

# DOMENSKO SPECIFIČNI MODELIRNI JEZIKI 2024/2025

7. predavanje

# Ponovimo

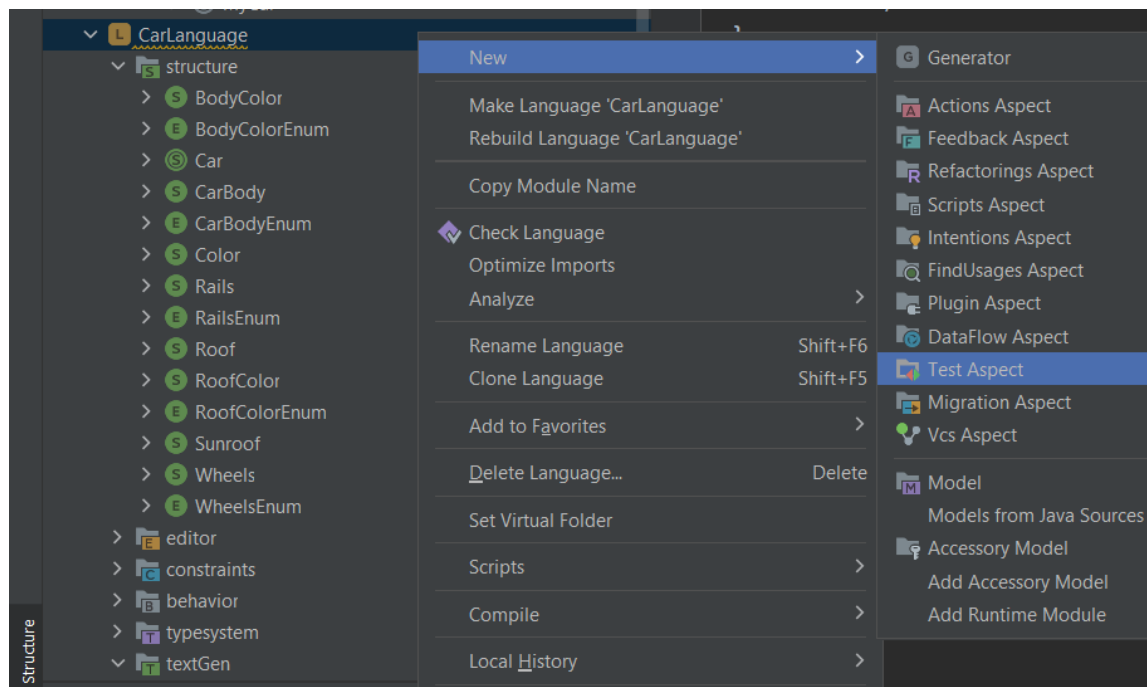
- Moderni tekstovni urejevalniki
- MPS
  - Označevanje sintakse
  - Dejanja
  - Transformacijski meniji
  - Trenutni kontekst
  - Omejitve
  - Omejitve s sistemom tipov
  - Hitri popravki
  - Projekcijski urejevalniki

# Povzetek

- MPS
  - Testi enot
  - Projekcijski urejevalniki
- Načrtovanje domensko-specifičnega jezika
  - Sintaksa
  - Semantika

# Testi enot v MPS

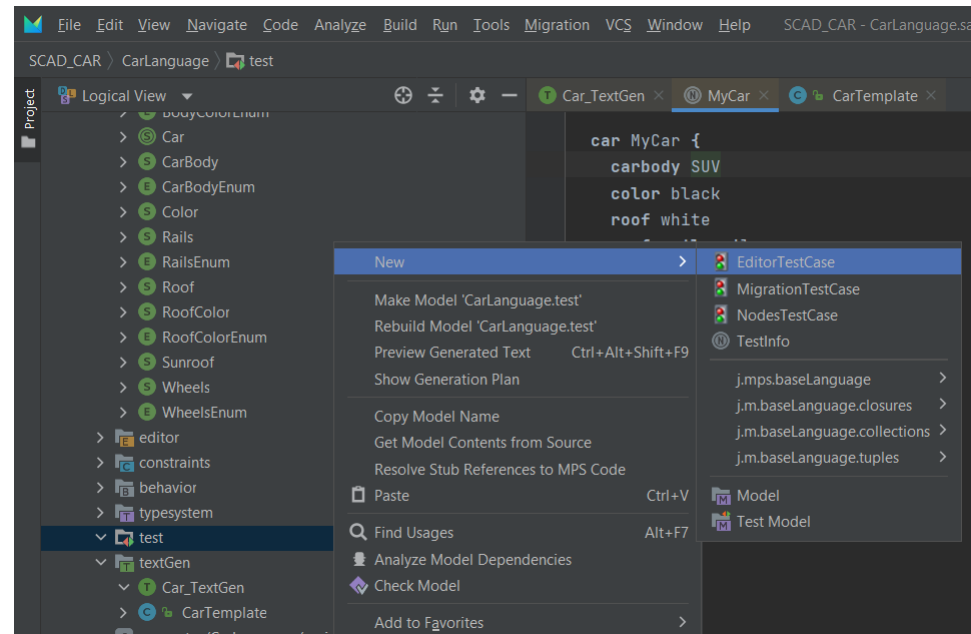
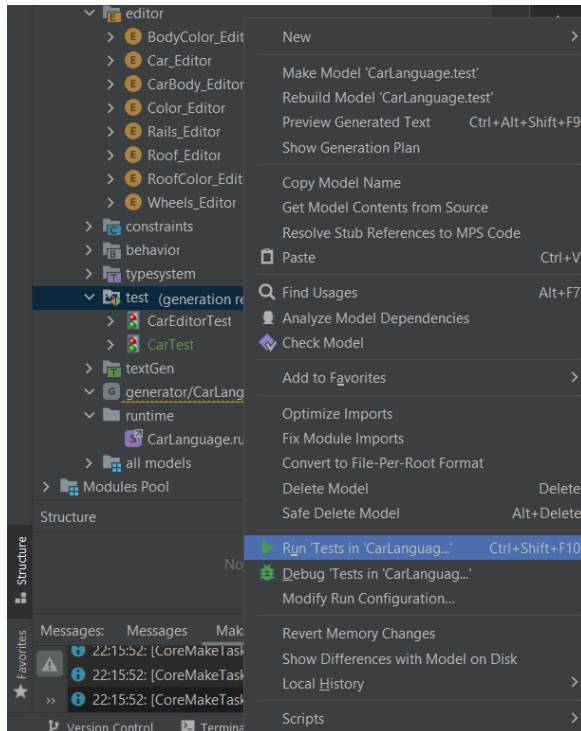
- **Avtomatizirani testi** (testi enot) so ključnega pomena za vzdrževanje in kakovost programske kode – enako velja za razvoj jezika
- Zelo priporočljivo, da jih zapišemo





# Testi enot v MPS

- Testiranje vozlišč, editorja, itd.
- Zapis testnega primera s programom
- Rezultat



# Projekcije v MPS

- Projekcijski urejevalnik
  - Na učinkovit način urejamo predstavitev kode v drevesu abstraktne sintakse (AST).
- Urejevalnik:
  - **besedila** za tekstovne jezike,
  - **diagramov** za grafične jezike,
  - **tabel** za urejanje tabel, itd.



# Razširitve v MPS

- MPS razširitve
  - <https://jetbrains.github.io/MPS-extensions/>
- Npr. urejevalnik matematičnih formul

MPS Math Editor

★★★★★

MPS extensions

Overview Versions Reviews

Mathematical notation for MPS Editors The support includes:

- Mathematical notation for MPS Editors



# MPS verzija

- Nove verzije MPSa ponavadi še ne vsebujejo podpore za razširitve
  - Potrebna je nižja različica (angl. downgrade) MPS

< > ↺

plugins.jetbrains.com/plugin/13252-mps-math-editor/versions

Channels: **Stable**

Version	Compatibility Range	Update Date	
2024			
<a href="#">2023.2.1</a>	2023.2 — 2023.2.2	Jun 10, 2024	<a href="#">Download</a>
<a href="#">2022.2.3</a>	2022.2 — 2022.2.4	Apr 25, 2024	<a href="#">Download</a>
<a href="#">2022.3.1</a>	2022.3 — 2022.3.3	Mar 26, 2024	<a href="#">Download</a>
<a href="#">2023.3</a>	2023.3 — 2023.3.2	Mar 26, 2024	<a href="#">Download</a>
<a href="#">2022.2.2</a>	2022.2 — 2022.2.4	Mar 10, 2024	<a href="#">Download</a>
<a href="#">2023.2</a>	2023.2 — 2023.2.2	Mar 10, 2024	<a href="#">Download</a>





# Tabele v MPS

- Razširitev:
  - <https://plugins.jetbrains.com/plugin/13255-mps-table-editor-component>

```
reykill MyWebGenerator {  
  name : NotesApplication  
  title : Notes application  
  description : This is a Notes Application  
  port : 4100  
  components : title : Home  
                url : /  
                menu : false  
                title : About  
                url : /about  
                menu : false  
                title : ToDo  
                url : /todo  
                menu : false  
  api : base : http://localhost:3000/
```

	Name	URL
1	items	/item-done
2	finished	/item-finished
3	todo	/item-todo

```
render : ( Home =>  
  # Hello to Notes Application  
  # Go to the following links:  
  Thi is our main page  
  ( About =>  
    # About  
    This is about page  
    ( ToDo =>  
      # ToDo page  
      Table with all items  
    )  
  )  
)
```




# Namestitev razširitve

- Prenesite ga v datotečni sistem
- Ohranite zip (datoteke ni potrebno ekstrahirati)
- Pojdite na MPS
  - Namestite vtičnik z diska ...

