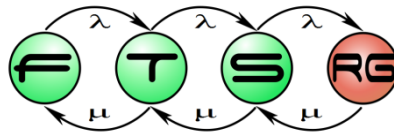


# Szöveges domain-specifikus nyelvek



# Összegzés

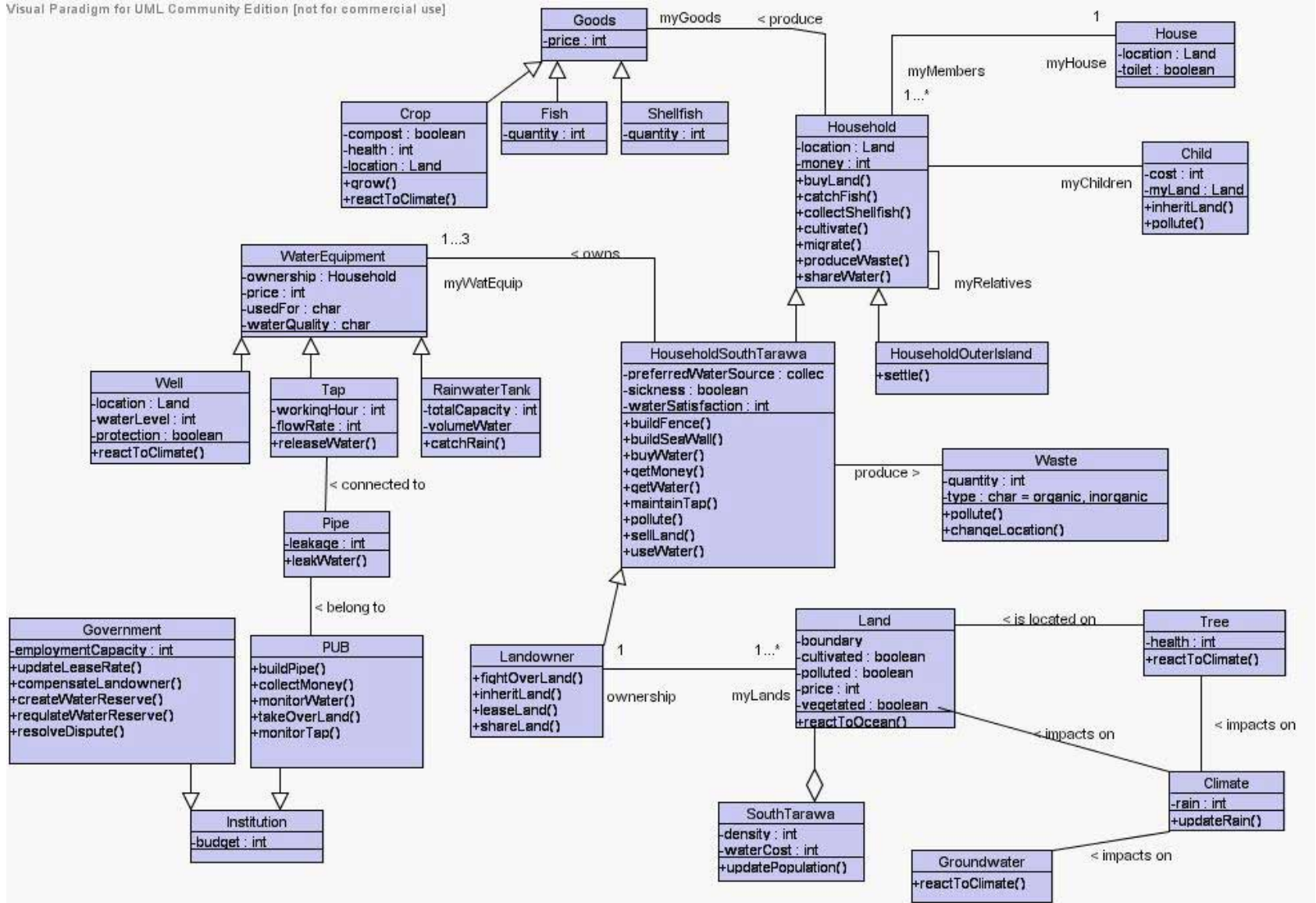
- Eddig
  - Metamodellek
  - Grafikus szerkesztők
  - Kódgenerálás

# Grafikus szerkesztők

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

# Nagyobb modell

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



# 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

# Támogatás szintjei

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



# Támogatás szintjei

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

# Támogatás szintjei

- Egyszerű szövegszerkesztő
  - Példa: Jegyzetömb
  - Alapműveletek (vágólapkezelés, keresés/cseré, stb.)
  - Eclipse környezetben 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

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

- Package Explorer:** Shows the project structure for 'org.eclipse.viatra2.gtasm.staticcheck.solver', including packages like 'constraint', 'tests', and 'variables', and files like 'ConstraintSolver.java'.
- Main Editor:** Displays the source code of 'ConstraintSolver.java'. The code includes a method 'buildTypeConstraint' that iterates over 'ASMTypedValue' objects and builds constraints based on their types. It uses 'IntegerSet' and 'Constraint' classes.
- Outline:** Lists the classes and methods in the current file, such as 'variableNumber', 'VariableRepository', and 'buildRelationConstraint'.
- Task List:** Shows a list of tasks or warnings, including 'Javadoc: Missing comment for public declaration'.
- Error Log:** Displays the warning: 'Javadoc: Missing comment for public declaration' in 'CSPEnabledSets.java' at line 37.

```
340 }
341
342 private Constraint buildTypeConstraint(ASMTypedValue value = constraintSource.getValue();
343 CSPIntegerVariable variable = getIntegerVariable(value);
344 Collection<ASMTypedValue> types = constraintSource.getTypes();
345 ArrayList<IntegerSet> modelElements = new ArrayList<IntegerSet>();
346 ArrayList<Integer> nativeTypes = new ArrayList<Integer>();
347 int modelElementFound = 0;
348 for (ASMTypedValue type : types) {
349     if (type instanceof ModelElementType) {
350         modelElementFound++;
351         List<Integer> modelCode = tr
352             .extractModelElementCode((ModelElementType) type);
353         modelElements.add(new IntegerSet(setDomainSize, modelCode));
354     } else {
355         nativeTypes.add(typeMapping.get(type));
356     }
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

Szerkesztő  
forráskód  
színezéssel

```
private Constraint buildTypeConstraint(TypeListConstraint constraintSource,
IASMTypedValue value = constraintSource.getValue();
CSPIntegerVariable variable = getIntegerVariable(value);
Collection<ASMTypedValue> types = constraintSource.getTypes();
ArrayList<IntegerSet> modelElements = new ArrayList<IntegerSet>();
ArrayList<Integer> nativeTypes = new ArrayList<Integer>();
int modelElementFound = 0;
for (ASMTypedValue type : types) {
    if (type instanceof ModelElementType) {
        modelElementFound++;
        List<Integer> modelCode = tr
            .extractModelElementCode((ModelElementType) type);
        modelElements.add(new IntegerSet(setDomainSize, modelCode));
    } else {
        nativeTypes.add(typeMapping.get(type));
    }
}
if (modelElementFound > 0) nativeTypes.add(modelElementCode);
Constraint intConstraint = null, setConstraint = null;
if (constraintSource.isPositive()) {
    intConstraint = new IntConstantDomainConstraint(solver, variable,
        nativeTypes.toArray(new Integer[nativeTypes.size()]));
}
if (modelElementFound > 1) {
    setConstraint = new OrConstraint(solver);
    CSPSetVariable set = getSetVariable(value);
```

Description	Resource	Path	Location	Type
▼ Documentation (1 item)				
⚠ Javadoc: Missing comment for public declarati	CSPEnabledSets.java	/org.eclipse.viatra2.gtasm.staticcheck.solver/src	line 37	Java Problem

org.eclipse.viatra2.gtasm.staticcheck.solver.ConstraintSolver.buildTyp...ource) : Constraint – org.eclipse.viatra2.gtasm.staticcheck.solver/src 210M pf 445M

# JDT szolgáltatások

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays the project structure for 'org.eclipse.viatra2.gtasm.staticcheck.solver'. The main editor shows the 'ConstraintSolver.java' file with the following code snippet:

```
340 }
341
342 private Constraint buildTypeCons
343 IASMTypedValue value = const
344 CSPIntegerVariable variable
345 Collection<ASMType> types =
346 ArrayList<IntegerSet> modelE
347 ArrayList<Integer> nativeTyp
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);
```

A callout box with the text "Fájl áttekintő nézete" (File overview view) points to the Outline view on the right. The Outline view shows a list of classes and methods, including:

- variableNumber : int
- vr : VariableRepository
- buildRelationConstraint(CSPSe
- buildRelationConstraint(CSPSe
- buildTypeConstraint(AndType
- buildTypeConstraint(Condition
- buildTypeConstraint(OrTypeC
- buildTypeConstraint(RelationF
- buildTypeConstraint(TypeCon
- buildTypeConstraint(TypeEqu
- buildTypeConstraint(TypeList
- fillTypeConstraint(TypeConstr
- getCreatedCSPConstraints() : I
- getCreatedSetNumber() : Integ
- getCreatedHandlerConstraints
- getFailedVariables() : List<IAS
- getHandlerState() : Constraint
- getIntegerVariable(IASMTyped
- getPossibleTypes(IASMTypedV
- getRelationParameter(Relation
- getSetVariable(IASMTypedValu
- getStatus() : CSPStatus
- initializeConstraintHandler(Va
- initializeConstraintHandler(Va
- initializeVariable(CSPIntegerV

The Error Log at the bottom shows a warning: "Javadoc: Missing comment for public declarati" in CSPEnabledSets.java at line 37.

Description	Resource	Path	Location	Type
Documentation (1 item)				
Javadoc: Missing comment for public declarati	CSPEnabledSets.java	/org.eclipse.viatra2.gtasm.staticcheck.solver/src	line 37	Java Problem

org.eclipse.viatra2.gtasm.staticcheck.solver.ConstraintSolver.buildTyp...ource) : Constraint - org.eclipse.viatra2.gtasm.staticcheck.solver/src 210M pf 445M



# JDT szolgáltatások

The screenshot displays the Eclipse IDE interface. On the left, the Package Explorer shows a project structure for 'org.eclipse.viatra2.gtasm.staticcheck.solver'. A red box highlights this tree, and a speech bubble points to it with the text 'Projektstruktúra kezelése és megjelenítése'. The main editor shows the 'ConstraintSolver.java' file with Java code. The right side features the Outline and Task List views. At the bottom, the Error Log shows a 'Javadoc: Missing comment for public declaration' error.

Projektstruktúra kezelése és megjelenítése

