



DSM TP 2017

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

**Montreal, Canada
10-14 July 2017**

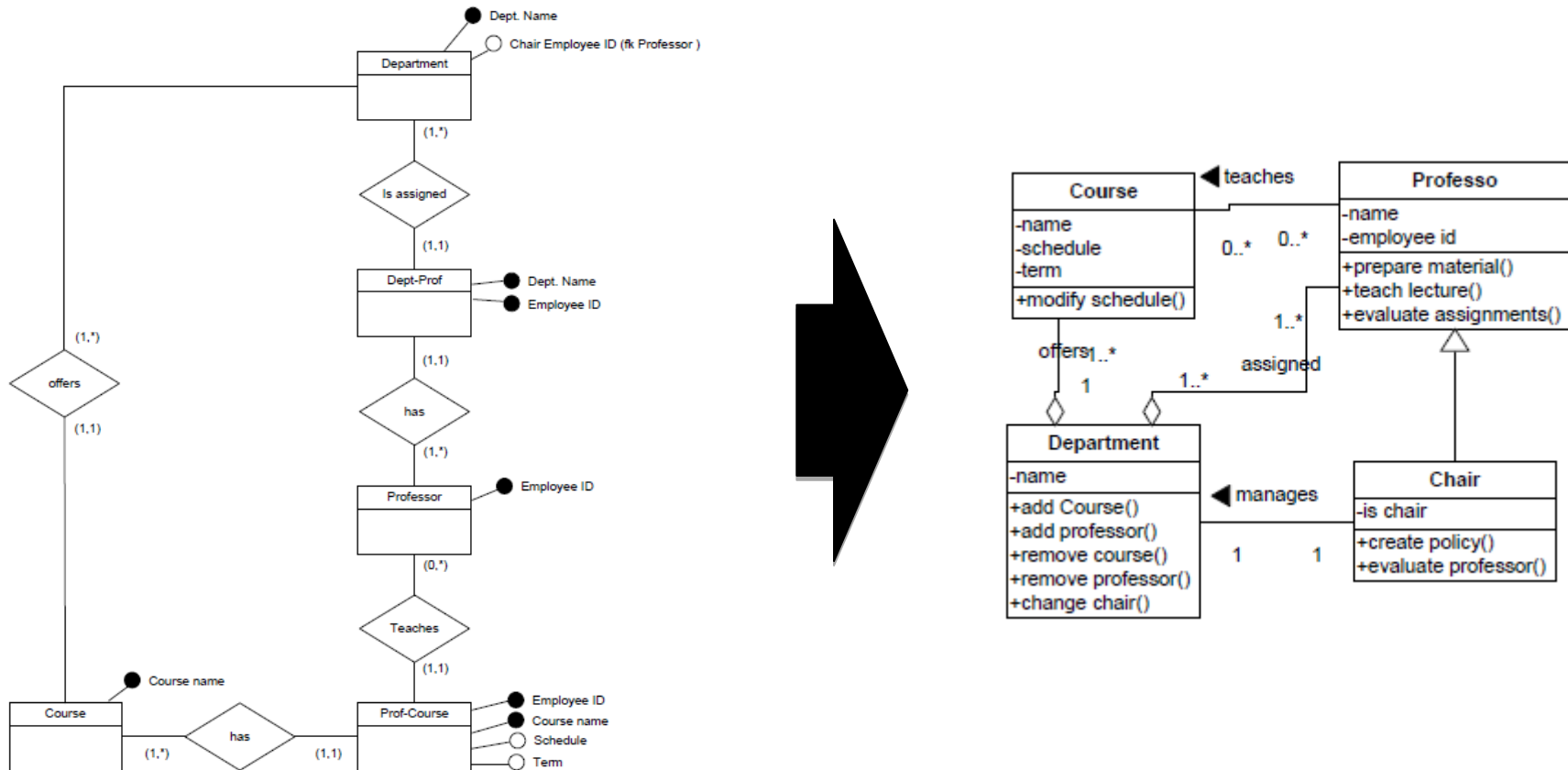
Model Transformation

Eugene Syriani

with a little help from Hans Vangheluwe

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!

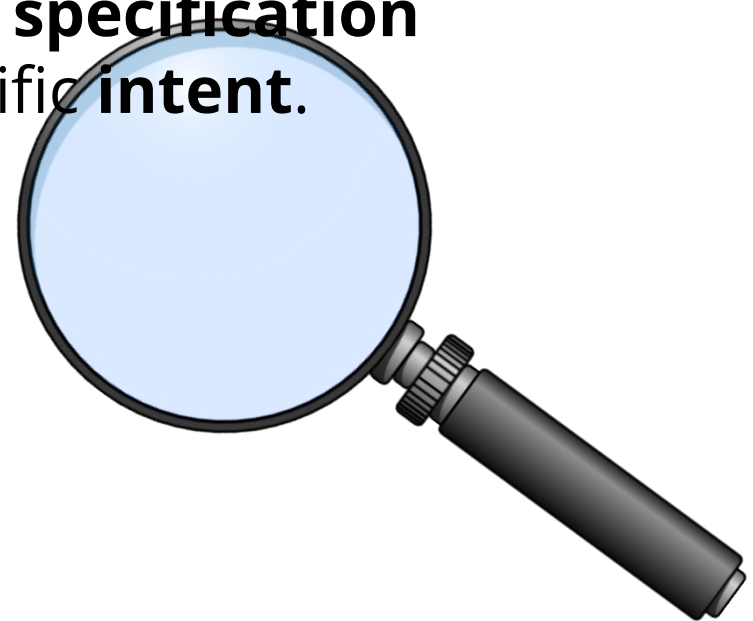
What's the difference?

- 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**

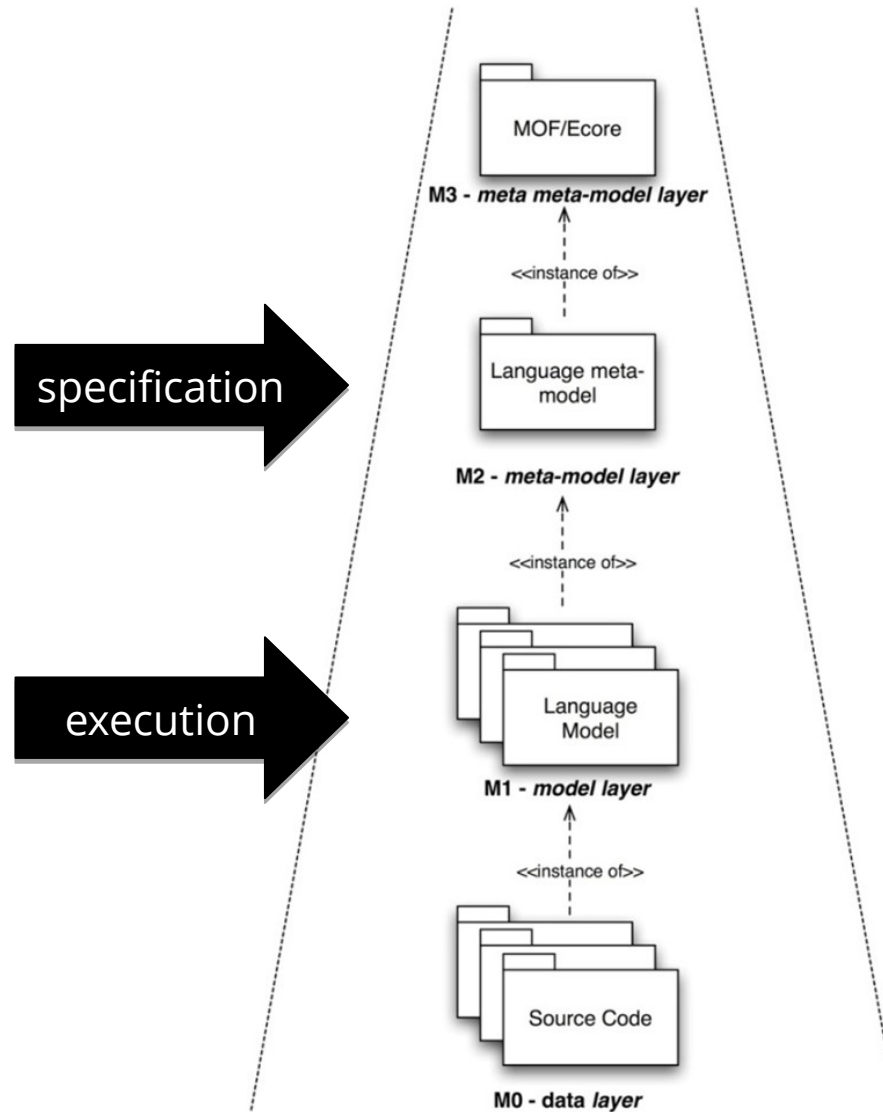
What is a model transformation?

Definition

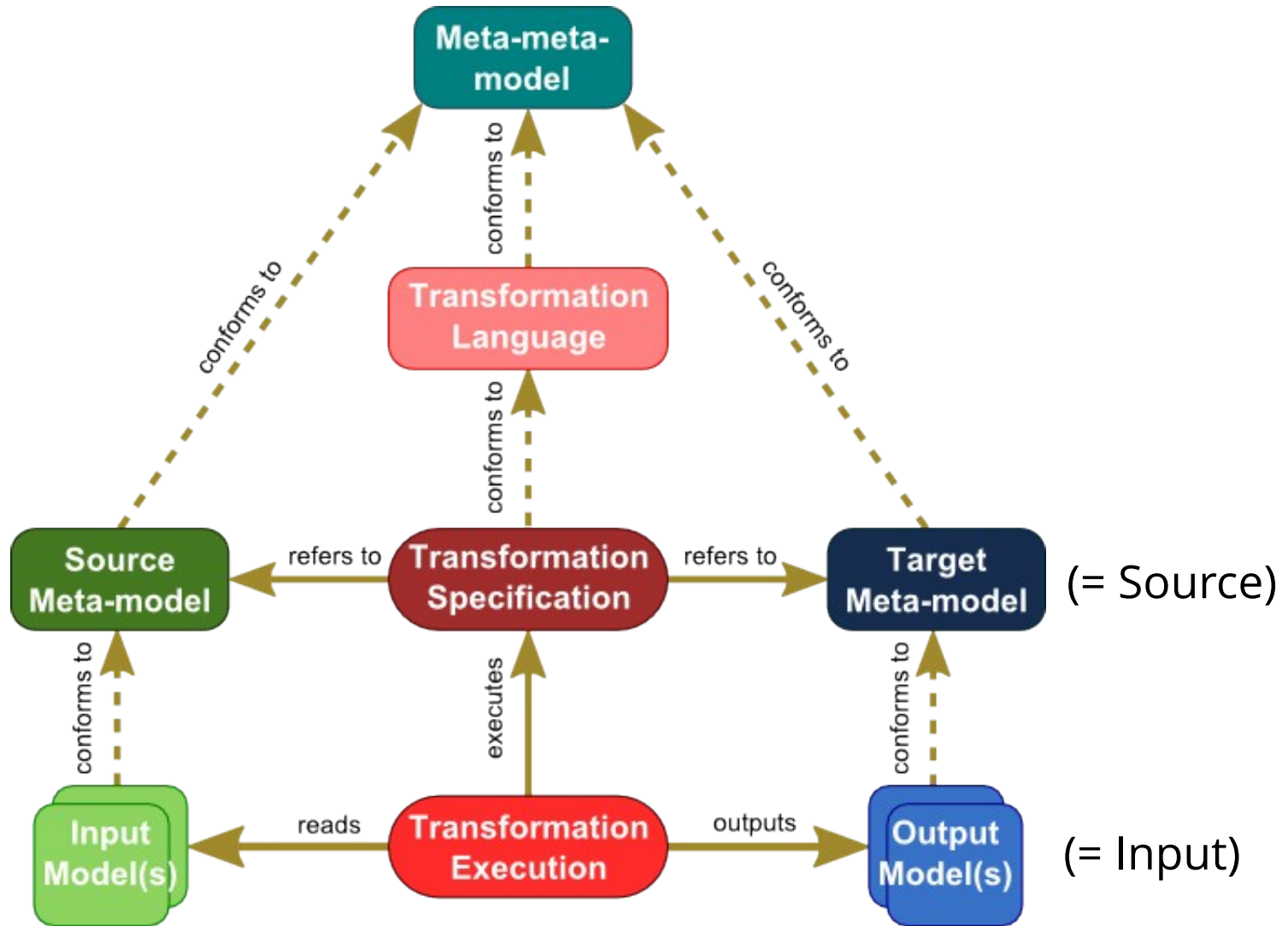
A model transformation is the **automatic** manipulation of **input** models to produce **output** models, that conforms to a **specification** and has a specific **intent**.



Where should MT be specified and executed?



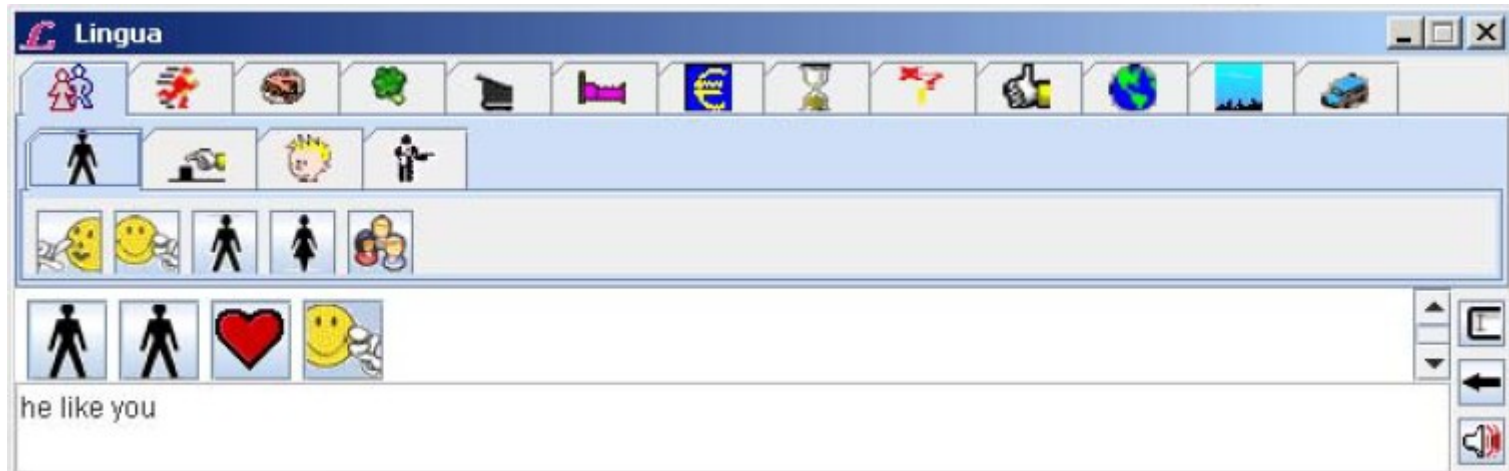
Terminology



Data structures to transform

Sequence

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



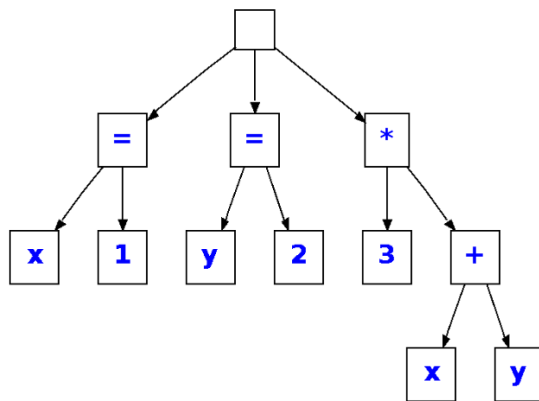
String rewriting

- 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**



```

<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

```

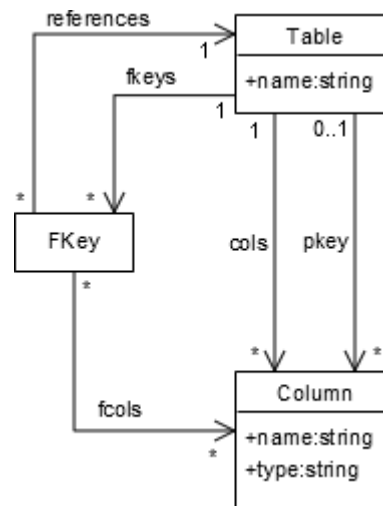
root expr(->X)
nonterm expr(->Expr)
  rule expr(->X): expr2(->X)
  rule expr(->add(X,Y)): expr(->X) "+" expr2(-
>Y)
nonterm expr2(->Expr)
  rule expr2(->mult(X,Y)): expr2(->X) "*"
expr2(->Y)
  rule expr2(->num(X)): Number(->X)
token Number(->INT)

```

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**

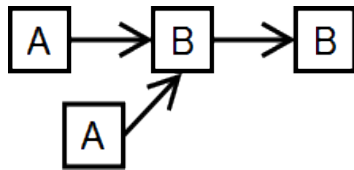


Graph transformation

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

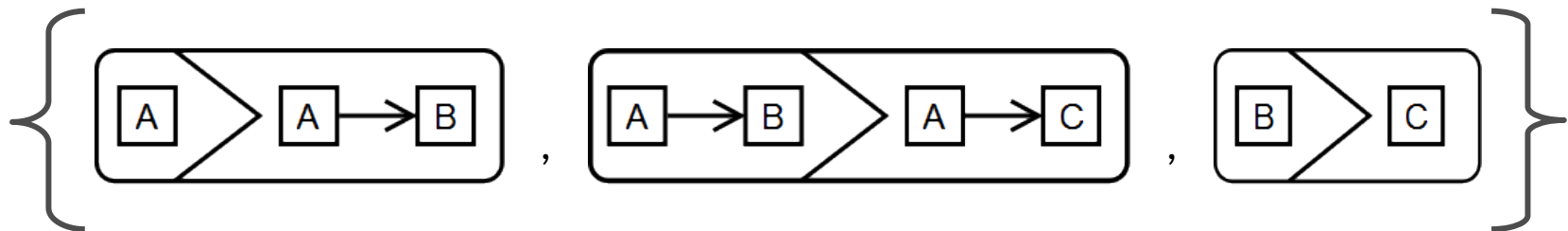
- T-Core

- 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

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

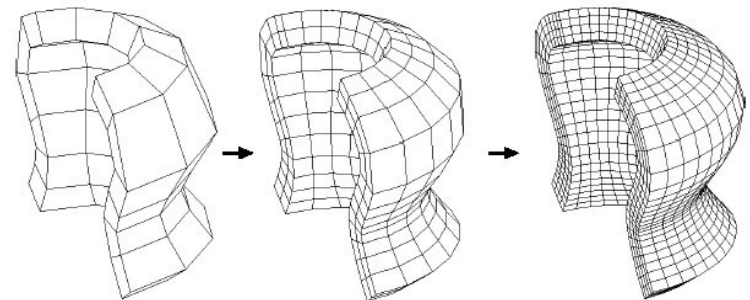
Model Composition

- Model Merging
- Model Matching
- Model Synchronization

Refinement category

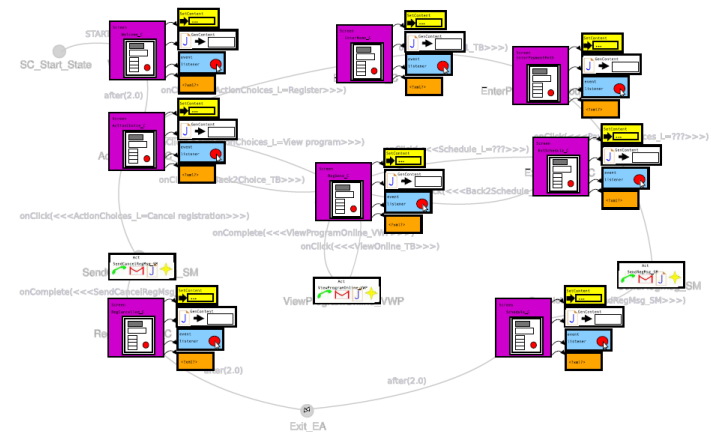
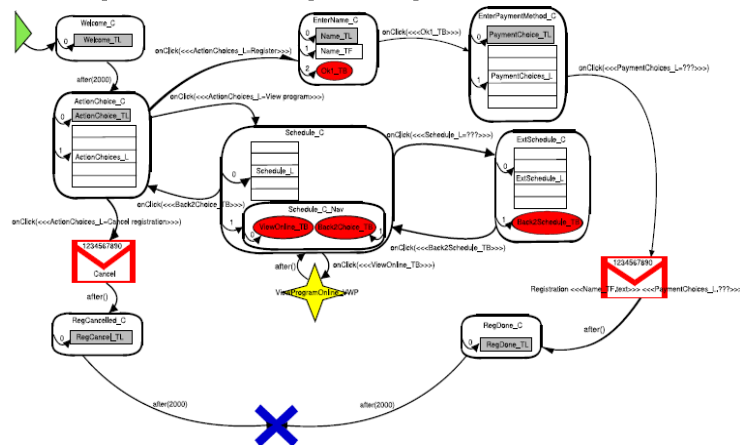
Groups intents that produce a more precise model by reducing design choices and ambiguities with respect to a target platform.

- 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

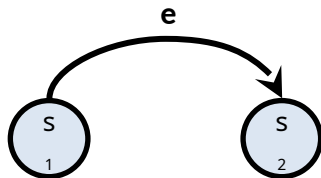


PhoneApps DSL of a conference registration mobile application Representation of the model in AndroidAppScreens

PhoneApps DSL To Android Activities

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



Statecharts to Python Compiler

