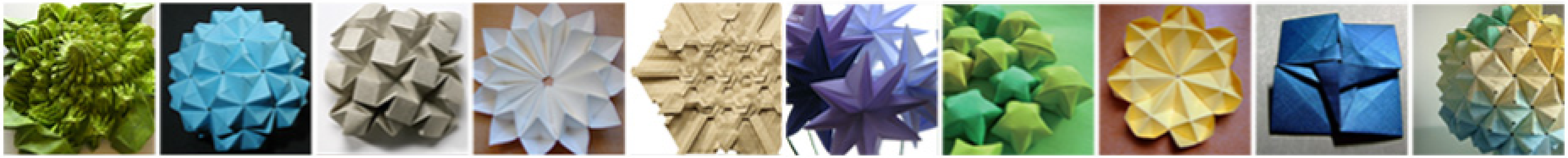


DSM TP 2017

8th International Summer School
on Domain-Specific Modeling
Theory and Practice

Montreal, Canada
10-14 July 2017

Model Transformation

Eugene Syriani

with a little help from Hans Vangheluwe

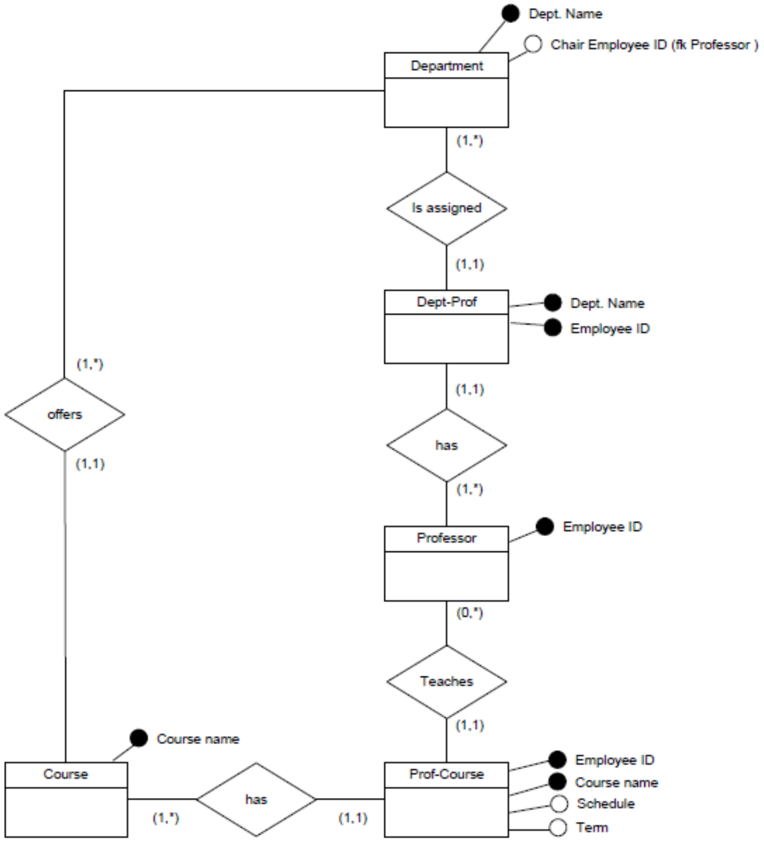
Software
Modeling &
Simulation

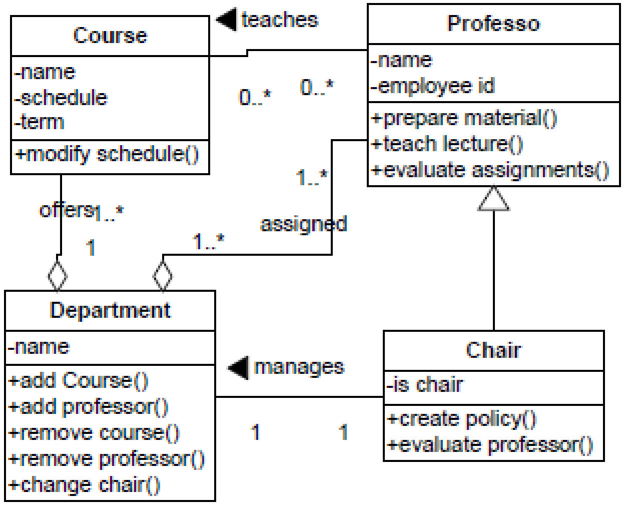
geodes
Département d'**informatique et de recherche opérationnelle**

Université de Montréal

Motivation

Suppose I ask you to provide a software that converts any E-R diagram into a UML class diagram, how would you achieve that?





The “programming” solution

- Write a program that takes as input a .ER file and outputs a .UML file
- What are the issues?
  - What if the ER file is a diagram? in XML format? Probably end up limiting input from a specific tool only
  - Similarly in UML, should I output a diagram (in Dia or Visio)? In XMI? In code (Java, C#)?
  - How do I organize my program?
    - Requires knowledge from both domains
    - Need a loader (from input file)
    - Need some kind of visitor to traverse the model, probably graph-like data structure
    - Need to encode a “transformer”
    - Need to develop a UML printer
- Not an easy task after all…

The “modeling” way

1. Describe a meta-model of ER
   - Define concepts and concrete visual syntax
   - Generate an editor
2. Describe a meta-model of UML
3. Define a transformation T: $MM_{ER} \rightarrow MM_{UML}$
   - This is done in the form of rules with pre/post-conditions
     - describes “what to transform” instead of “how to transform”

- Transformation model is executed (compiled or interpreted) to produce the result
- Some model transformation languages give you a bi-directional solution for free!

What’s the difference?

- Typically encounter the **same problems** in modeling as in programing solutions
- The difference is that you can **find** the problems more easily, **fix** them very quickly and **re-deploy** the solution automatically
- Changed the level of **abstraction** to reduce **accidental complexity**
- Developers not required to be programmers:
  “He, who expresses the problem shall specify its solution”

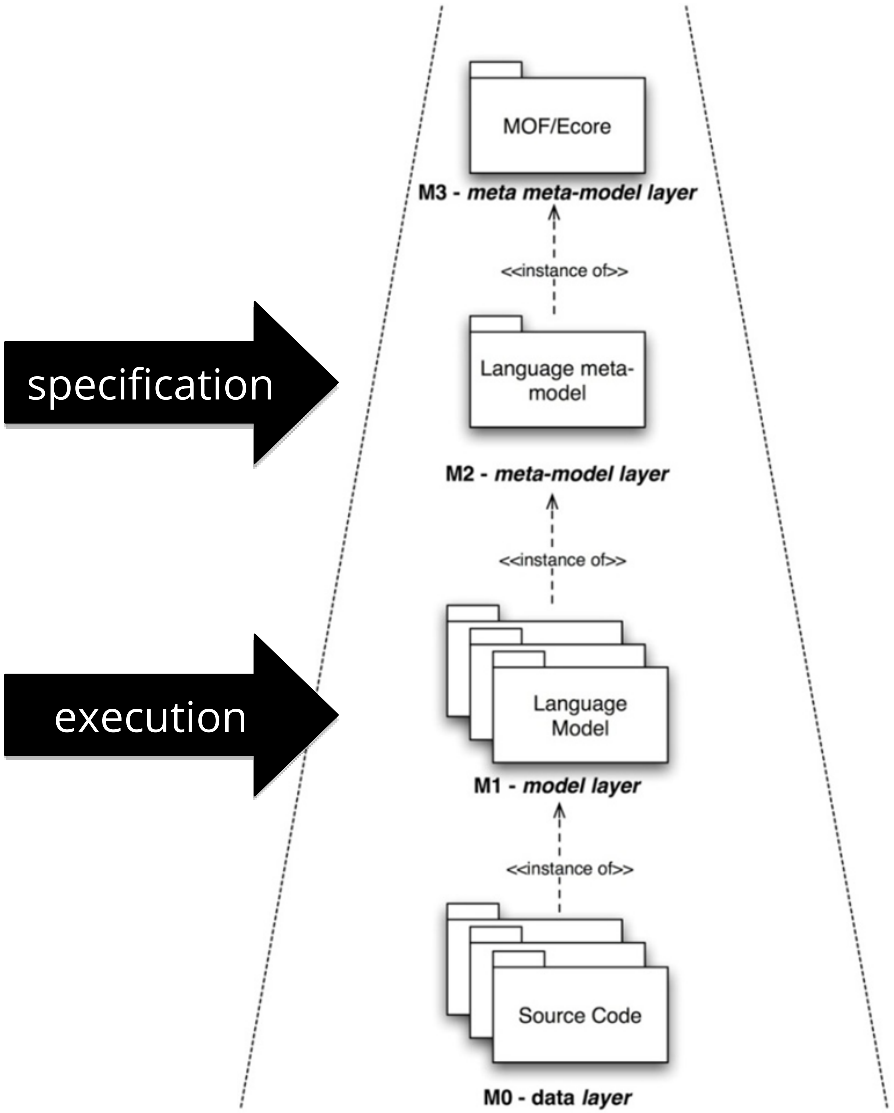
What is a model transformation?

Definition

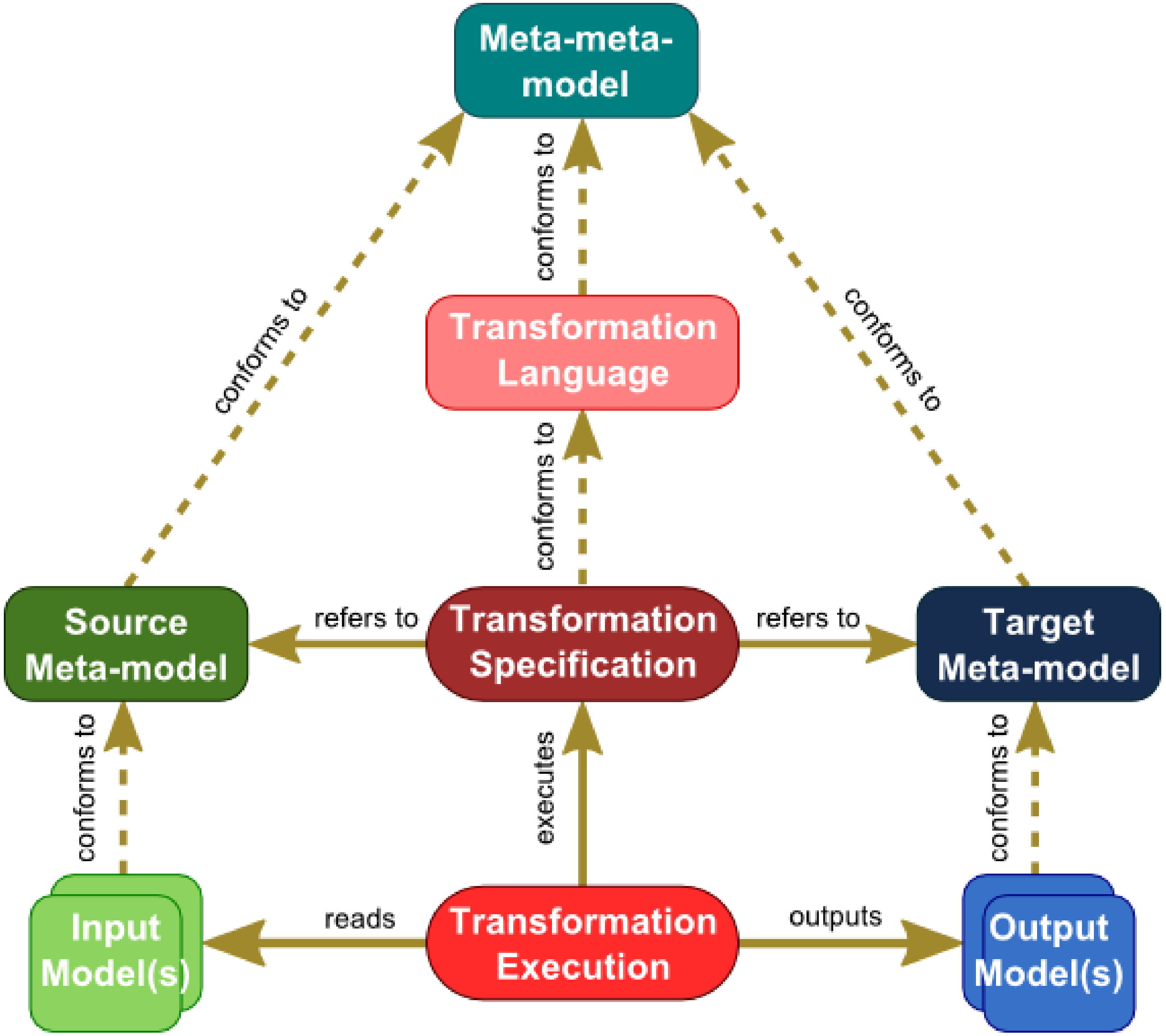
A model transformation is the **automatic** manipulation of **input** models to produce **output** models, that conforms to a **specification** and has a specific **intent**.

L. Lúcio, M. Amrani, J. Dingel, L. Lambers, R. Salay, G. Selim, E. Syriani & M. Wimmer. Model transformation intents and their properties. *Software & Systems Modeling*: 15(3), pp. 647-684 (2016).

Where should MT be specified and executed?



Terminology



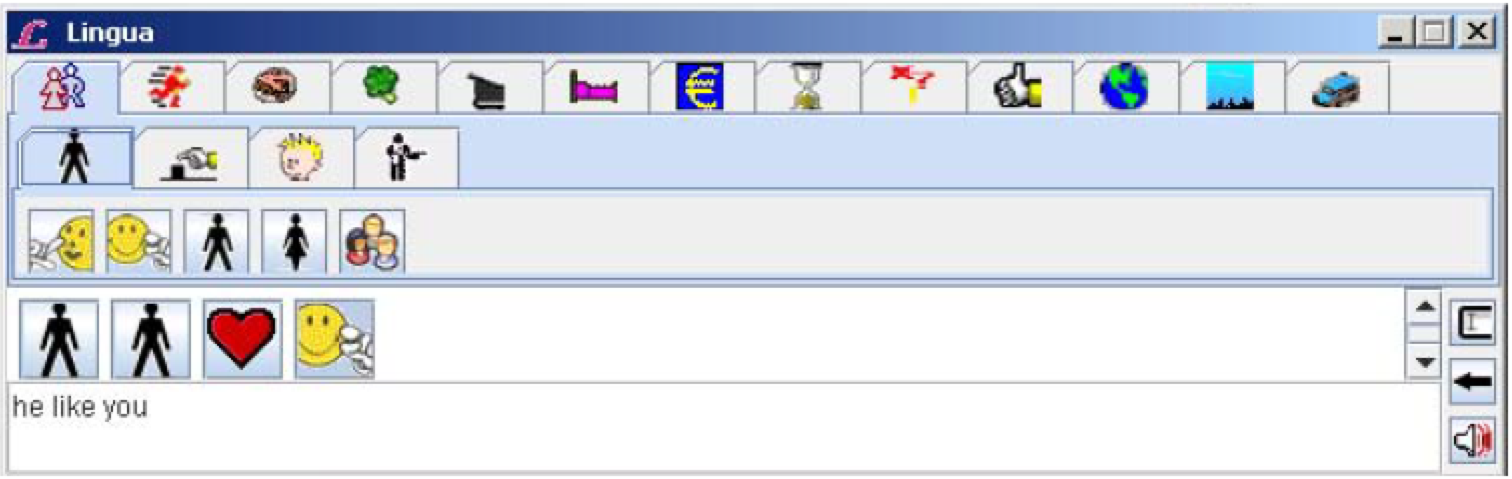
(= Source)

(= Input)

Data structures to transform

Sequence

- Linear sequence of symbols
  - Data: symbol
  - Connector: successor
- Example: string, iconic sentence
- Manipulation through **string rewriting**



he like you

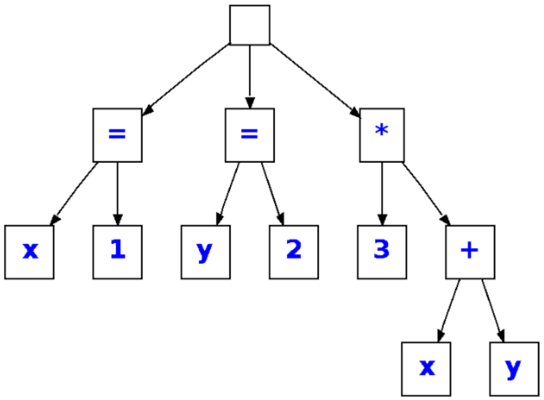
String rewriting

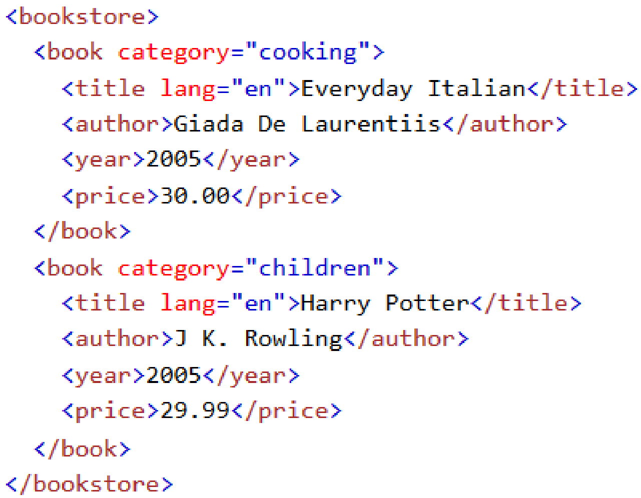
- Model transformation paradigm: **regular expression**
  - Stream Editor (sed)
- Model “`Hello world`”
- Metamodel `.*`
- Model transformation `s/(.*)\s([a-z]*)/\2\t\1/g`
- Transformation language is rule-based, regular expression
  - s/ LHS to be matched
  - / RHS /g to rewrite, with labels

Data structures to transform

Tree

- Acyclic connected simple graph
  - Data: nodes N
  - Connector: edges $E \subseteq N \times N : |E| = |N| - 1$
- Example: Abstract syntax tree of a program, XML
- Manipulation through **tree rewriting**





```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Tree rewriting

- Model transformation paradigm: **parser**
  - Gentle compiler construction system
- Model
- Metamodel

```
expression ::= expression "+" expr2 | expr2
expr2 ::= expr2 "*" expr2 | Number
```

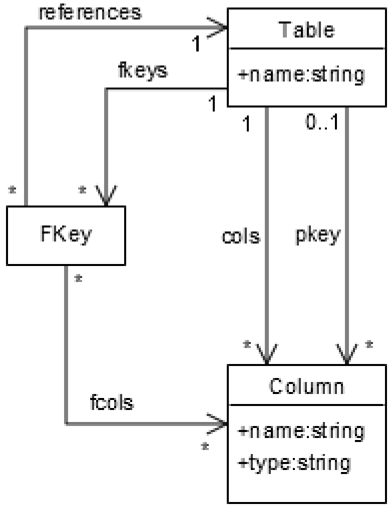
- Model transformation
  - Transformation language is **term rewriting** with production rules

```
root expr(->X)
nonterm expr(->Expr)
  rule expr(->X): expr2(->X)
  rule expr(->add(X,Y)): expr(->X) "+" expr2(->Y)
nonterm expr2(->Expr)
  rule expr2(->mult(X,Y)): expr2(->X) "*" expr2(->Y)
  rule expr2(->num(X)): Number(->X)
token Number(->INT)
```

Data structures to transform

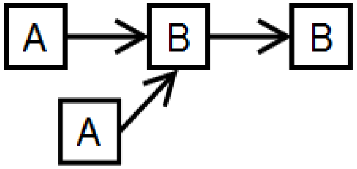
Graph

- Typed attributed graphs, hypergraphs, multigraphs
  - Data: nodes N
  - Connector: edges $E \subseteq N \times N$
- Example: Class diagrams, Statecharts
- Manipulation through **graph transformation**

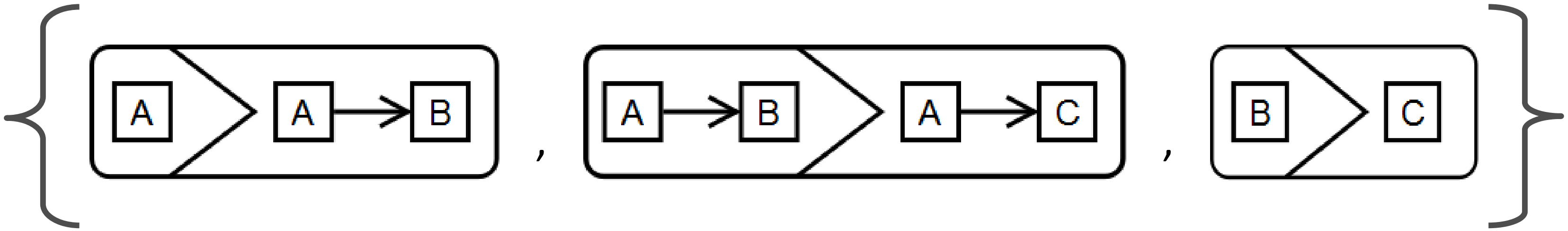


Graph transformation

- Model transformation paradigm: **algebraic graph transformation**
  - T-Core
- Model



- Model transformation
  - Rule-based Graph Transformation vs. Graph Grammar



Transformations for language engineering

- Abstract syntax to abstract syntax
  - Tree rewriting
  - Graph transformation (Model-to-model and simulation)
- Abstract syntax to concrete syntax (textual)
  - Model-to-text transformation
- Concrete syntax to concrete syntax (textual)
  - String rewriting
- Concrete syntax to abstract syntax
  - Tree rewriting (Parsing)

Two main transformation types in MDE

- Model-to-text
  - **Visitor-based**: traverse the model in an object-oriented framework
  - **Template-based**: target syntax with meta-code to access source model
- Model-to-Model
  - **Direct manipulation**: access to the API of M3 and modify the models directly
  - **Operational**: similar to direct manipulation but at the model-level (OCL)
  - **Rule-based**
    - **Graph transformation**: implements directly the theory of graph transformation, where models are represented as typed, attributed, labelled, graphs in category theory. It is a declarative way of describing operations on models.
    - **Relational**: declarative, describing mathematical relations. It define constraints relating source and target elements that need to be solved. They are naturally multi-directional, but in-place transformation is harder to achieve

K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal:* 45(3), 621-645 (2006).

Typical use cases of model transformation

Model transformation intent classification

Refinement

- Refinement
- Synthesis
  - Serialization

Abstraction

- Abstraction
- Reverse Engineering
- Restrictive Query
- Approximation

Semantic Definition

- Translational Semantics
- Simulation

Language Translation

- Translation
- Migration

Constraint Satisfaction

- Model Finding
- Model Generation

Analysis

Editing

- Model Editing
- Optimization
- Model Refactoring
- Normalization
  - Canonicalization

Model Visualization

- Animation
- Rendering
- Parsing

Model Composition

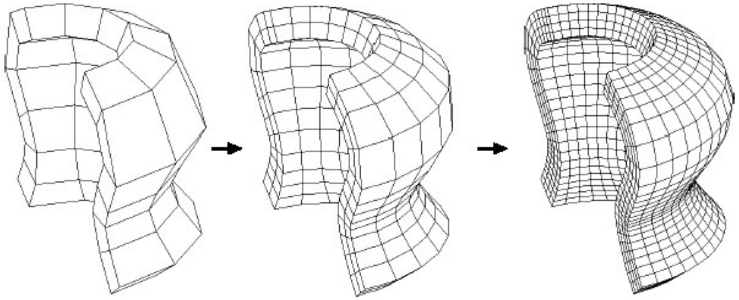
- Model Merging
- Model Matching
- Model Synchronization

L. Lúcio, M. Amrani, J. Dingel, L. Lambers, R. Salay, G. Selim, E. Syriani & M. Wimmer. Model transformation intents and their properties. *Software & Systems Modeling*: 15(3), pp. 647-684 (2016).

Refinement category

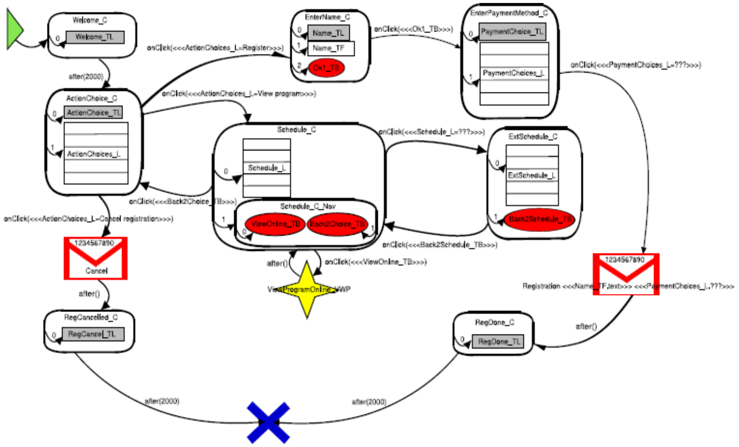
Groups intents that produce a more precise model by reducing design choices and ambiguities with respect to a target platform.

- Refinement (model-to-model)
- Synthesis (model-to-text)

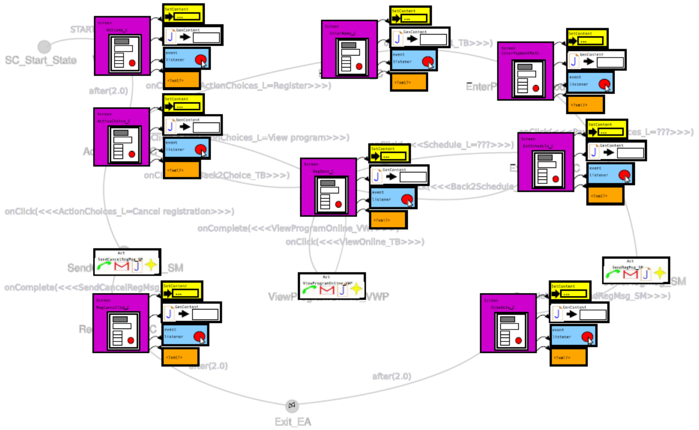


Refinement

- Transform from a higher level specification (e.g., PIM) to a lower level description (e.g., PSM)
- Adds information to models
- M_1 refines M_2 if M_1 can answer all questions that M_2 can for a specific purpose



PhoneApps DSM of a conference registration mobile application


SC_Start_State
after(2.0)
ActionChoices_L=Register>>>)
Choices_L=View program>>>)
<<<Schedule_L=???>>>)
ack2Choice_TB>>>)
<<<Back2Schedule
onClick(<<<ActionChoices_L=Cancel registration>>>)
onComplete(<<<ViewProgramOnline_VW
onClick(<<<ViewOnline_TB>>>)
onComplete(<<<SendCancelRegMsg
dRegMsg_SM>>>)
VWP
SM
after(2.0)
after(2.0)
Exit_EA

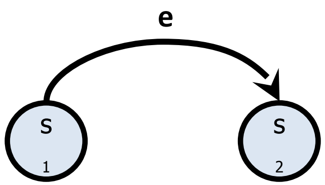
Representation of the model in AndroidAppScreens

PhoneApps DSL To Android Activities

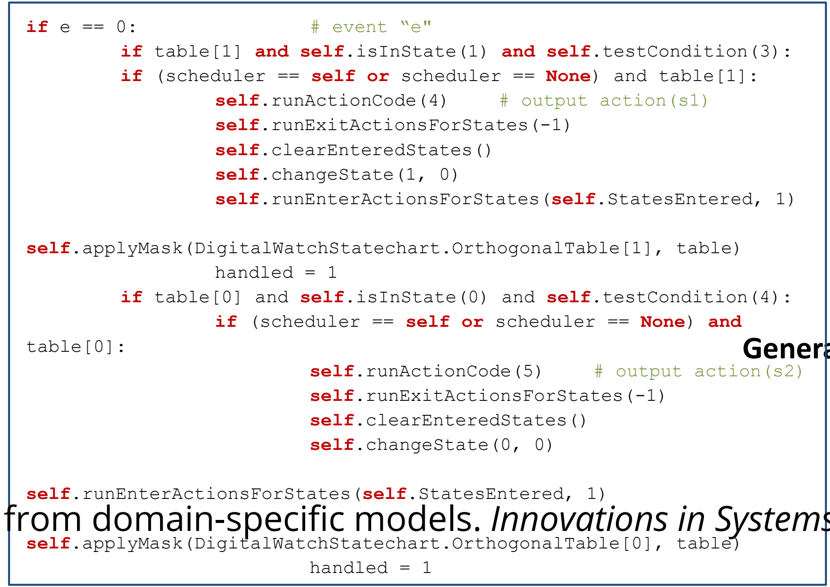
J. Denil, A. Cicchetti, M. Biehl, P. De Meulenaere, R. Eramo, S. Demeyer, & Vangheluwe, H. Automatic deployment space exploration using refinement transformations. *Electronic Communications of the EASST*: 50 (2012).

Synthesis

- Refinement where the output is an **executable artifact** expressed in a well-defined language format
  - Typically textual
- **Model-to-code generation**: transformation that produces source code in a target programming language
- Refinement often precedes synthesis



Statecharts model