```
340 }
341
342 private Constraint buildTypeConstraint(TypeListConstraint constraintSource,
343     IASMTypedValue value = constraintSource.getValue();
344     CSPIntegerVariable variable = getIntegerVariable(value);
345     = constraintSource.getTypes();
346     Elements = new ArrayList<IntegerSet>();
347     Types = new ArrayList<Integer>();
348
349     (ModelElementType type) {
350         ModelElementCode = tr
351         ModelElementCode((ModelElementType) type);
352         new IntegerSet(setDomainSize, modelCode));
353
354         nativeTypes.add(typeMapping.get(type));
355     }
356 }
357
358 if (modelElementFound > 0) nativeTypes.add(modelElementCode);
359
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);
367     }
368 }
```

Description	Resource	Path	Location	Type
▼ Documentation (1 item)				
Javadoc: Missing comment for public declaration	CSPEnabledSets.java	/org.eclipse.viatra2.gtasm.staticcheck.solver/src	line 37	Java Problem

org.eclipse.viatra2.gtasm.staticcheck.solver.ConstraintSolver.buildType...ource) : Constraint - org.eclipse.viatra2.gtasm.staticcheck.solver/src 210M pf 445M

# JDT szolgáltatások

The screenshot displays the Eclipse IDE interface. The main editor shows the `ConstraintSolver.java` file 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<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 {
362     intDomainConstraint(solver, variable,
363     new Integer[nativeTypes.size()]);
364
365     straint(solver);
366     setVariable(value);
367 }
```

A red speech bubble with the text "Hibák összegyűjtése" (Error collection) is overlaid on the code. Below the code, the Error Log window is visible, showing a warning:

Description	Source	Path	Location	Type
Documentation (1 item)				
Javadoc: Missing comment for public declaration	CSPEnabledSets.java	/org.eclipse.viatra2.gtasm.staticcheck.solver/src	line 37	Java Problem

The status bar at the bottom indicates the current file is `ConstraintSolver.java` with 210M of 445M memory used.

# JDT szolgáltatások: Hibajelzés és javítás

```
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             Code((ModelElementType) type);
352             rSet(setDomainSize, modelCode));
353         } else {
354             get(type));
355         }
356     }
357 }
358
359 if (modelElements.add(modelElementCode);
360 Constraint = null;
361 if (constraintSource.getTypes().contains(modelElementCode));
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 }
```



# JDT szolgáltatások: Dokumentáció

The screenshot displays an IDE window with the following components:

- Main Editor:** Shows the `ConstraintSolver.java` file. The code includes a `private Constraint buildTypeConstraint` method. A tooltip is visible over the `extractModelElementCode` call on line 353, providing details about the `List<Integer>` return type and the method's parameters and return value.
- Outline:** Located on the right, it lists various methods and variables, including `variableNumber`, `vr`, and several `buildRelationConstraint` and `buildTypeConstraint` methods.
- Task List:** Also on the right, it shows a list of tasks or actions.

```
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 intConstraint = new OrConstraint(solver);
361     if (constraintSource instanceof CSPIntegerVariable) {
362       intConstraint.add(new CSPIntegerVariable(solver,
363         ((CSPIntegerVariable) constraintSource).getIntegerVariable(),
364         ((CSPIntegerVariable) constraintSource).getIntegerVariable().getIntegerValue());
365     } else {
366       CSPSetVariable set = getSetVariable(value);
367       intConstraint.add(new CSPSetVariable(solver, set));
368     }
369   }
370 }
```

**Tooltip for `extractModelElementCode`:**

- Return Type:** `List<Integer>`
- Method Signature:** `org.eclipse.viatra2.gtasm.staticcheck.variables.TypeRepository.extractModelElementCode(ModelElementType type)`
- Description:** Extract a model element code set from the model element type
- Parameters:** `type` the model element type
- Returns:** the extracted set

# 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 interface. On the left, the Package Explorer displays a project structure with folders for 'tpmvisitors' and 'variables', and a sub-folder 'types' containing several Java files, including 'ModelElementType.java'. The main editor window shows the source code of 'ConstraintSolver.java' with a cursor at line 352, where the word 'model' is being typed. A content assist popup is visible, listing several classes that match the prefix 'model'. The popup includes a description of the 'ModelElementType' class and a list of related classes with their fully qualified names.

```
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     }
}
```

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

- modelElementFound : int
- modelElements : ArrayList<org.eclipse.viatra2.gtasm.staticcheck.solver.Int
- modelElementCode : Integer - ConstraintSolver
- ModelElementType - org.eclipse.viatra2.gtasm.staticcheck.variables.types
- 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