```

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)

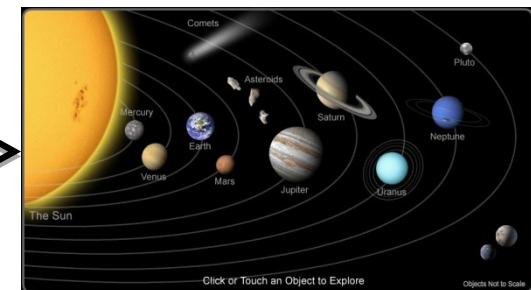
self.runEnterActionsForStates(self.StatesEntered, 1)
self.applyMask(DigitalWatchStatechart.OrthogonalTable[0], table)
    handled = 1
  
```

Generated Python
code

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:

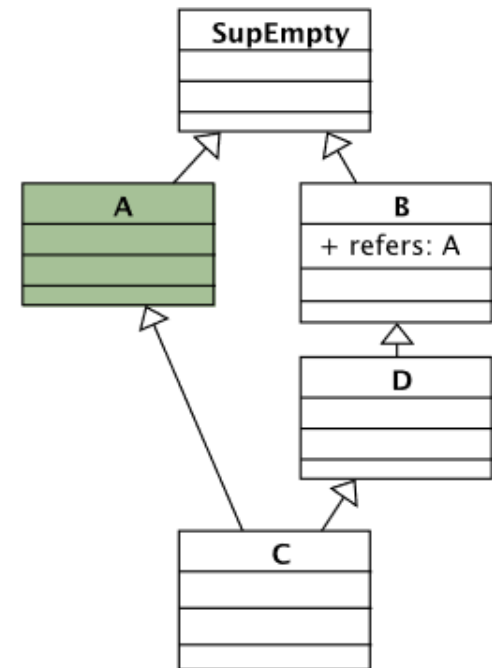
"Find all actors who played together in at least 3 movies and assign the average rating to each clique" outputs a view of a model representing a subset of IMDB represented as a graph composed of strongly connected components with the ratings aggregating individual ratings.

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

