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

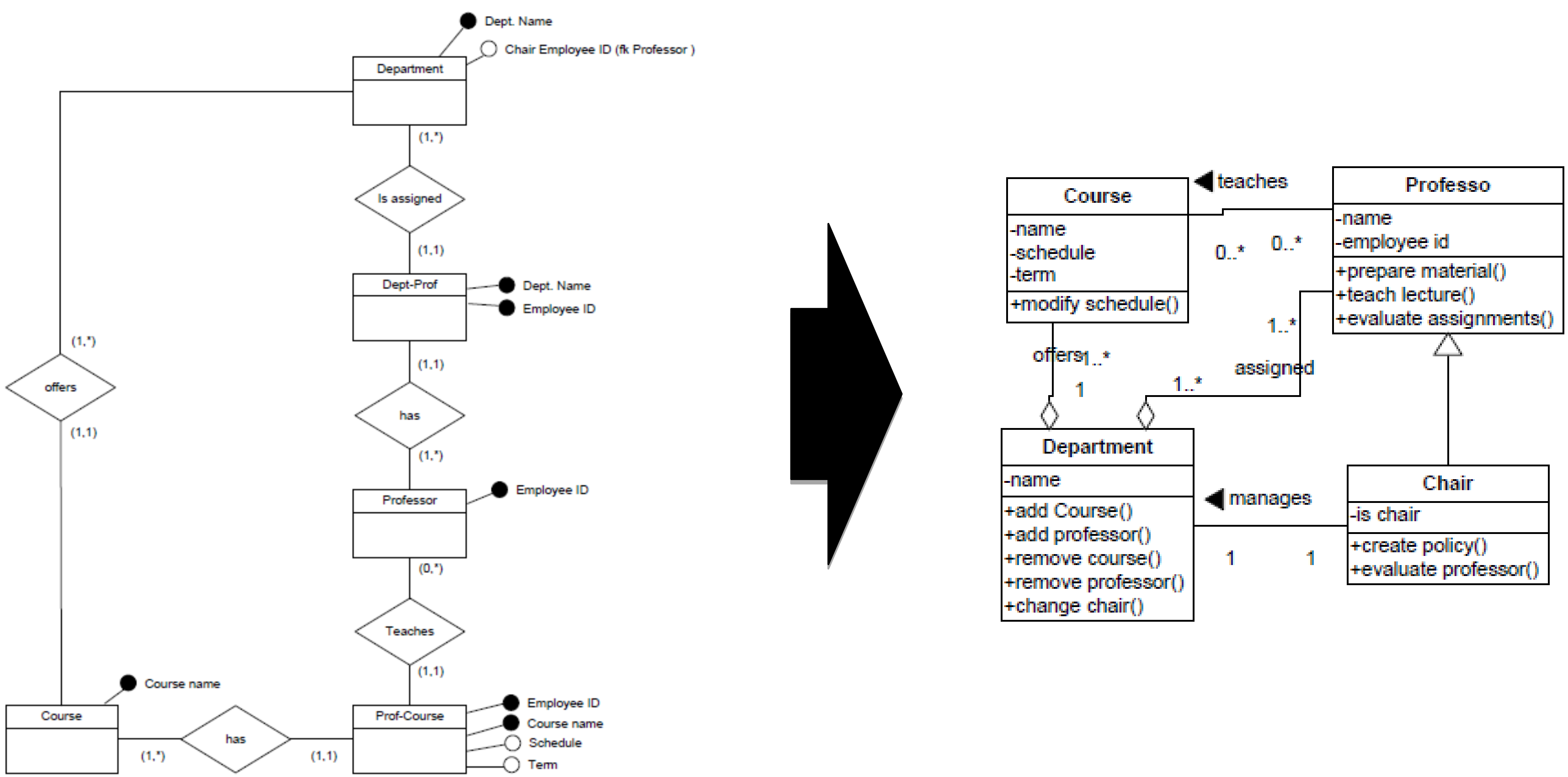
Software Modeling & Simulation

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” instead of “how”

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

What’s the difference?

- Typically encounter the **same problems** (need for documentation, testing, debugging, …) in modeling as in programming
- 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**

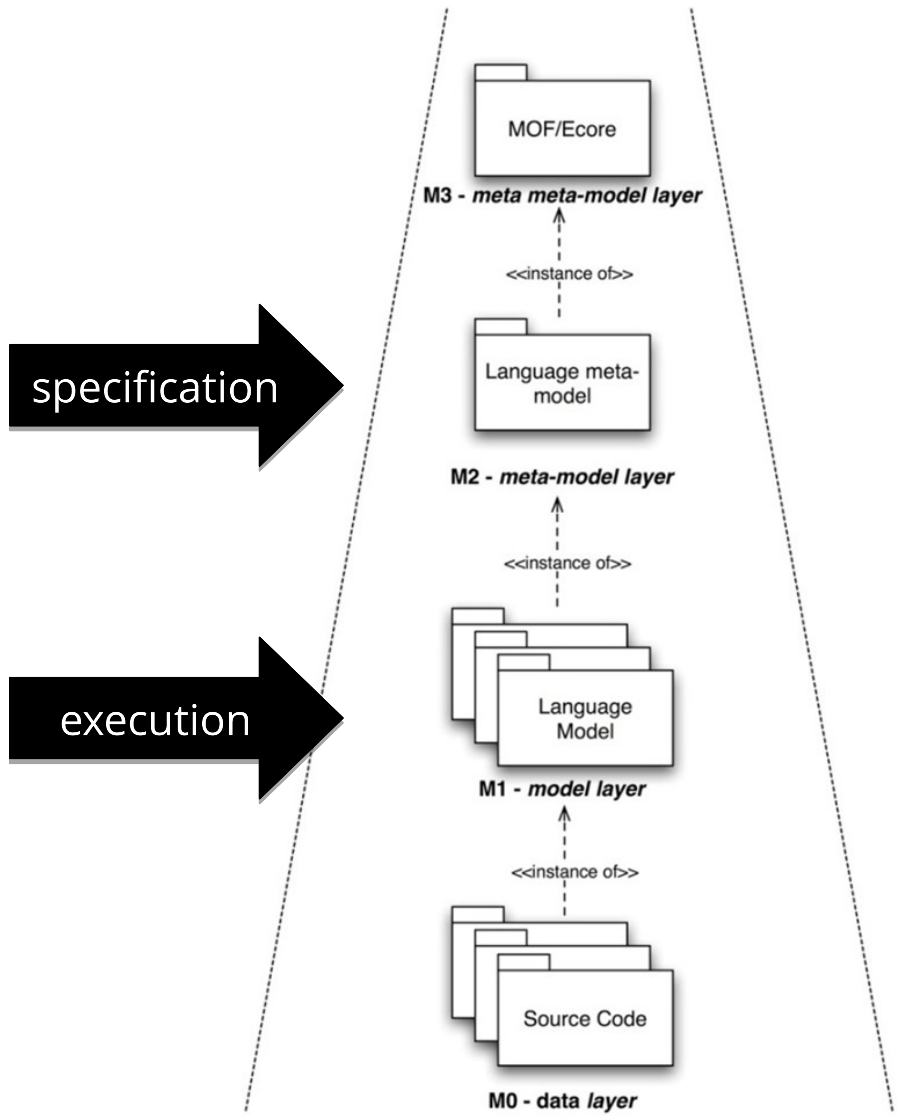
What is a model transformation?

Definition

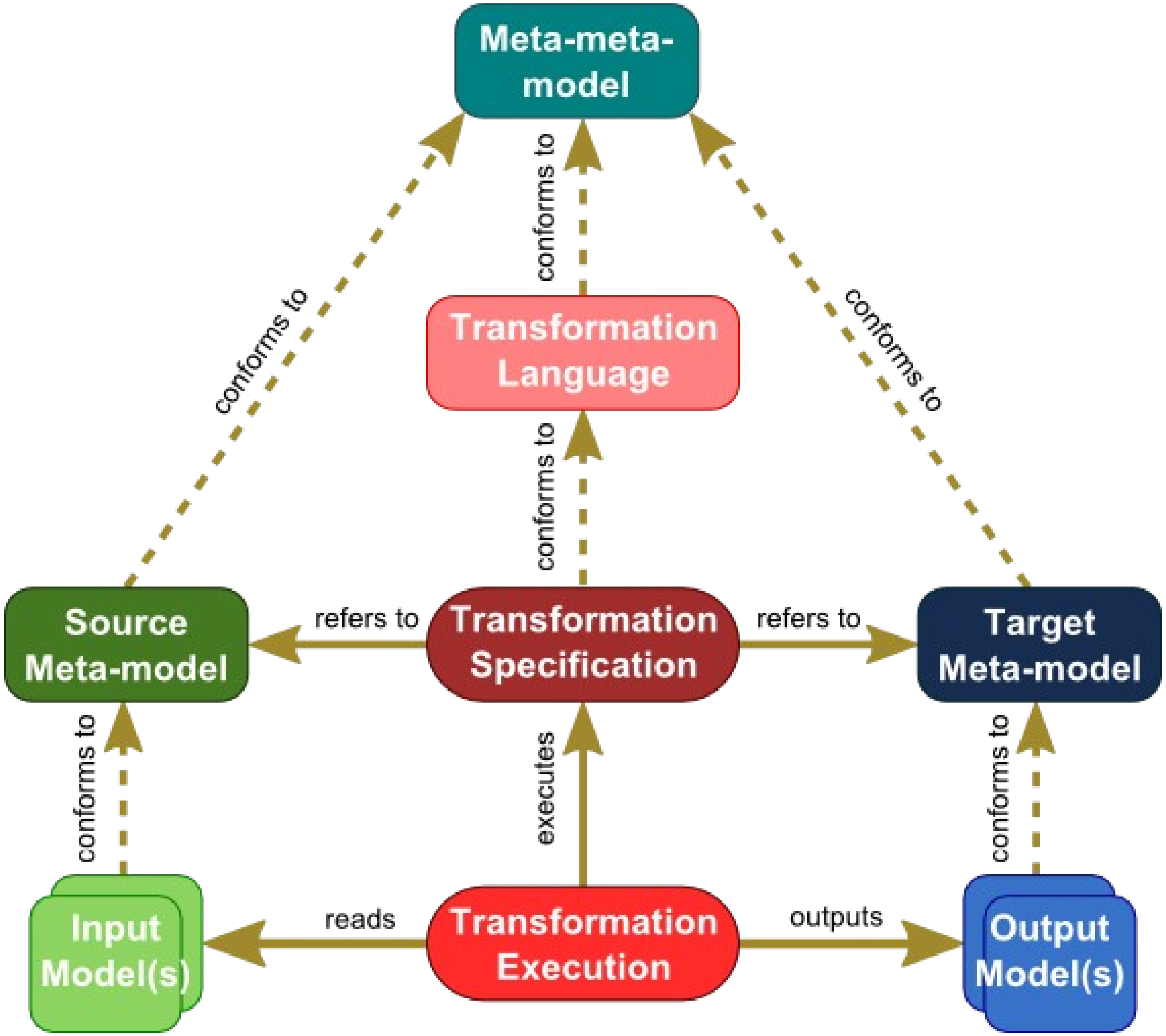
A model transformation is the **automatic** manipulation of **input** models to produce **output** models, that conforms to a **specification** and has a specific **intent**.

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



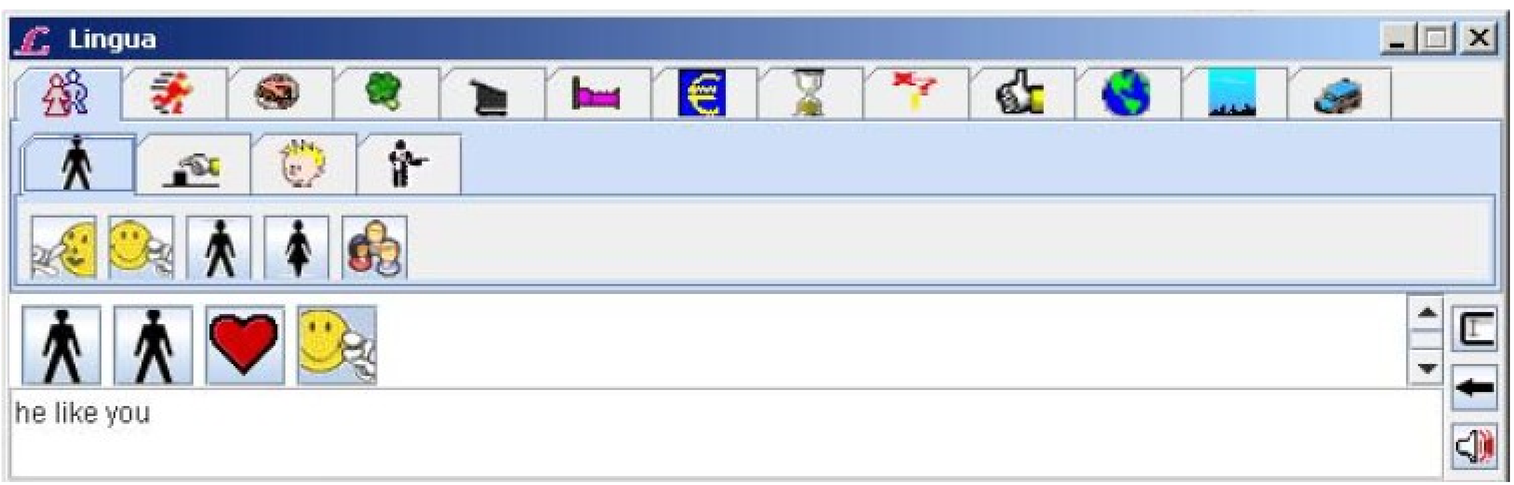
(= Source)

(= Input)

Data structures to transform

Sequence

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



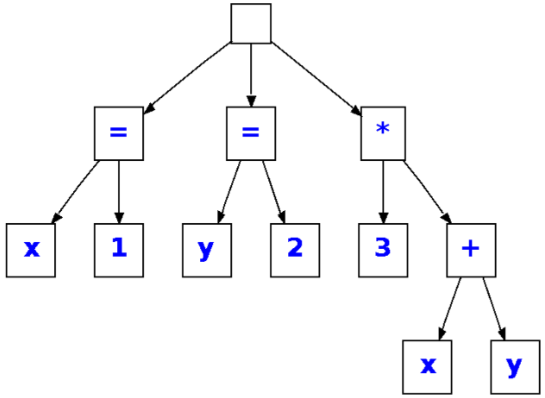
String rewriting

- 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**




```xml
<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
```

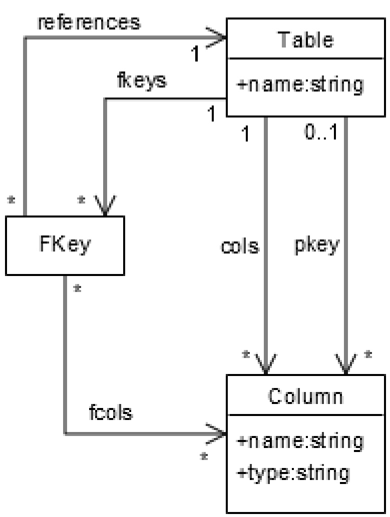
- Model transformation
  - Transformation language is **term rewriting** with production rules

```
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

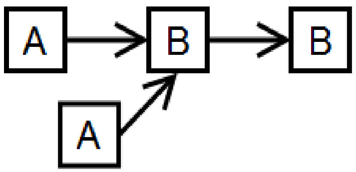
Graph

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

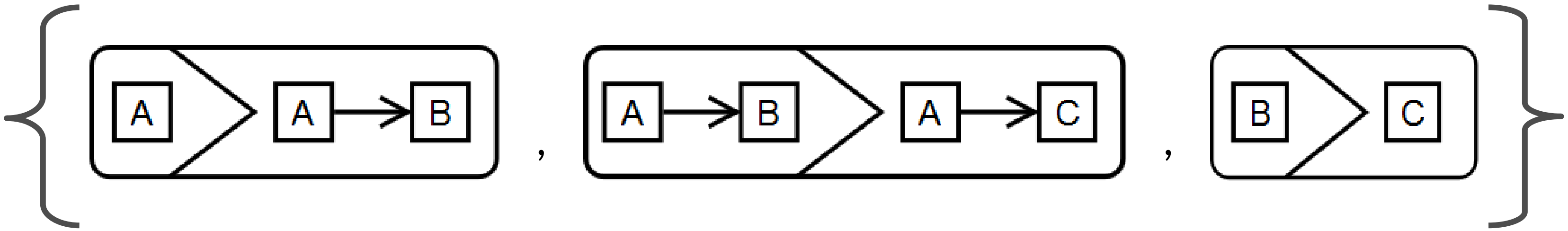


Graph transformation

- 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

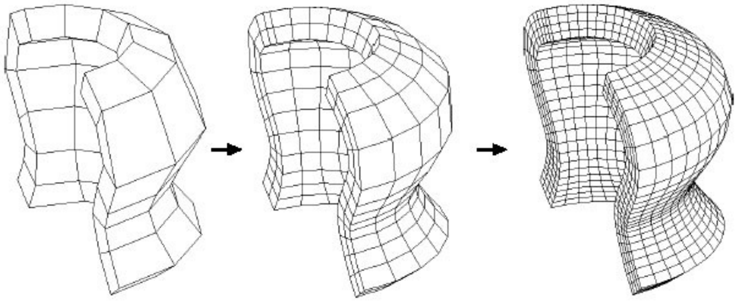
Model Composition
- Model Merging
- Model Matching
- Model Synchronization

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

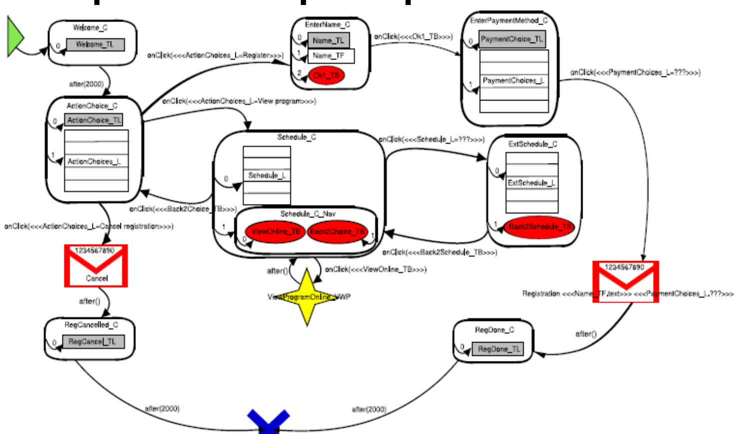
Groups intents that produce a more precise model by reducing design choices and ambiguities with respect to a target platform.

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

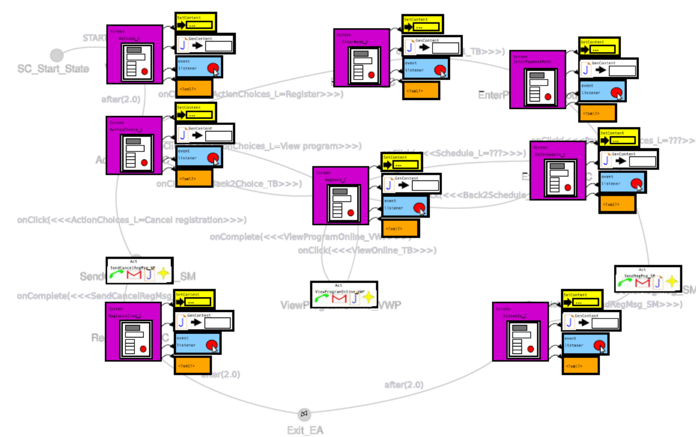


Refinement

- 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



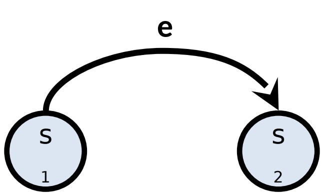
Representation of the model in AndroidAppScreens

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

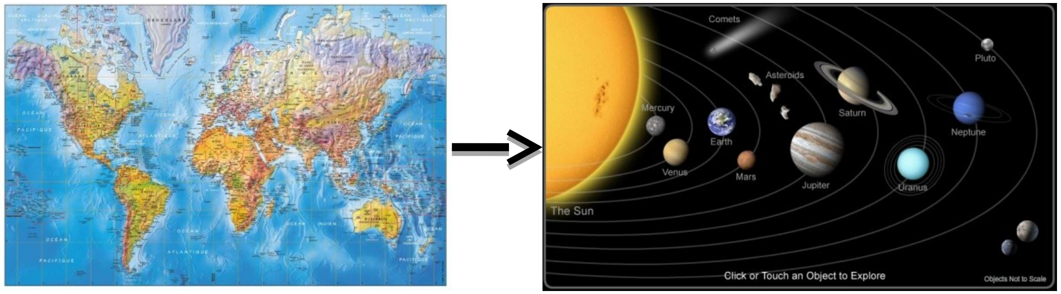
Statecharts to Python Compiler

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

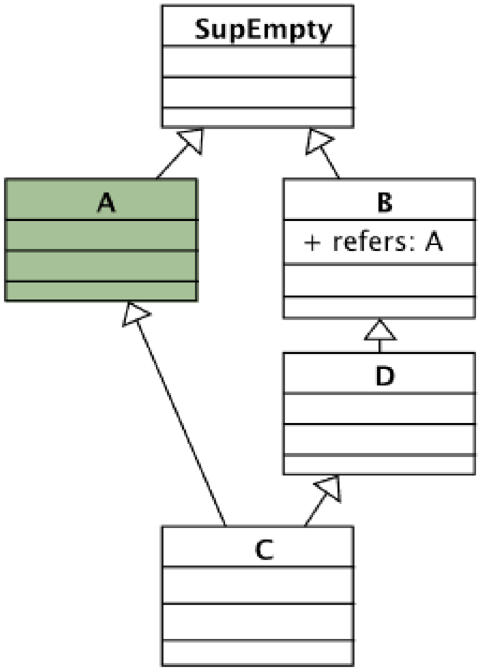
- A query requests some information about a model and returns that information in the form of a proper sub-model or a view
  - Projection of a sub-set of of the properties of M
  - View of a model that is not a sub-model, but an aggregation of some of its information is also a abstraction
- Example: "*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")));
}
```



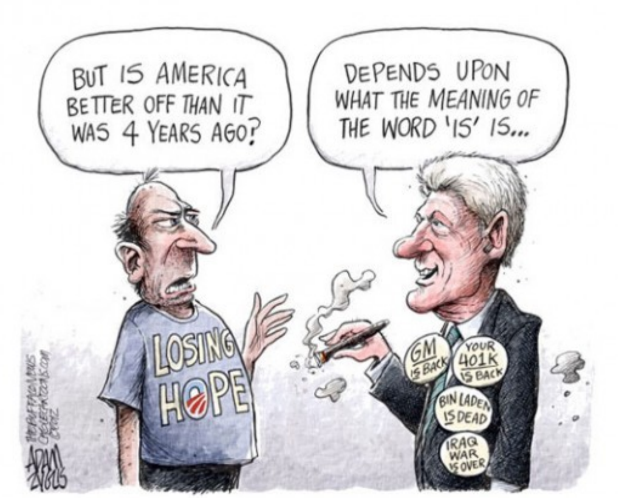
Z. Ujhelyi, et al. EMF-IncQuery: An integrated development environment for live model queries. *Science of Computer Programming:* 98, 80-99 (2015).