Press 'Tab' from proposal table or click for focus

Press '^Space' to show Template Propos

# 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
  - Előző alkalommal
    - Builder
    - Projektek

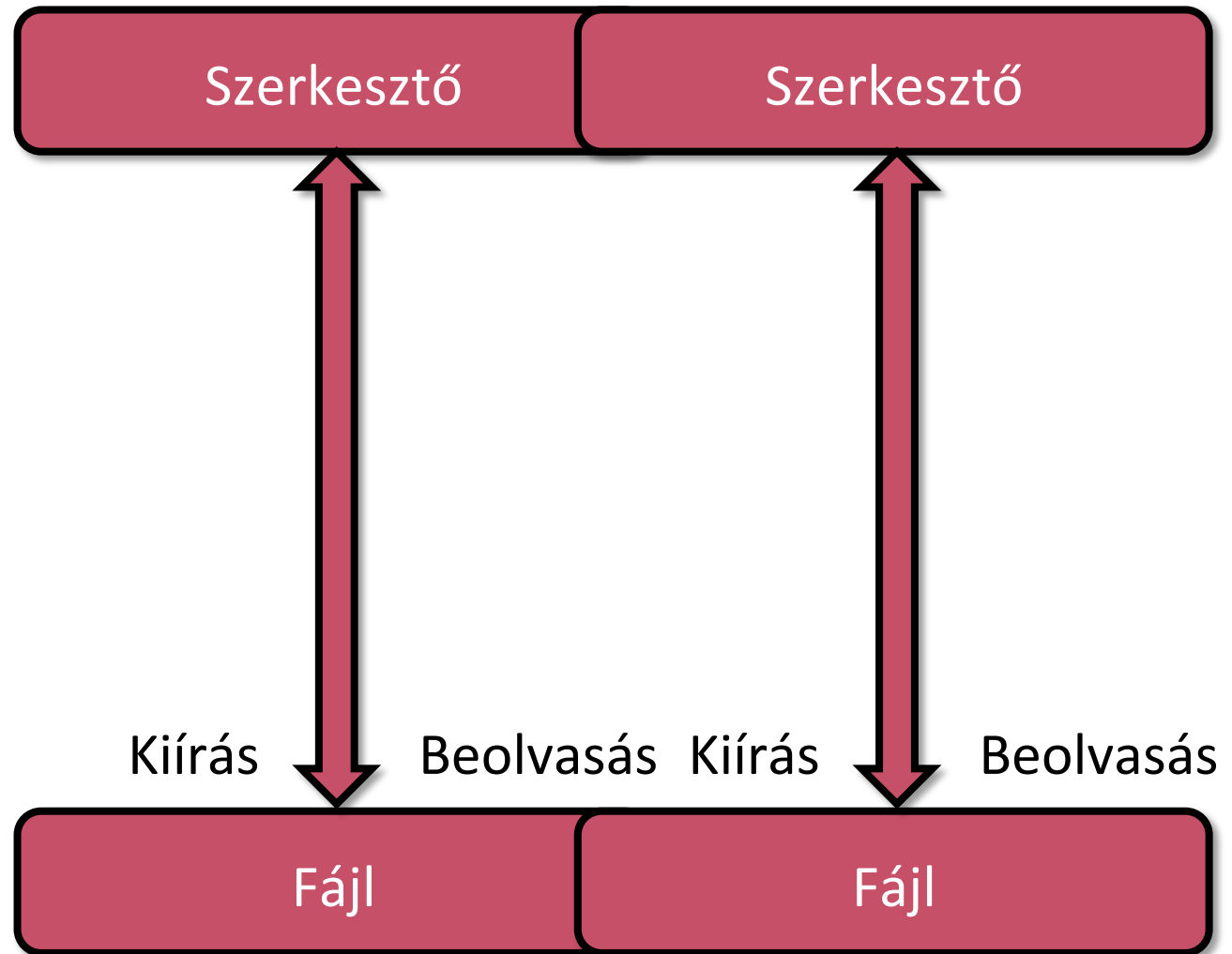
# Szöveges editorok Eclipse alatt

# Architektúra

- Egyszerű szövegszerkesztő

# Architektúra

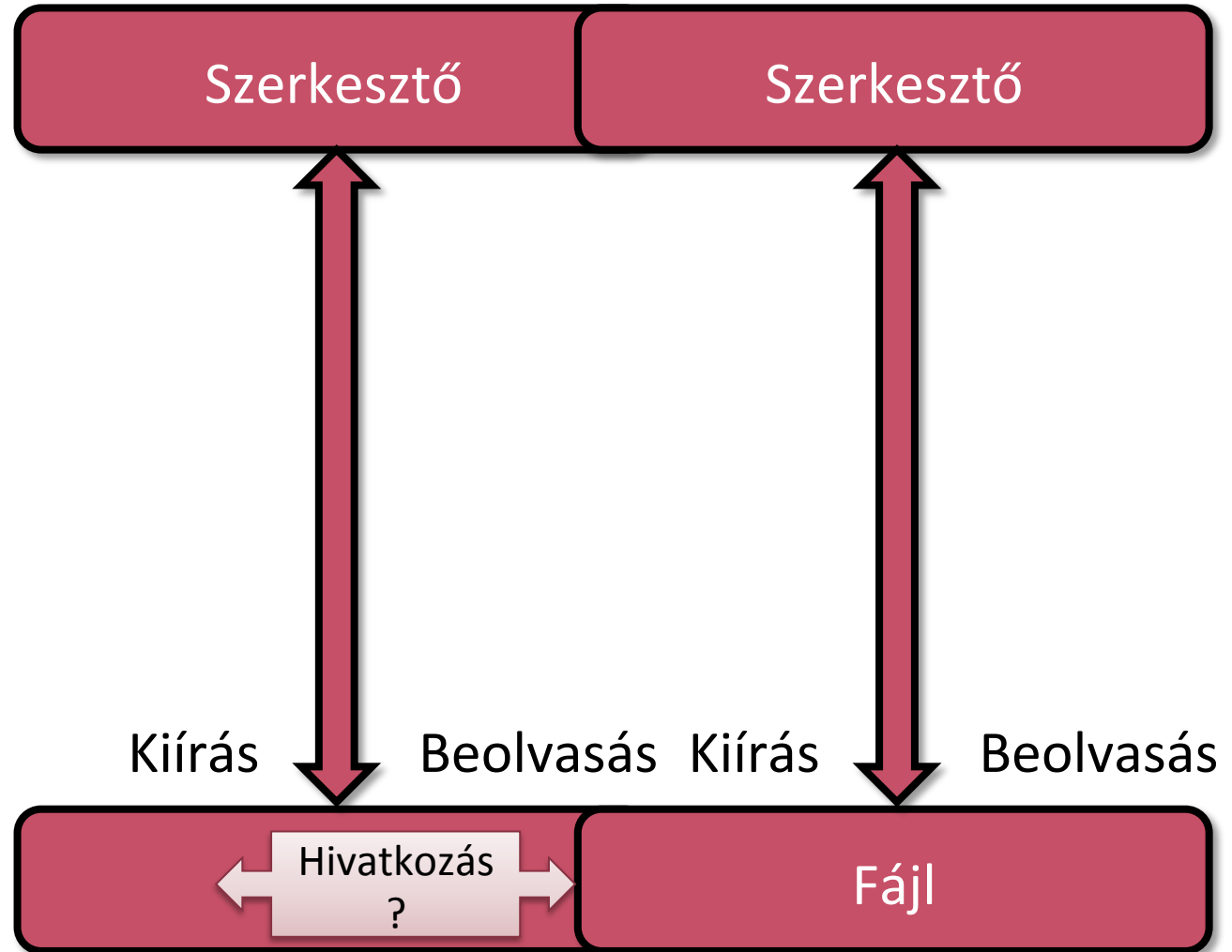
- Egyszerű szövegszerkesztő





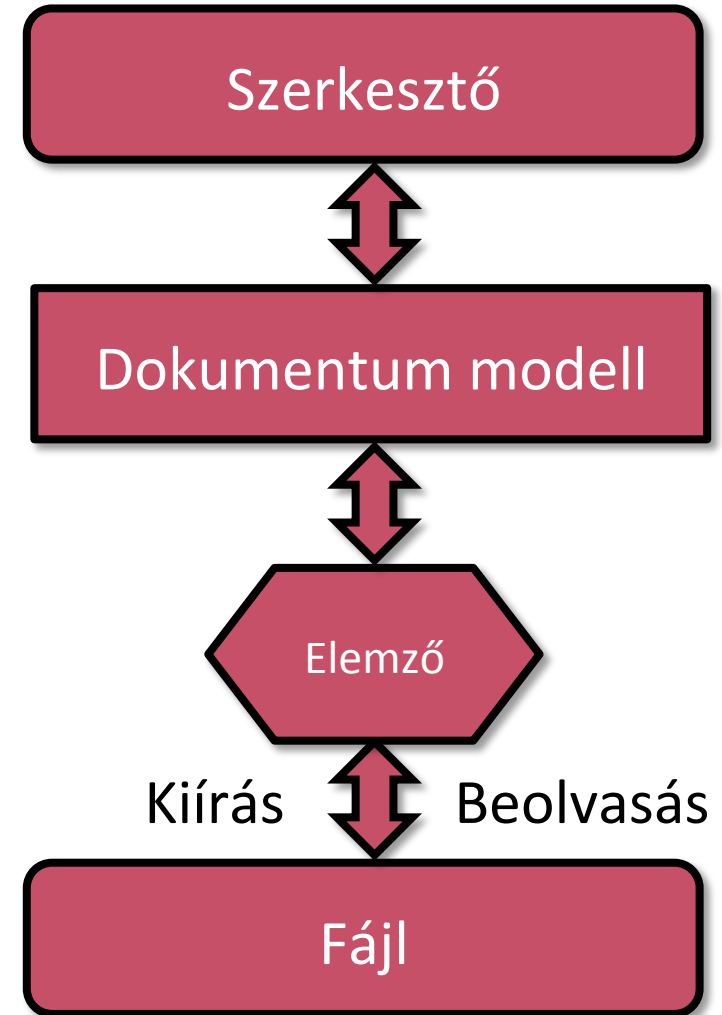
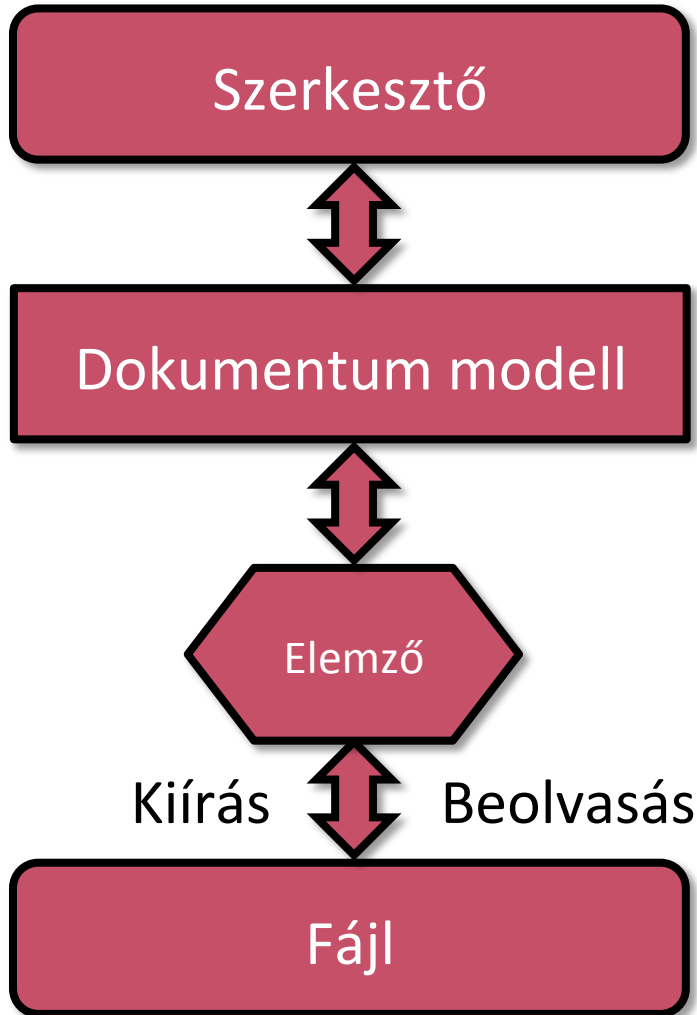
# Architektúra