```
if e == 0:                    # event "e"
        if table[1] and self.isInState(1) and self.testCondition(3):
        if (scheduler == self or scheduler == None) and table[1]:
                self.runActionCode(4)     # output action(s1)
                self.runExitActionsForStates(-1)
                self.clearEnteredStates()
                self.changeState(1, 0)
                self.runEnterActionsForStates(self.StatesEntered, 1)

self.applyMask(DigitalWatchStatechart.OrthogonalTable[1], table)
                handled = 1
        if table[0] and self.isInState(0) and self.testCondition(4):
                if (scheduler == self or scheduler == None) and
table[0]:
                        self.runActionCode(5)     # output action(s2)
                        self.runExitActionsForStates(-1)
                        self.clearEnteredStates()
                        self.changeState(0, 0)

self.runEnterActionsForStates(self.StatesEntered, 1)

self.applyMask(DigitalWatchStatechart.OrthogonalTable[0], table)
                        handled = 1
```

Generated Python code

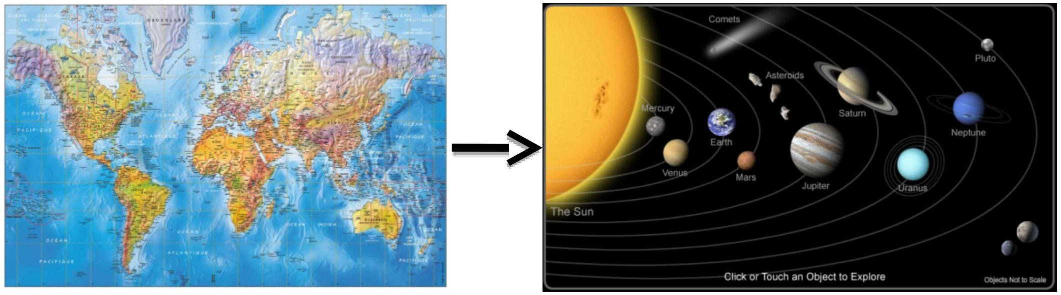
Statecharts to Python Compiler

M. Raphael & H. Vangheluwe. Modular artifact synthesis from domain-specific models. *Innovations in Systems and Software Engineering*: 8(1) pp. 65-77 (2012).

Abstraction category

Inverse of refinement category.
Groups intents where some information of a model is aggregated or discarded to simplify the model and emphasize specific information.

- Abstraction (model-to-model)
- Query
- *Reverse Engineering*
- *Approximation*



Abstraction

- Inverse of refinement
- Implication of satisfaction of properties
- If M_1 refines M_2 then M_2 is an abstraction of M_1

Example:

“*Find all actors who played together in at least 3 movies and assign the average rating to each clique*” outputs a view of a model representing a subset of IMDB represented as a graph composed of strongly connected components with the ratings aggregating individual ratings.

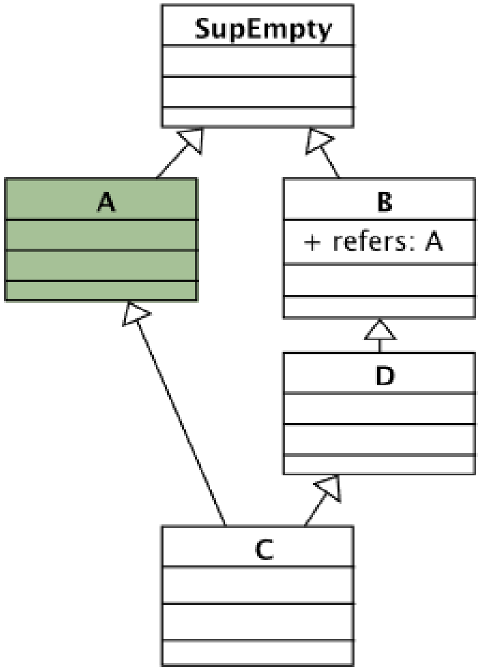
T. Horn et al. The TTC 2014 Movie Database Case. *Transformation Tool Contest* 2014.

Query

- A query requests some information about a model and returns that information in the form of a proper sub-model or a view
  - Projection of a sub-set of of the properties of M
  - View of a model that is not a sub-model, but an aggregation of some of its information is also a abstraction
- Example: “*Get all the leaves of a tree*”
- Tool support: EMF-IncQuery

Querying models with IncQuery

```
pattern superClass(sub : Class, sup : Class) {
  Generalization.specific(gen, sub);
  Generalization(gen);
  Generalization.general(gen, sup);
}

pattern hasOperation(cl : Class, op : Operation) {
  Class.ownedOperation(cl, op);
} or {
  find superClass+(cl, owner);
  Class.ownedOperation(owner, op);
}