```
1 pattern superClass(sub : Class, sup : Class) {
2   Generalization.specific(gen, sub);
3   Generalization(gen);
4   Generalization.general(gen, sup);
5 }
6
7 pattern hasOperation(cl : Class, op : Operation) {
8   Class.ownedOperation(cl, op);
9 } or {
10  find superClass+(cl, owner);
11  Class.ownedOperation(owner, op);
12 }
13
14 pattern emptyClass(cl : Class) {
15  neg find hasOperation(cl, _op);
16  neg find hasProperty(cl, _pr);
17  Class.name(cl, n);
18  check(!(n.endsWith("Empty")));
19 }
```



Semantic Definition category

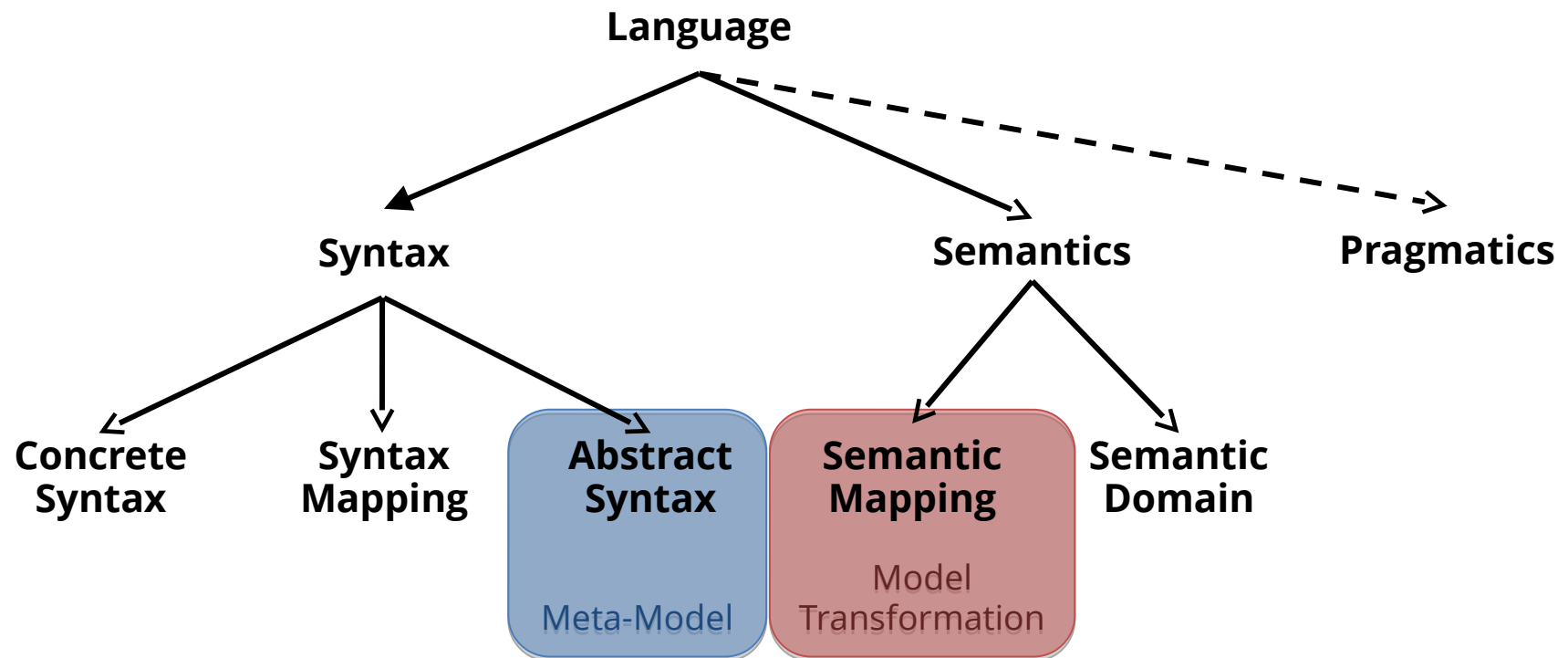
Groups intents whose purpose is to define the semantics of a modeling language.

- Translational Semantics (model-to-model)
- Operational Semantics
(simulation by graph transformation)



Translational Semantics

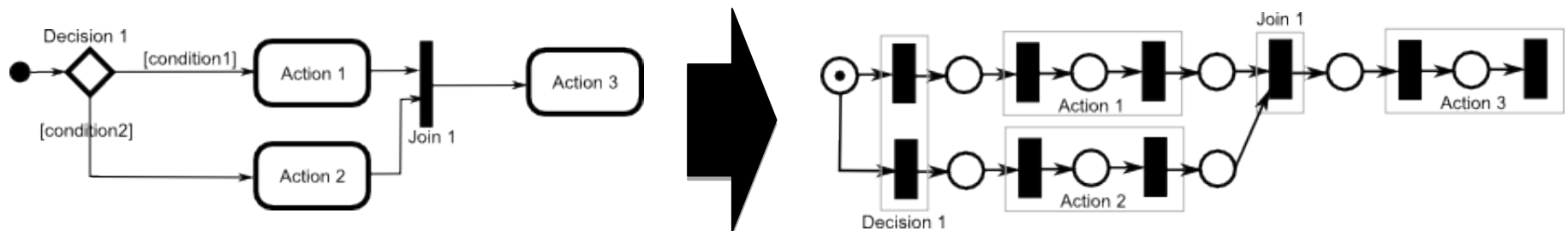
- Gives the **meaning** of a model in a source language in terms of the concepts of another target language
- Typically used to capture the semantics of **new DSLs**



Translational Semantics

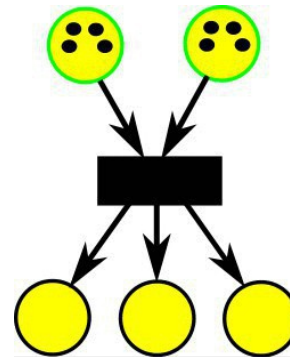
- Simulink Block Diagram's semantics expressed as Ordinary Differential Equations

- UML activity diagrams semantics expressed as Petri nets



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

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 \neq 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

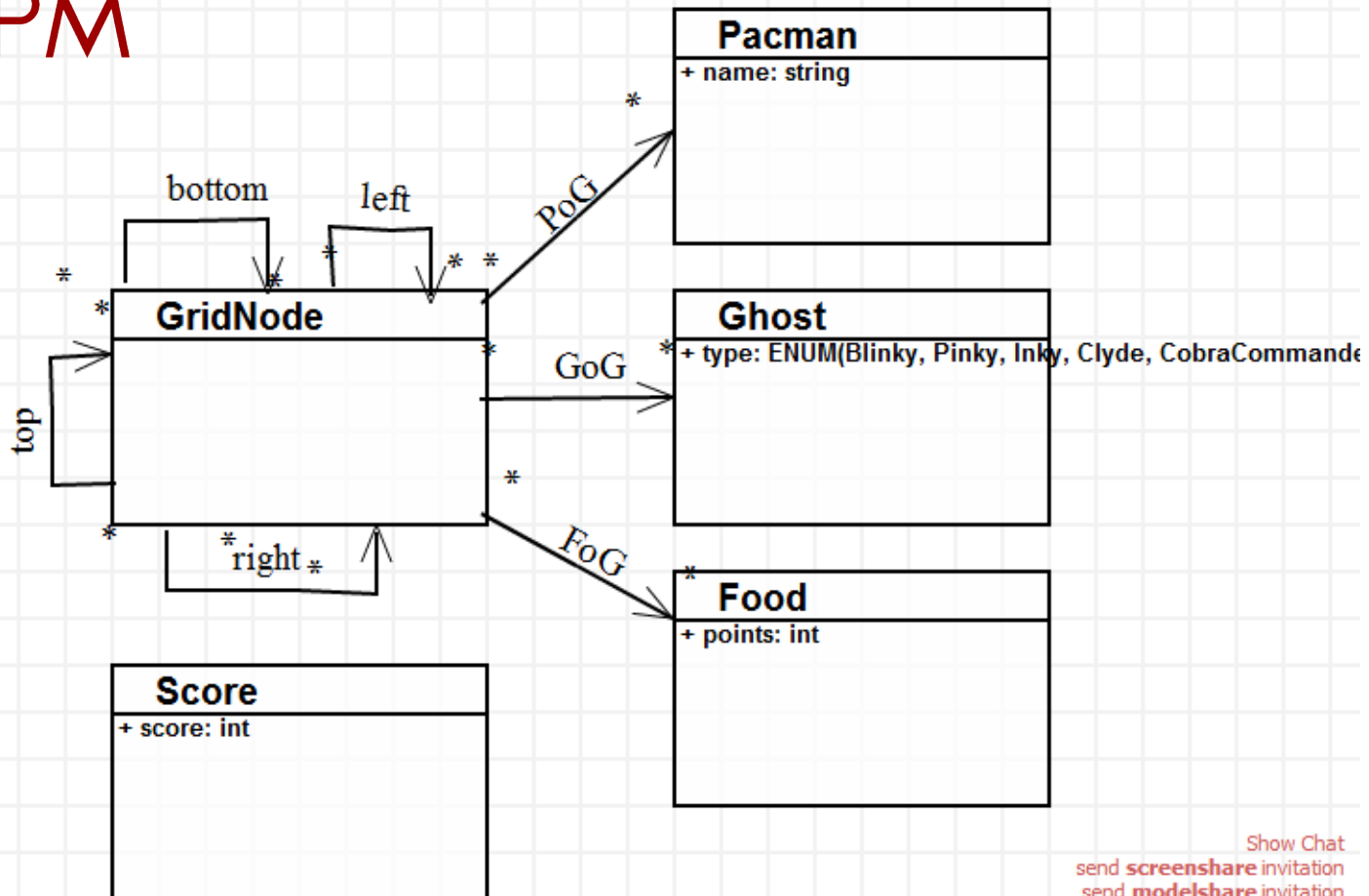
Graph transformation for simulation

- 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



ATOMPM



Show Chat
send [screenshot](#) invitation
send [modelshare](#) invitation

Concrete syntax

logout

Score

Pacmanicon

Ghosticon

GridNodelcon

Foodicon

Scoreboardicon

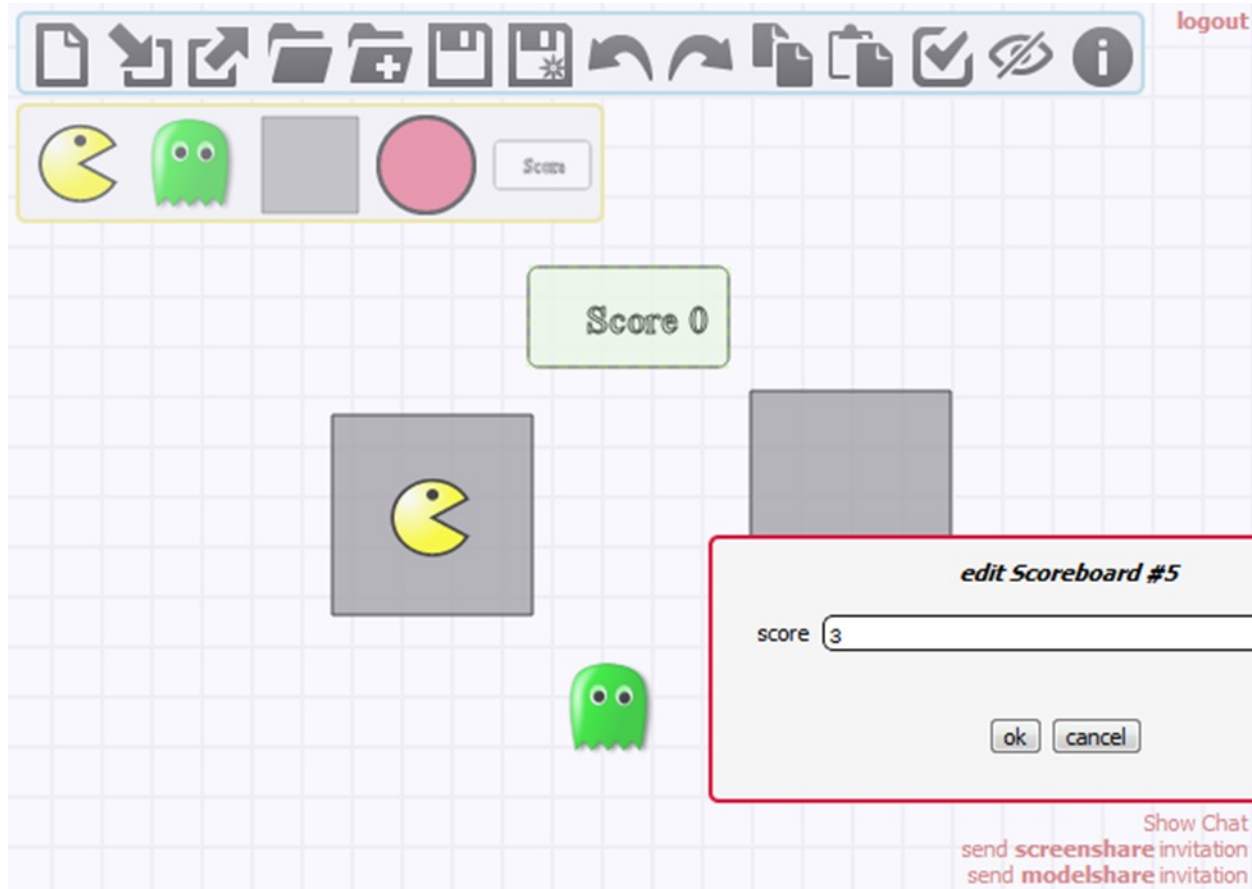
PoGLink

FoGLink

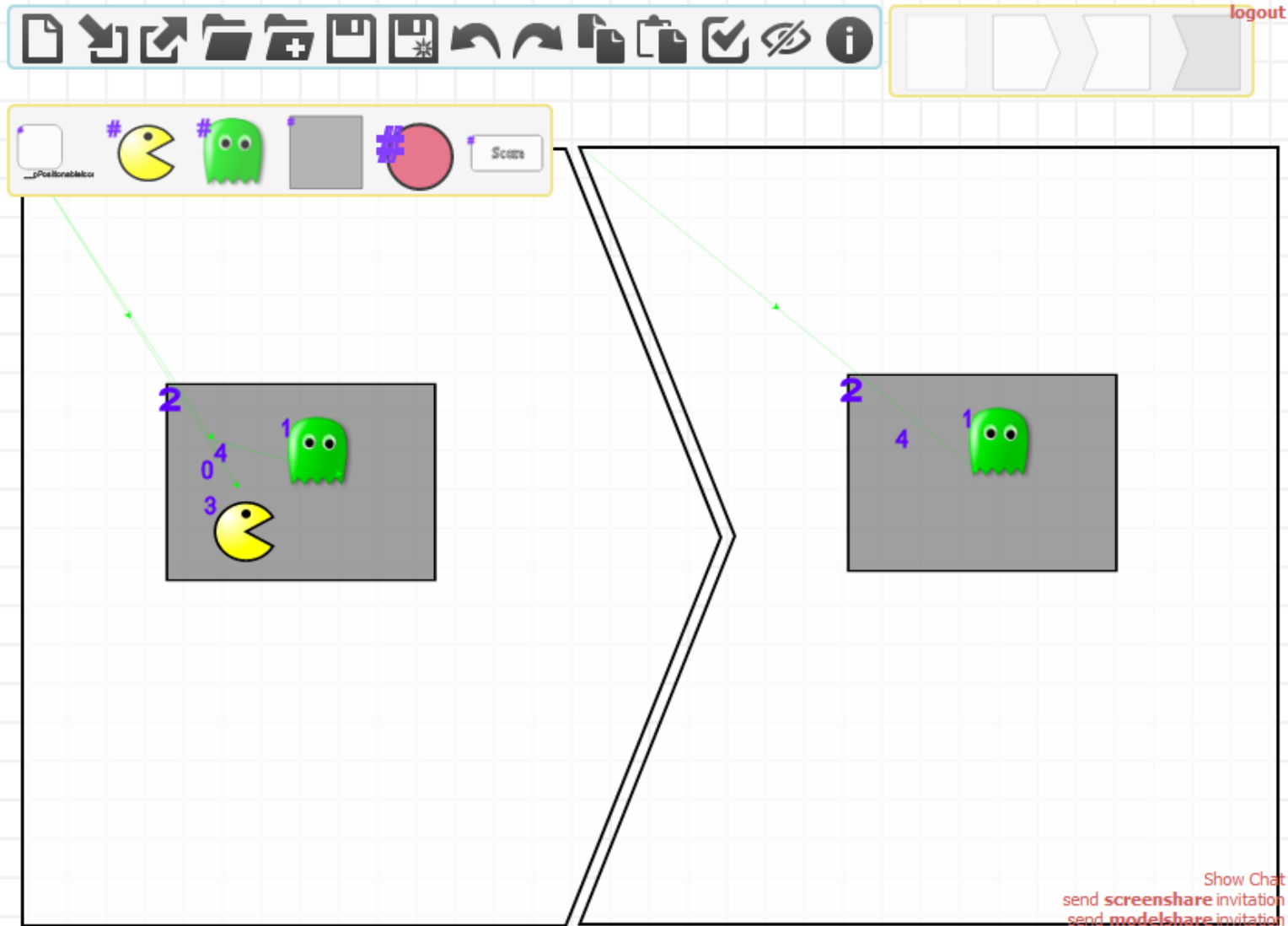
GoGLink

Show Chat
send [screenshare](#) invitation
send [modelshare](#) invitation

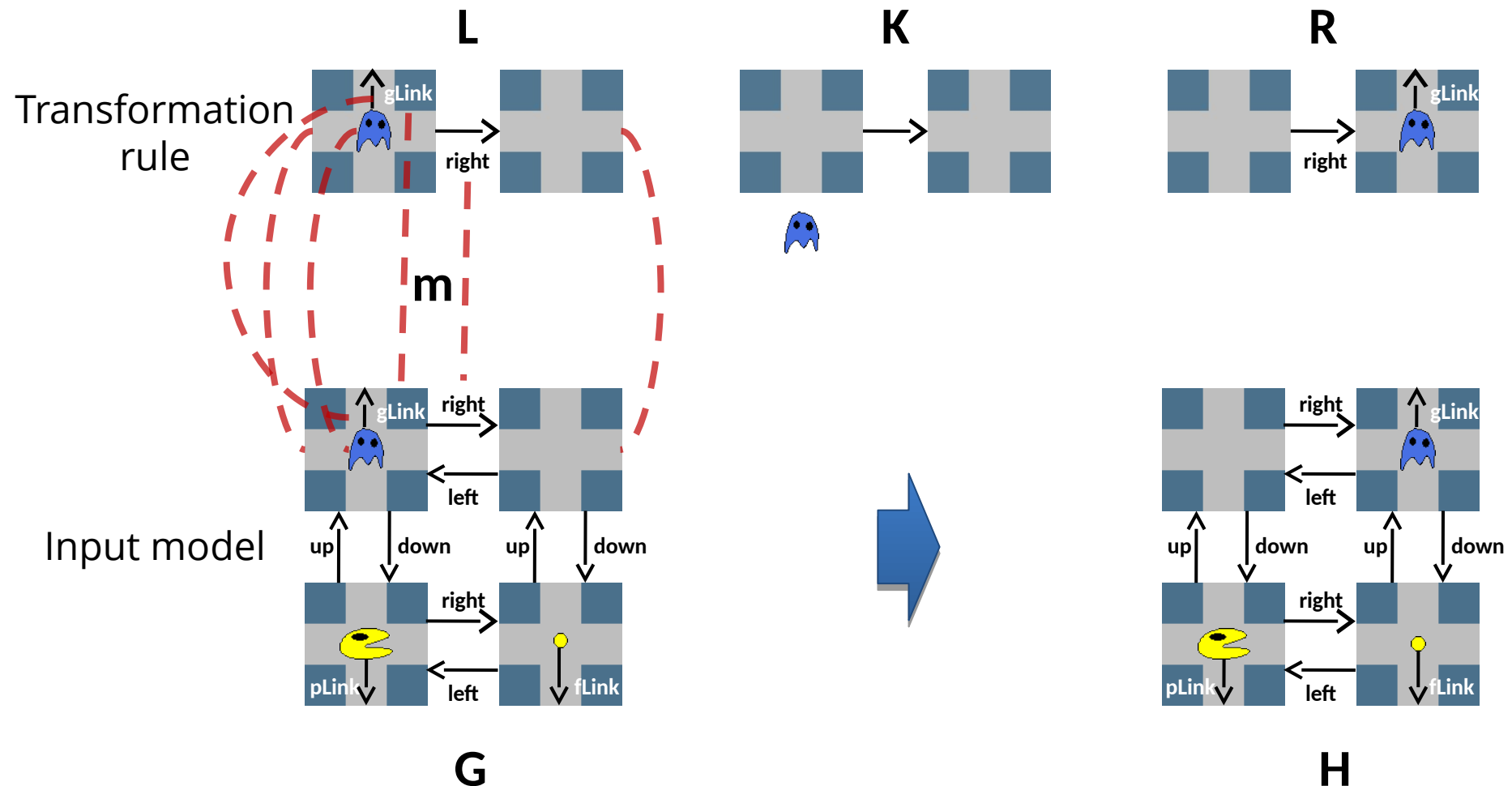
Generate modeling environment



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:

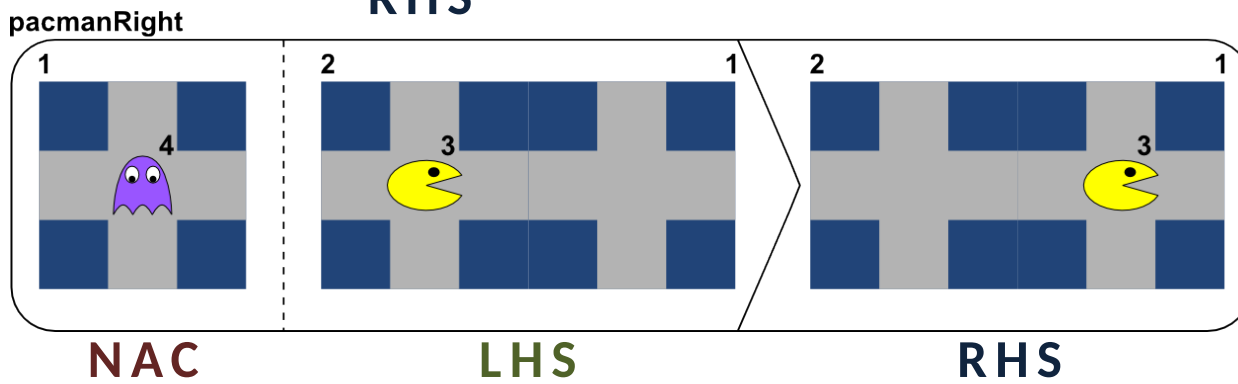