- Egyszerű szövegszerkesztő



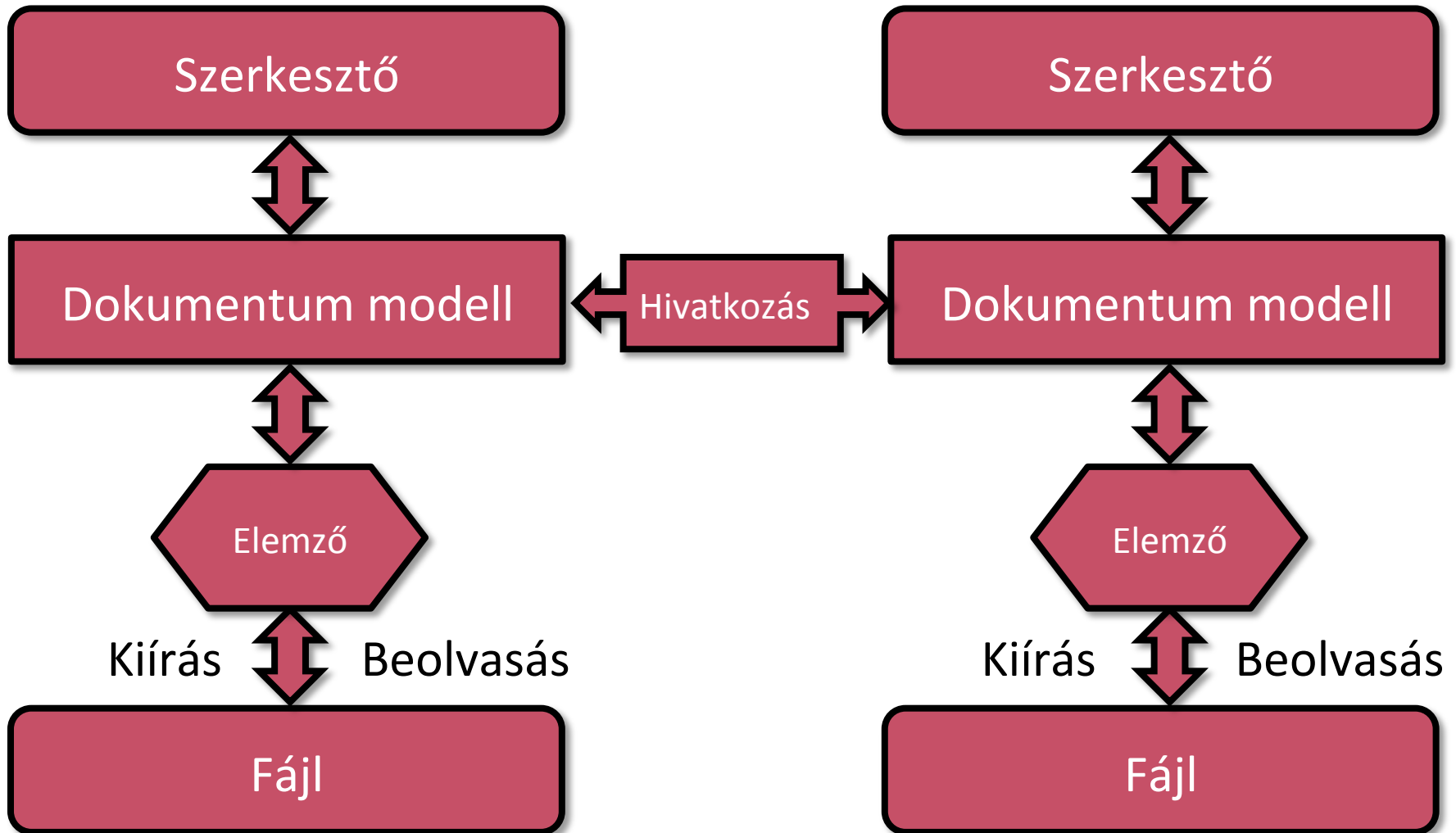
# Architektúra

- Dokumentum modell (MVC architektúra)



# Architektúra

- Dokumentum modell (MVC architektúra)



# Dokumentum modell

- Objektum reprezentáció a forrásfájllhoz
- 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

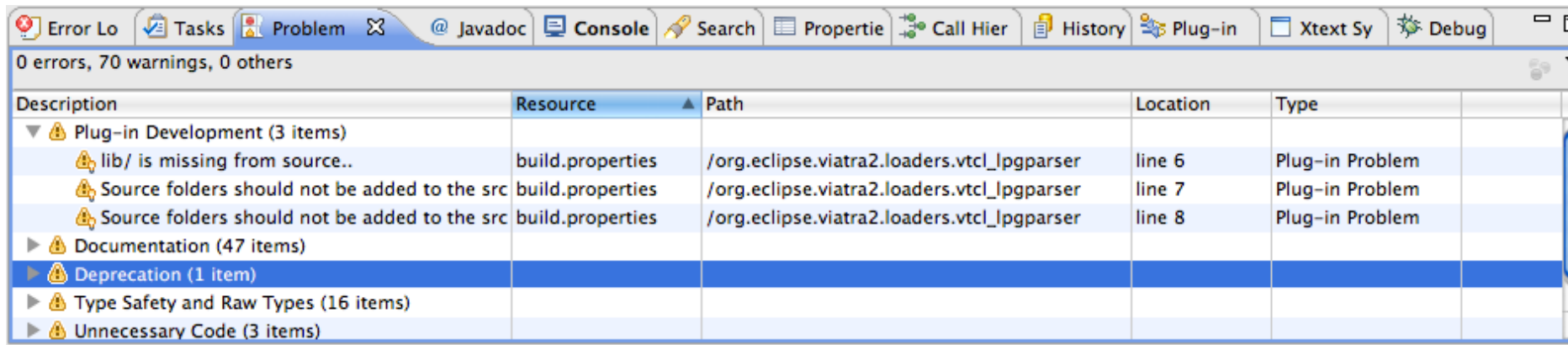
# Jelölők

- **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
  - Hiba esetén súlyosság
  - Üzenet

# 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, the status bar indicates "0 errors, 70 warnings, 0 others". The main area displays a table of problem markers.

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)				

# Automatikus javítás

- Hibajelölőkhöz javítás kapcsolható
  - Lehetséges módosítás(ok)
  - Cél: probléma javítása
  - Modell szintjén praktikus elvégezni

# 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

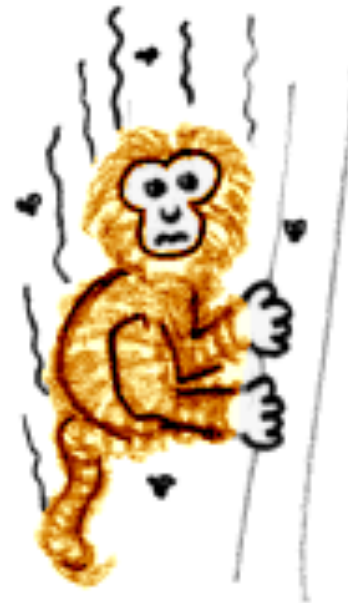
- Természetes nyelvek
- XML feldolgozás
- Nyelvtan alapú feldolgozás
  - Kézzel kódolt elemző
  - Reguláris kifejezések
  - Generált elemző

# Természetes nyelvek feldolgozása

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

# Környezet

We gave the monkeys<sub>1</sub> the bananas<sub>2</sub>  
... because they<sub>1</sub> were hungry.  
... because they<sub>2</sub> were ripe.



# XML feldolgozás

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

# Problémák az XML használatával

- Nehéz írni
  - Lezáró címkék
  - Entitások (pl. &gt; &#8222;)
- 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?



# További probléma

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



# További probléma

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

## Hibajelzés!

*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>

# Nyelvek és nyelvtanok

Formális nyelvek gyorstalpaló

# Formális nyelv

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

# Példa: Természetes számok

# 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», ...

# 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», ...

# 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», ...



# 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

# 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

# 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»



# 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
  - Eldönthetetlenségi problémák
  - Speciális esetek működhetnek

# Környezetfüggő nyelvek (Type 1)

## ■ Kétféle definíció

### ○ Környezetfüggő

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

### ○ Monoton

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

# Környezetfüggő nyelvek (Type 1)

## ■ Kétféle definíció

### ○ Környezetfüggő

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

Az  $A$  nem-terminális kicserélhető  $\gamma$ -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

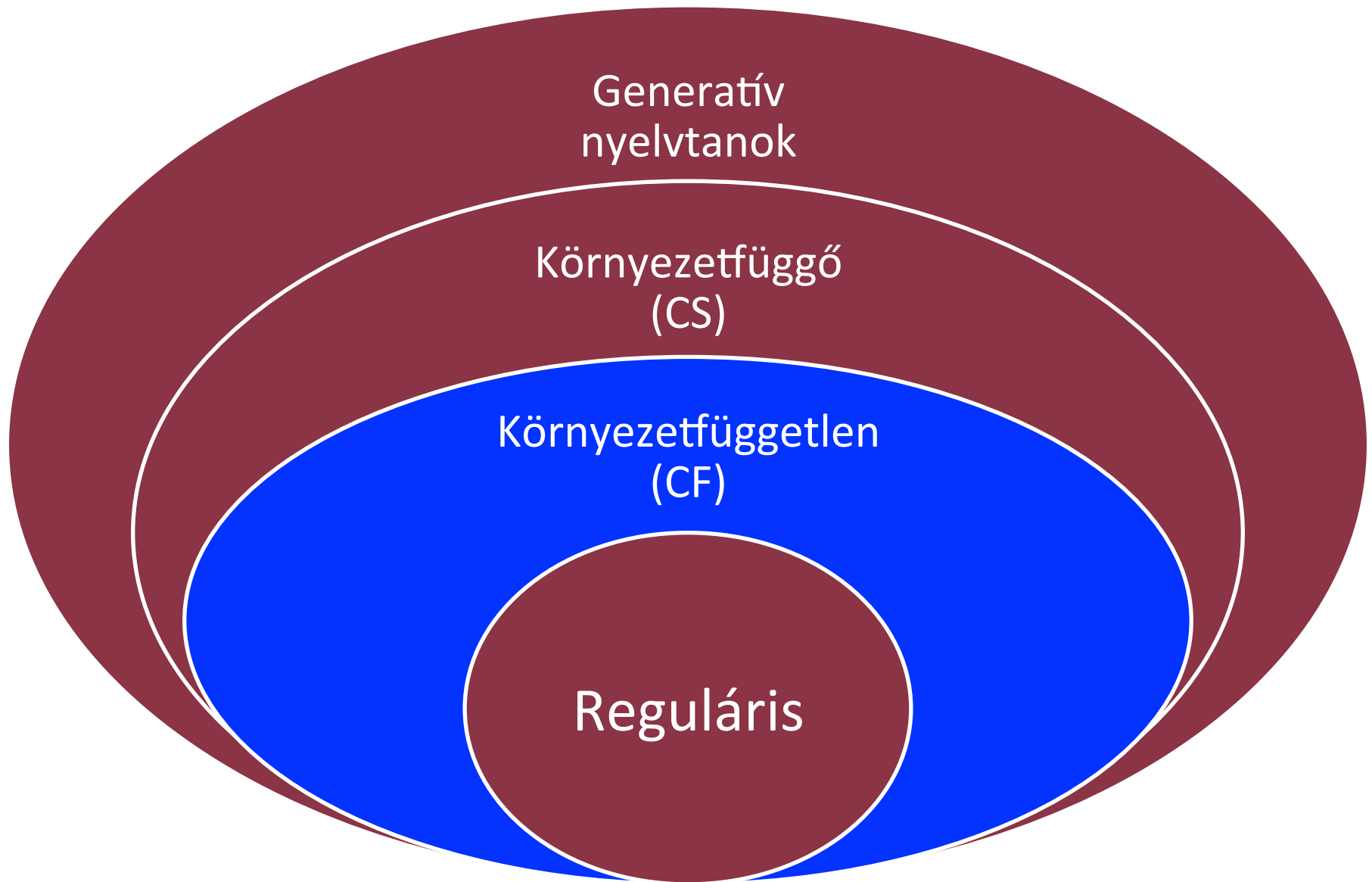
# Reguláris nyelvek (Type 3)