Semantic Definition category

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

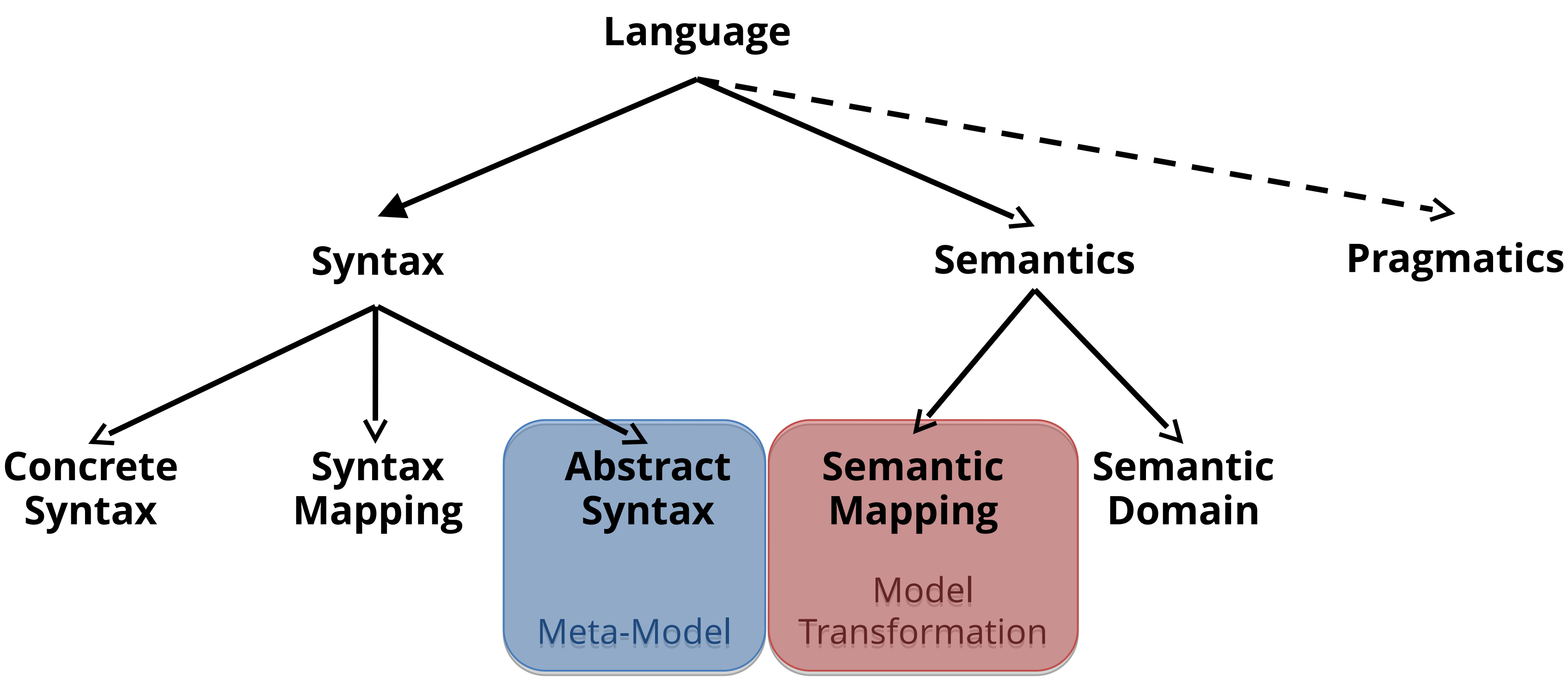
- Translational Semantics (model-to-model)
- Operational Semantics

  (simulation by graph transformation)



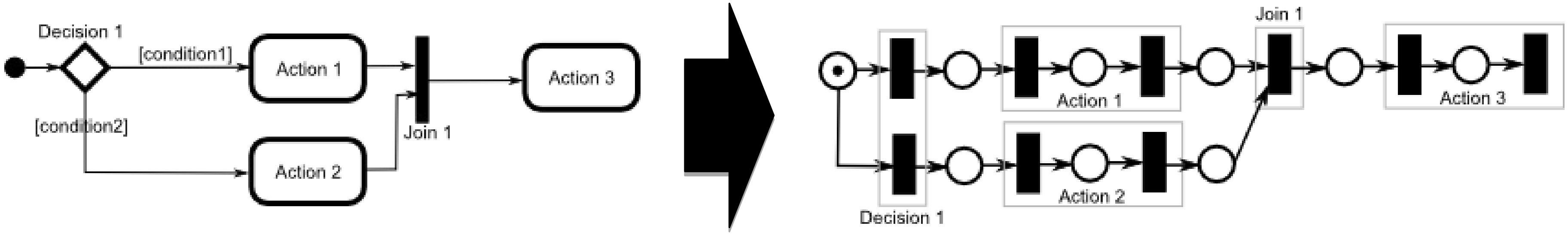
Translational Semantics

- 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

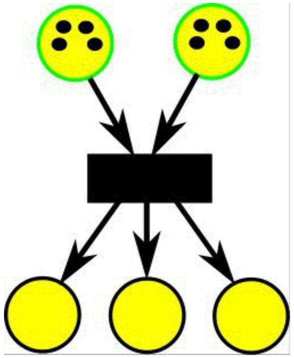
- Simulink Block Diagram’s semantics expressed as Ordinary Differential Equations
- UML activity diagrams semantics expressed as Petri nets



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).

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

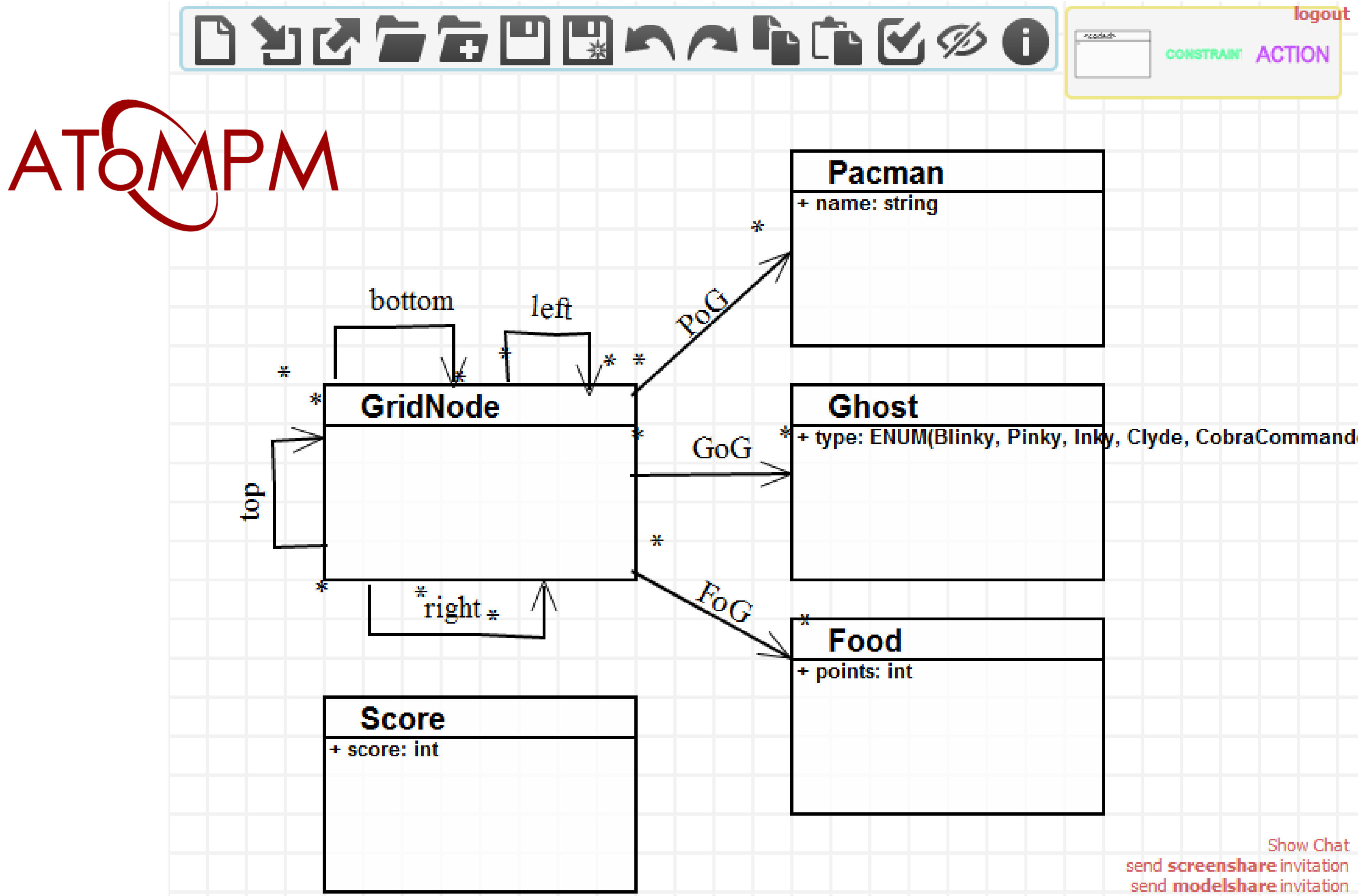
Rule-based model transformation

Graph transformation for simulation

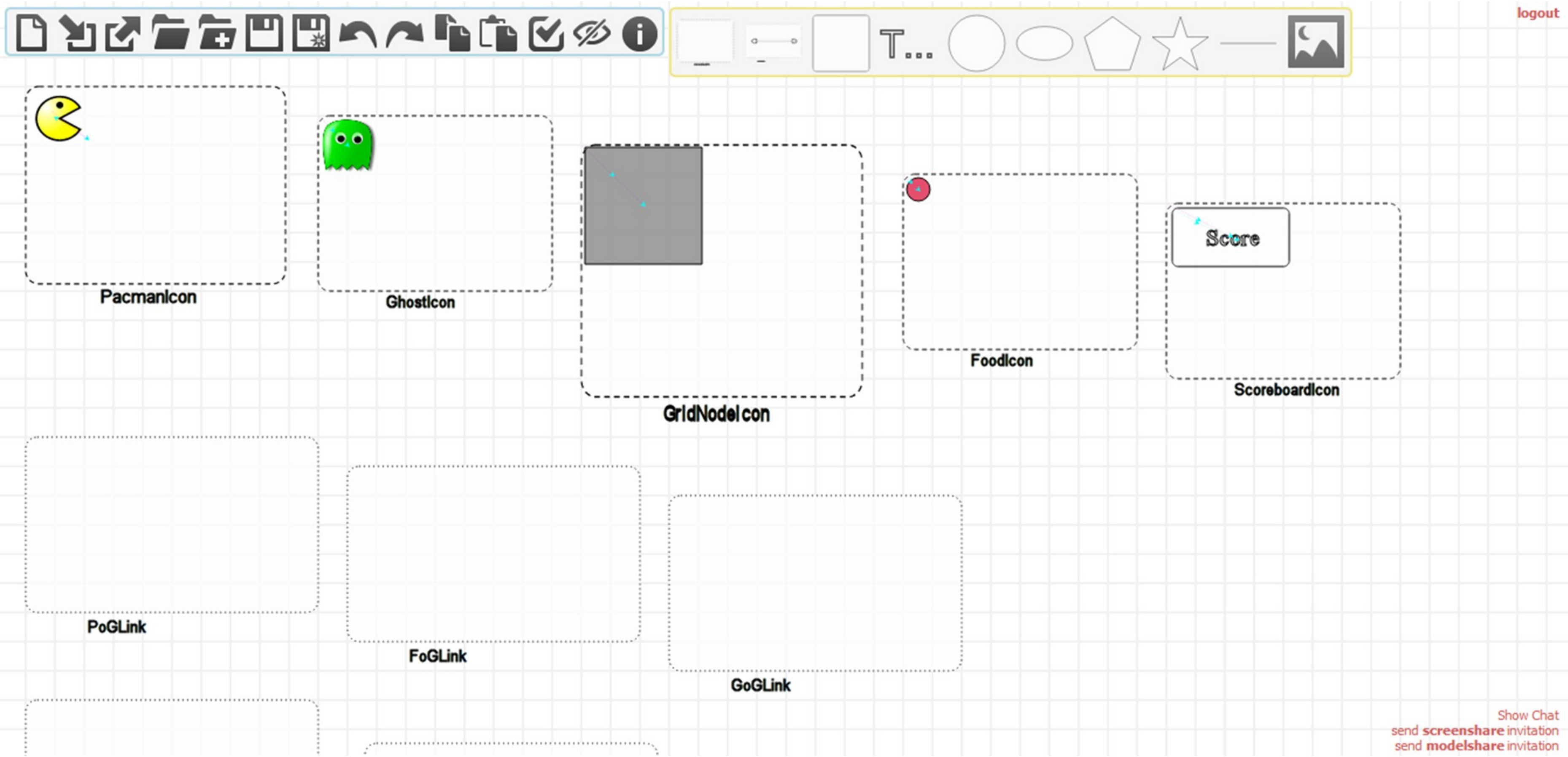
- Models are considered as directed, typed, attributed graphs
- Transformations on such graphs are considered as graph rewritings
- Features:
  - Declarative paradigm
  - Rules defined as pre- and post-conditions

