5.1. Introduction to Graph Transformation

Juan de Lara, Elena Gómez, Esther Guerra
{Juan.deLara, MariaElena.Gomez, Esther.Guerra}@uam.es
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Index

- **Introduction.** 
  - **Motivation.**
  - **Types of Transformation.**
  - **Types of Languages.**
- Overview of Graph Transformation.
- Exercises.
- Control Languages.
- Meta-modelling and Graph Transformation.
- Applications.
- Tools.
- Conclusions.

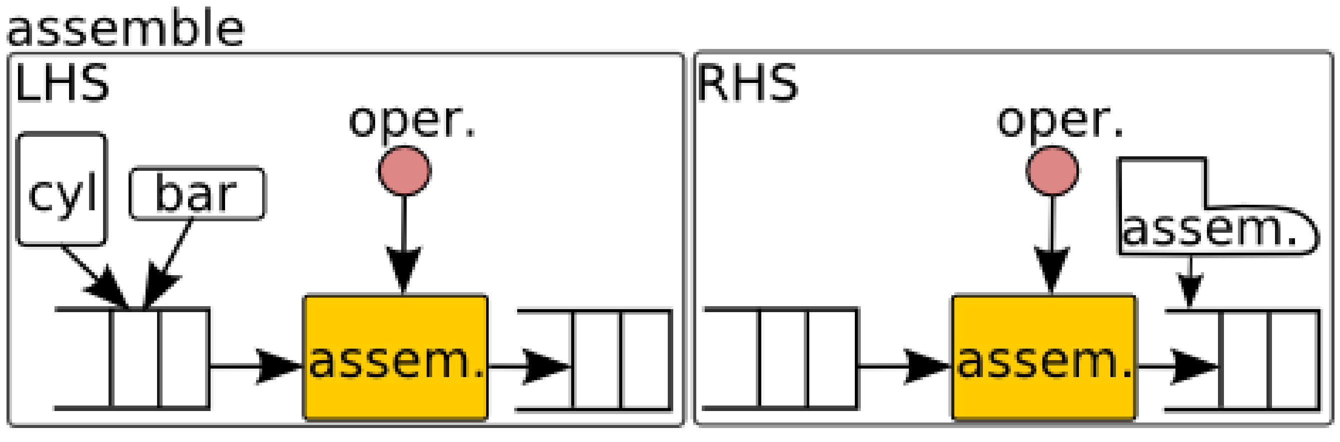
Motivation

- In Model-Driven Engineering (MDE), model transformation is a fundamental activity:
  - Transformation into other languages (e.g. for analysis).
  - In-place transformations:
    - simulations or animations.
    - optimisations and refactorings.
  - Code generation.
  - …

- Need of a suitable (domain-specific) language for model transformation.
  - Intuitive.
  - Analysable.

Goals

- In MDE, it is common to manipulate Domain-Specific (Visual) models.
- Why not using the (Visual) concrete syntax of the models in the transformation language?



- Similar to programming by demonstration.
- No need to learn a new language!

Goals

- How do we know if our transformation is correct?
  - Are there conflicting rules?
  - What are the rule dependencies?
  - Can they be executed in any order?
  - …?

- Language founded on mathematics that allows answering these questions.

Types of Transformations

Triple Graph Grammars

Level of Abstraction

Domain	Horizontal	Vertical
Inter-Formalism (exogenous)	Analysis, PIM-PIM, Migration	Simulation, PIM-PSM
Intra-Formalism (endogenous)	Optimisation, Re-designs, Re-factoring, Animation	Abstraction, Refinement

Graph Transformation

Index

- Introduction.
- **Overview of Graph Transformation.**
  - **Rule.**
  - **Match and Derivations.**
  - **Application Conditions.**
  - **Grammar.**
  - **Attributes.**
  - **Dangling Edges and non-injectivity.**
- Exercises.
- Control Languages.
- Meta-modelling and Graph Transformation.
- Applications.
- Tools.
- Conclusions.

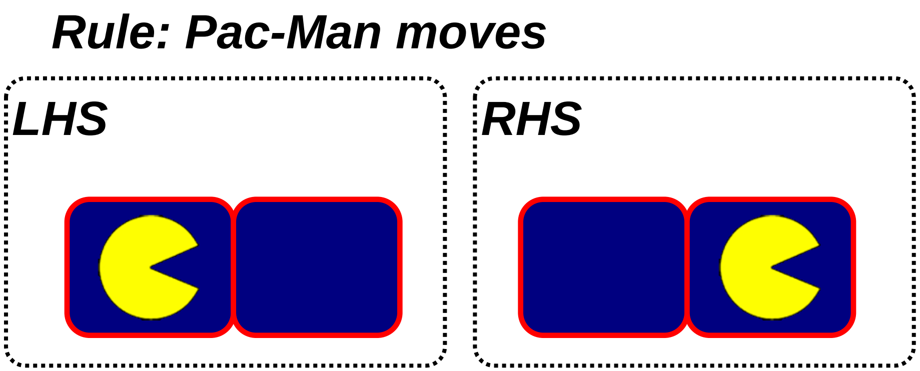
What is Graph Transformation?

- Graph Transformation (GT) is a declarative, graphical, formal way of manipulating graphs.
  - A graph has nodes and edges….
  - … nodes and edges can have attributes…
  - … we can also add types to nodes and edges…
  - … so we can encode models as graphs.
- Formal technique, many formalisations, the most popular ones based on category theory.
- Hartmut Ehrig and others at TU Berlín in the 70s.

What is GT useful for?

- GT has been used for:
  - Generation of (graph) languages.
    - An alternative to meta-models.
  - Language recognition (parsing).
  - Definition of the semantics of Domain Specific Visual Languages.
    - Specification of simulators or animators.
  - Definition of refactorings, redesigns.
  - Model Transformations.
  - Mutation operators for search-based software engineering (e.g., genetic programming)
  - Computations on graphs.
    - Google's Pregel, Apache Giraph (used at Facebook)

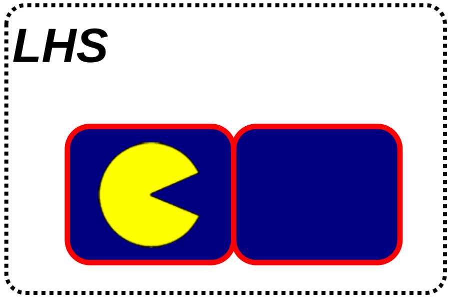
Rules



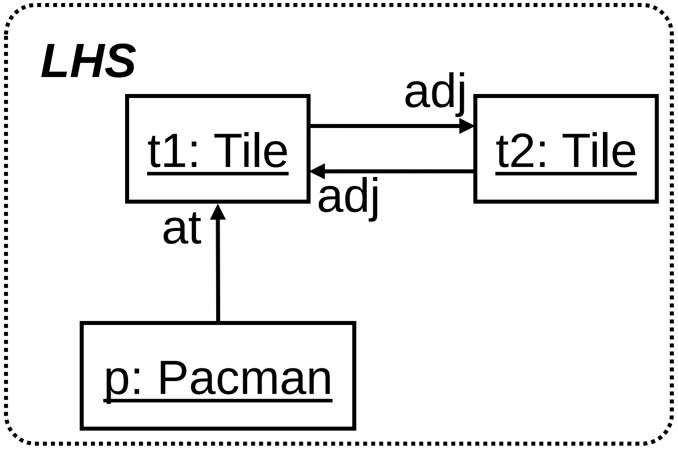
- A rule is made of a *Left Hand Side* (LHS) and a *Right Hand Side* (RHS).
- Both are graphs.
- They describe an action declaratively through pre- and post-conditions.

Rules

- Are they really graphs?



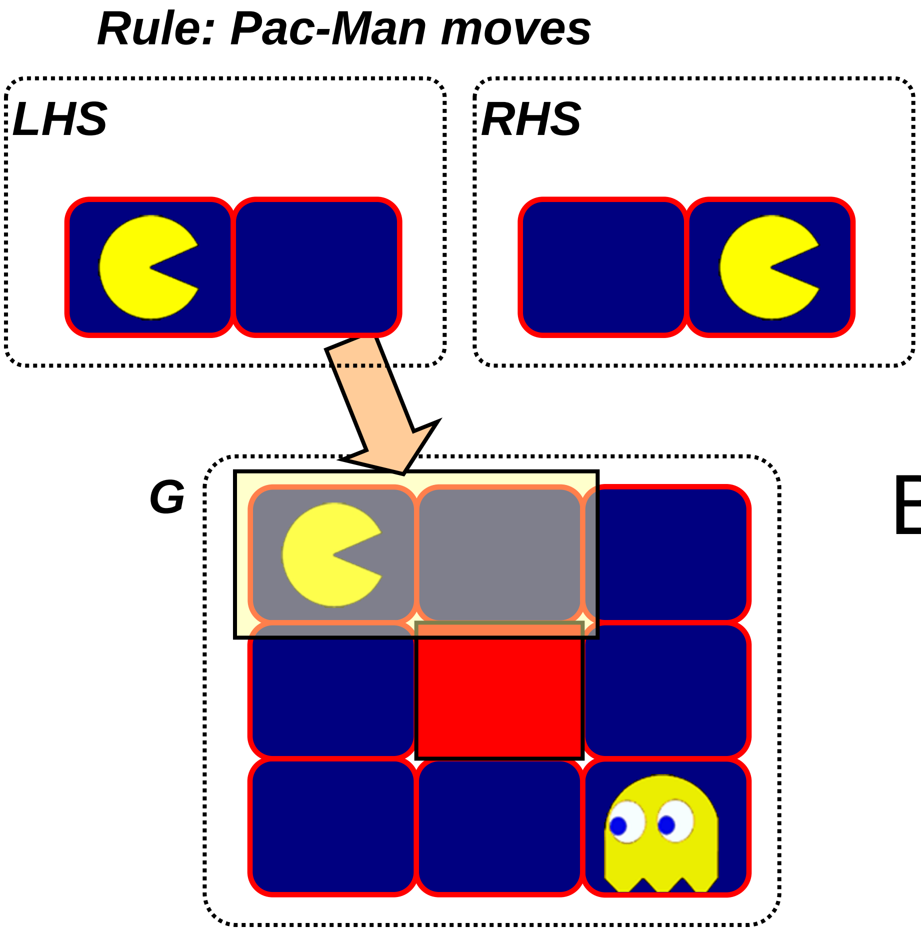
LHS in concrete syntax



LHS in abstract syntax

- Yes, we can use the *concrete syntax* of the language we are manipulating.
- We can also use the *abstract syntax*.

Direct Derivation

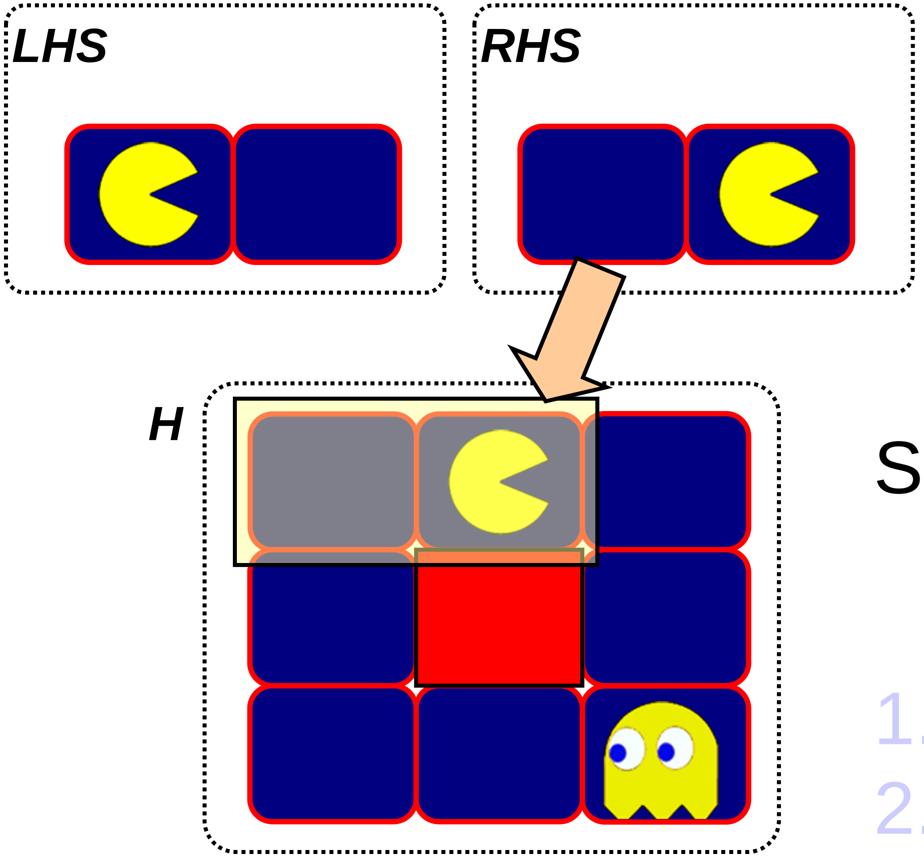


By pattern matching:

Step 1: Find an occurrence of the LHS in the *host graph* G.

Direct Derivation

Rule: Pac-Man moves



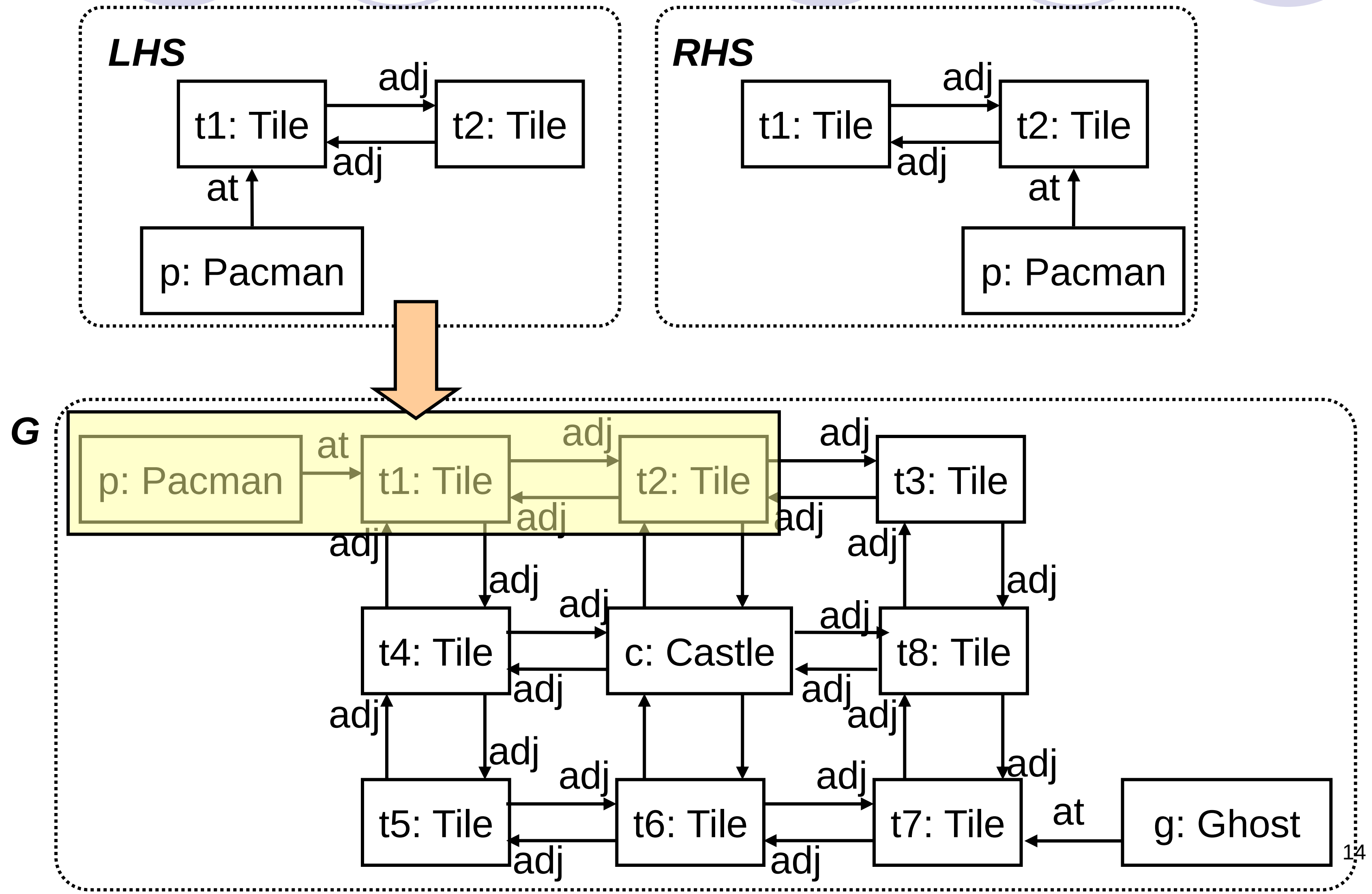
Step 2: Do the substitution, which means:

1. Delete LHS-(LHS∩RHS)
2. Create RHS-(LHS∩RHS)
3. Preserve LHS∩RHS.

We write $G \Rightarrow^r H$

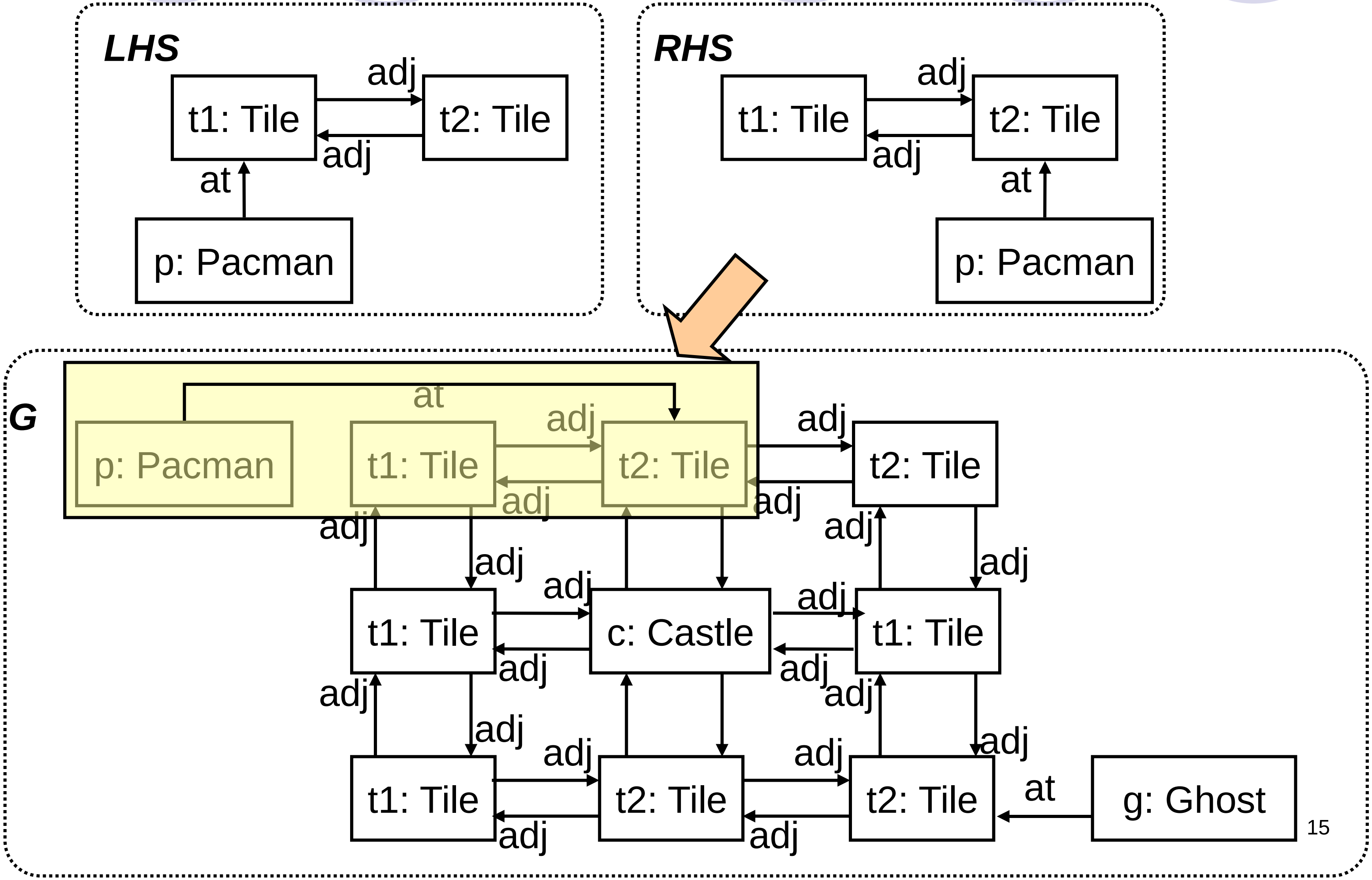
Derivation

in abstract syntax



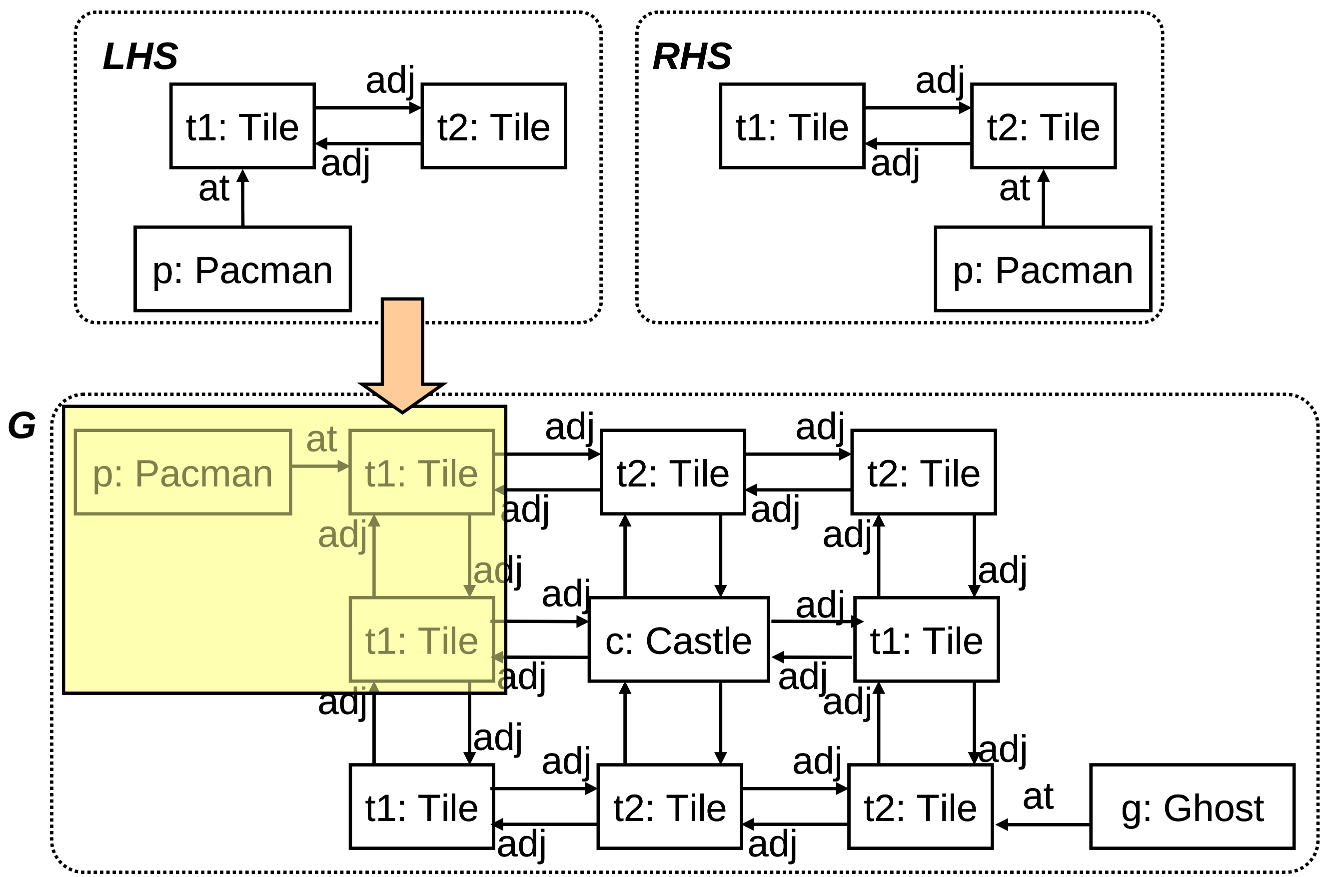
Derivation

in abstract syntax



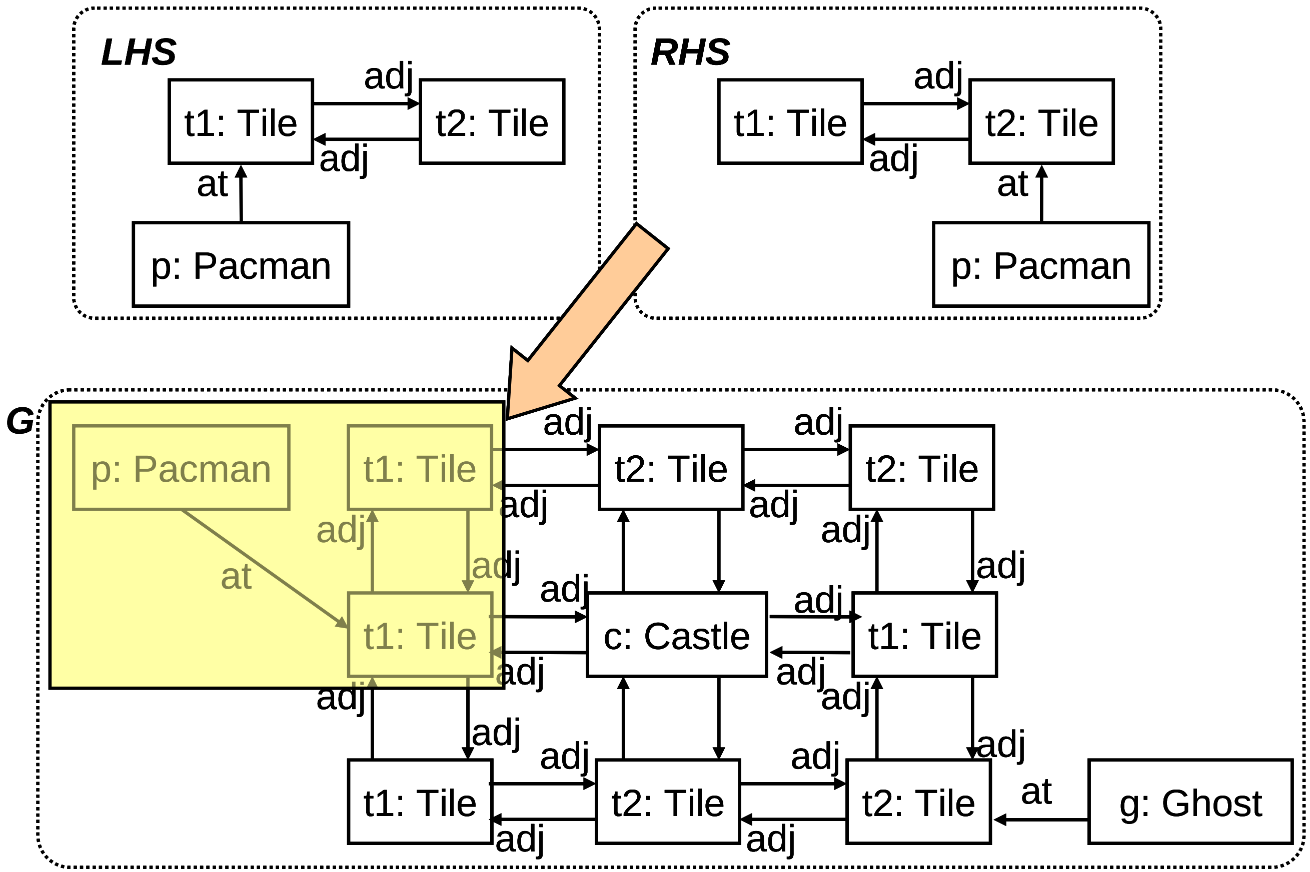
Match

- The place in the graph where the rule is applied.
- More than one valid match may exist for the rule, one has to be selected.



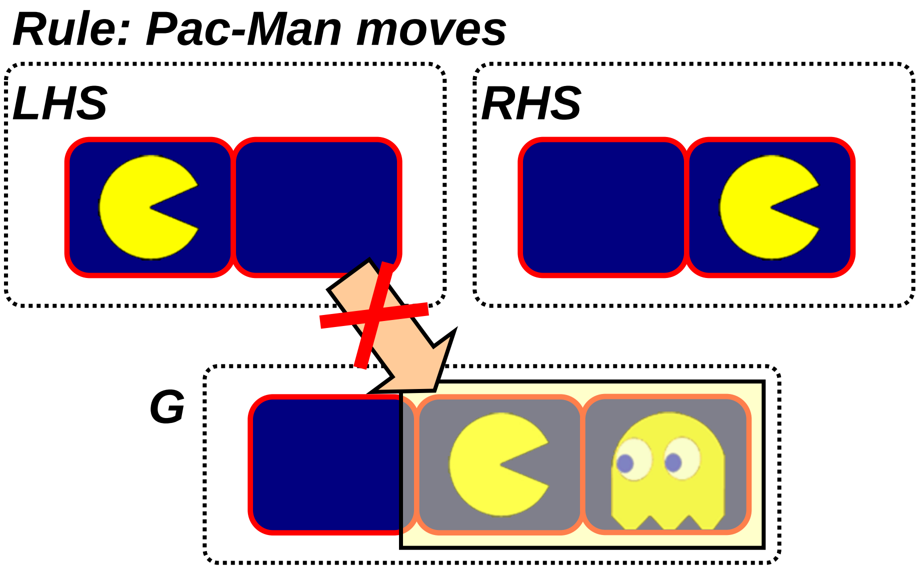
Match

- The place in the graph where the rule is applied.
- More than one valid match can exist for the rule, one has to be selected.



Application Conditions

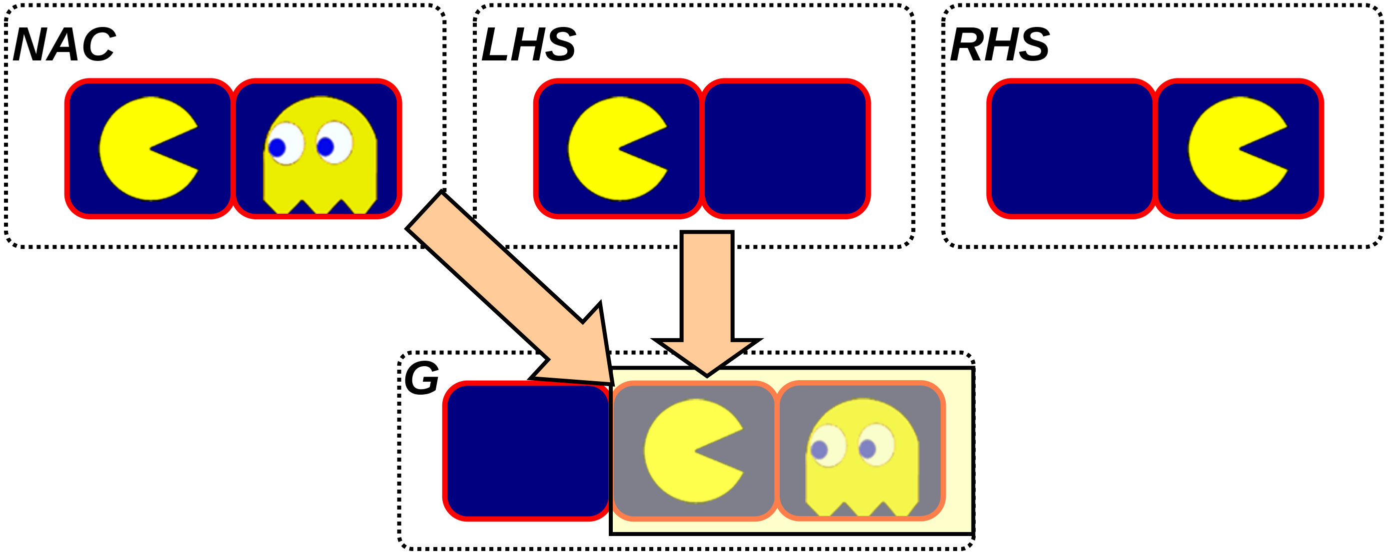
- The LHS of the rule expresses positive conditions for the rule to be applied.
- We can incorporate more complex application conditions into the rules, in particular to test negative conditions.
  - These are called NACs (Negative Application Conditions).



- Avoid certain matches. A smarter PAC-MAN!

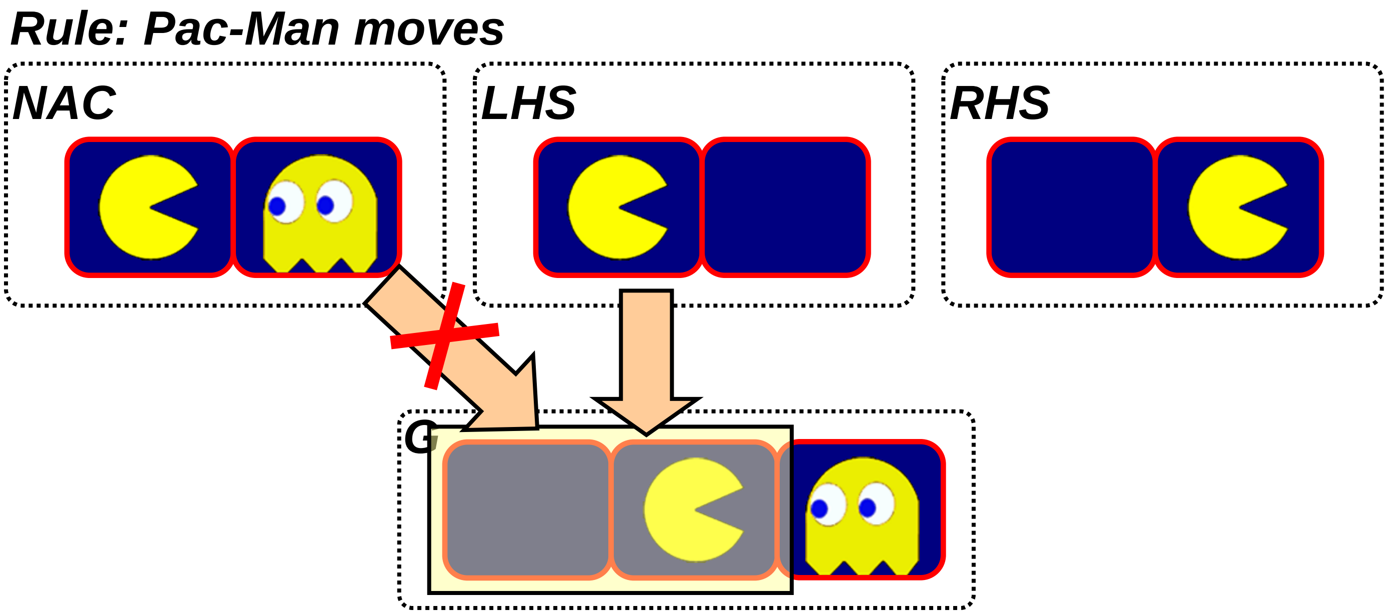
Application Conditions

Rule: Pac-Man moves



- This match is invalid, because there is an occurrence of the NAC.

Application Conditions

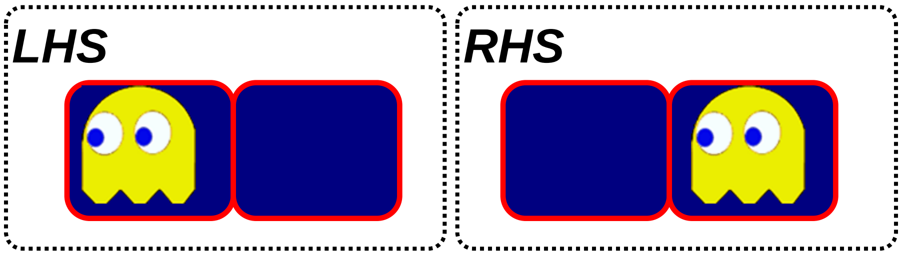


- This match is valid, because there is NO occurrence of the NAC.

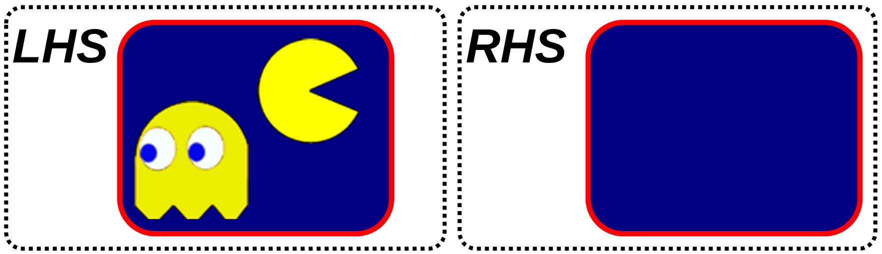
Grammars

- A grammar is a set of rules, and an initial host graph, to which the rules are applied.
- The rules are applied in arbitrary order, until none is applicable.