pattern emptyClass(cl : Class) {
  neg find hasOperation(cl, _op);
  neg find hasProperty(cl, _pr);
  Class.name(cl, n);
  check(!(n.endsWith("Empty")));
}
```



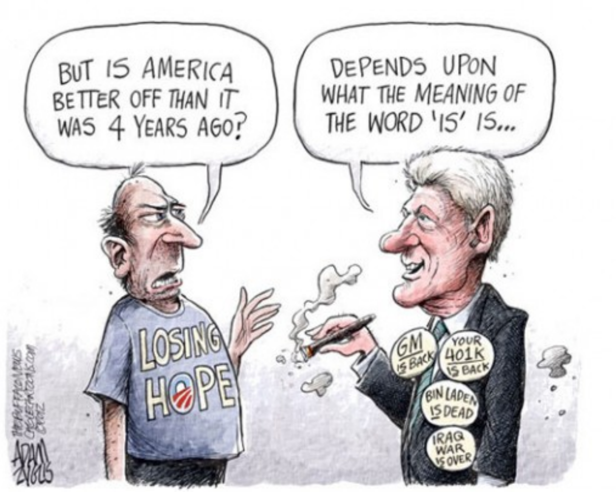
Z. Ujhelyi, et al. EMF-IncQuery: An integrated development environment for live model queries. *Science of Computer Programming:* 98, 80-99 (2015).

Semantic Definition category

Groups intents whose purpose is to define the semantics of a modeling language.

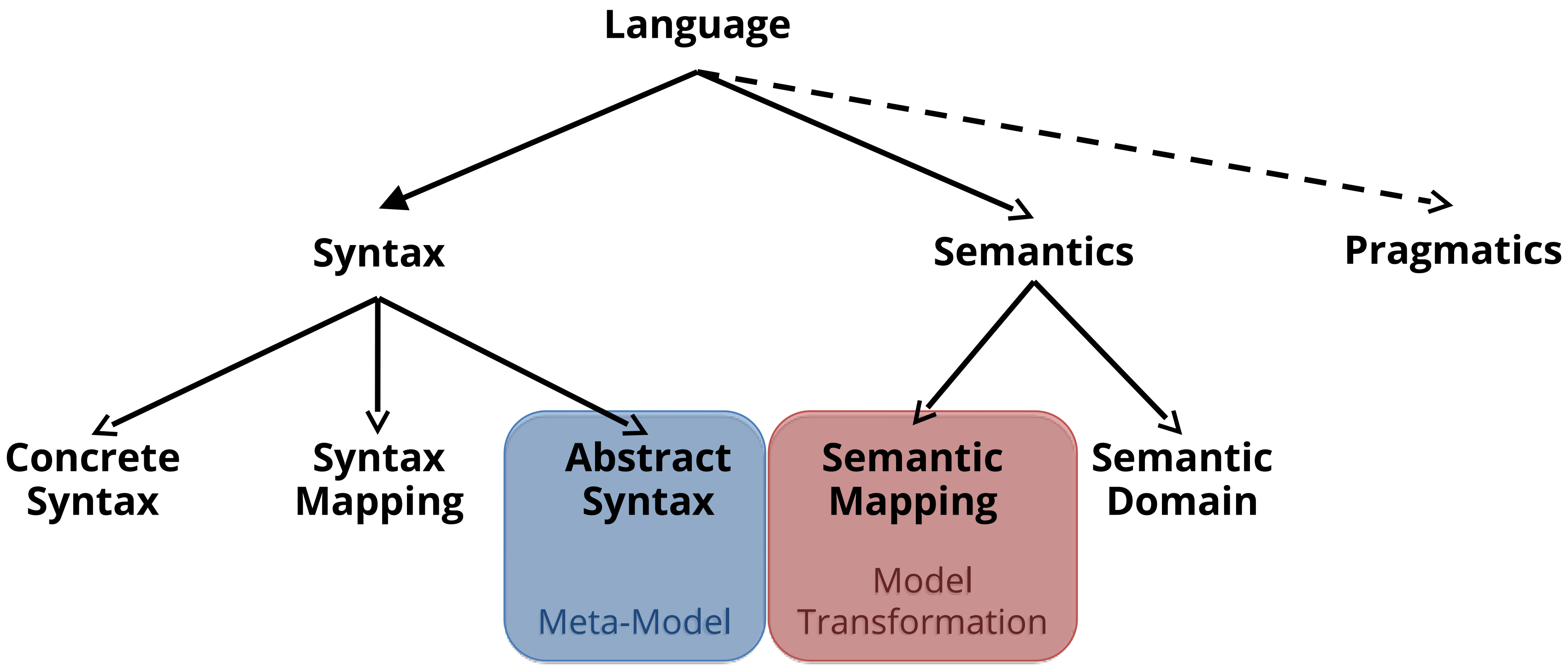
- Translational Semantics (model-to-model)
- Operational Semantics

  (simulation by graph transformation)


BUT IS AMERICA BETTER OFF THAN IT WAS 4 YEARS AGO?
DEPENDS UPON WHAT THE MEANING OF THE WORD 'IS' IS...
LOSING HOPE
GM IS BACK
YOUR 401K IS BACK
BIN LADEN IS DEAD
IRAQ WAR IS OVER

Translational Semantics

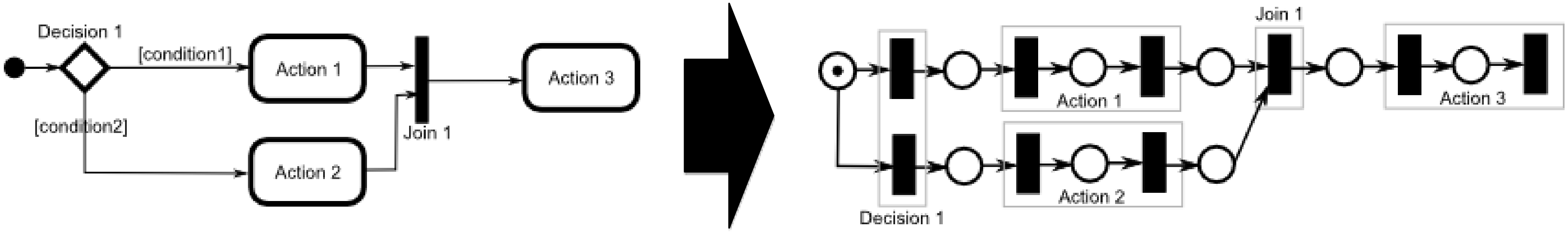
- Gives the **meaning** of a model in a source language in terms of the concepts of another target language
- Typically used to capture the semantics of **new DSLs**



Translational Semantics

- Simulink Block Diagram’s semantics expressed as Ordinary Differential Equations

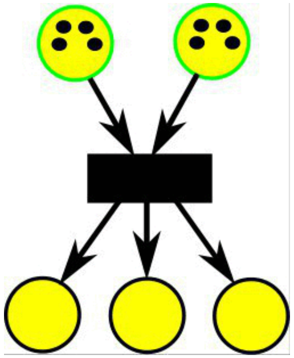
- UML activity diagrams semantics expressed as Petri nets



E. Syriani and H. Ergin. Operational Semantics of UML Activity Diagram: An Application in Project Management. *RE 2012 Workshops*: pp. 1-8, IEEE (2012)

Simulation

- Defines the **operational semantics** of a modeling language that updates the state of the system modeled
- The source and target meta-models are **identical**
- The target model is an **“updated”** version of the source model: no new model is created
- Simulation updates the abstract syntax, which may trigger modifications in the concrete syntax



Petri nets simulator

T. Kühne, G. Mezei, E. Syriani, H. Vangheluwe and M. Wimmer. Explicit Transformation Modeling. *MODELS 2009 Workshops*, LNCS: 6002, pp. 240-255, Springer (2010).

QUESTION

Generate JavaDocs from a class diagram.

Input: Class diagram

Output: HTML document

- Synthesis

Abstraction
Analysis
Animation
Approximation
Canonicalization
Migration
Model Editing
Model Finding
Model Generation
Model Matching
Model Merging
Model Refactoring
Model Synchronization
Normalization
Optimization
Parsing
Refinement
Rendering
Query
Reverse Engineering
Serialization
Simulation
Synthesis
Translation
Translational Semantics

QUESTION

Augment a class diagram by adding navigability, role names, attribute types, method return and parameter types.

Input: Class diagram

Output: Class diagram

- Refinement

Abstraction
Analysis
Animation
Approximation
Canonicalization
Migration
Model Editing
Model Finding
Model Generation
Model Matching
Model Merging
Model Refactoring
Model Synchronization
Normalization
Optimization
Parsing
Refinement
Rendering
Query
Reverse Engineering
Serialization
Simulation
Synthesis
Translation
Translational Semantics

QUESTION

Define the actions performed by a traffic light to transition from one state to another.

Input: Traffic light model

Output: Traffic light model

- Simulation

Abstraction
Analysis
Animation
Approximation
Canonicalization
Migration
Model Editing
Model Finding
Model Generation
Model Matching
Model Merging
Model Refactoring
Model Synchronization
Normalization
Optimization
Parsing
Refinement
Rendering
Query
Reverse Engineering
Serialization
Simulation
Synthesis
Translation
Translational Semantics

QUESTION

Extract the classes with no super-class from a class diagram.

Input: Class diagram

Output: Class diagram

- ➢ Query

Abstraction
Analysis
Animation
Approximation
Canonicalization
Migration
Model Editing
Model Finding
Model Generation
Model Matching
Model Merging
Model Refactoring
Model Synchronization
Normalization
Optimization
Parsing
Refinement
Rendering
Query
Reverse Engineering
Serialization
Simulation
Synthesis
Translation
Translational Semantics

QUESTION

Map a custom DSML for stop watches into a Statecharts model in order to define its behavior.

Input: Watch DSM

Output: Statechart

- Translational Semantics

Abstraction
Analysis
Animation
Approximation
