek

Környezetfüggő nyelvek (Type 1)

■ Kétféle definíció

○ Környezetfüggetlen

- $\alpha A \beta \rightarrow \alpha \gamma \beta$
- α és β : kontextus
- A : nem-terminális
- γ : cserélendő érték

Az A nem-terminális kicserélhető γ -ra

○ Monoton

- Szabály jobboldala nem rövidebb, mint a baloldal
- Szabály baloldala tartalmaz nem-terminálist

Környezetfüggő nyelvek (Type 1)

■ Elemzés

- Monoton nyelvtanok
 - Felsorolhatjuk az összes mondatot
 - Adott hosszig
- Lassú, de eldönthető

Környezetfüggetlen nyelvek (Type 2)

- Baloldal

- $A \rightarrow \gamma$

- Egy darab nem-terminális
 - Jobb oldal: terminális és nemterminálisok sorozata

- Környezetfüggő nyelv, de kontextus üres

Környezetfüggetlen nyelvtanok (Type 2)

- Elemzés
 - Többféle megközelítés létezik
 - Jól ismert előnyök/hátrányok
 - Később részletesebben

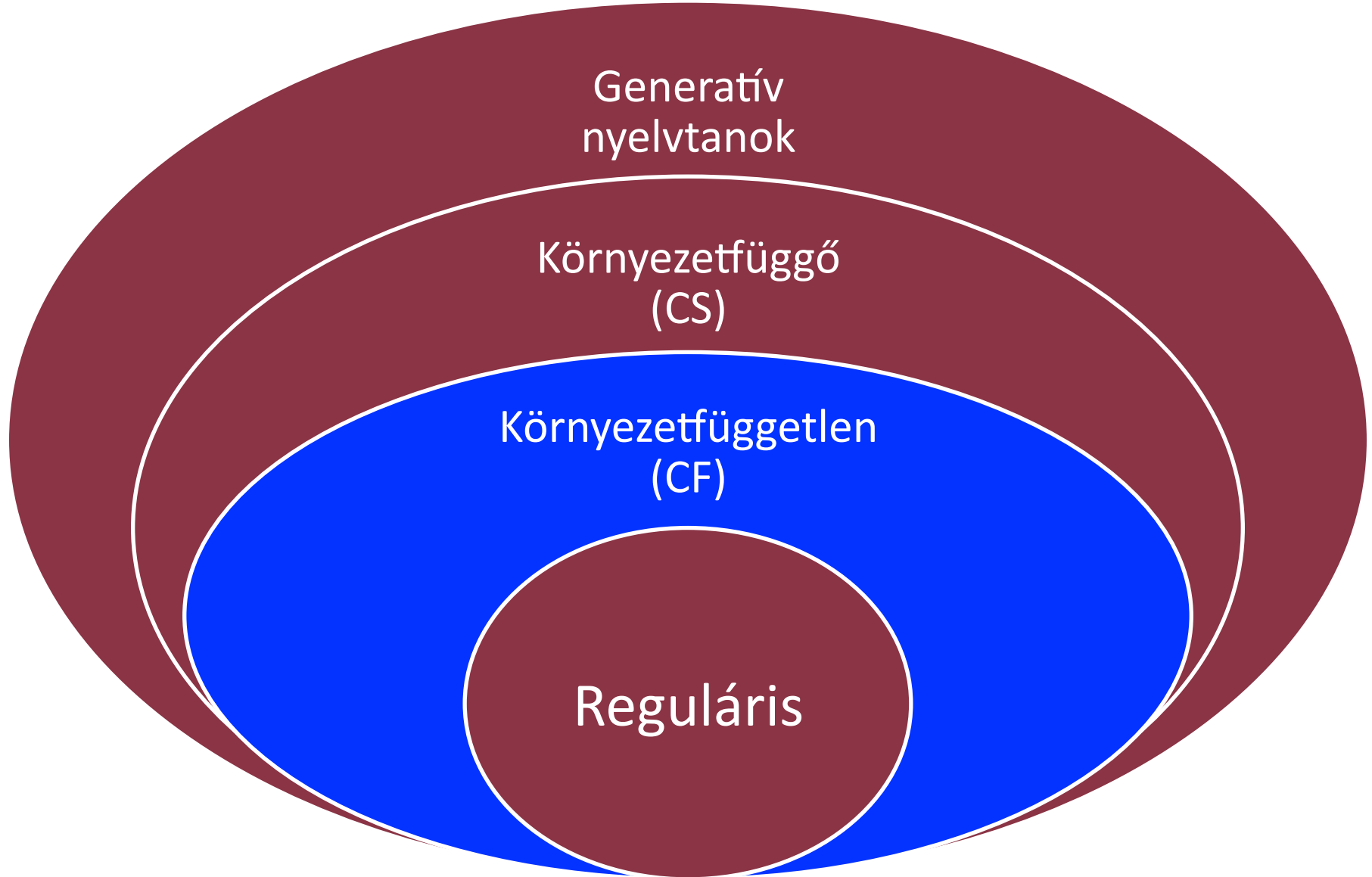
Reguláris nyelvek (Type 3)