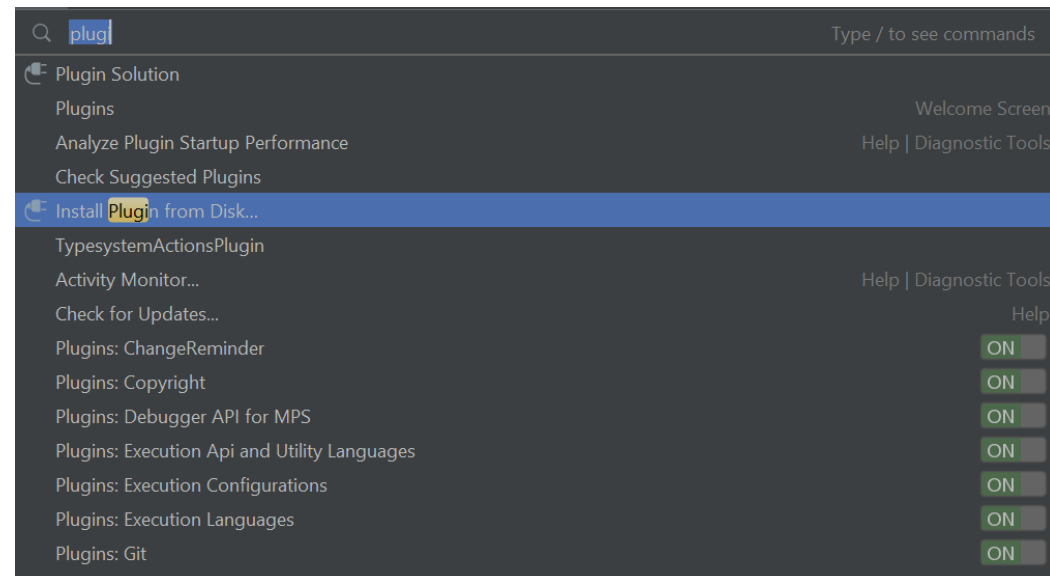
## MPS Table Editor Component Plugin Is Being Downloaded

### How to Install 'MPS Table Editor Component'

1. Install an IDE if you don't have one:
2. Open your IDE and press **Ctrl+Alt+S** to open the IDE settings.
3. Select **Plugins**, click  and then click **Install Plugin from Disk**.
4. Select the plugin archive file and click **OK**.
5. Click **OK** to apply the changes and restart your IDE if prompted.

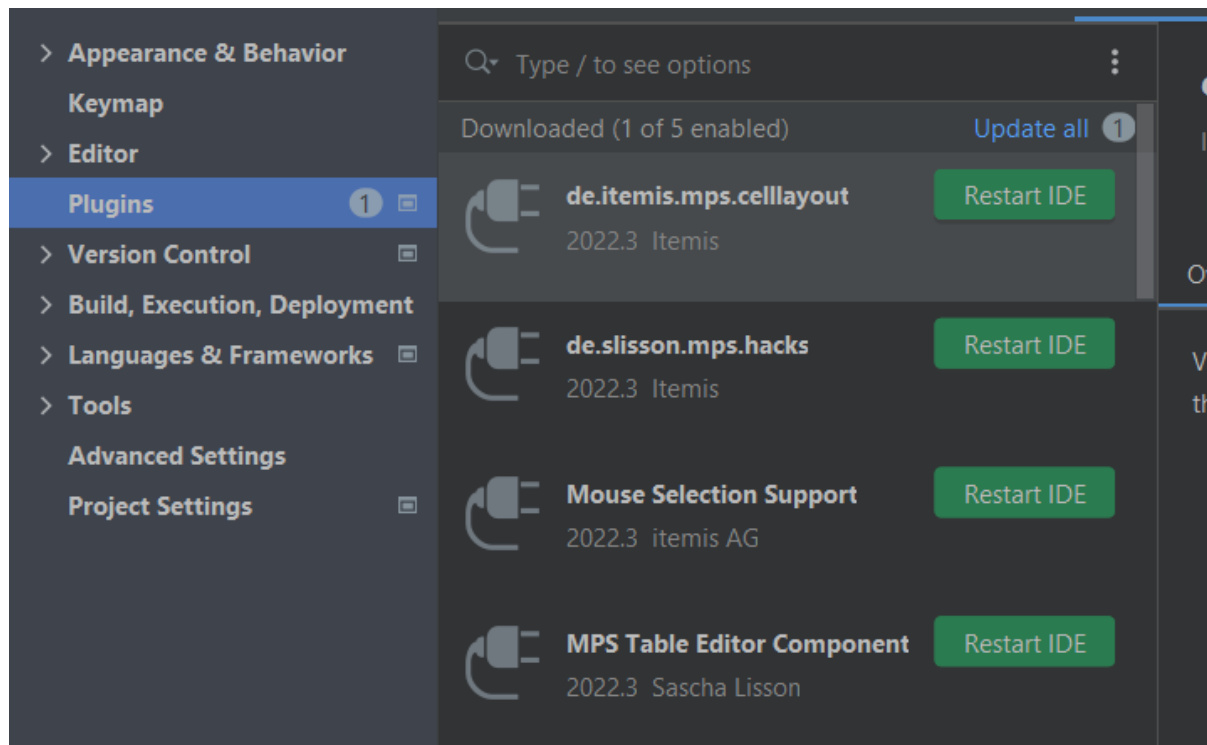
☐ Don't show this again

Got it





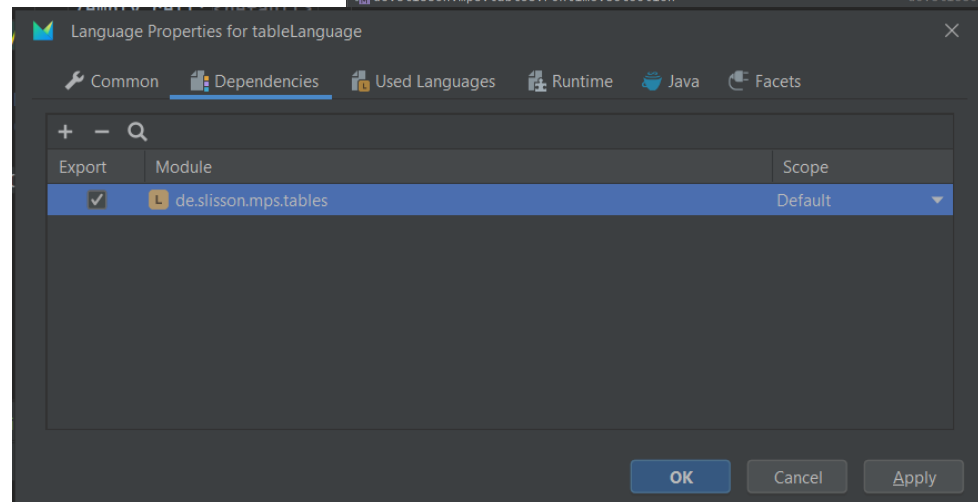
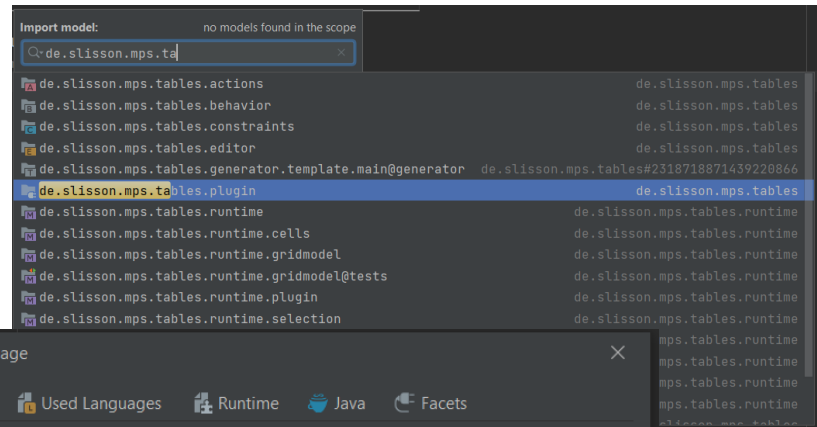
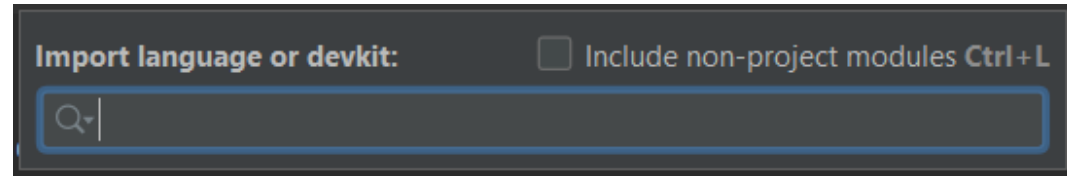
# Namestitev razširitve





# Vključevanje v projekt

- Vključevanje jezika
- Vključevanje modela
- Odvisnosti





# Zapis tabele

```
<default> editor for concept RApi
node cell layout:
[/
  [- base : { base } -]
  [- <constant> -]
  table {
    vertical%endPointList% r< query {
      getHeaders Headers (node, editorContext)->join(string | EditorCell | node<> | Iterable) {
        return node.endPointList.select({~it => Integer.toString(it.index + 1); });
      }
      insert new header (node, index)->void {
        node.endPointList.add new initialized(<default>);
      }
      on delete: (node, index)->void {
        if (node.endPointList.size > 1) {
          node.endPointList.removeAt(index);
        }
      }
    } > c< <no columnHeaders> >
  }
  [- <constant> -]
  /]
```

inspected cell layout:  
<choose cell model>

```
concept RApi extends BaseConcept
  implements <none>

instance can be root: false
alias: <no alias>
short description: <no short description>

properties:
  base : string

children:
  endPointList : REndPoint[0..n]

references:
  << ... >>
```

api : base : http://localhost:3000/

	Name	URL
1	items	/item-done
2	finished	/item-finished
3	todo	/item-todo

New Cell



# Zapis delne (angl. partial) tabele

```
<default> editor for concept REndPoint
node cell layout:
  partial table {
    horizontal r<> {
      cell { name } c<"Name"> r<>
      cell { url } c<"URL"> r<>
    }
  }
```

```
inspected cell layout:
<choose cell model>
```

```
concept REndPoint extends BaseConcept
  implements <none>

instance can be root: false
alias: <no alias>
short description: <no short description>

properties:
  name : string
  url  : string

children:
  << ... >>

references:
  << ... >>
```

```
api : base : http://localhost:3000/
```