Canonicalization
Migration
Model Editing
Model Finding
Model Generation
Model Matching
Model Merging
Model Refactoring
Model Synchronization
Normalization
Optimization
Parsing
Refinement
Rendering
Query
Reverse Engineering
Serialization
Simulation
Synthesis
Translation
Translational Semantics

Vocabulary

- Relationship between source & target meta-models
  - **Endogenous**: Source meta-model = Target meta-model
  - **Exogenous**: Source meta-model ≠ Target meta-model
- Relationship between source & target models
  - **In-place**: Transformation executed within the same model
  - **Out-place**: Transformation produces a different model

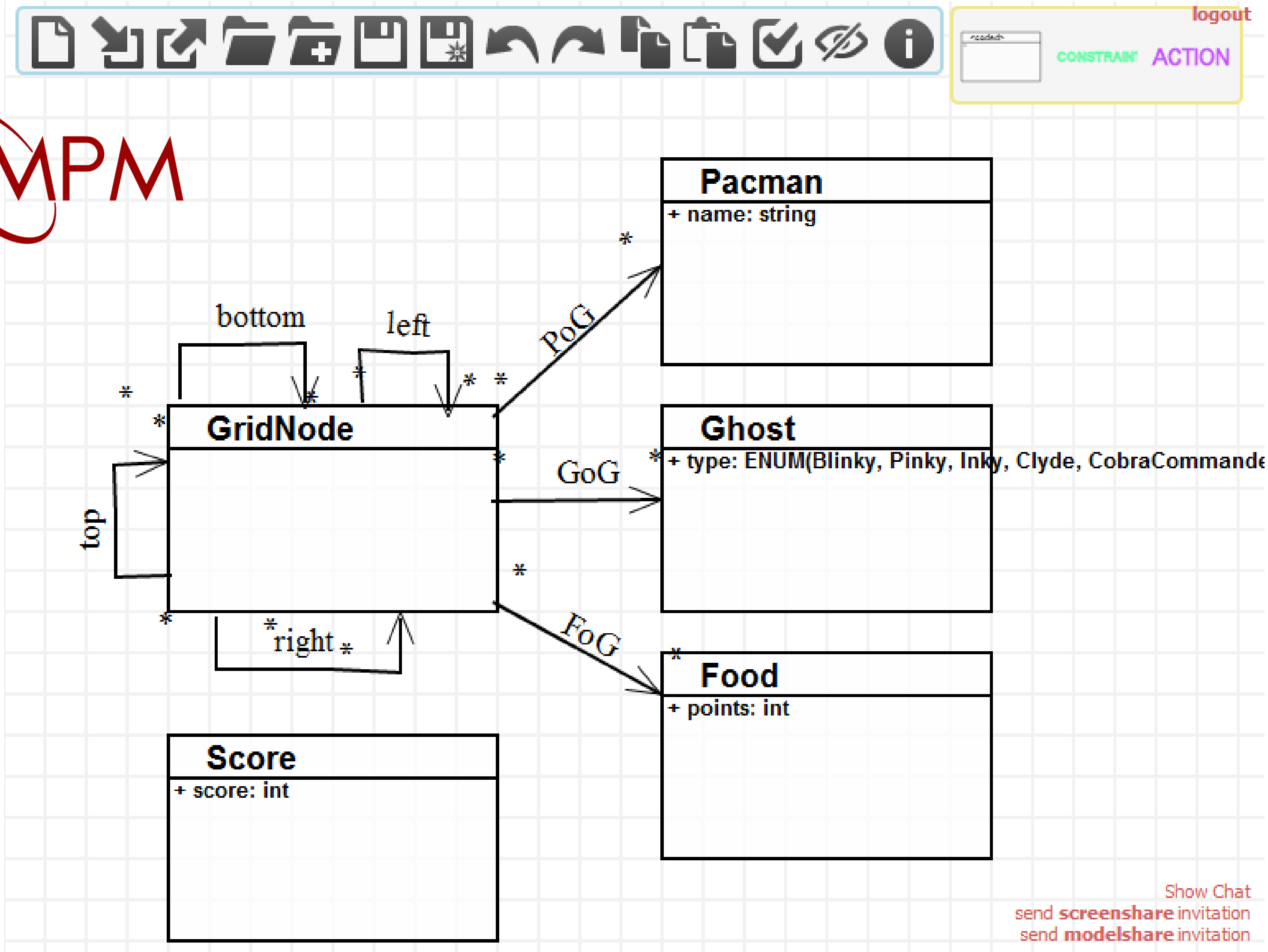
Exogenous	Outplace	Inplace
Refinement, Synthesis, Translational semantics	Refinement, Query	Simulation

Rule-based model transformation

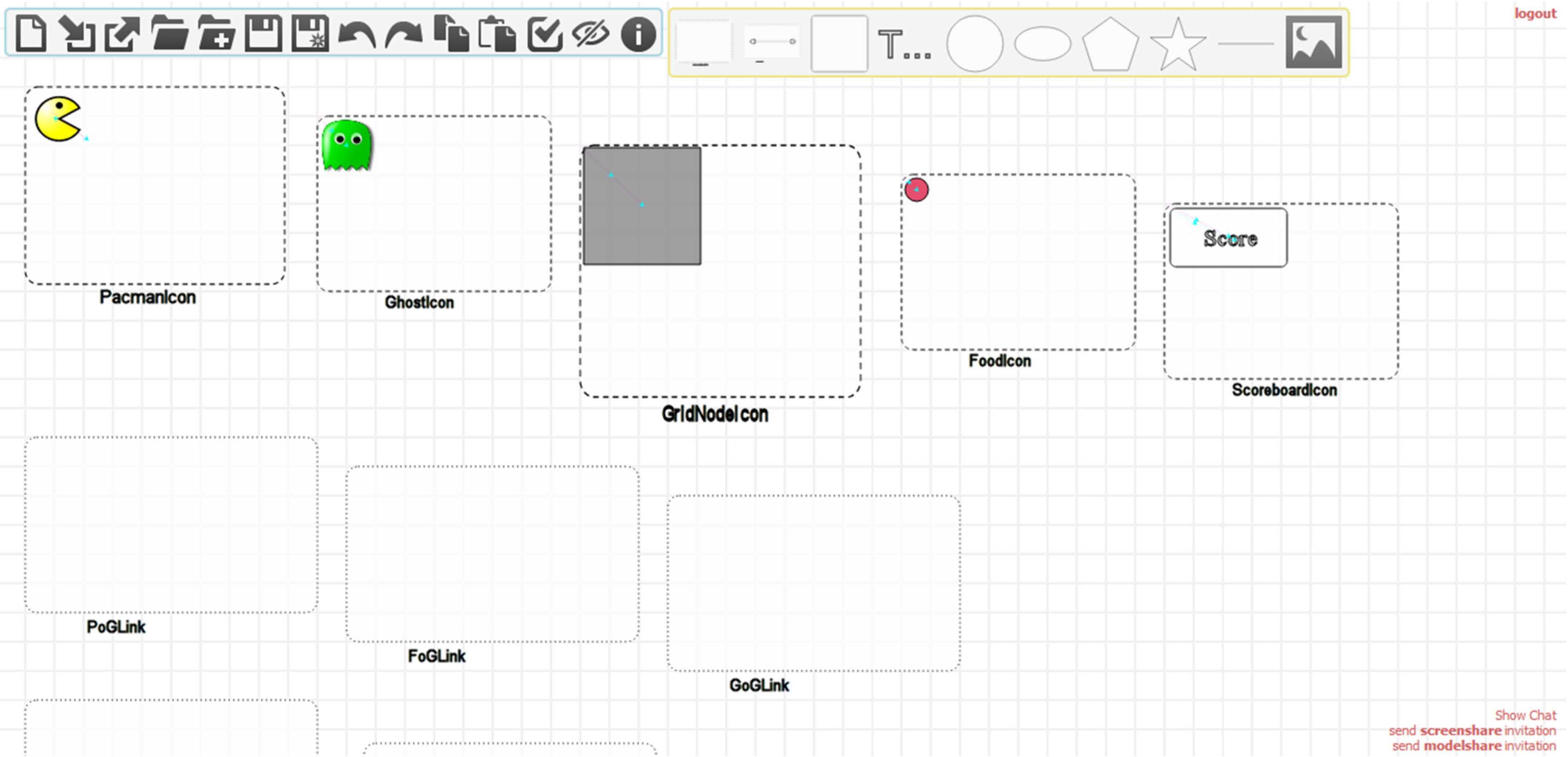
Graph transformation for simulation

- Models are considered as directed, typed, attributed graphs
- Transformations on such graphs are considered as graph rewritings
- Features:
  - Declarative paradigm
  - Rules defined as pre- and post-conditions
- Tools: **MoTif**, Henshin, GReAT

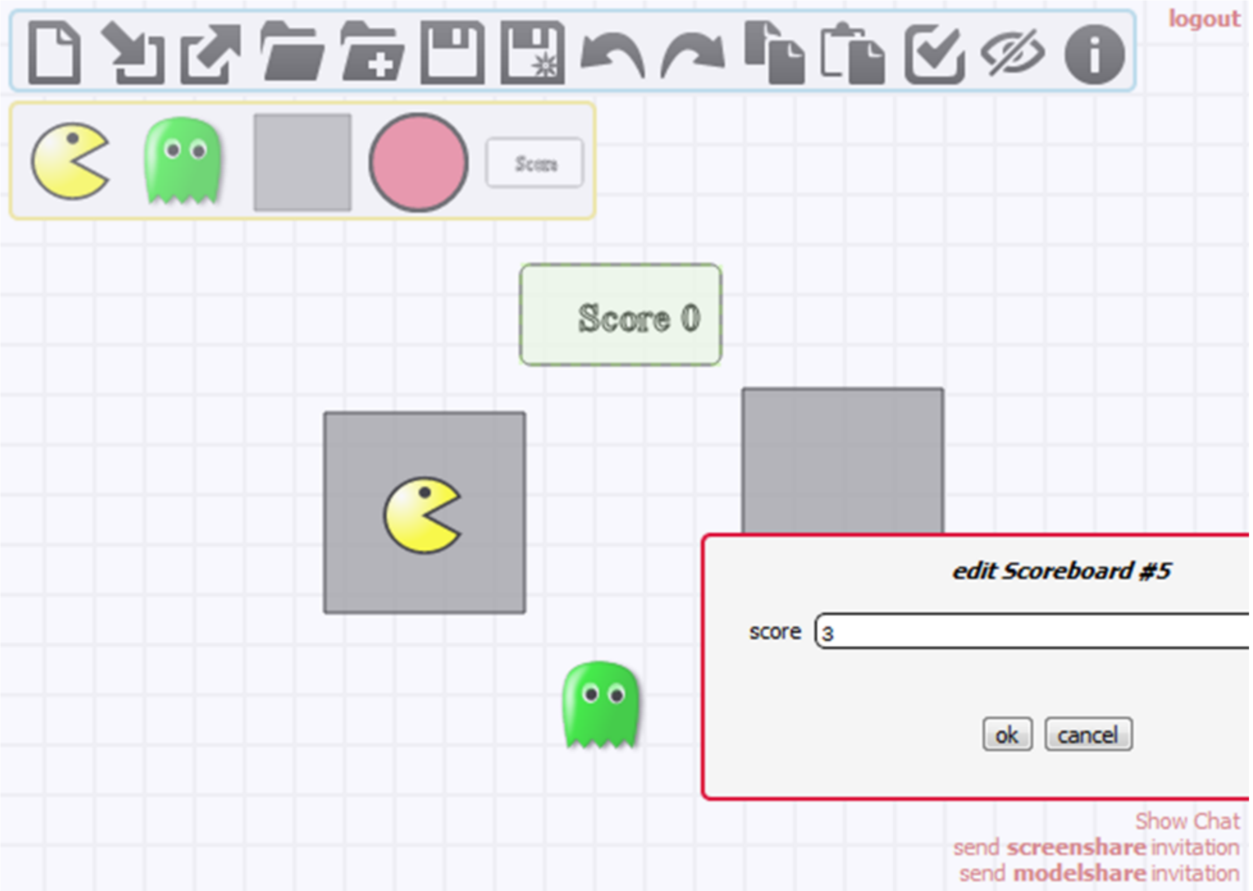
Metamodel of Pacman



Concrete syntax



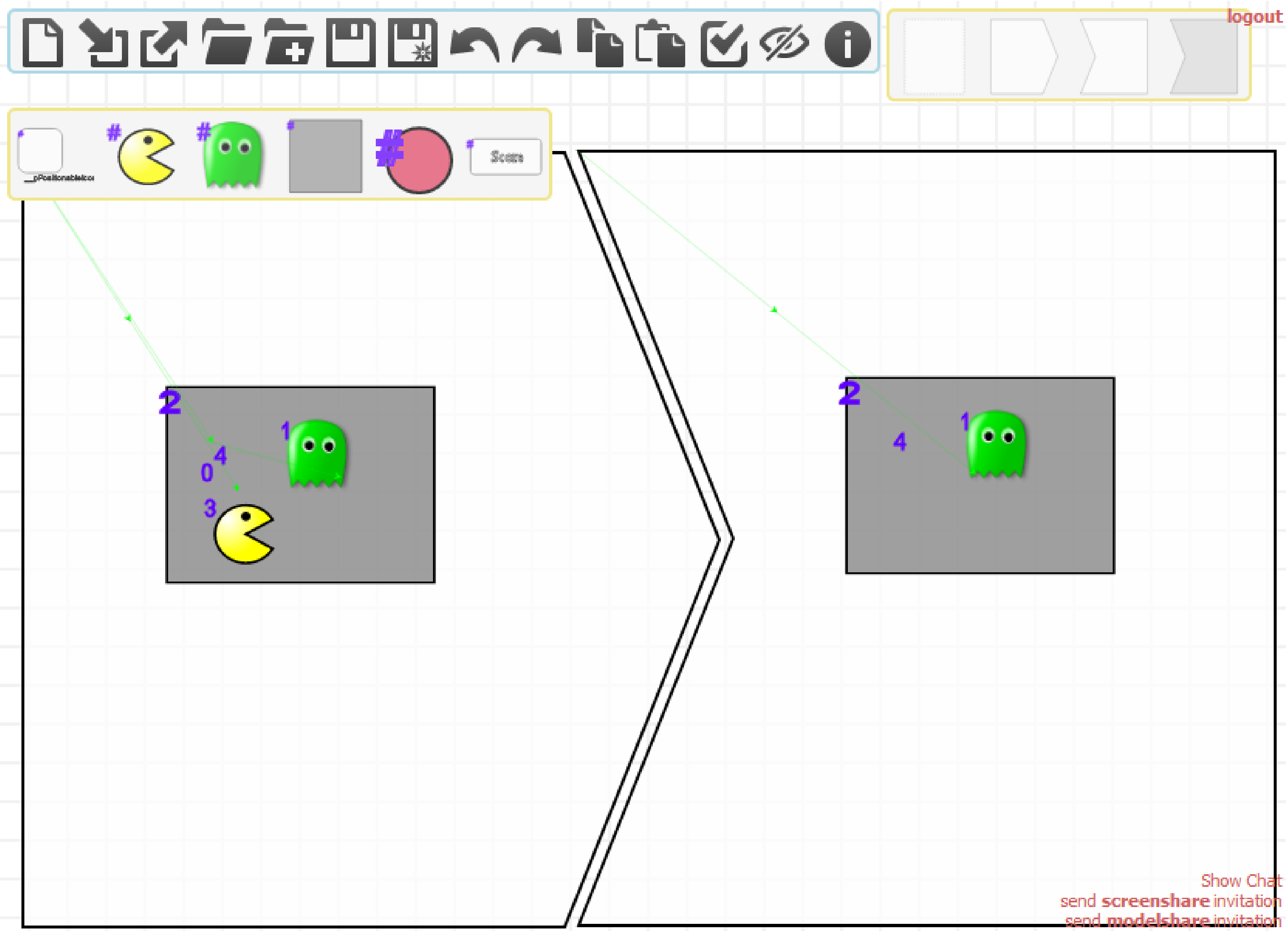
Generate modeling environment



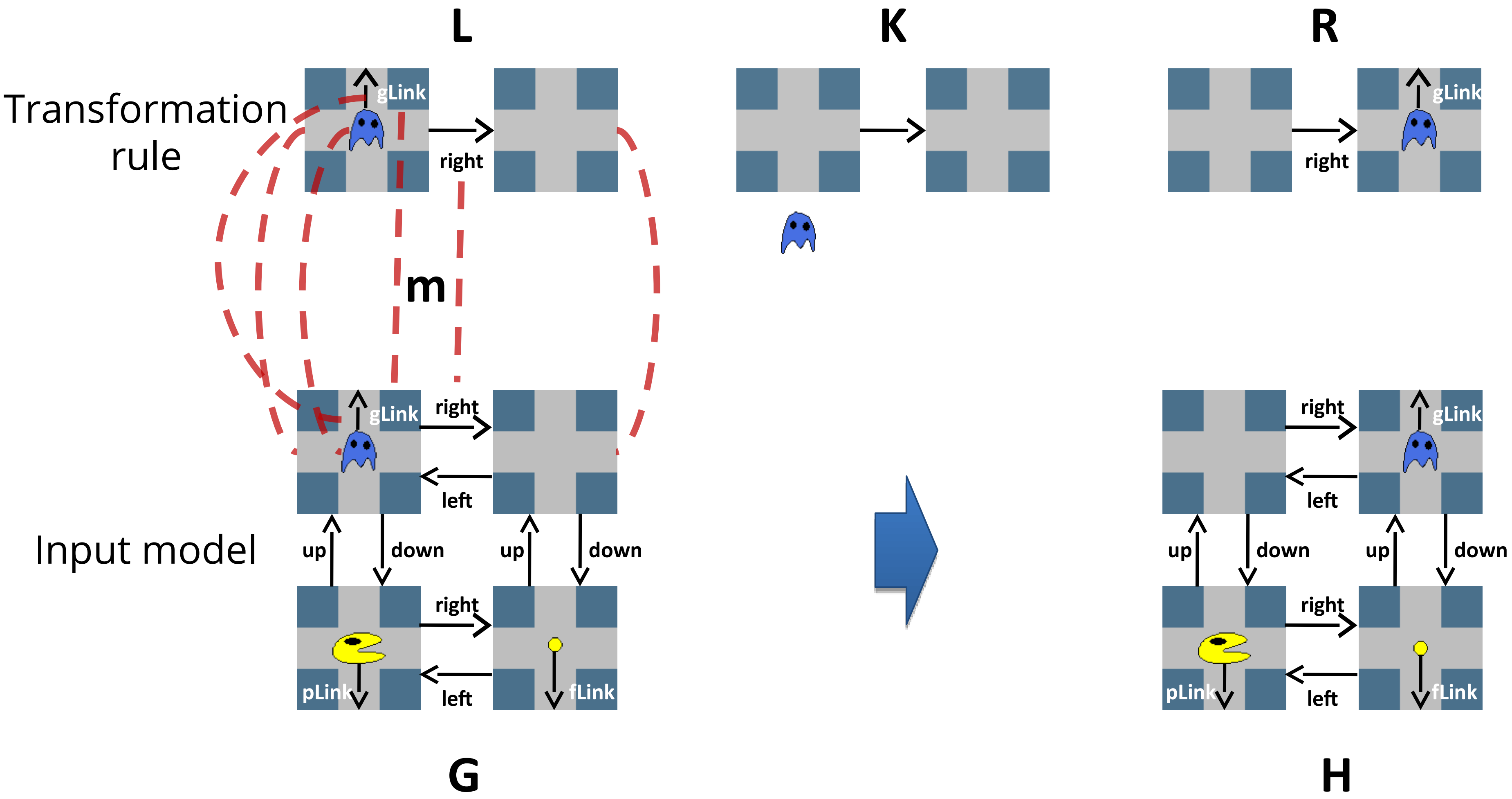
edit Scoreboard #5

score: 3

Graph transformation rule



Rule-based graph transformation



If there exists an occurrence of **L** in **G** then replace it with **R**

Mechanics of rule application

1. Matching Phase
   - Find an embedding m of the LHS pattern L in the host graph G
   - An occurrence of L is called a **match**: $m(L)$
   - Thus, $m(L)$ is a sub-graph of G
2. Rewriting Phase
   Transform G so that it satisfies the RHS pattern:
   - **Remove** all elements from $m(L-K)$ from G
   - **Create** the new elements of $R-K$ in G
   - **Update** the properties of the elements in $m(L\cap K)$

- When a match of the LHS can be found in G, the rule is **applicable**
- When the rewriting phase has been performed, the rule was **successfully applied**

QUESTION

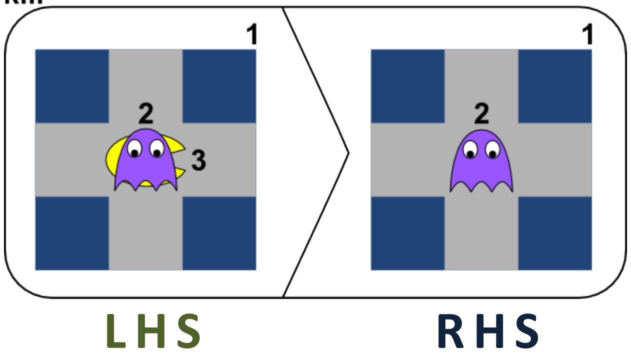
What is the worst upper-bound of the complexity for applying a graph transformation rule?

- $O\left(|G|^{|L|} + |R|\right) = O\left(|G|^{|L|}\right)$ CRUD operations

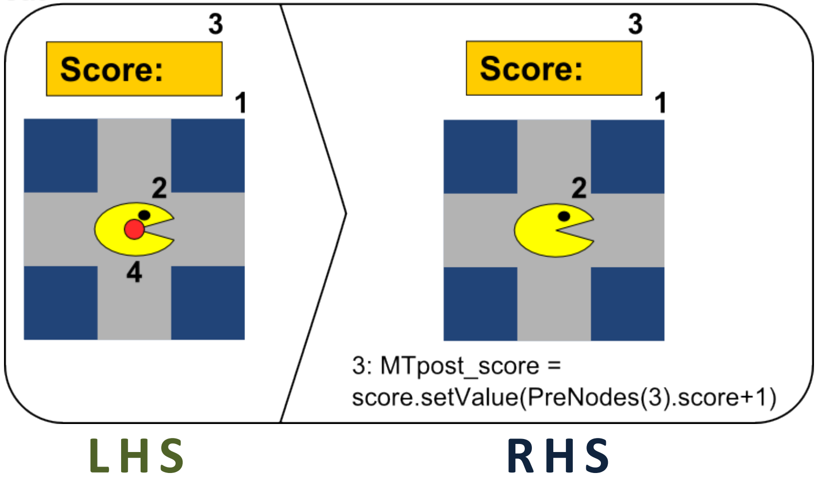
CRUD operations

Operational semantics

kill

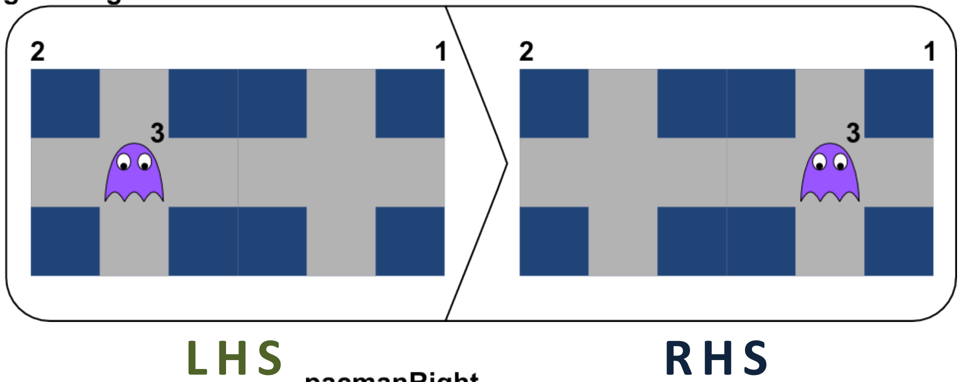


eat

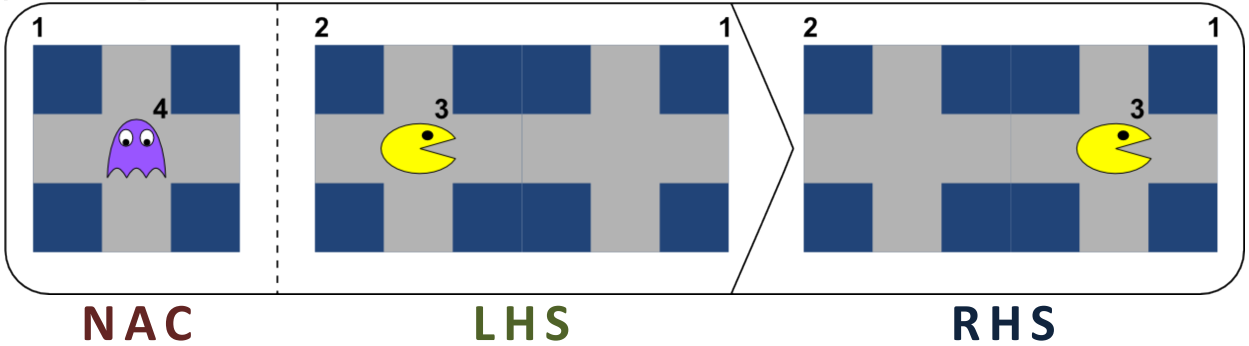


3: MTpost_score = score.setValue(PreNodes(3).score+1)

ghostRight

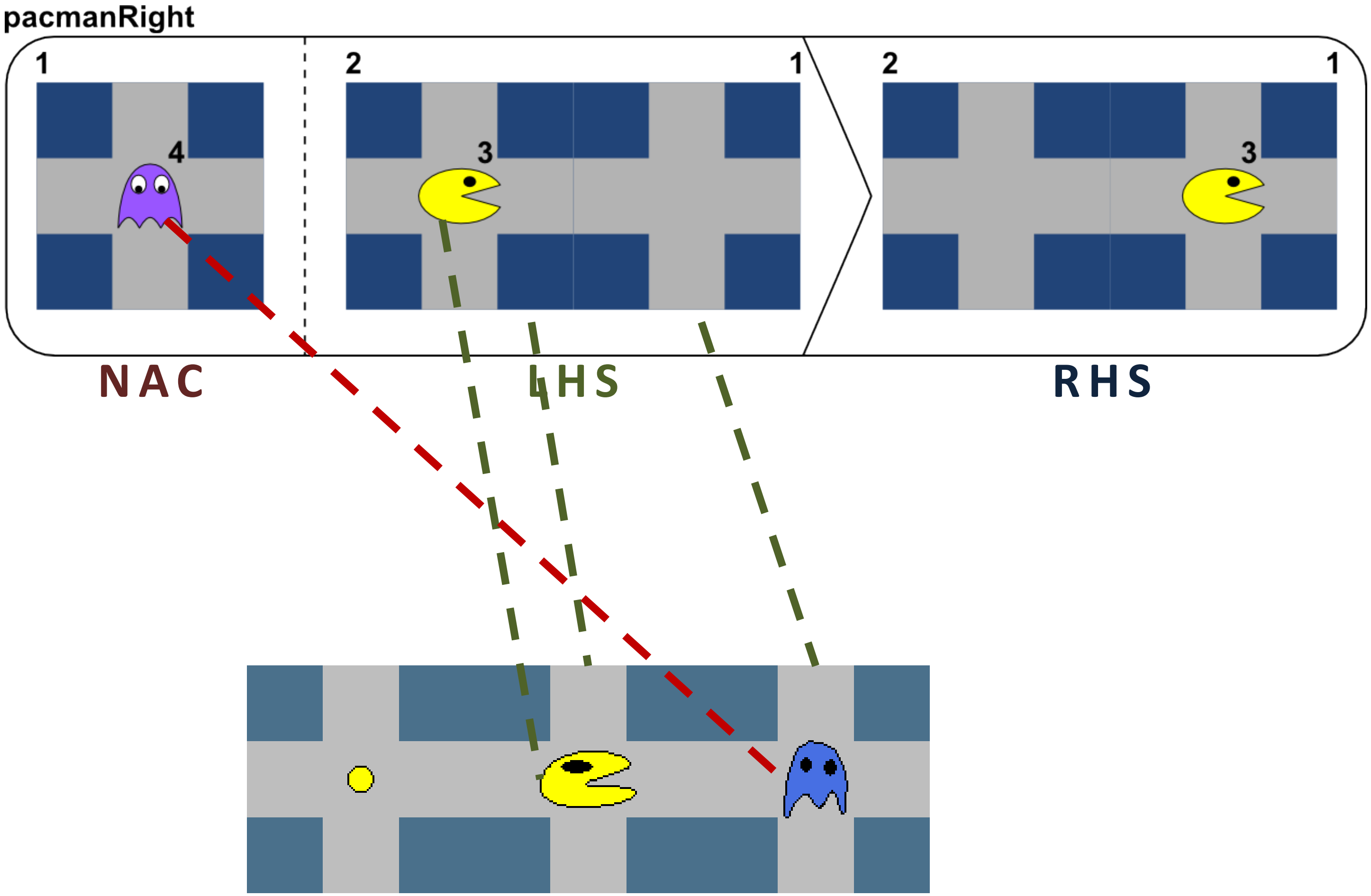


pacmanRight



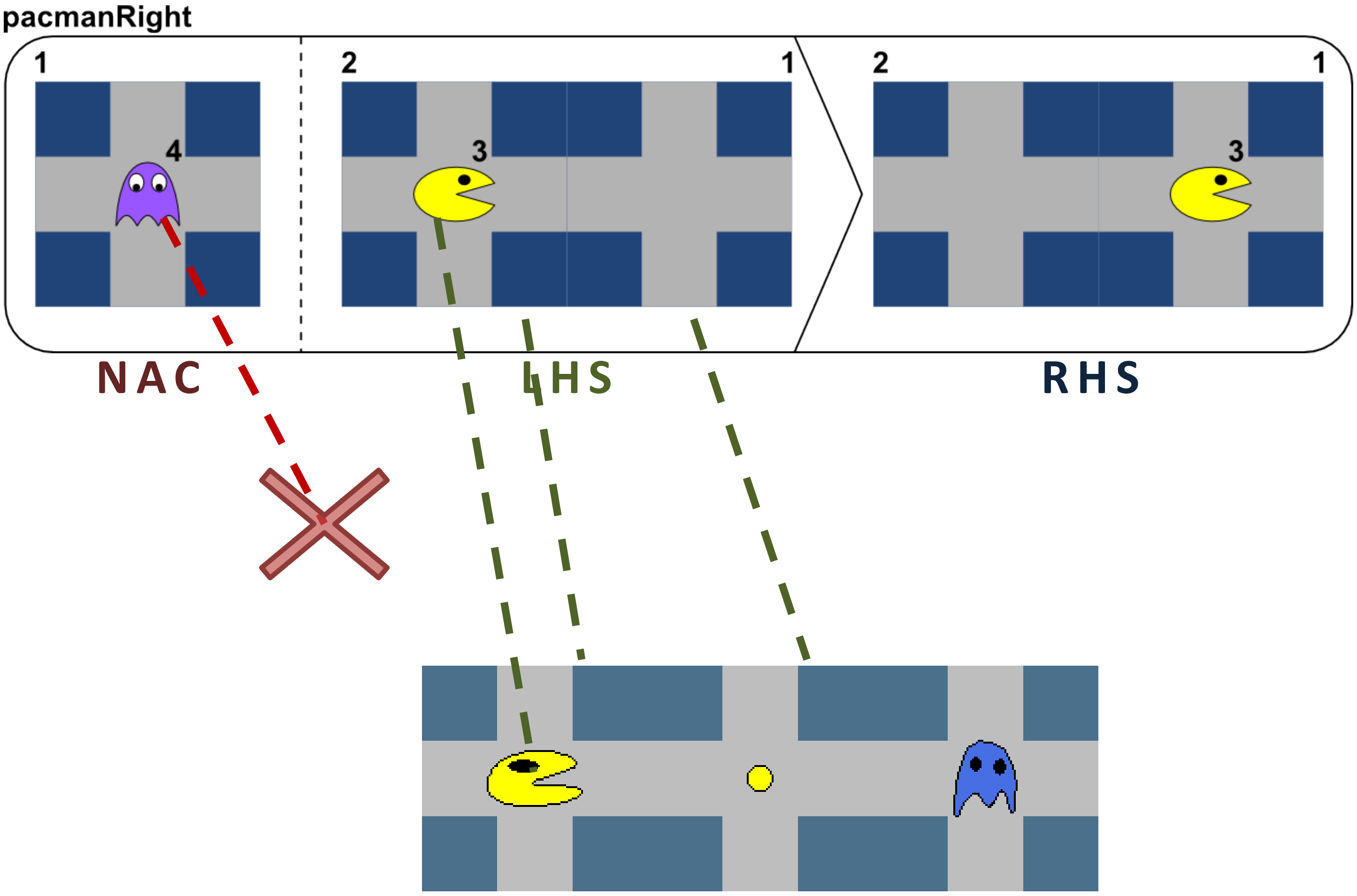
Negative application conditions

Non-applicable rule



Negative application conditions

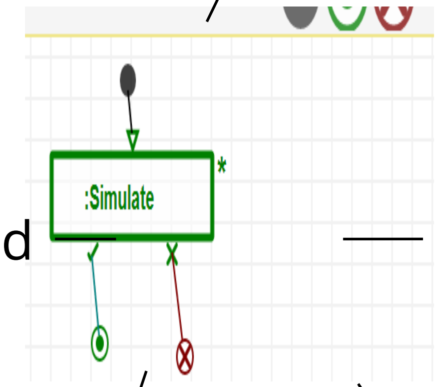
Applicable rule



Rule scheduling

- In what order should the rules be executed?
  - Don’t care: randomly, non-deterministically
  - Partial order
  - Explicit ordering
- **MoTif** is the transformation language of AToMPM

Initial rule



Rule type

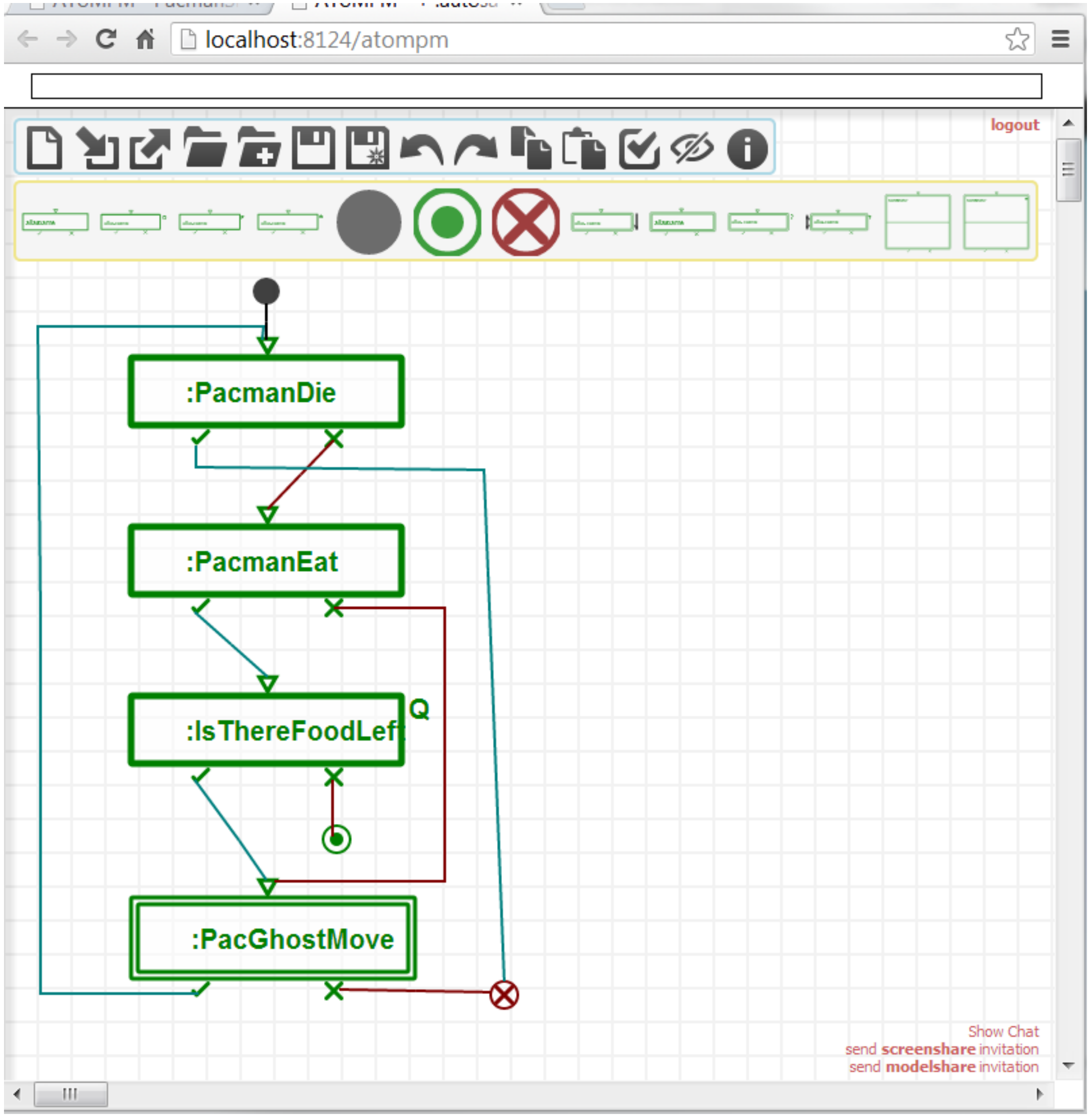
If match found

If no match found

Terminate transformation in success

Terminate transformation in failure

Scheduling of the rules



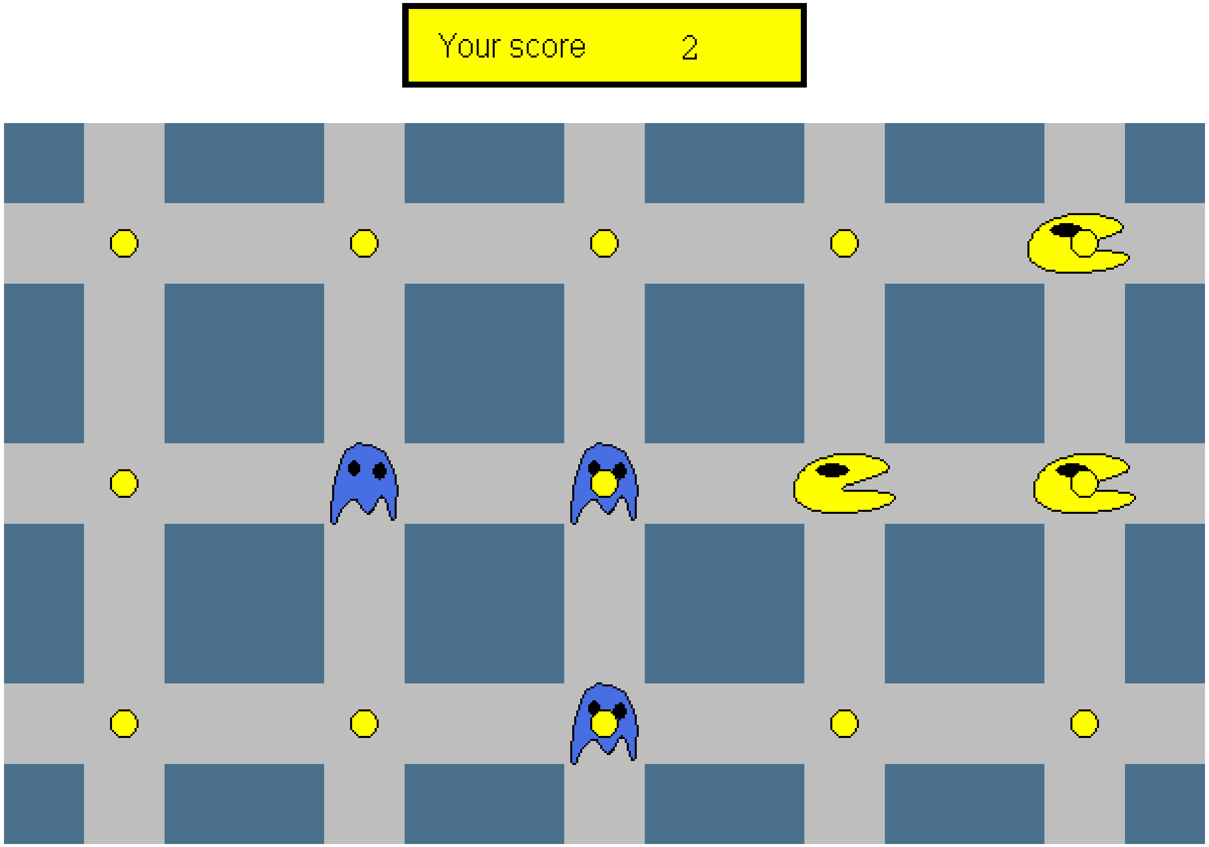
QUESTION

How to specify the rule `IsThereFoodLeft`?

- Rule with only a LHS
- LHS consists of solely a food element
- It will be encapsulated in a negative query
  1. If rule is applicable: FAIL
  2. Otherwise: SUCCESS

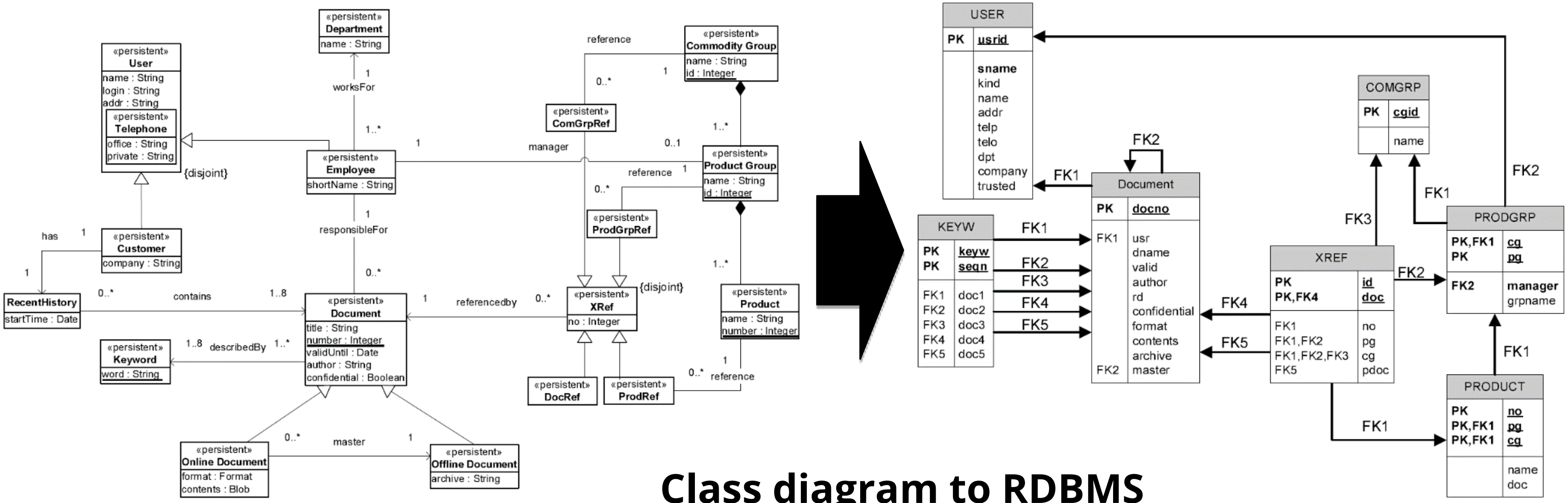
Simulation of a model

1. pacmanDie
2. pacmanEat
3. isThereFoodLeft
4. ghostMoveLeft
5. ghostMoveRight
6. ghostMoveUp
7. ghostMoveDown
8. pacmanMoveLeft
9. pacmanMoveRight
10. pacmanMoveUp
11. pacmanMoveDown



Translation

- **Maps** concepts of a model in a source language to concepts of another target language, while translating the **semantics** of the former in terms of the latter
- Similar to translational semantics, but the source language already has a semantics



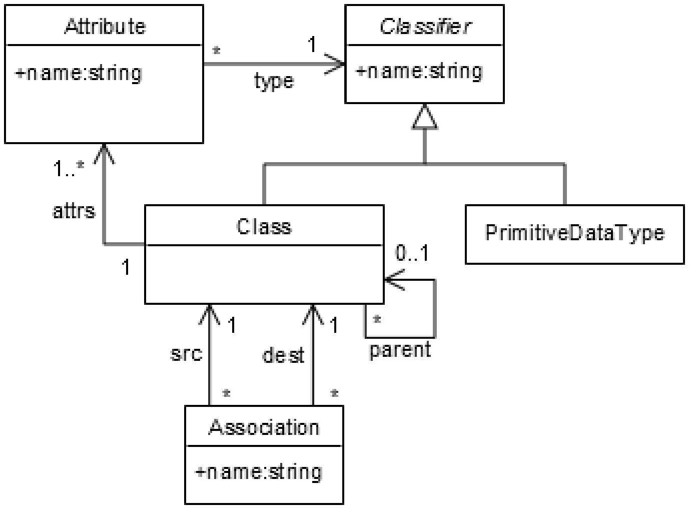
Class diagram to RDBMS