- Kétféle szabály:
 - $A \rightarrow \alpha$
 - $A \rightarrow \alpha B$
- Kb. reguláris kifejezéseknek megfelelő
 - Implementációk valamivel többet engednek

Reguláris nyelvek (Type 3)

- Elemző: véges automata
- Egyszerű nyelvekre jó
- DE:
 - Beágyazás (nesting) nem megy
 - Ez tipikus probléma elemzők esetén

Nyelvosztályok hierarchiája



Nyelvek osztályozása

- Eddig:
 - Nyelvtanok osztályozása
- Nyelv:
 - Lehetséges nyelvtanok szerint
 - Legszigorúbb osztálynak megfelelő nyelvtan

Példa: Nevek felsorolása

- Ábécé
 - Dániel, István, Zoltán, és, “”, sorvége
- Nem-terminális szimbólumok
 - «Név», «Mondat»

① Dániel → «Név»; István → «Név»; Zoltán → «Név»

② «Mondat» → «Név»

③ «Mondat» → «Mondat», «Név»

④ «Mondat» végén , «Név» → és «Név»

⑤ Kiindulás: «Mondat»

4. szabály:
0. nyelvosztály

Példa: CF nyelvtan a nevekhez

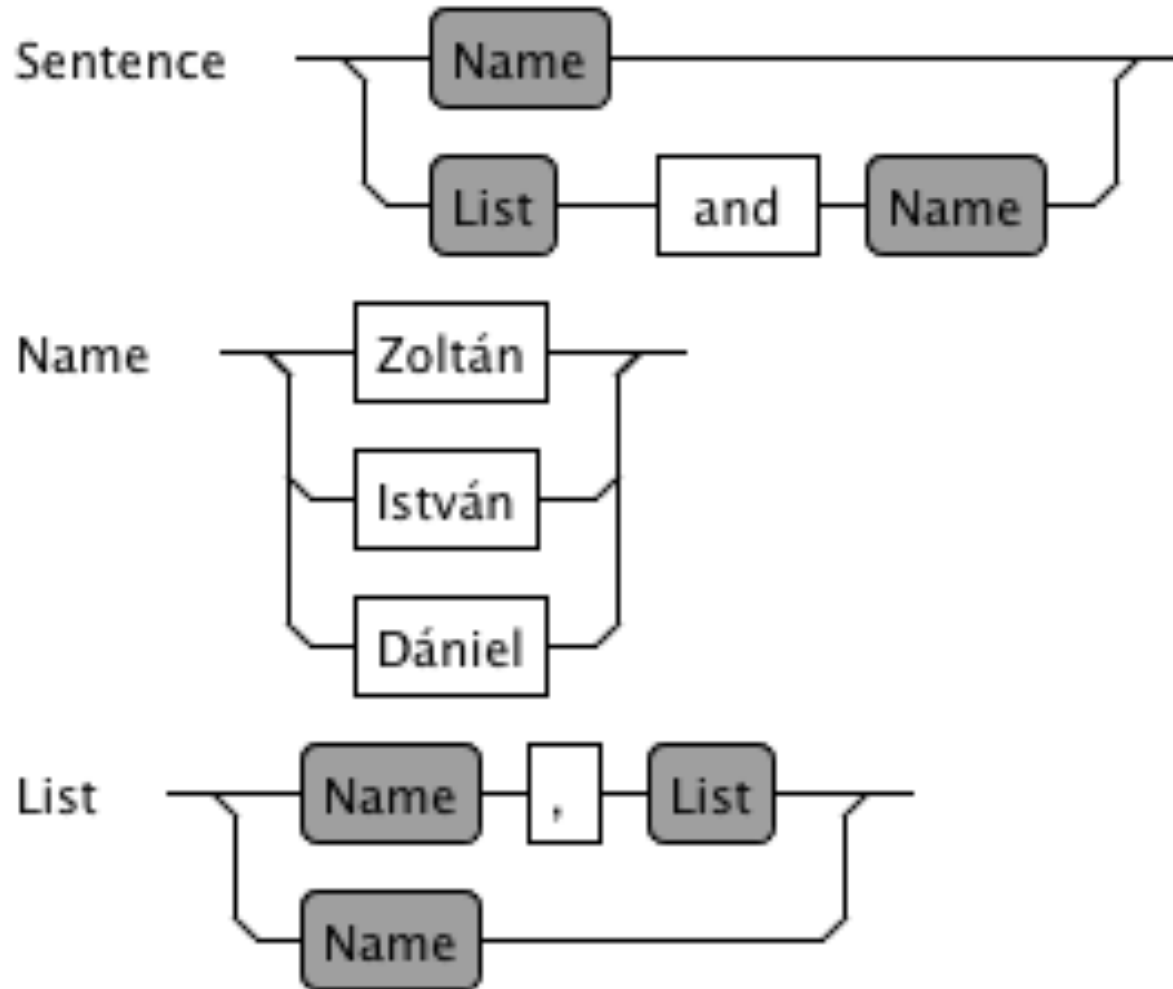
- Ábécé
 - Dániel, István, Zoltán, és, “,” , sorvége
- Nem-terminális szimbólumok
 - «Név», «Mondat», «Lista»