Rule: Ghost moves



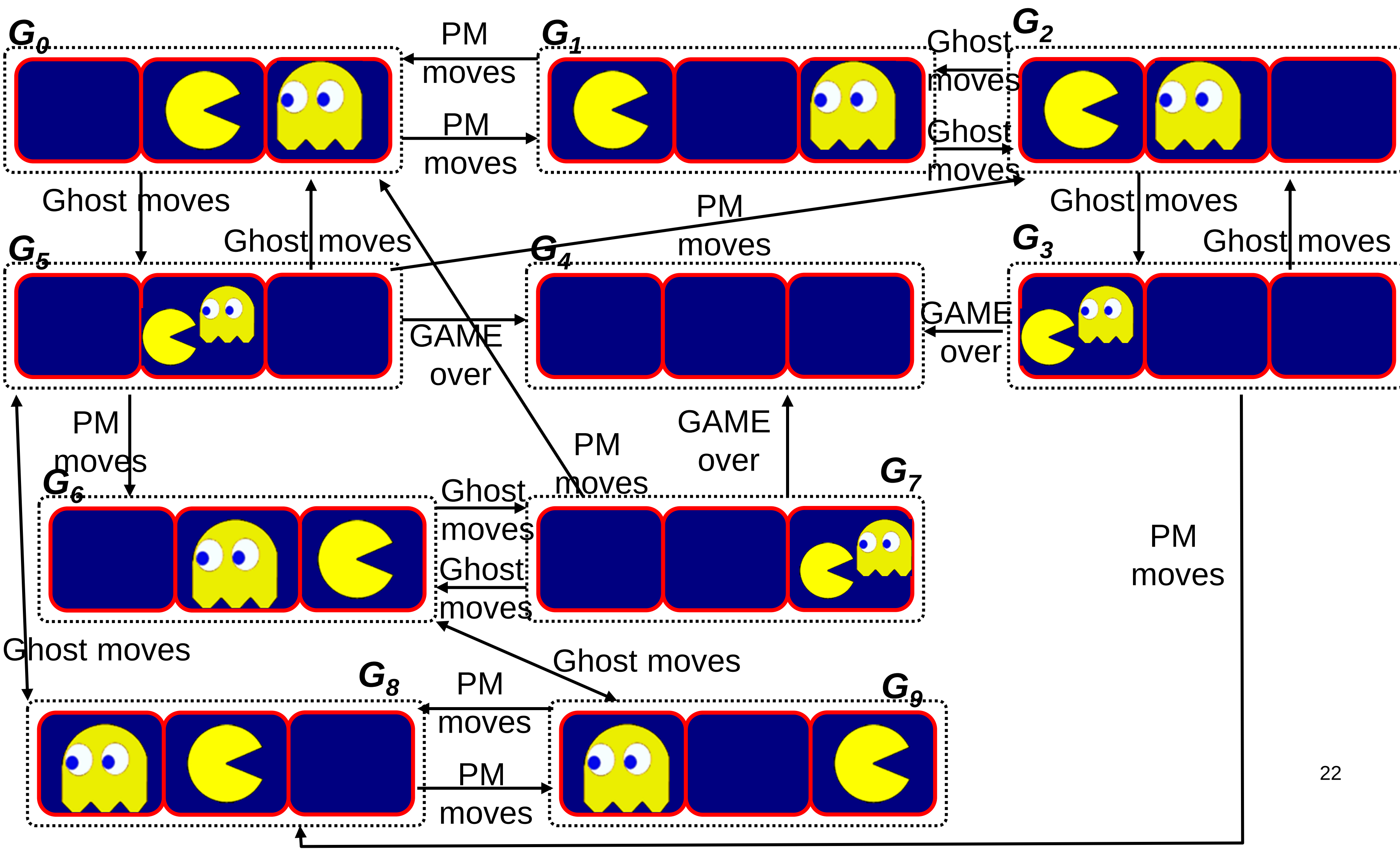
Rule: GAME Over



- Game={{PACMAN moves, Ghost moves, GAME Over}, G}.
- The semantics of the grammar are all reachable graphs.
  - SEM(Game)={ H | G $\Rightarrow^*$ H}
  - In this case the set is finite (if the graphs are finite), but there are derivations of infinite length.

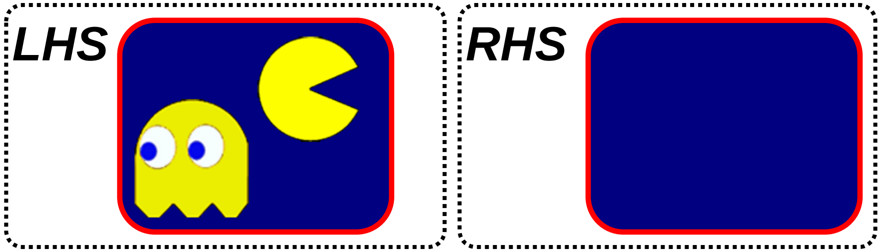
PAC-MAN Grammar

What is wrong? And how can we fix it?

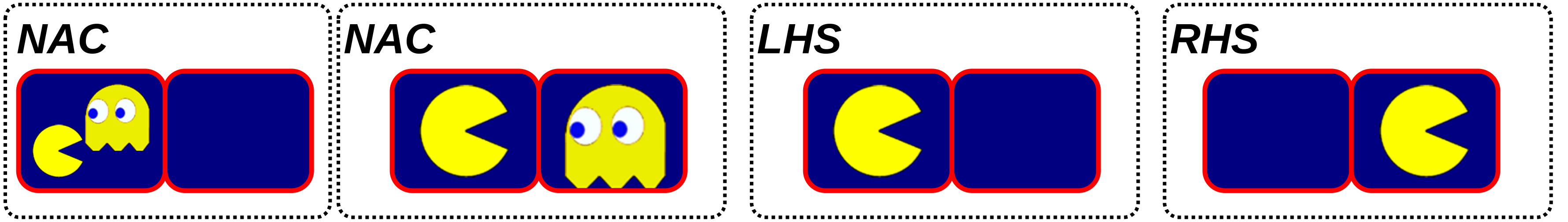


PAC-MAN Grammar

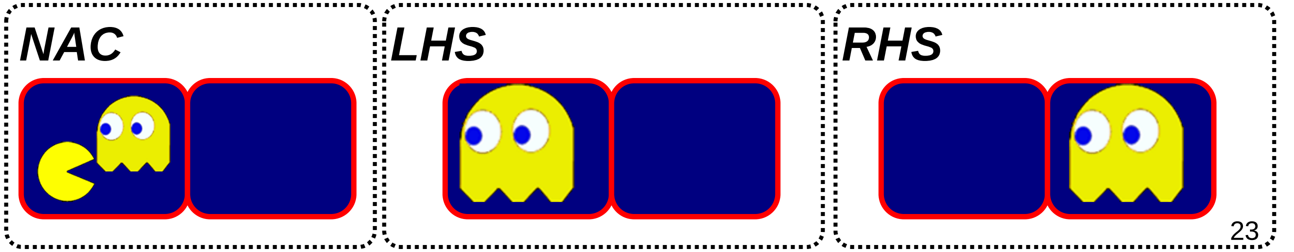
Rule: GAME Over



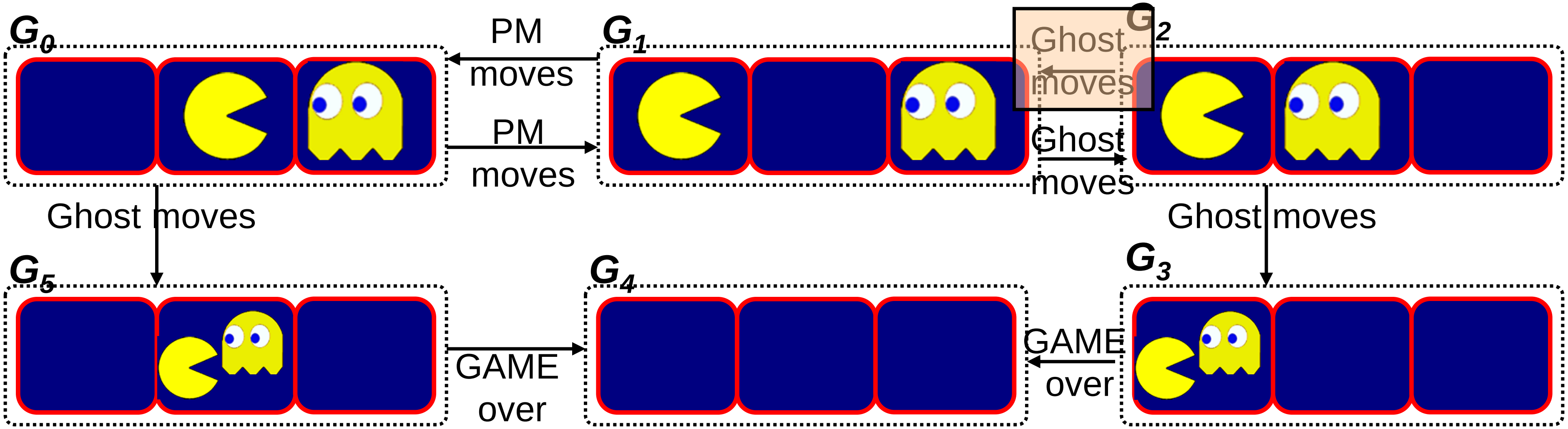
Rule: Pac-Man moves



Rule: Ghost moves



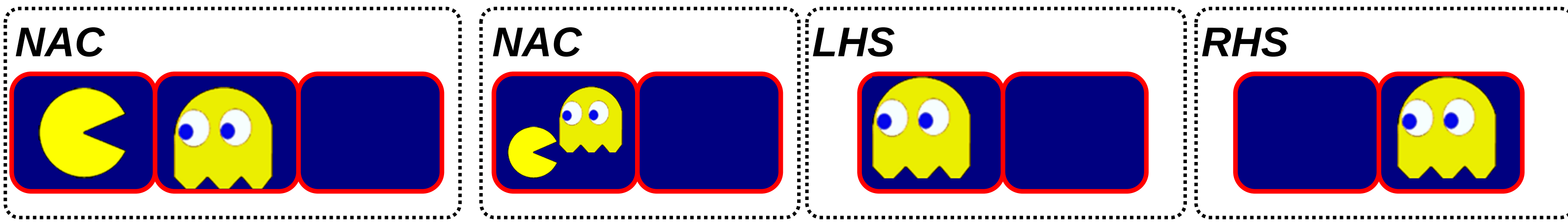
PAC-MAN Grammar

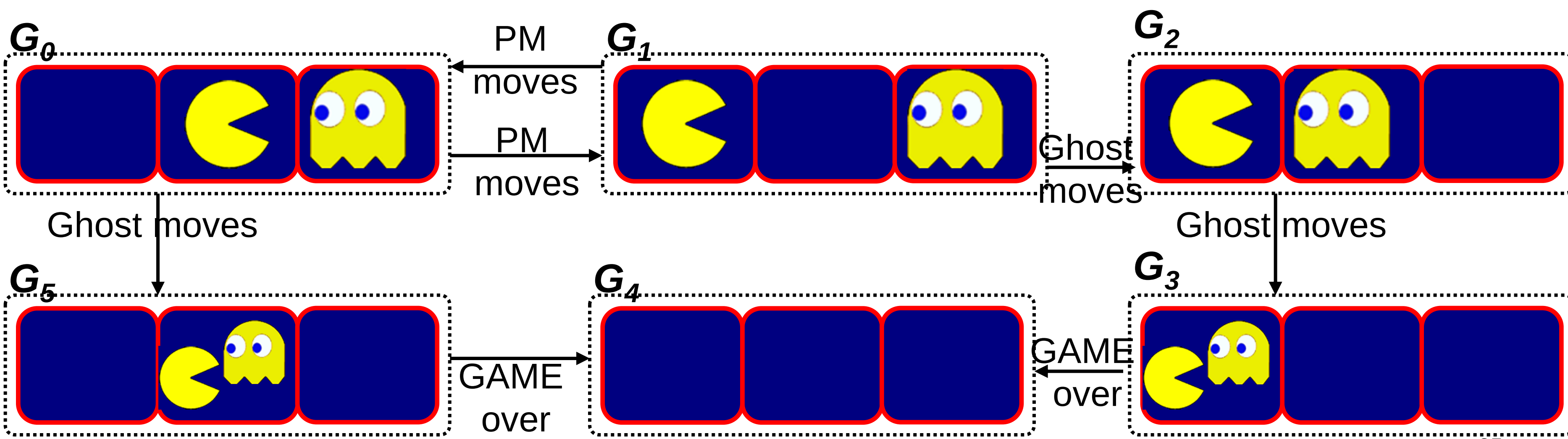


Question: How do we make the Ghost Smarter?

A Smarter Ghost

Rule: Ghost moves



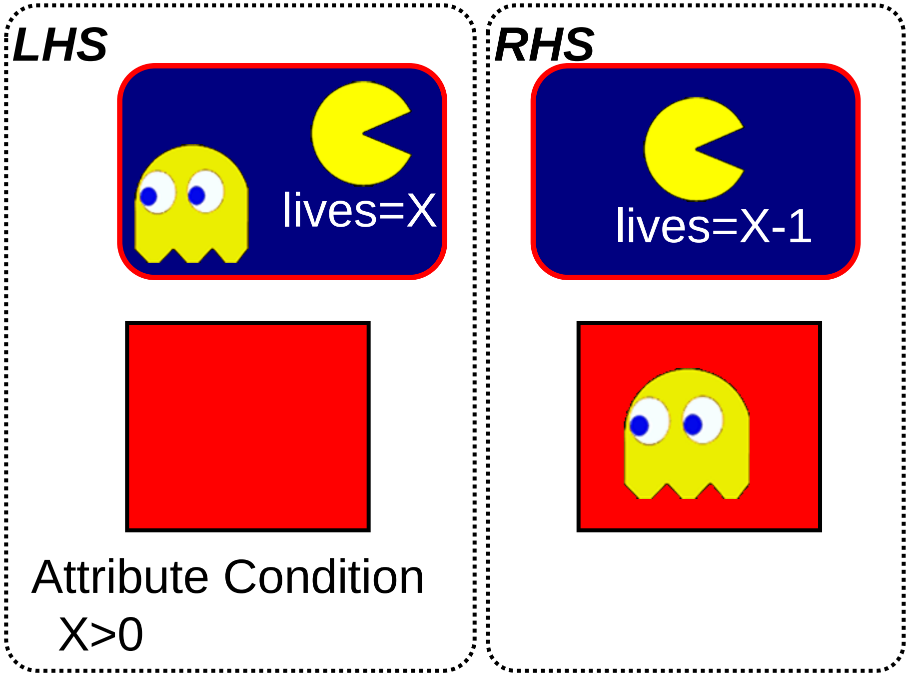


Attributes

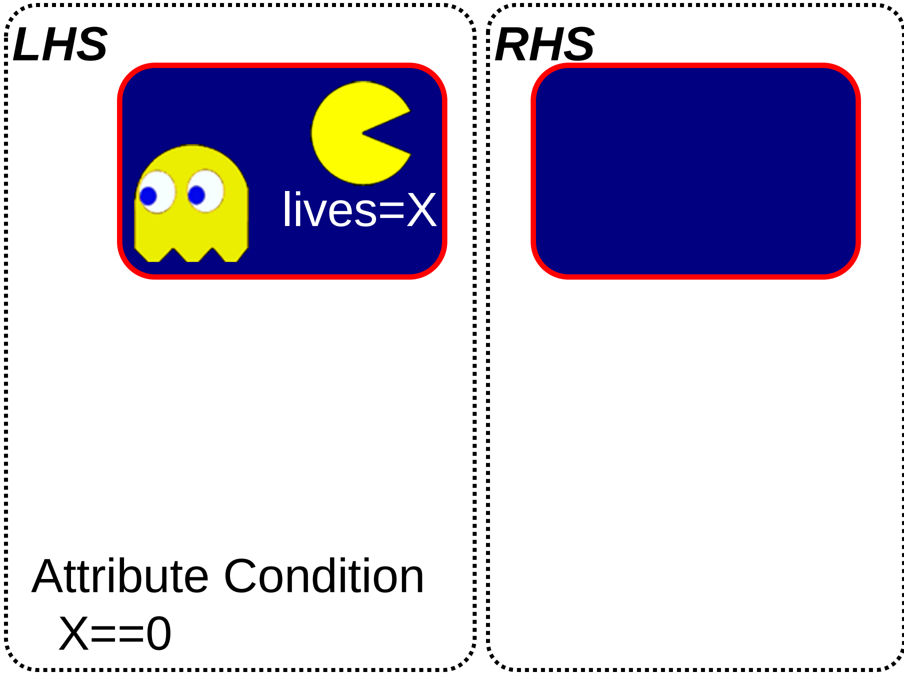
- Nodes and edges in graphs can be assigned attributes of some sort.
- Nodes and edges in rules can be assigned attributes, whose values are variables.
- The possible values that the variables can take are controlled by formulae: ***Attribute conditions***.
- We can assign new values to the attributes in the RHS: ***Attribute computations***.