- 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»

# 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»

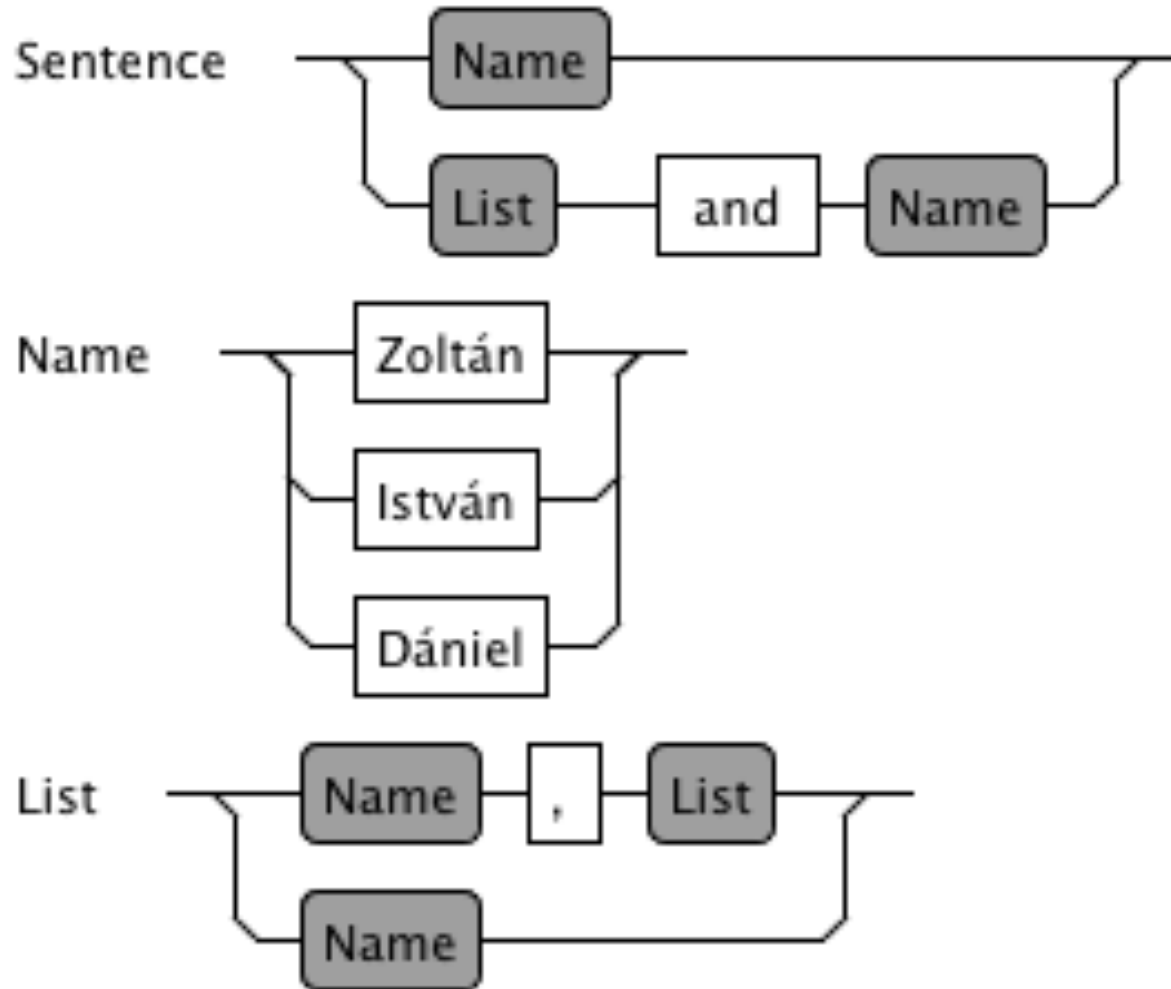
«Név» ::= Zoltán | István | Dániel

«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

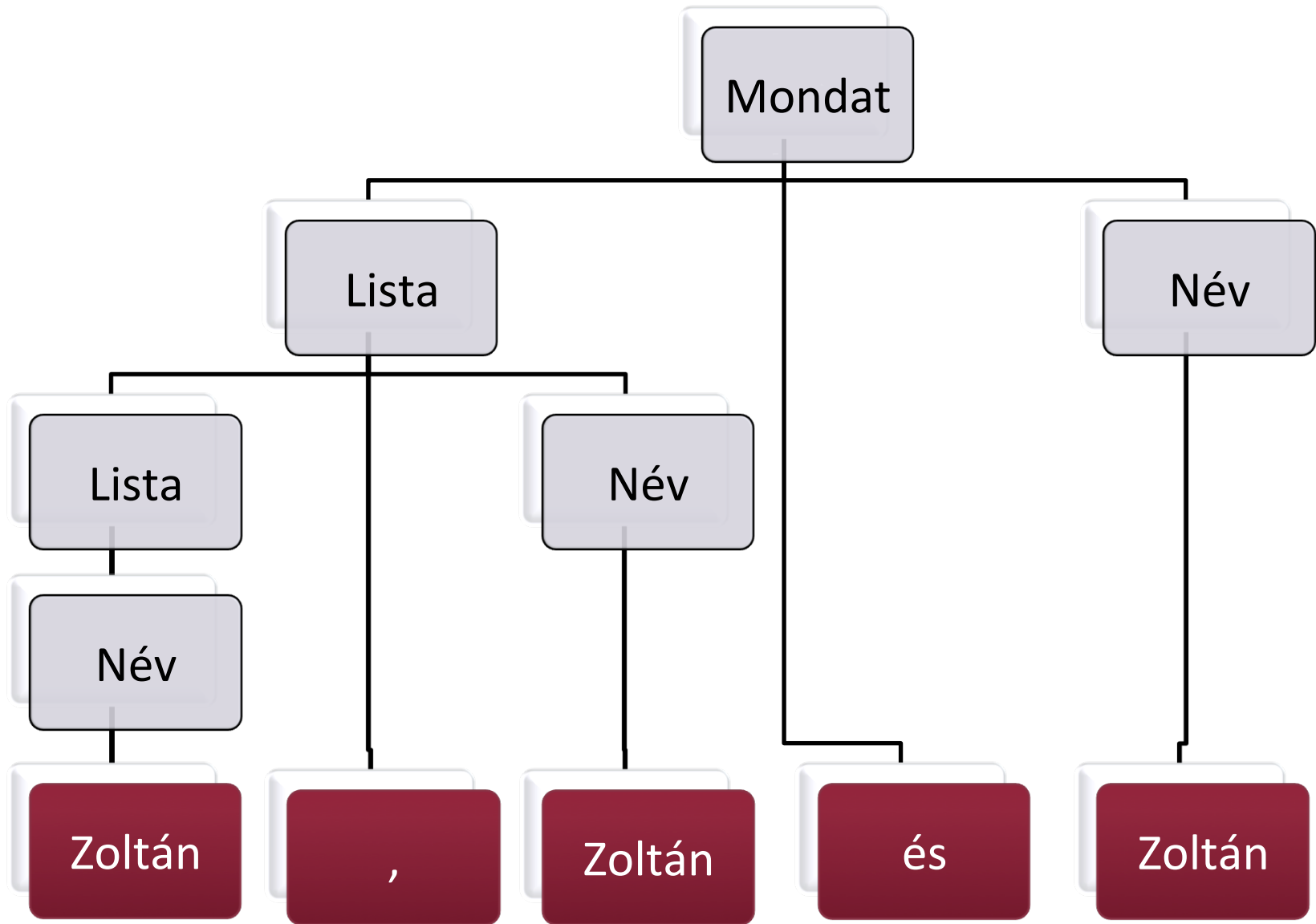
«Mondat» ::= «Név» | «Lista» és «Név»