	Name	URL
1	items	/item-done
2	finished	/item-finished
3	todo	/item-todo

# Matematična projekcija v MPS

```
MyTest
EquationTest_Editor x MyTest x
math
text : before node
  100
  ---
 2000000000
text : after node
```



# Podpora ulomkom (v urejevalniku)

```
EquationTest x MyTest x

concept EquationTest extends BaseConcept
    implements INamedConcept

instance can be root: true
alias: <no alias>
short description: <no short description>

properties:
before : string
x      : integer
y      : integer
after  : string

children:
<< ... >>

references:
<< ... >>
```

```
EquationTest_Editor x MyTest x

<default> editor for concept EquationTest
node cell layout:
[/]
[- text : { before } -]
math {
    child cells: upper { x } 1.0
                  lower { y } 1.0

    symbols: << ... >>
    variables: << ... >>

    init: <no initFunction>

    layout: (childCells, cell, symbols)->void {...}

    paint: (g, childCells, cell, symbols)->void {...}

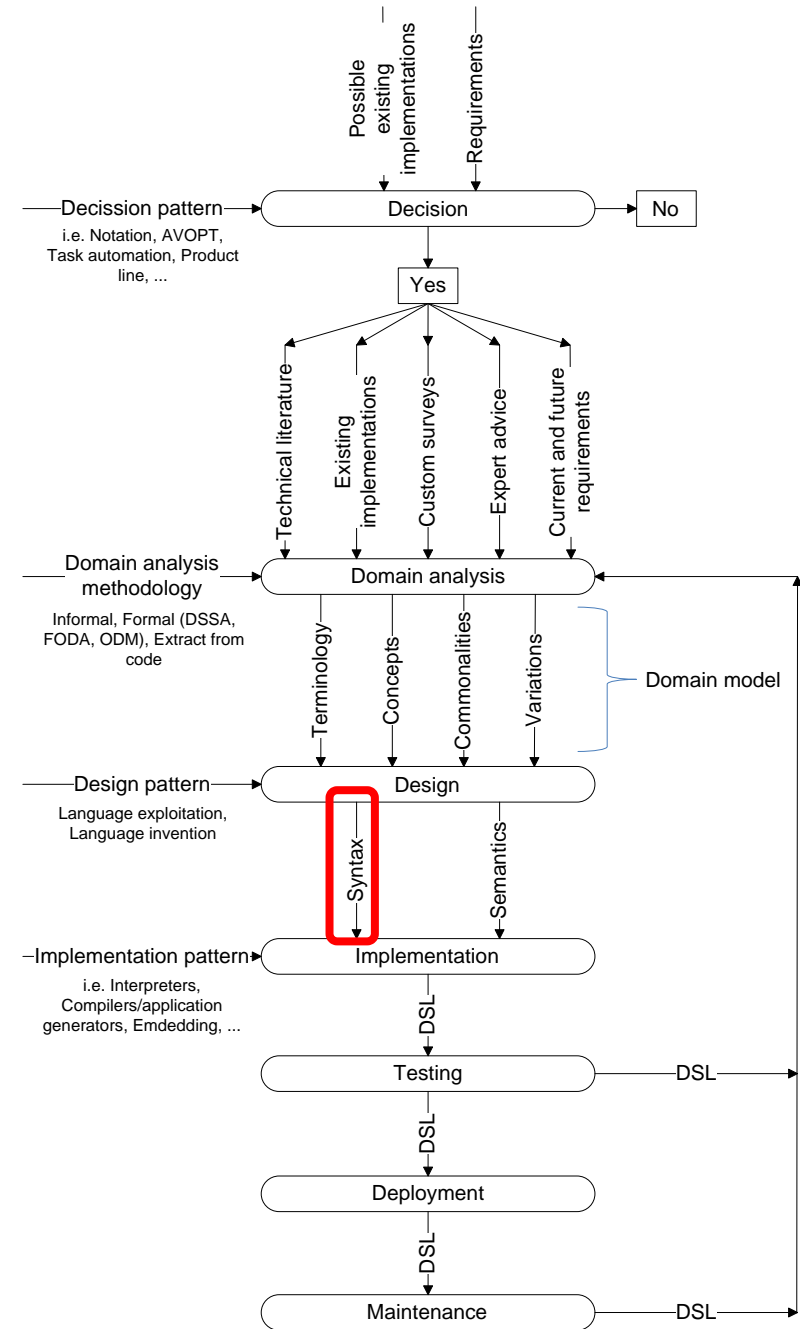
    getCenterY: (childCells, cell)->int {
        upper.getHeightInt();
    }
}
[- text : { after } -]
[/]
```



# Podpora ulomkom (v urejevalniku)

```
layout: (childCells, cell, symbols)->void {  
    cell.setWidth(Math.max(upper.getWidthInt(), lower.getWidthInt()));  
    upper.setX((cell.getWidthInt() - upper.getWidthInt()) / 2);  
    lower.setX((cell.getWidthInt() - lower.getWidthInt()) / 2);  
    upper.setY(0);  
    lower.setY(20);  
}  
  
paint: (g, childCells, cell, symbols)->void {  
    int x = Math.min(lower.getXInt(), upper.getXInt());  
    int x2 = Math.max(lower.getXInt() + lower.getWidthInt(), upper.getXInt() + upper.getWidthInt());  
    g.drawLine(x, 22, x2, 22);  
}
```

# Življenjski cikel



# Sintakse v DS(M)L

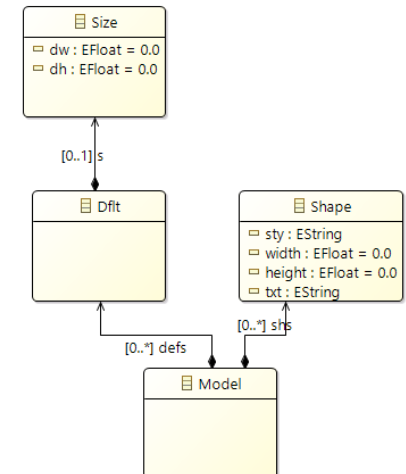
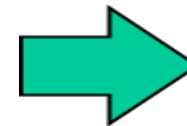
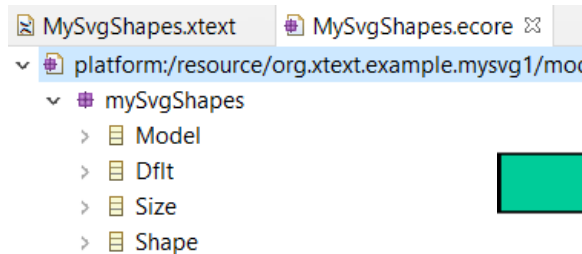
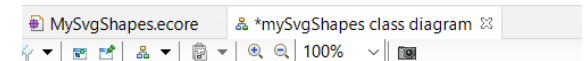
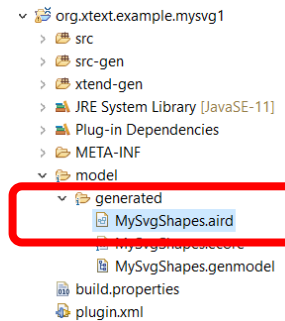
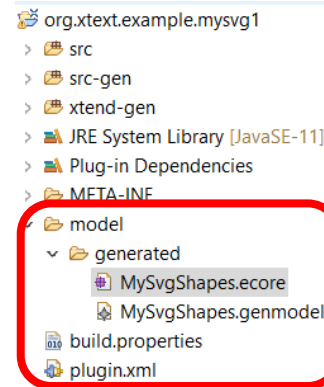
- Različni zapisi sintakse DSL
  - Kontestno proste gramatike (CFG)
  - DTD in XML sheme
  - JSON, itd.
- Zapis sintakse DSML
  - **Konceptualni razredni diagram oz. metamodeli**