«Név» ::= Zoltán | István | Dániel

«Mondat» ::= «Név» | «Lista» és «Név»

«Lista» ::= «Lista», «Név» | «Név»

Railroad diagram



CF nyelvek elemzése

Nyelvtanok alkalmazása

- Szabályok alapján behelyettesítés
- Indeterminisztikus választás
 - Elemzési algoritmusok leköthetik!
- Cél:
 - Szabályok sorozatával előállítani a bemenetet

Példa: Levezetés

«Név» ::= Zoltán | István | Dániel

«Mondat» ::= «Név» | «Lista» és «Név»

«Lista» ::= «Lista», «Név» | «Név»

Mondatszerű forma	Felhasznált szabály
«Mondat»	

Példa: Levezetés

«Név» ::= Zoltán | István | Dániel

«Mondat» ::= «Név» | «Lista» és «Név»

«Lista» ::= «Lista», «Név» | «Név»

Mondatszerű forma	Felhasznált szabály
«Mondat»	
«Lista» és «Név»	«Mondat» -> «Lista» és «Név»

Példa: Levezetés

«Név» ::= Zoltán | István | Dániel

«Mondat» ::= «Név» | «Lista» és «Név»

«Lista» ::= «Lista», «Név» | «Név»

Mondatszerű forma	Felhasznált szabály
«Mondat»	
«Lista» és «Név»	«Mondat» -> «Lista» és «Név»
«Lista», «Név» és «Név»	«Lista» -> «Lista», «Név»

Példa: Levezetés

«Név» ::= Zoltán | István | Dániel

«Mondat» ::= «Név» | «Lista» és «Név»

«Lista» ::= «Lista», «Név» | «Név»

Mondatszerű forma	Felhasznált szabály
«Mondat»	
«Lista» és «Név»	«Mondat» -> «Lista» és «Név»
«Lista», «Név» és «Név»	«Lista» -> «Lista», «Név»
«Név», «Név» és «Név»	«Lista» -> «Név»