- Tools: **MoTif**, Henshin, GReAT

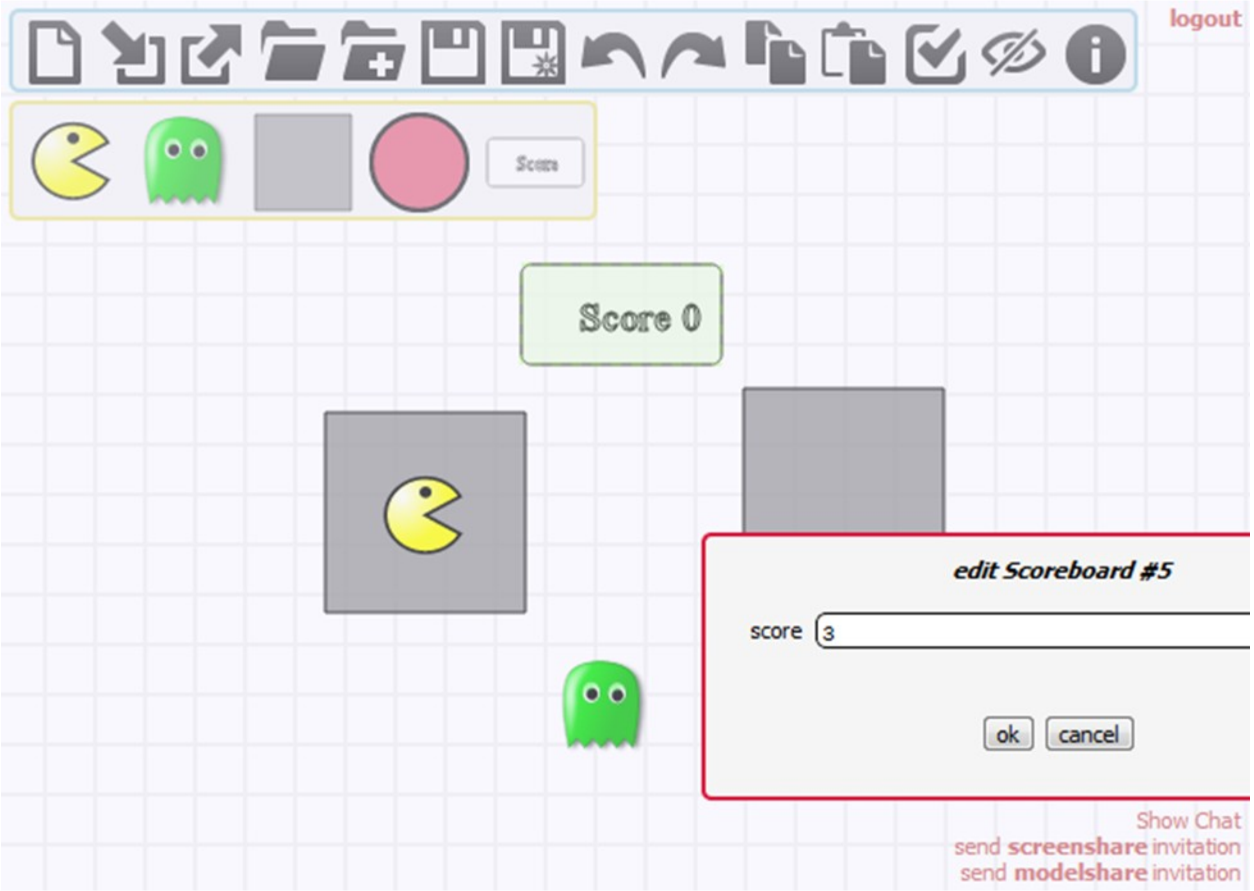
Metamodel of Pacman



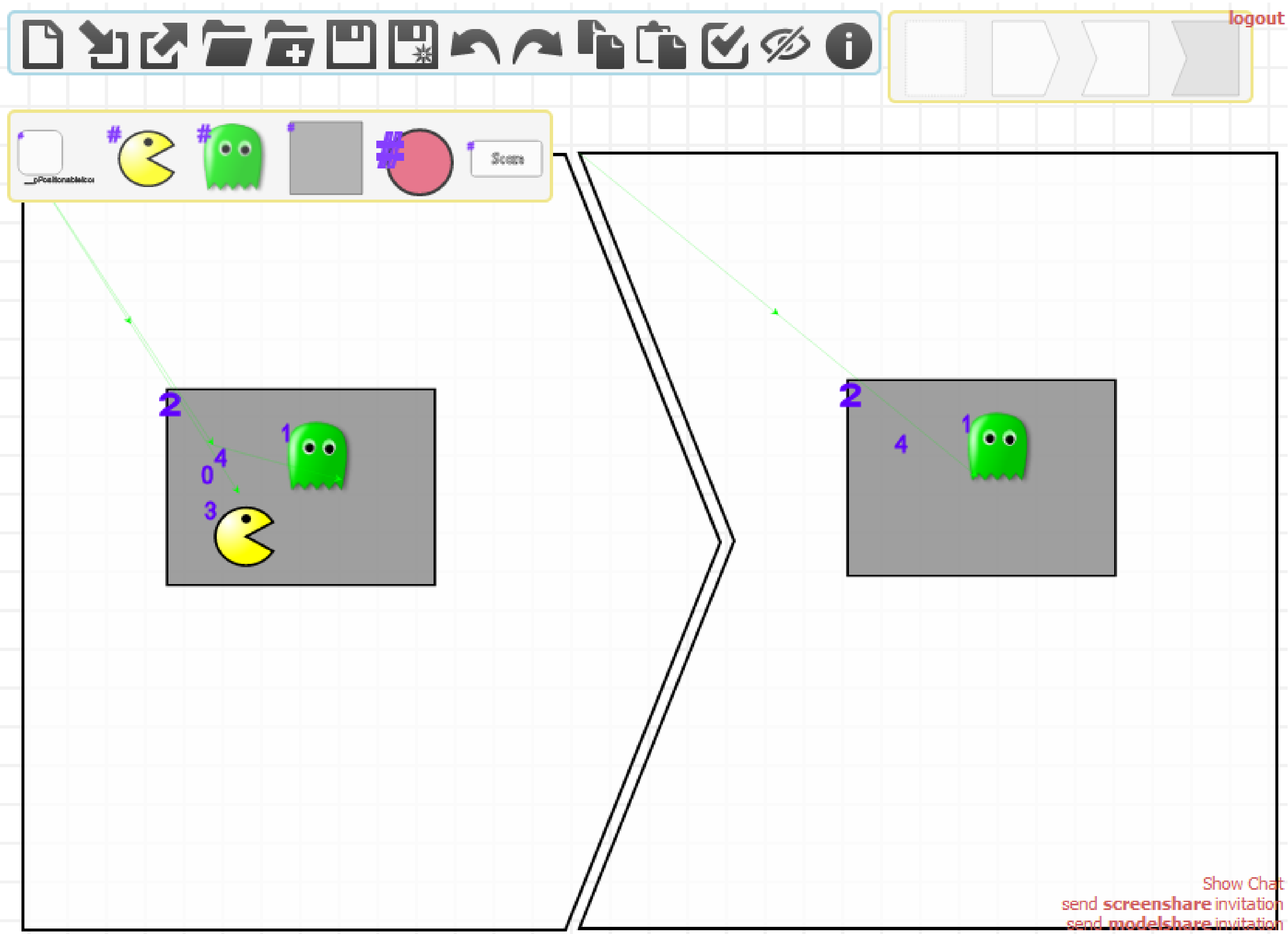
Concrete syntax



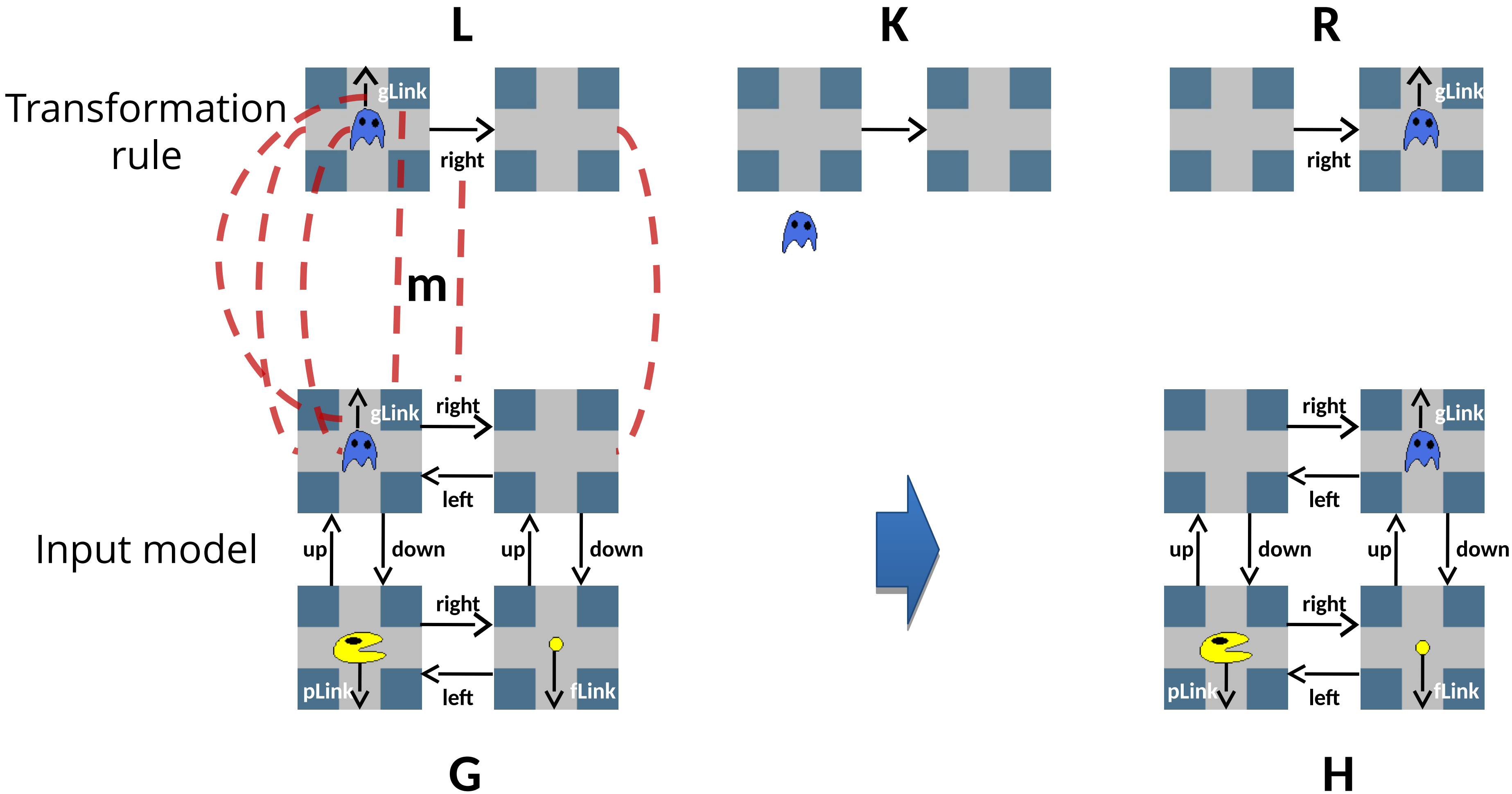
Generate modeling environment



Graph transformation rule



Rule-based graph transformation



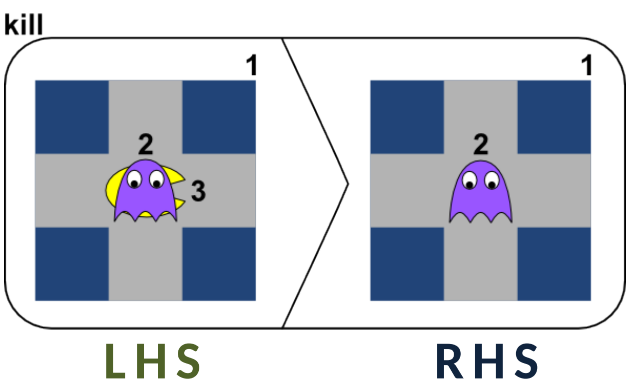
If there exists an occurrence of **L** in **G** then replace it with **R**

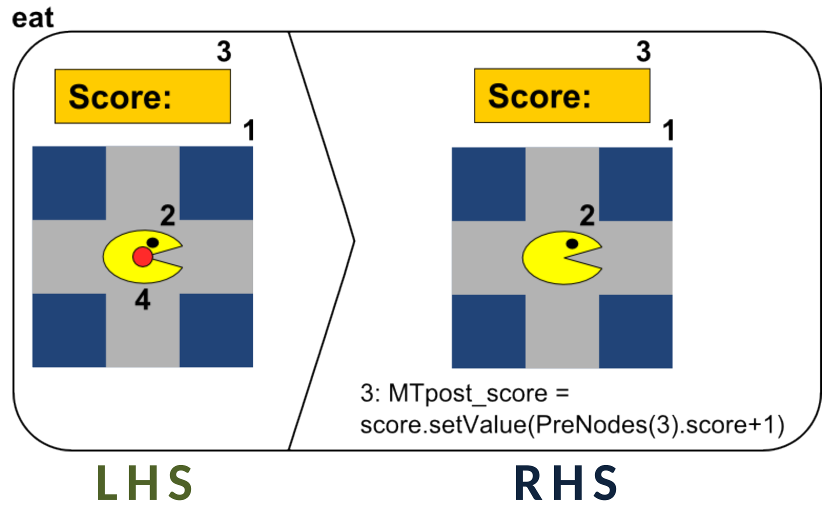
Mechanics of rule application

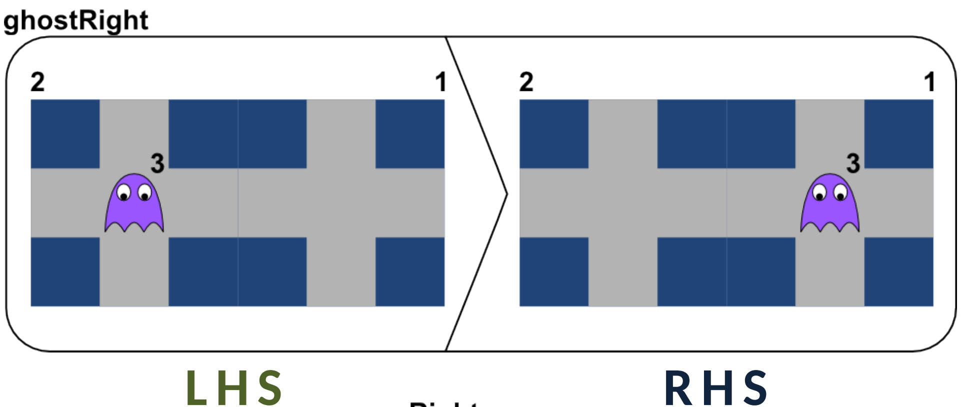
1. Matching Phase
   - Find an embedding m of the LHS pattern L in the host graph G
   - An occurrence of L is called a **match**: $m(L)$
   - Thus, $m(L)$ is a sub-graph of G
2. Rewriting Phase
   Transform G so that it satisfies the RHS pattern:
   - **Remove** all elements from $m(L - K)$ from G
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

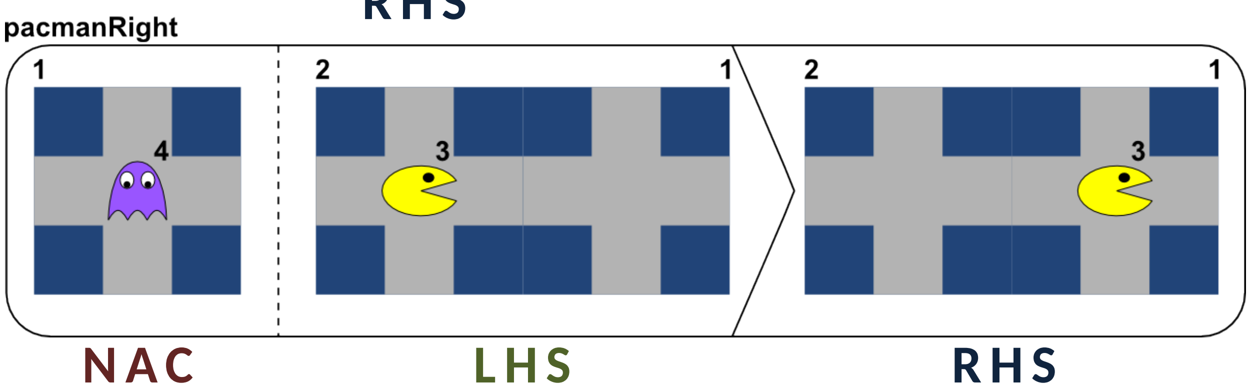
- When a match of the LHS can be found in G, the rule is **applicable**
- When the rewriting phase has been performed, the rule was **successfully applied**