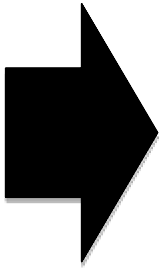
J. Bézivin et al. Model Transformation in Practice Workshop Announcement. *MODELS*, 2005

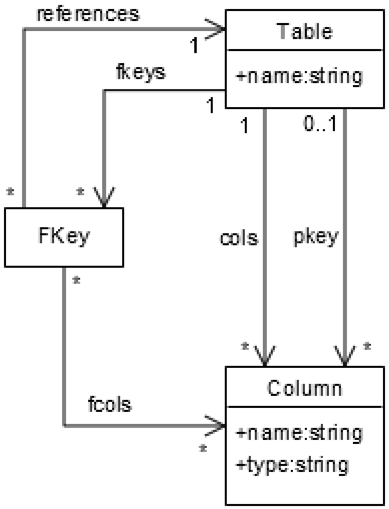
CD to RDBMS transformation

CD metamodel

RDBMS metamodel





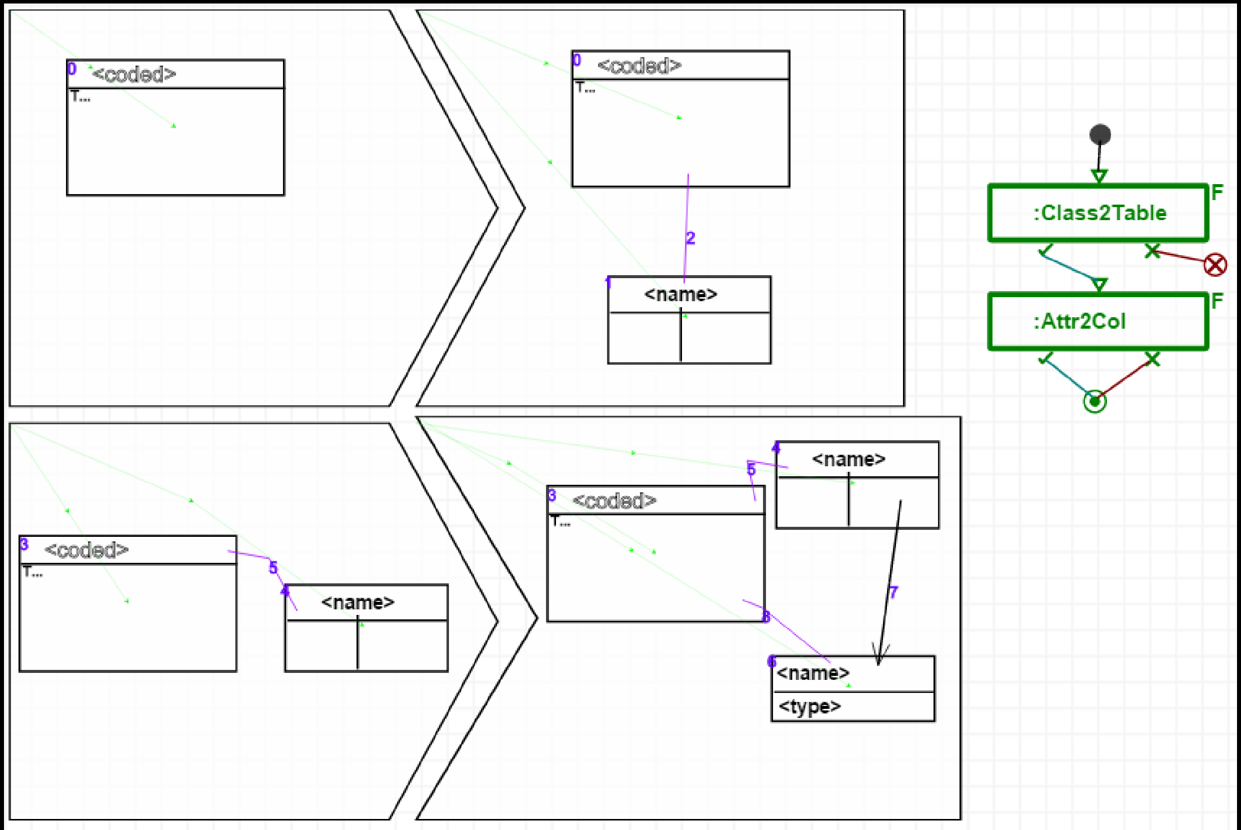


QUESTION

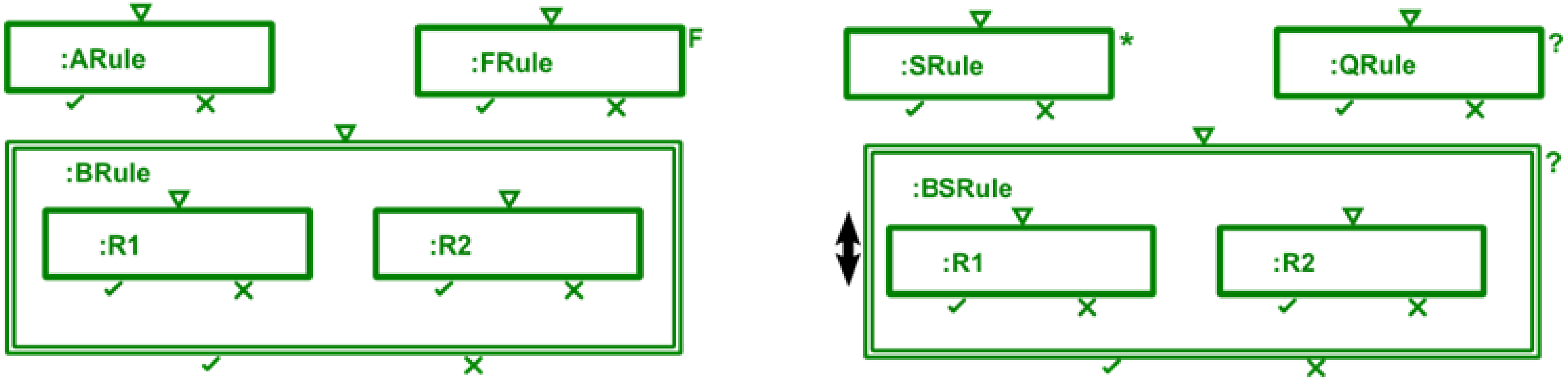
Implement in MoTif the transformation for:

Classes to tables

Attributes to columns

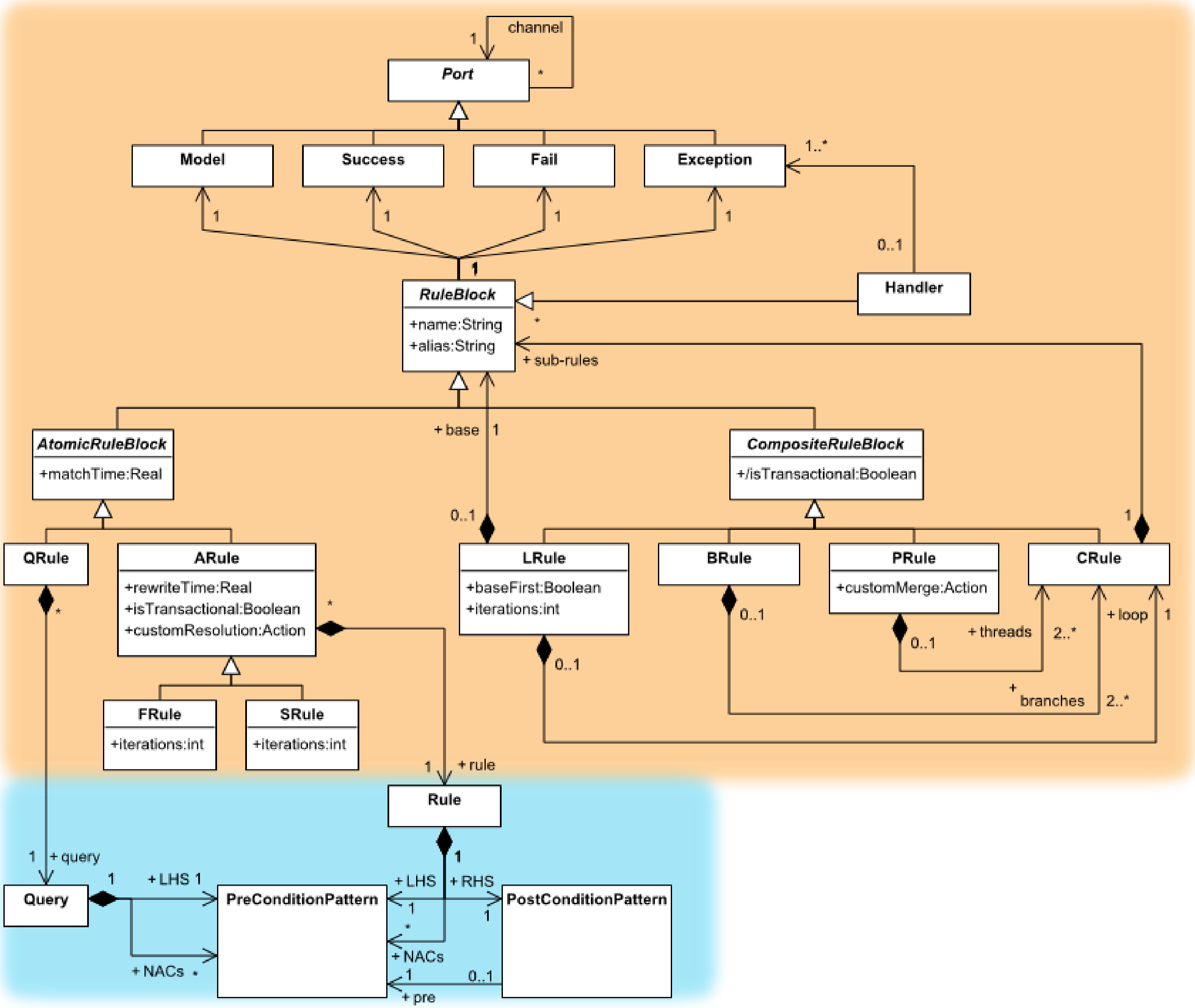


MoTif main rule types

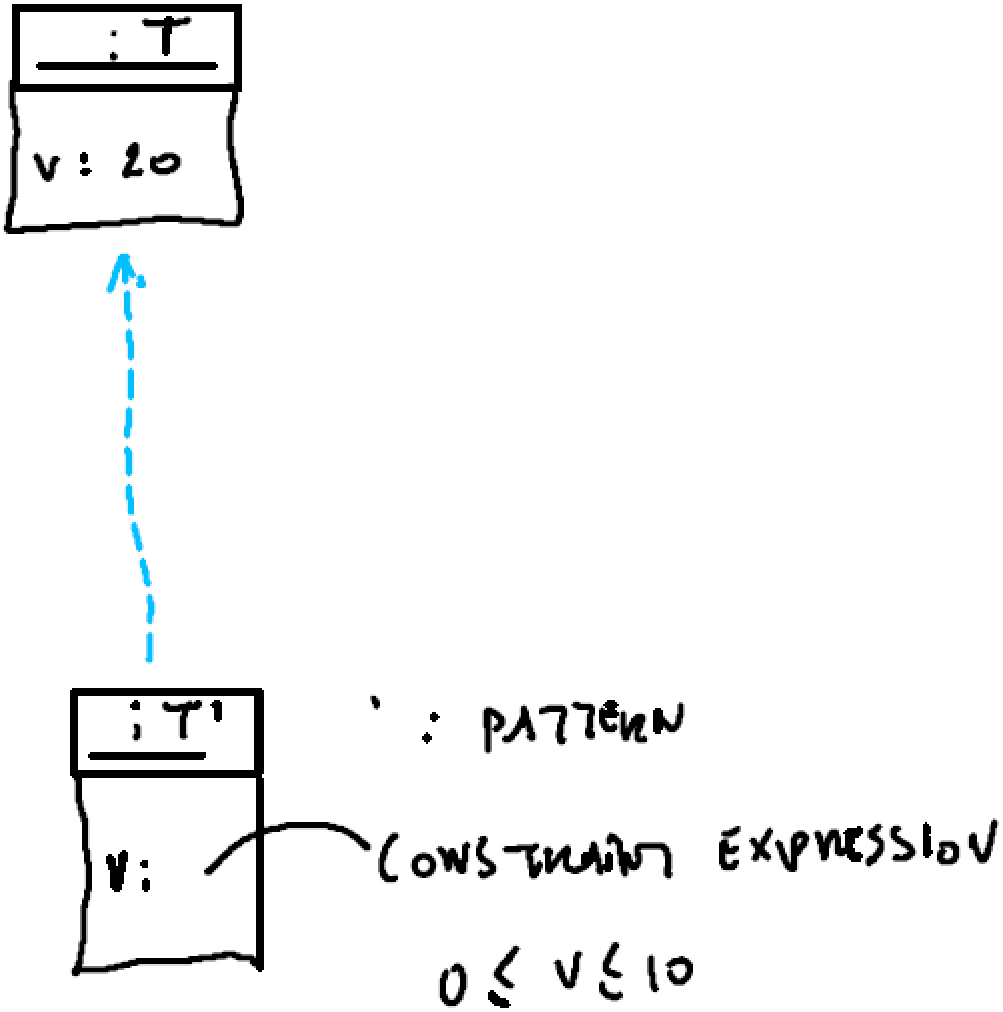


- **ARule:** (atomic) Applies rule on one match
- **FRule:** (for all) Applies rule on all matches found in parallel
- **SRule:** (star) Applies rule recursively as long as a match is found
- **QRule:** (query) Finds a match, only LHS no RHS
- **BRule:** (branch) Randomly (uniformly!) selects one matching rule
- **BSRule:** (branch star) Applies BRule as long as one rule matches

Model Transformation



Pattern model <> Instance model

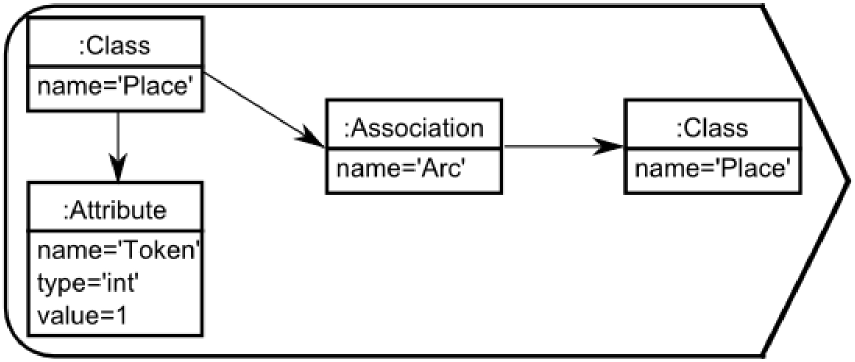


Instance model

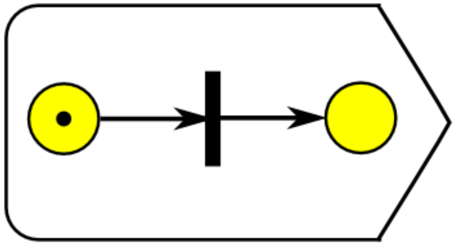
Pattern model

Pattern language

1. Generic pattern language
   - + Most economic solution
   - - Generic concrete syntax (MOF-like)
   - - Allow to specify patterns that will never occur

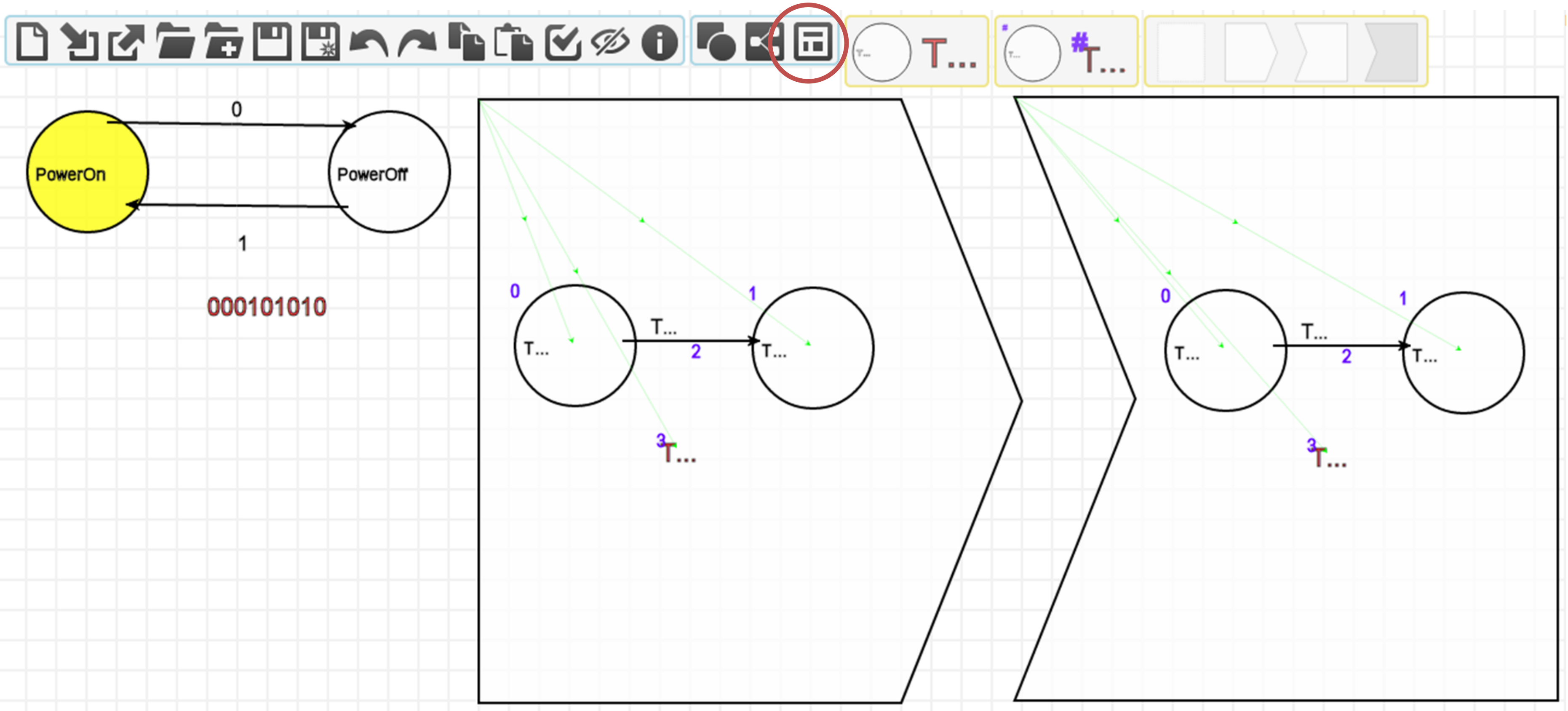


2. Customized pattern language
   - + Concrete syntax adapted to the source/target languages (DSL)
   - + Exclude patterns that do not have a chance to match
   - - More work for the tool builder



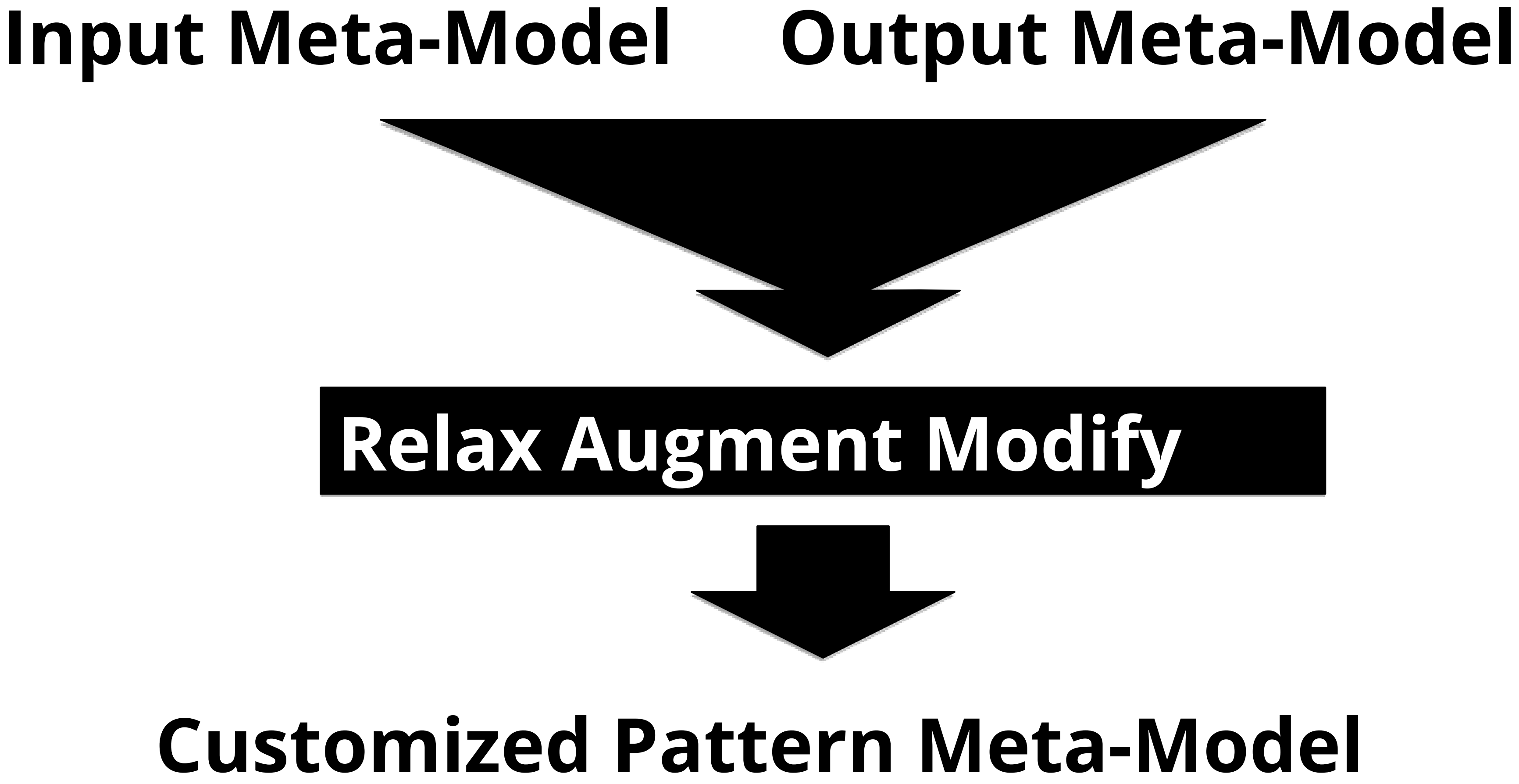
[E.b1]

RAMification process



Domain-specific pattern languages

Ramification Process: automatically generated environment for pattern language



Input Meta-Model **Output Meta-Model**

Relax Augment Modify

Customized Pattern Meta-Model

RAMification process

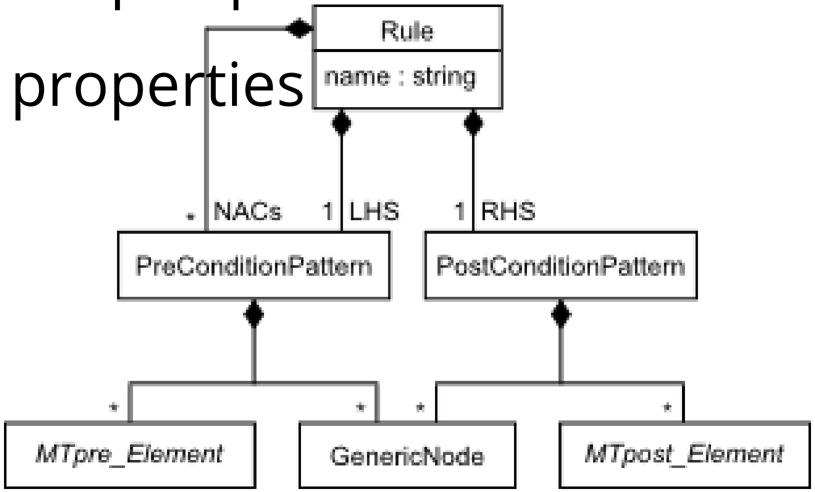
Relaxation

- Relaxes the constraints imposed by the meta-model of the domain
- Instantiation of originally abstract classes
- Reduction of minimal multiplicity of every association end
- Constraints filtering (manual)
  - Removed
  - Preserved
  - Depends on static semantics of language

RAMification process

Augmentation

- Augments the resulting meta-model with additional information
- Classes & associations integrated in rule meta-model
- Re-typing of all meta-model entities to pre/post
- Add model transformation specific properties
  - Labels
  - Parameter passing (pivots)
- Allow abstract rules
- Augmented constraints
- Connection with generic/trace elements

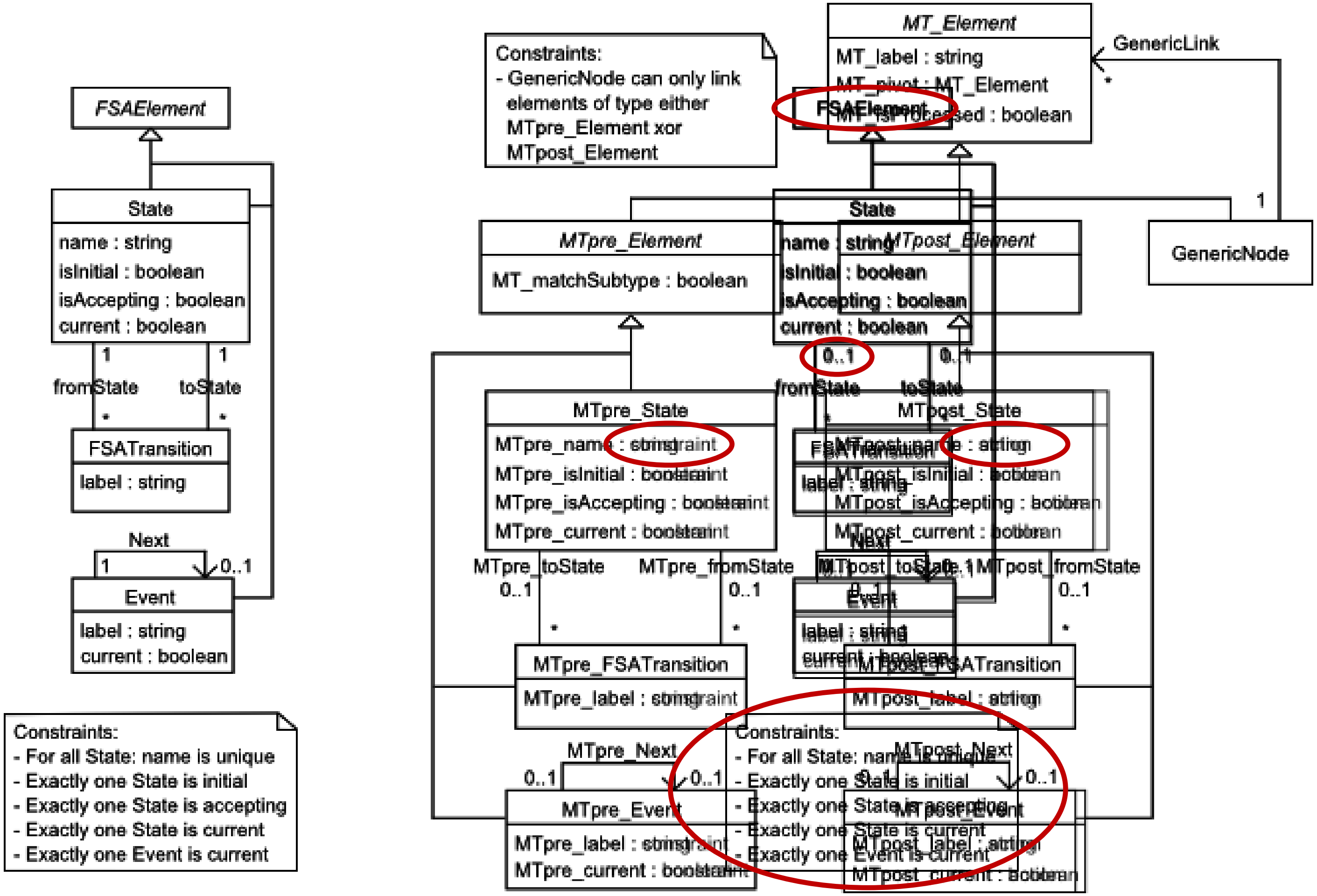


RAMification process

Modification

- Performs further modifications on the resulting meta-model
- Update namespaces
- Change type of attributes
  - Pre-condition classes: constraint type
  - Post-condition classes: action type
  - But preserve knowledge of original type for well-formedness
- Adaptation of concrete syntax (semi-automatic)
  - Abstract classes
  - Association ends
  - Other (e.g., replace topological visual syntax constraints)

RAMification process



FSAElement

State
name : string
isInitial : boolean
isAccepting : boolean
current : boolean

1 fromState *
1 toState *

FSATransition
label : string

Next 1 0..1

Event
label : string
current : boolean

Constraints:
- For all State: name is unique
- Exactly one State is initial
- Exactly one State is accepting
- Exactly one State is current
- Exactly one Event is current

Constraints:
- GenericNode can only link elements of type either MTpre_Element xor MTpost_Element

MT_Element
MT_label : string
MT_pivot : MT_Element
MT_isProcessed : boolean

GenericLink * 1
GenericNode

MTpre_Element
MT_matchSubtype : boolean

MTpost_Element

MTpre_State
MTpre_name : constraint
MTpre_isInitial : constraint
MTpre_isAccepting : constraint
MTpre_current : constraint

MTpost_State
MTpost_name : string
MTpost_isInitial : boolean
MTpost_isAccepting : boolean
MTpost_current : boolean

MTpre_toState 0..1 *
MTpre_fromState 0..1 *
MTpost_toState 0..1 *
MTpost_fromState 0..1 *

MTpre_FSATransition
MTpre_label : constraint

MTpost_FSATransition
MTpost_label : string

MTpre_Next 0..1 0..1
MTpost_Next 0..1 0..1