Attributes

Rule: PACMAN hurted

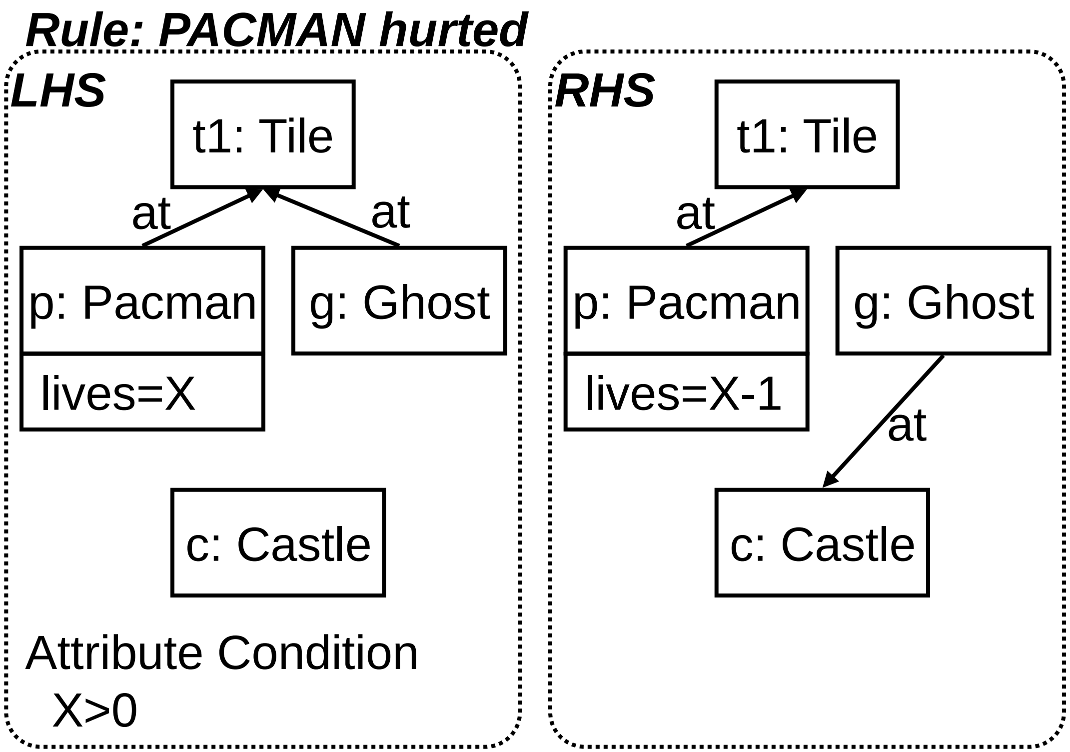


Rule: GAME over



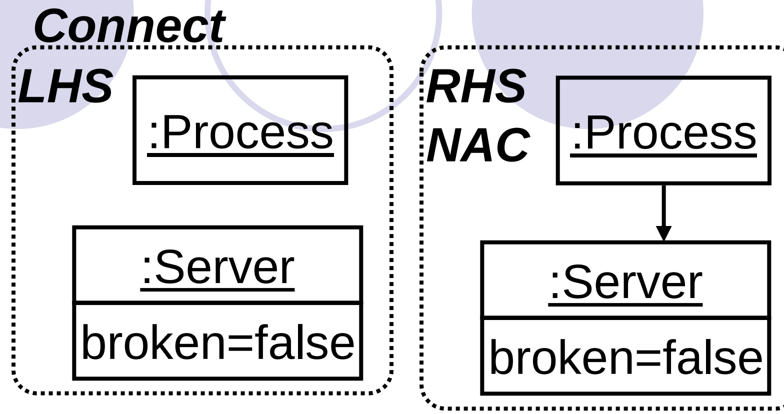
Attributes

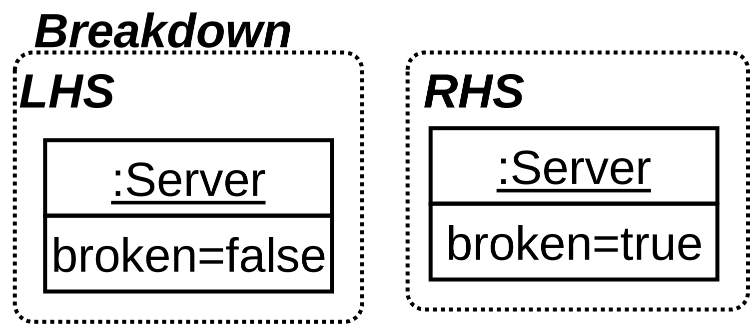
In abstract syntax

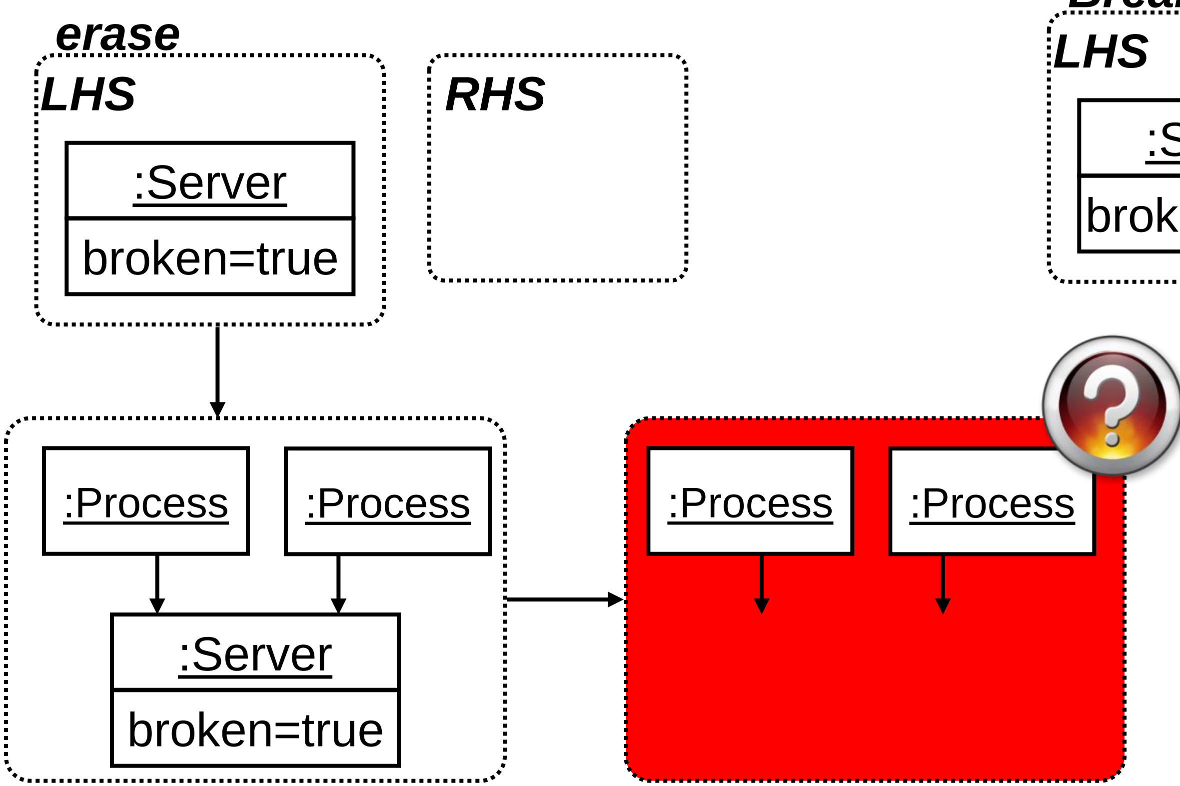


Dangling Edges

- What happens if a rule states that a node should be deleted, and such node receives arbitrary edges in the host graph?

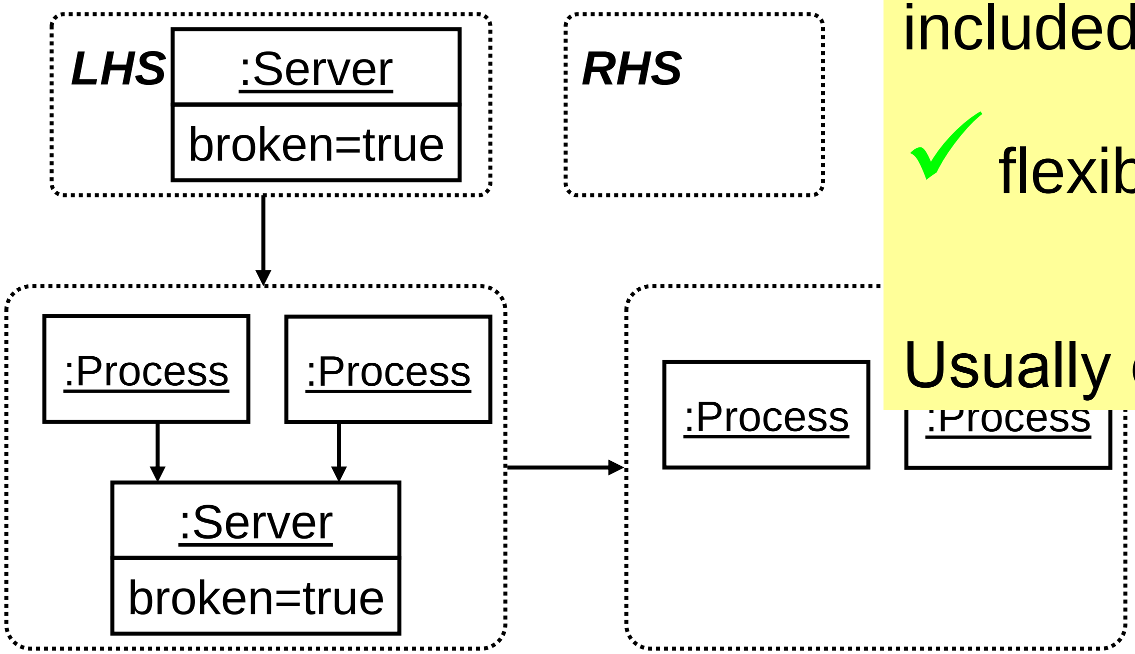






Dangling Edges

- We no longer have a graph, as several dangling edges appear.
- Two options:
  1. Delete the dangling edges.



✘ the rule may have secondary effects, not included in its definition.

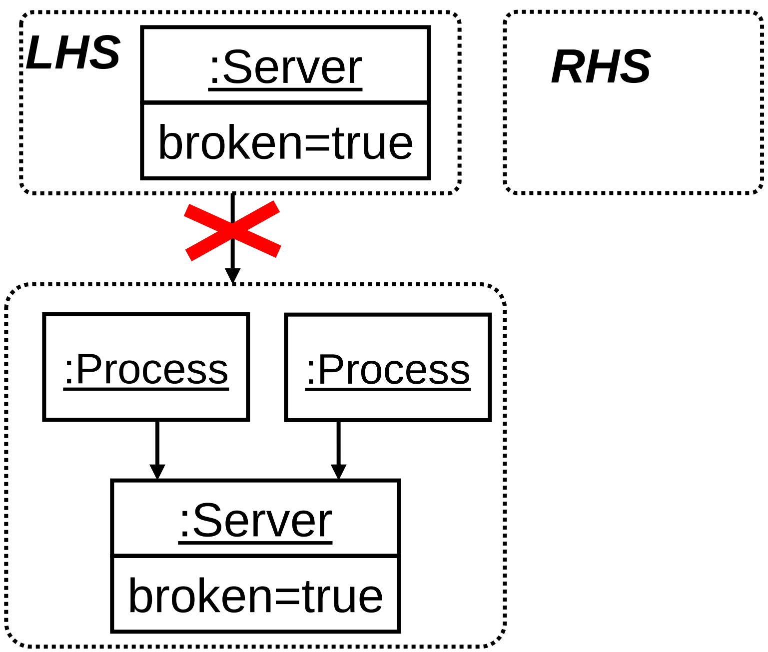
✓ flexibility.

Usually called ‘SPO’

Dangling Edges

- We no longer have a graph, as several dangling edges appear.
- Two options:

  2. Forbid rule application.



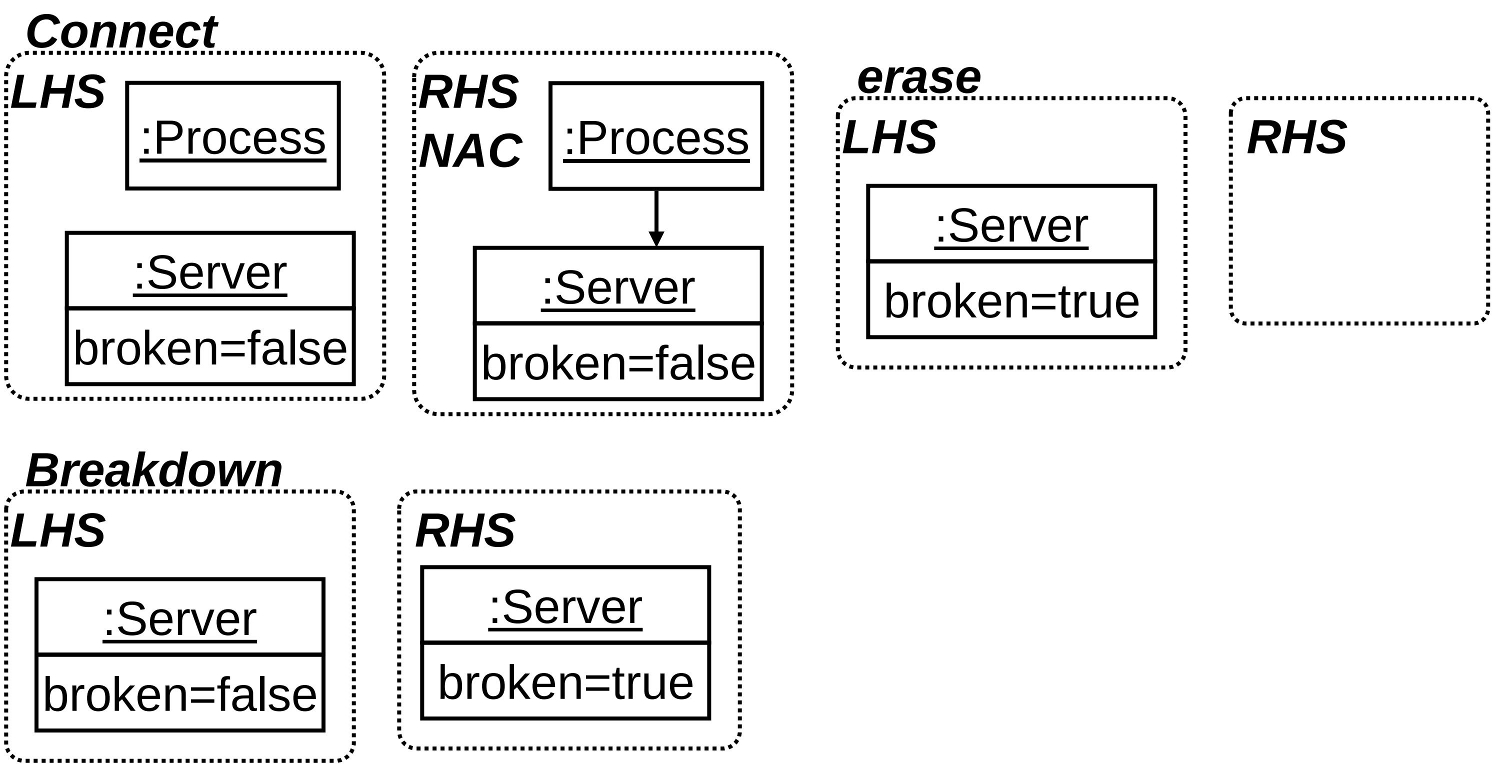
✗ We need additional rules to explicitly delete the node’s context.

✓ rules easier to analyse.

Usually called ‘DPO’

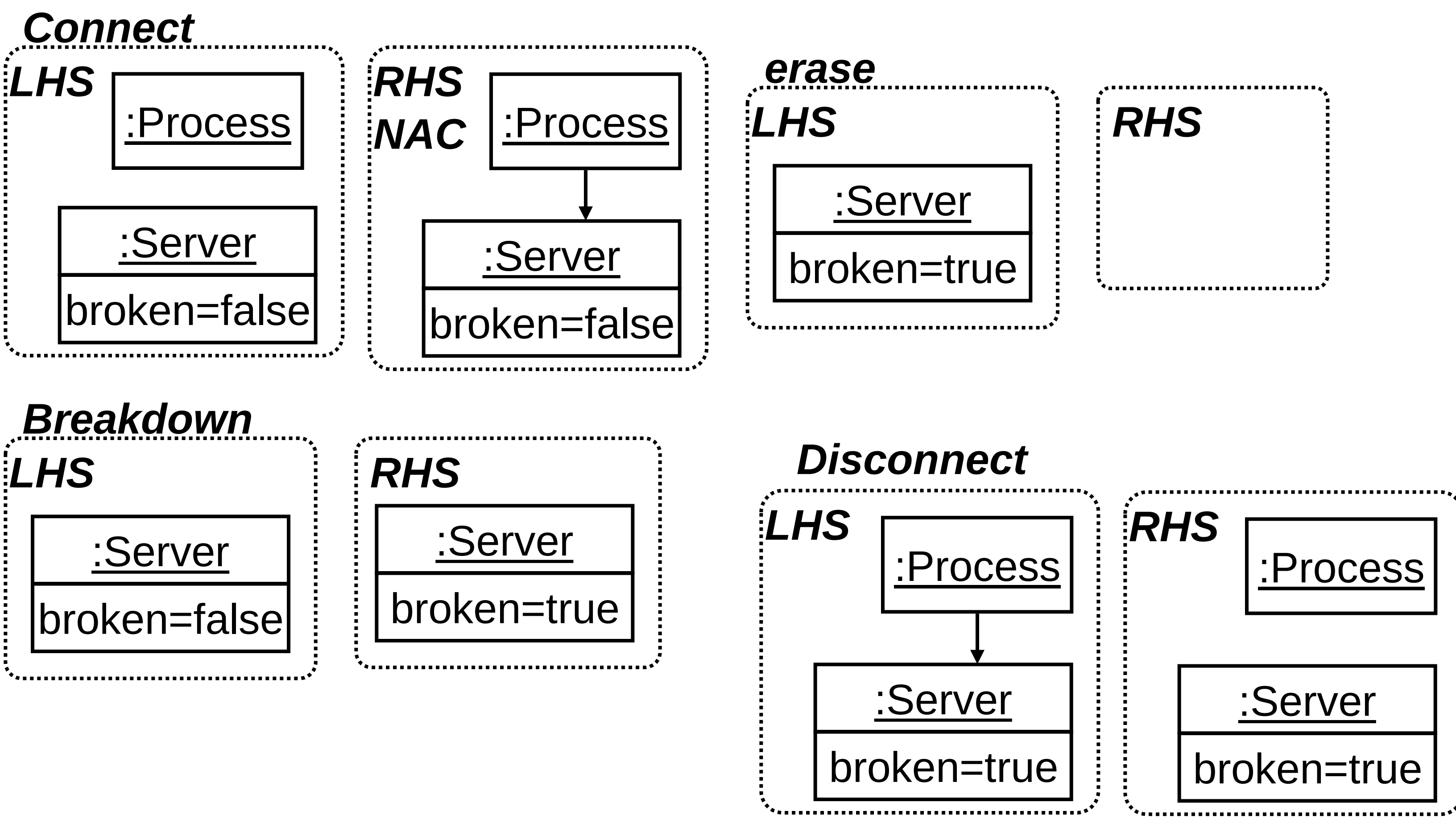
Dangling Edges

- Exercise: Design the missing rule(s) for the DPO case.