Operational semantics



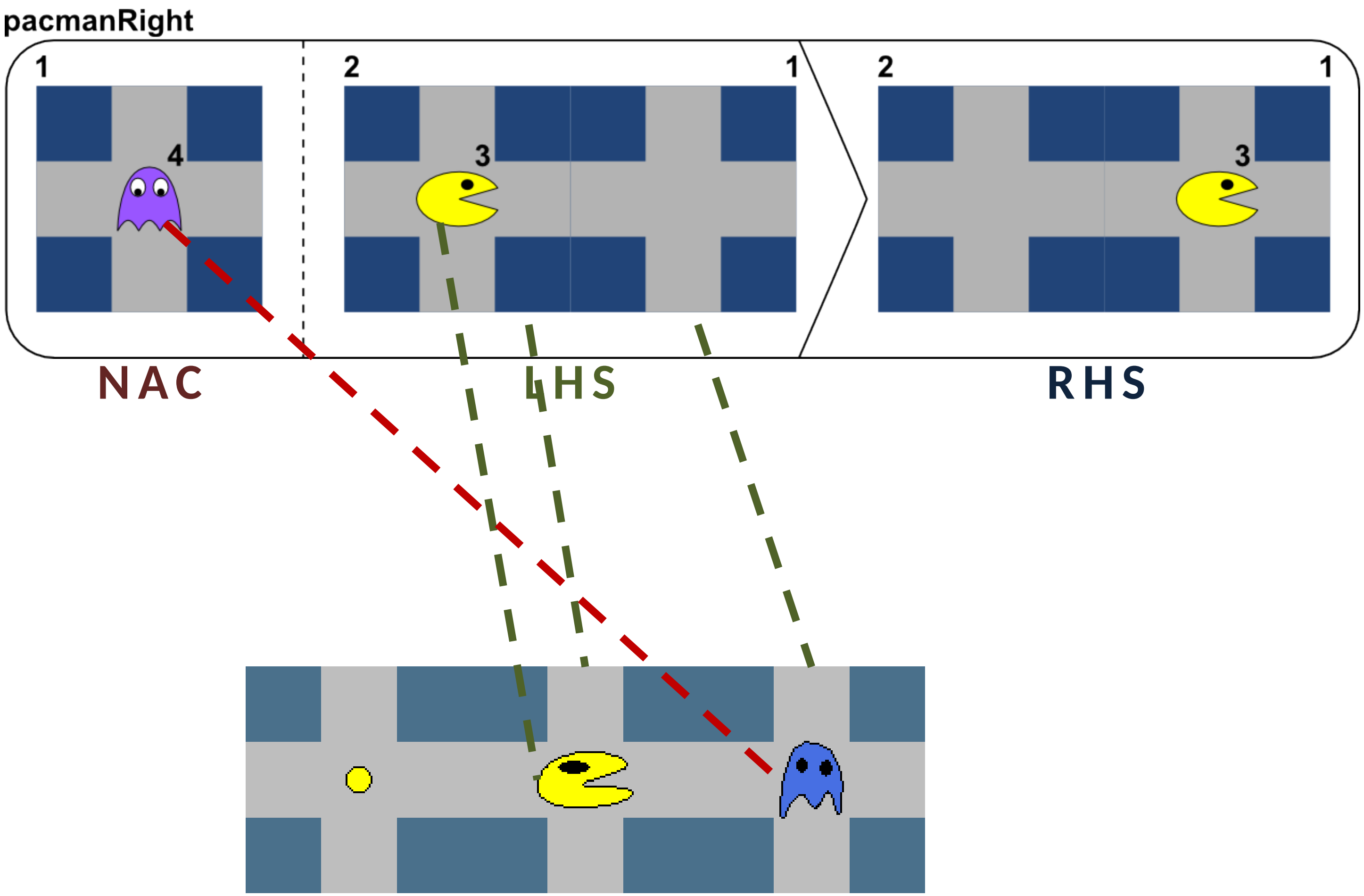






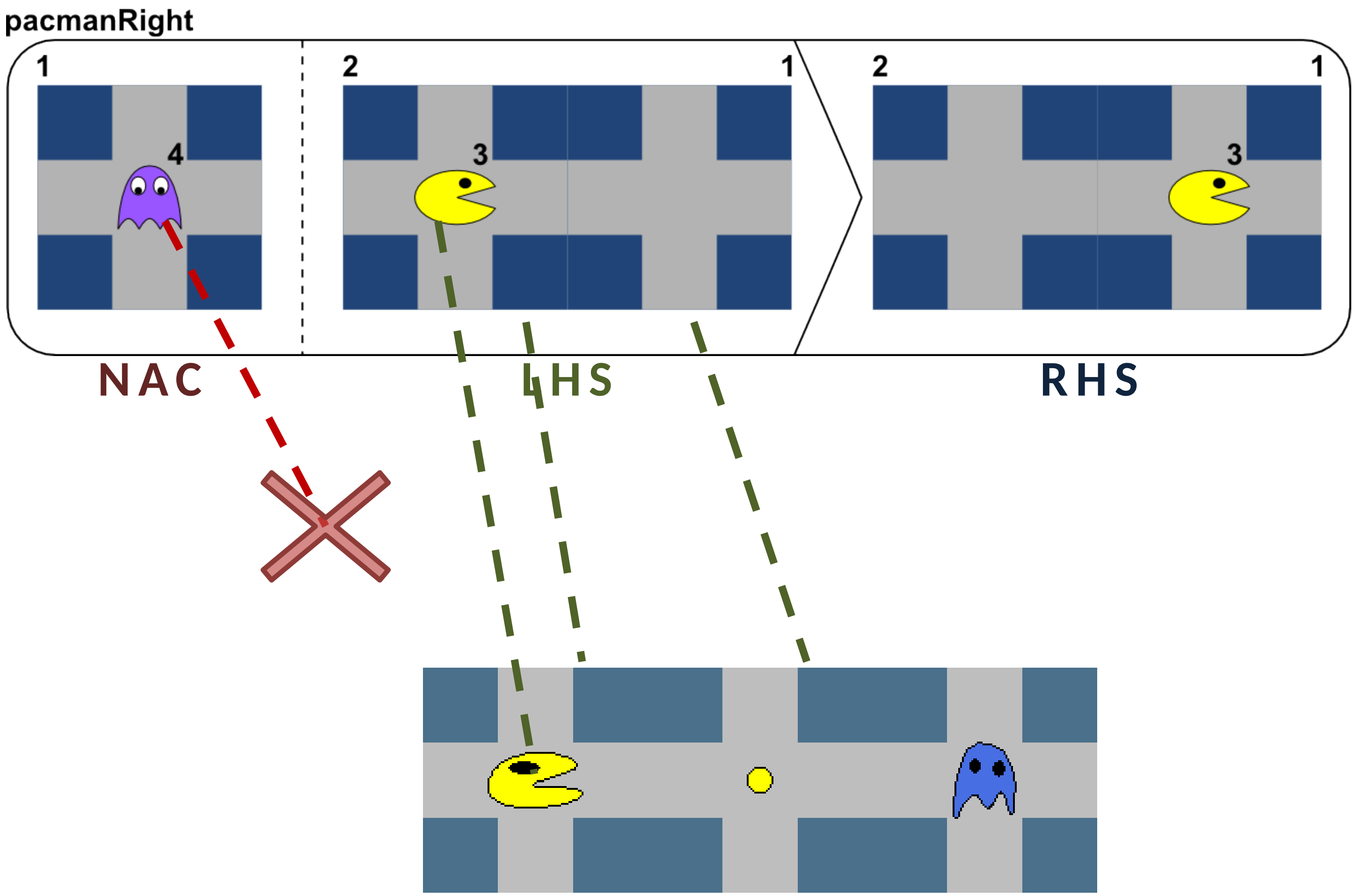
Negative application conditions

Non-applicable rule



Negative application conditions

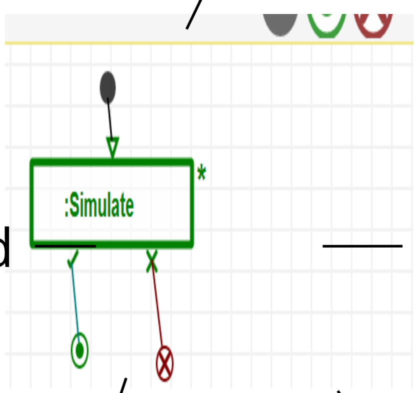
Applicable rule



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

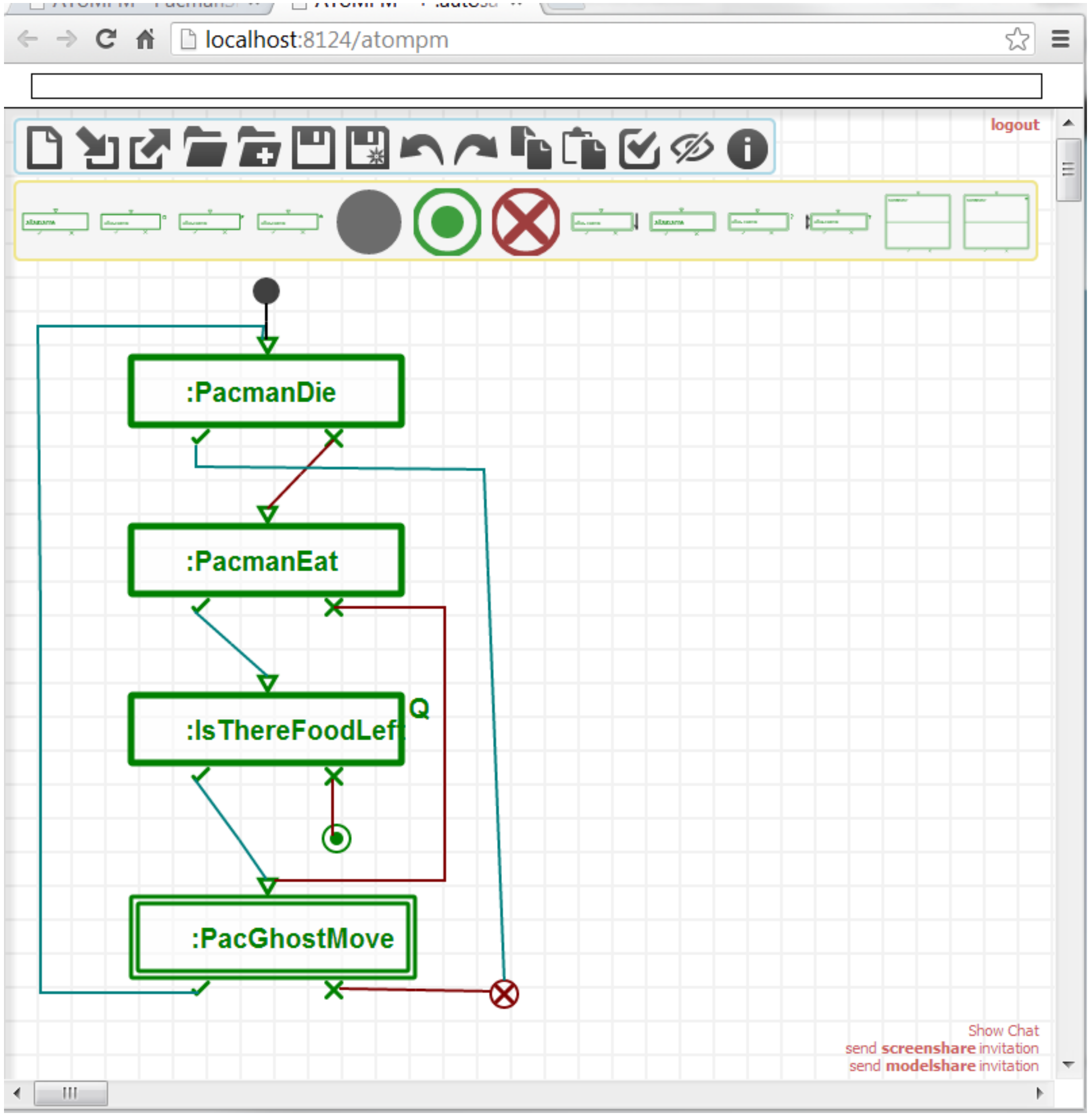
If match found

If no match found

Terminate transformation in success

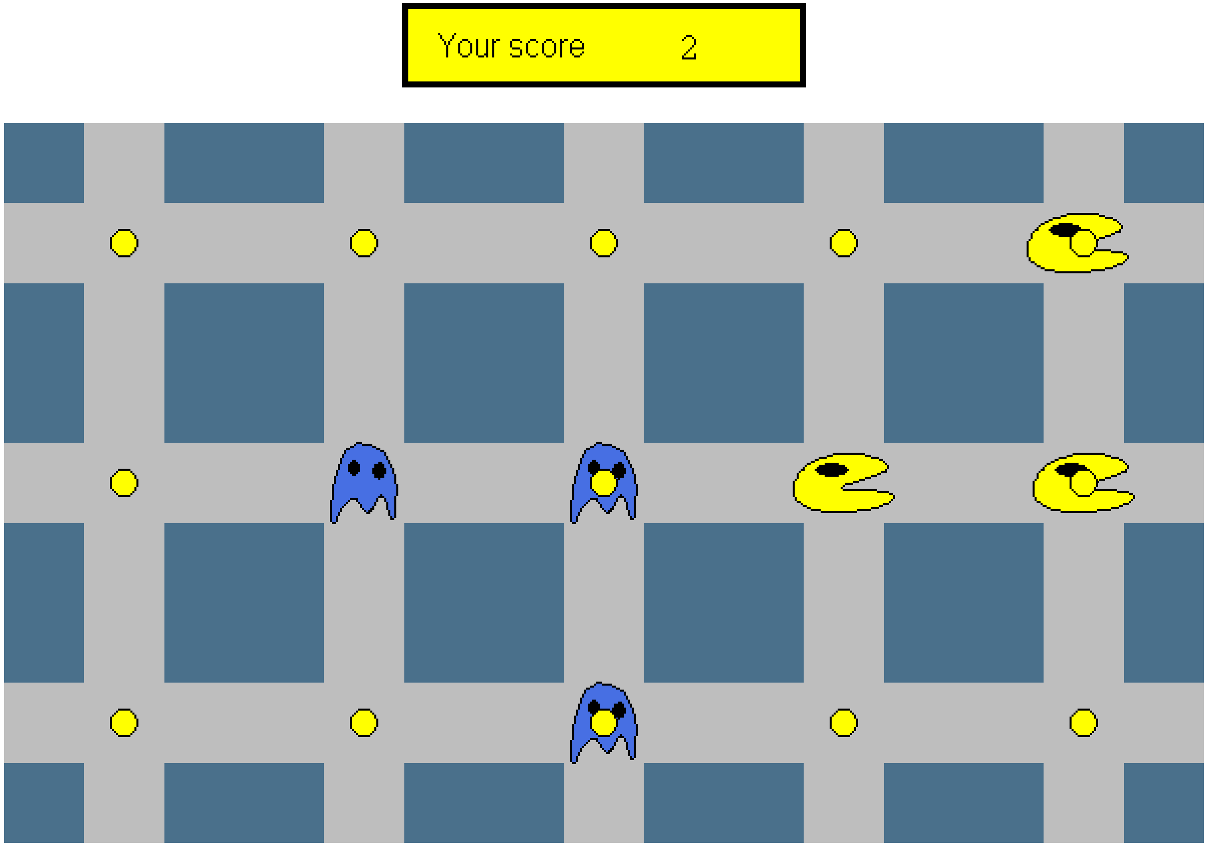
Terminate transformation in failure

Scheduling of the rules



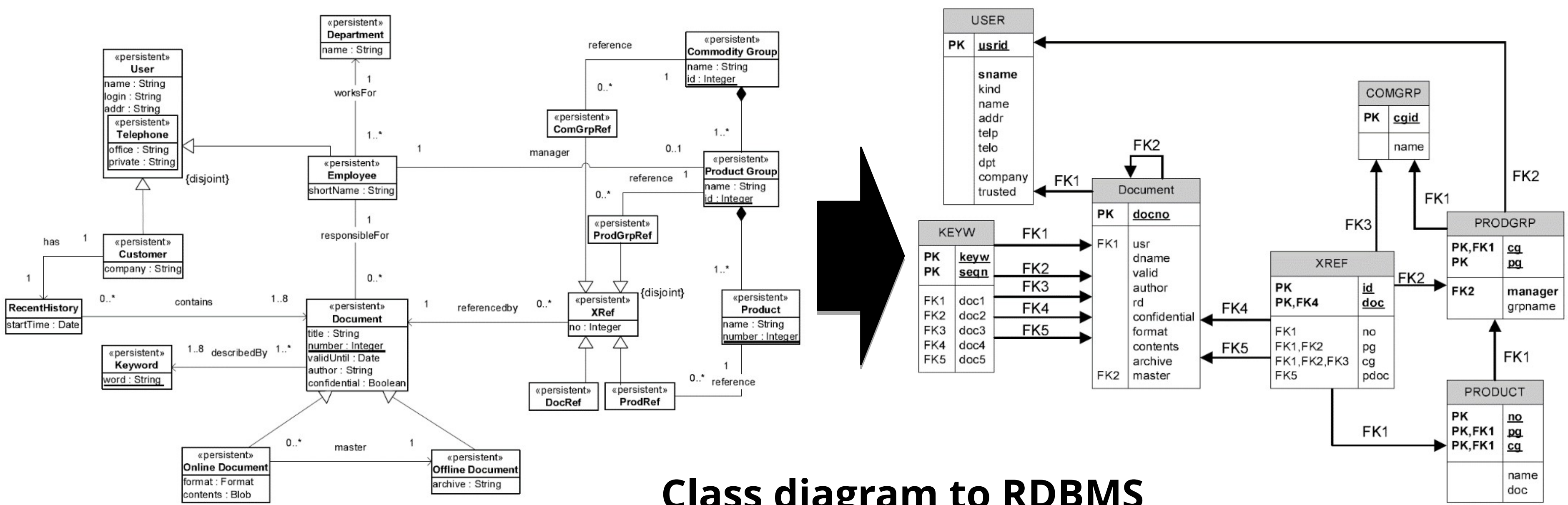
Simulation of a model

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



Translation

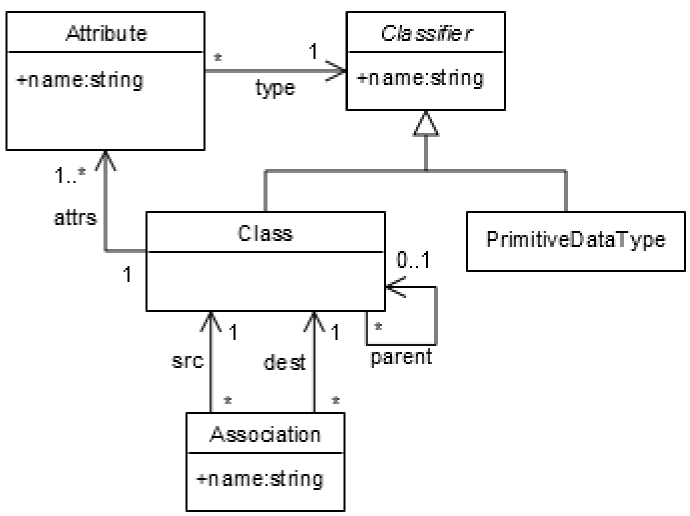
- **Maps** concepts of a model in a source language to concepts of another target language, while translating the **semantics** of the former in terms of the latter
- Similar to translational semantics, but the source language already has a semantics



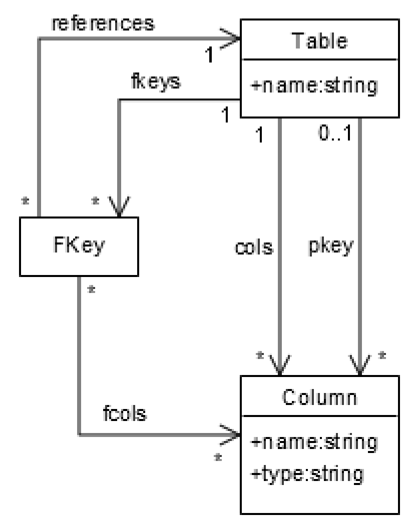
Class diagram to RDBMS

CD to RDBMS transformation

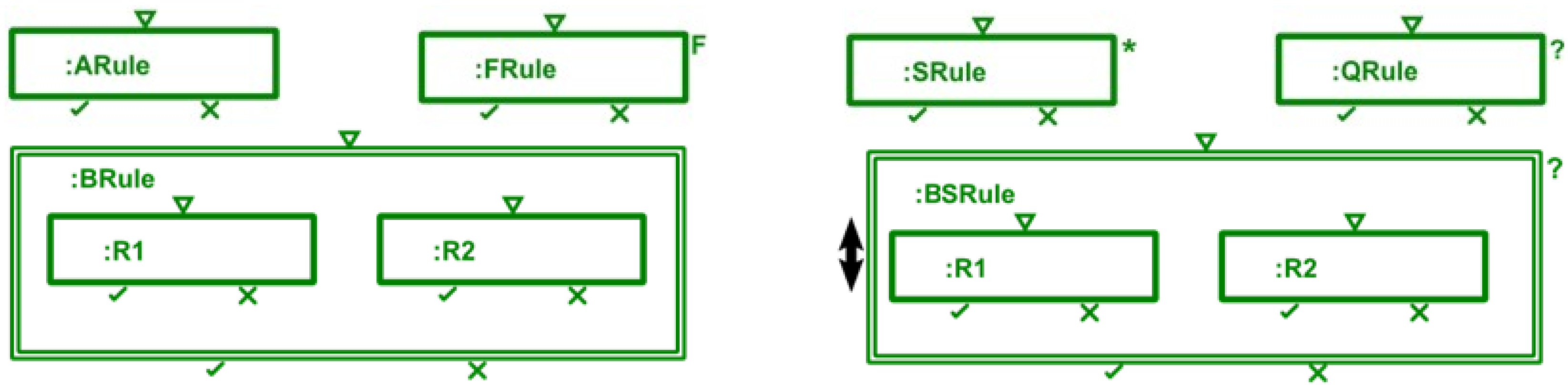
CD metamodel



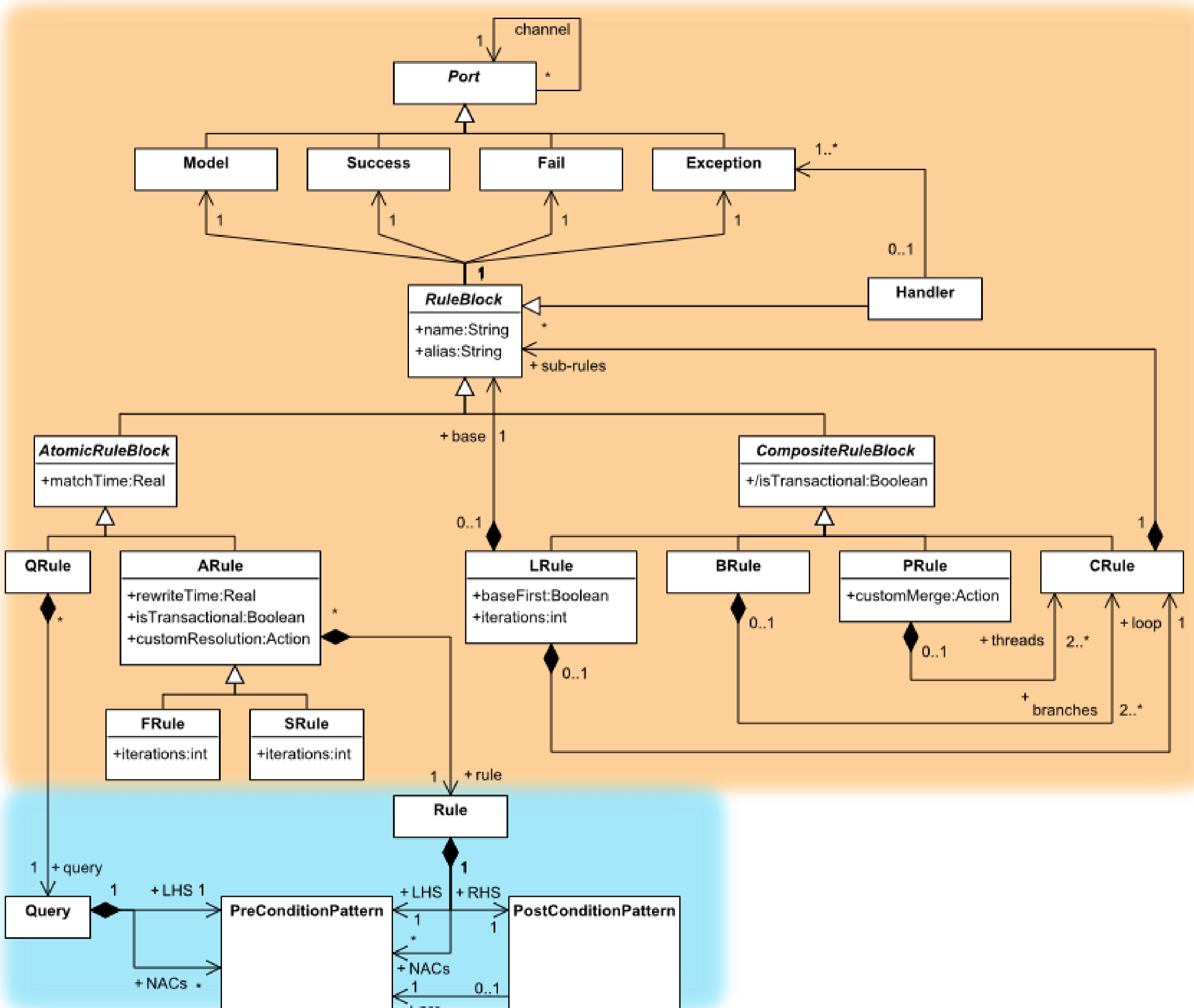
RDBMS metamodel



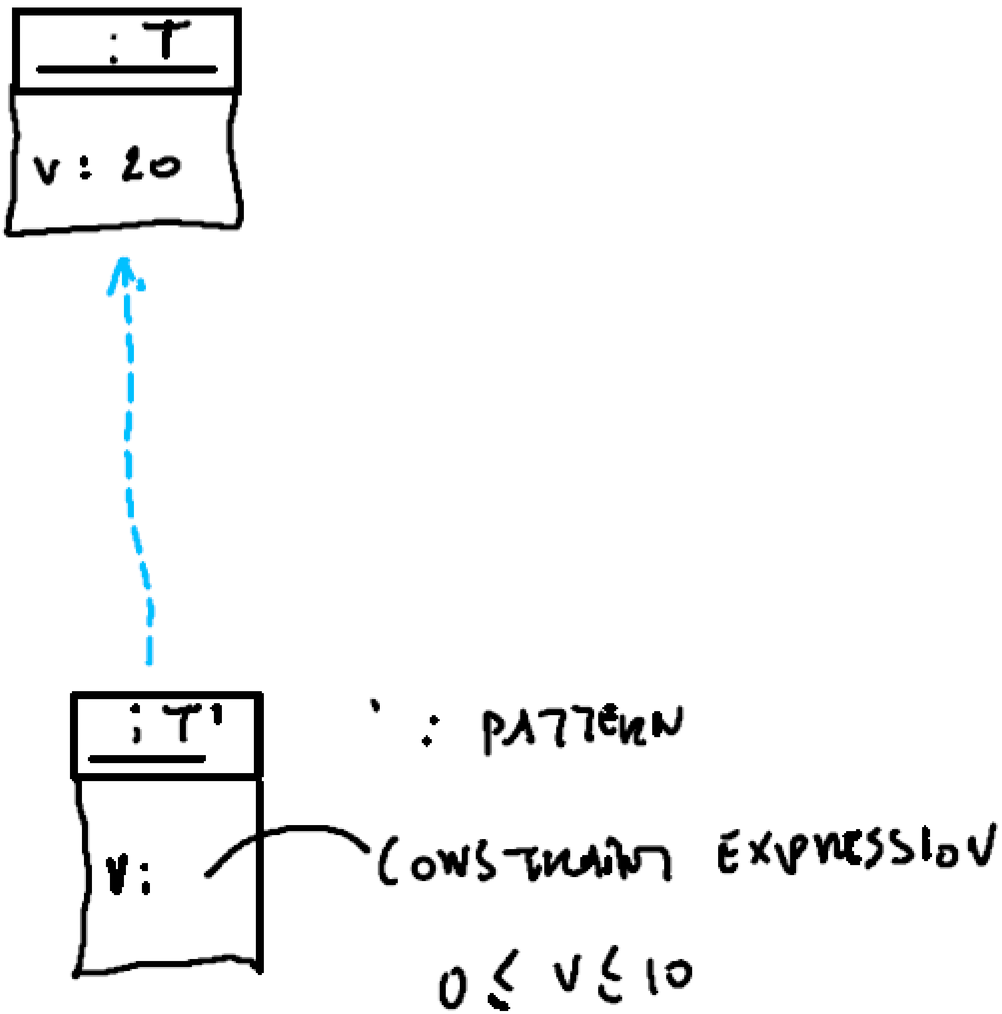
MoTif main rule types



- **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


channel
1
*
Port
Model
Success
Fail
Exception
1..*
1
1
1
1
0..1
1
RuleBlock
+name:String
+alias:String
Handler
*
+ sub-rules
+ base
1
AtomicRuleBlock
+matchTime:Real
CompositeRuleBlock
+/isTransactional:Boolean
0..1
1
QRule
ARule
+rewriteTime:Real
+isTransactional:Boolean
+customResolution:Action
LRule
+baseFirst:Boolean
+iterations:int
BRule
PRule
+customMerge:Action
CRule
*
*
0..1
0..1
+ threads
2..*
+ loop
1
0..1
+ branches
2..*
FRule
+iterations:int
SRule
+iterations:int
1
+ rule
Rule
1
+ query
1
Query
1
+ LHS 1
PreConditionPattern
+ LHS
+ RHS
PostConditionPattern
1
1
*
+ NACs
1
0..1
+ pre
+ NACs
*

Pattern model <> Instance model

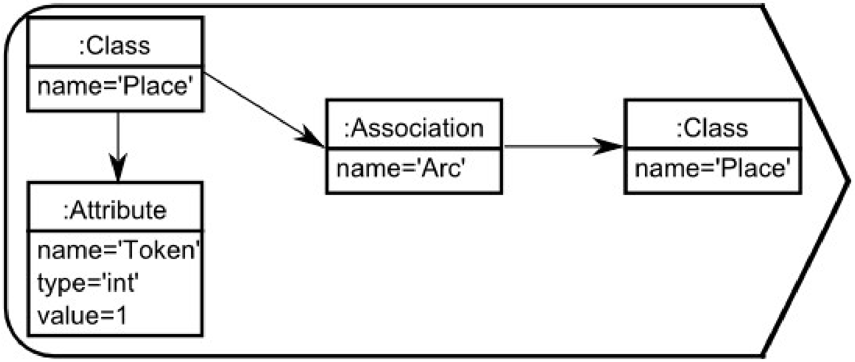


Instance model

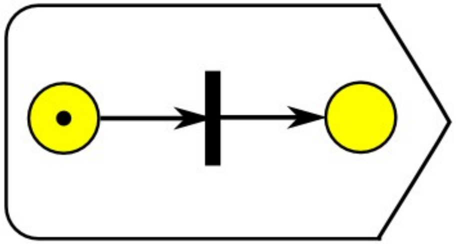
Pattern model

Pattern language

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

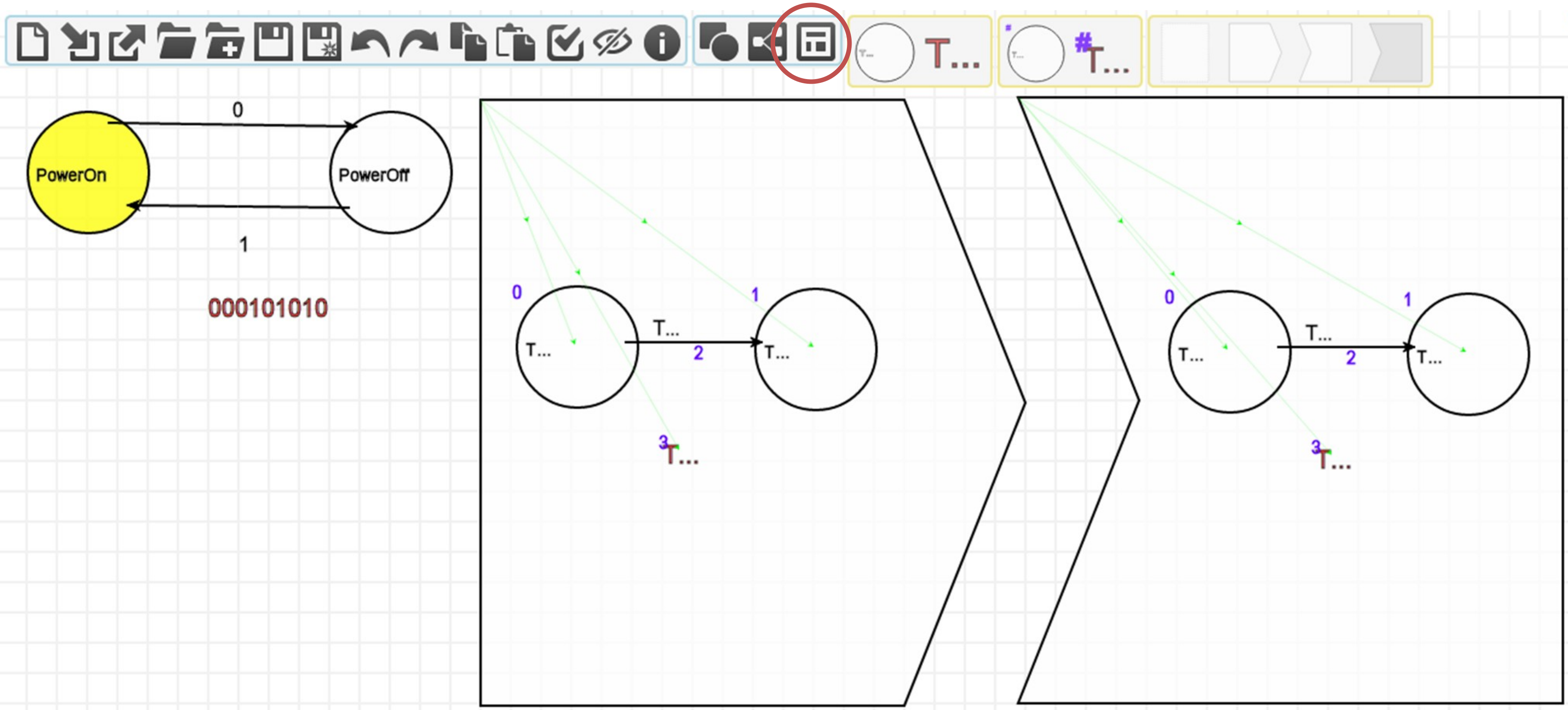


2. Customized pattern language
   + Concrete syntax adapted to the source/target languages (DSL)
   + Exclude patterns that do not have a chance to match
   - More work for the tool builder



[E.b1]

RAMification process



Domain-specific pattern languages

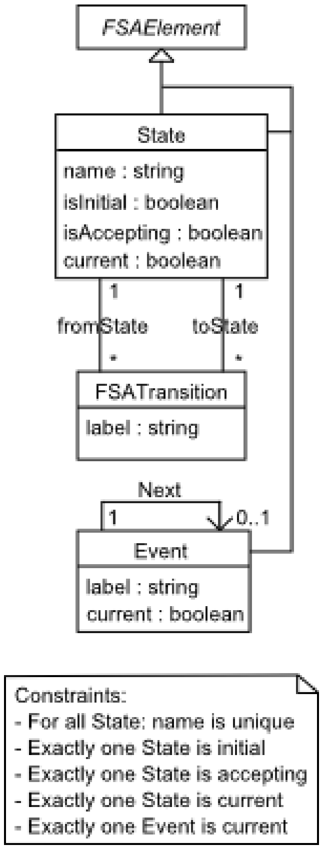
Ramification Process: automatically generated environment for pattern language