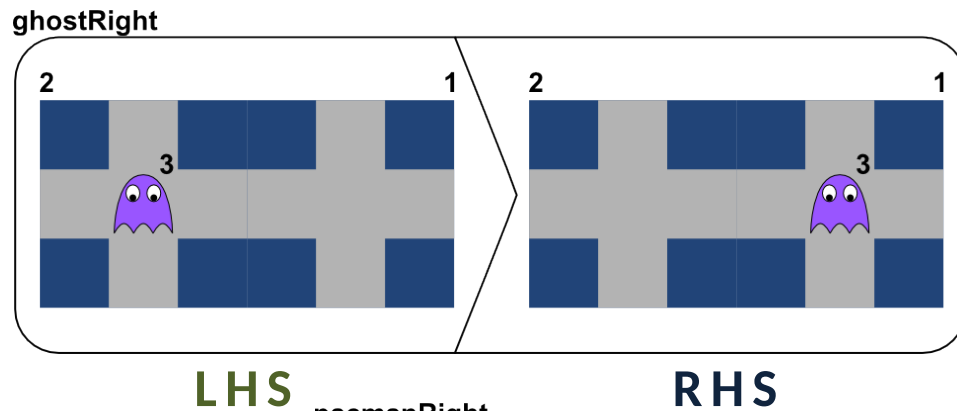
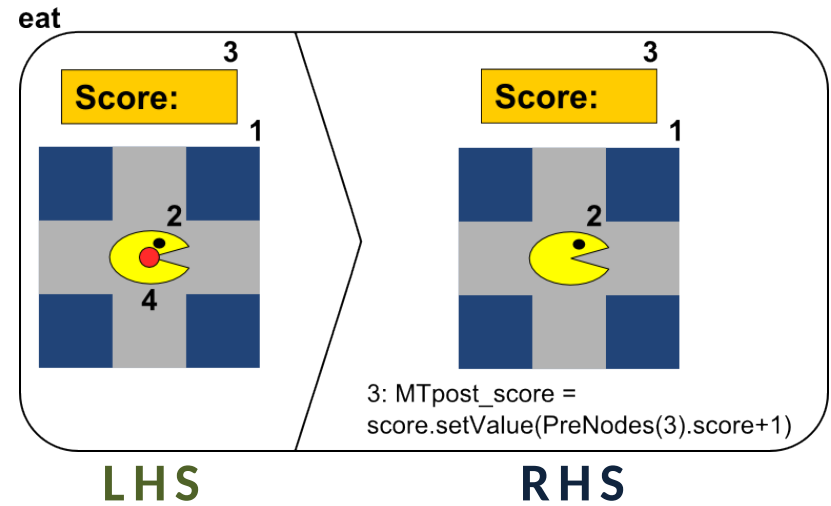
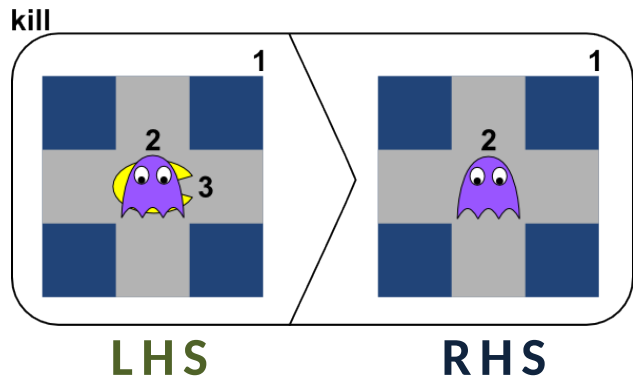
- **Remove** all elements from $m(L - K)$ from G
 - **Create** the new elements of $R - K$ in G
 - **Update** the properties of the elements in $m(L \cap K)$
- When a match of the LHS can be found in G , the rule is **applicable**
 - When the rewriting phase has been performed, the rule was **successfully applied**

QUESTION

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

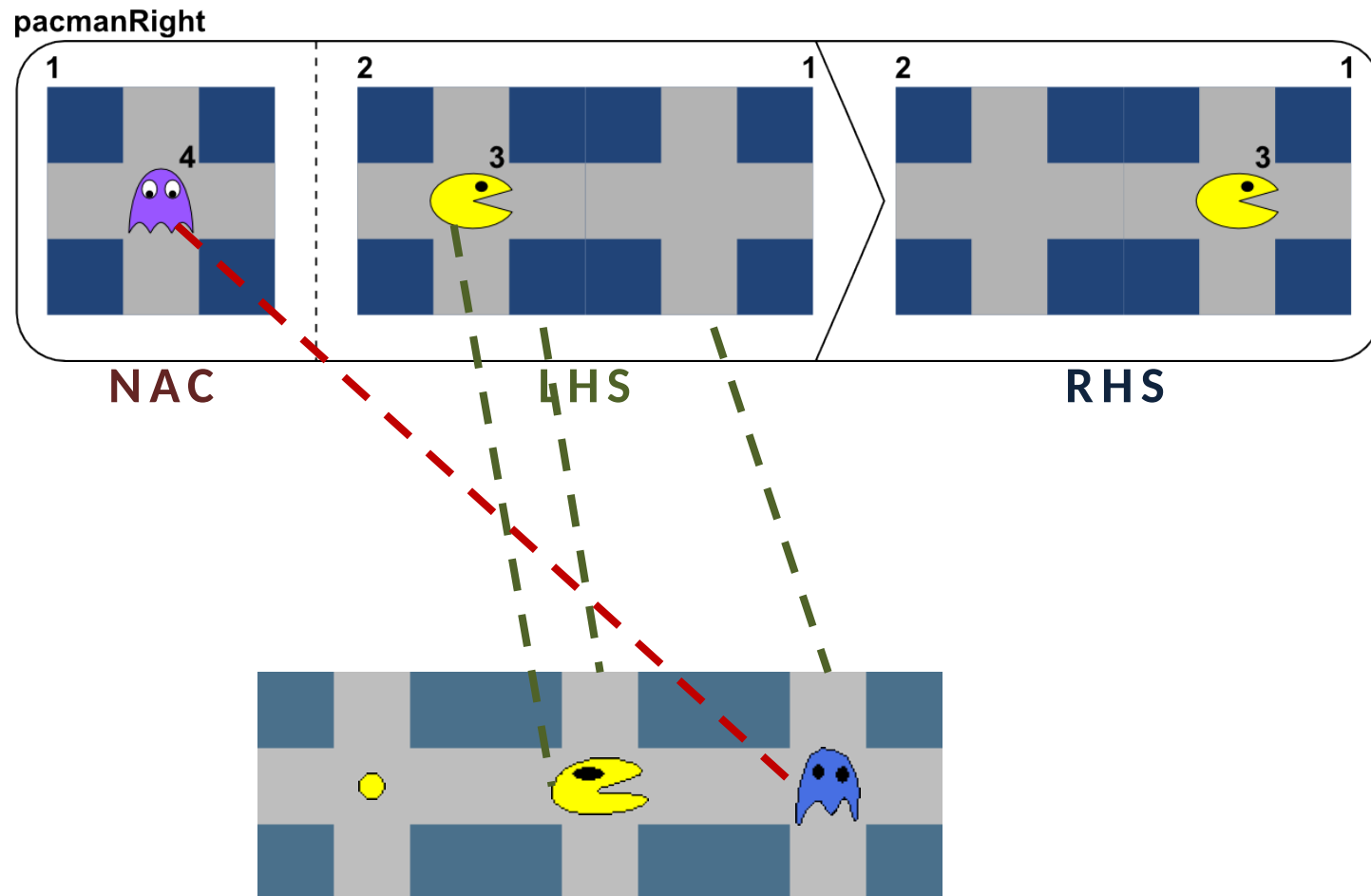
➤ ~~$\mathcal{O}(\text{CRUD operations})$~~ $\mathcal{O}(|G|^{L|L|})$ CRUD operations

Operational semantics



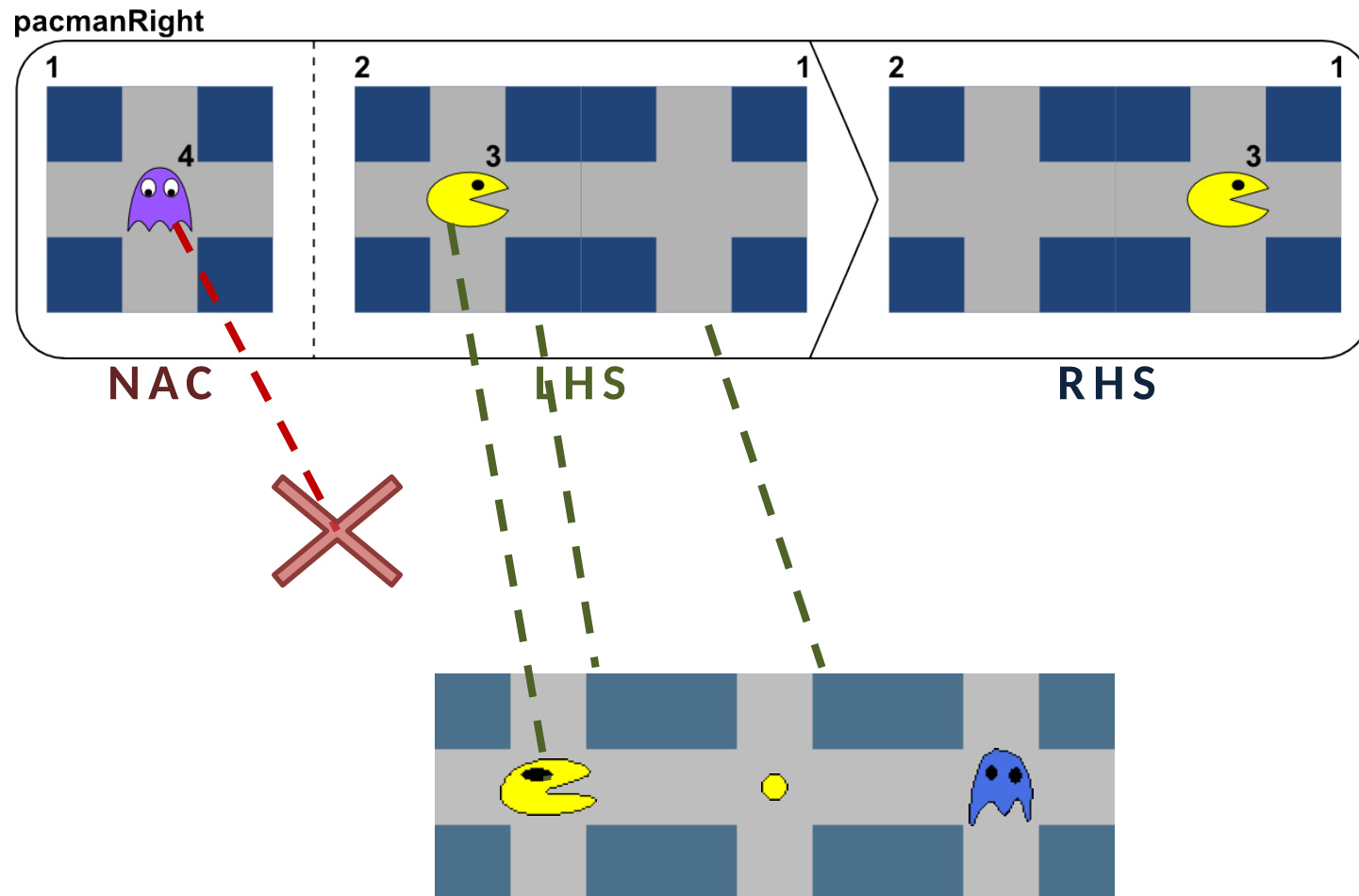
Negative application conditions

Non-applicable rule



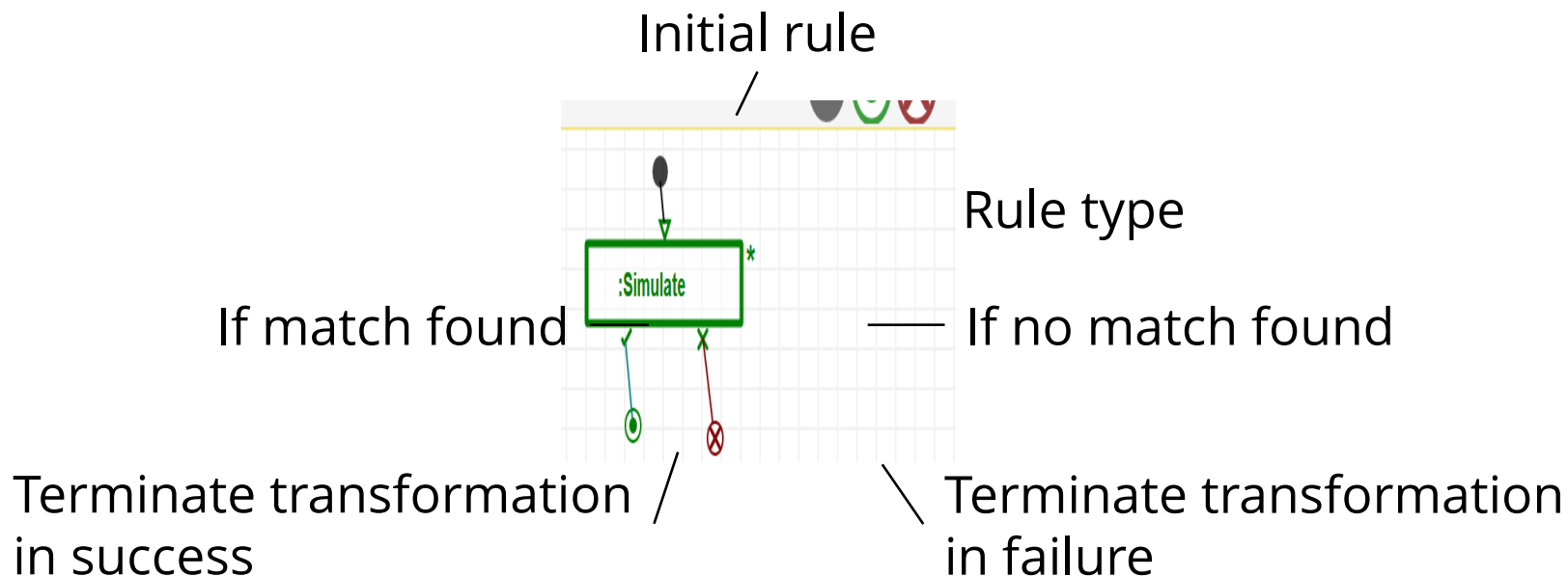
Negative application conditions

Applicable rule

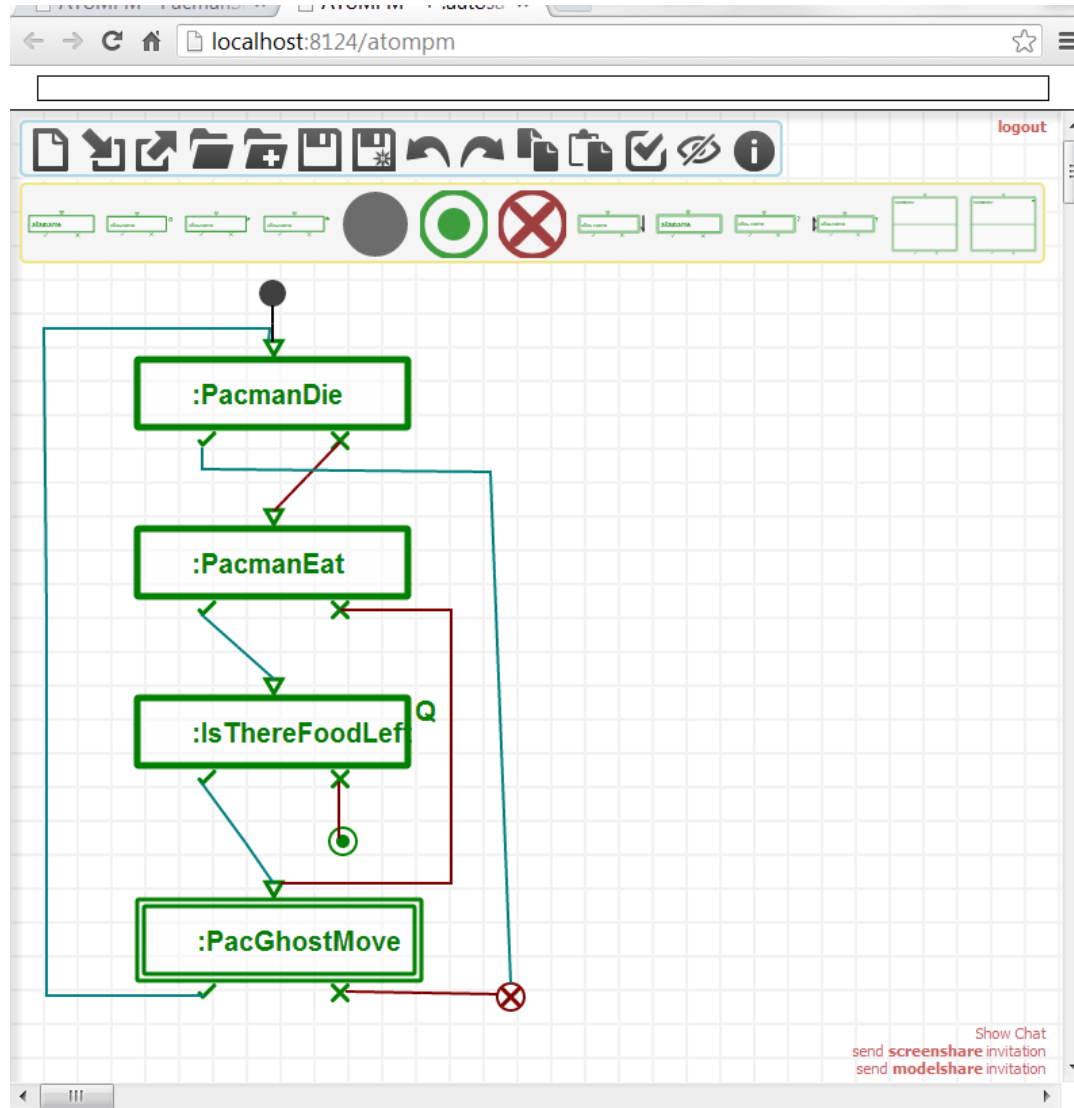


Rule scheduling

- 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



Scheduling of the rules



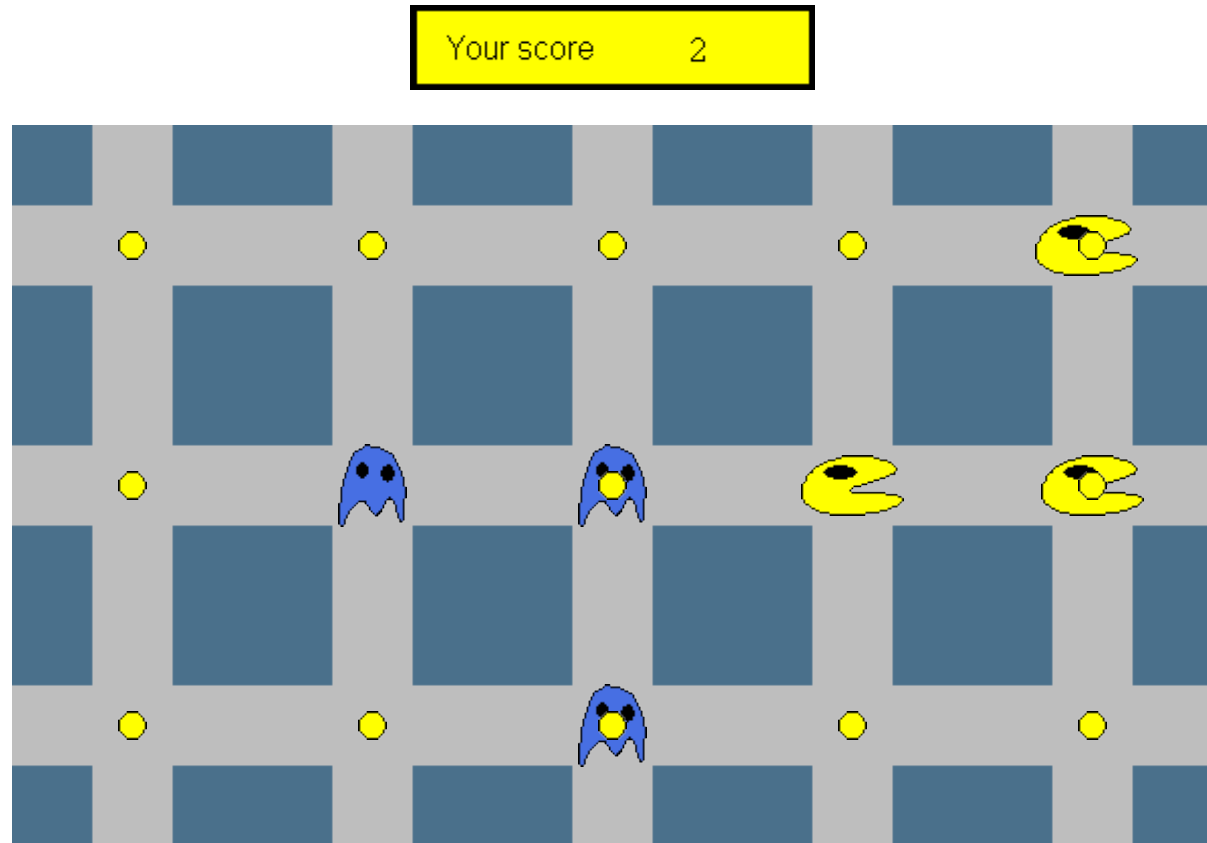
QUESTION

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

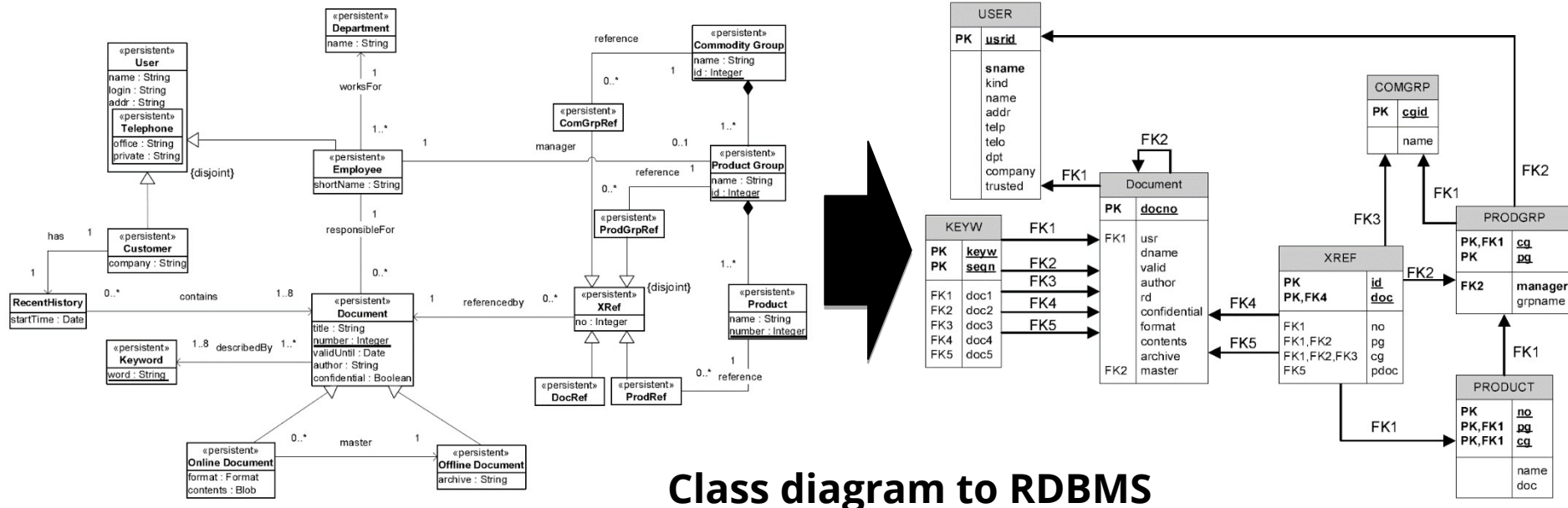
Simulation of a model

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



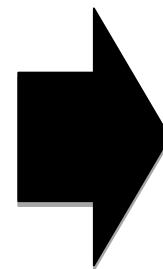
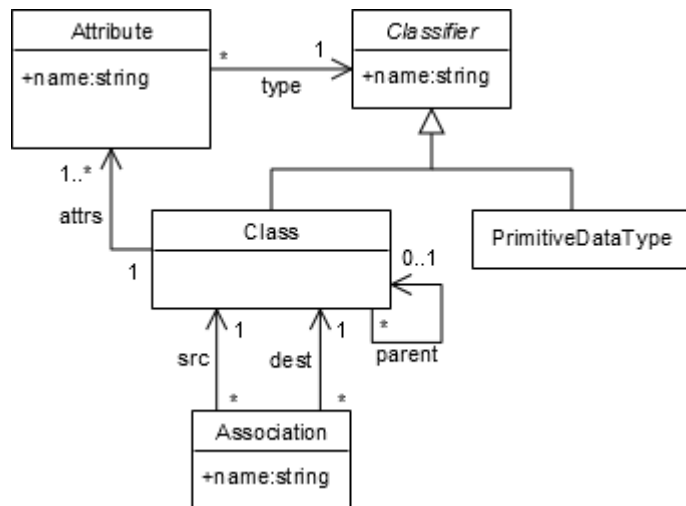
Translation

- **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

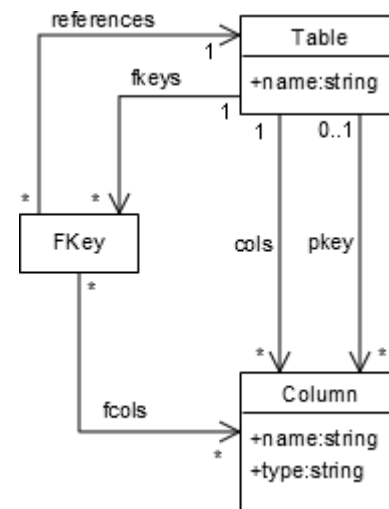


CD to RDBMS transformation

CD metamodel

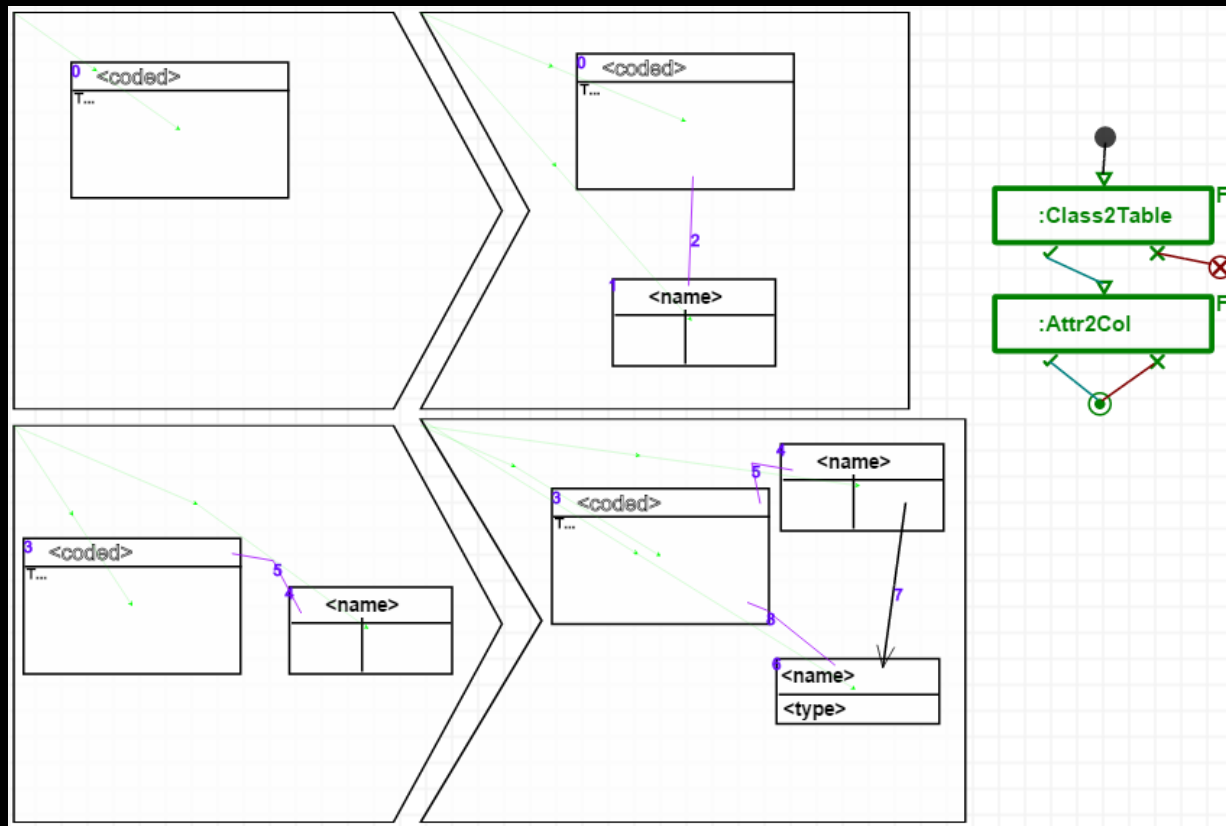


RDBMS metamodel

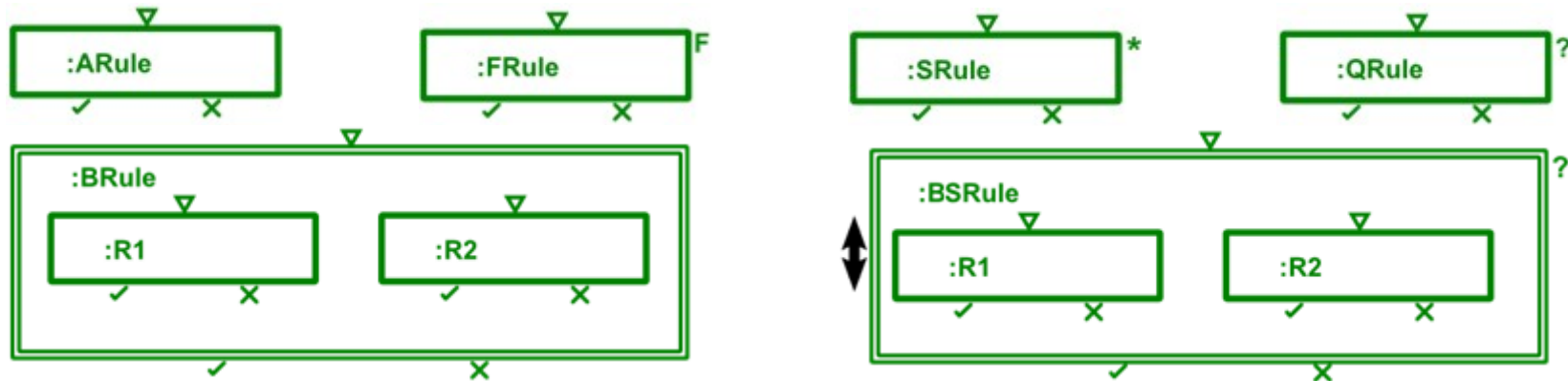


QUESTION

Implement in MoTif the transformation for:
Classes to tables
Attributes to columns

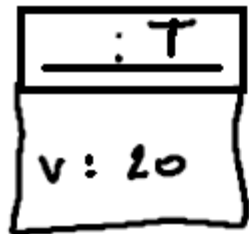


MoTif main rule types



- **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

Pattern model <> Instance model



Instance model



: PATTERN

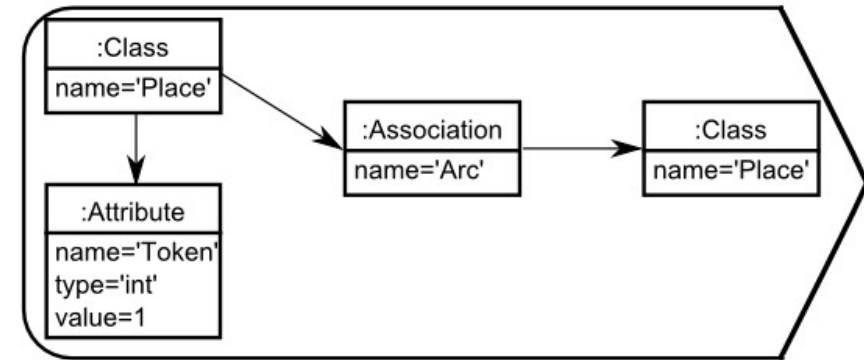
CONSTRAINT EXPRESSION

$0 \leq v \leq 10$

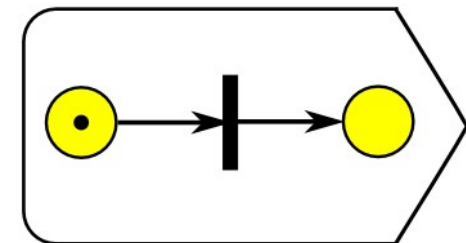
Pattern model

Pattern language

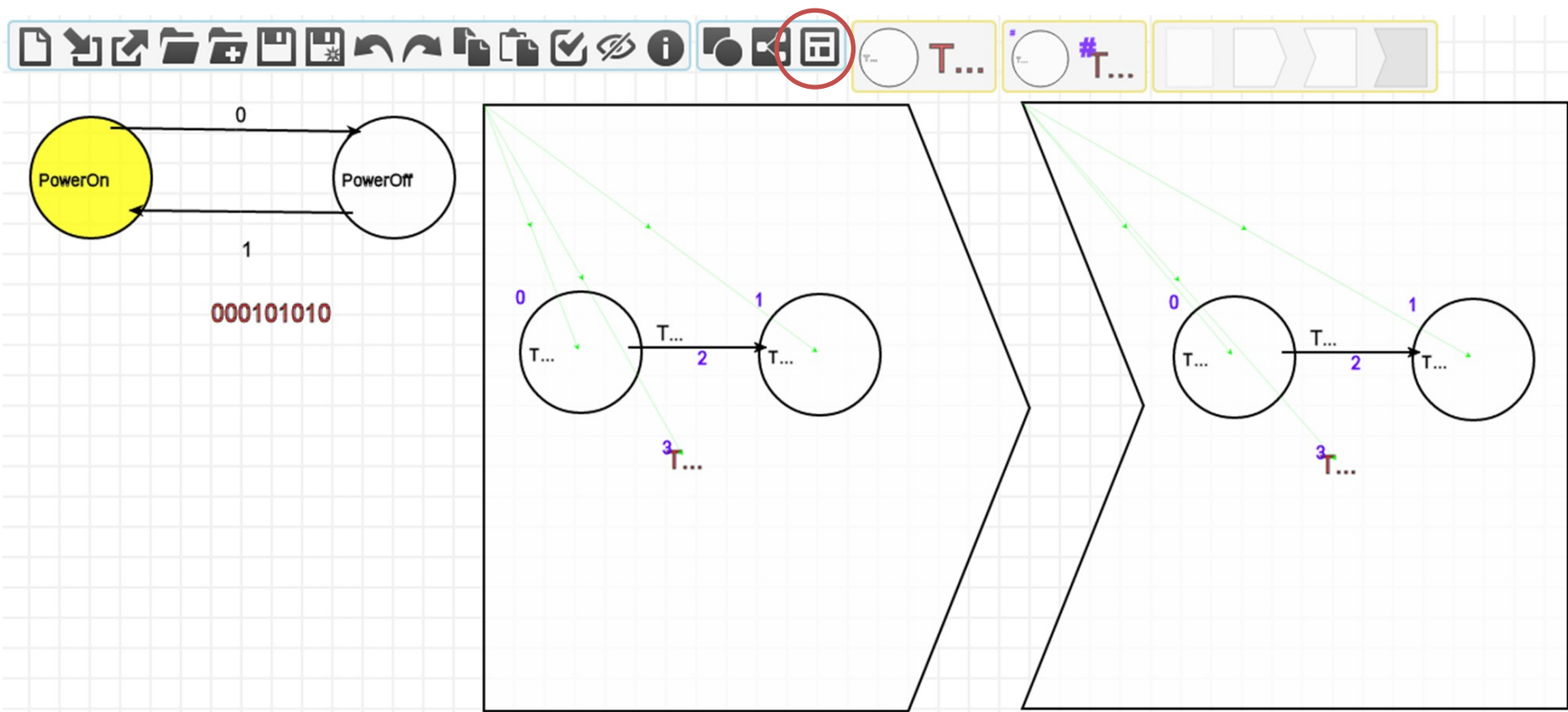