Példa: Levezetés

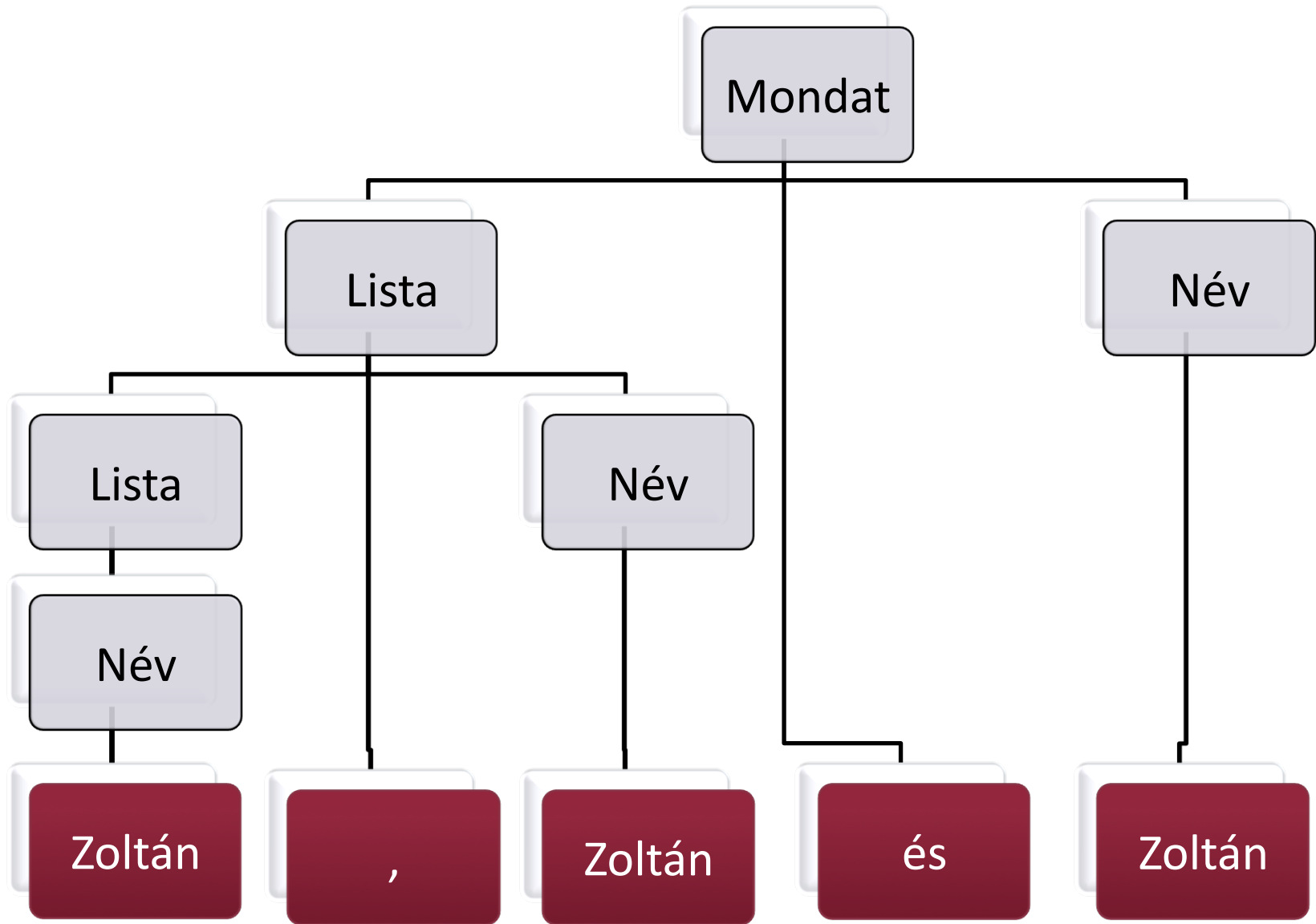
«Név» ::= Zoltán | István | Dániel

«Mondat» ::= «Név» | «Lista» és «Név»

«Lista» ::= «Lista», «Név» | «Név»

Mondatszerű forma	Felhasznált szabály
«Mondat»	
«Lista» és «Név»	«Mondat» -> «Lista» és «Név»
«Lista», «Név» és «Név»	«Lista» -> «Lista», «Név»
«Név», «Név» és «Név»	«Lista» -> «Név»
Zoltán, Zoltán és Zoltán	