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

Relax Augment Modify

Customized Pattern Meta-Model

RAMification process: starting point meta-model(s)

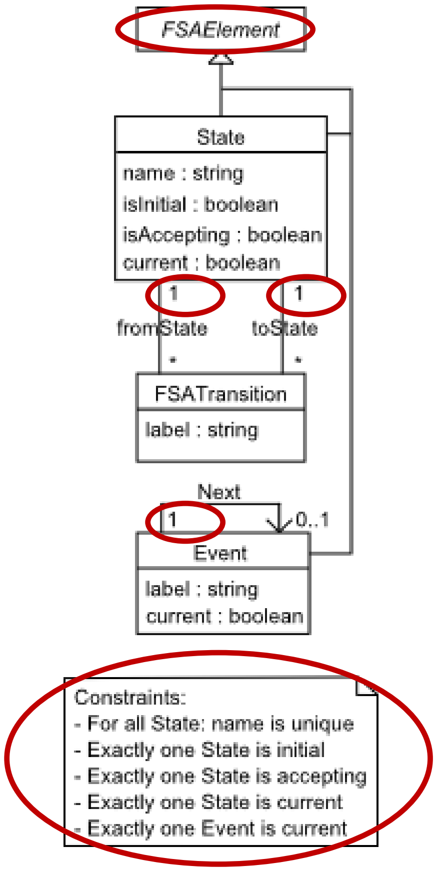


RAMification process

Relaxation

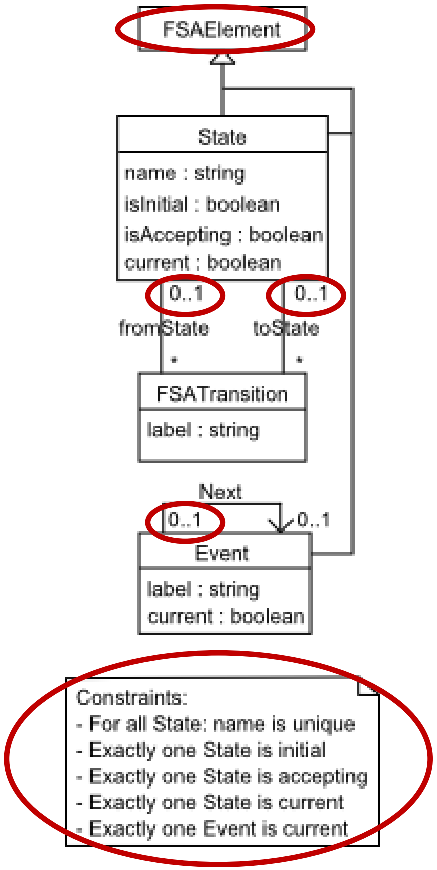
- Relaxes the constraints imposed by the meta-model(s) of the domain (source and target when exogenous)
- Make abstract classes concrete (to allow instantiation)
- Reduction of minimal multiplicity of every association end
- Constraints relaxation (manual)
  - Removed
  - Preserved
  - Depends on static semantics of the language(s)

RAMification process



Relax

RAMification process

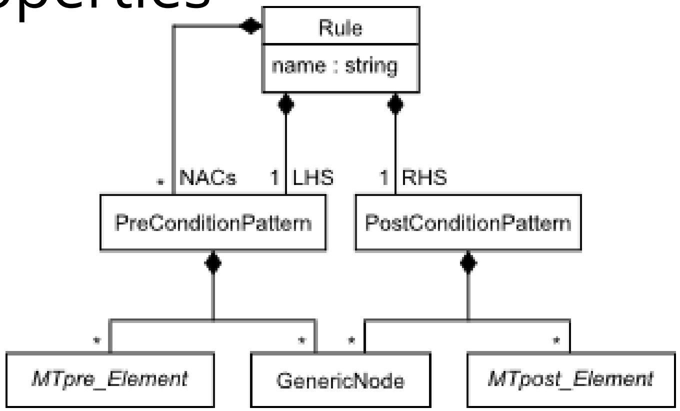


Relax

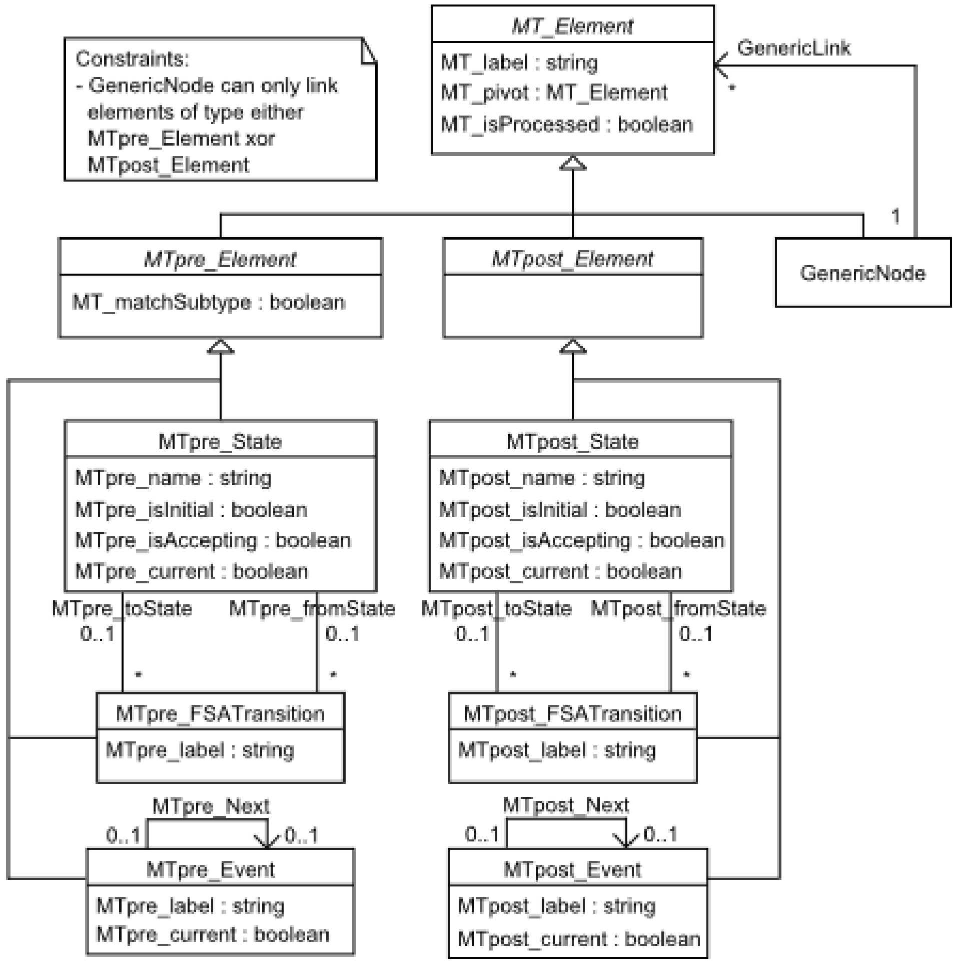
RAMification process

Augmentation

- Augments the resulting meta-model with additional information
- Classes and associations integrated in a 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



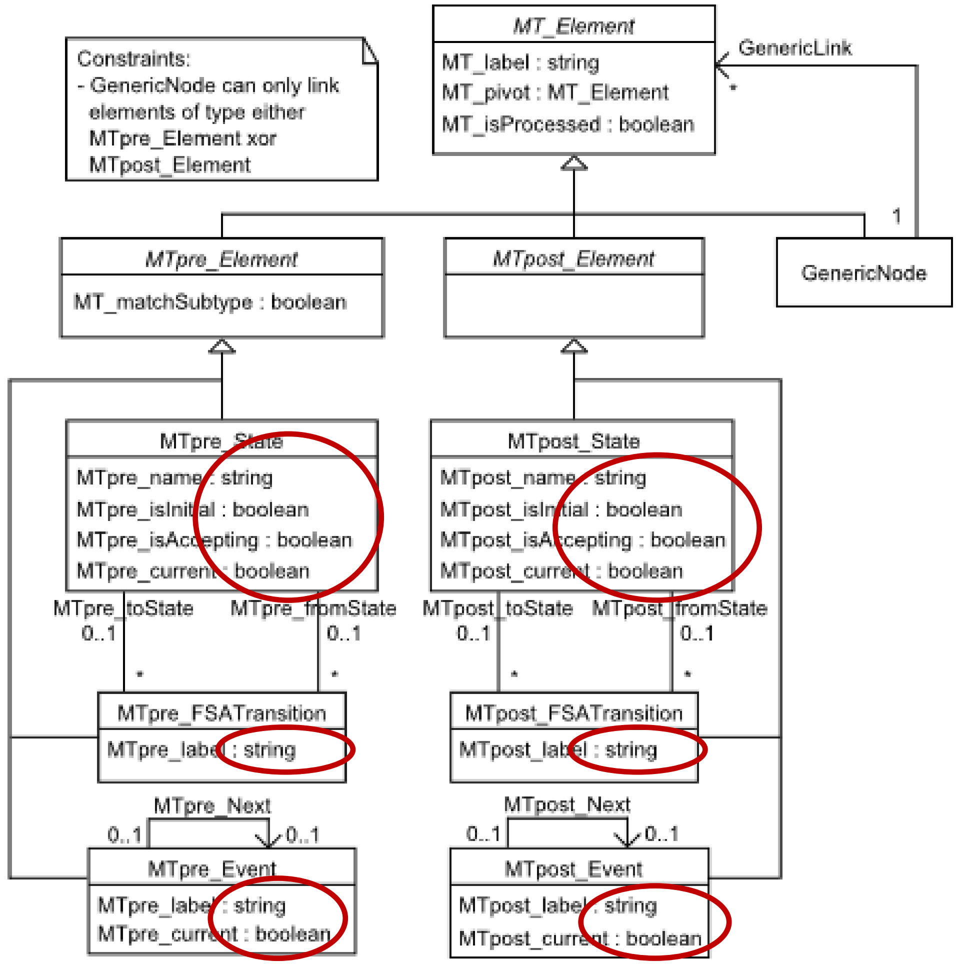
Augment

RAMification process

Modification

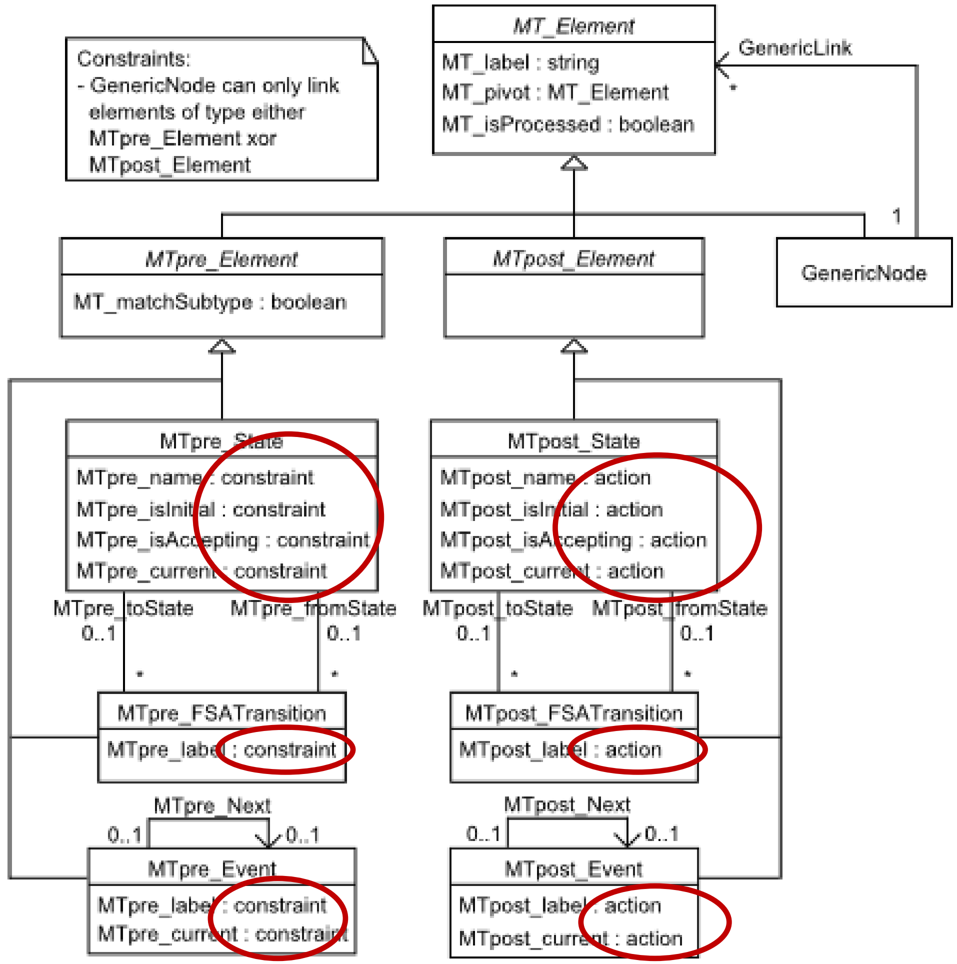
- Performs further modifications on the resulting meta-model
- Update namespaces
- Change type of attributes
  - Pre-condition classes: constraint type
  - Post-condition classes: action type
  - But preserve knowledge of original type for well-formedness
- Adaptation of concrete syntax (semi-automatic)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

RAMification process



Modify

RAMification process



Modify

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

GRAMMAR

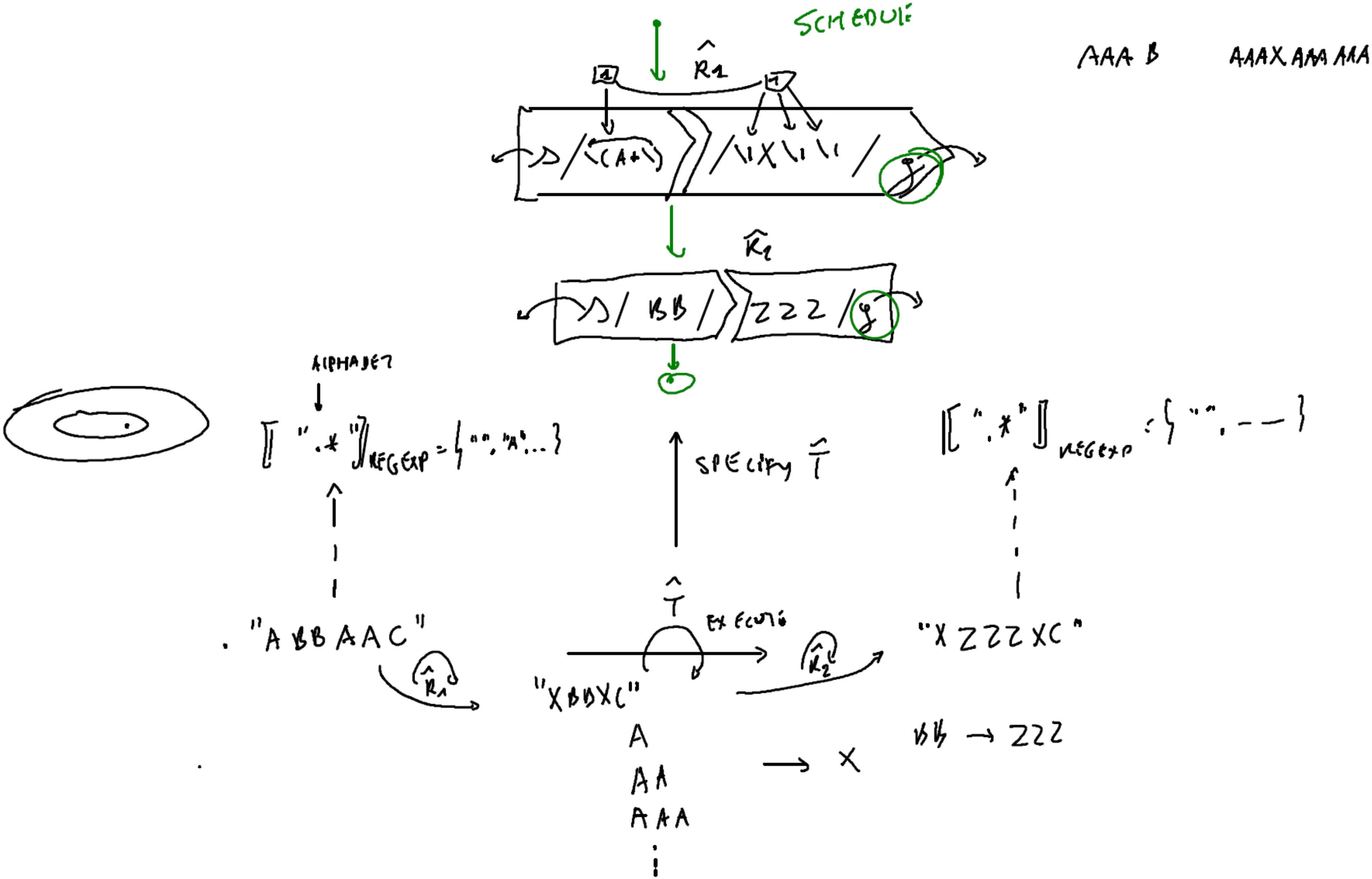
GME ∈ ∈

MM_{CD}

GRAPH GRAMMAR

" TRANSFORMATION

Model Transformation



Transforming Strings, Trees, or Graphs?

STRINGS / SEQUENCE ⊆ TREES ⊆ GRAPHS

JAVA/Python ... TRSF ⊆ TRSF ⊆ TRSF

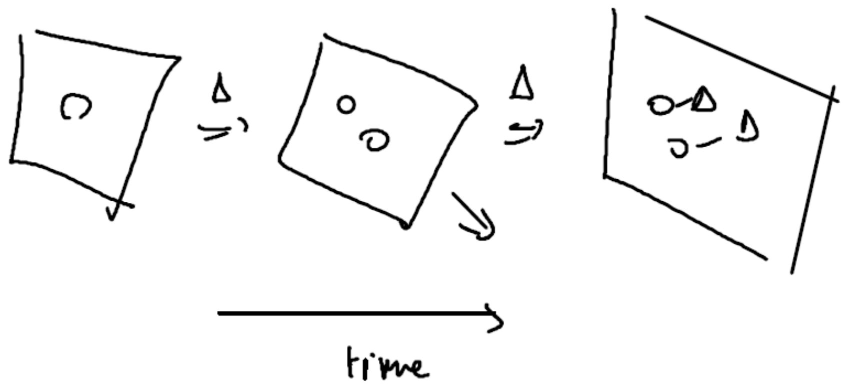
COGNITIVE

HOW

WHAT?

DECLARATIVE

DYNAMIC STRUCTURE



WHY NOT ALWAYS USE GRAPHS / GRAPH TRSF. ?

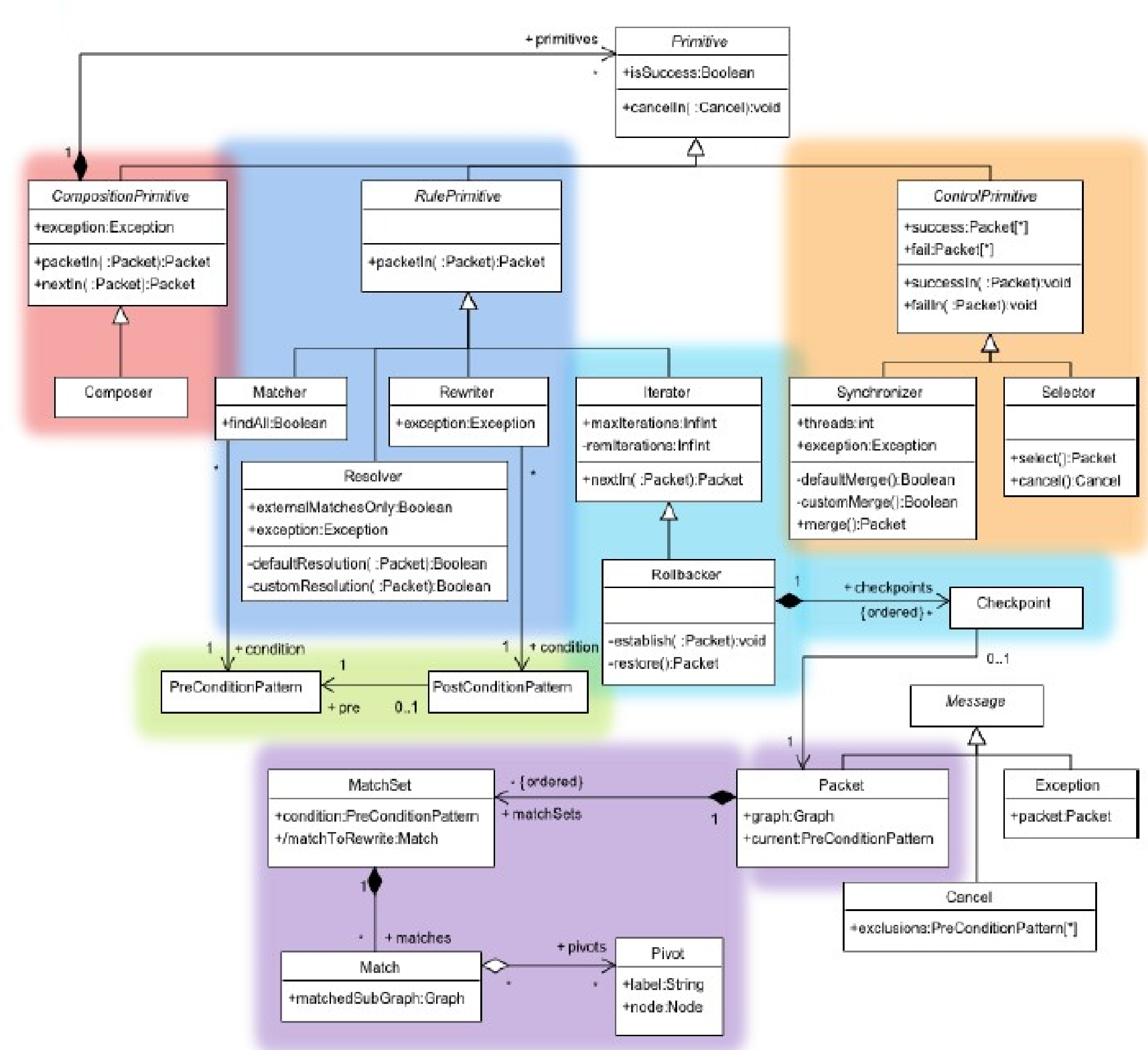
① COMPLEXITY → PERFORMANCE / EFFICIENCY

② COGNITIVE / UNDERSTANDING

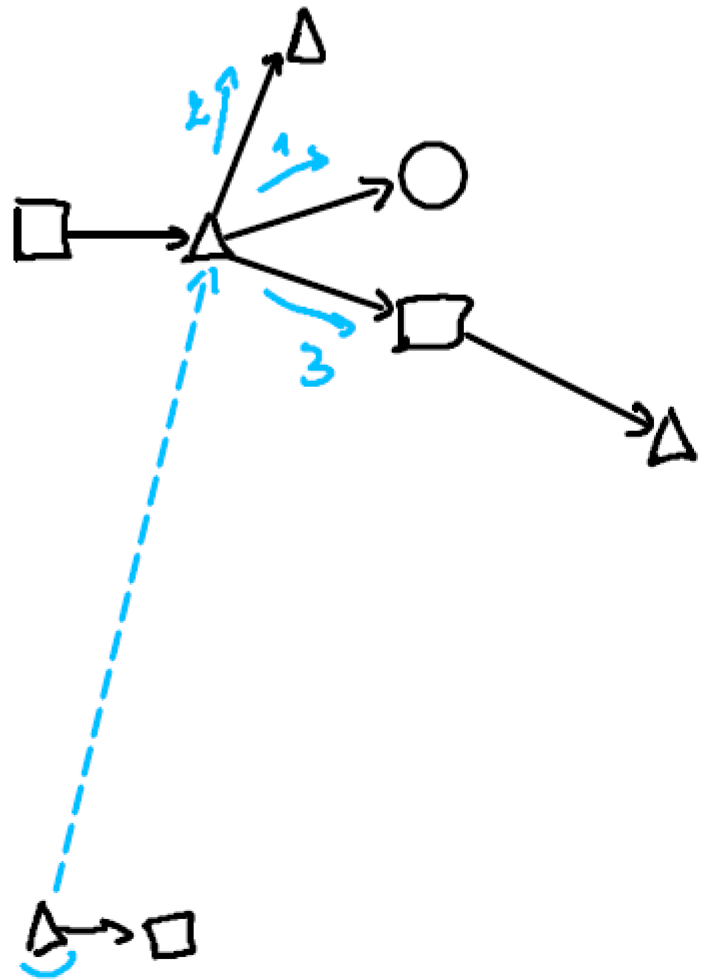
MOST APPROPRIATE
- ABSTRACTION
- FORMALISM