1. Generic pattern language
 - + Most economic solution
 - Generic concrete syntax (MOF-like)
 - Allow to specify patterns that will never occur



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



RAMification process

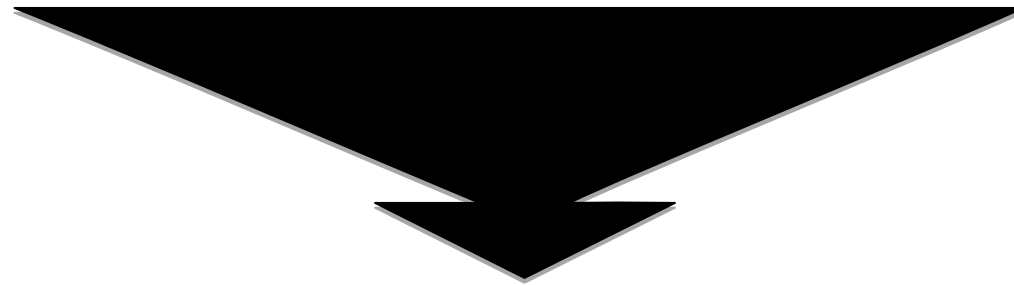


Domain-specific pattern languages

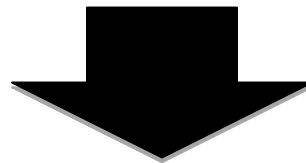
Ramification Process: automatically generated environment for pattern language

Input Meta-Model

Output Meta-Model

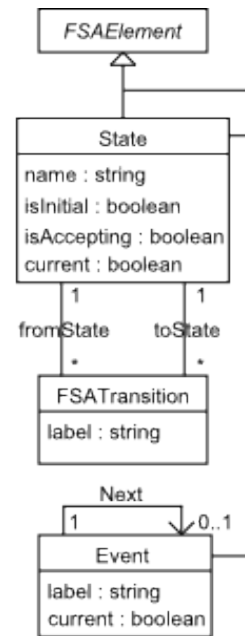


Relax Augment Modify



Customized Pattern Meta-Model

RAMification process: starting point meta-model(s)



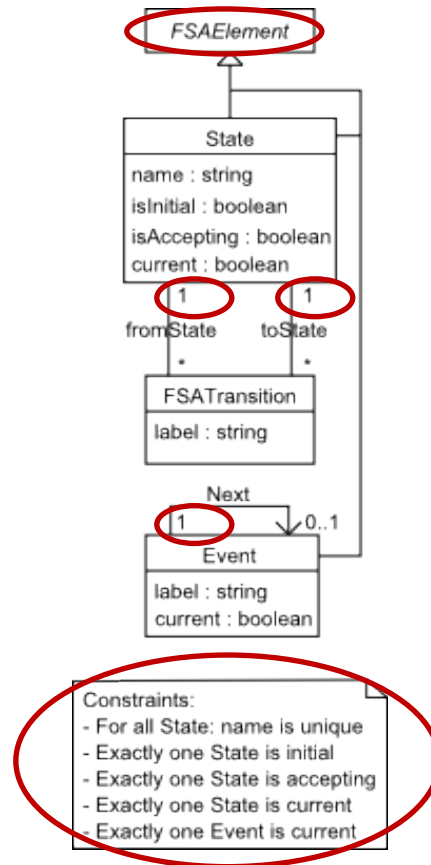
- Constraints:
- For all State: name is unique
 - Exactly one State is initial
 - Exactly one State is accepting
 - Exactly one State is current
 - Exactly one Event is current

RAMification process

Relaxation

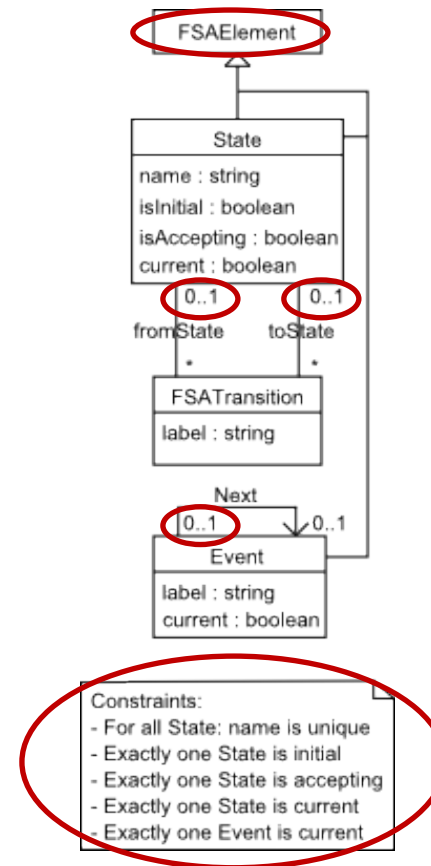
- Relaxes the constraints imposed by the meta-model(s) of the domain (source and target when exogenous)
- Make abstract classes concrete (to allow instantiation)
- Reduction of minimal multiplicity of every association end
- Constraints relaxation (manual)
 - Removed
 - Preserved
 - Depends on static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

RAMification process



Relax

RAMification process

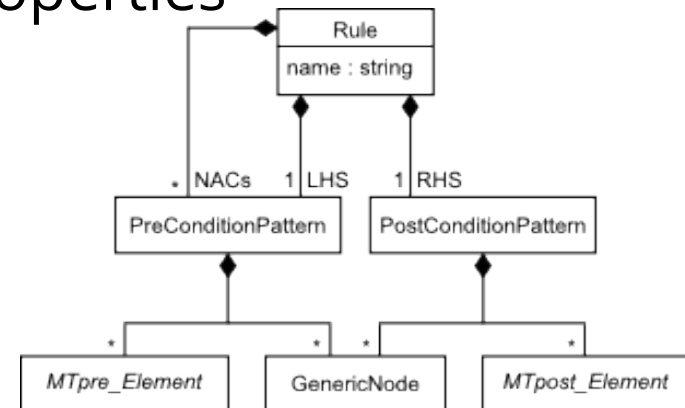


Relax

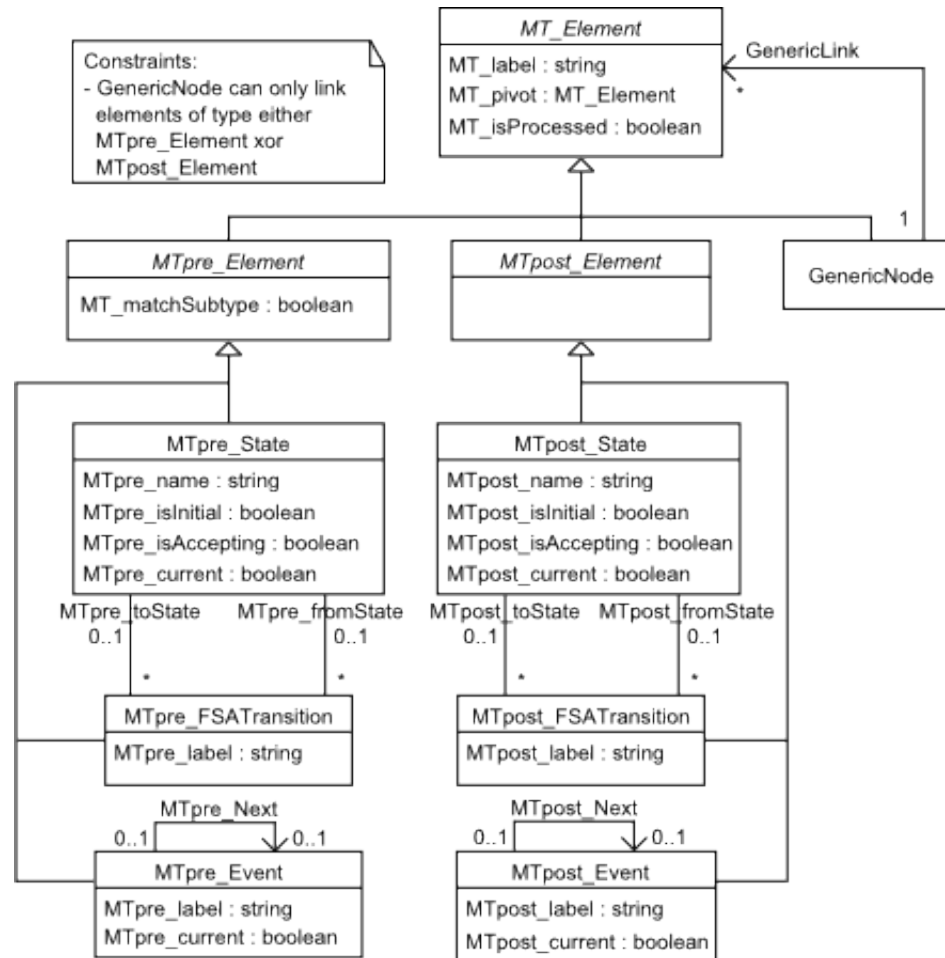
RAMification process

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



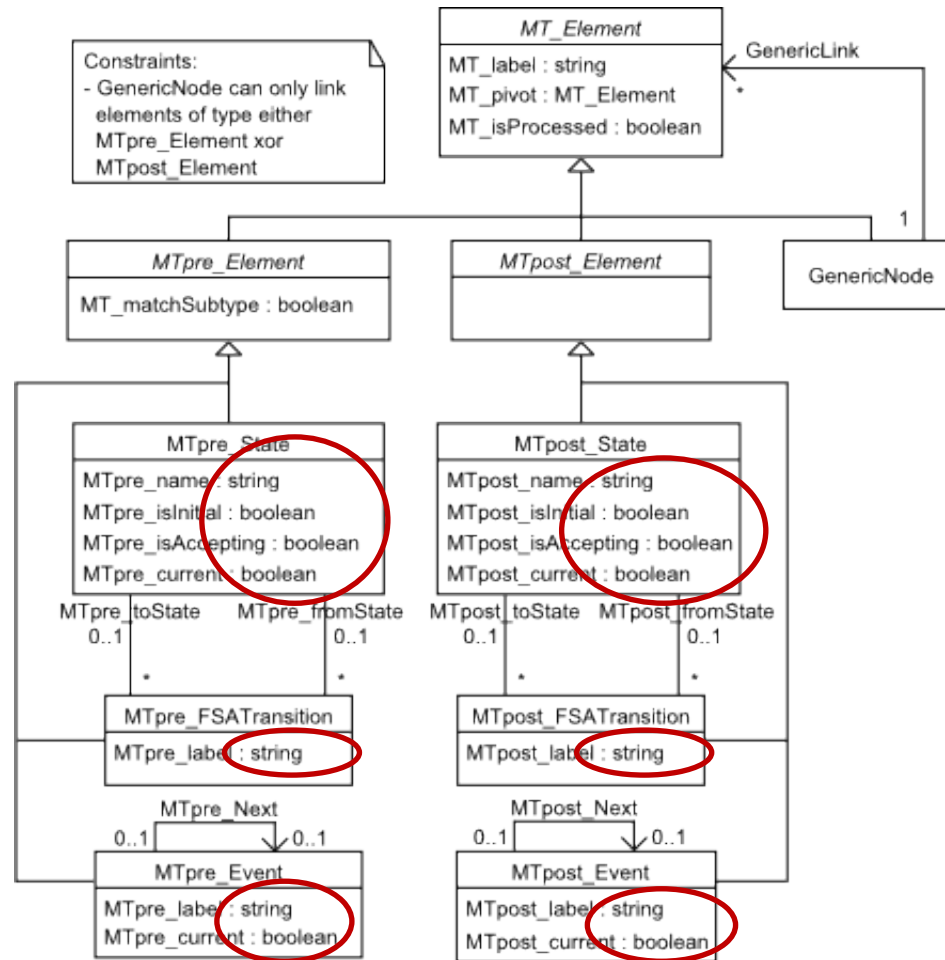
Augment

RAMification process

Modification

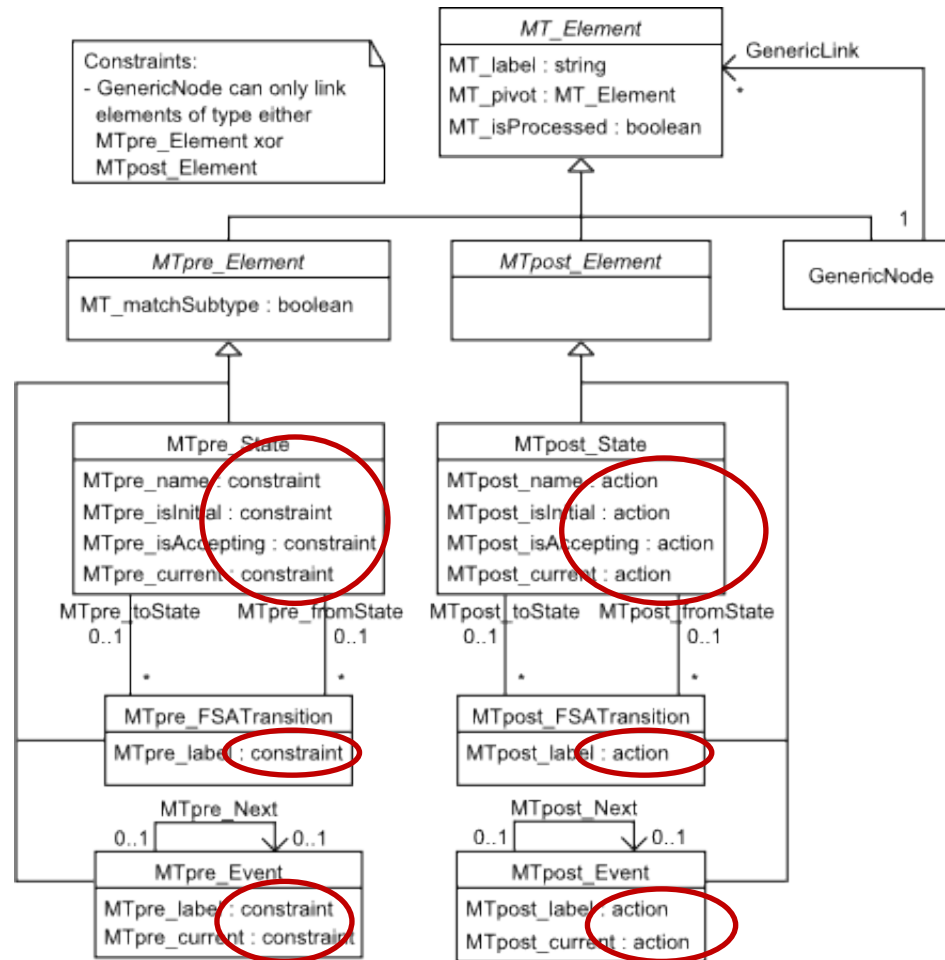
- Performs further modifications on the resulting meta-model
- Update namespaces
- Change type of attributes
 - Pre-condition classes: constraint type
 - Post-condition classes: action type
 - But preserve knowledge of original type for well-formedness
- Adaptation of concrete syntax (semi-automatic)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

RAMification process



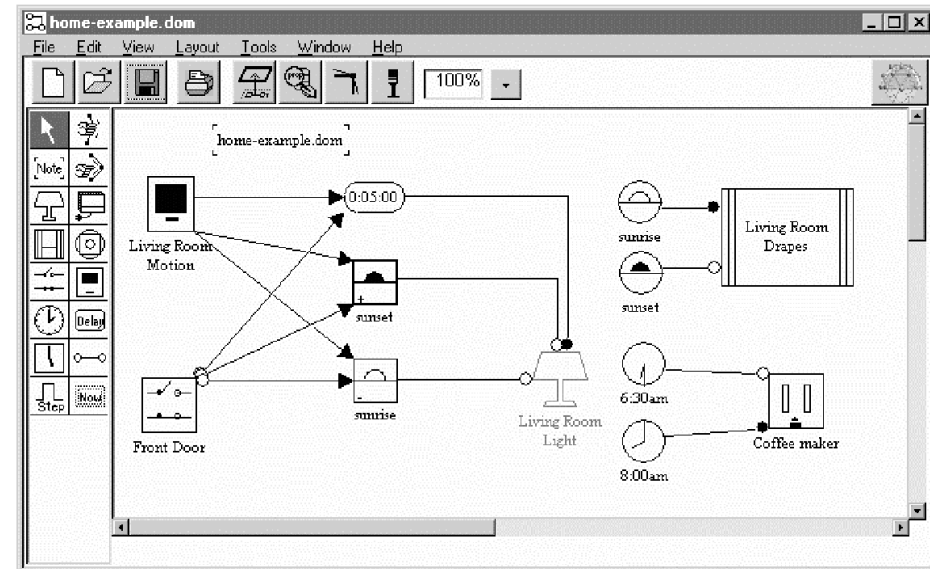