# Avtomatična pretvorba (možna v nekaterih orodjih)

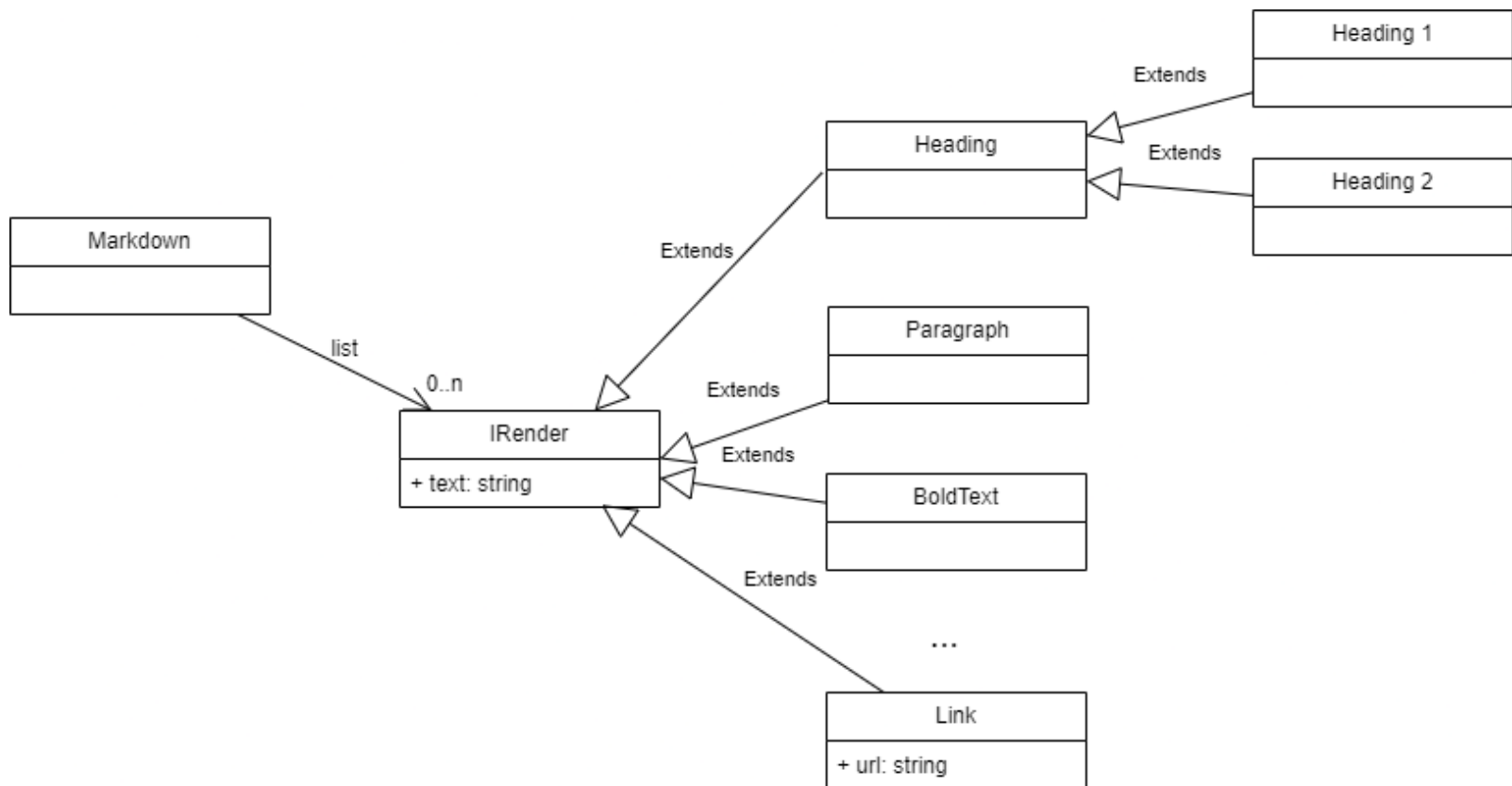
- Xtext → Ecore → konceptualni razredni diagram

```
MySvgShapes.xtext
1 grammar org.xtext.example.mysvg.MySvgShapes with org.eclipse.xtext.common.Terminals
2
3 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
4
5 generate mySvgShapes "http://www.xtext.org/example/mysvg/MySvgShapes"
6
7 @Model: defs+=Dflt* shs+=Shape*
8 ;
9
10 Dflt: {Dflt} 'default' Skip? s=Size?
11 ;
12
13 Skip: 'Skip' FLOAT FLOAT FLOAT FLOAT;
14
15 Size: 'Size' dw=FLOAT dh=FLOAT;
16
17 @Shape: {Shape} ( sty=ShapeStyle | ('choice' ChoiceEdge ChoiceEdge ChoiceEdge ChoiceEdge) )
18 ( (width=FLOAT height=FLOAT)? & txt=STRING);
19
20 ShapeStyle: 'oval' | 'box' ;
21
22 @ChoiceEdge: '.'
23 | 'yes'
24 | 'no'
25 ;
26
27 FLOAT returns ecore::EFloat: '-'? INT ('.' INT)? ;
```



# Načrtovanje DSL - sintaksa

- Primer konceptualnega razrednega diagrama



# Načrtovanje DSL - sintaksa

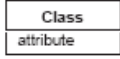


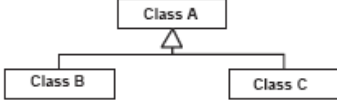


- Kontekstno prosta gramatika
  - Koncepti in relacije med njimi so zapisani z **neterminali** in **terminali**.
- Konceptualni razredni diagram
  - Predstavitev konceptov iz domene z **razredi**.
    - Pozor. Ni nujno, da konceptualni razredi opisujejo razrede v programski kodi!
- Zato lahko za zapis sintakse jezika uporabimo oba formalizma.

# Načrtovanje DSL - sintaksa

- Konceptualni model
  - Razredni (konceptualni) diagram iz UML diagramskih tehnik se uporablja za modeliranje domene
  - Koncepti domene
    - Domeno sestavljajo **ime in atributi** razreda
    - Razred **nima metod(!)**
  - Relacije
    - **Povezava z labelo** predstavlja razmerje med dvema razredoma domene
    - Relacija vsebuje omejitev **kardinalnost**

# Načrtovanje DSL - sintaksa


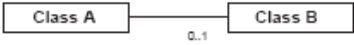
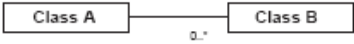

- Obstajajo pretvorbena pravila
  - Metamodel  $\rightarrow$  CFG

Association	Class diagram element	Grammar
Class		Class (non-terminal) attribute (terminal)
Association		$A ::= B$
Navigability		$A ::= B$
Generalization		$A ::= B \mid C$
Aggregation		$A ::= B$ $(\neg \exists X \in N, X \Rightarrow B)$ $\wedge X \neq A$
Composition		$A ::= B$

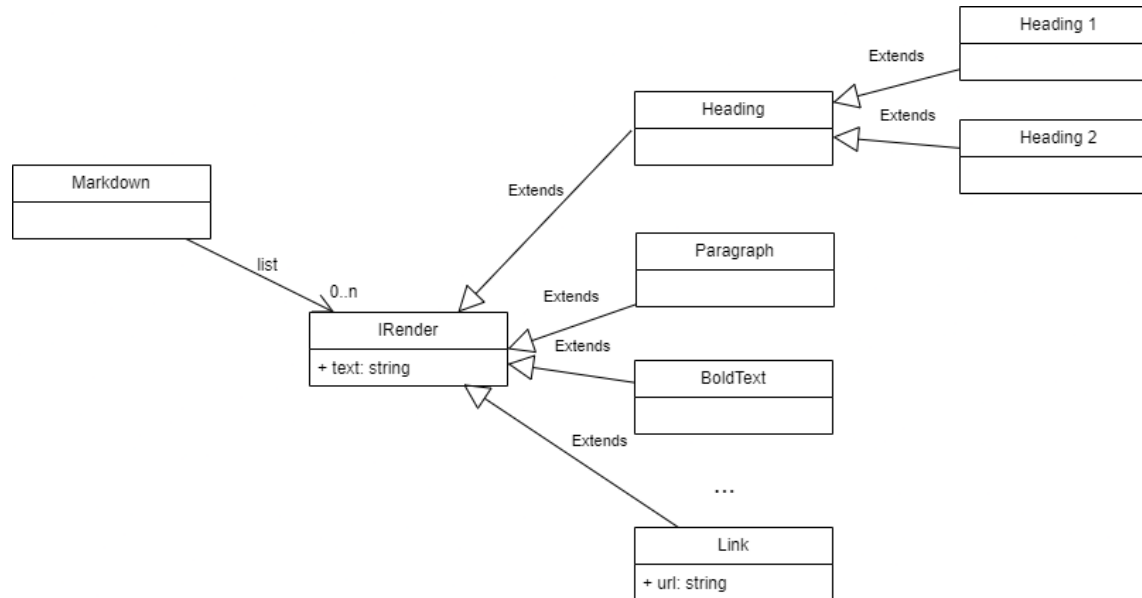




# Načrtovanje DSL - sintaksa

Cardinality	Class diagram element	Grammar	EBNF
Multiplicity exactly one		$A ::= B$	
Optional multiplicity		$A ::= B \mid \epsilon$	$A ::= B?$
Multiplicity $[0..m]$		$A ::= \text{MoreB}$ $\text{MoreB} ::= \text{MoreB } B \mid \epsilon$	$A ::= B^*$
Multiplicity many		$A ::= \text{MoreB}$ $\text{MoreB} ::= \text{MoreB } B \mid B$	$A ::= B^+$

# Sintaksa: EBNF vs. konceptualni razredni diagram (primer 1)



- Pretvorba razrednega diagrama v EBNF:

Markdown	::= IRender+
IRender	::= (Heading   Paragraph   BoldText   ...   Link) text
Heading	::= Heading1   Heading2
Heading1	::= '#'
Heading1	::= '##'
...	



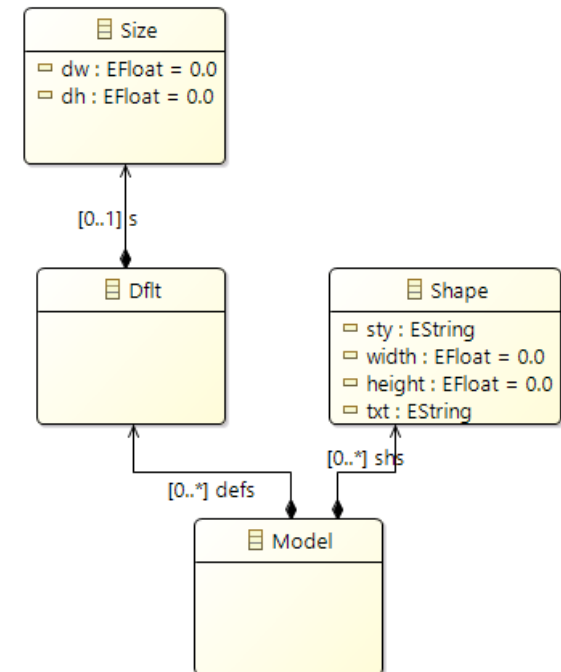
# Sintaksa: EBNF vs. konceptualni razredni diagram (primer 3)