«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	



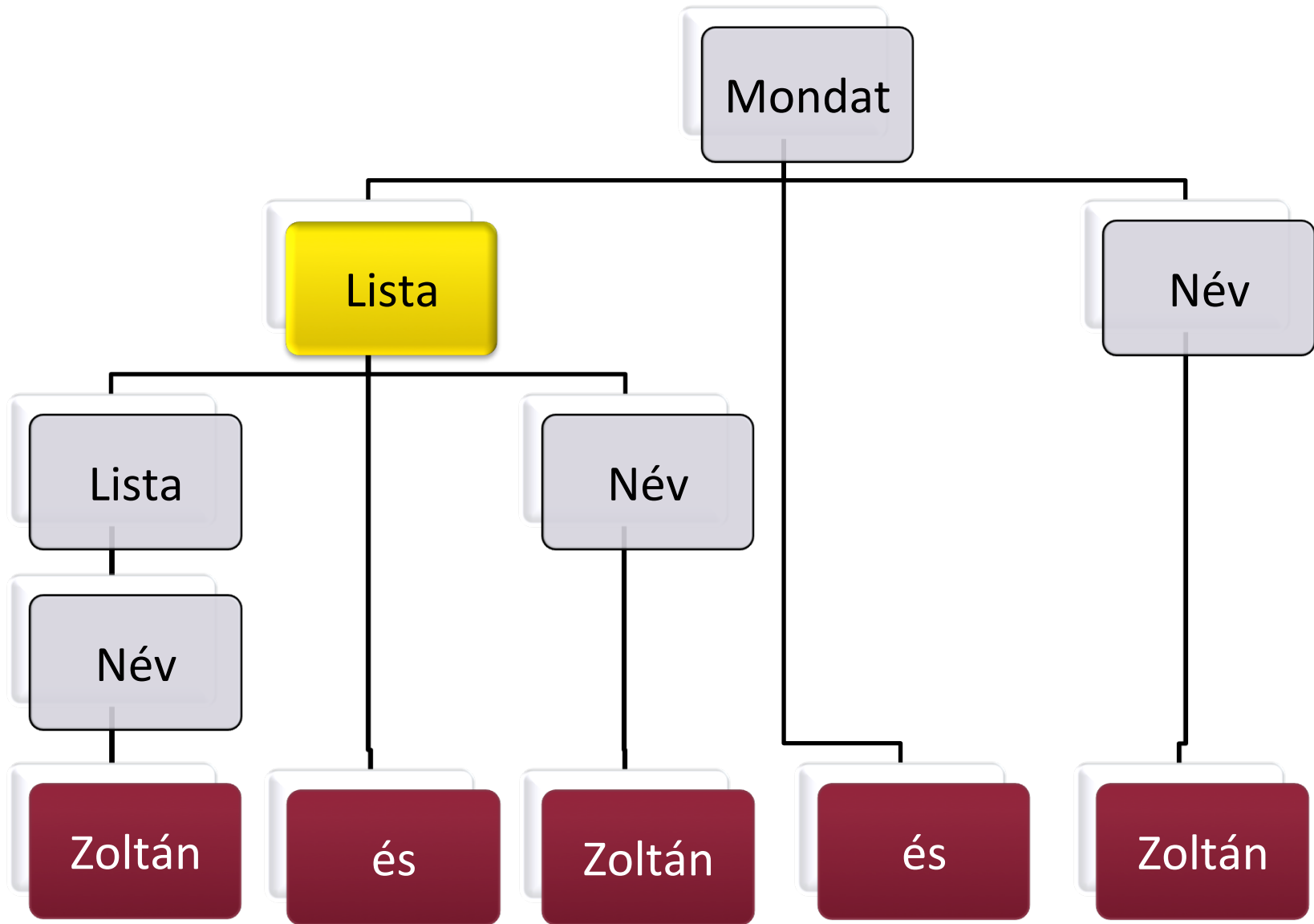
# Levezetési fa



# 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

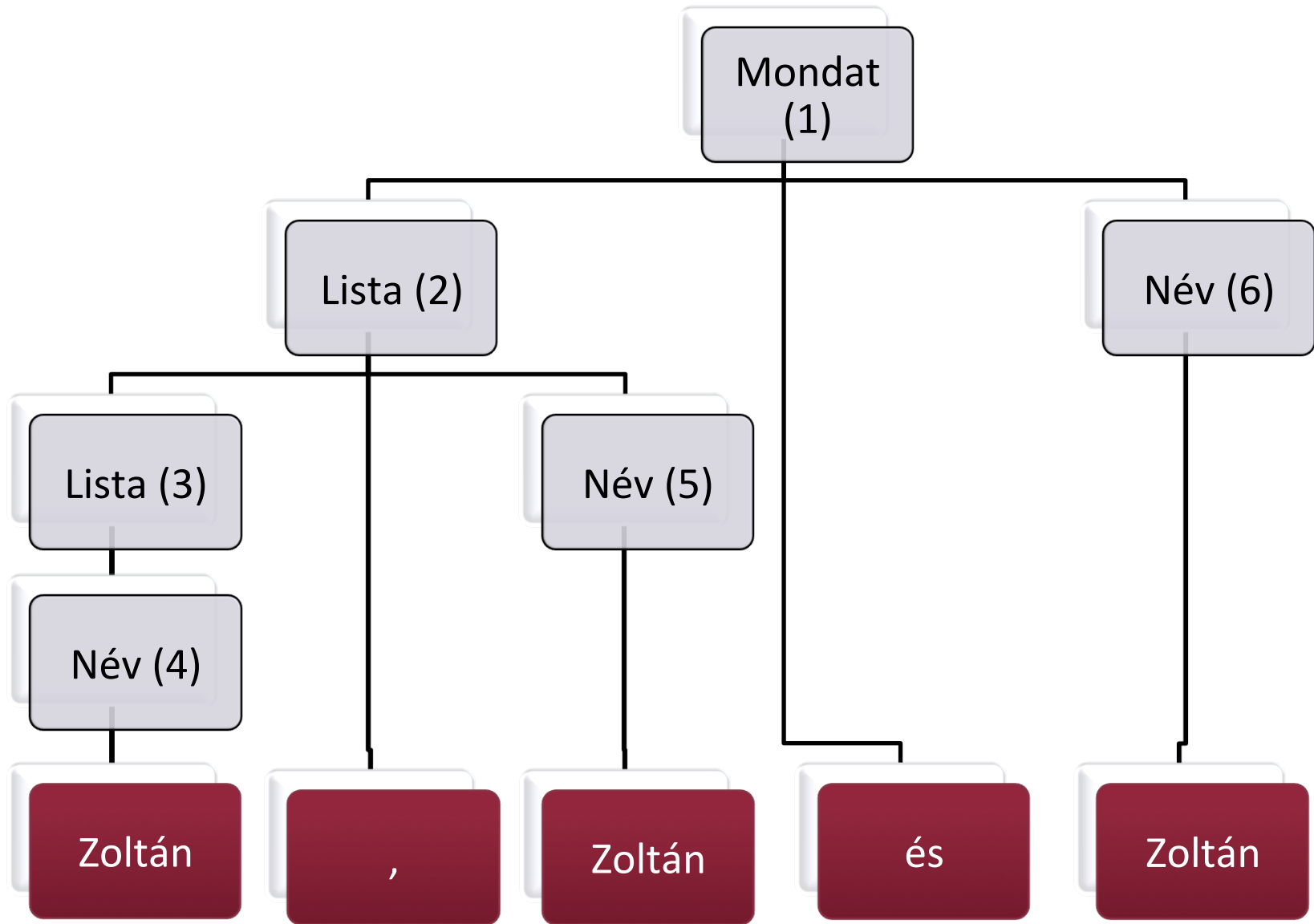
- Kiindulás: Mondatszimbólum
- Cél: szabályalkalmazásokkal eljutni a mondatig
- Elágazási lehetőségek
  - Predict and match
  - Backtracking szükség esetén