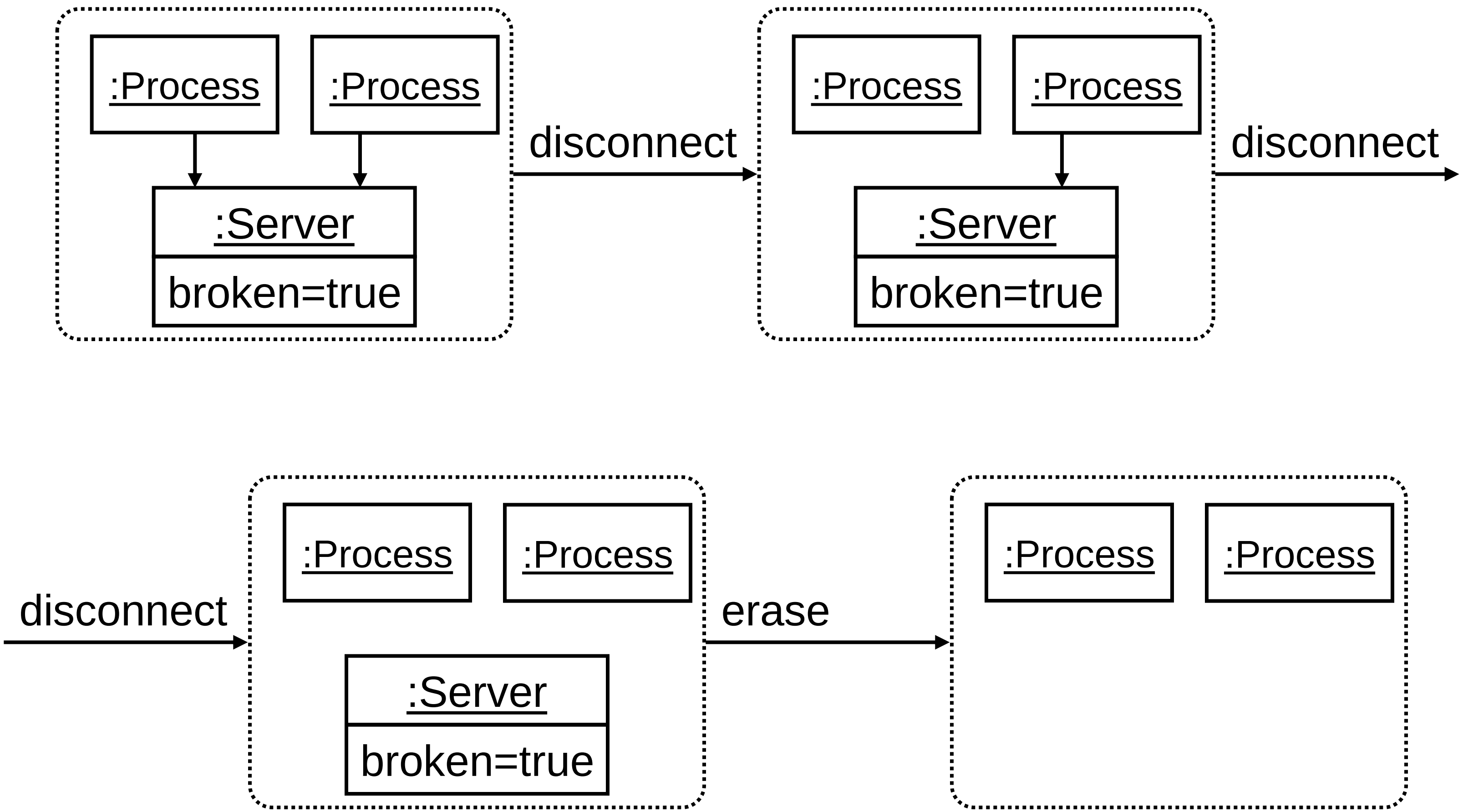
Dangling Edges

- Exercise: Design the missing rule(s) for the DPO case.



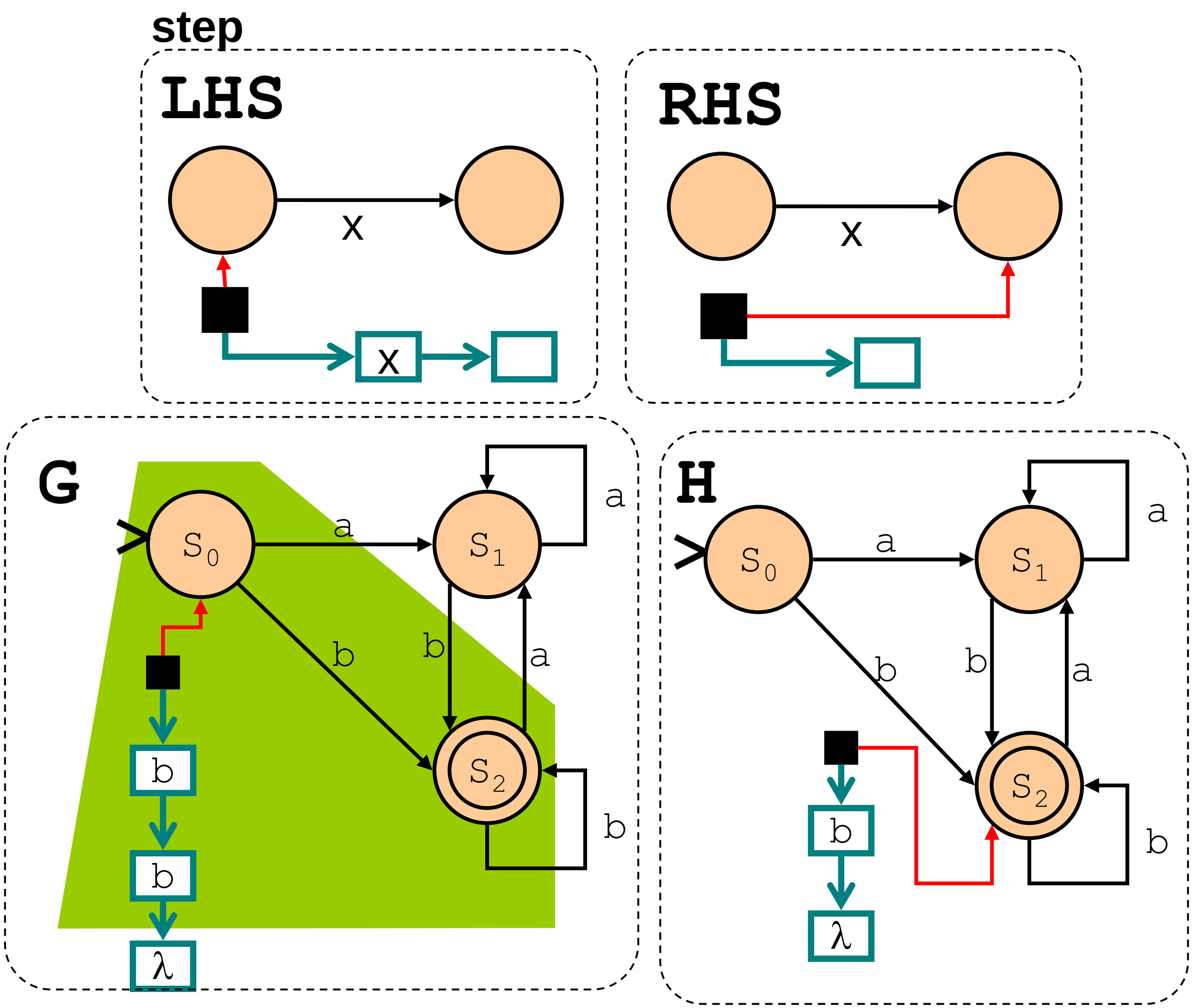
Dangling Edges

- DPO:



Non-Injective Match

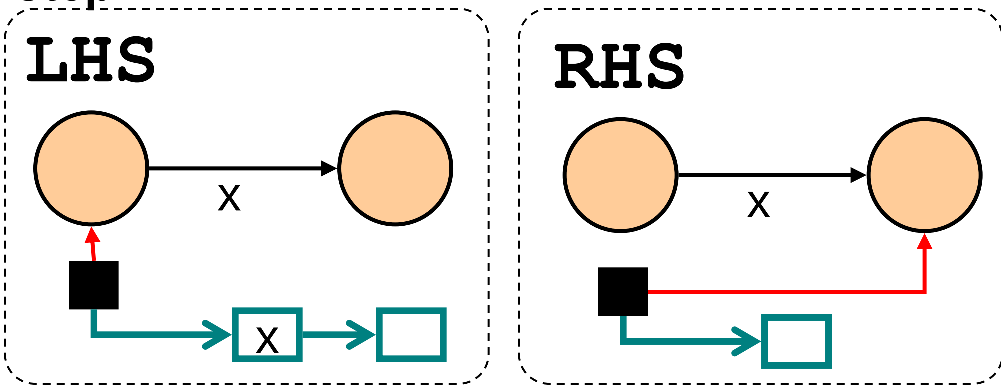
Example: Simulating a State Automaton

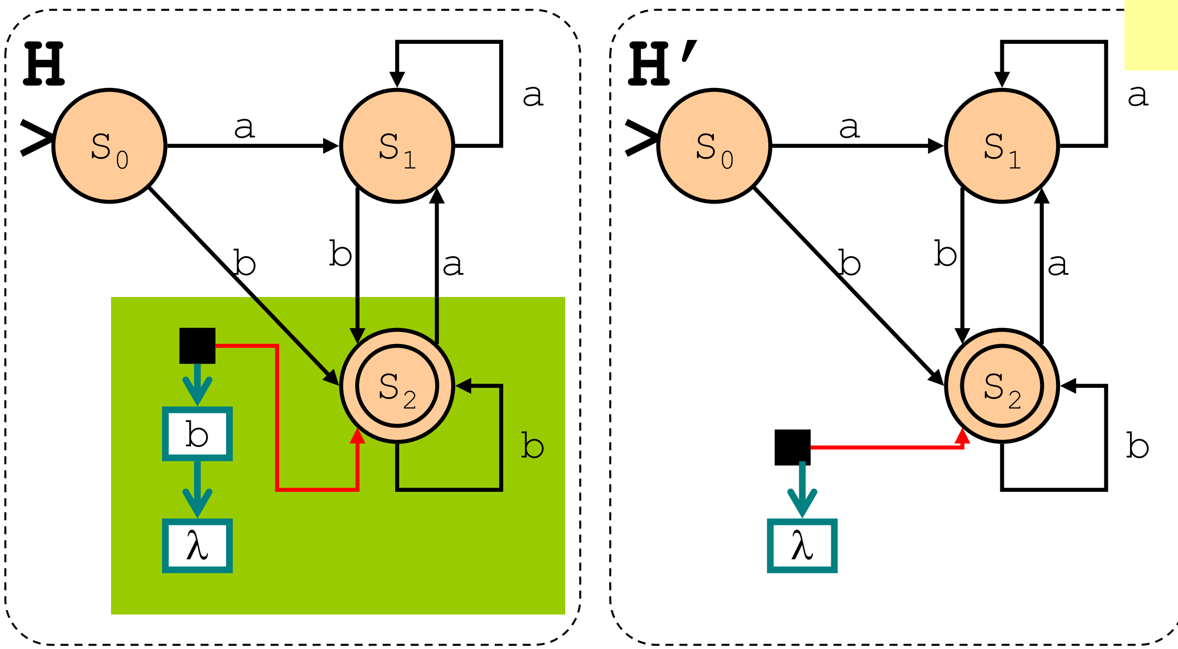


Non-Injective Match

Example: Simulating a State Automaton

step

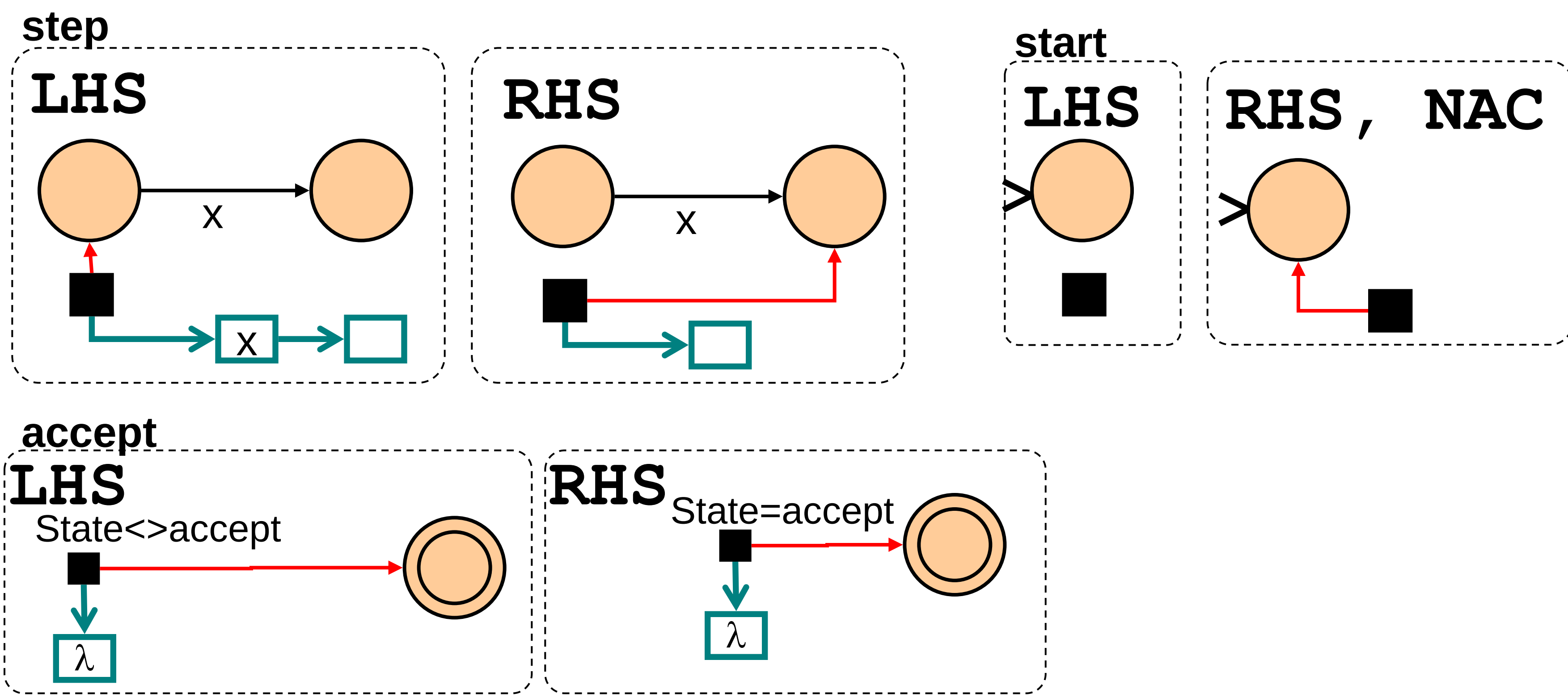




- Non-Injective match: Several elements of the rule can be identified to the same element in the host graph.