- Specifična pretvorba za orodje
- Xtext → konceptualni razredni diagram

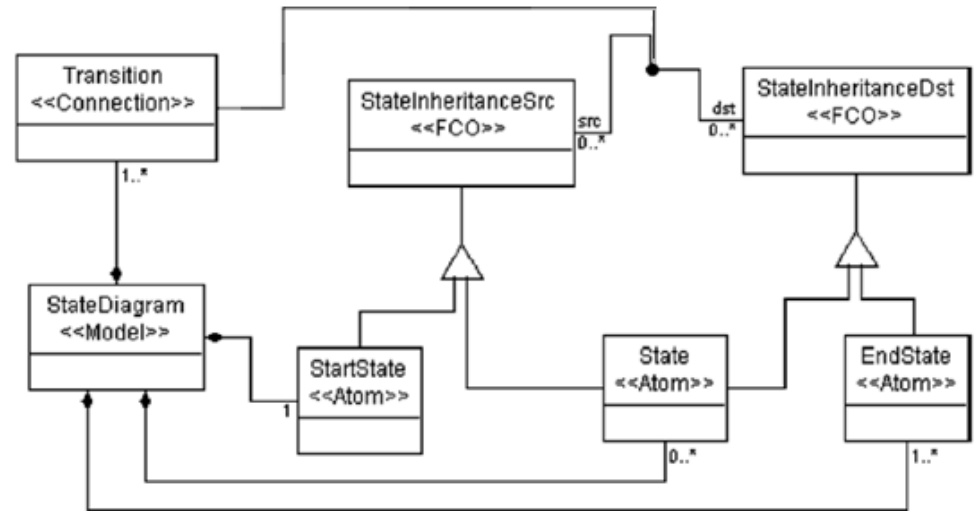
```

1 grammar org.xtext.example.mysvg.MySvgShapes with org.eclipse.xtext.common.Terminals
2
3 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
4
5 generate mySvgShapes "http://www.xtext.org/example/mysvg/MySvgShapes"
6
7 @Model: defs+=Dflt* shs+=Shape*
8 ;
9
10 Dflt: {Dflt} 'default' Skip? s=Size?
11 ;
12
13 Skip: 'Skip' FLOAT FLOAT FLOAT FLOAT;
14
15 Size: 'Size' dw=FLOAT dh=FLOAT;
16
17 Shape: {Shape} ( sty=ShapeStyle | ('choice' ChoiceEdge ChoiceEdge ChoiceEdge ChoiceEdge) )
18           ( (width=FLOAT height=FLOAT)? & txt=STRING);
19
20 ShapeStyle: 'oval' | 'box' ;
21
22 ChoiceEdge: '.'
23             | 'yes'
24             | 'no'
25 ;
26
27 FLOAT returns ecore::EFloat: '-'? INT ('.' INT)? ;
  
```

MySvgShapes.ecore \*mySvgShapes class diagram



# Sintaksa: EBNF vs. konceptualni razredni diagram (primer 4)



- Pretvorba metamodela v EBNF:

StateDiagram ::= startState state\* endState+ Transition+

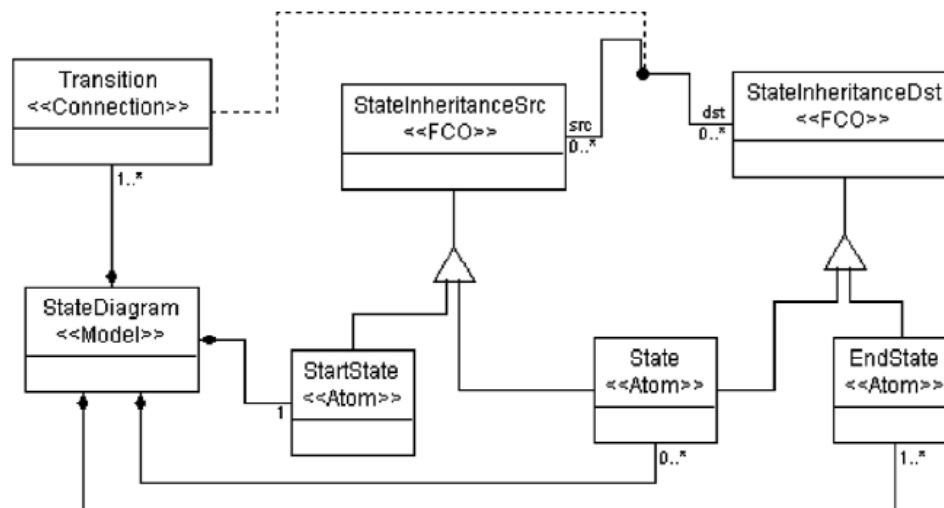
Transition ::= Source Destination

Source ::= startState | state

Destination ::= state | endstate

# Analogija z načrtovanjem DSML

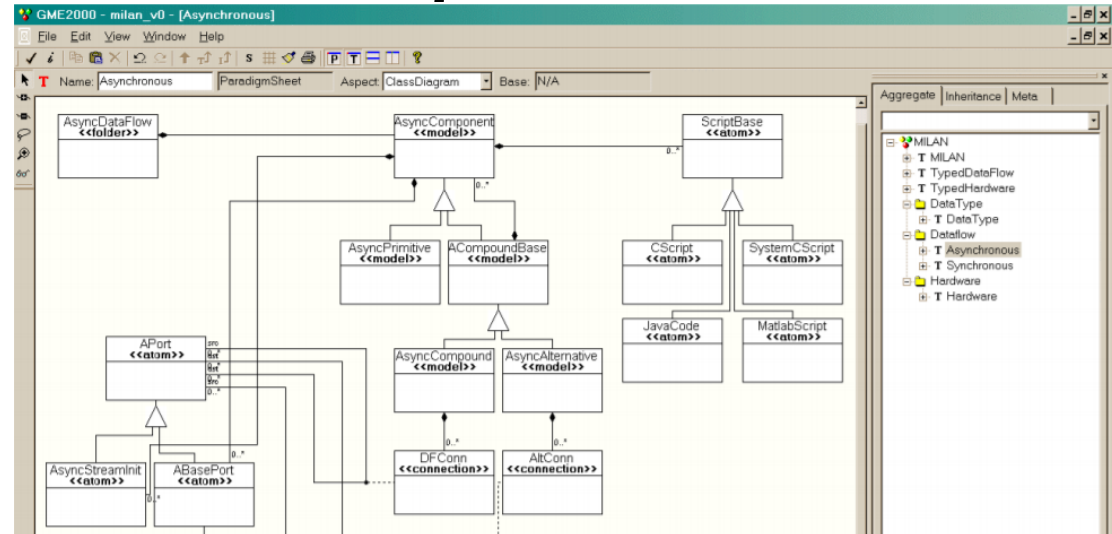
- Konceptualni razredni diagram se uporablja za definiranje **sintakse v domensko-specifičnih modelirnih jezikih (DSML)**
- DSML – konceptualni model v orodju GME
  - **Atom**: generična entiteta, ki nima strukture (notranjih objektov)
  - **Model**: generična entiteta, ki vsebuje vse ostale entitete v modelu
  - **Connections**: primarni koncepti, ki opisujejo povezave med objekti
  - **FCO** (angl. first class objects): entitete v modelu