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

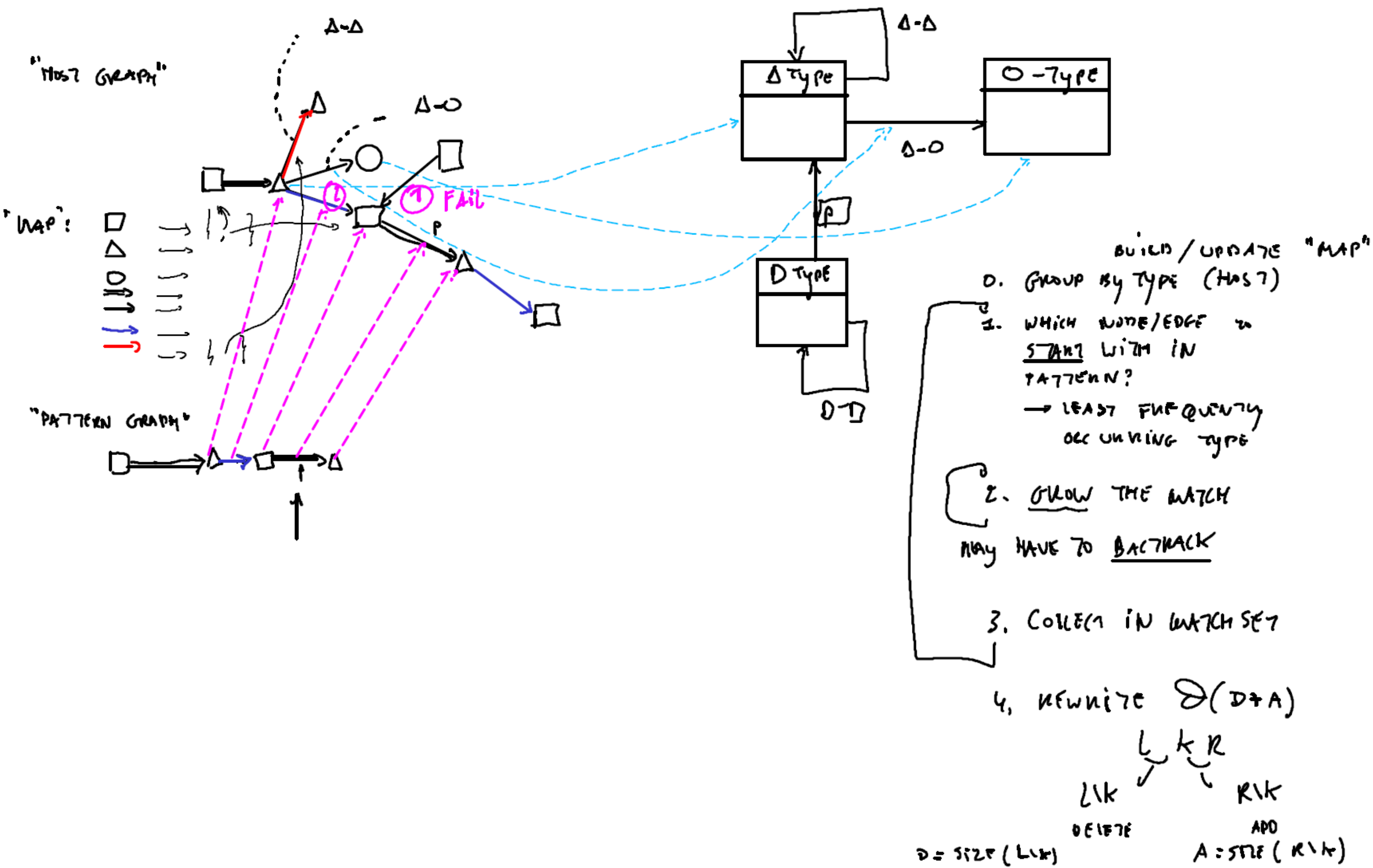
matching, pivot, scope



Matching Algorithms (1): Search Plans



Matching Algorithms (2): Constraint Satisfaction



Matching Algorithms (2): improving performance through (user) "hints"

PIVOT PARTIAL MATCH

SUB-GRAPH OF HOST GRAPH

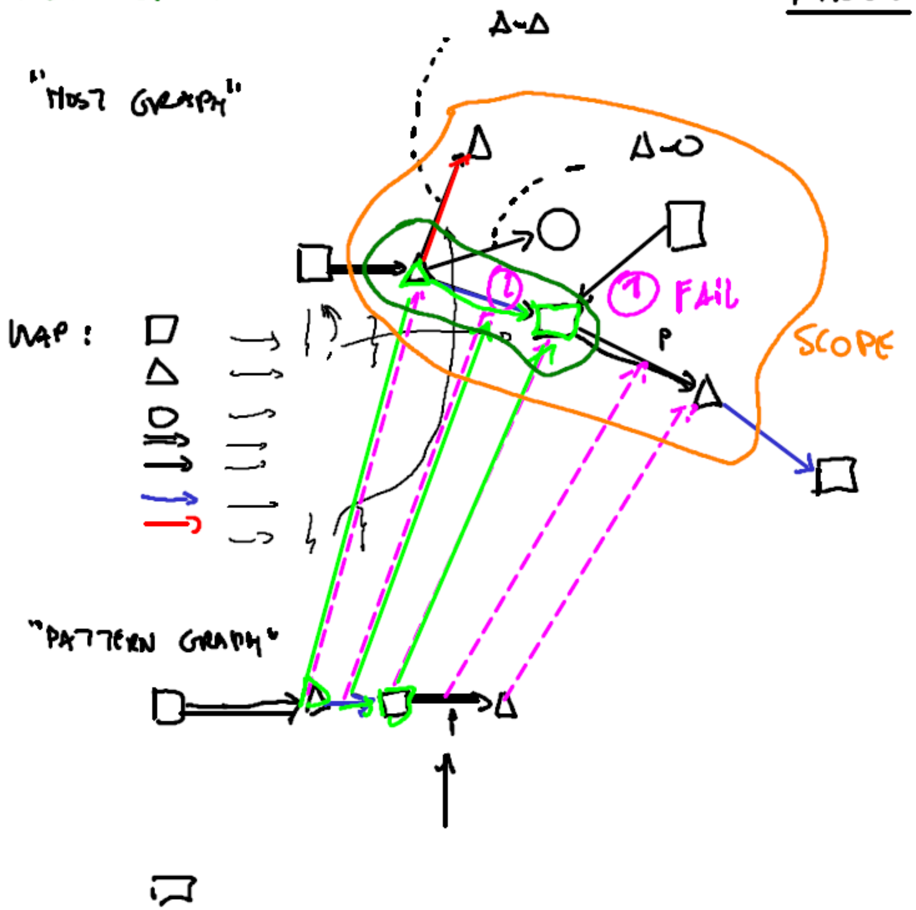
WHERE TO START

FIRST MATCH

Θ (SIZE (PATTERN GRAPH))

LINEAR!

SIZE (HOST GRAPH) $> 10^6$



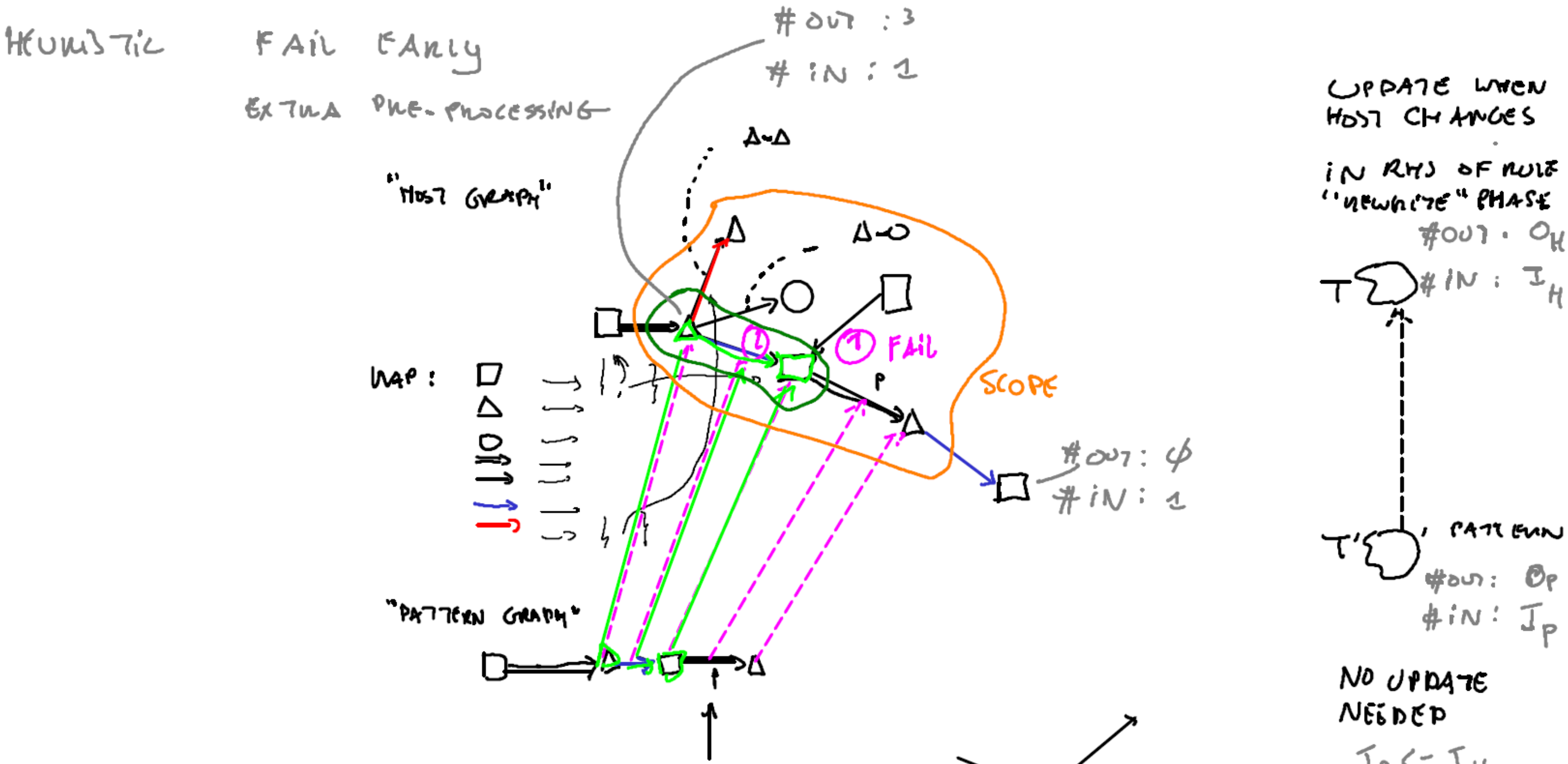
"COMPLEX PATTERN"

→ FAIL EARLIER

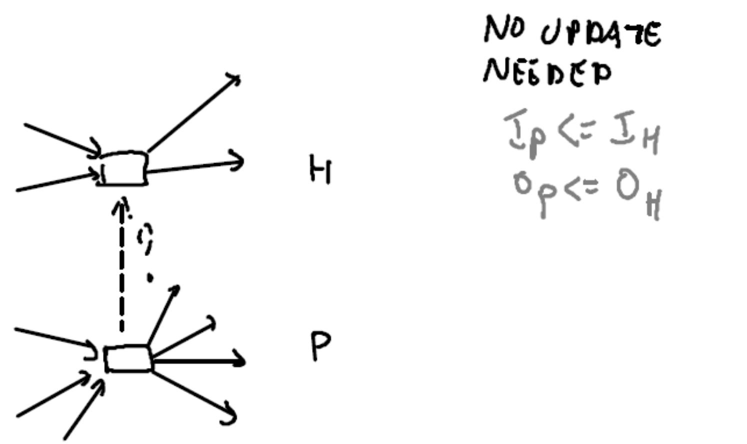
(FEWER MATCHES)

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

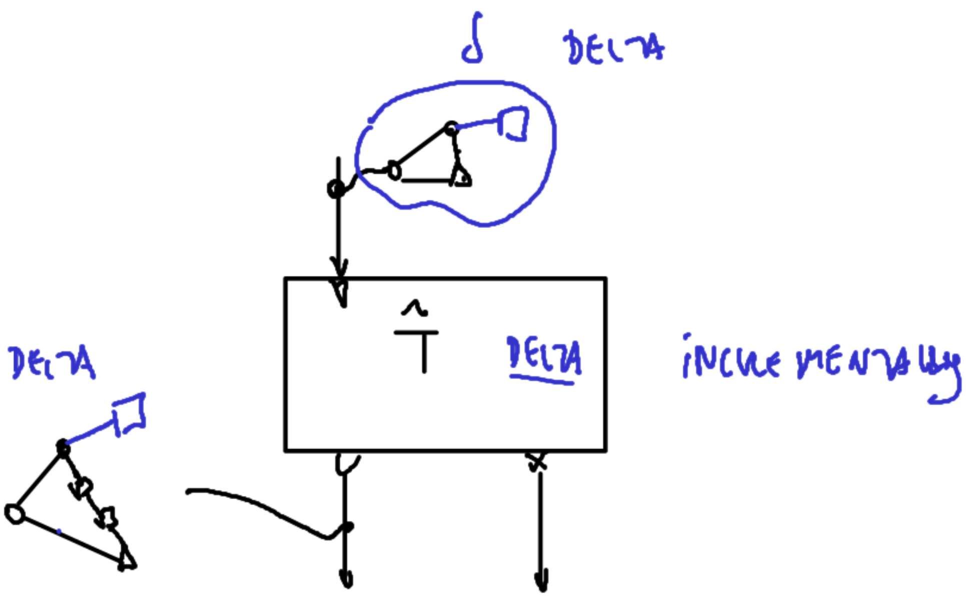
Matching Algorithms (2): improving performance through heuristics

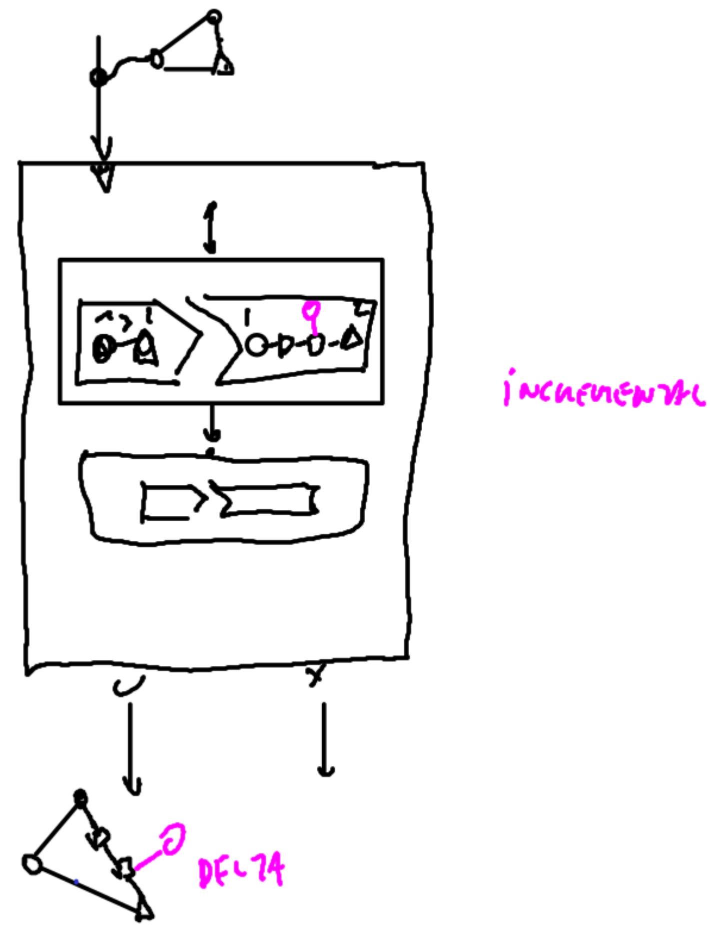


VF2 "Very Fast 2"



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

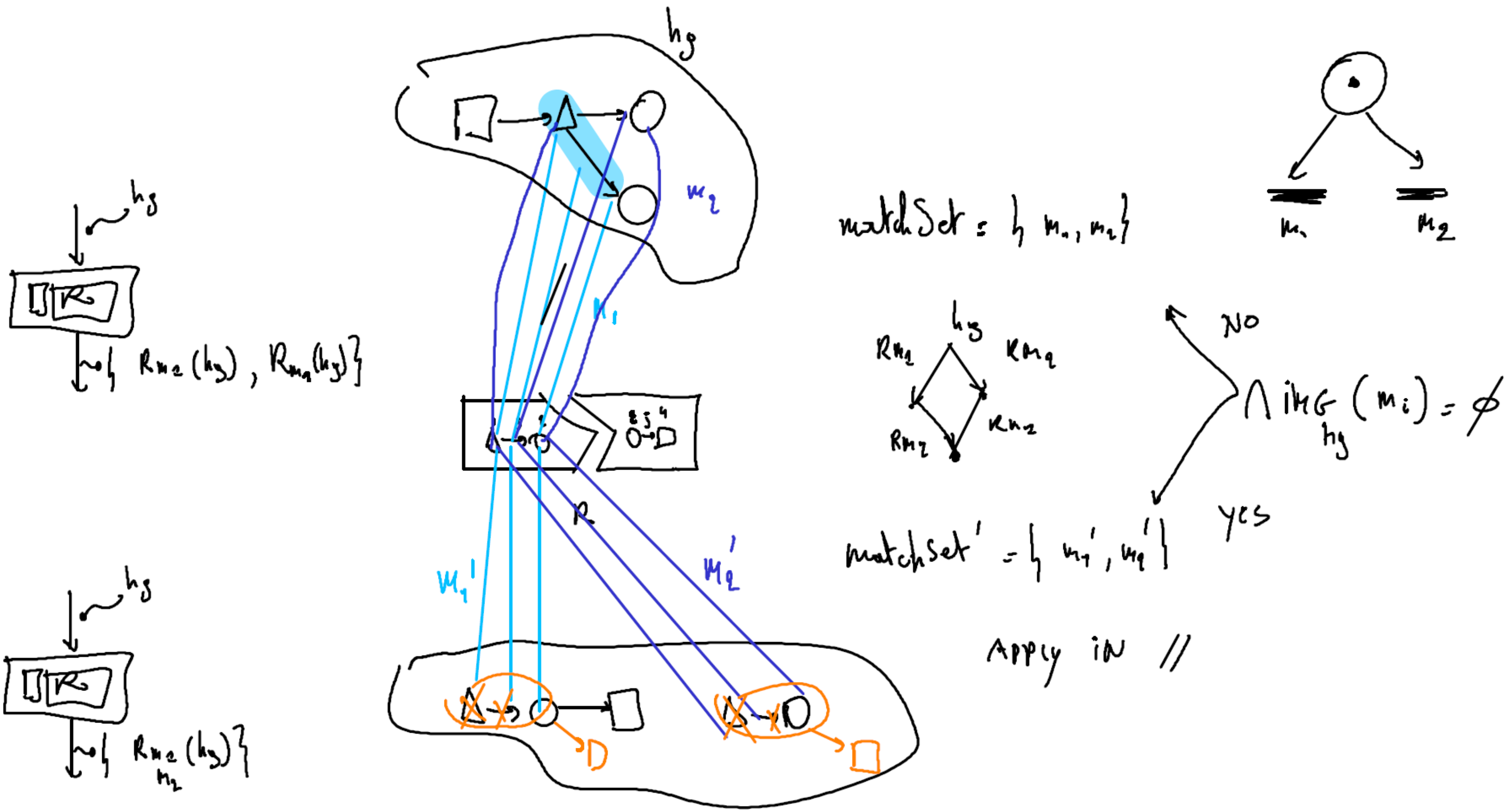




RETE

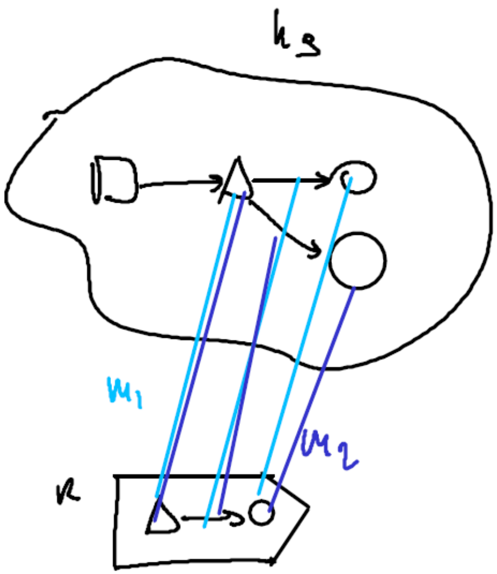
Prolog

Choice → parallel independence, critical pairs

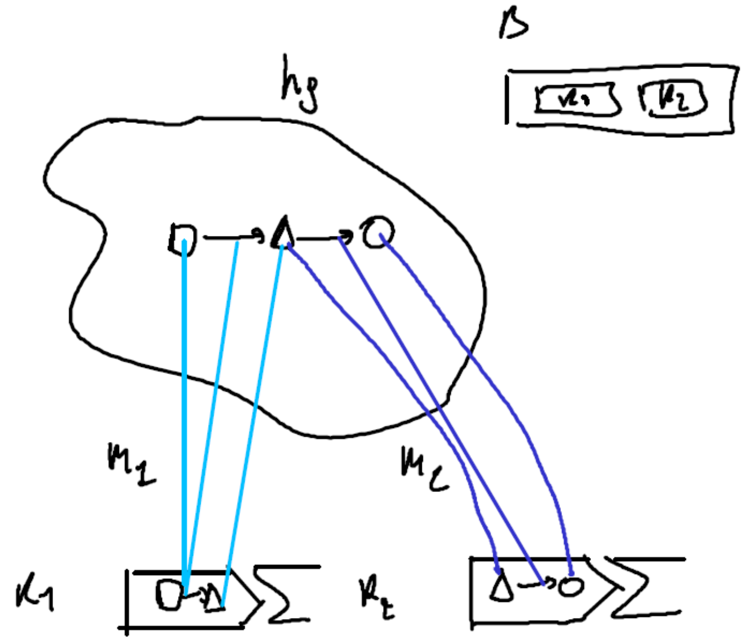


Choice (vs. enumerate all, see Petri Nets reachability graph)