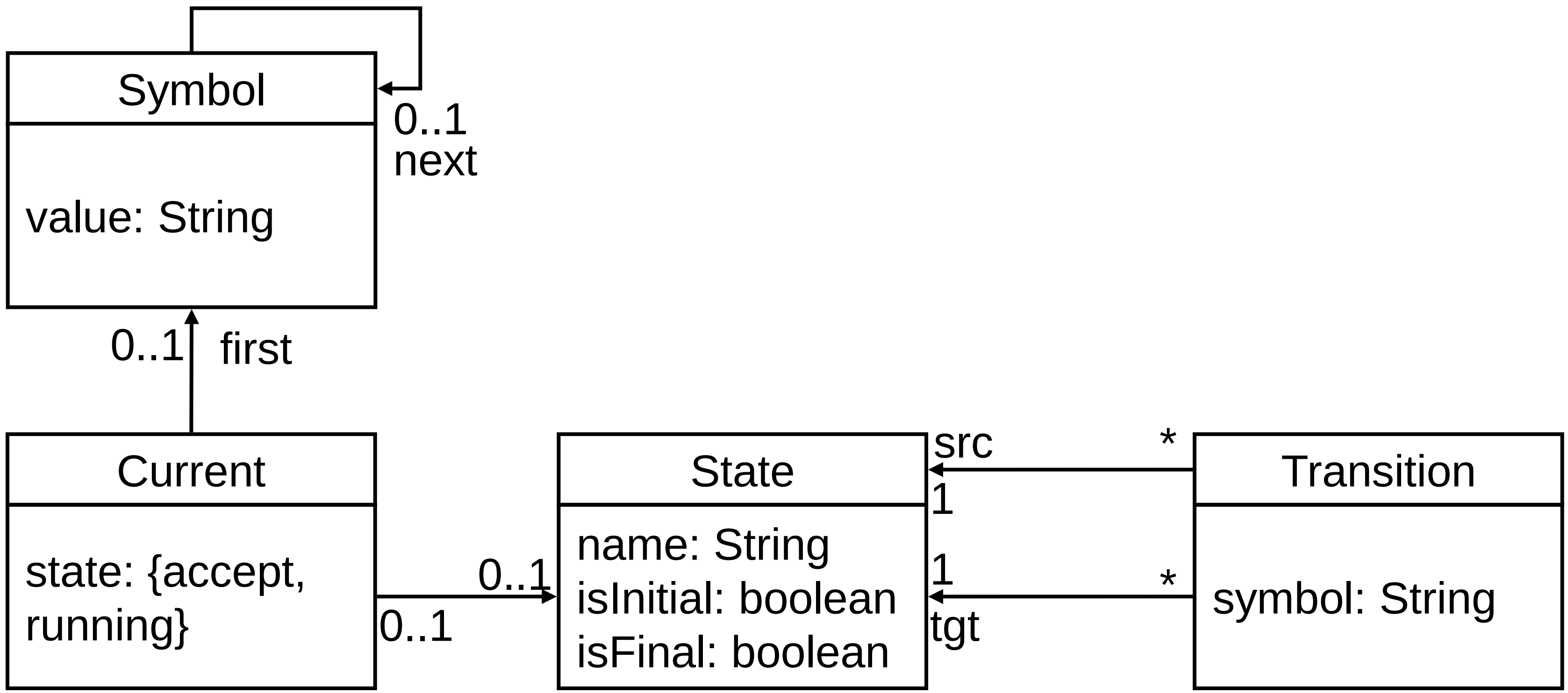
Non-Injective Match

Example: Simulating a State Automaton



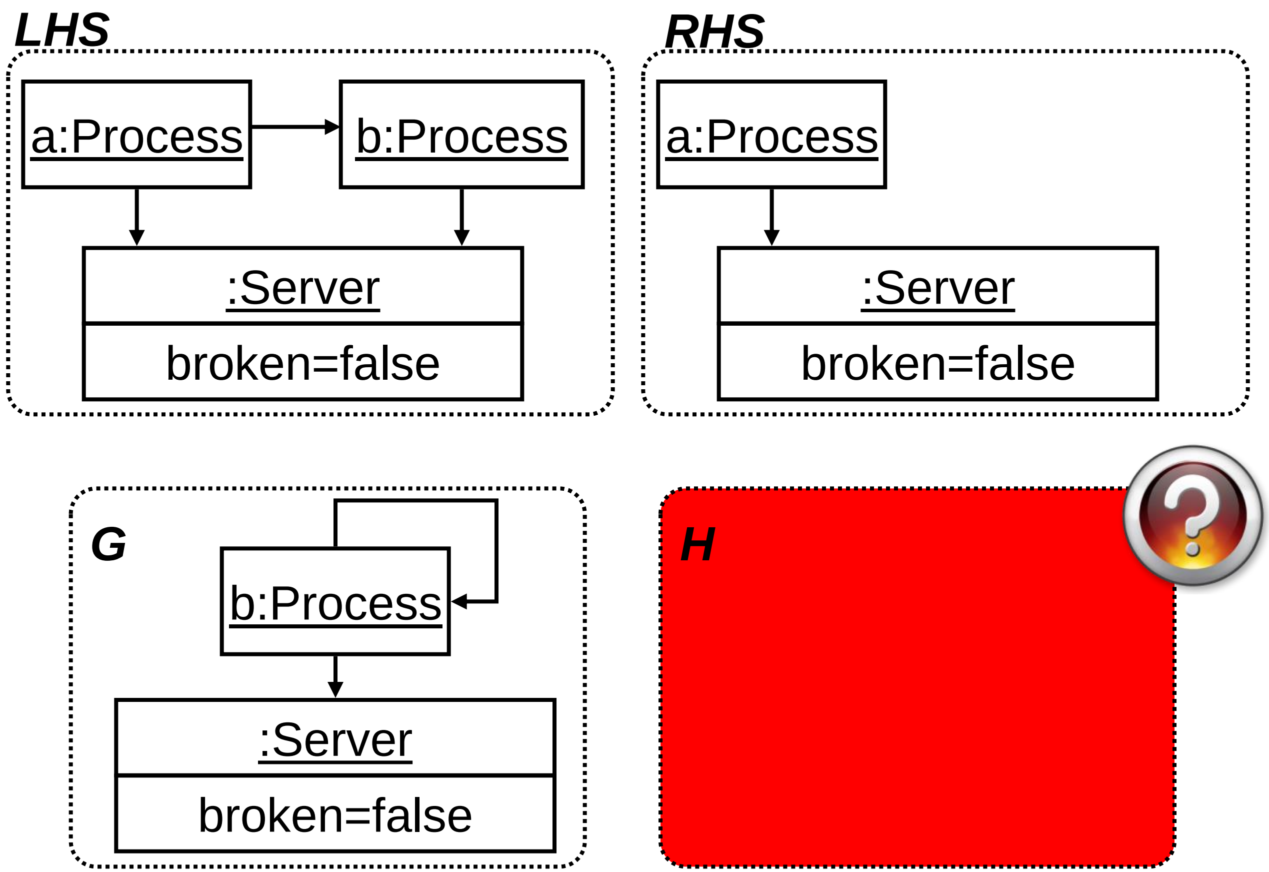
Non-Injective Match

Example: Simulating a State Automaton



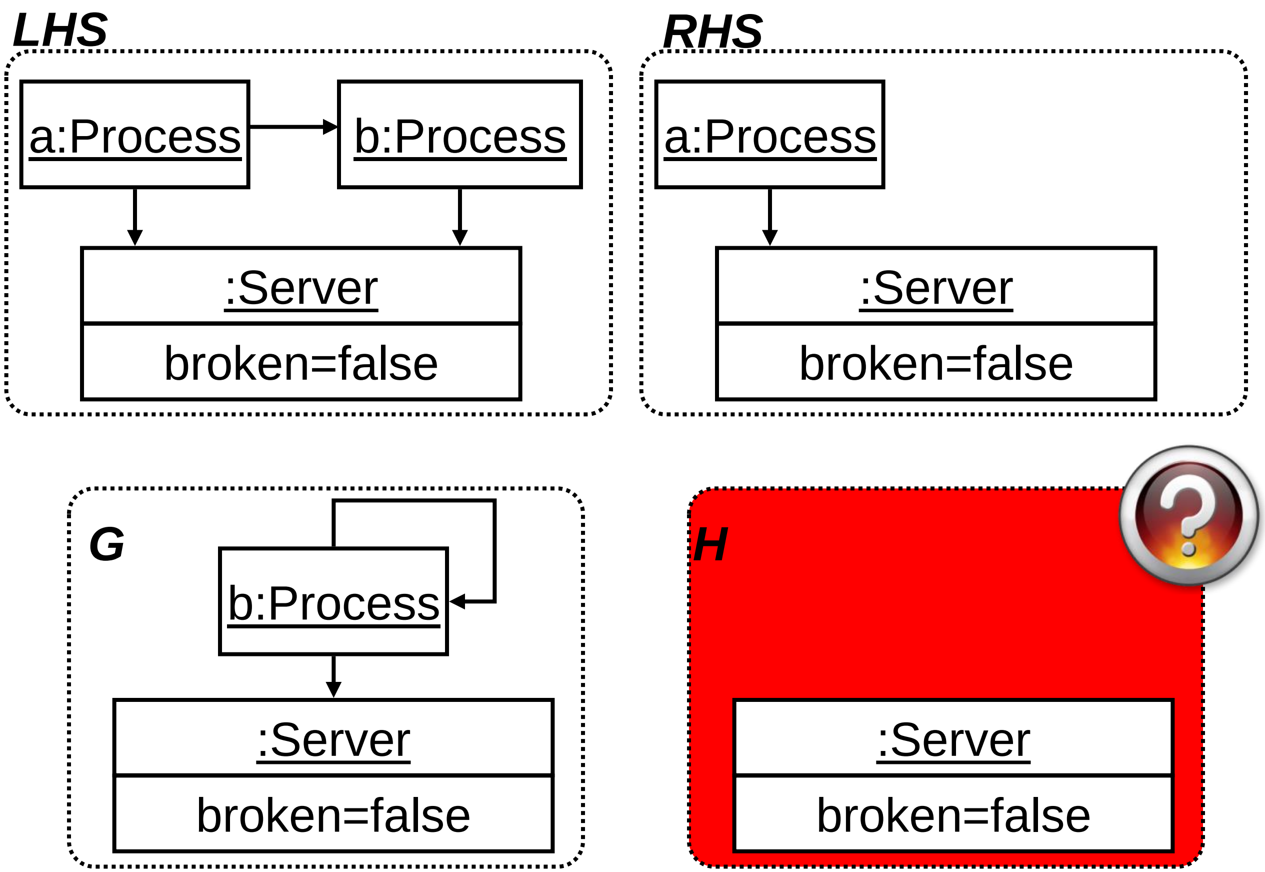
Non-Injetive Matches

- **Identification condition:** What happens if two elements are identified into one, and the rule states that one should be deleted and the other preserved?.



Non-Injetive Matches

- **Option 1 [SPO]:** All elements are deleted.



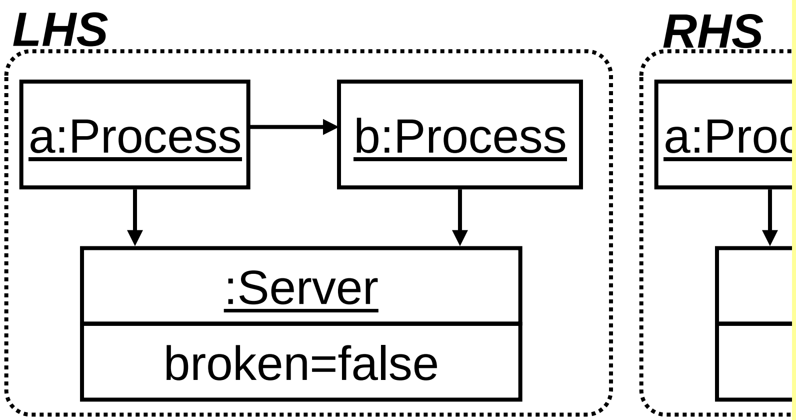
Non-Injetive Matches

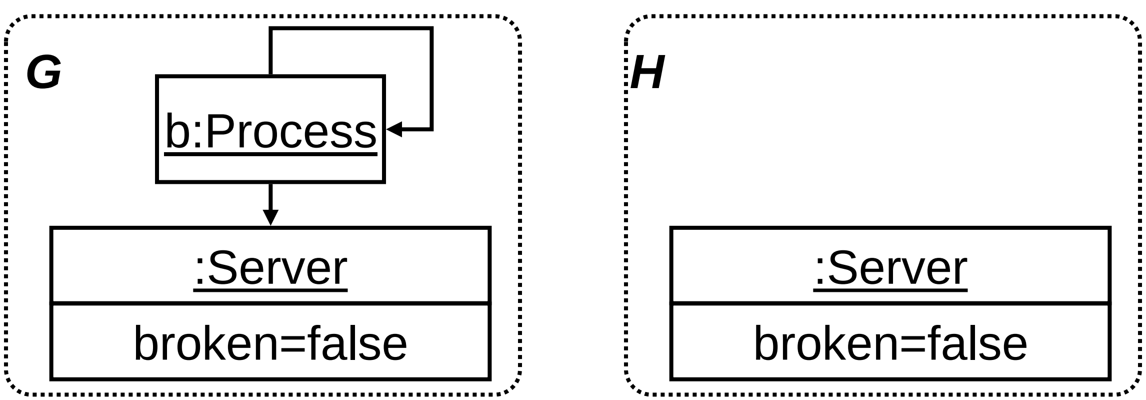
- **Option 1 [SPO]:** All elements are deleted

In SPO dangling edges are deleted.

✘ the rule may have secondary effects, not included in its definition.

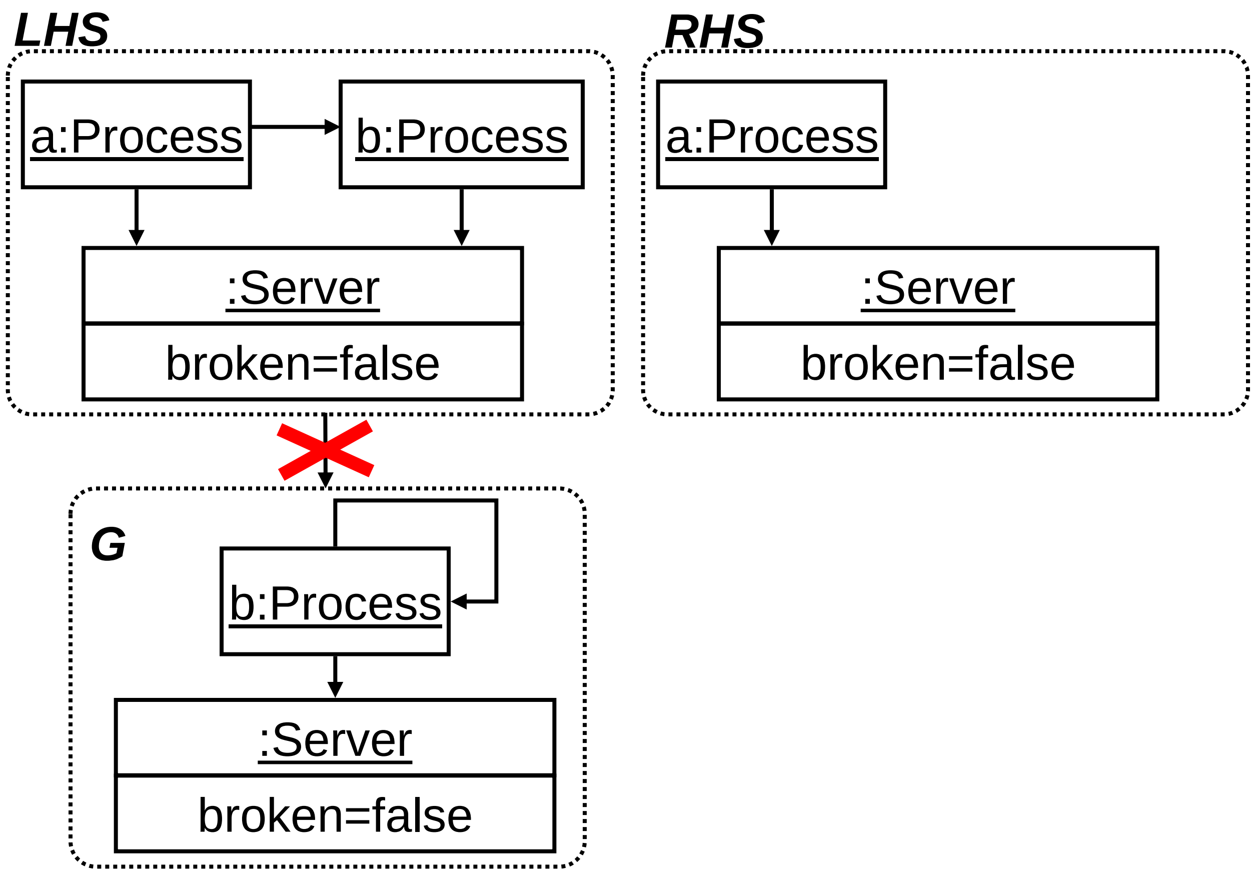
✓ flexibility.





Non-Injetive Matches

- **Option 2 [DPO]:** the rule cannot be applied at that match.



Index

- Introduction.
- Overview of Graph Transformation.
- **Exercises** 
- Control Languages.
- Meta-modelling and Graph Transformation.
- Applications.
- Tools.
- Conclusions.

Exercise

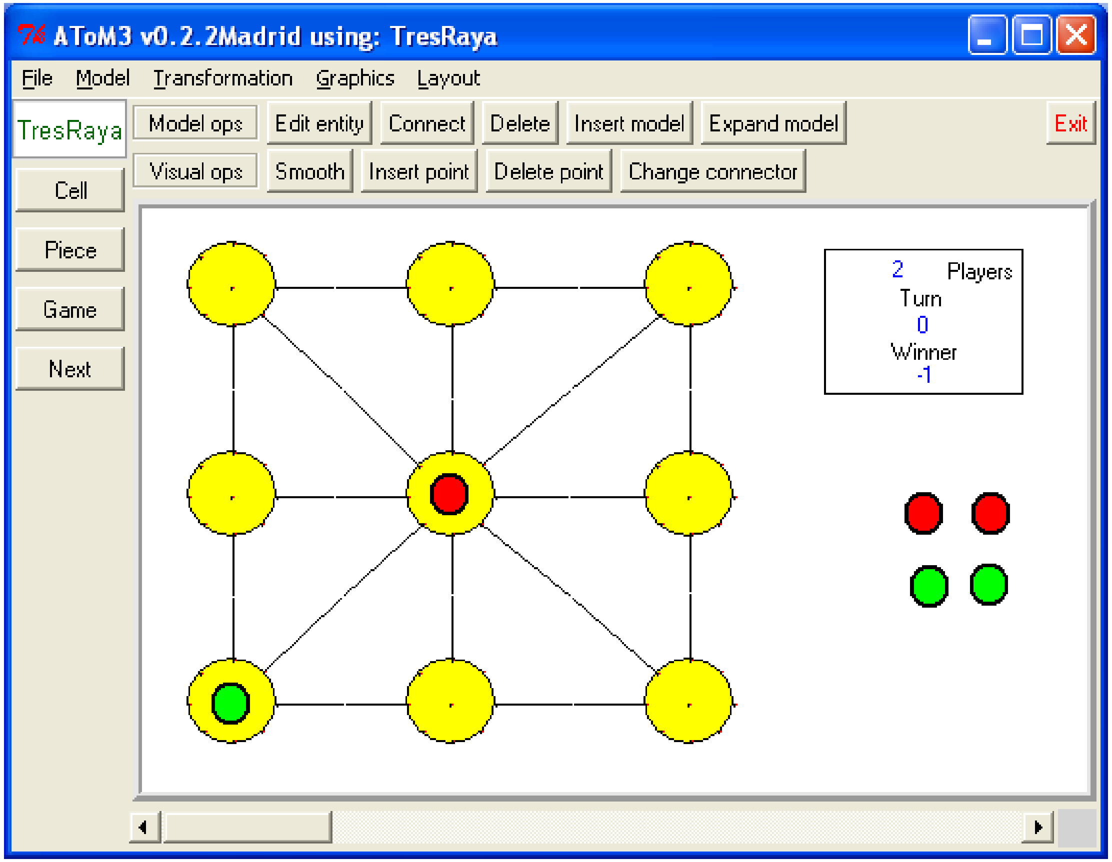
- Model the dynamics of a simple messaging protocol:
  - The network is made of nodes and routers.
  - Nodes are connected to routers, and routers can be connected to other routers.
  - Nodes can create messages, directed to a given target node.
  - Nodes propagate messages to routers, and these to one of the routers they are connected with, randomly. But if the destination node is connected to a given router, the message is forwarded to the target node.
  - When a node receives a message directed to it, it sends back an ack message to the sender.

Exercise

- Model the dynamics of simple messaging protocol:
  - The network is made of nodes and routers.
  - Nodes are connected to routers, and routers can be connected to other routers.
  - Nodes can create messages, directed to a given target node.
  - Nodes propagate messages to routers, and these to one of the routers they are connected with, randomly. But if the destination node is connected to a given router, the message is forwarded to the target node.
  - When a node receives a message directed to it, it sends back an ack message to the sender.
  - Routers have a maximum message capacity, so that they cannot receive more messages if they reach such capacity.

Exercise

- Model the dynamics of TIC-TAC-TOE.

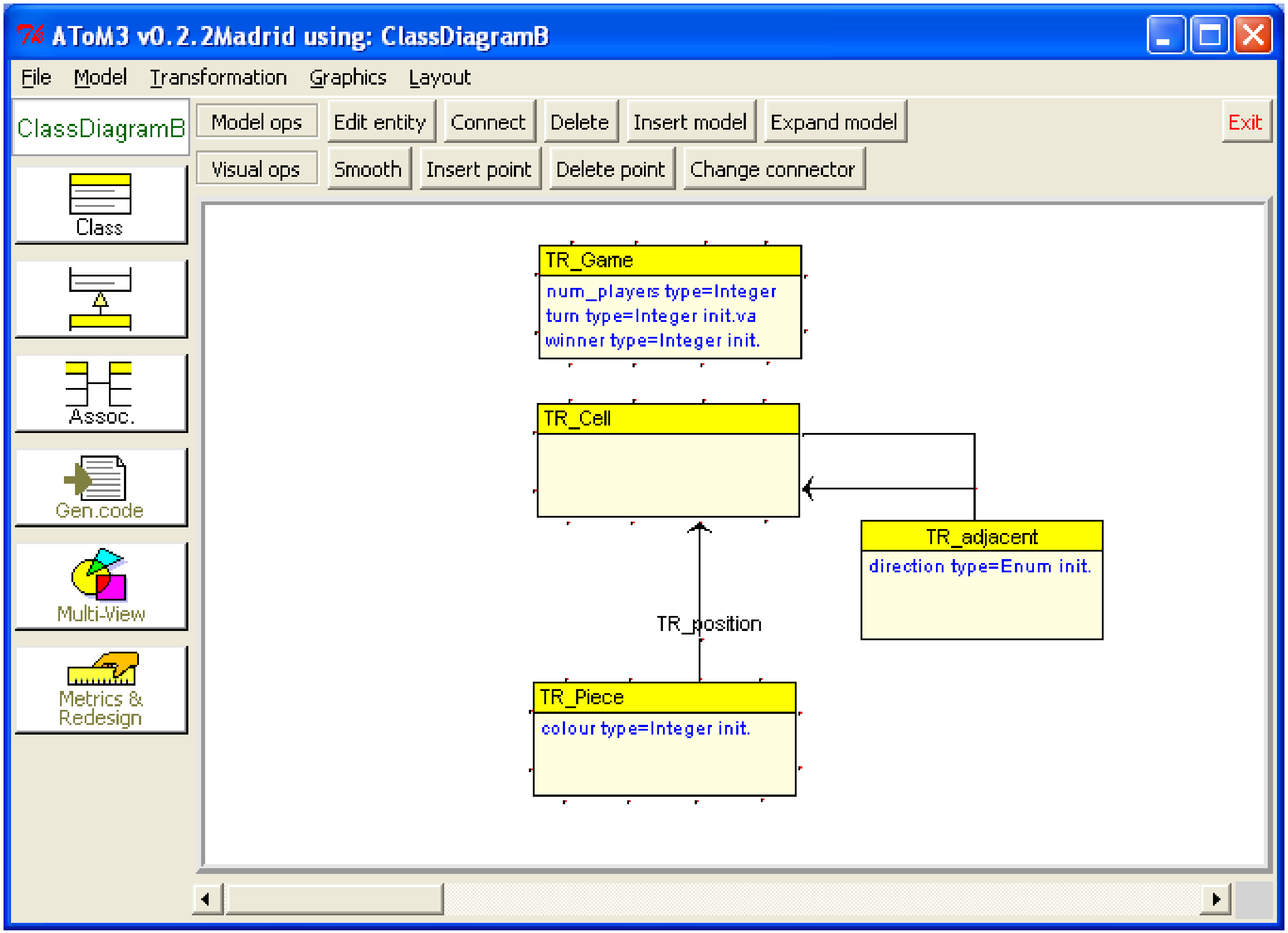


Exercise

- Step 1: Meta-Model

- Step 2: GT rules.