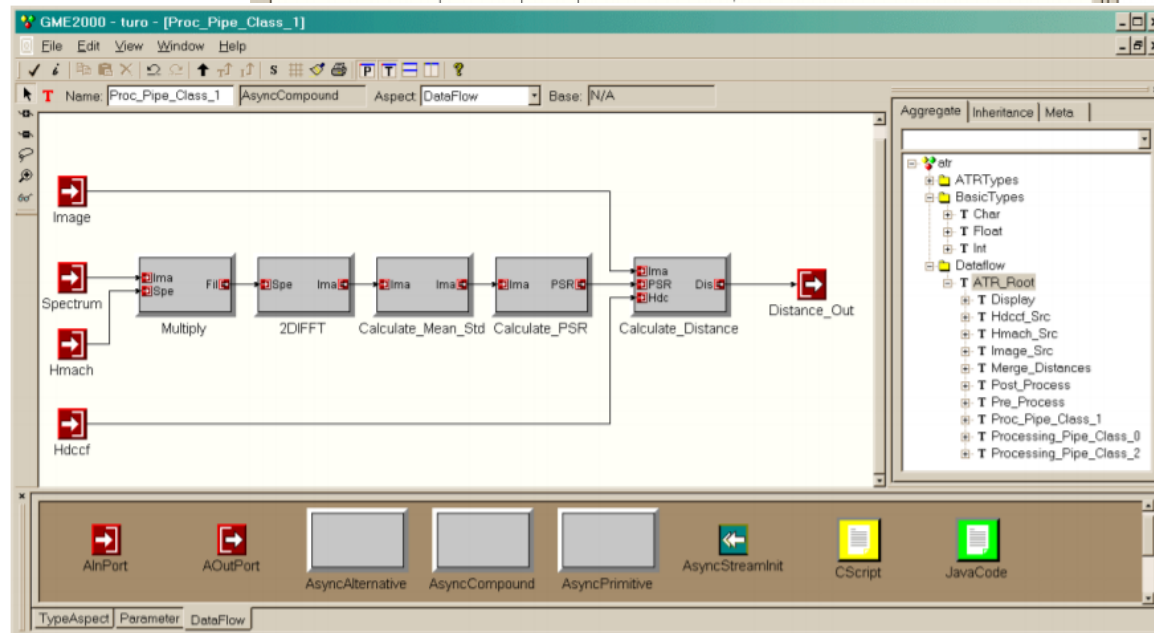


# Analogija z načrtovanjem DSML

- Metamodel

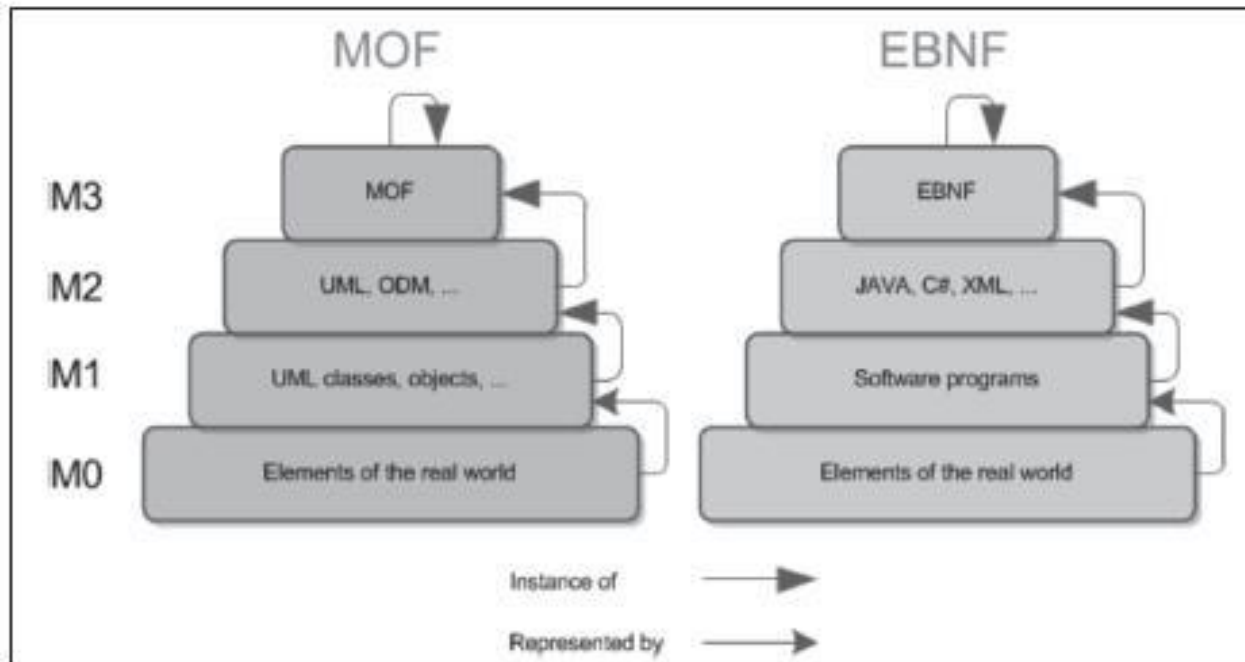


- Model



# Analogija z načrtovanjem DSML

- A brief introduction to model-driven engineering



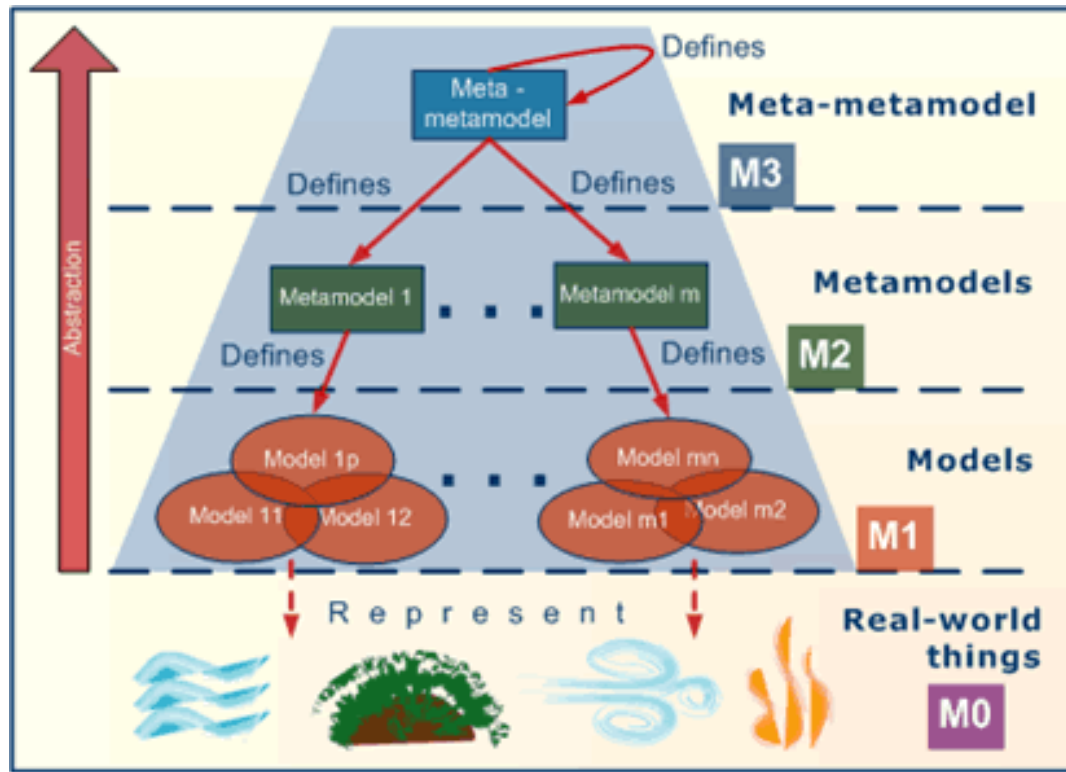
Vir: [link](#)



# Analogija z načrtovanjem DSML

- The Tao of Modeling Spaces

Vir: [link](#)



# Primerjava sintakse DSL z DSML

- Kontekstno proste gramatike (CFG) vs. konceptualni razredni diagram (metamodel)
  - Metamodeli vsebujejo **več informacij** (relacij, atributov).
  - CFG lahko vsebuje **odvečne podatke** (imena podstruktur).
  - V CFG je **zaporedje elementov** v produkciji natančno določeno.
  - CFGji opisujejo **družino dreves**, metamodeli pa opisujejo **družino grafov**.
  - V CFG je **začetni simbol** (neterminal) znan, medtem ko ta **v primeru metamodelov** ni nujen.
  - CFGje je **lažje razpoznati** (procesirati) kot metamodela.

# Načrtovanje DS(M)L

- Različni zapisi sintakse
  - Kontestno proste gramatike (CFG)
  - Konceptualni razredni diagram oz. metamodeli
  - **DTD, XML sheme**
  - JSON, itd.

# Načrtovanje jezikov z XML

- XML je v bistvu zapis za **opis drevesne strukture**.
- **Vozlišča v drevesu** so prikazana v parih oznak:
  - odpiranje (<a>) in
  - zapiranje (</a>).
- Vozlišča v drevesu so označena (npr. <a>).
- Vozlišče lahko ima tudi **atribut**. Prikaže se tako, da attribute dodamo skupaj z vrednostmi v začetno oznako.
- **DTD** (Document Type Definition) in **XML scheme**
  - Še en formalizem za opisovanje sintakse jezika

# Načrtovanje DSL - sintaksa

- Primer DTDja:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE people [
  <!ELEMENT people (person*)>
  <!ELEMENT person (first , last)>
  <!ATTLIST person
    pid ID #REQUIRED
    sex (male|female) #REQUIRED
  >
  <!ELEMENT first (#PCDATA)>
  <!ELEMENT last (#PCDATA)>
]>
```



# Načrtovanje DSL - sintaksa

- Primer programa:

```
<people>
  <person pid="54343" sex="female">
    <first>Mary</first>
    <last>Jones</last>
  </person>
  <person pid="54344" sex="male">
    <first>John</first>
    <last>Jones</last>
  </person>
</people>
```



# Sintaksa: EBNF vs. XML

- EBNF vs. XML

- Ekvivalentni primer programa:

```
54343 female Mary Jones
54344 male      John Jones
```

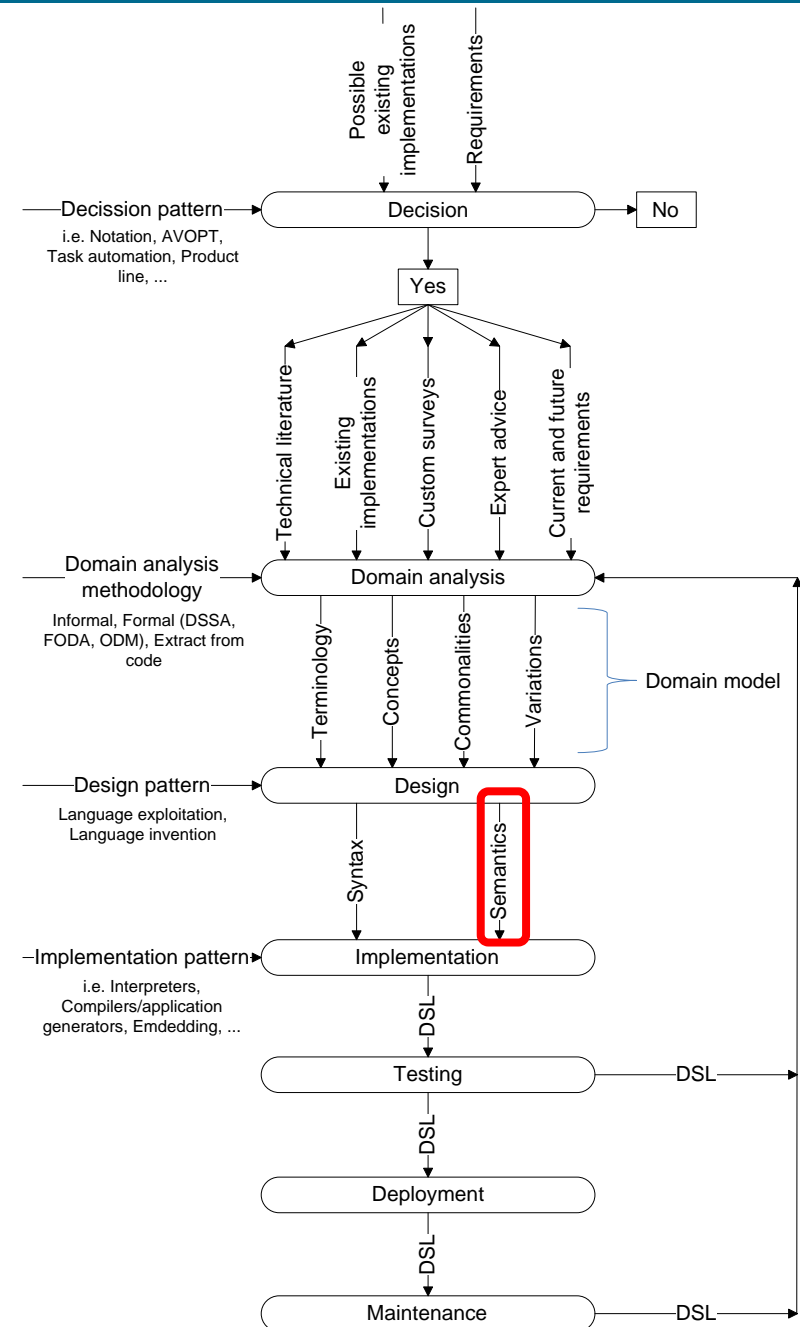
```
<people>
  <person pid="54343" sex="female">
    <first>Mary</first>
    <last>Jones</last>
  </person>
  <person pid="54344" sex="male">
    <first>John</first>
    <last>Jones</last>
  </person>
</people>
```

- Ekvivalentni primer v EBNF:

```
PEOPLE ::= PERSON*
PERSON ::= PID SEX FIRST LAST
PID ::= #number
SEX ::= female | male
FIRST ::= #name
LAST ::= #name
```

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE people [
  <!ELEMENT people (person*)>
  <!ELEMENT person (first , last)>
  <!ATTLIST person
    pid ID #REQUIRED
    sex (male|female) #REQUIRED
  >
  <!ELEMENT first (#PCDATA)>
  <!ELEMENT last (#PCDATA)>
]>
```

# Življenjski cikel





# Načrtovanje DSL

- Domensko-specifični jeziki
  - Sintaksa - EBNF je splošno sprejet za formalen opis sintakse
  - **Semantika** - ni standardne metode; obstajajo številni pristopi: atributna gramatika, aksiomska semantika, **operacijska semantika**, denotacijska semantika, algebraična semantika, akcijska semantika, **translacijska semantika**, ...
- Domensko specifični modelirni jeziki
  - Sintaksa - metamodeli
  - **Semantika** - več alternativ

# Načrtovanje DSL – operacijska semantika

- **Operacijska semantika** določa programske jezike v smislu izvajanja programa na abstraktnih strojih.
- Operacijska semantika je **podana v pravilih**. Vsako pravilo je sestavljeno iz **predpogojev**, ki jih je treba izpolniti, da se uporabi **pravilo**, in **učinke**, ki na nek način spremenijo sedanje stanje.
- Uporaba pravil operacijske semantike daje **nova stanja**, ki postopoma razširjajo in končno **nadomestijo** sintaktične strukture s pomožnimi **konstrukti in vrednostmi**. Končna stanja tega prehodnega sistema vsebujejo samo vrednosti in predstavljajo rezultate specifikacij.