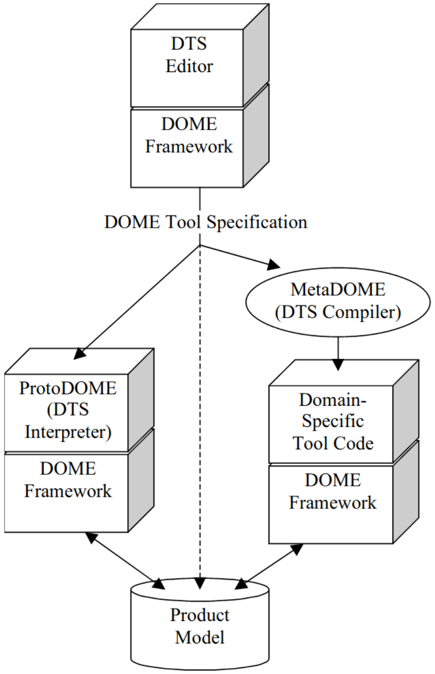
MTpre_Event
MTpre_label : constraint
MTpre_current : constraint

MTpost_Event
MTpost_label : string
MTpost_current : boolean

Relax Augment Modify

1990s: Honeywell's DoME

Domain Modeling Environment


DTS Editor

DOME Framework

DOME Tool Specification

MetaDOME (DTS Compiler)

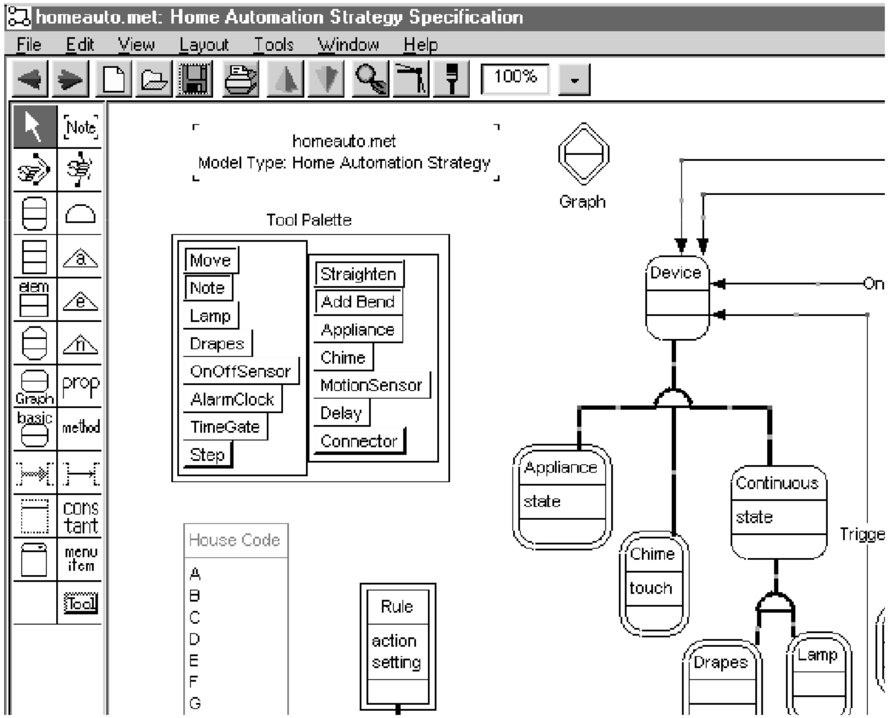
ProtoDOME (DTS Interpreter)

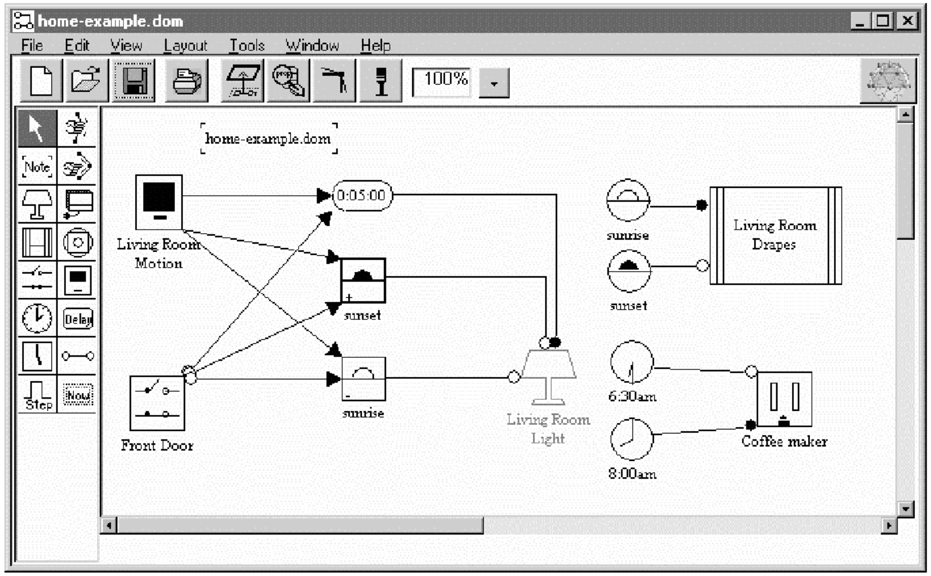
DOME Framework

Domain-Specific Tool Code

DOME Framework

Product Model





http://msdl.cs.mcgill.ca/people/mosterman/campam/cca01/cacsd00a/index.html/ek.pdf

Model Transformation

$[\![\text{"A+B"}]\!]_{REGEXP} = \{ \text{"AB"}, \text{"AAB"}, \dots \}$

GRAMMAR

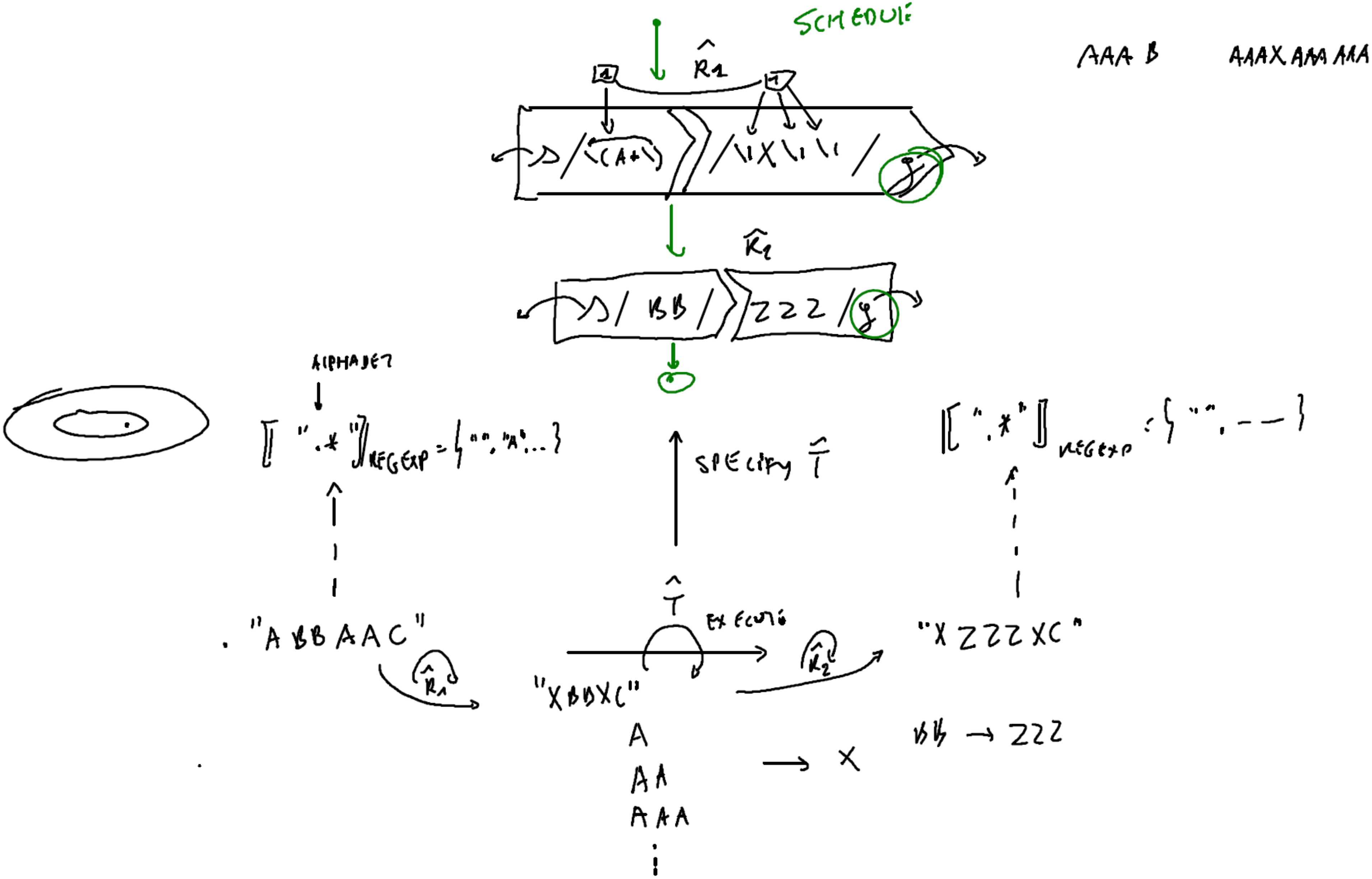
GME ε ε

MM_{CD}

GRAPH GRAMMAR

" TRANSFORMATION

Model Transformation



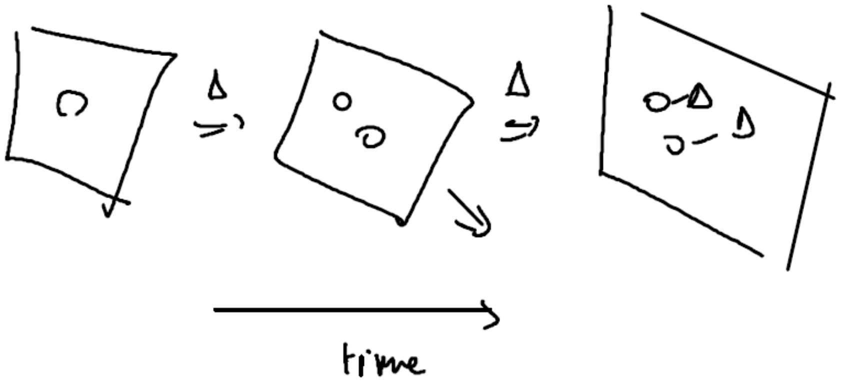
Transforming Strings, Trees, or Graphs?

STRINGS / SEQUENCE ⊆ TREES ⊆ GRAPHS

JAVA/Python ... ↔ || TRSF ⊆ TRSF ⊆ T~~RS~~F

HOW ↑ COGNITIVE ↑ WHAT? ↑ ↑ DECLARATIVE

DYNAMIC STRUCTURE



WHY NOT ALWAYS USE GRAPHS / GRAPH TRSF. ?

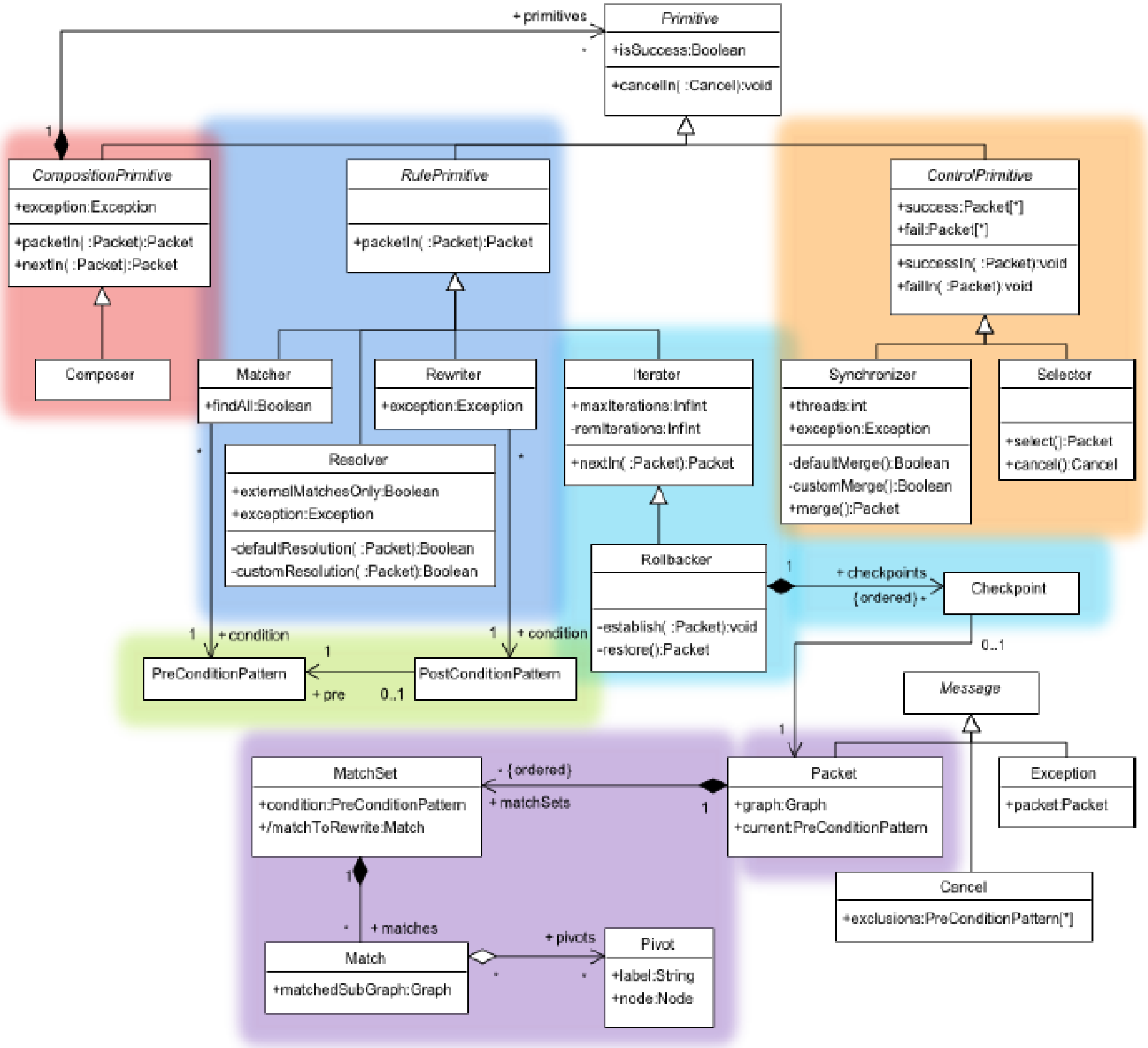
💻 ① COMPLEXITY → PERFORMANCE / EFFICIENCY

🧍 ① COGNITIVE / UNDERSTANDING

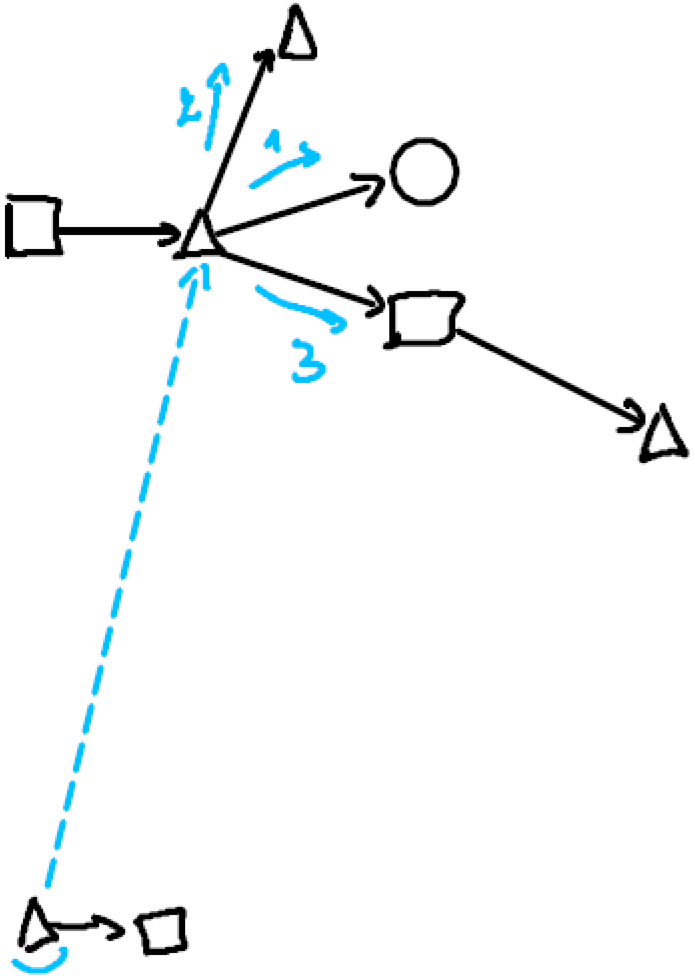
MOST APPROPRIATE - ABSTRACTION - FORMALISM

(DATA / COMPUTATION ... ENERGY / COST / ...)

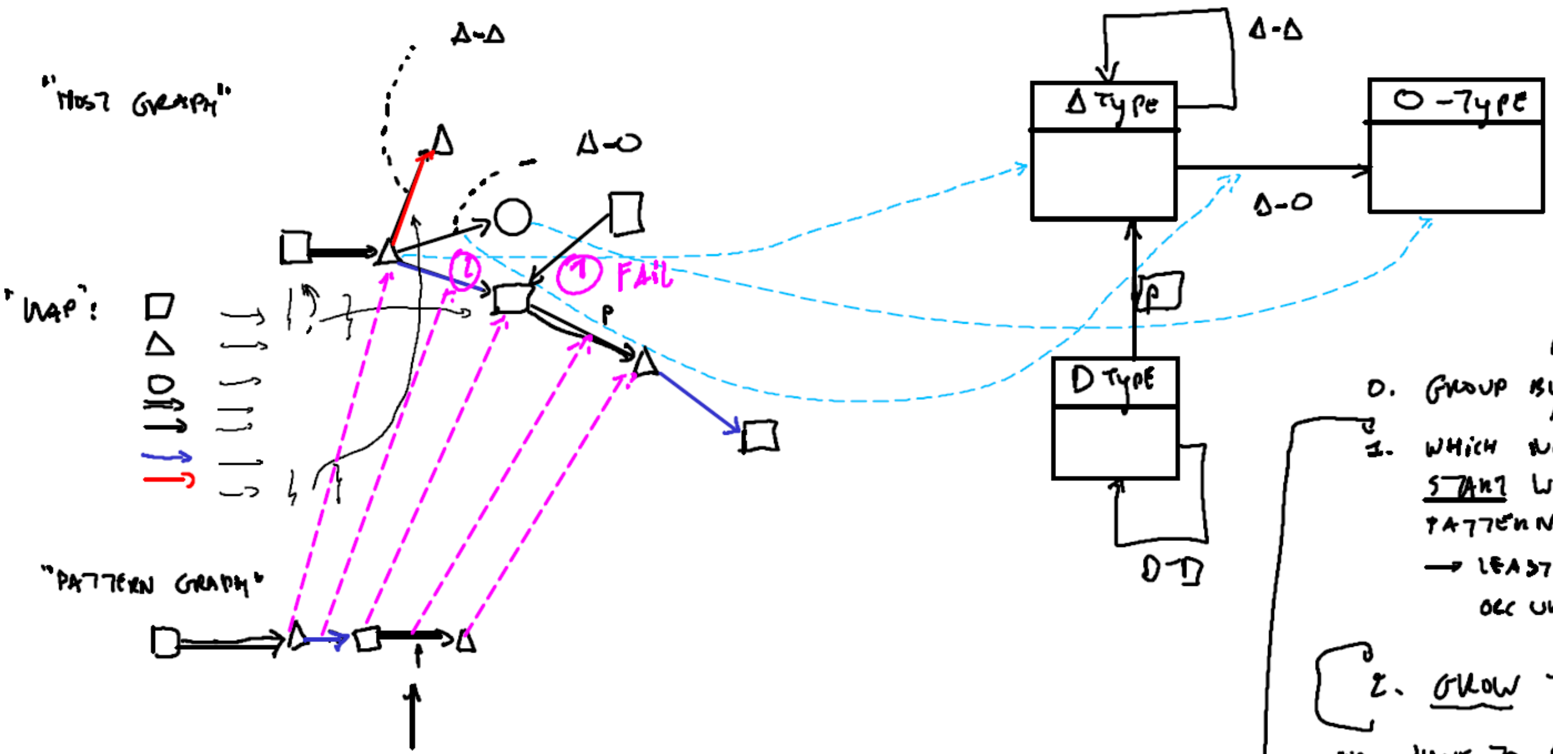
matching, pivot, scope



Matching Algorithms (1): Search Plans



Matching Algorithms (2): Constraint Satisfaction



BUILD / UPDATE "MAP"

0. GROUP BY TYPE (HOST)

1. WHICH NODE/EDGE to START WITH IN PATTERN?
→ LEAST FREQUENTLY OCCURRING TYPE

2. GROW THE MATCH
MAY HAVE TO BACKTRACK

3. COLLECT IN MATCH SET

4. REWRITE $O(D+A)$

L K R

L\K DELETE

R\K ADD

D = SIZE (L\K)

A = SIZE (R\K)

VF2 "Very Fast 2"

Matching Algorithms (2): improving performance through (user) “hints”

PIVOT PARTIAL MATCH

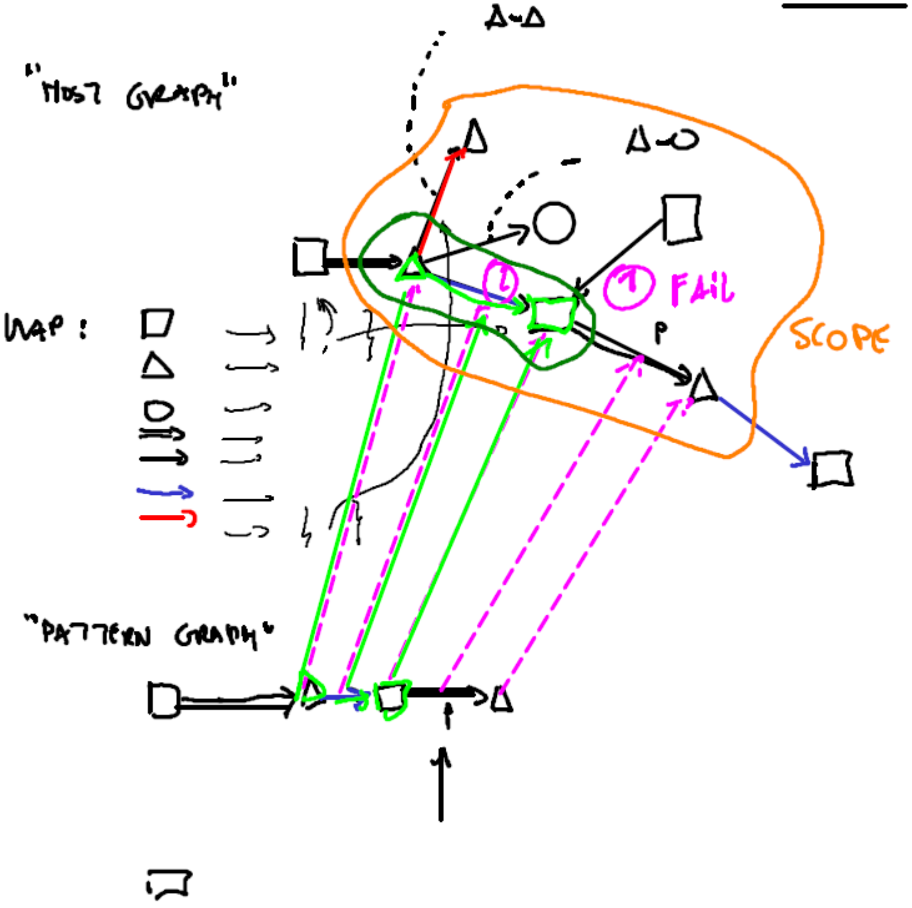
SUB-GRAPH OF HOST GRAPH

WHERE TO START

FIRST MATCH

$\Theta($ SIZE (PATTERN GRAPH)$)$

LINEAR!



SIZE (HOST GRAPH) $> 10^6$

"COMPLEX PATTERN"

→ FAIL EARLIER

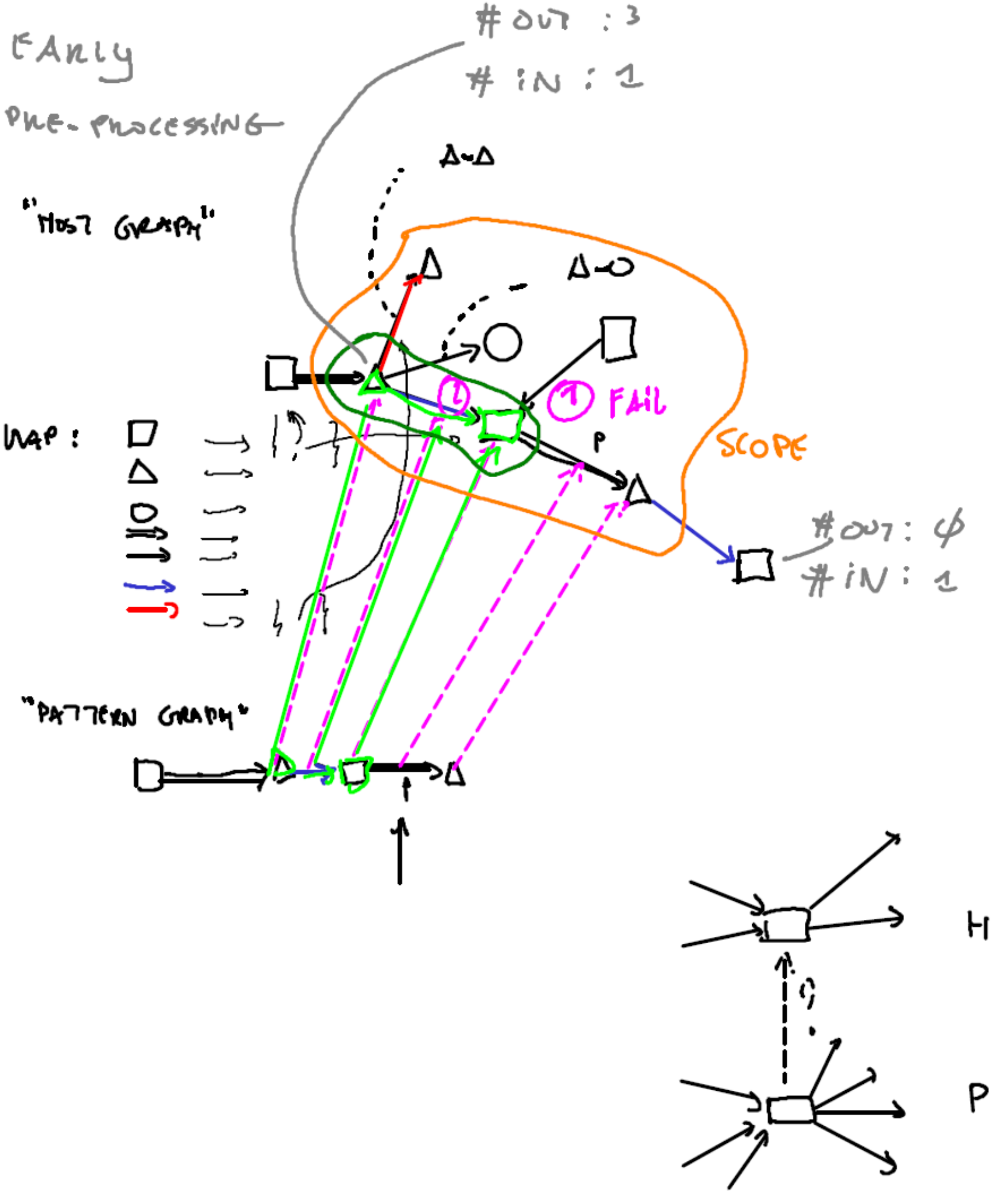
(FEWER MATCHES)

SCOPE WHAT TO EXCLUDE FROM HOST GRAPH ~ SIZEOF (SCOPE GRAPH)

Matching Algorithms (2): improving performance through heuristics

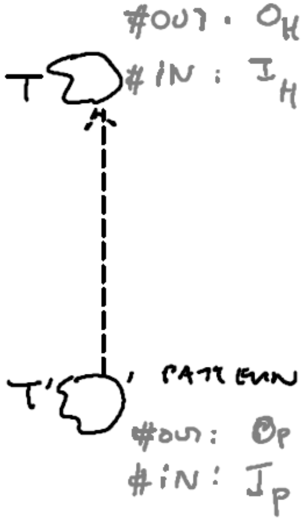
HEURISTIC FAIL EARLY

EXTRA PRE-PROCESSING



UPDATE WHEN HOST CHANGES

IN RHS OF RULE "REWRITE" PHASE



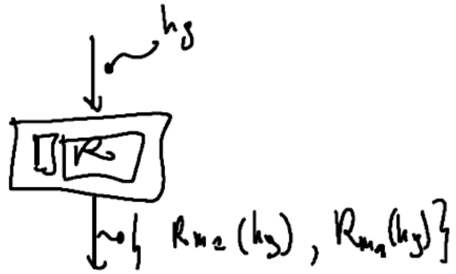
NO UPDATE NEEDED

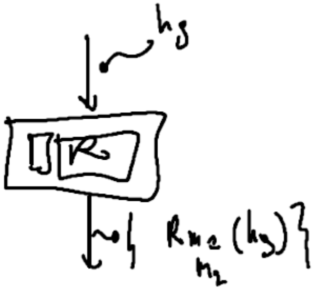
$I_P <= I_H$

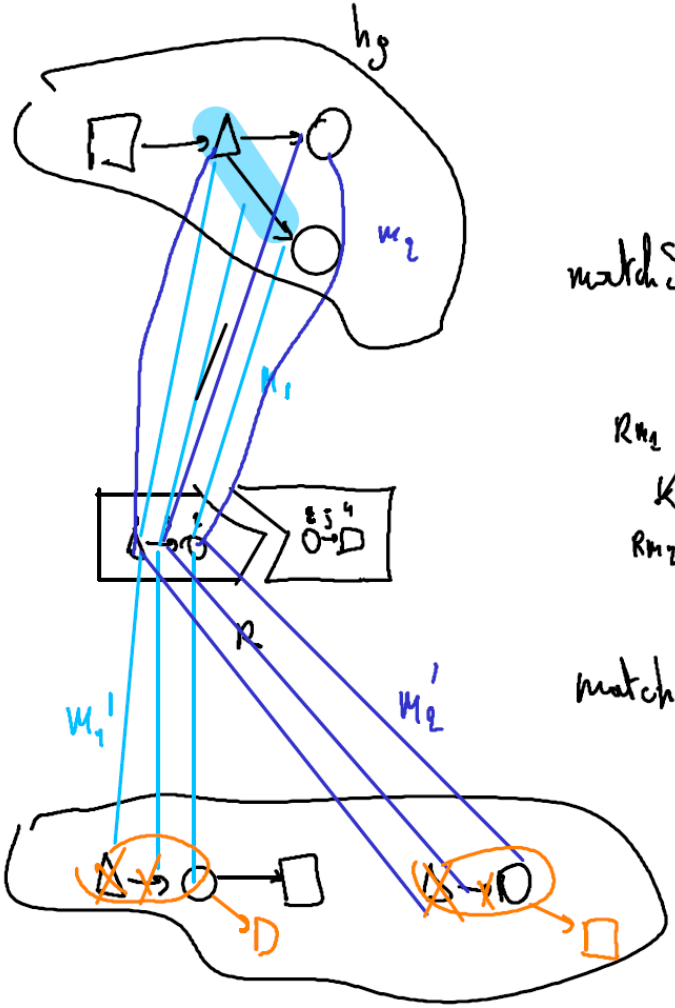
$O_P <= O_H$

Matching Algorithms: improving performance of “incremental” model transformation: the Rete algorithm

Choice → parallel independence, critical pairs







matchSet = { m1, m2 }

$$\begin{tikzcd} & hg \arrow[dl, "R_{m_1}"'] \arrow[dr, "R_{m_2}"] & \\ \cdot \arrow[dr, "R_{m_2}"'] & & \cdot \arrow[dl, "R_{m_1}"] \\ & \cdot & \end{tikzcd}$$

NO

$\bigcap \mathrm{img}_{hg}(m_i) = \phi$

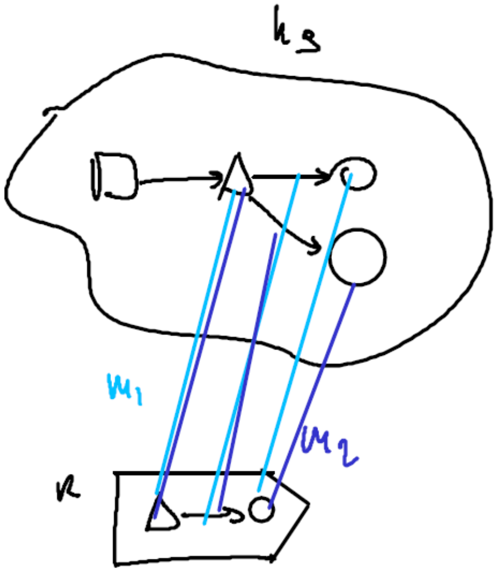
YES

matchset' = { m1', m2' }

Apply in //

Choice

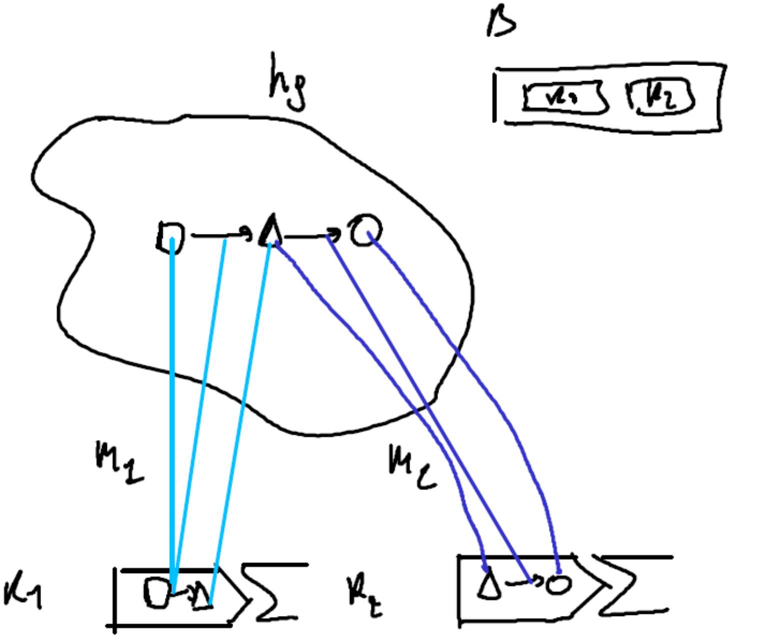
single rule, multiple matches



~~$U_I([0,1])$ MATCH~~

OPPORTUNISTIC

multiple rules, multiple matches



RANDOM CHOICE (REPEATABLE)

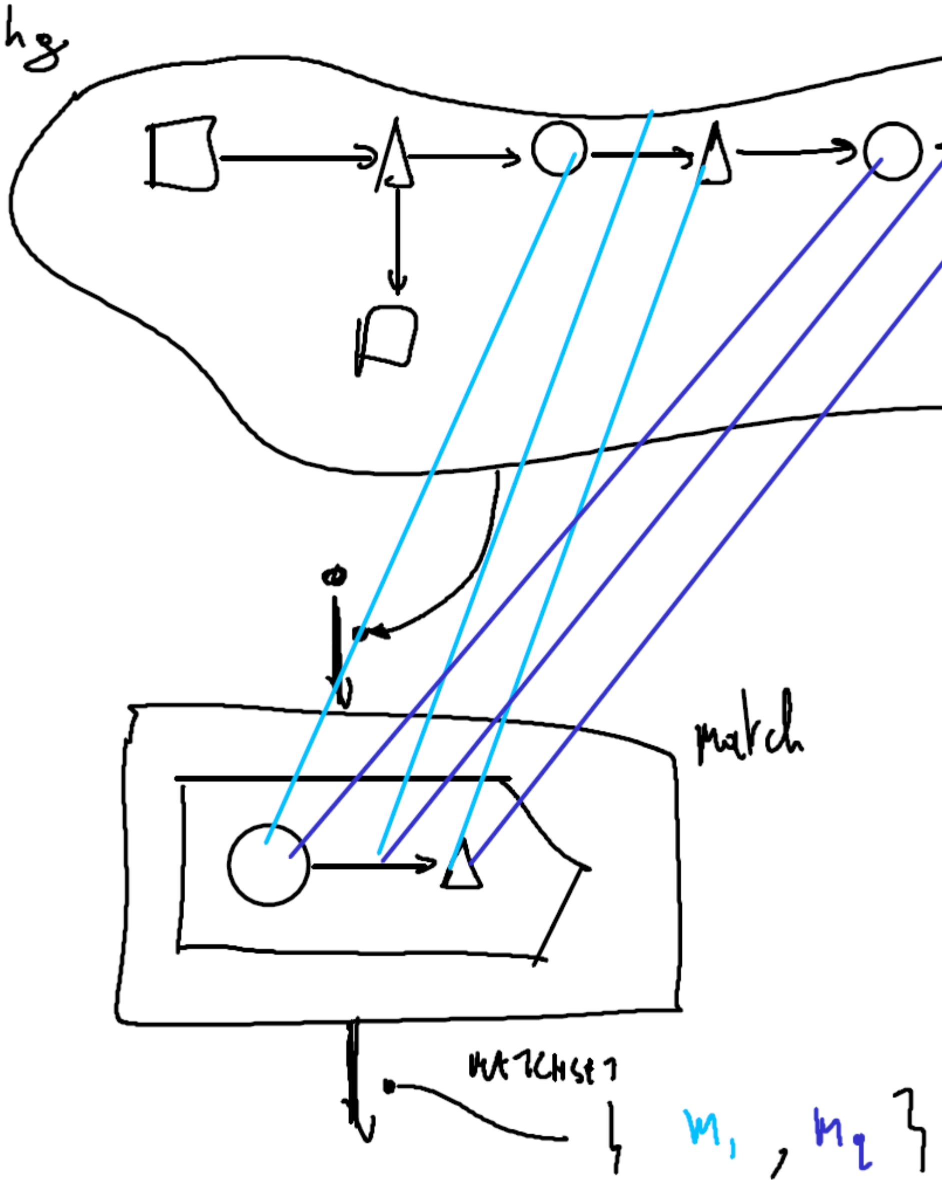
FROM ALL MATCHES

SELECT($\{\overset{\phi}{m_1}, \overset{1}{m_2}\}$)

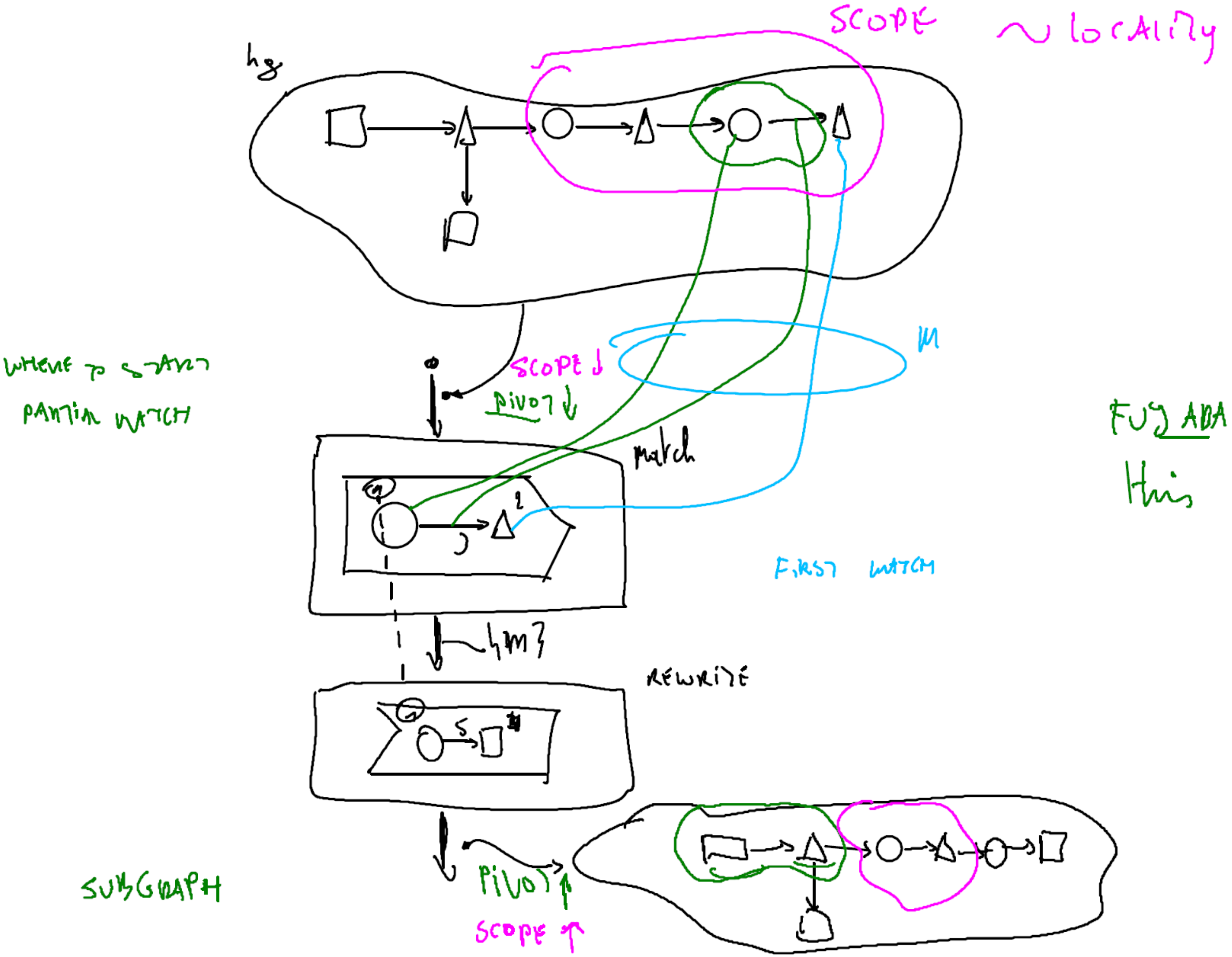
$U_I([0,1])$

PSEUDO-RANDOM NR. GENERATOR (SEED)

MatchSet = the set of all the matches (morphisms)



De-constructing a rule in Matching and Re-Writing



Model-to-model transformation for translation

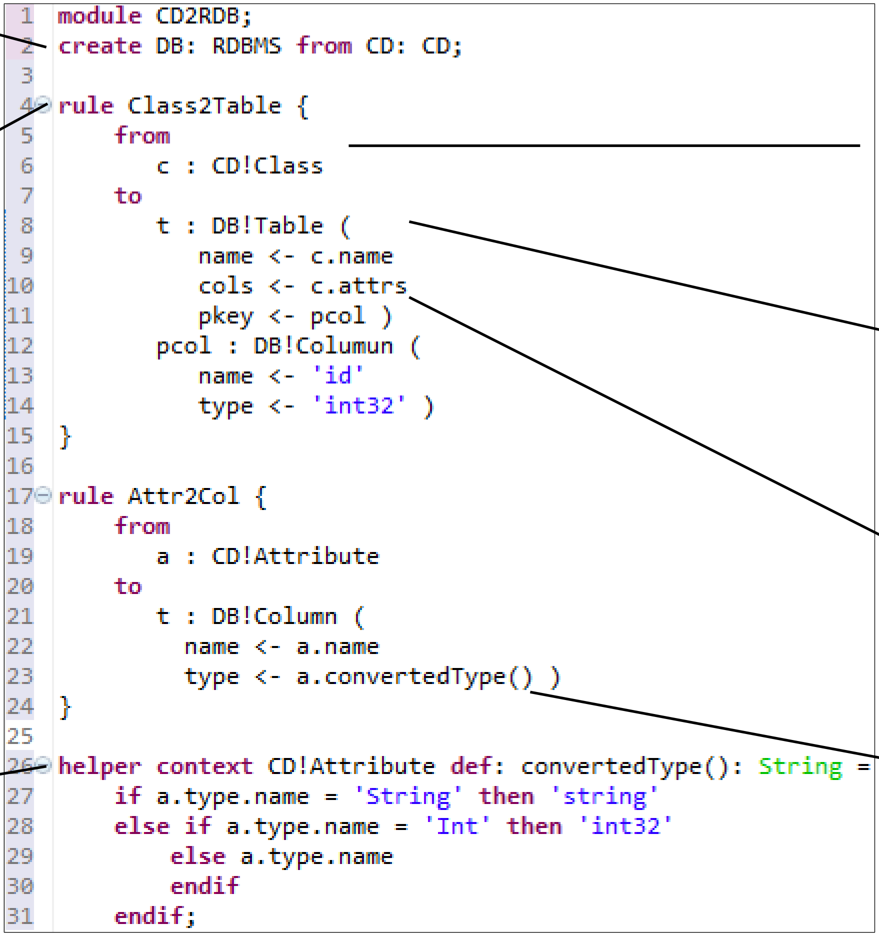
- Declarative paradigm
- Rules defined as non-destructing pre- and post-conditions
  - Source pattern to be matched in the source model
  - Target pattern to be created/updated in the target model for each match during rule application
- Typically models are represented in Ecore
- Input model is read-only
- Output model is write-only
- Tools: **ATL**, ETL, QVT-R

ATL transformation

Classes–Tables + Attributes–Columns

Create new model

Standard rule