# LL(k) elemzők

## ■ Tulajdonságok

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

# Baloldali levezetés



# Alulról felfelé elemzés

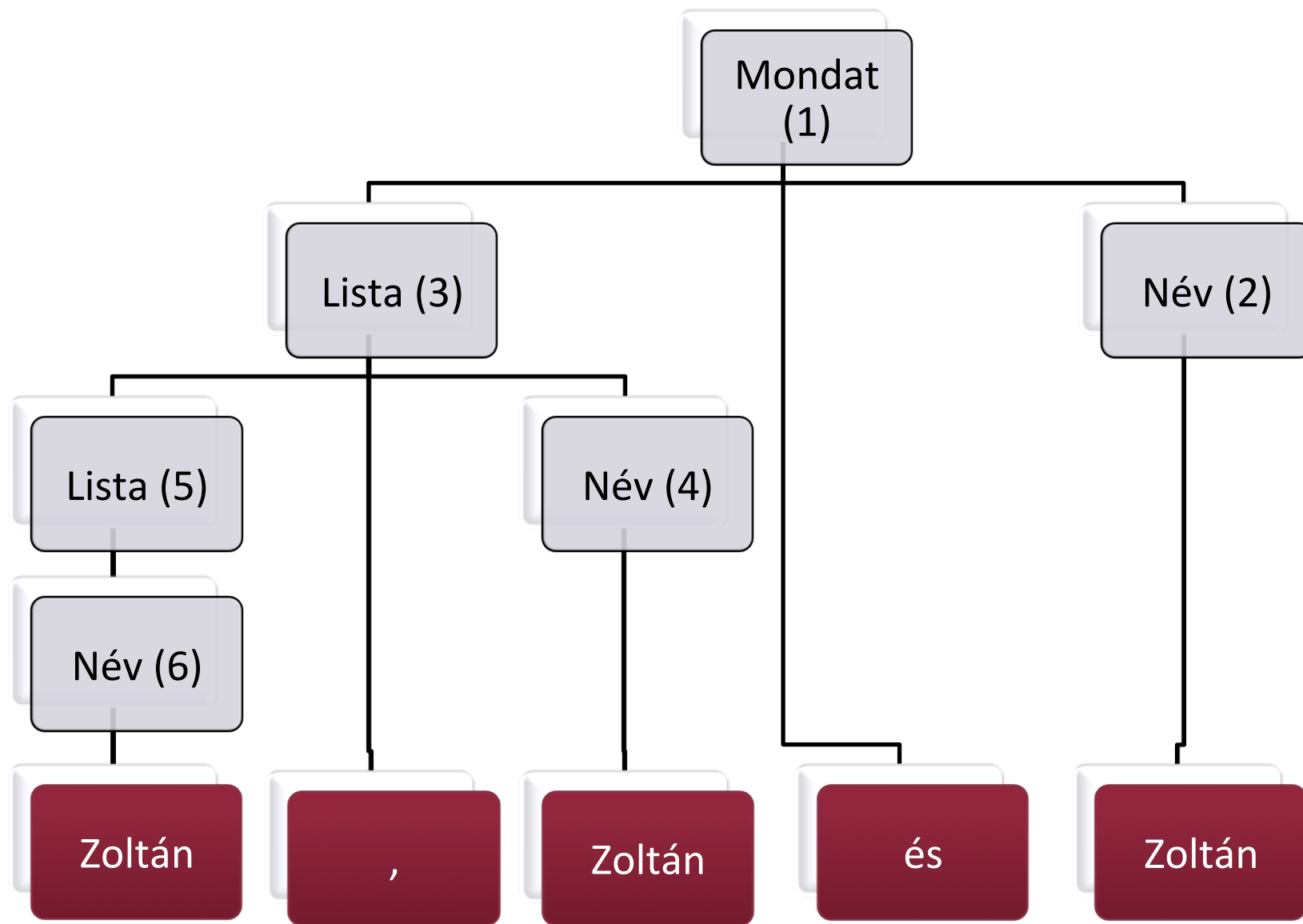
- Kiindulás: Mondat
- Cél: Eljutni a Mondatszimbólumig
- Szabály jobb oldalát illesztjük a szövegre
  - Illeszkedés esetén alkalmazhatjuk a szabályt
  - Visszafelé!



# LR(k) elemzők

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

# Jobboldali levezetés



# Elemzők választása

- $LL(k) \subset LR(k)$
- $LR(k) \subset CF$
  
- Ötletek
  - $LL(k)$ 
    - Egyszerűbb algoritmus
    - Könnyebben érthető a futás -> jobb hibajelzés
  - $LR(k)$ 
    - Nagyobb kifejezőerő
    - Kisebb memóriaigény

# Elemzők a gyakorlatban

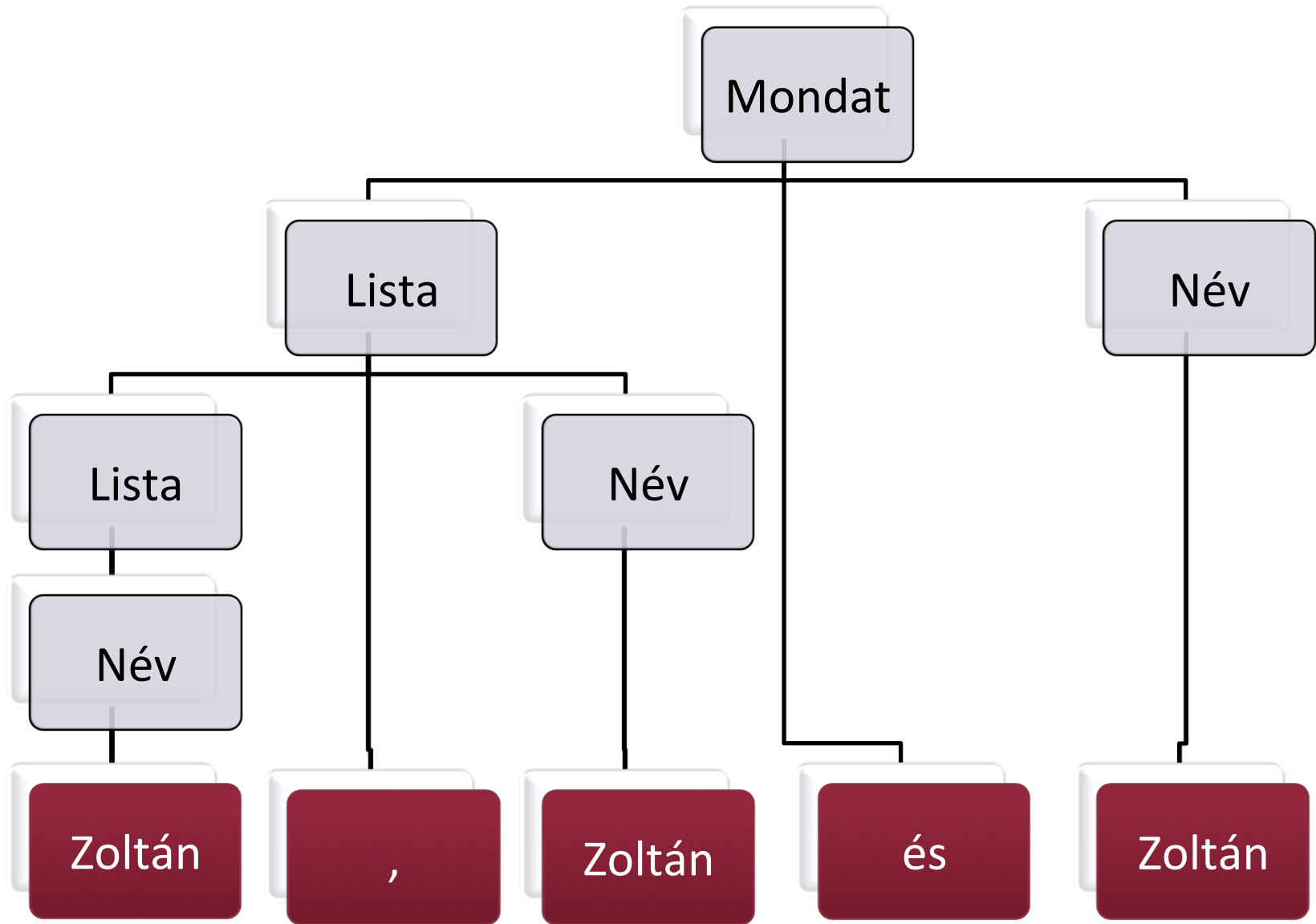
# Áttekintő

- Eddig
  - Nyelvtan
  - Elemző
- Mi hiányzik még?

# Áttekintő

- Eddig
  - Nyelvtan
  - Elemző
- Mi hiányzik még?
  - Bemeneti karakterfolyam kezelés
  - Változók bevezetése
  - Magasabb szintű elemzések

# Bemenet



# Bemenet

Tényleges bemenet:

'Z' 'o' 'l' 't' 'á' 'n' ' ' ' , ' 'Z'  
'o' 'l' 't' 'á' 'n' ' ' ' 'é' 's' ' ' '  
'Z' 'o' 'l' 't' 'á' 'n'

Zoltán

,

Zoltán

és

Zoltán



# Bemenet kezelése

# Bemenet kezelése

- Bemenet:
  - Karakterfolyam
- Elemző bemenete
  - Magasabb szintű tokenek
    - «Név», ‘,’ vagy “és”
  - Kettő közötti kapcsolat: lexer
- Miért hasznos ez az indirekció?
  - Hibakezelés
  - Teljesítmény
  - Probléma dekompozíció

# Lexer

- Cél:
  - Karakterfolyam tokenizálás
  - Hasonlít elemzők problémájához
    - De egyszerűbb – tipikusan reguláris kifejezés
    - Csak szavak/tokenek azonosítása
    - (Kommentek elhagyása)
  - Egyszerűsíti az elemző dolgát

# Változók bevezetése

```
a=3;
```

```
System.out.println(a);
```

## ■ Változó

- Futási időben
  - Érték behelyettesítés
- Szerkesztési/elemzési időben
  - Hivatkozás
  - AST más részére

# Változó kezelés

- Változó hivatkozás (*variable reference*)
  - Egy változó felhasználása
- Változó definíció (*variable definition*)
  - Egy változó megadása
- Egyedi név (jólformáltsági kényszer)
  - Változó definícióknak azonosíthatónak kell lennie
  - Külön fázis elemzés után

# Változó kezelés

- Elemző biztosítja
  - Érvényes változónév-e (szintaktikailag)
- Feloldás ellenőrzi
  - Definíció létezése
    - Hatókör probléma (*scope*)
  - Definíció egyedisége