# Načrtovanje DSL – operacijska semantika

- Primer:
  - Abstraktni stroj AM ima konfiguracijo oblike  $\langle c, e, s \rangle$  kjer
    - $c$  je **zaporedje ukazov**, ki jih je treba izvesti
    - $e$  je **sklad**
    - $s$  je **pomnilnik**
  - $e \in \text{Stack}$       $\text{Stack} = (\text{Int} \cup \text{Bool})^*$
  - $s \in \text{State}$       $\text{State} = \text{Var} \rightarrow \text{Int}$
  - $\langle c, e, s \rangle \in \text{Code} \times \text{Stack} \times \text{State}$

# Načrtovanje DSL – operacijska semantika

- Primer **operacijske semantike** za zbirni jezik

- Jezik:

$c ::= \text{inst}^+$

$\text{inst} ::= \text{PUSH } n \mid \text{ADD} \mid \text{MULT} \mid \text{SUB} \mid \text{TRUE} \mid \text{FALSE} \mid \text{EQ} \mid \text{LE} \mid \text{AND} \mid$   
 $\text{NEG} \mid \text{FETCH } x \mid \text{STORE } x \mid \text{NOOP} \mid \text{BRANCH } (c, c) \mid \text{LOOP } (c, c)$

- Program:

$\text{PUSH } 2 : \text{STORE } x :$

$\text{LOOP}(\text{PUSH } 1 : \text{FETCH } x : \text{EQ} : \text{NEG}, \text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x)$

- **Konfiguracija**  $\langle c, e, s \rangle$

- **Semantika ukazov** AM je podana z operacijsko semantiko in je določena s sistemom  $\langle c, e, s \rangle \Rightarrow \langle c', e', s' \rangle$

- Ideja je, da vsak korak transformira konfiguracijo  $\langle c, e, s \rangle \rightarrow \langle c', e', s' \rangle$ .

# Načrtovanje DSL – operacijska semantika

- Semantična pravila

$\langle \text{PUSH } n : c, e, s \rangle \Rightarrow \langle c, n : e, s \rangle$

$\langle \text{ADD} : c, z1 : z2 : e, s \rangle \Rightarrow \langle c, (z1 + z2) : e, s \rangle \quad \text{if } z1, z2 \in \text{Int}$

$\langle \text{MULT} : c, z1 : z2 : e, s \rangle \Rightarrow \langle c, (z1 * z2) : e, s \rangle \quad \text{if } z1, z2 \in \text{Int}$

$\langle \text{SUB} : c, z1 : z2 : e, s \rangle \Rightarrow \langle c, (z1 - z2) : e, s \rangle \quad \text{if } z1, z2 \in \text{Int}$

$\langle \text{TRUE} : c, e, s \rangle \Rightarrow \langle c, \text{true} : e, s \rangle$

$\langle \text{FALSE} : c, e, s \rangle \Rightarrow \langle c, \text{false} : e, s \rangle$

$\langle \text{EQ} : c, z1 : z2 : e, s \rangle \Rightarrow \langle c, (z1 = z2) : e, s \rangle \quad \text{if } z1, z2 \in \text{Int}$

$\langle \text{LE} : c, z1 : z2 : e, s \rangle \Rightarrow \langle c, (z1 \leq z2) : e, s \rangle \quad \text{if } z1, z2 \in \text{Int}$

$\langle \text{AND} : c, t1 : t2 : e, s \rangle \Rightarrow$

$\langle c, \text{true} : e, s \rangle \quad \text{if } t1 = \text{true} \wedge t2 = \text{true}$

$\langle c, \text{false} : e, s \rangle \quad \text{otherwise}$



# Načrtovanje DSL – operacijska semantika

- Semantična pravila

$$\begin{aligned} \langle \text{NEG} : c, t : e, s \rangle &\Rightarrow \langle c, \text{false} : e, s \rangle \text{ if } t = \text{true} \\ &\quad \langle c, \text{true} : e, s \rangle \text{ otherwise} \end{aligned}$$
$$\langle \text{FETCH } x : c, e, s \rangle \Rightarrow \langle c, (s \ x) : e, s \rangle$$
$$\langle \text{STORE } x : c, z : e, s \rangle \Rightarrow \langle c, e, s[x \rightarrow z] \rangle$$
$$\langle \text{NOOP} : c, e, s \rangle \Rightarrow \langle c, e, s \rangle$$
$$\begin{aligned} \langle \text{BRANCH } (c1, c2) : c, t : e, s \rangle &\Rightarrow \langle c1 : c, e, s \rangle \text{ if } t = \text{true} \\ &\quad \langle c2 : c, e, s \rangle \text{ otherwise} \end{aligned}$$
$$\begin{aligned} \langle \text{LOOP } (c1, c2) : c, e, s \rangle &\Rightarrow \\ &\quad \langle c1 : \text{BRANCH } (c2 : \text{LOOP } (c1, c2), \text{NOOP}) : c, e, s \rangle \end{aligned}$$



# Načrtovanje DSL – operacijska semantika

- Kaj je pomen naslednjih ukazov:

- PUSH 2 : STORE x :**

**LOOP(PUSH 1 : FETCH x : EQ : NEG, PUSH 1 : FETCH x : SUB : STORE x)**

- Postopek:

- $\langle \text{PUSH } n : c, e, s \rangle \Rightarrow \langle c, n : e, s \rangle$   
 $\langle \text{PUSH 2 : STORE } x : \text{LOOP(PUSH 1 : FETCH } x : \text{EQ : NEG, PUSH 1 : FETCH } x : \text{SUB : STORE } x), \text{nil}, \{\} \rangle \Rightarrow$
- $\langle \text{STORE } x : c, z : e, s \rangle \Rightarrow \langle c, e, s[x \rightarrow z] \rangle$   
 $\langle \text{STORE } x : \text{LOOP(PUSH 1 : FETCH } x : \text{EQ : NEG, PUSH 1 : FETCH } x : \text{SUB : STORE } x), 2 : \text{nil}, \{\} \rangle \Rightarrow$
- $\langle \text{LOOP}(c1, c2) : c, e, s \rangle \Rightarrow \langle c1 : \text{BRANCH}(c2 : \text{LOOP}(c1, c2), \text{NOOP}) : c, e, s \rangle$   
 $\langle \text{LOOP(PUSH 1 : FETCH } x : \text{EQ : NEG, PUSH 1 : FETCH } x : \text{SUB : STORE } x), \text{nil}, \{x \rightarrow 2\} \rangle \Rightarrow$
- $\langle \text{PUSH } n : c, e, s \rangle \Rightarrow \langle c, n : e, s \rangle$   
 $\langle \text{PUSH 1 : FETCH } x : \text{EQ : NEG: BRANCH(PUSH 1 : FETCH } x : \text{SUB : STORE } x : \text{LOOP(PUSH 1 : FETCH } x : \text{EQ : NEG, PUSH 1 : FETCH } x : \text{SUB : STORE } x), \text{NOOP}), \text{nil}, \{x \rightarrow 2\} \rangle \Rightarrow$
- $\langle \text{FETCH } x : c, e, s \rangle \Rightarrow \langle c, (s \ x) : e, s \rangle$   
 $\langle \text{FETCH } x : \text{EQ : NEG: BRANCH(PUSH 1 : FETCH } x : \text{SUB : STORE } x : \text{LOOP(PUSH 1 : FETCH } x : \text{EQ : NEG, PUSH 1 : FETCH } x : \text{SUB : STORE } x), \text{NOOP}), 1 : \text{nil}, \{x \rightarrow 2\} \rangle \Rightarrow$
- $\langle \text{EQ} : c, z1 : z2 : e, s \rangle \Rightarrow \langle c, (z1 = z2) : e, s \rangle$   
 $\langle \text{EQ : NEG: BRANCH(PUSH 1 : FETCH } x : \text{SUB : STORE } x : \text{LOOP(PUSH 1 : FETCH } x : \text{EQ : NEG, PUSH 1 : FETCH } x : \text{SUB : STORE } x), \text{NOOP}), 2 : 1 : \text{nil}, \{x \rightarrow 2\} \rangle \Rightarrow$
- $\langle \text{NEG} : c, t : e, s \rangle \Rightarrow \langle c, \text{false} : e, s \rangle \text{ if } t = \text{true}$   
 $\langle c, \text{true} : e, s \rangle \text{ otherwise}$   
 $\langle \text{NEG: BRANCH(PUSH 1 : FETCH } x : \text{SUB : STORE } x : \text{LOOP(PUSH 1 : FETCH } x : \text{EQ : NEG, PUSH 1 : FETCH } x : \text{SUB : STORE } x), \text{NOOP}), \text{false} : \text{nil}, \{x \rightarrow 2\} \rangle \Rightarrow$