single rule, multiple matches



multiple rules, multiple matches



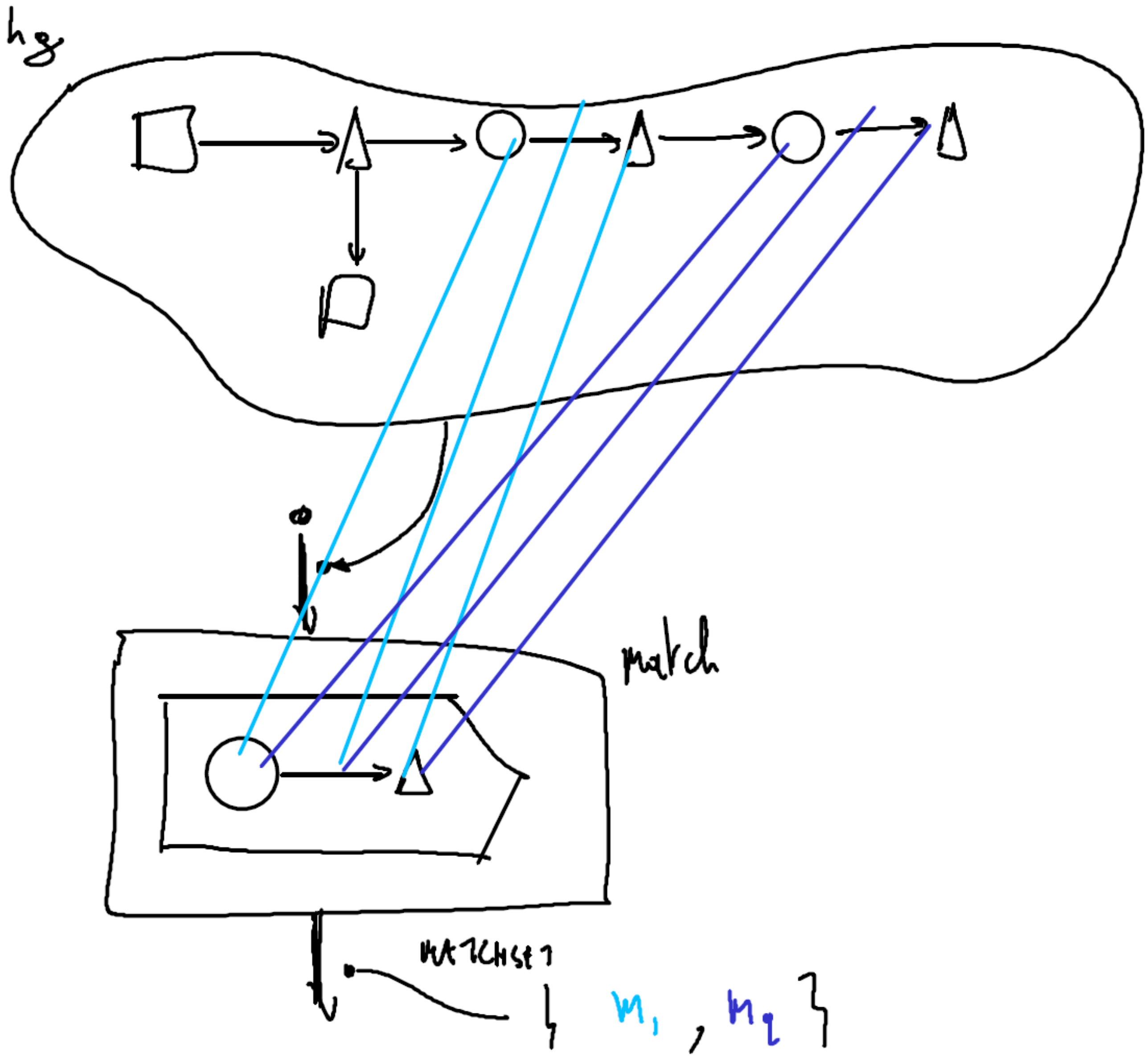
RANDOM CHOICE (REPEATABLE)

FROM ALL MATCHES

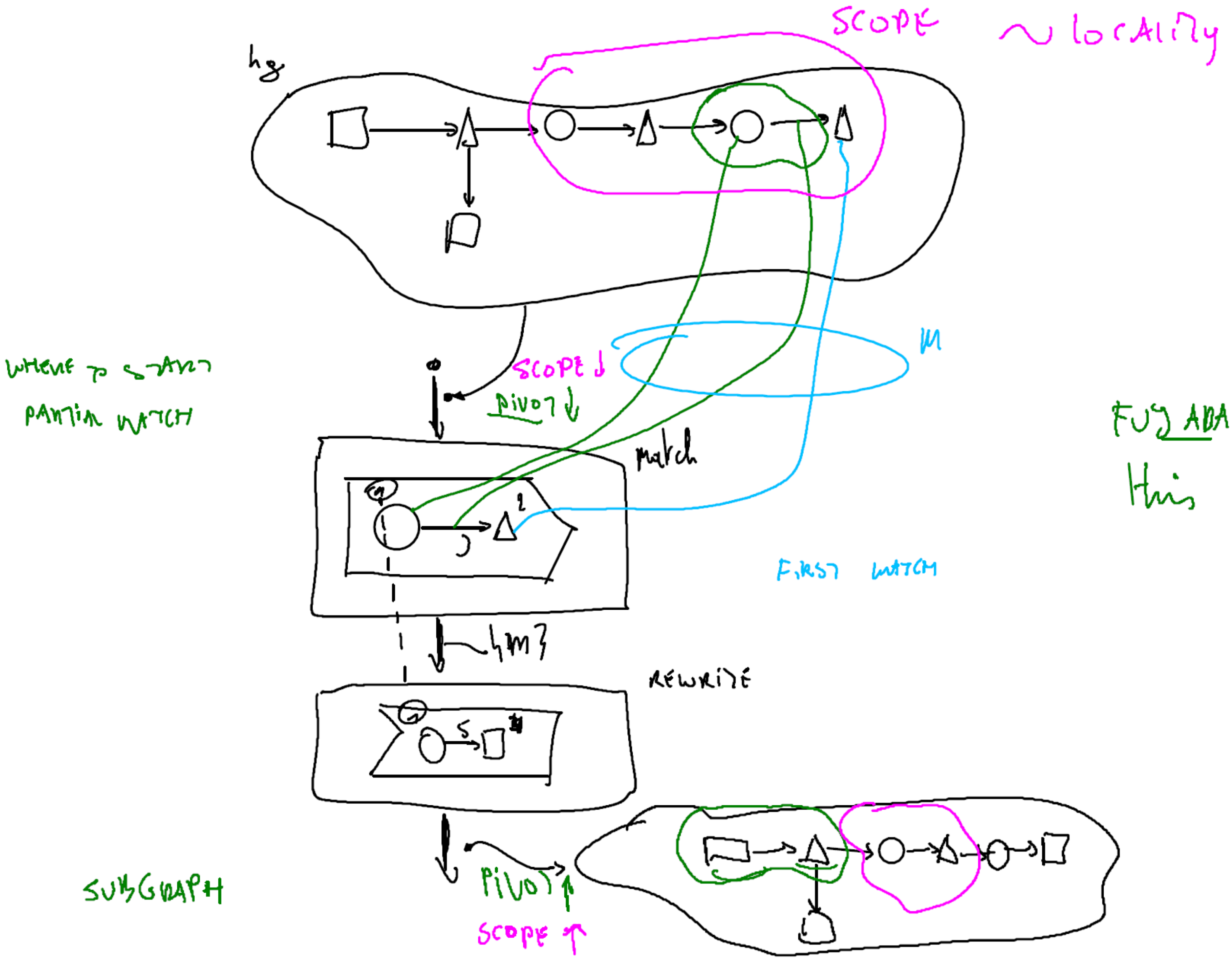
SELECT({m1, m2})

$U_I([0,1])$ PSEUDO-RANDOM NR. GENERATOR (SEED)

MatchSet = the set of all the matches (morphisms)



De-constructing a rule into 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

Helper in OCL

```
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

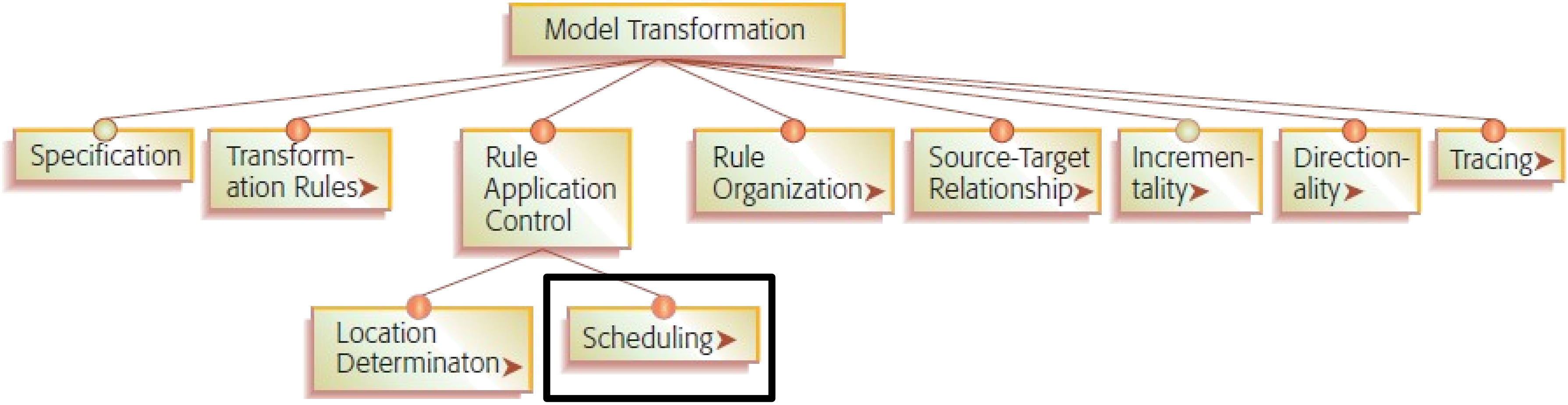
Call temporary queries

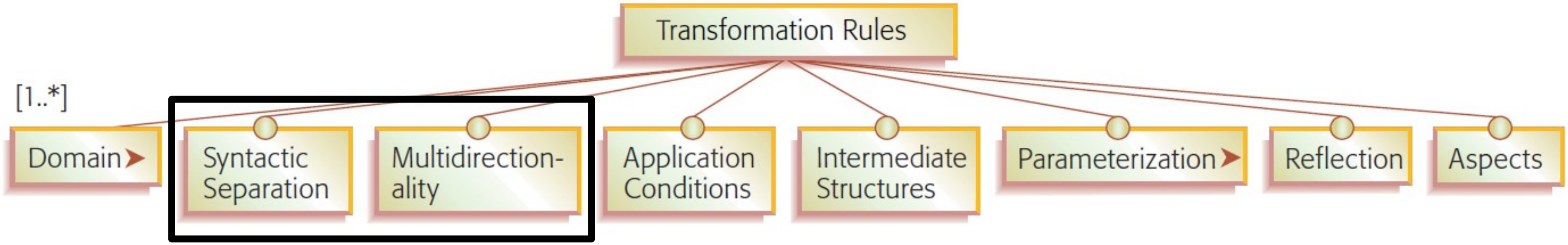
Execution of a declarative rule in ATL

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



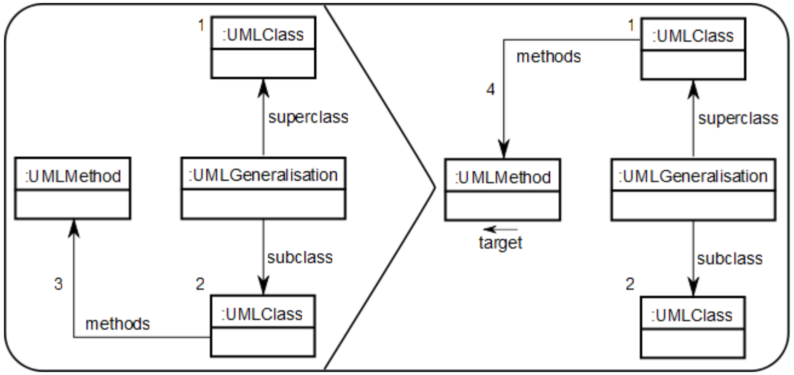


K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal:* 45(3), 621-645 (2006).

Rule patterns

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

MoTif rule

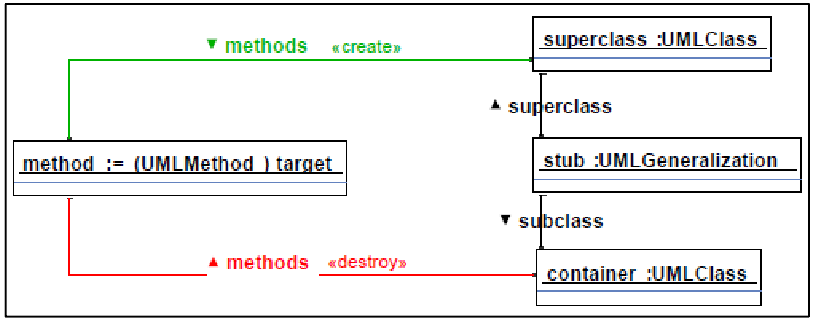


ATL rule

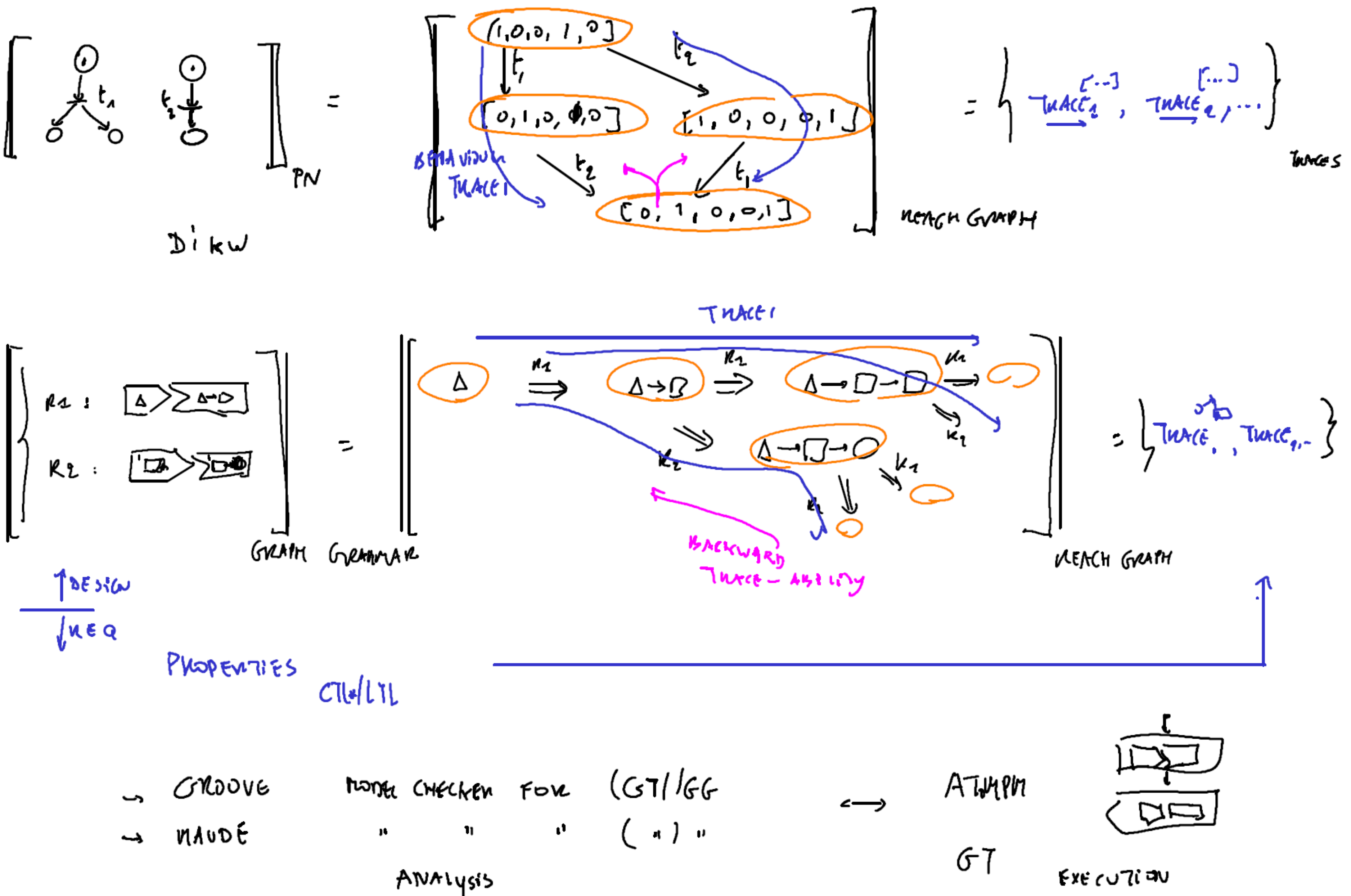
```
module Person2Contact;
create OUT: MMb from IN: MMa {

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

FUJABA/Henshin compact notation

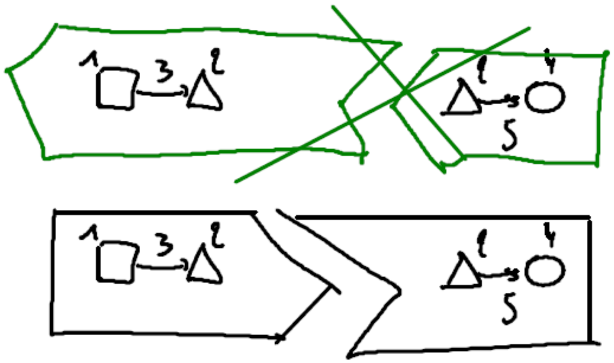


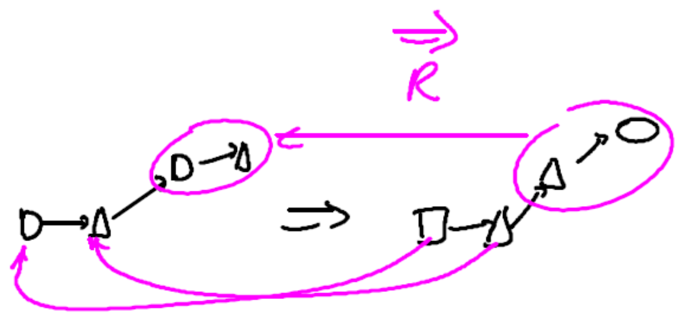
Choice → explore all possibilities → analysis over all traces