```
module CD2RDB;
create DB: RDBMS from CD: CD;

rule Class2Table {
    from
       c : CD!Class
    to
       t : DB!Table (
          name <- c.name
          cols <- c.attrs
          pkey <- pcol )
       pcol : DB!Columun (
          name <- 'id'
          type <- 'int32' )
}

rule Attr2Col {
    from
       a : CD!Attribute
    to
       t : DB!Column (
         name <- a.name
         type <- a.convertedType() )
}

helper context CD!Attribute def: convertedType(): String =
    if a.type.name = 'String' then 'string'
    else if a.type.name = 'Int' then 'int32'
        else a.type.name
        endif
    endif;
```

LHS: 1 element type

RHS: elements to create in new model

Call implicitly another rule

Call temporary queries

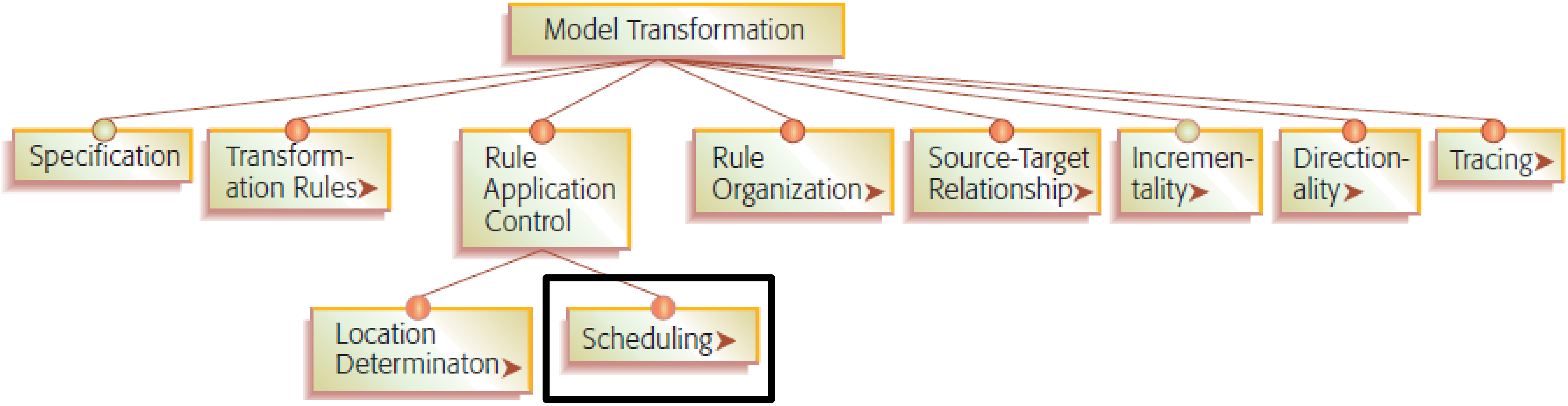
Helper in OCL

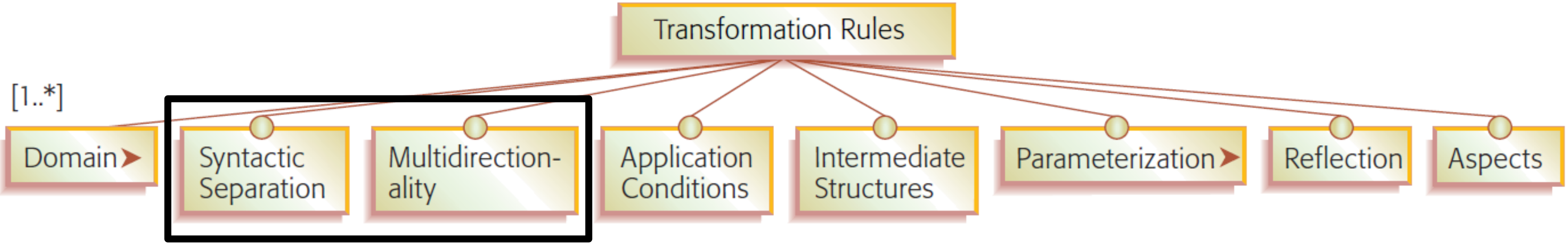
Execution of a declarative rule in ATL

1. Find all possible matches in the source model
2. Create elements specified in the target pattern on a target model
3. Initialize attributes and links of the newly created elements
4. Create **traceability** links from the elements in the source model matched by the source pattern to the created elements in the target model

- **Standard ATL rule** applied once for each match
  - Like FRule

Feature-Based Survey of Model Transformation Approaches



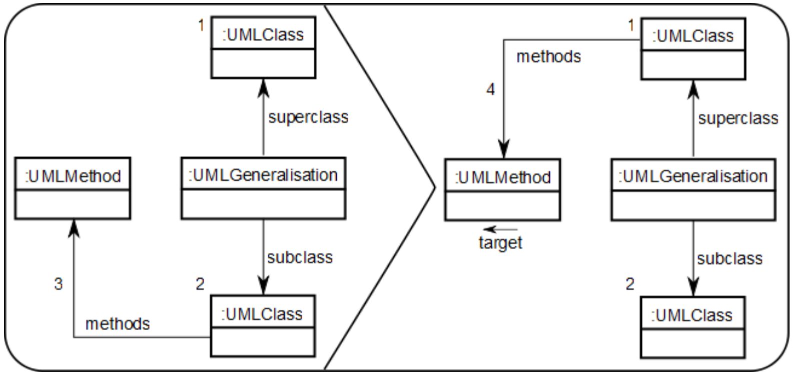


K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal:* 45(3), 621-645 (2006).

Rule patterns

- Model fragments
- Using abstract or concrete syntax
- Syntactic separation

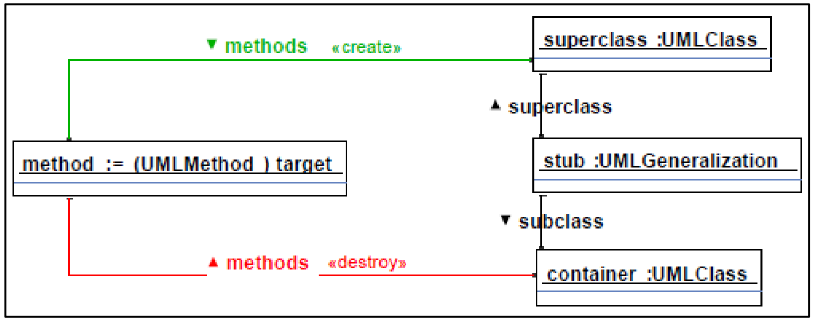
MoTif rule



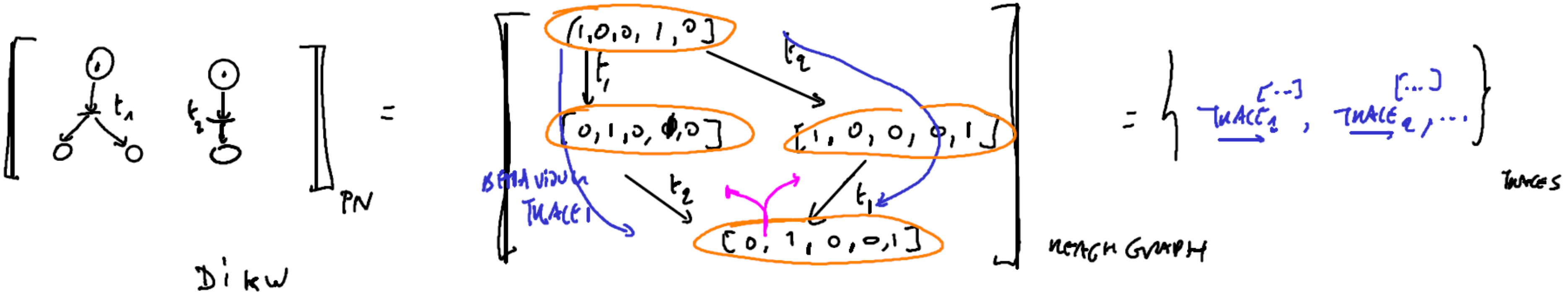
ATL rule

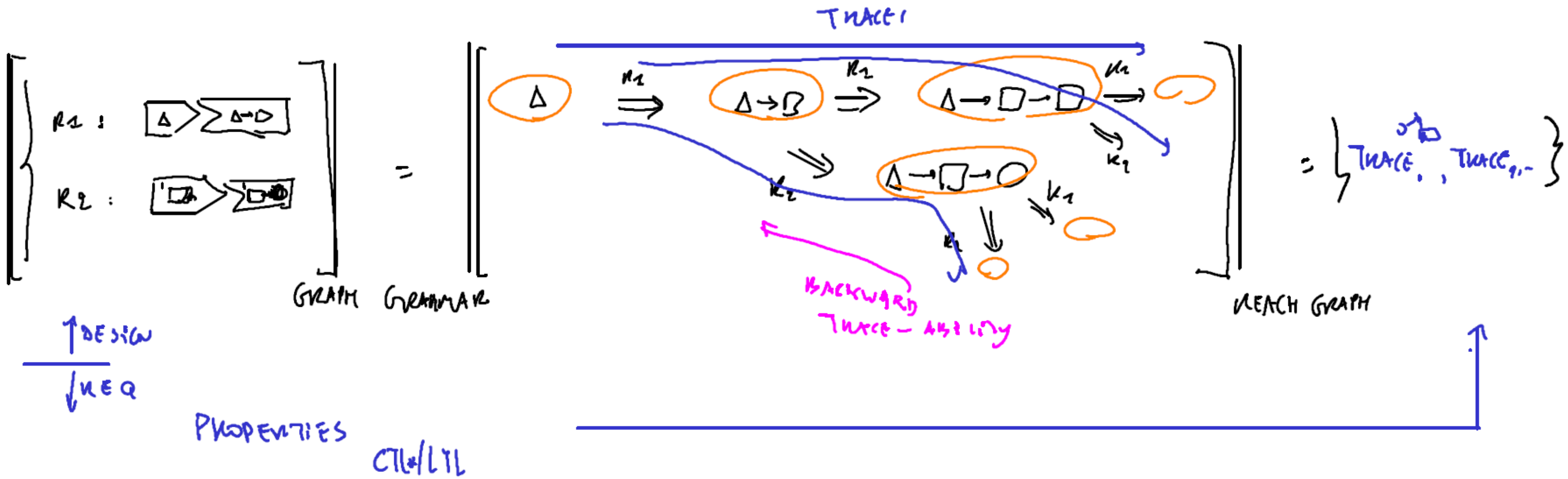
```
module Person2Contact;
create OUT: MMb from IN: MMa {