# Hatókör probléma

```
private int value;
```

```
public void setValue(int value) {  
    this.value = value;  
}
```

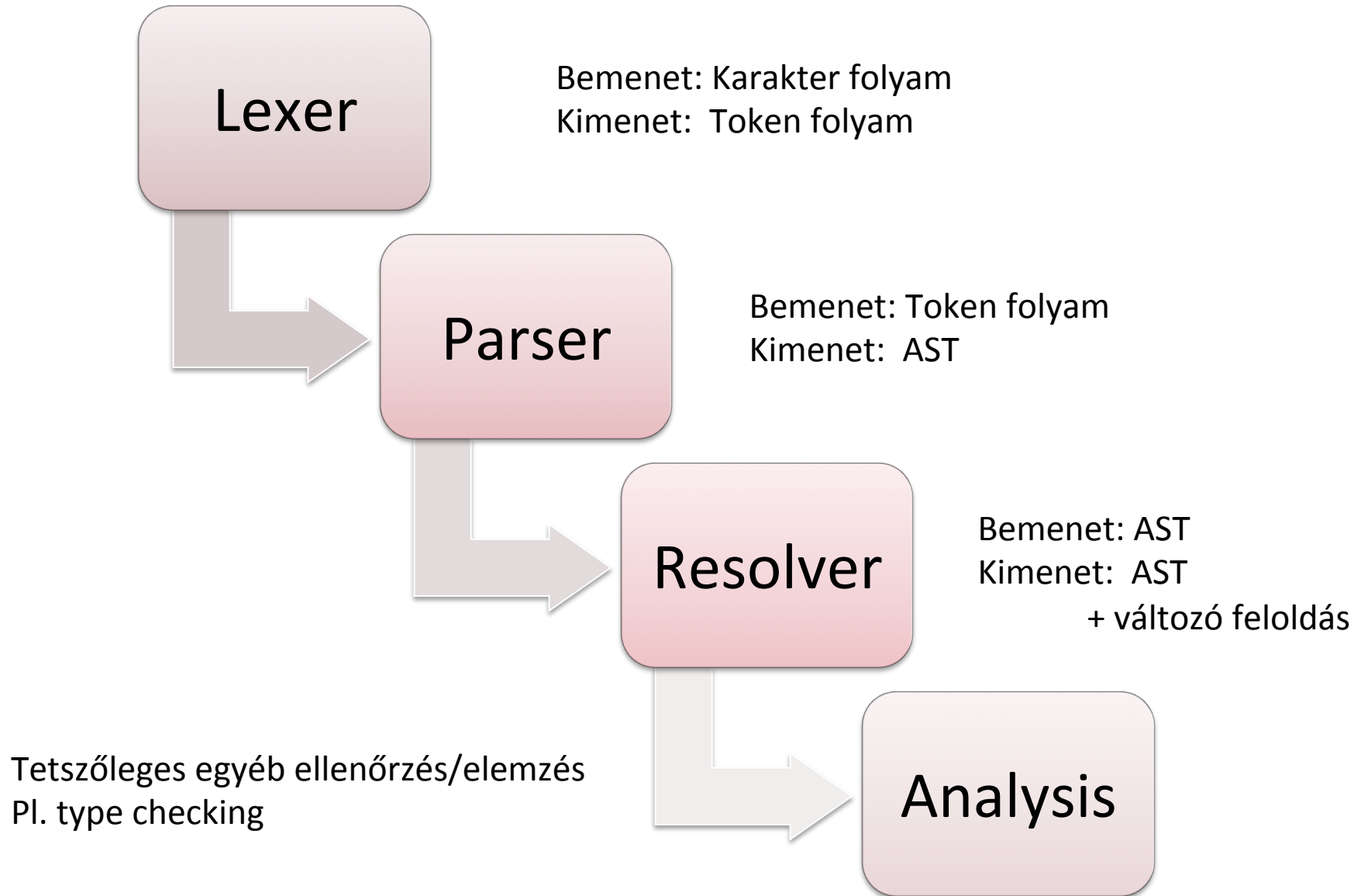
Mire hivatkozik a  
'value' referencia?

# Hatókör probléma

- Megoldás
  - Feloldó szintjén definiálандó
- Lehetséges megközelítések
  - Legspecifikusabb előfordulás
  - Hibajelzés konfliktus esetén
  - Minősített hivatkozások (*qualified name*)
  - ...

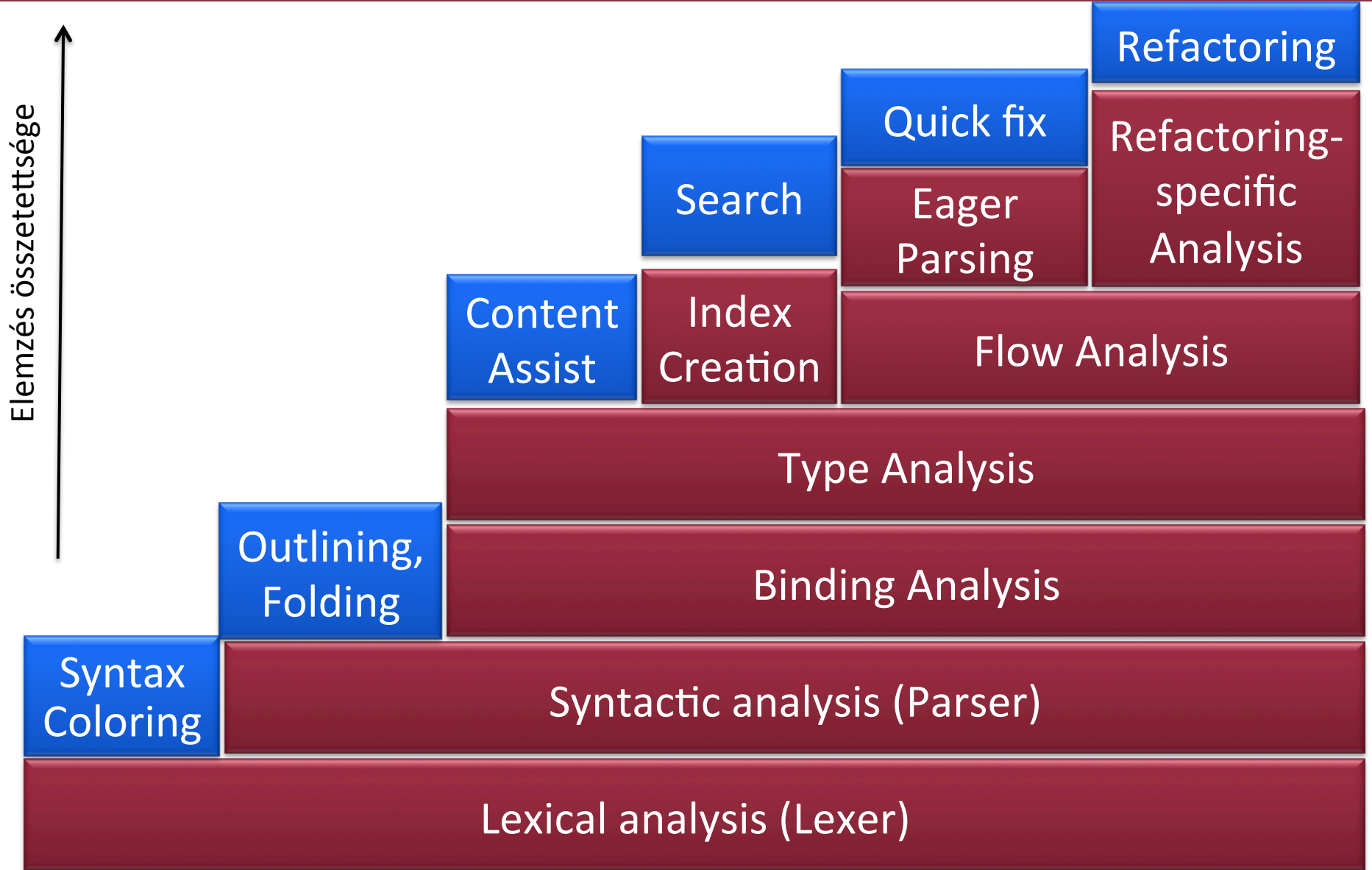


# Elemzés menete



# Elemzők és szerkesztők

# Elemzés fázisai és szerkesztő szolgáltatások



# Elemzés fázisai és szerkesztő szolgáltatások

- Sokféle szolgáltatás
  - Sokféle kapcsolódó elemzés
  - Összefüggések
- Célok
  - Hibatűrés
    - Használható hibajelzés!
  - Teljesítmény

# Szolgáltatások I.

- Kódszínezés (*Syntax Coloring*)
  - Tokenekhez szín/formázás rendelhető
- Vázlat nézet (*Outlining*)
  - AST csomópontok bejárása
- Kódrészletek összehúzása (*Folding*)
  - AST csomópontok bejárása
- Felhasználás megjelenítése (*Mark Occurences*)
  - AST csomópontok + változófeloldás

# Szolgáltatások II.

- Dokumentáció megjelenítése (*Hover*)
  - AST bejárás (feloldás után)
  - Kellhet lexer adat is (pl. Javadoc kommentek)
- Automatikus kiegészítés (*Content Assist*)
  - Összetett elemzések
  - Tipikusan HIBÁS szöveg/modell felett

# Szolgáltatások III.

- Keresés (*Search*)
  - Index-alapú keresése bizonyos típusú AST csomópontoknak
- Automatikus javítás (*Quick Fix*)
  - Hibákhoz javítási lehetőségek keresése
- Refaktor (*Refactoring*)
  - Ekvivalens átalakítások (pl. átnevezés)
- ...

# Megvalósítási lehetőségek I.

- Eclipse DocumentViewer API
  - JFace API – mindent kézzel
- Eclipse IMP projekt
  - Univerzális editor koncepció
  - Szolgáltatások regisztrációja



# Megvalósítási lehetőségek II.

- EMFText és Eclipse Xtext projektek
  - Nyelvtan + Metamodel alapján
  - EMF AST származtatás
  - Nagyon hasonló logika
  - Különböző célok – részletesebben jövő héten