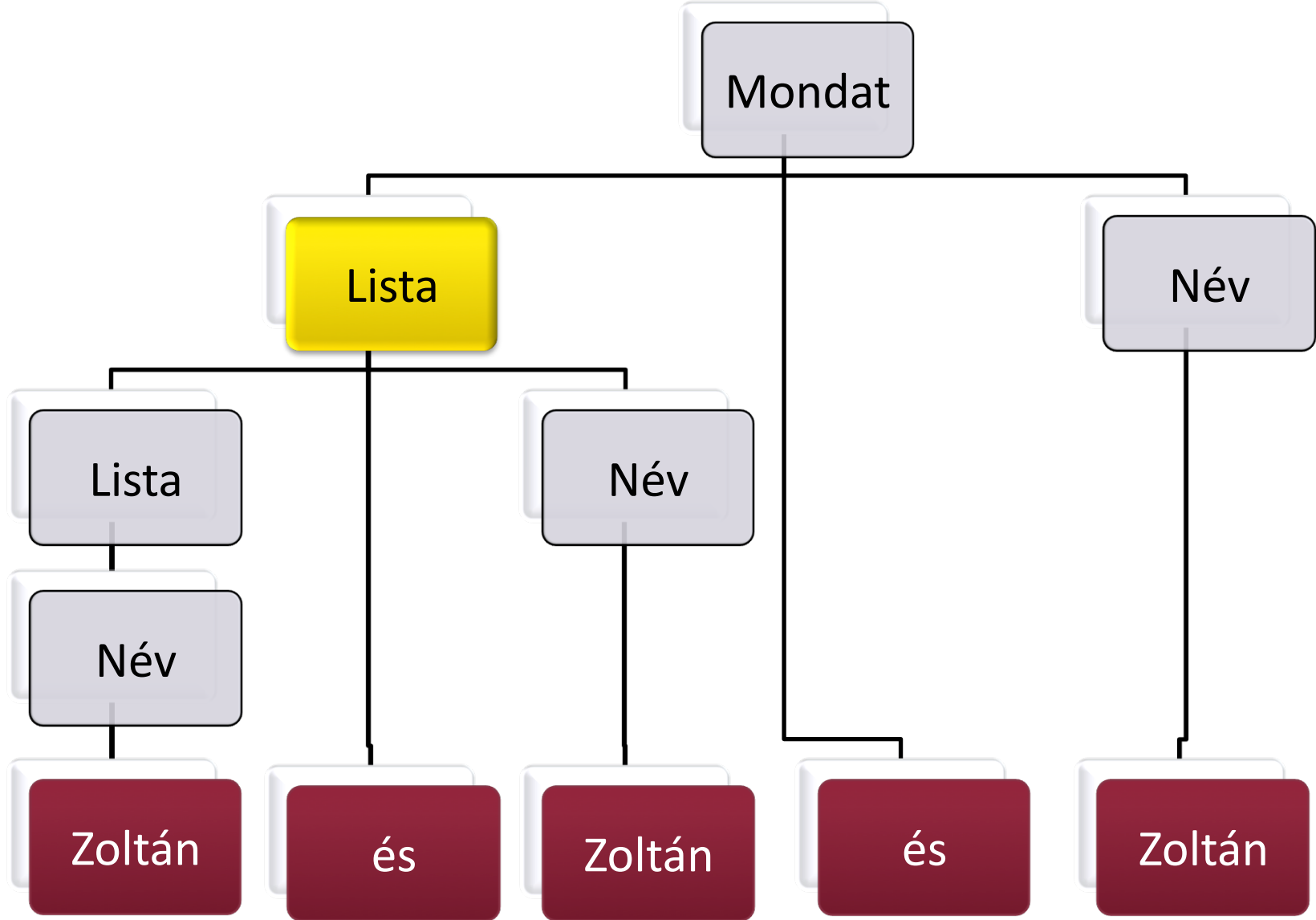
Levezetési fa



Elemzés

- Cél:
 - Levezetési fa meghatározása
 - Miért kell a levezetési fa?

Hibajelzés



Elemzés

- Cél:
 - Levezetési fa meghatározása
 - Miért kell a levezetési fa?
- Megközelítések
 - Nem irányított módszerek (Non-directed methods)
 - Irányított módszerek (Directed methods)
 - Keresési technikák
 - **Felülről lefelé**
 - **Alulról felfelé**