rule Start {
   form p: MMa!Person(
      p.function = 'Boss'
   )
   to c: MMb!Contact(
      name <- p.first_name + p.last_name)
}
```

FUJABA/Henshin compact notation



Choice → explore all possibilities → analysis over all traces



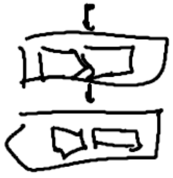


→ GROOVE MODEL CHECKER FOR (GT//GG ←→ ATOMPM

→ MAUDE " " " (") "

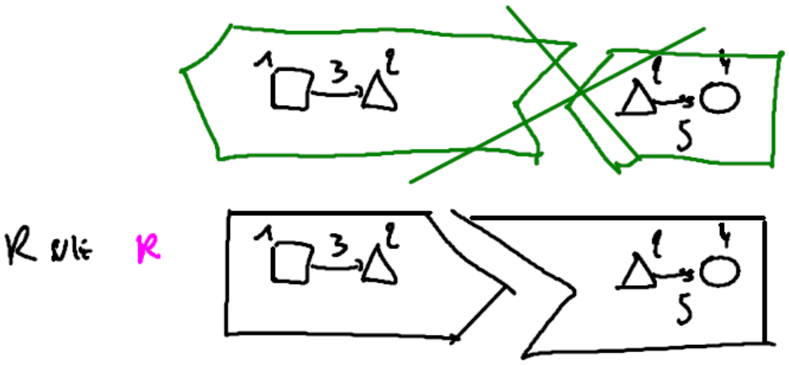
ANALYSIS

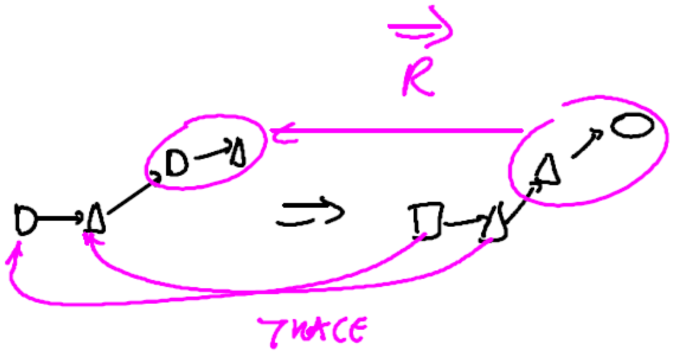
GT

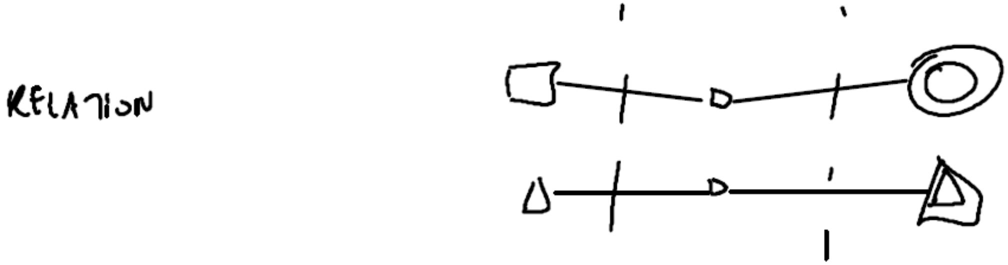


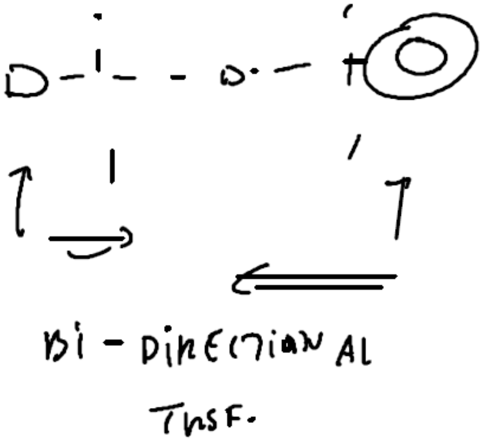
EXECUTION

Trace of Transformation **Execution** vs. **Bi-Directional** Transformations

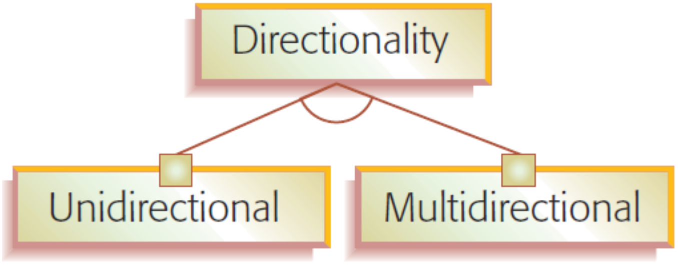




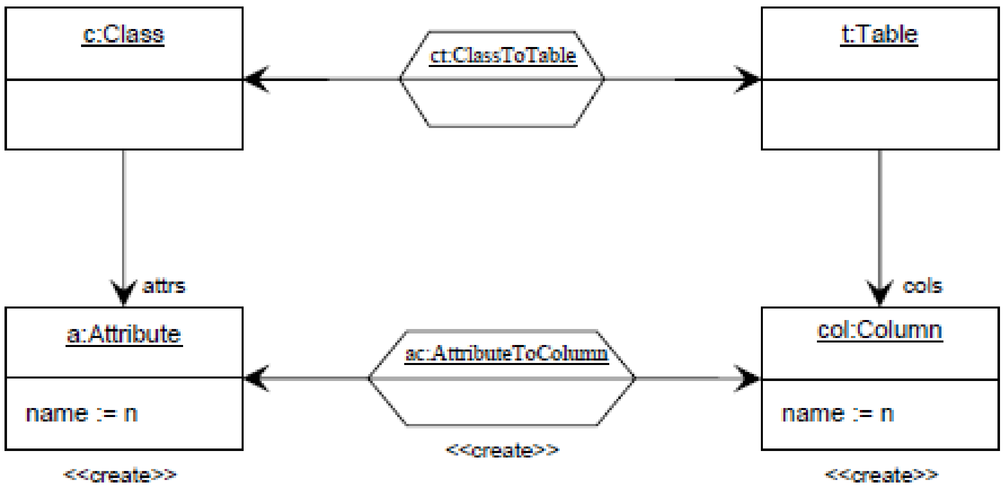


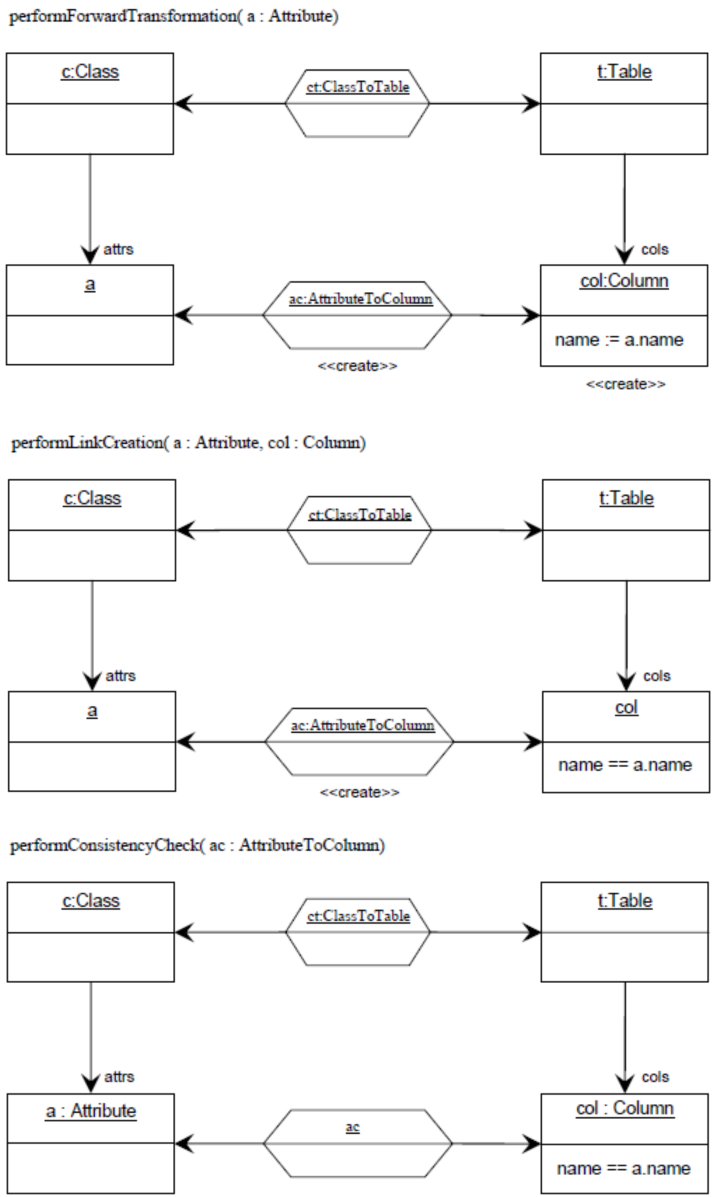


Multi-directional rules



TGG rule




performForwardTransformation(a : Attribute)

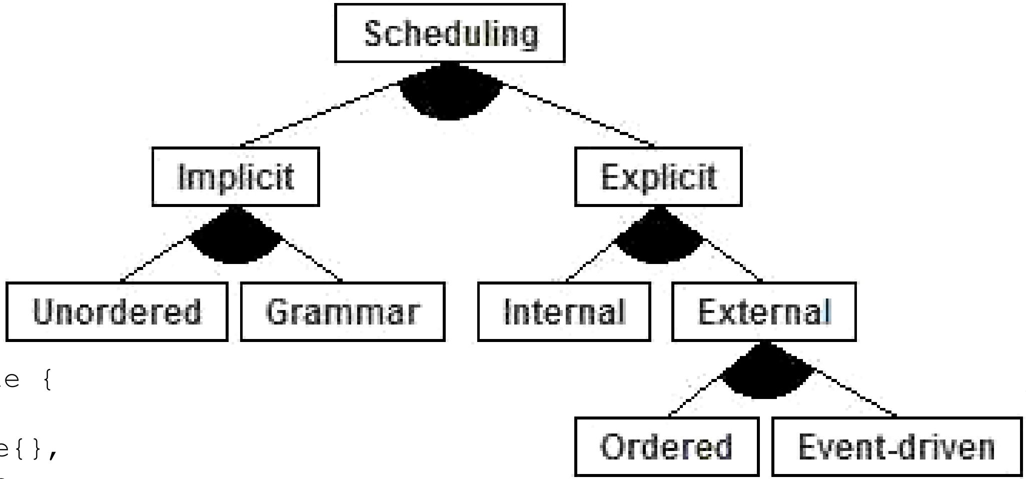
performLinkCreation(a : Attribute, col : Column)

performConsistencyCheck(ac : AttributeToColumn)

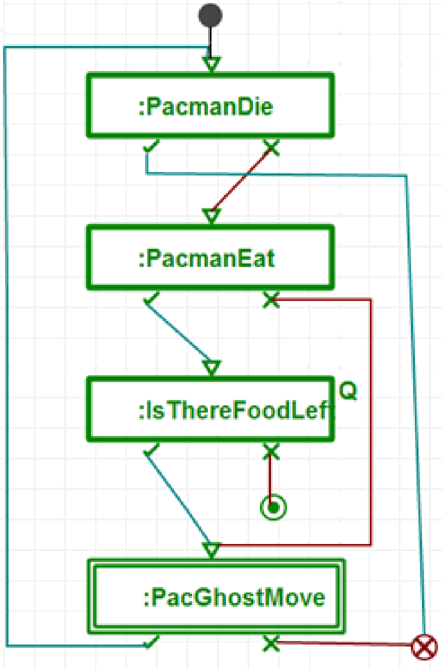
TGG operational rules

Rule scheduling strategies

Explicit



```
top relation ClassToTable {
   domain uml c:Class {
     package = p:Package{},
     isPersistent = true,
     name = cn
   }
   domain rdbms t:Table {
   schema = s:Schema{},
   name = cn,
   cols = cl:Column {
   name = cn + '_tid',
   type = 'NUMBER'},
   pkey = cl
   }
   when {
   PackageToSchema (p, s);
   }
   where {
   AttributeToColumn (c, t);
   }
}
```



Rule Scheduling (aka Control)

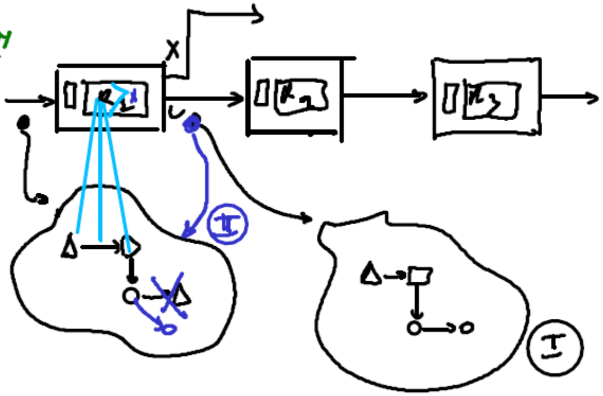
SCHEDULING LANGUAGES
+
RULE LANGUAGE
STRING / TREE / GRAPH

GRAPH
CBD / DATA FLOW

OPERATIONS : GRAPH TRSF.
SPECIFIED BY RULE

⊥ Hierarchy
+

1.



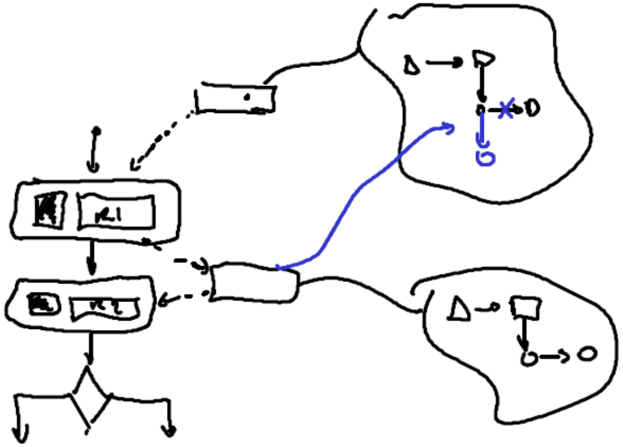
GREAT (GME)

I OUT
II IN
PLACE

2.

ACTIVITY DIAGRAM

CONTROL FLOW

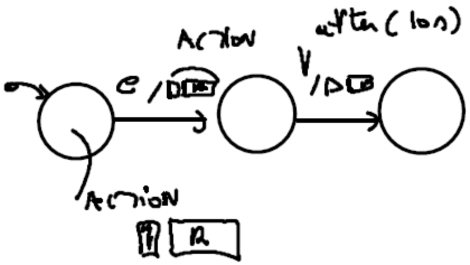


FUJABA
"this"
PIVOT
FIRST MATCH

Rule Scheduling (aka Control)

3. TIMED STATE AUTOMATA

STATECHARTS

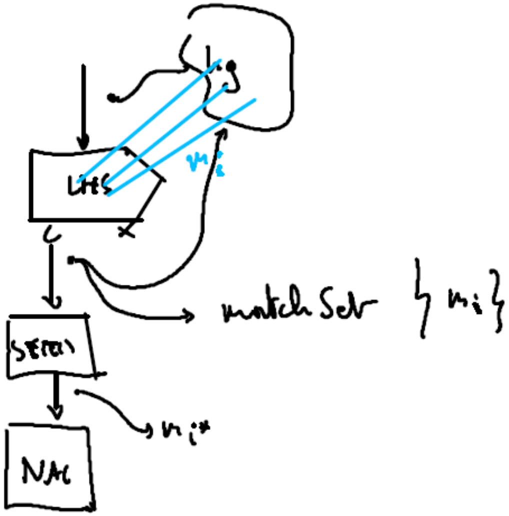


TIMED MT

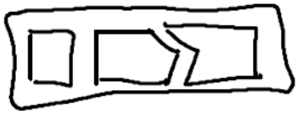
CONCURRENCE

4. DEVS

- TIME
- HIERARCHY
- DEV
- CONCURRENCY



motif



A RULE

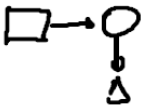
Rule Scheduling (aka Control)

3. PROGRAMMING LANGUAGE

GREAT FUJABA MOTIF

COMPILER

```
hg = GRAPH()
LHSrule = LHSRULE()
matcher = MATCHER()
matchSet = matcher.match(LHSrule, hg)
match = select(matchSet)
    ...
    REWRITER()
```



NAC LHS RHS

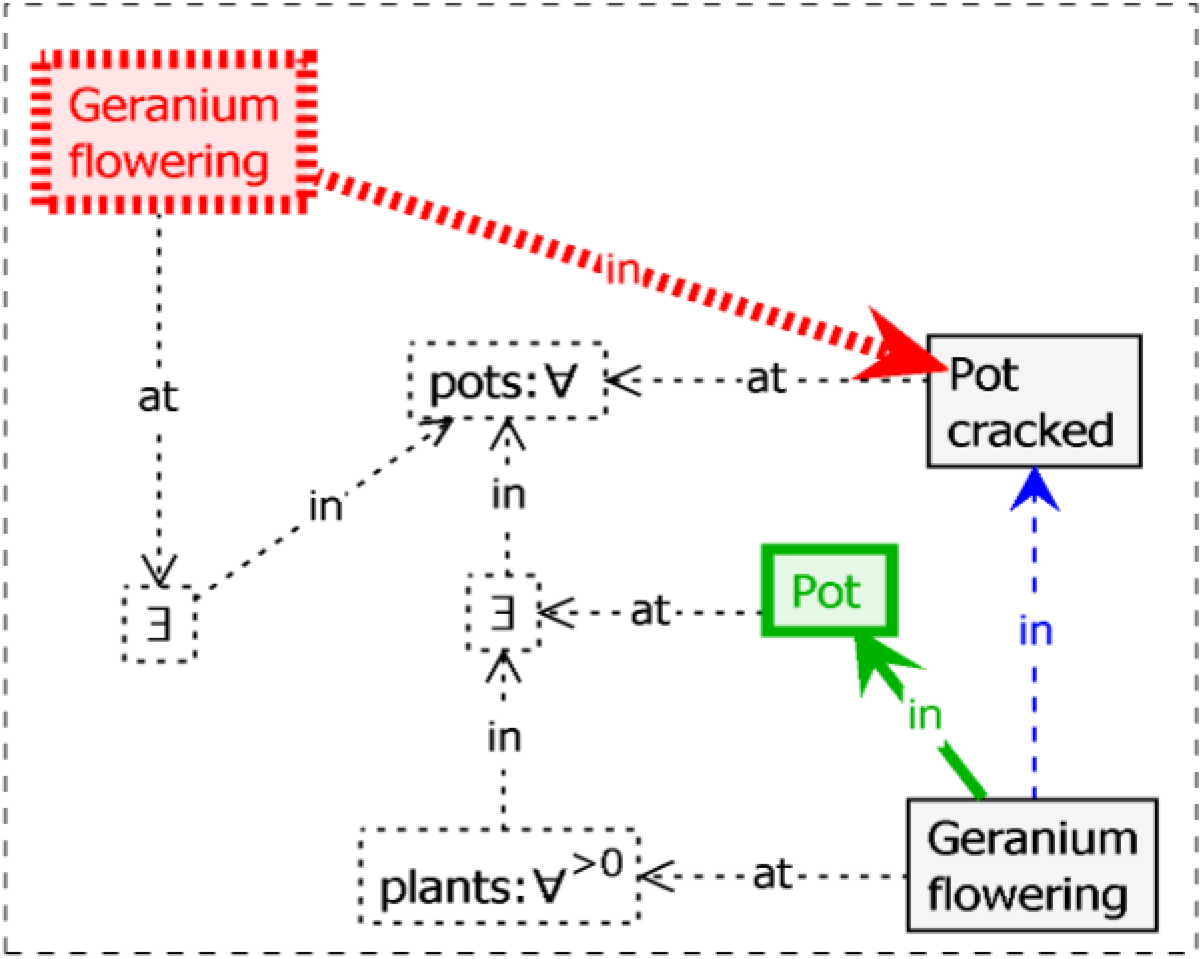
T-Core

Increased Expressiveness: rule amalgamation

Arend Rensink and Jan-Hendrik Kuperus. *Repotting the Geraniums: On Nested Graph Transformation Rules.* Graph Transformation and Visual Modeling Techniques (GT-VMT). In ECEASST Volume 18. 2009.

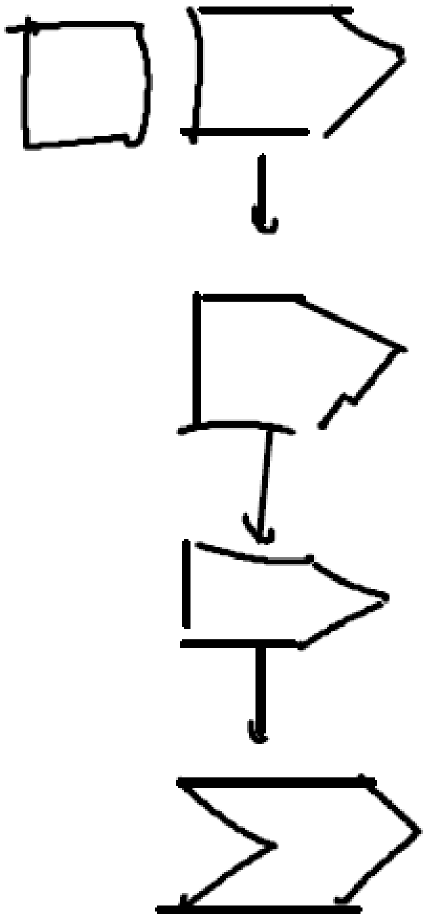
https://journal.ub.tu-berlin.de/eceasst/article/view/260

We have a number of flower pots, each of which contains a number of geranium plants. These tend to fill all available space with their roots, and so some of the pots have cracked. For each of the cracked pots that contains a geranium that is currently in flower, we want to create a new one, and moreover,to move all flowering plants from the old to the new pot. Create a single parallel rule that achieves this in a single application, without the use of control expressions.

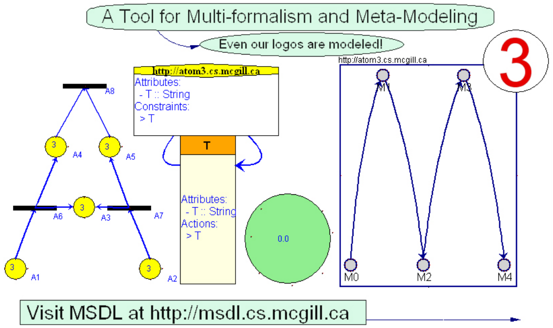


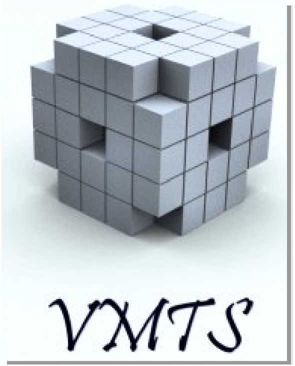
Increased Expressiveness: rule amalgamation

Operationally (in terms of T-Core building blocks):
Match – Match - … - Re-Write



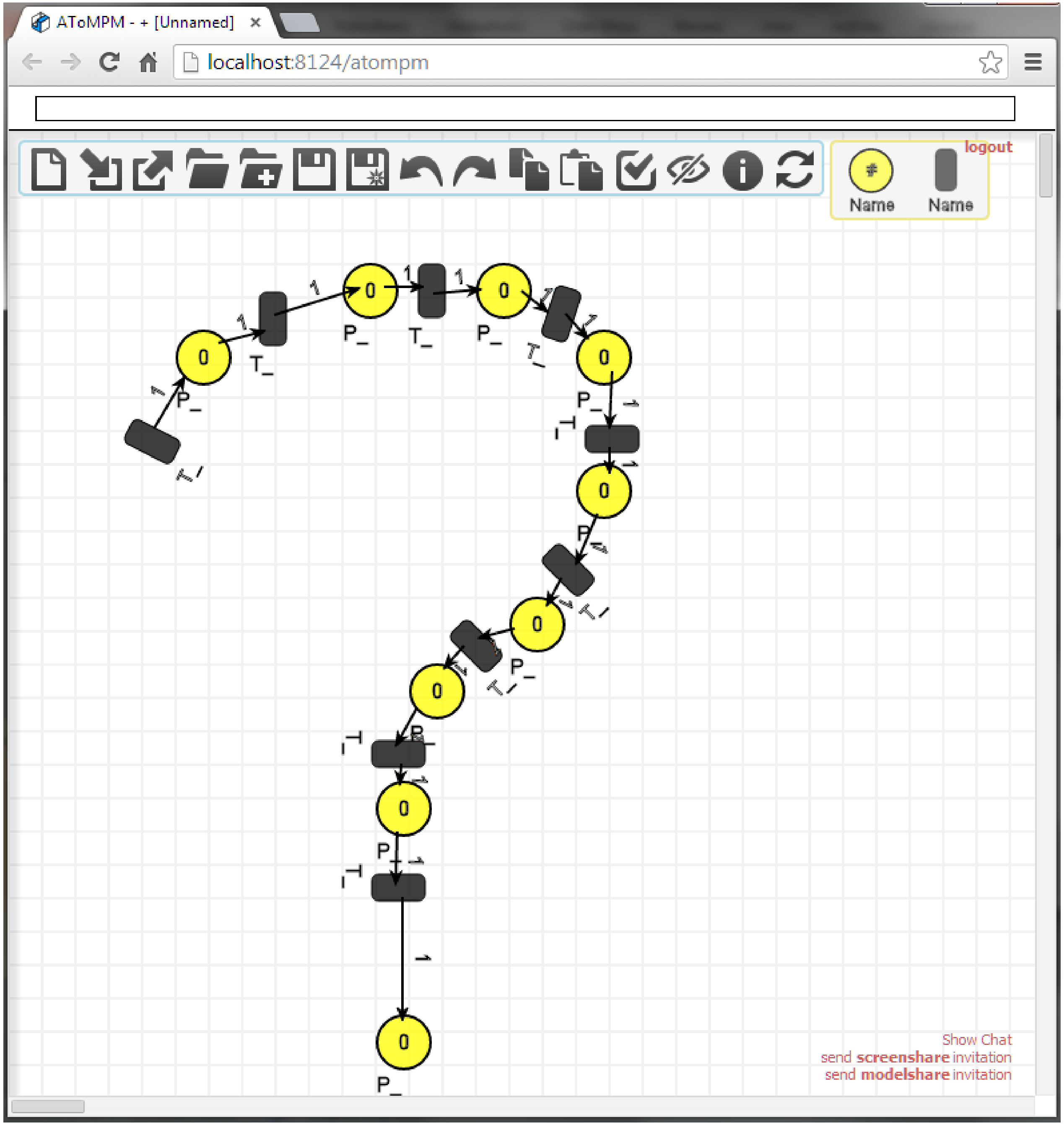
Plethora of model transformation languages


A Tool for Multi-formalism and Meta-Modeling
Even our logos are modeled!
3
Visit MSDL at http://msdl.cs.mcgill.ca



GReAT DSLTrans ProGReS MOLA

Model Transformation



AToMPM - + [Unnamed]
localhost:8124/atompm
logout
Name
Name
0
P_
T_
1
Show Chat
send screenshare invitation
send modelshare invitation