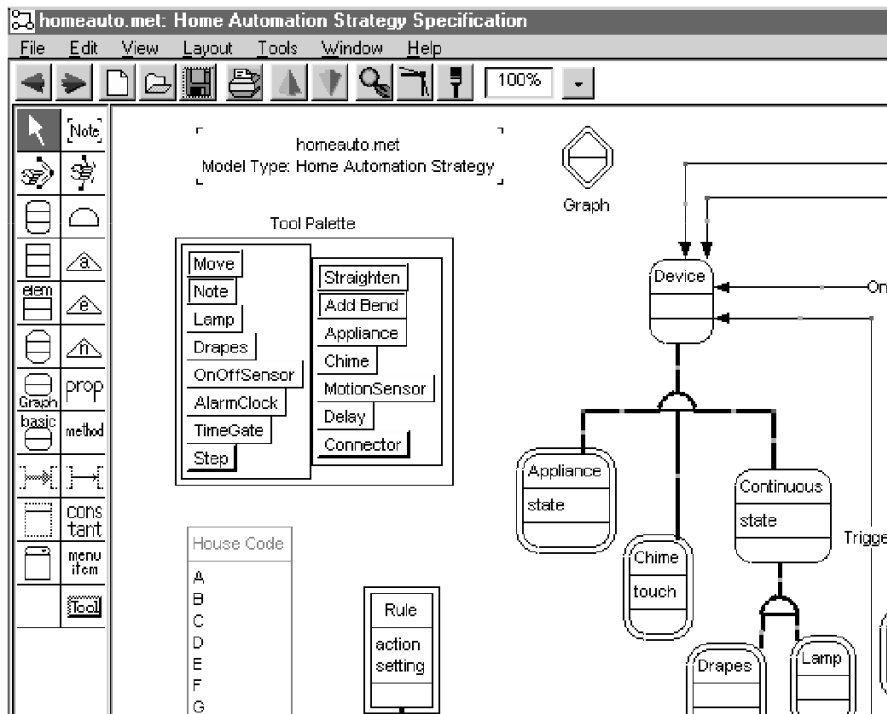
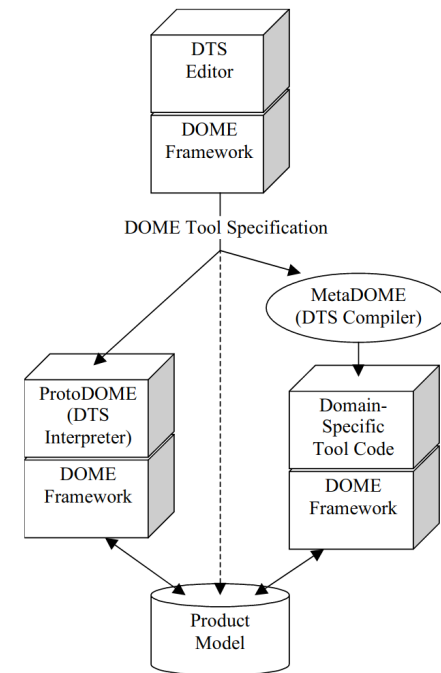
Modify

RAMification process



Modify

1990s: Honeywell's DoME Domain Modeling Environment



$$\cdot \quad [\text{"A+B"}]_{\text{REG EXP}} = \{ \text{"AB"}, \text{"AAB"}, \dots \}$$

GME

GRAMMAR

ε

ε

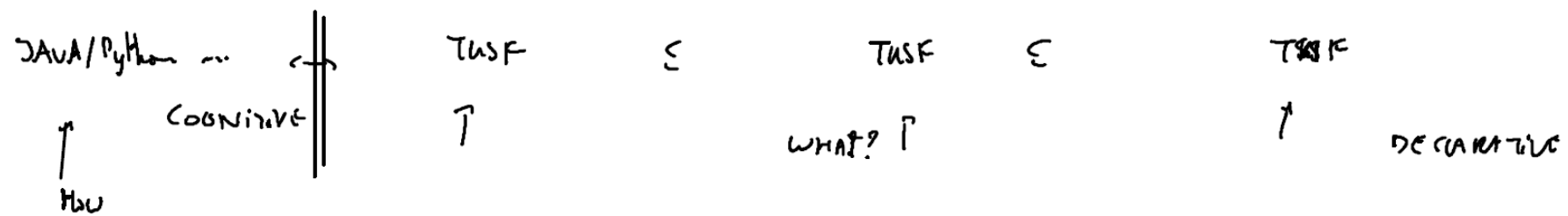
N M C D

GRAPH GRAMMAR

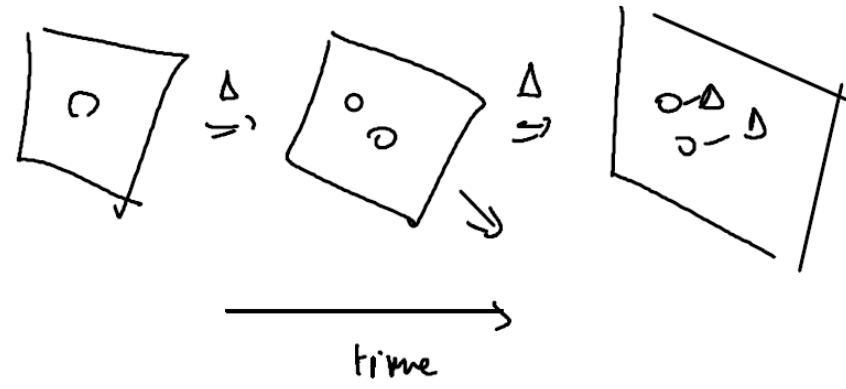
* TRANSFORMATION

Transforming Strings, Trees, or Graphs?

STRINGS / SEQUENCE \subseteq TREES \subseteq GRAPHS



DYNAMIC STRUCTURE



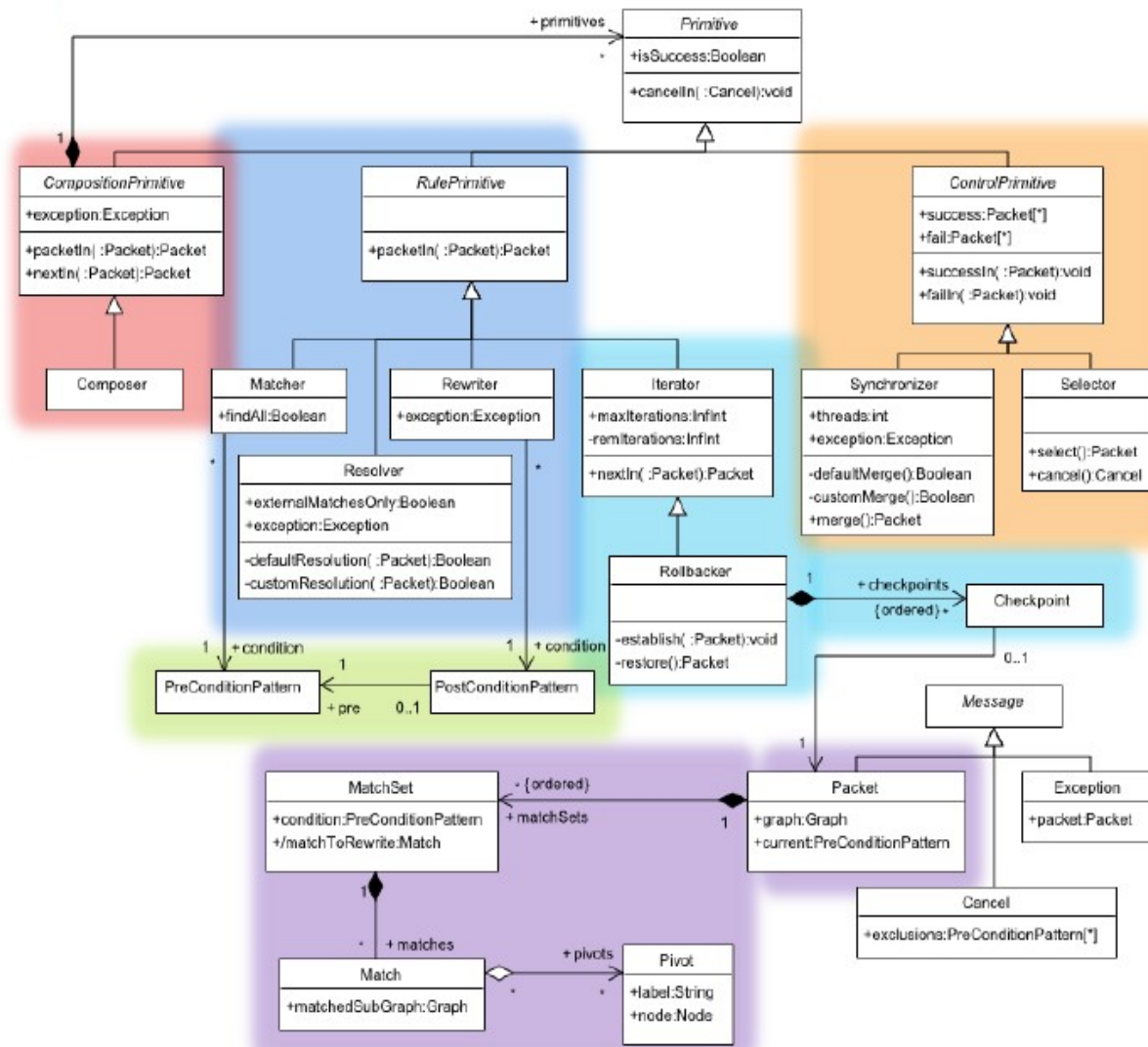
Why not always use GRAPHS / GRAPH TUSF. ?

- ① COMPLEXITY \rightarrow PERFORMANCE / EFFICIENCY
- ② COGNITIVE / UNDERSTANDING

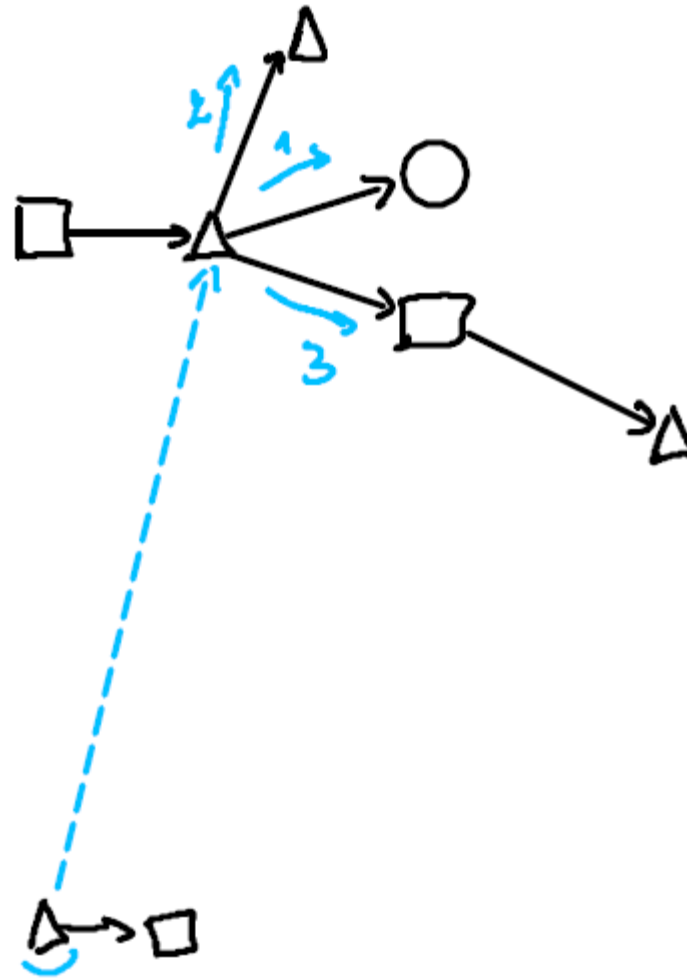
BEST APPROPRIATE - ABSTRACTION - FORMALISM

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

matching, pivot, scope



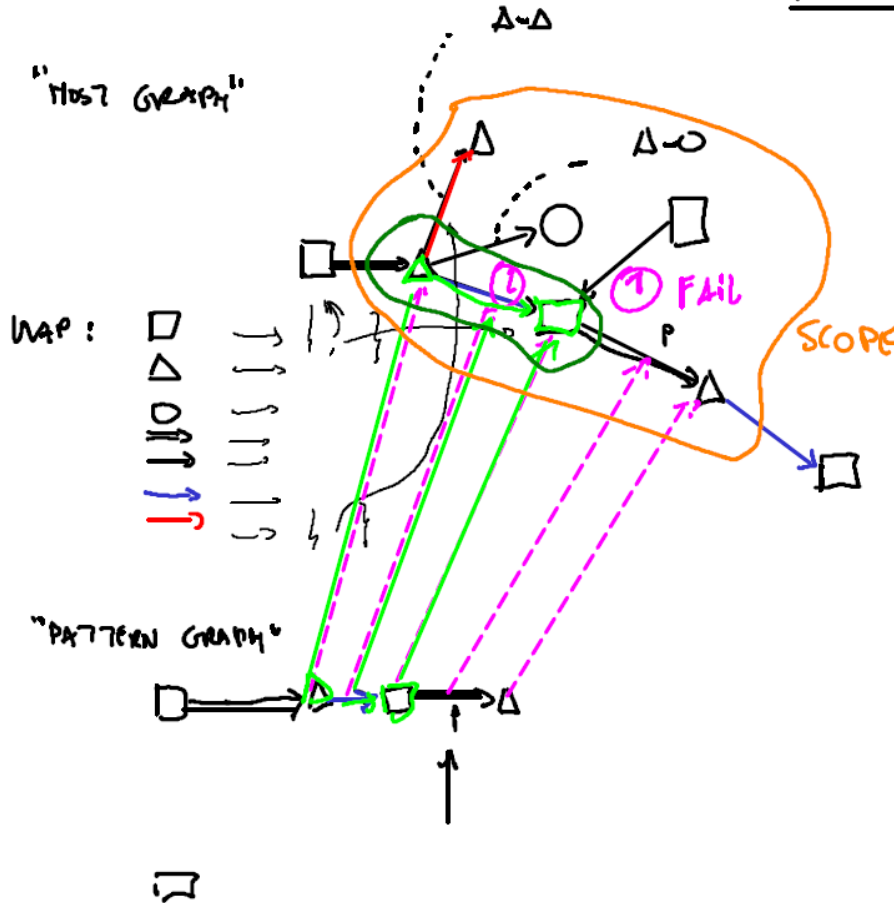
Matching Algorithms (1): Search Plans



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

Pivot Partition Match

Sub-Graph of Host Graph



WHERE TO START

FIRST MATCH

$$\Theta(\text{SIZE}(\text{PATTERN GRAPH}))$$

LINEAR!

$$\text{SIZE}(\text{HOST GRAPH}) > 10^6$$

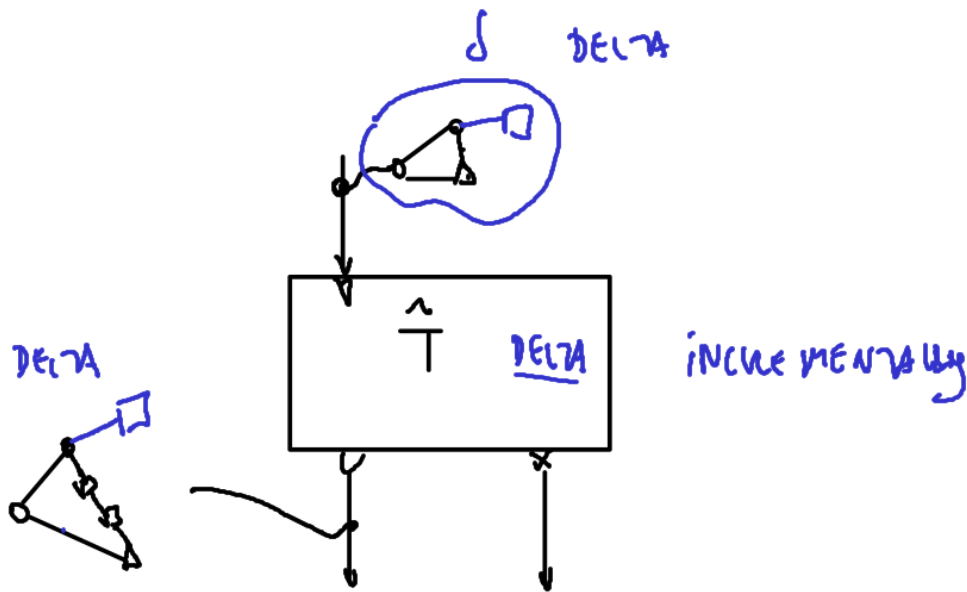
"COMPLEX PATTERN"
 → FAIL EARLY
 (FEWER MATCHES)

SCOPE

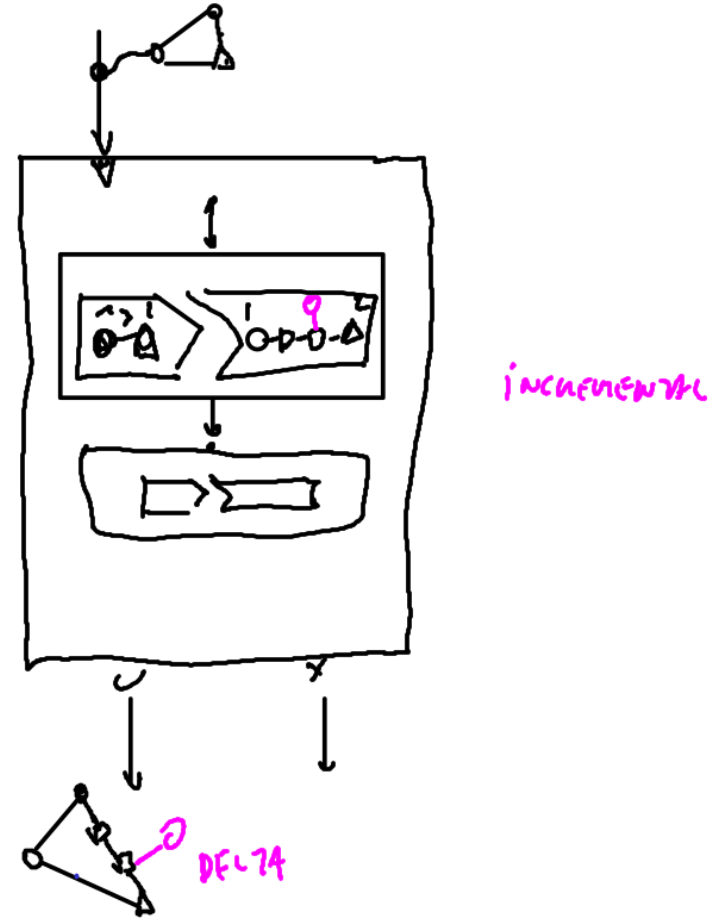
WHAT TO EXCLUDE FROM HOST GRAPH

~ SIZE OF (SCOPE GRAPH)

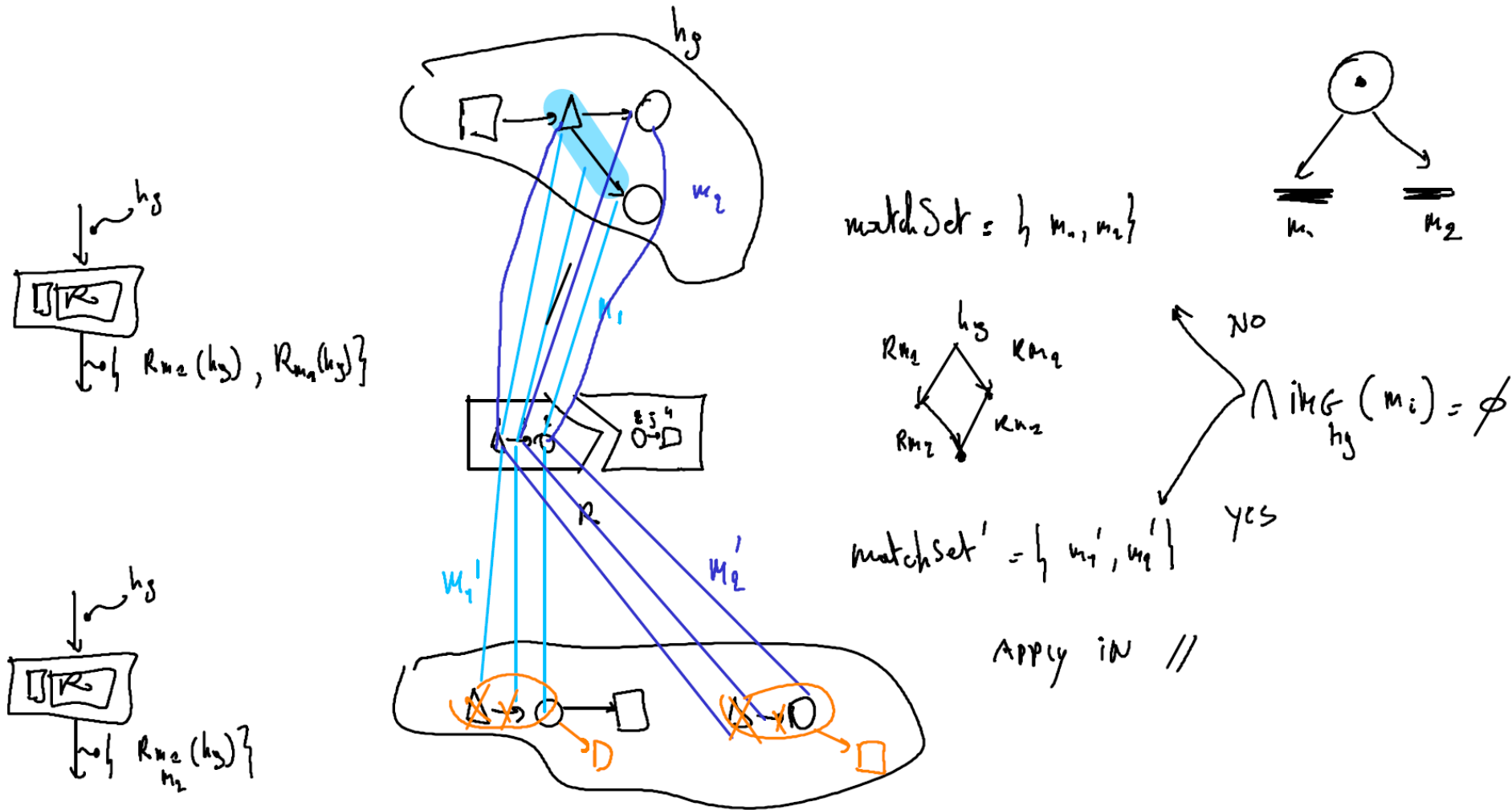
Matching Algorithms: improving performance of "incremental" model transformation: the Rete algorithm



RETE
PhoLoG



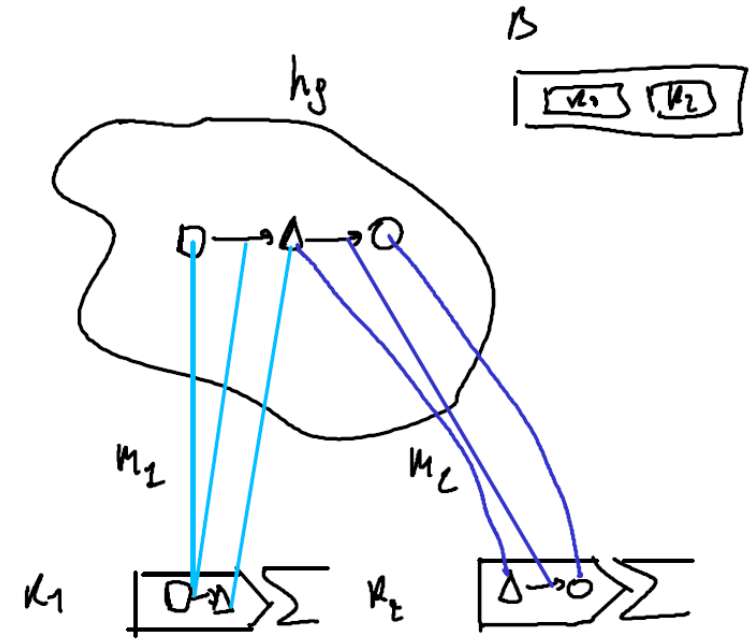
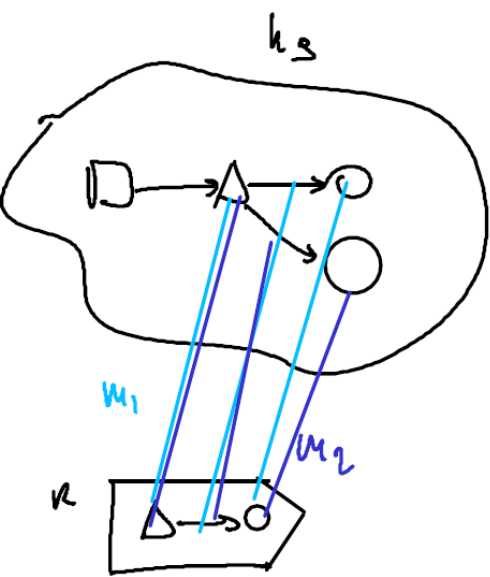
Choice → parallel independence, critical pairs



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

single rule, multiple matches

multiple rules, multiple matches



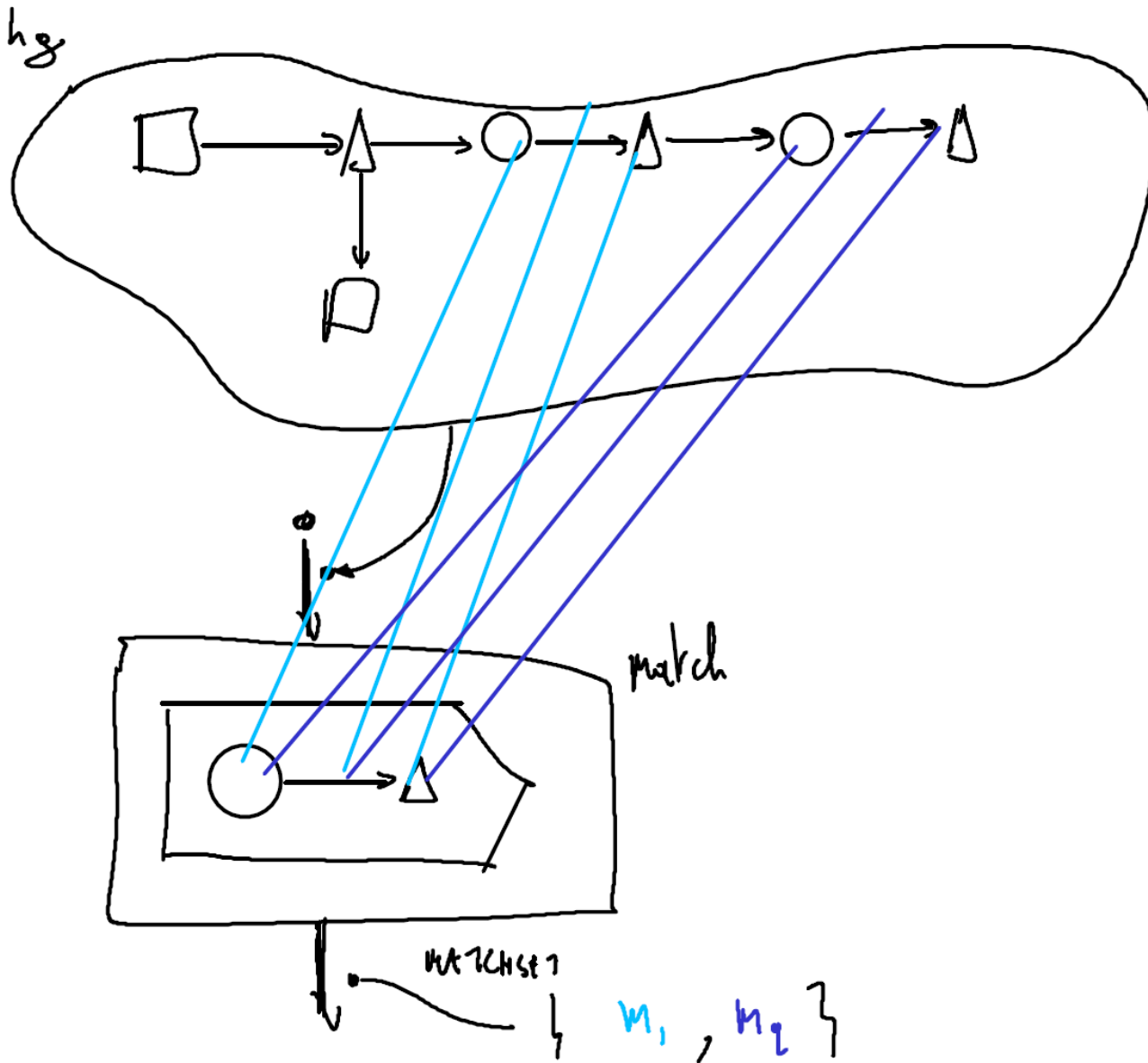
RANDOM CHOICE (REPEATABLE)
FROM ALL MATCHES

SELECT ($\begin{matrix} \phi & 1 \\ \{m_1, m_2\} \end{matrix}$)

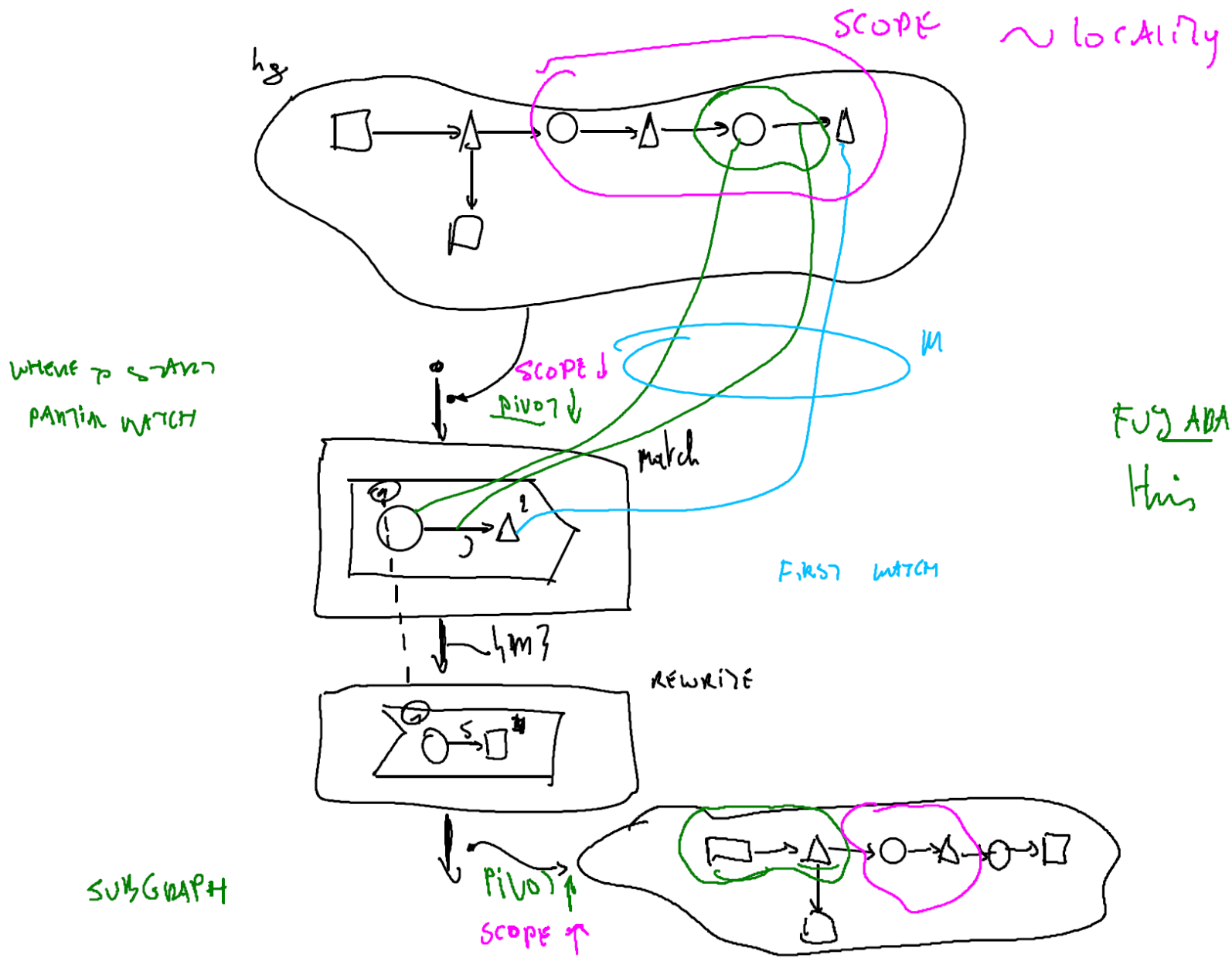
$U_I([0, 1])$

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