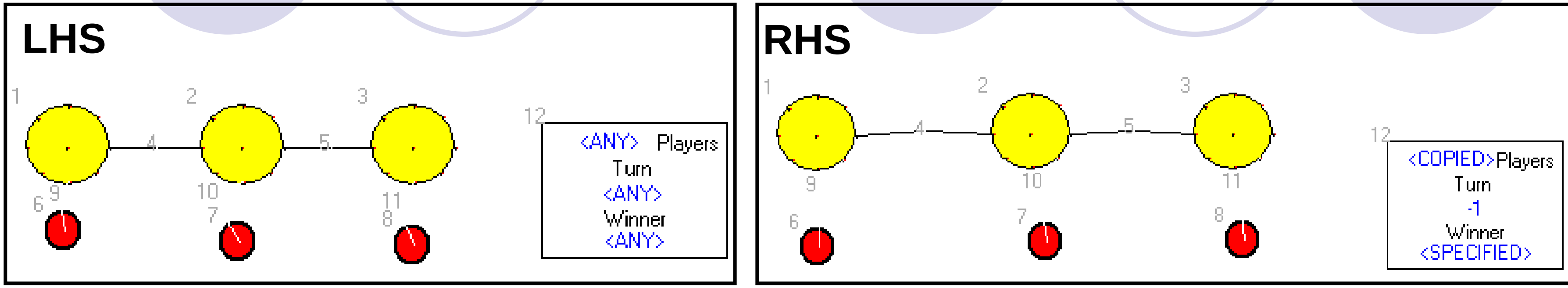
Exercise



AToM3 v0.2.2Madrid using: ClassDiagramB
File Model Transformation Graphics Layout
ClassDiagramB
Model ops
Edit entity
Connect
Delete
Insert model
Expand model
Exit
Visual ops
Smooth
Insert point
Delete point
Change connector
Class
Assoc.
Gen.code
Multi-View
Metrics & Redesign
TR_Game
num_players type=Integer
turn type=Integer init.va
winner type=Integer init.
TR_Cell
TR_adjacent
direction type=Enum init.
TR_position
TR_Piece
colour type=Integer init.

Rules

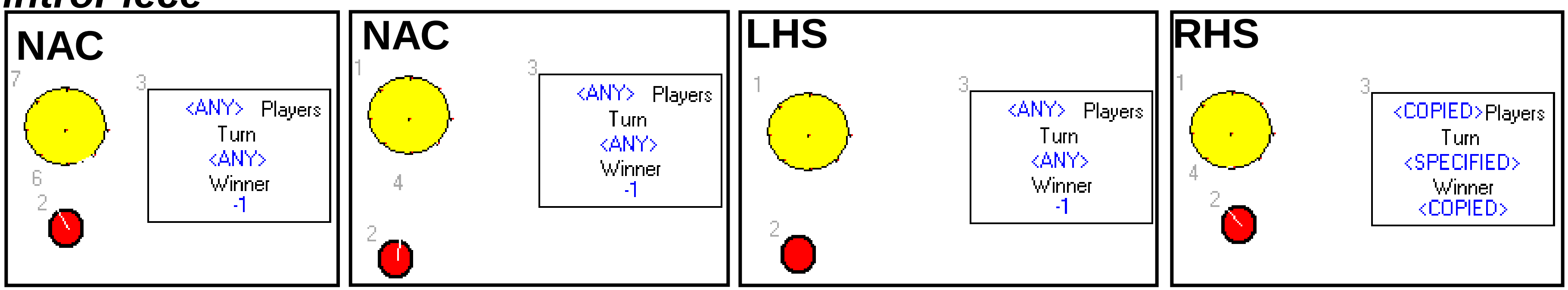
winGame

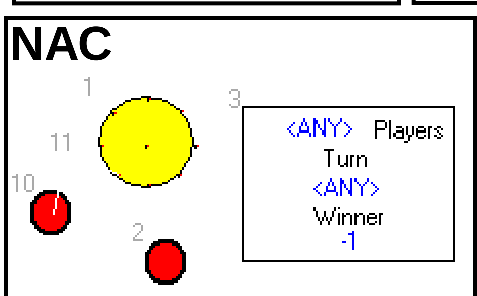


ATTRIBUTE CONDITIONS

(n(4).direction==n(5).direction) and (n(12).winner==-1) and (n(6).colour==n(7).colour) and (n(7).colour==n(8).colour)

introPiece





ATTRIBUTE CONDITIONS

n(3).turn==n(2).colour

ATTRIBUTE COMPUTATIONS

n(3).turn = (n(3).turn+1)%n(3).num_players

Index

- Introduction.
- Overview of Graph Transformation.
- Exercise: Playing tic-tac-toe.
- **Control Language.** 
- Meta-modelling and Graph Transformation.
- Applications.
- Tools.
- Conclusions.

Control Languages

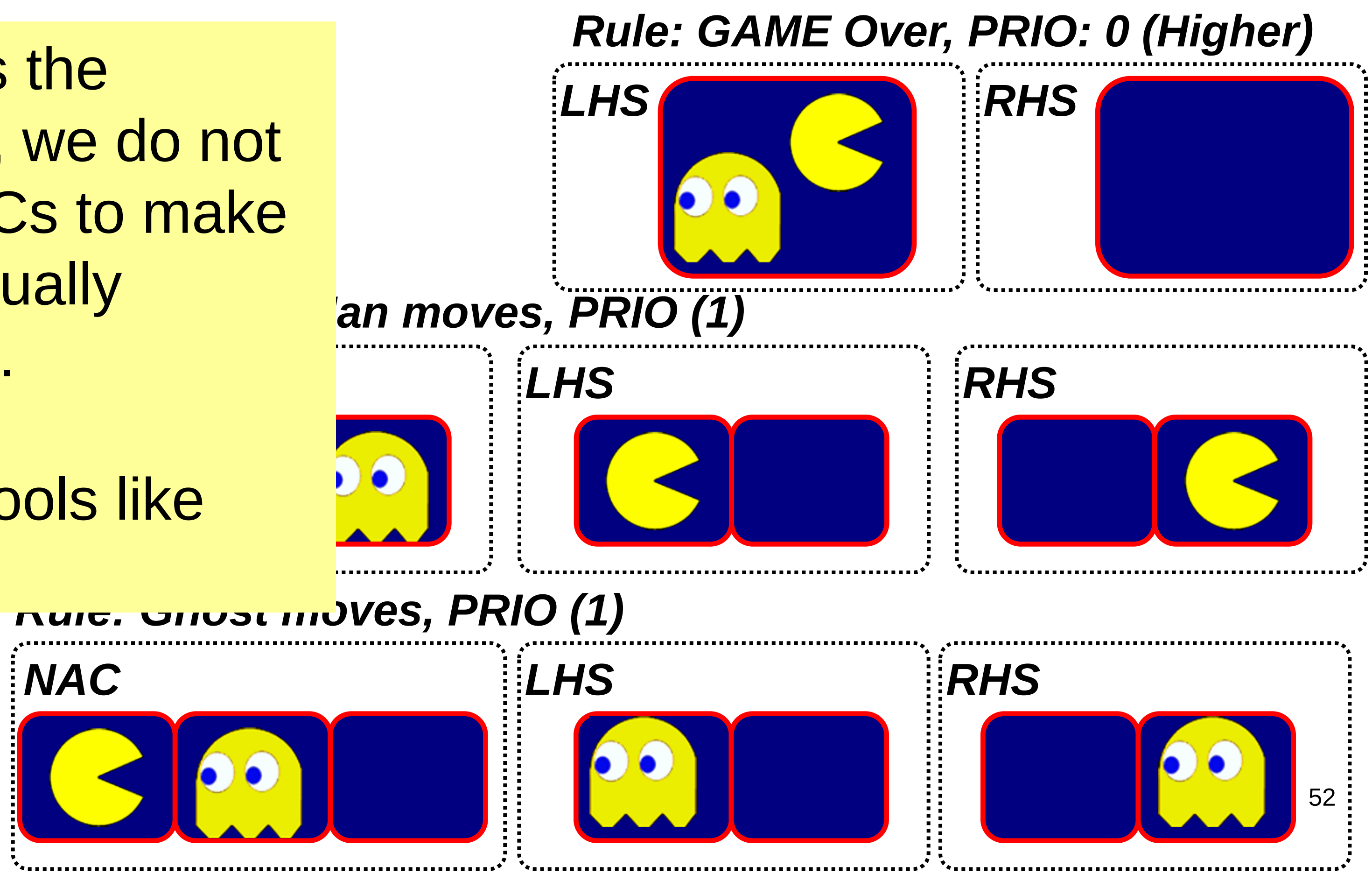
- Putting all rules into a set is not practical for complex problems.
- A means to control the order of rule execution:
  - Rule priorities.
  - Rule layers.
  - Full-fledged control languages:
    - Transformation Units.
    - Story diagrams (similar to activity diagrams).

Priorities

- If several rules are applicable, take the one with higher priority.

Simplifies the grammar, we do not need NACs to make rules mutually exclusive.

Used in tools like AToM^3



Layers

- Phasing mechanism, rules are assigned to layers.
  - Start with layer 0. Apply all its rules as long as possible until none is applicable anymore.
  - Follow with next layer, until all layers are visited.

Useful to express pre-processings and processes to be done in sequential order.

Used in tools like AGG

Transformation Units.

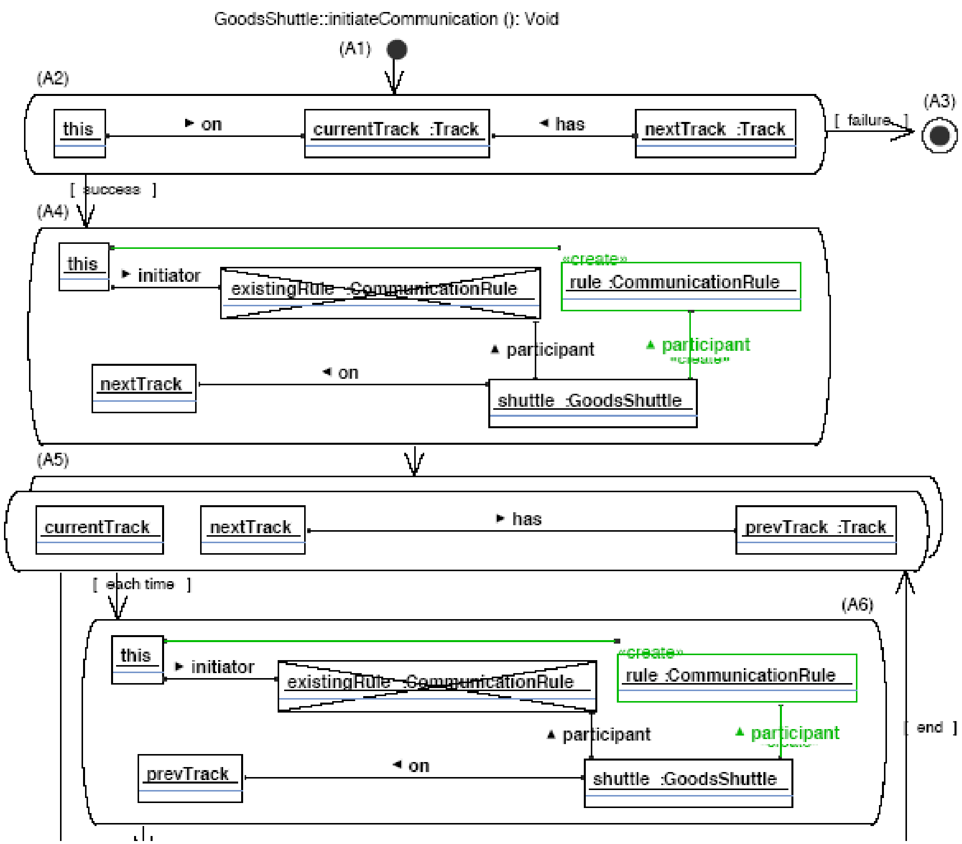
- Basic control structures (sequencing, as long as possible, etc).
  - Similar to regular expressions, where symbols are rule names.
  - It is also possible to control the conditions for graphs to be terminal.
  - Strong theoretical basis.

- Used in tools like Henshin
  - Independent unit: choose an applicable subunit at random
  - Loop unit: *
  - Priority unit: executes the subunit with highest priority
  - …

Hans-Jörg Kreowski, Sabine Kuske, Grzegorz Rozenberg. 2008. “Graph Transformation Units – An Overview”. In Concurrency, Graphs and Models, pp.: 57–75, 2008, vol 5065 of LNCS (Springer)

Story diagrams.

- Similar to activity diagrams, each state has associated a rule.



Complex execution flow.
Conditional branchings.

Used in tools like Fujaba (http://www.fujaba.de/)

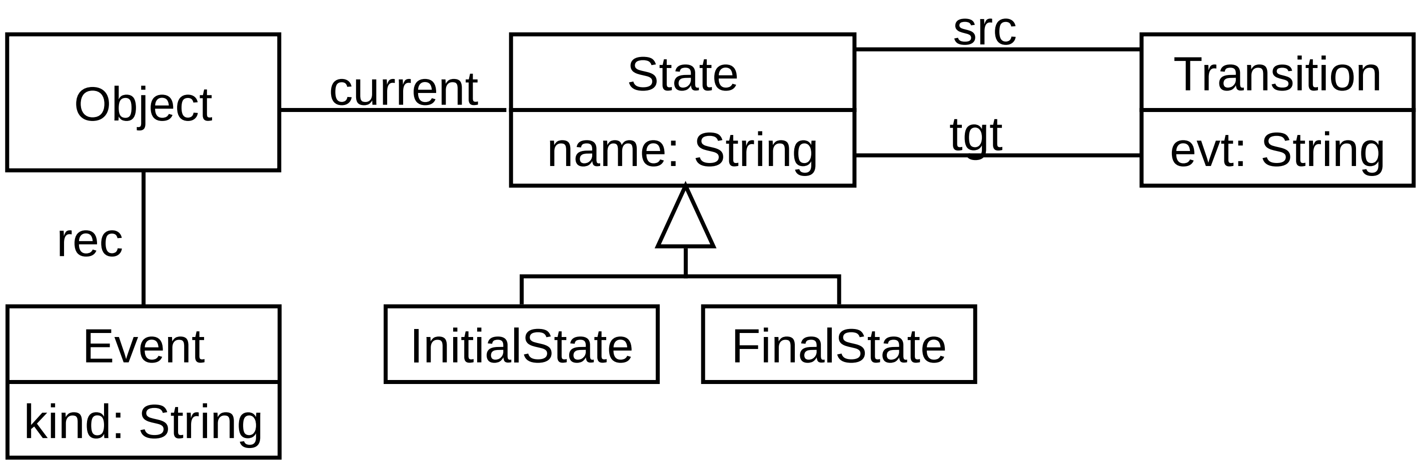
Index

- Introduction.
- Overview of Graph Transformation.
- Exercise: Playing tic-tac-toe.
- Control Language.
- **Meta-modelling and Graph Transformation.**

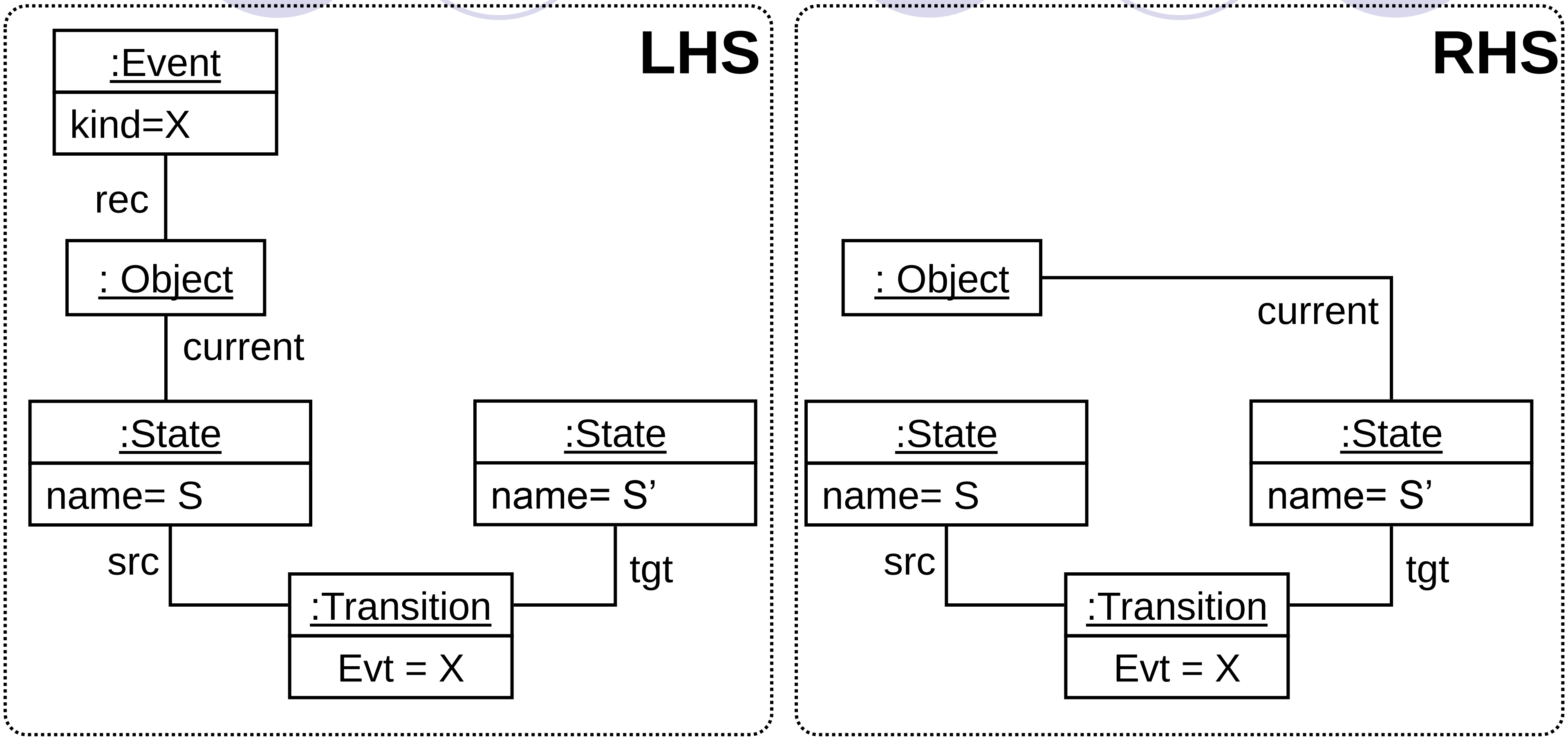
- Applications.
- Tools.
- Conclusions.