# Načrtovanje DSL – operacijska semantika

## *Domača naloga!*

$\langle \text{BRANCH (PUSH 1 : FETCH x : SUB : STORE x : LOOP(PUSH 1 : FETCH x : EQ : NEG, PUSH 1 : FETCH x : SUB : STORE x), NOOP), true : nil, \{x \rightarrow 2\}} \rangle \Rightarrow$

$\langle \text{PUSH 1 : FETCH x : SUB : STORE x : LOOP(PUSH 1 : FETCH x : EQ : NEG, PUSH 1 : FETCH x : SUB : STORE x), nil, \{x \rightarrow 2\}} \rangle \Rightarrow$

$\langle \text{FETCH x : SUB : STORE x : LOOP(PUSH 1 : FETCH x : EQ : NEG, PUSH 1 : FETCH x : SUB : STORE x), 1: nil, \{x \rightarrow 2\}} \rangle \Rightarrow$

$\langle \text{SUB : STORE x : LOOP(PUSH 1 : FETCH x : EQ : NEG, PUSH 1 : FETCH x : SUB : STORE x), 2: 1 : nil, \{x \rightarrow 2\}} \rangle \Rightarrow$

$\langle \text{STORE x : LOOP(PUSH 1 : FETCH x : EQ : NEG, PUSH 1 : FETCH x : SUB : STORE x), 1 : nil, \{x \rightarrow 2\}} \rangle \Rightarrow$

$\langle \text{LOOP(PUSH 1 : FETCH x : EQ : NEG, PUSH 1 : FETCH x : SUB : STORE x), nil, \{x \rightarrow 1\}} \rangle \Rightarrow$



# Načrtovanje DSL – operacijska semantika

## *Domača naloga!*

$\langle \text{PUSH } 1 : \text{FETCH } x : \text{EQ} : \text{NEG} : \text{BRANCH}(\text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x : \text{LOOP}(\text{PUSH } 1 : \text{FETCH } x : \text{EQ} : \text{NEG}, \text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x), \text{NOOP}), \text{nil}, \{x \rightarrow 1\} \rangle \Rightarrow$

$\langle \text{FETCH } x : \text{EQ} : \text{NEG} : \text{BRANCH}(\text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x : \text{LOOP}(\text{PUSH } 1 : \text{FETCH } x : \text{EQ} : \text{NEG}, \text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x), \text{NOOP}), 1 : \text{nil}, \{x \rightarrow 1\} \rangle \Rightarrow$

$\langle \text{EQ} : \text{NEG} : \text{BRANCH}(\text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x : \text{LOOP}(\text{PUSH } 1 : \text{FETCH } x : \text{EQ} : \text{NEG}, \text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x), \text{NOOP}), 1 : 1 : \text{nil}, \{x \rightarrow 1\} \rangle \Rightarrow$

$\langle \text{NEG} : \text{BRANCH}(\text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x : \text{LOOP}(\text{PUSH } 1 : \text{FETCH } x : \text{EQ} : \text{NEG}, \text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x), \text{NOOP}), \text{true} : \text{nil}, \{x \rightarrow 1\} \rangle \Rightarrow$

$\langle \text{BRANCH}(\text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x : \text{LOOP}(\text{PUSH } 1 : \text{FETCH } x : \text{EQ} : \text{NEG}, \text{PUSH } 1 : \text{FETCH } x : \text{SUB} : \text{STORE } x), \text{NOOP}), \text{false} : \text{nil}, \{x \rightarrow 1\} \rangle \Rightarrow$

$\langle \text{NOOP}, \text{nil}, \{x \rightarrow 1\} \rangle \Rightarrow$

$\langle \epsilon, \text{nil}, \{x \rightarrow 1\} \rangle$

*$\langle \text{NOOP} : c, e, s \rangle \Rightarrow \langle c, e, s \rangle$*

- Program v zbirniku je ustrezal naslednjemu zapisu v višjenivojskih jezikih:  
 $x := 2; \text{while not } (x = 1) \text{ do } (x := x - 1)$



# Načrtovanje DSL – translacijska semantika

- **Translacijska semantika**

- Bistvo translacijske semantike je **prevajanje konstruktov** iz enega jezika v drug jezik, ki že ima semantiko.

- **Tokrat obraten primer:**

- Jezik WHILE v zbirni jezik
- Program:

**CS** [  $y := 1$ ; **while not** ( $x = 1$ ) **do** ( $y := y * x$ ;  $x := x - 1$ ) ]

- Semantična pravila:

- **Aritmetični izrazi**

**CA** : Aexp  $\rightarrow$  Code

**CA** [n] = PUSH n

**CA** [x] = FETCH x

**CA** [ $a1 + a2$ ] = **CA** [ $a2$ ] : **CA** [ $a1$ ] : ADD

**CA** [ $a1 * a2$ ] = **CA** [ $a2$ ] : **CA** [ $a1$ ] : MULT

**CA** [ $a1 - a2$ ] = **CA** [ $a2$ ] : **CA** [ $a1$ ] : SUB

# Načrtovanje DSL – translacijska semantika

- Semantična pravila
  - **Logični izraz**

**$CB$**  : Bexp  $\rightarrow$  Code

**$CB$**  [true] = TRUE

**$CB$**  [false] = FALSE

**$CB$**  [a1 = a2] =  **$CA$**  [a2] :  **$CA$**  [a1] : EQ

**$CB$**  [a1 <= a2] =  **$CA$**  [a2] :  **$CA$**  [a1] : LE

**$CB$**  [not b] =  **$CB$**  [b] : NEG

**$CB$**  [b1 and b2] =  **$CB$**  [b2] :  **$CB$**  [b1] : AND

# Načrtovanje DSL – translacijska semantika

- Semantična pravila
  - Ukazi

**CS**:  $\text{Stm} \rightarrow \text{Code}$

**CS** [ $x := a$ ] = **CA** [ $a$ ] : STORE  $x$

**CS** [**skip**] = NOOP

**CS** [ $S1; S2$ ] = **CS** [ $S1$ ] : **CS** [ $S2$ ]

**CS** [**if**  $b$  **then**  $S1$  **else**  $S2$ ] = **CB** [ $b$ ] : BRANCH(**CS** [ $S1$ ], **CS** [ $S2$ ])

**CS** [**while**  $b$  **do**  $S$ ] = LOOP(**CB** [ $b$ ], **CS** [ $S$ ])

# Načrtovanje DSL – translacijska semantika

- Program v jeziku While

**CS** [  $x := 2$ ; **while not** ( $x = 1$ ) **do** ( $y := y * x$ ;  $x := x - 1$ ) ]

# Načrtovanje DSL – translacijska semantika

$$CS [S1; S2] = CS [S1] : CS [S2]$$

$CS [x := 2; \text{while not } (x = 1) \text{ do } (y := y * x; x := x - 1)] =$

$CS [x := 2] : CS [\text{while not } (x = 1) \text{ do } (y := y * x; x := x - 1)] =$



# Načrtovanje DSL – translacijska semantika

$CS [x := 2; \text{while not } (x = 1) \text{ do } (y := y * x; x := x - 1)] =$   
 $CS [x := 2] : CS [\text{while not } (x = 1) \text{ do } (y := y * x; x := x - 1)] =$   
 $CA [2] : \text{STORE } x \quad \text{LOOP } (CB [\text{not } (x = 1)], CS [y := y * x; x := x - 1]) =$

$CS [x := a] = CA [a] : \text{STORE } x$

$CS [\text{while } b \text{ do } S] = \text{LOOP}(CB [b], CS [S])$



# Načrtovanje DSL – translacijska semantika

$CS [x := 2; \text{while not } (x = 1) \text{ do } (y := y * x; x := x - 1)] =$   
 $CS [x := 2] : CS [\text{while not } (x = 1) \text{ do } (y := y * x; x := x - 1)] =$   
 $CA [2] : \text{STORE } y : \text{LOOP } (CB [\text{not } (x = 1)], CS [y := y * x; x := x - 1]) =$   
 $\text{PUSH } 2 : \text{STORE } x : \text{LOOP } (CB [x = 1] : \text{NEG}, CS [y := y * x] : CS [x := x - 1]) =$

$CA [n] = \text{PUSH } n$

$CB [\text{not } b] = CB [b] : \text{NEG}$



# Načrtovanje DSL – translacijska semantika

$CS [x := 2; \text{while not } (x = 1) \text{ do } (y := y * x; x := x - 1)] =$   
 $CS [x := 2] : CS [\text{while not } (x = 1) \text{ do } (y := y * x; x := x - 1)] =$   
 $CA [2] : STORE\ x : LOOP (CB [\text{not } (x = 1)], CS [y := y * x; x := x - 1]) =$   
 $PUSH\ 2 : STORE\ x : LOOP (CB [x = 1] : NEG, CS [y := y * x] : CS [x := x - 1]) =$   
 $PUSH\ 2 : STORE\ x : LOOP (CA [1] : CA [x] : EQ : NEG,$   
 $\quad CA [y * x] : STORE\ y : CA [x - 1] : STORE\ x) =$   
 $PUSH\ 2 : STORE\ x : LOOP (PUSH\ 1 : FETCH\ x : EQ : NEG,$   
 $\quad CA [x] : CA [y] : MULT : STORE\ y : CA [1] : CA [x] : SUB : STORE\ x) =$

- Končni rezultat translacijske semantike jezika While:

$PUSH\ 2 : STORE\ x :$   
 $LOOP (PUSH\ 1 : FETCH\ x : EQ : NEG,$   
 $\quad FETCH\ x : FETCH\ y : MULT : STORE\ y :$   
 $\quad PUSH\ 1 : FETCH\ x : SUB : STORE\ x)$

# Vaje in formalna semantika

- Translacijska semantika
  - Jezik ReakylIDSL

# Vaje

- Neobvezni nalogi
  - Naloga 7.1 (neobvezna)
    - Za jezik ReakylIDSL iz naloge 5.1 dodajte 3 testne primerke, ki testirajo celotni program in posamezne dele DSL programa.
  - Naloga 7.2 (neobvezna)
    - Za jezik ReakylIDSL iz naloge 5.1 implementirajte projekcijski urejevalnik v obliki tabele (ali podobno projekcijo).

# VPRAŠANJA