```

1 module CD2RDB;
2 create DB: RDBMS from CD: CD;
3
4 rule Class2Table {
5   from
6     c : CD!Class
7   to
8     t : DB!Table (
9       name <- c.name
10      cols <- c.attrs
11      pkey <- pcol )
12     pcol : DB!Column (
13       name <- 'id'
14       type <- 'int32' )
15 }
16
17 rule Attr2Col {
18   from
19     a : CD!Attribute
20   to
21     t : DB!Column (
22       name <- a.name
23       type <- a.convertedType() )
24 }
25
26 helper context CD!Attribute def: convertedType(): String =
27   if a.type.name = 'String' then 'string'
28   else if a.type.name = 'Int' then 'int32'
29   else a.type.name
30   endif
31   endif;

```

LHS: 1 element type

RHS: elements
to create in
new model

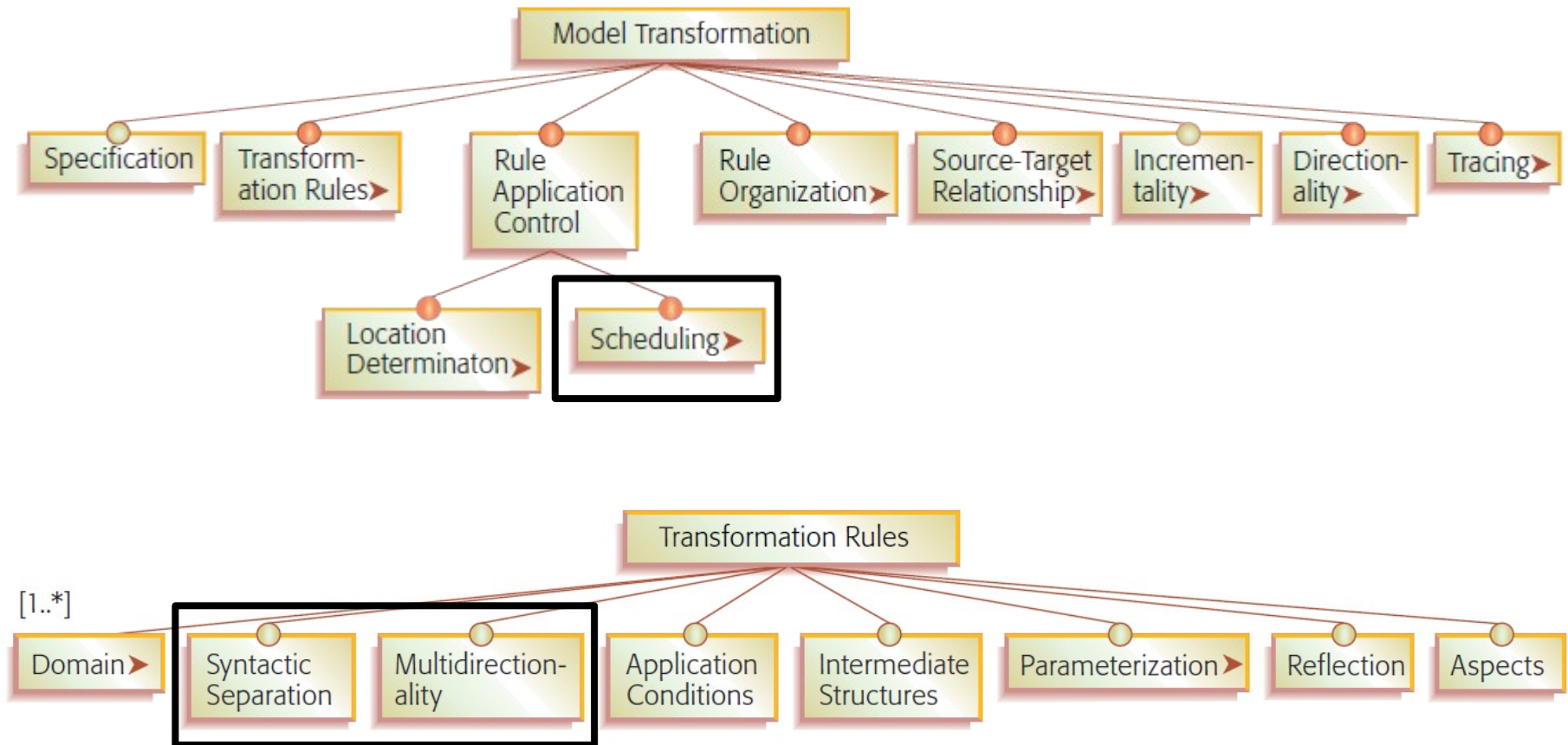
Call implicitly
another rule

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

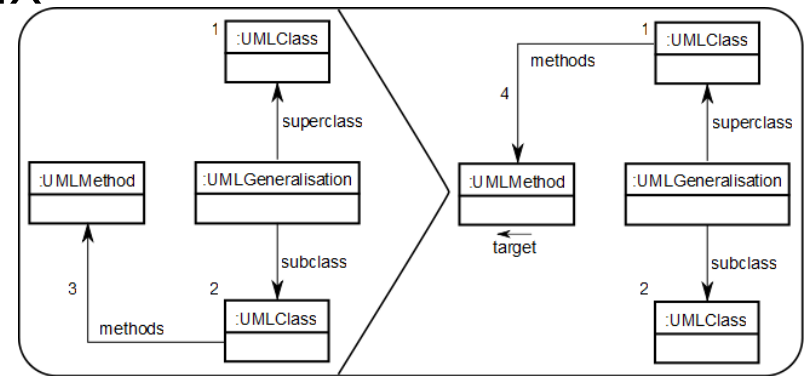
Feature-Based Survey of Model Transformation Approaches



Rule patterns

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

MoTif rule



ATL rule