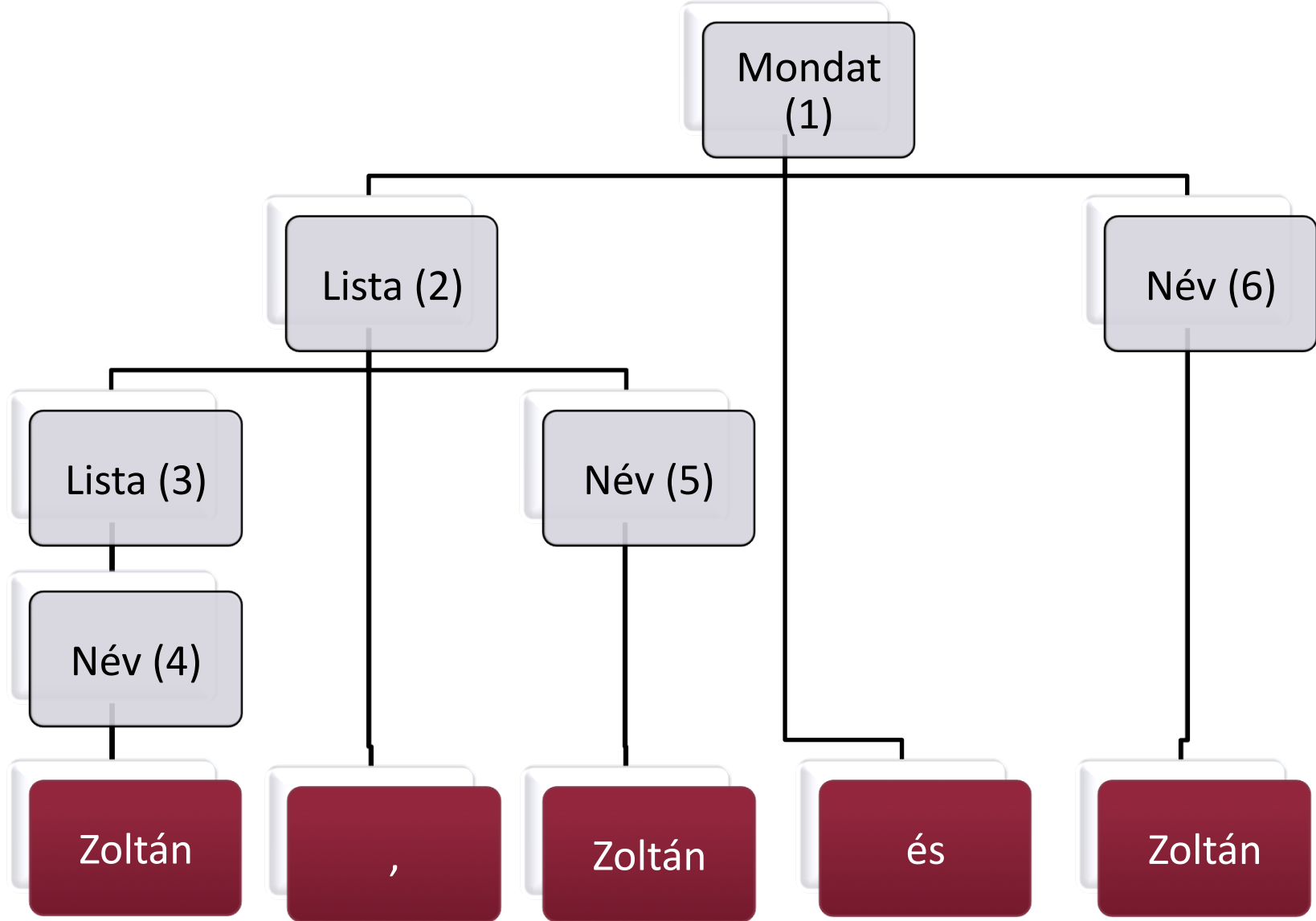
Felülről lefelé elemzés

- Kiindulás: Mondatszimbólum
- Cél: szabályalkalmazásokkal eljutni a mondatig
- Elágazási lehetőségek
 - Predict and match
 - Backtracking szükség esetén

LL(k) elemzők

- Tulajdonságok
 - Left-to-right (balról jobbra olvasás)
 - Leftmost derivation (jobboldali levezetés)
 - k character look-ahead (előrettekintés k karakterrel)