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

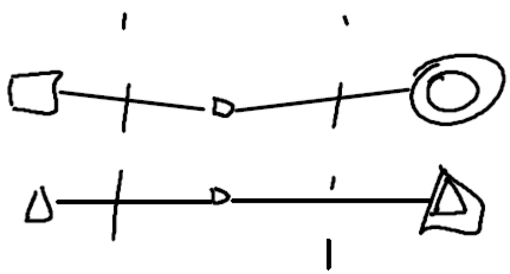
RULE R

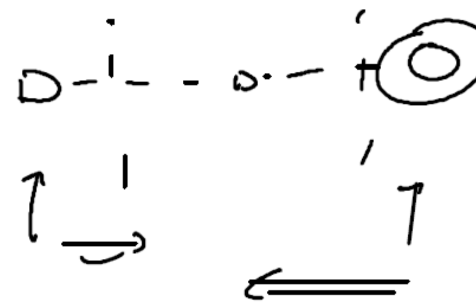




TRACE

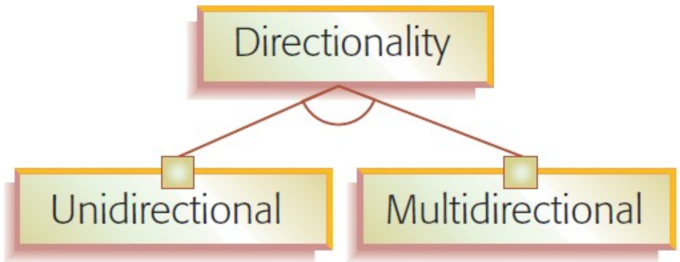
RELATION



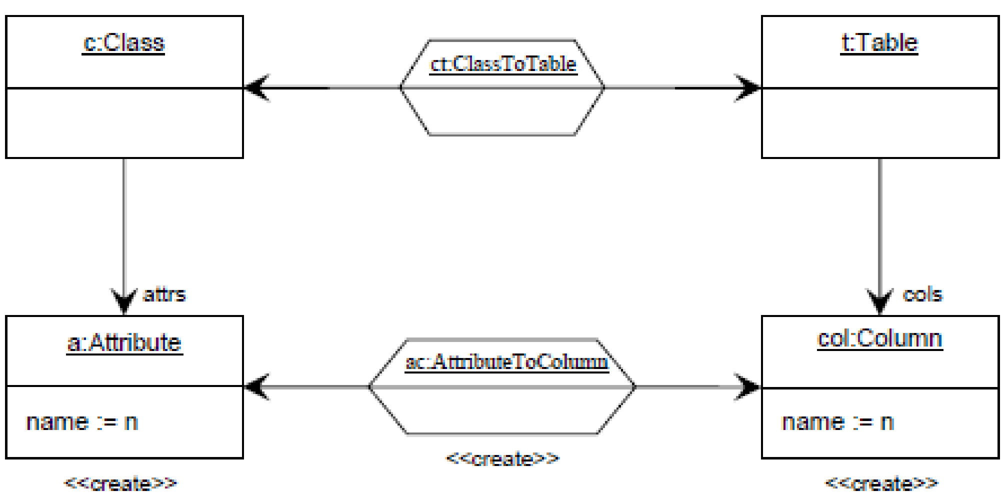


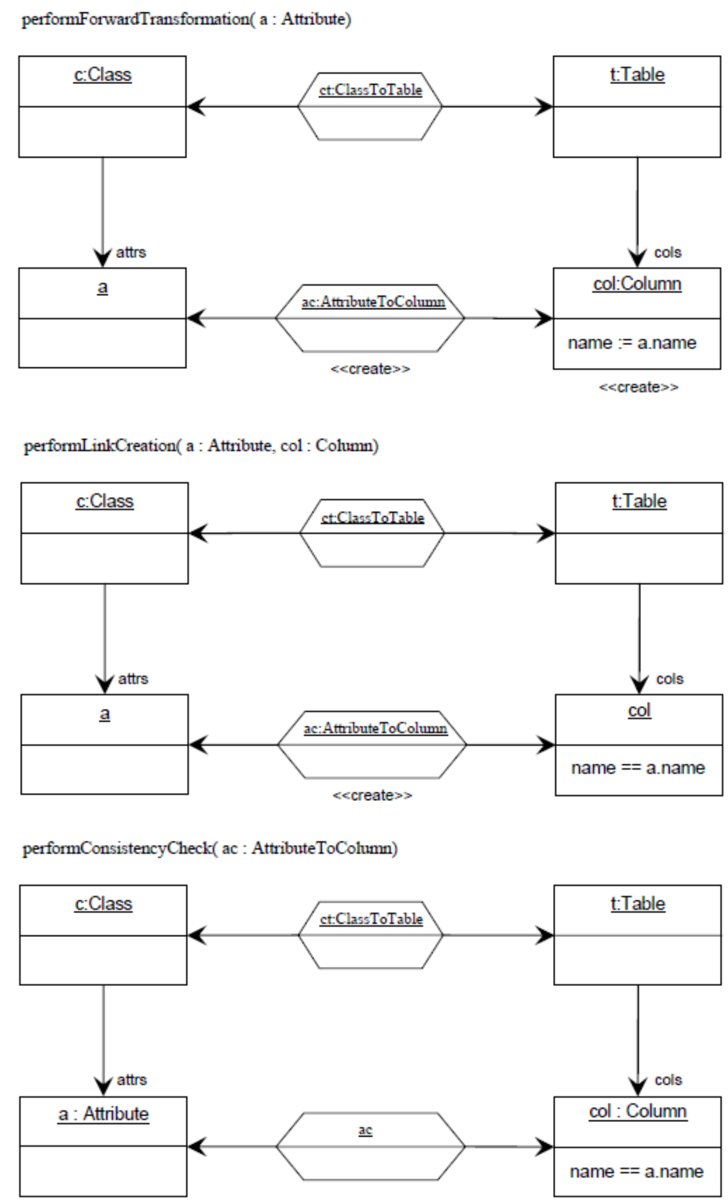
BI-DIRECTIONAL
TRSF.

Multi-directional rules



TGG rule

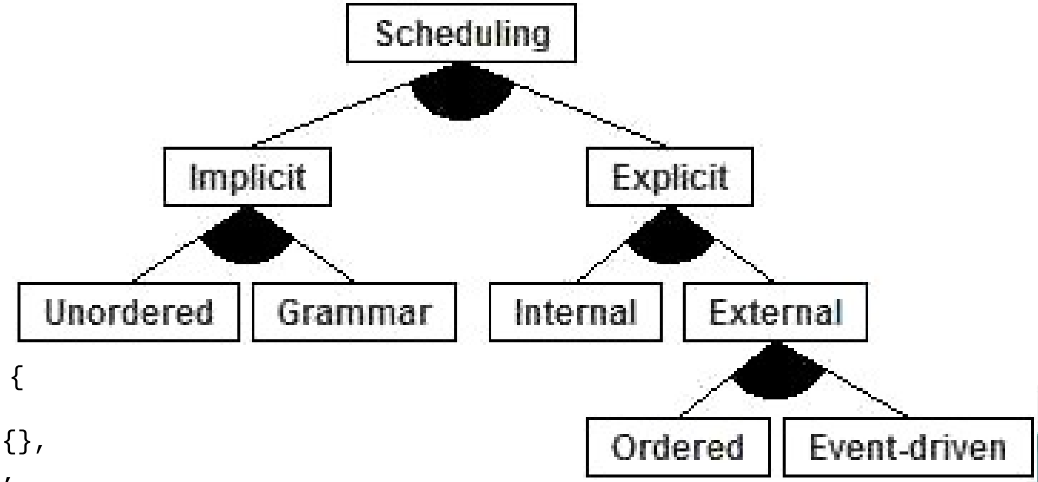




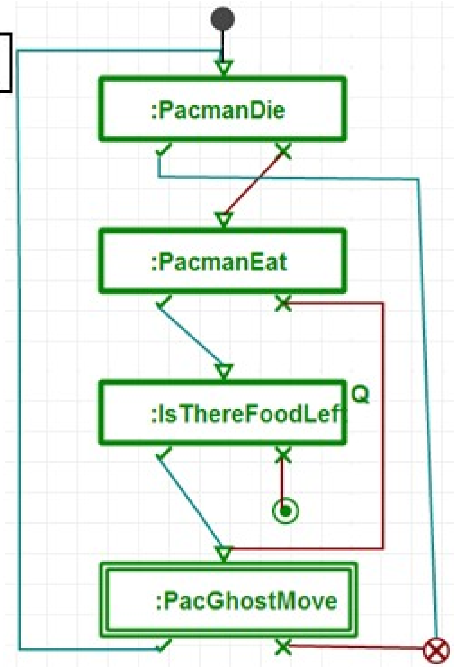
TGG operational rules

Rule scheduling strategies

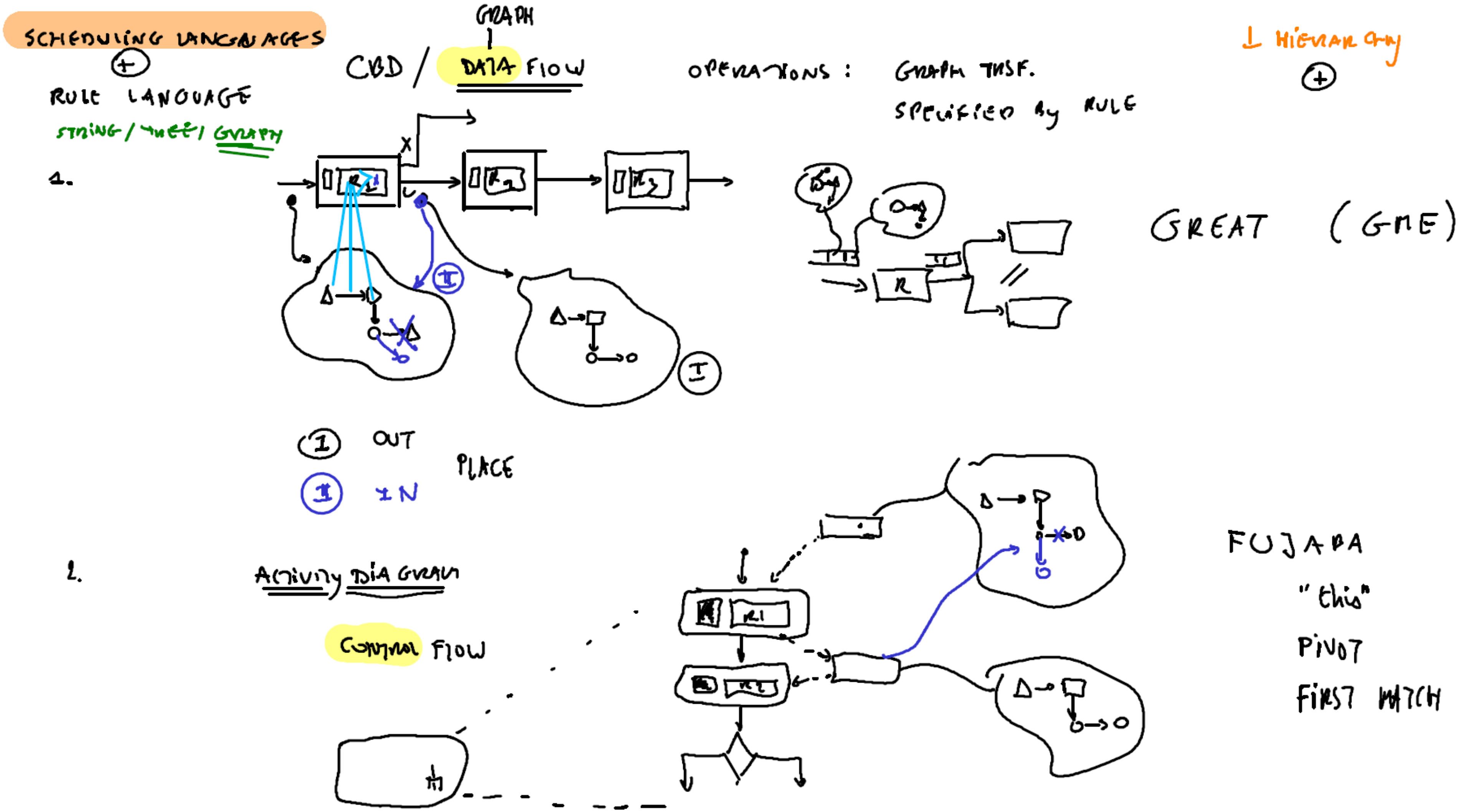
Explicit




```
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);
   }
}
```



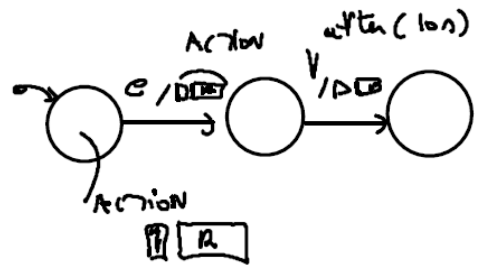
Rule Scheduling (aka Control)



Rule Scheduling (aka Control)

3. TIMED STATE AUTOMATA

STATECHARTS

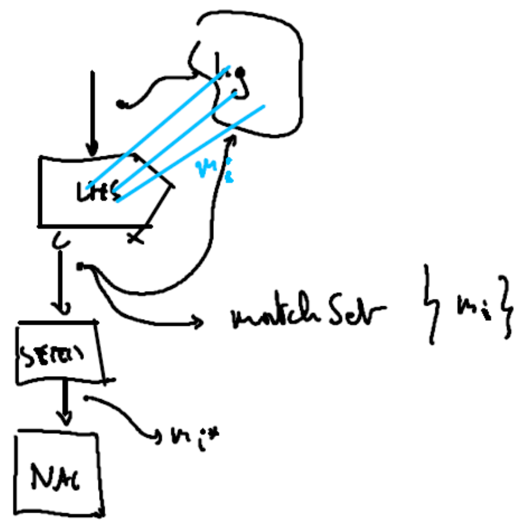


TIMED MT

CONCURRENCE

4. DEVS

- TIME
- HIERARCHY
- DEV
- CONCURRENCY



Motif

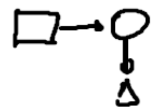
A RULE

Rule Scheduling (aka Control)

GREAT FUJABA MOTIF

COMPILER

3. PROGRAMMING LANGUAGE



NAC LHS RHS

hg = GRAPH()

LHSrule = LHSRULE()

matcher = MATCHER()

matchSet = matcher.match(LHSrule, hg)

match = select(matchSet)

...

REWRITER()

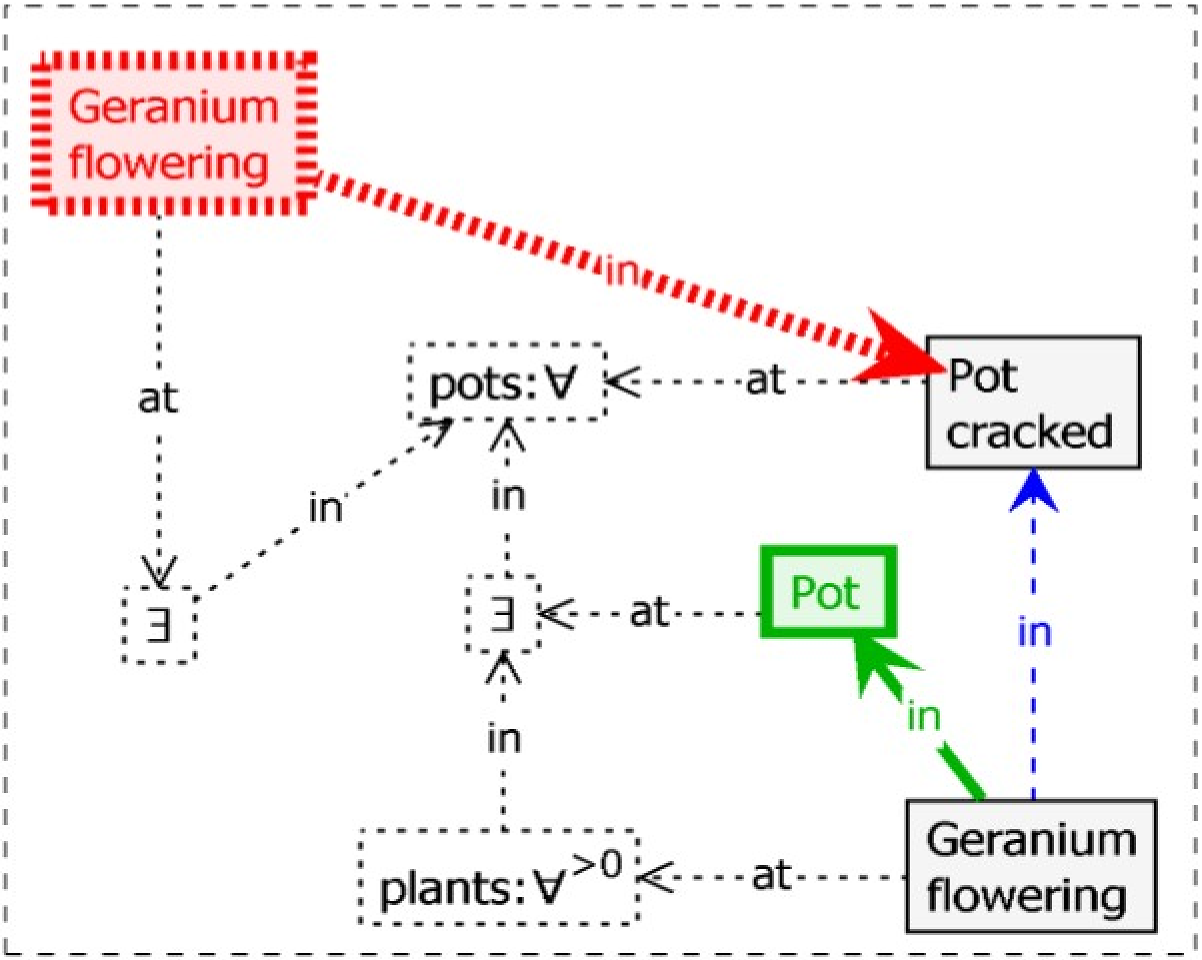
T-Core

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.

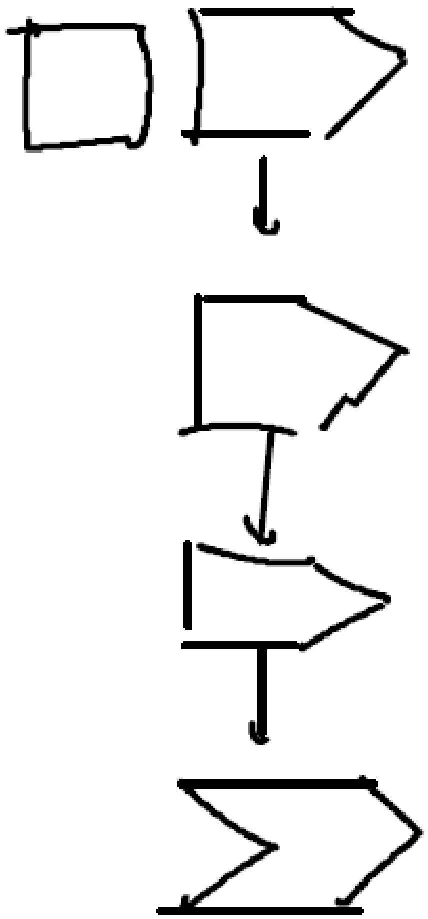
https://journal.ub.tu-berlin.de/eceasst/article/view/260

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

ATL

medini QVT

SmartQVT

A Tool for Multi-formalism and Meta-Modeling

Even our logos are modeled!

Visit MSDL at http://msdl.cs.mcgill.ca

FUJABA Tool Suite

MoTif

GrGen.NET

AGG

epsilon

VIATRA2

EMF-IncQuery

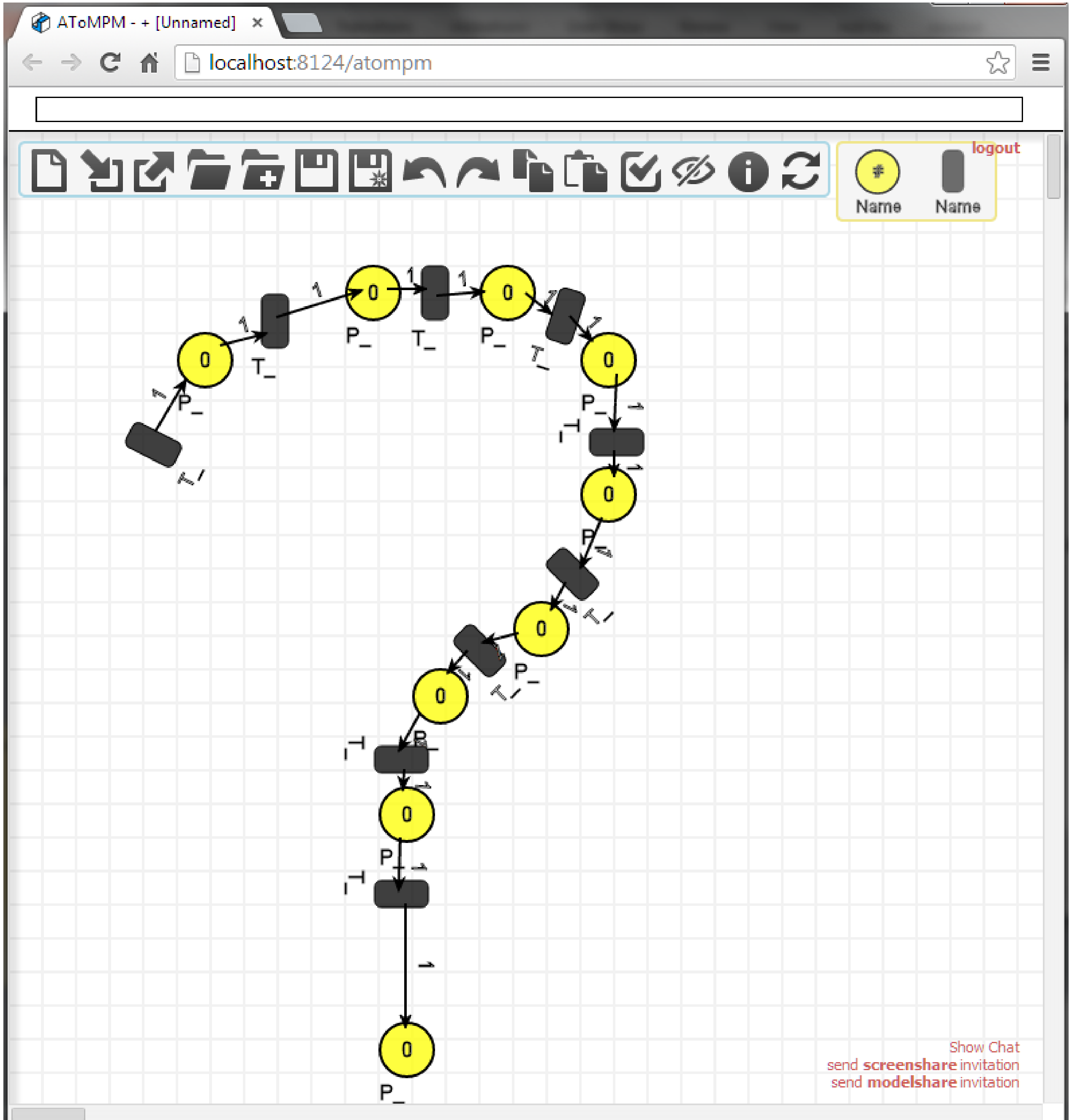
Henshin emf

moflon

MOF LON

GReAT DSLTrans ProGReS MOLA

VMTS


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