```

module Person2Contact;
create OUT: MMb from IN: MMA {

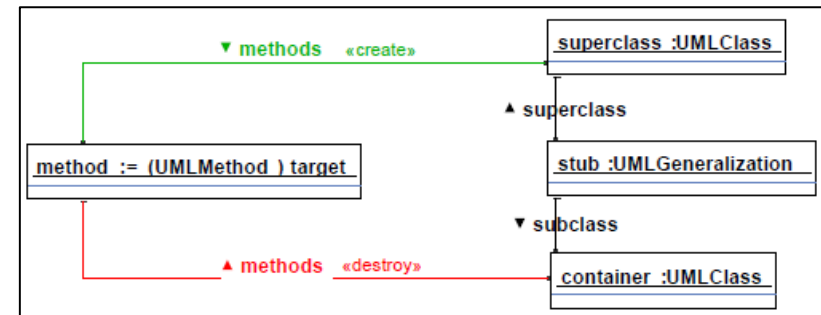
```

```

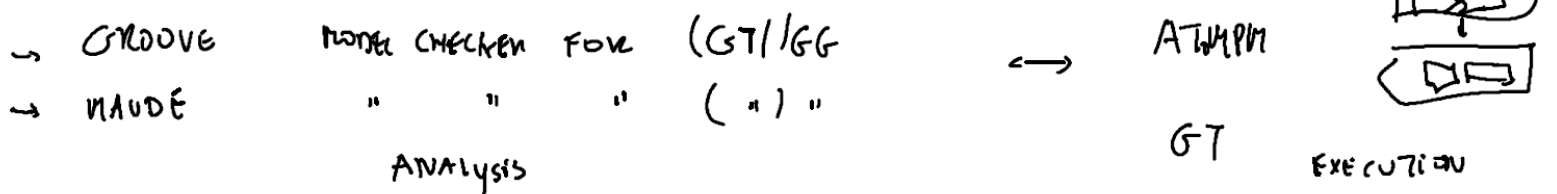
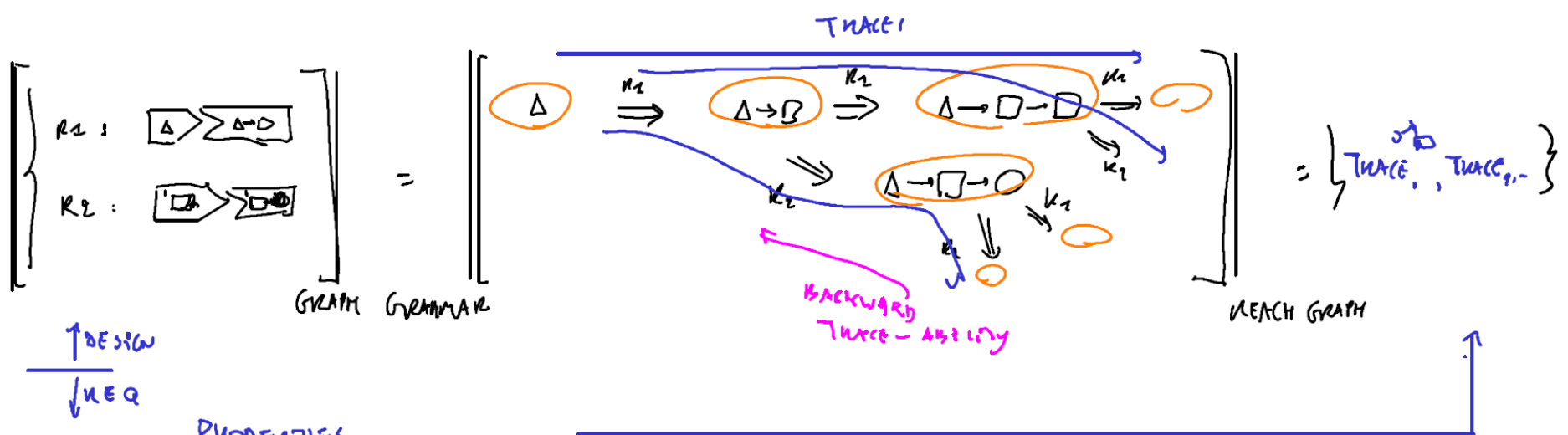
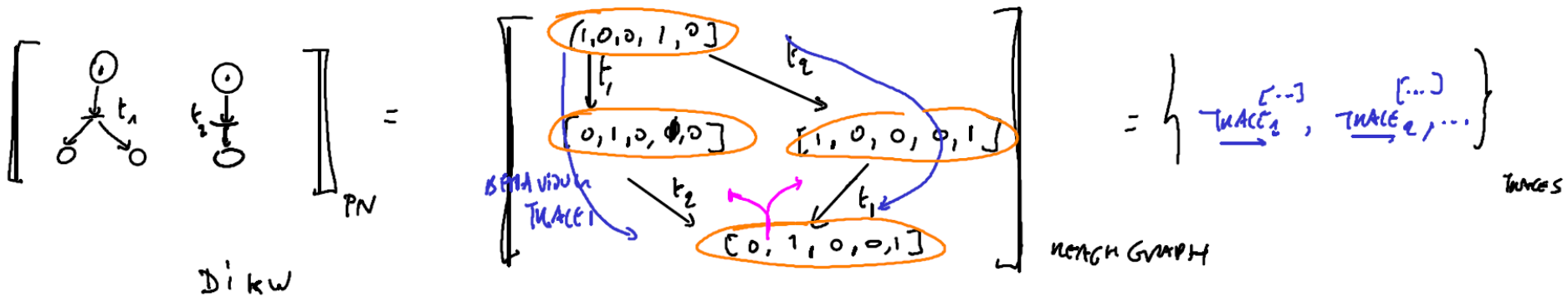
rule Start {
  form p: MMA!Person(
    p.function = 'Boss'
  )
  to c: MMb!Contact(
    name <- p.first_name + p.last_name)
}

```

FUJABA/Henshin compact notation



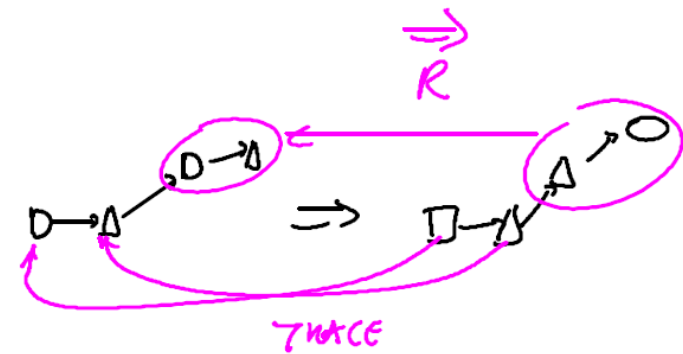
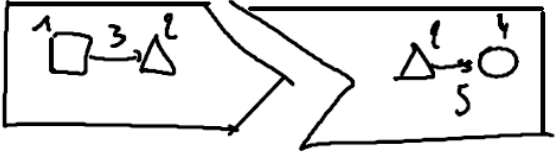
Choice → explore all possibilities → analysis over all traces



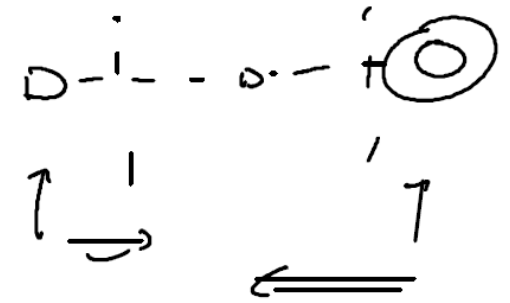
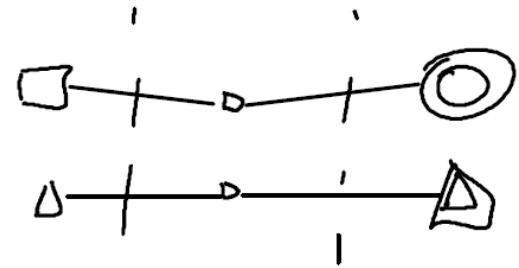
Trace of Transformation Execution vs. Bi-Directional Transformations



Rule R

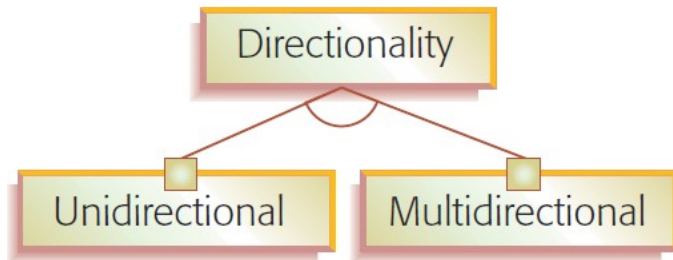


RELATION

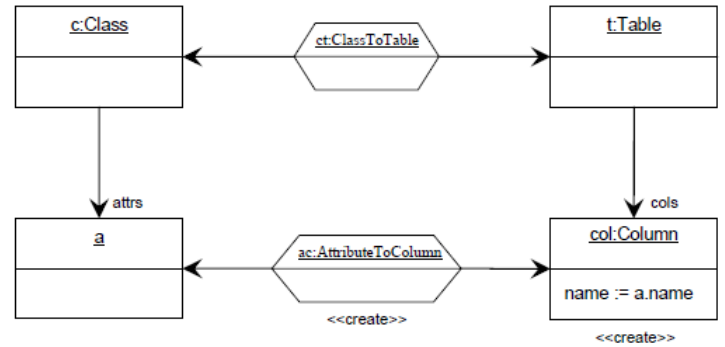


Bi-DIRECTIONAL TRSF.

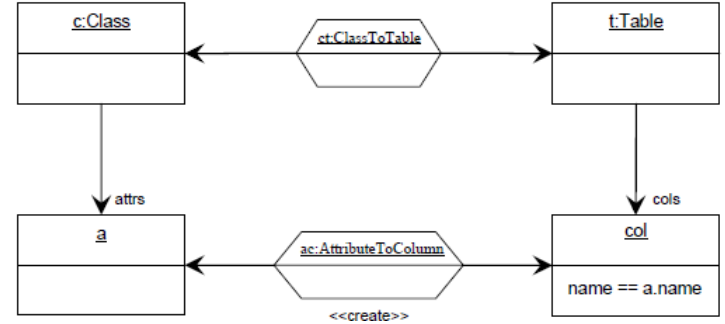
Multi-directional rules



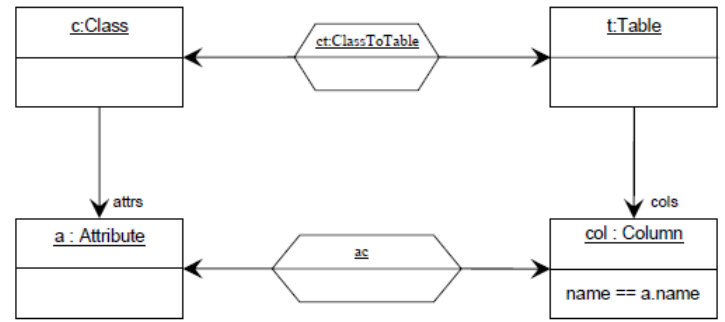
performForwardTransformation(a : Attribute)



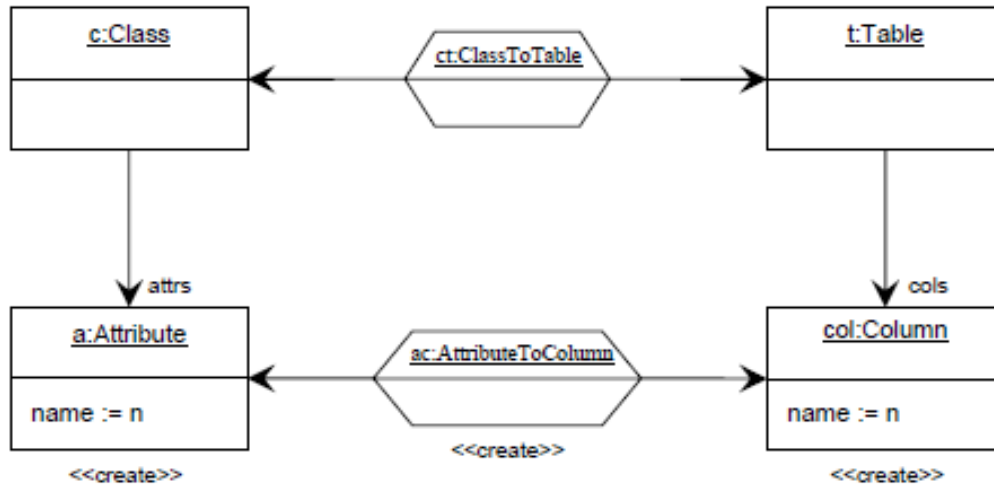
performLinkCreation(a : Attribute, col : Column)



performConsistencyCheck(ac : AttributeToColumn)



TGG rule



TGG operational rules

Rule Scheduling (aka Control)

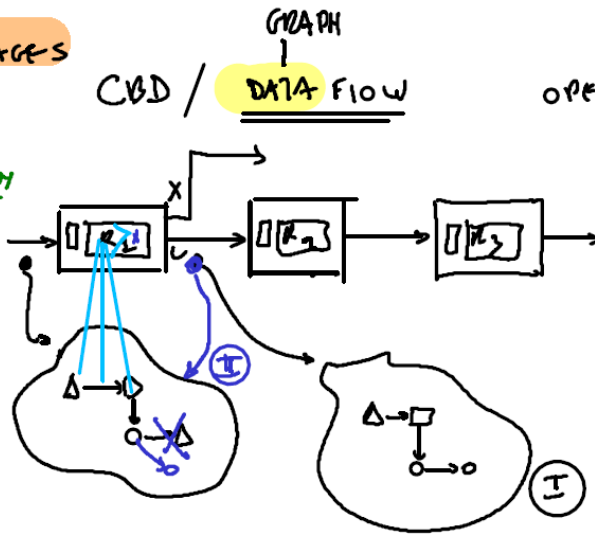
SCHEDULING LANGUAGES

⊕

RULE LANGUAGE

STRING / TREE / GRAPH

1.



OPERATIONS : GRAPH TRSF. SPECIFIED BY RULE

↓ Hierarchy

⊕

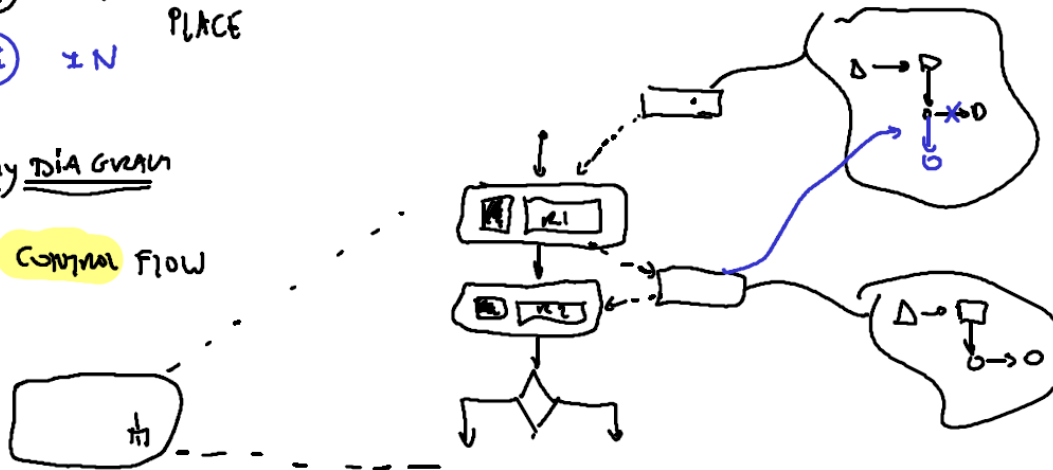
GREAT (GME)

⊖ OUT PLACE
⊕ IN

2.

ACTIVITY DIA GRAM

CONTROL FLOW



FUJABA

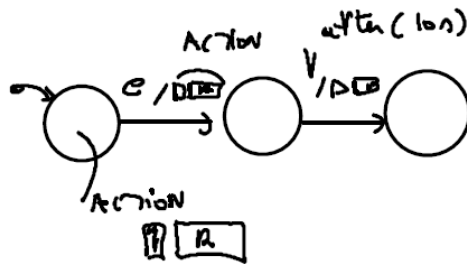
"this"

PIVOT

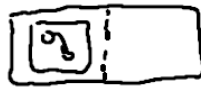
FIRST MATCH

Rule Scheduling (aka Control)

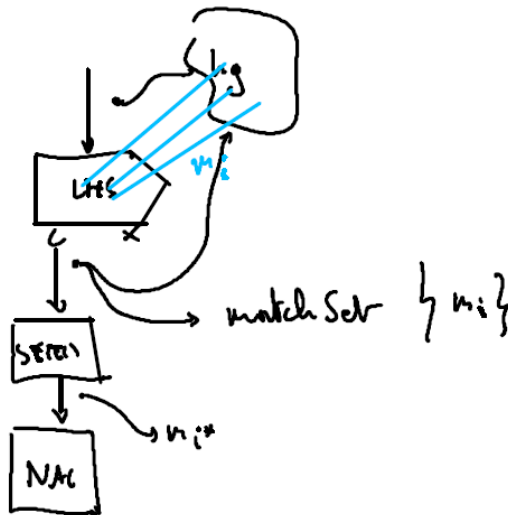
3. TIMEO STATE AUTOMATA
STATE CHARTS



TIMEO HT
CONCURRENCE



4. DEVS
- TIME
- HIERARCHY
- DEV
- CONCURRENCE



Notif



Rule Scheduling (aka Control)

5. PROGRAMMING LANGUAGE

hg = GRAPH()

LHS rule = LHSRULE()

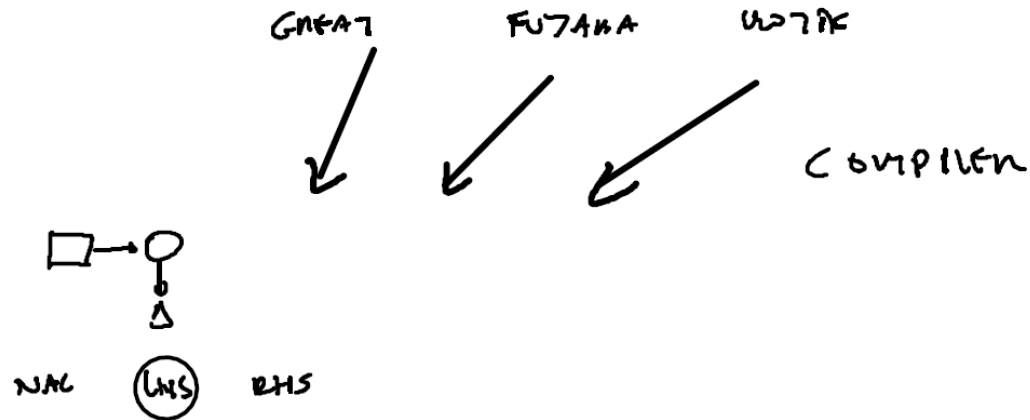
matcher = MATCHER()

matchSet = matcher.match(LHSrule, hg)

match = select(matchSet)

...

REWRITER()



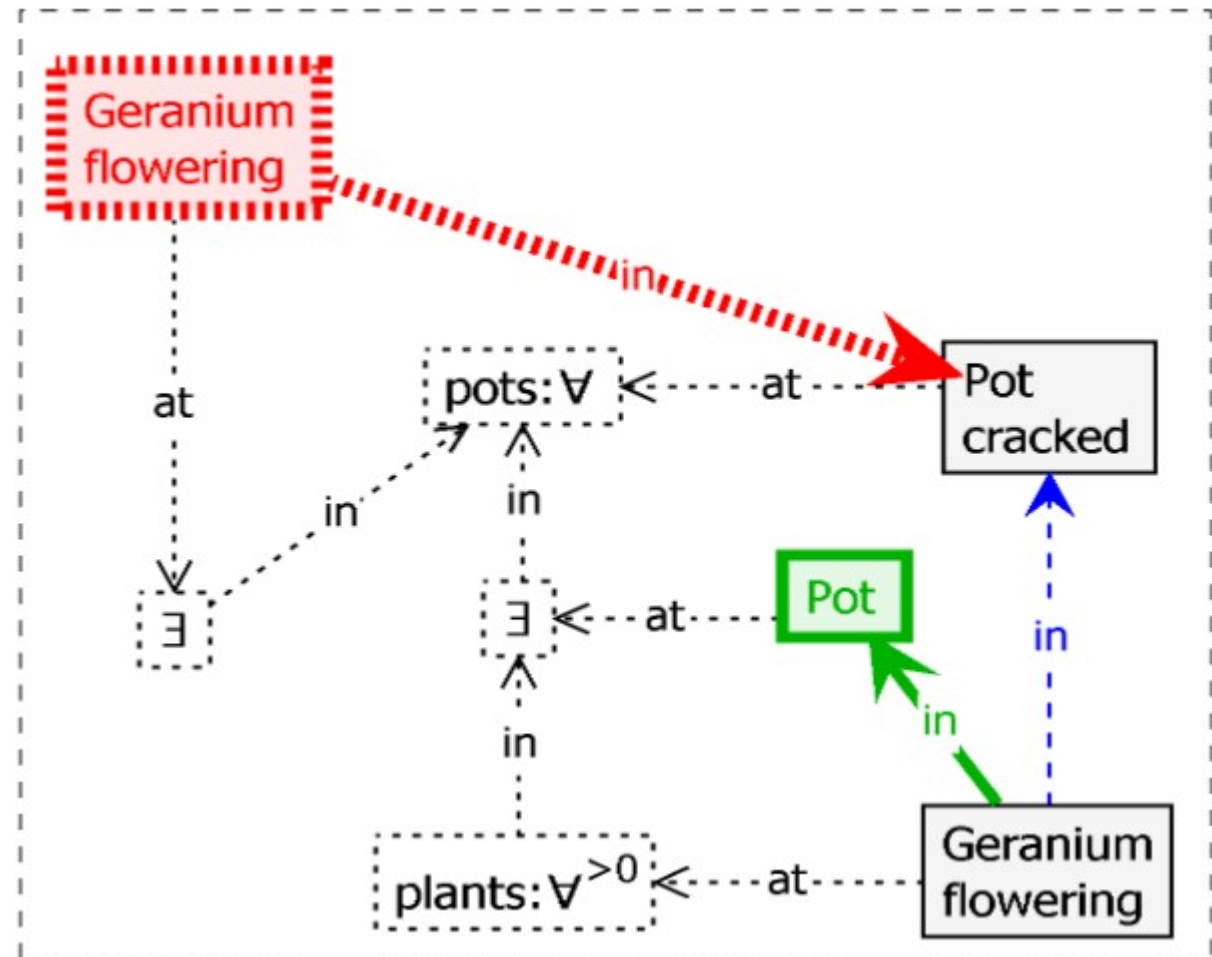
T-core

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.uu.tu-berlin.de/eceasst/article/view/260>

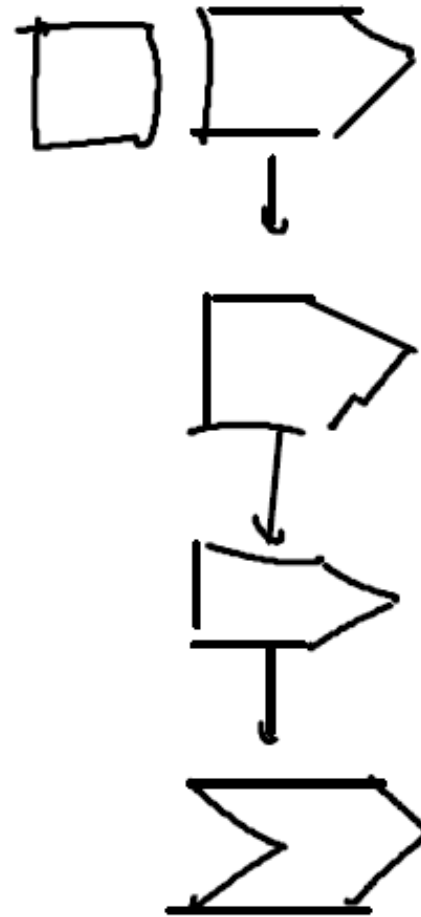
We have a number of flower pots, each of which contains a number of geranium plants. These tend to fill all available space with their roots, and so some of the pots have cracked. For each of the cracked pots that contains a geranium that is currently in flower, we want to create a new one, and moreover, 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.



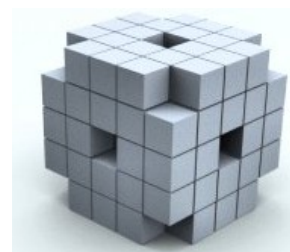
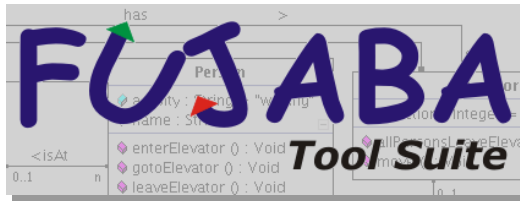
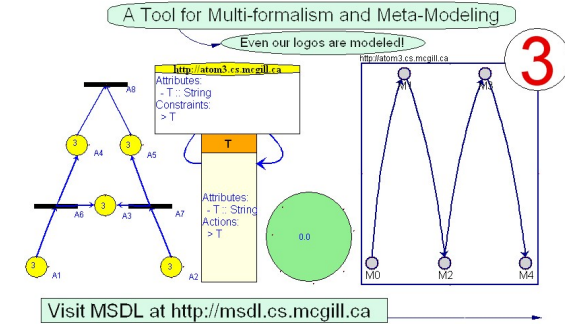
Increased Expressiveness: rule amalgamation

Operationally (in terms of T-Core building blocks):

Match - Match - ... - Re-Write



Plethora of model transformation languages



GReAT

DSLTrans

ProGReS

MOLA

VMTS