Baloldali levezetés



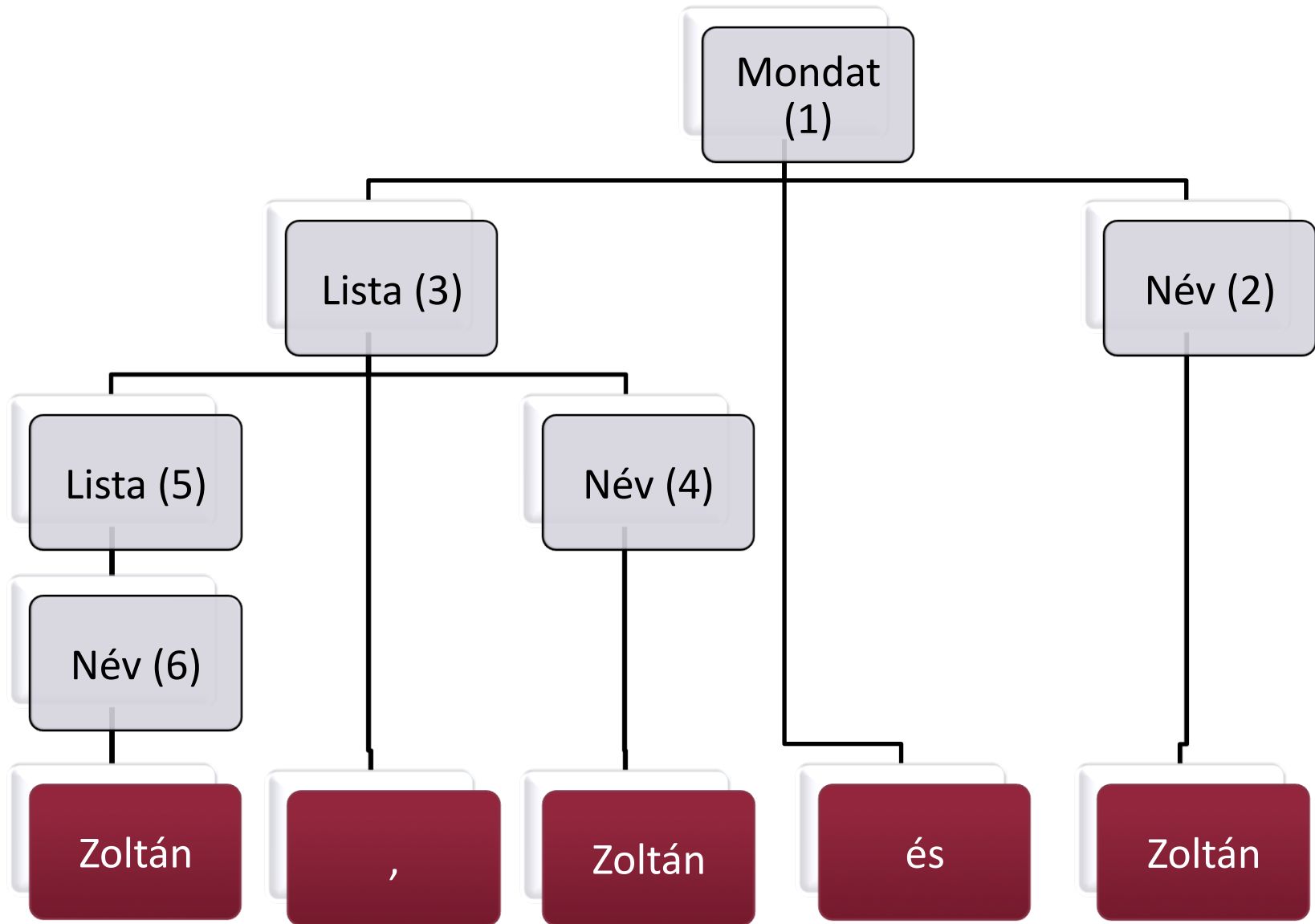
Alulról felfelé elemzés

- Kiindulás: Mondat
- Cél: Eljutni a Mondatszimbólumig
- Szabály jobb oldalát illesztjük a szövegre
 - Illeszkedés esetén alkalmazhatjuk a szabályt
 - Visszafelé!

LR(k) elemzők

- Tulajdonságok
 - Left-to-right
 - Rightmost derivation
 - k character look-ahead

Jobboldali levezetés



Elemzők választása

- $LR(k) \subset LL(k)$
- $LR(k) \subset CF$

- Ötletek
 - $LL(k)$ egyszerűbb algoritmus
 - $LR(k)$ nagyobb kifejezőerő