Meta-Modelling and GT

- Use the inheritance hierarchy of the meta-model in rules.
- Objects in rules can be matched by objects of any subclass in host graphs.
- More compact rules.

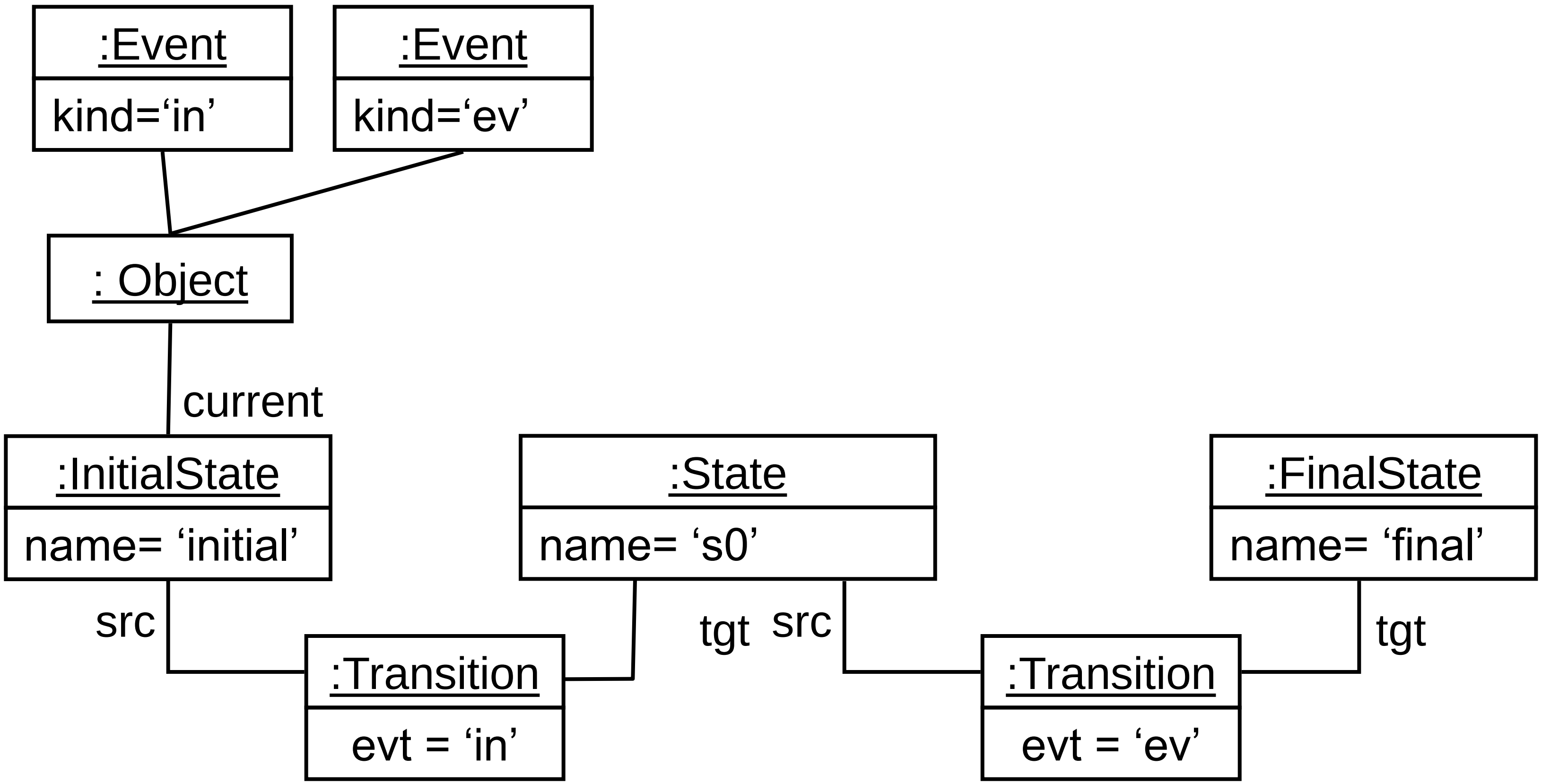


Meta-Modelling and GT

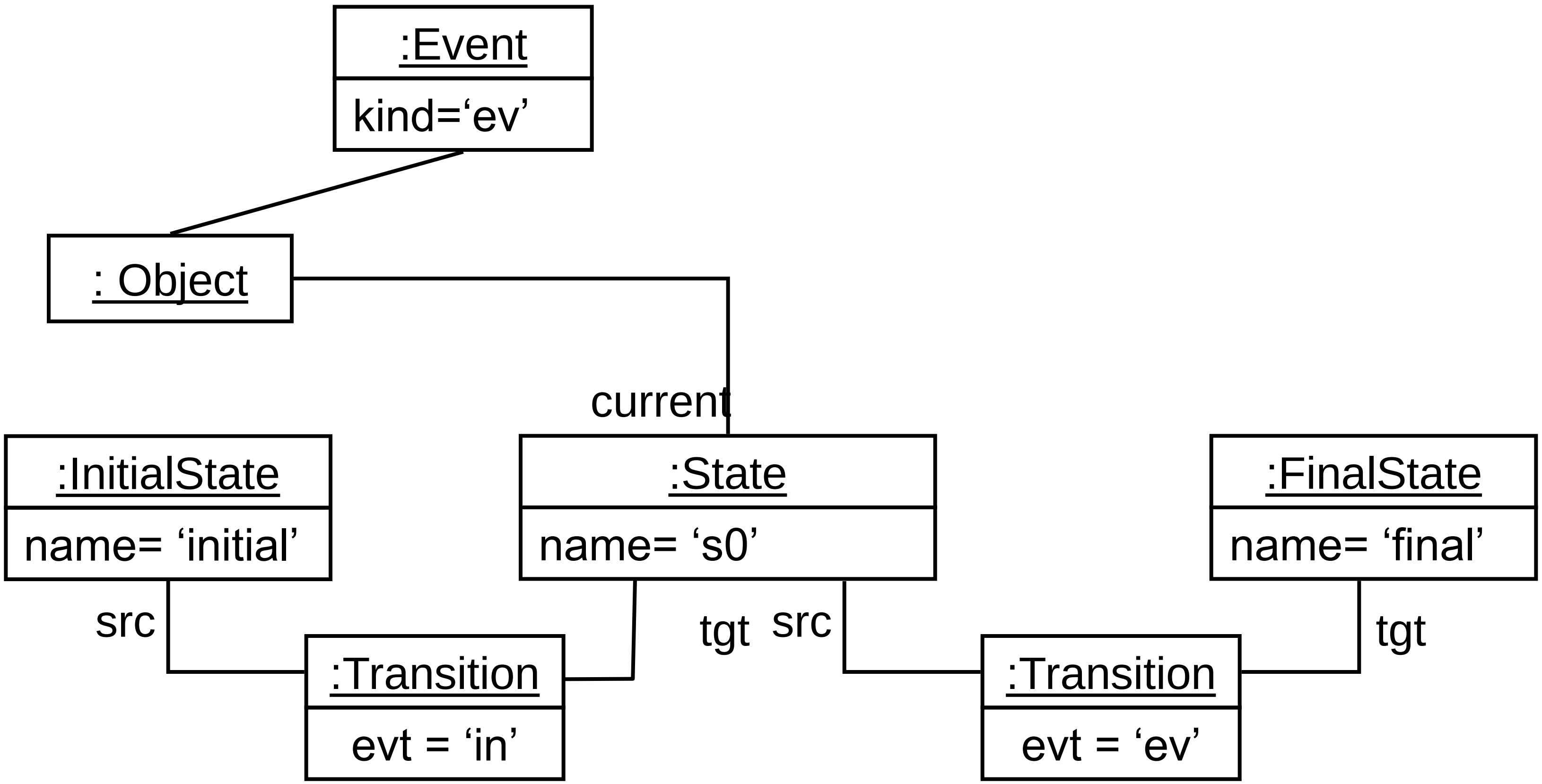


- The rule is equivalent to a set of 9 rules, by substituting the type of both State objects by all subtypes of State.

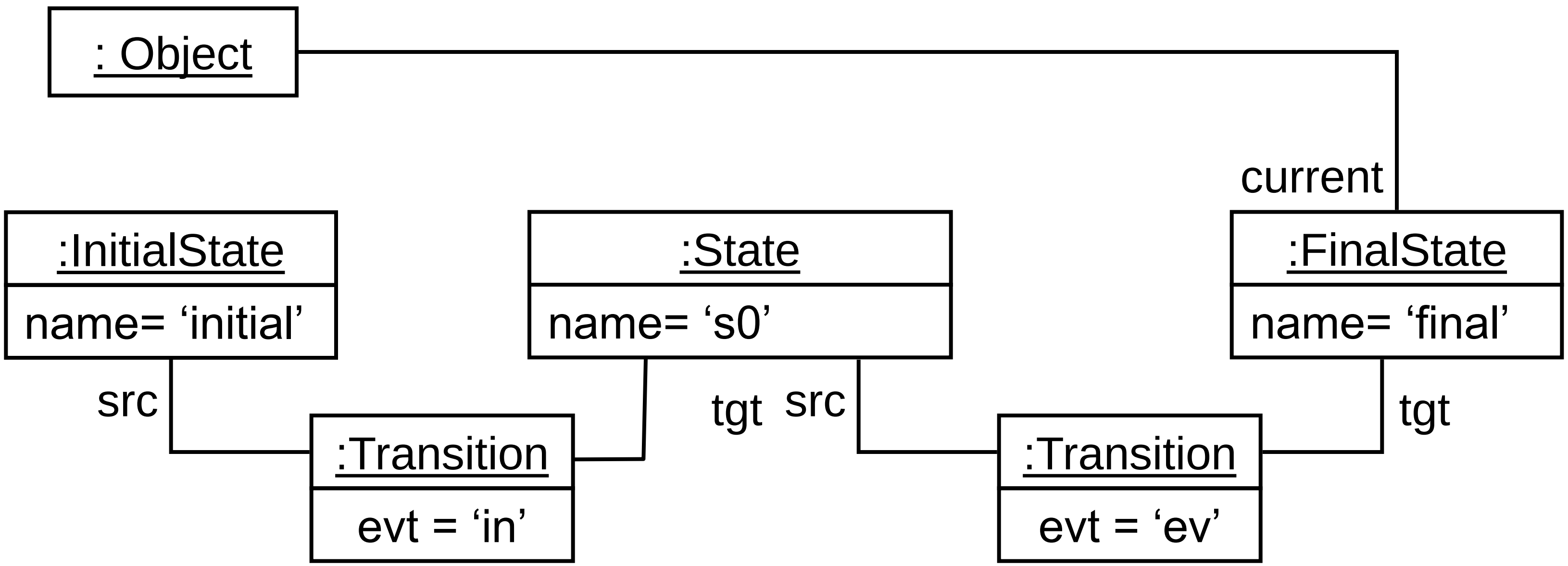
Meta-Modelling and GT



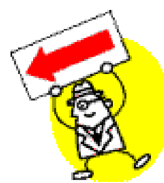
Meta-Modelling and GT



Meta-Modelling and GT



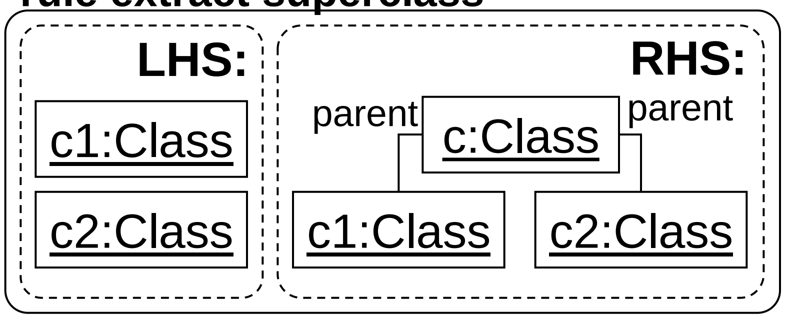
Index

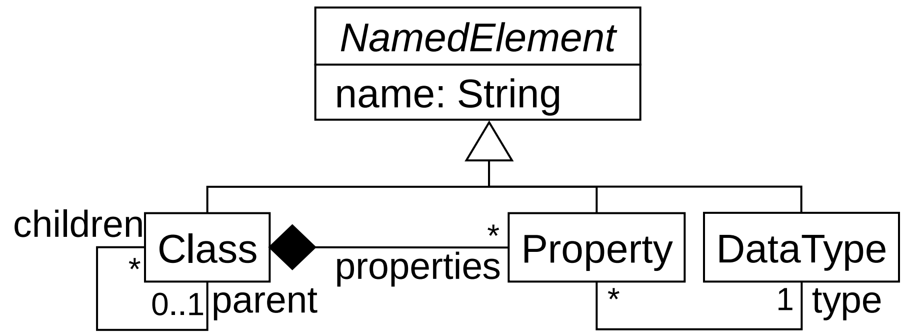
- Introduction.
- Overview of Graph Transformation.
- Exercise: Playing tic-tac-toe.
- Control Language.
- **Applications.** 
- Tools.
- Conclusions.

Applications

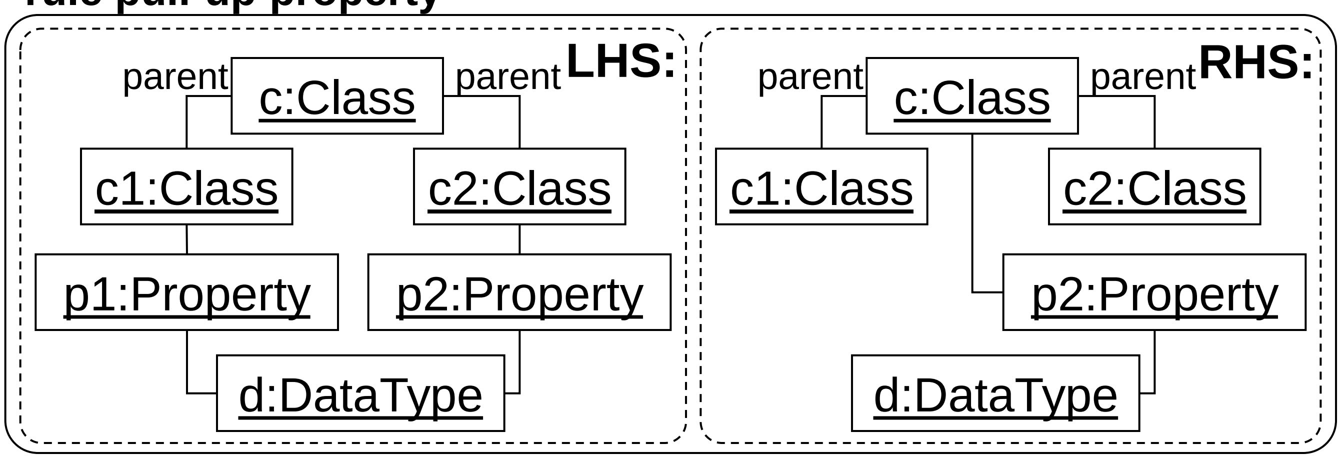
Refactoring of OO models.

rule extract superclass





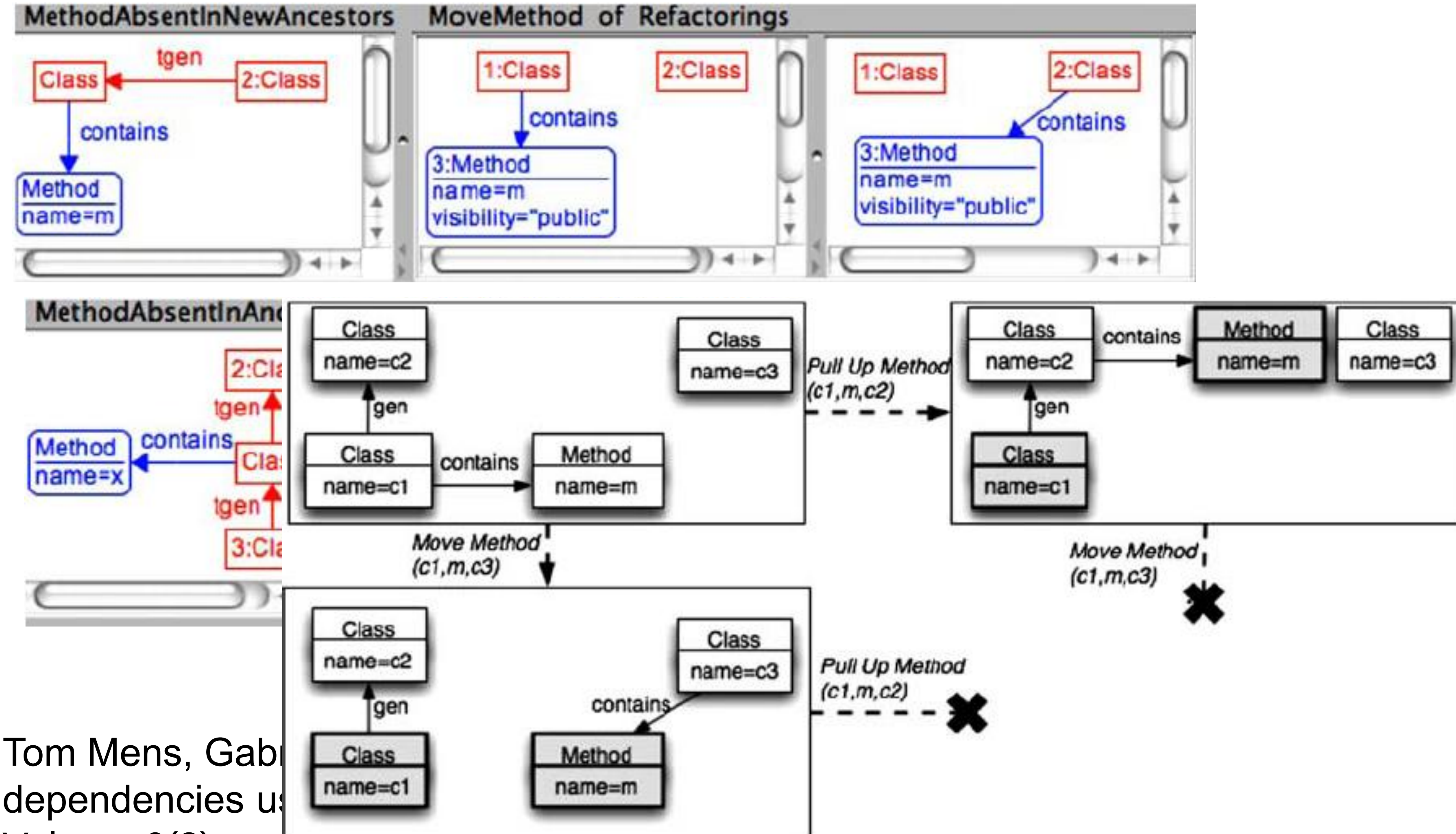
rule pull-up property



Application condition: p1.name=p2.name

Applications

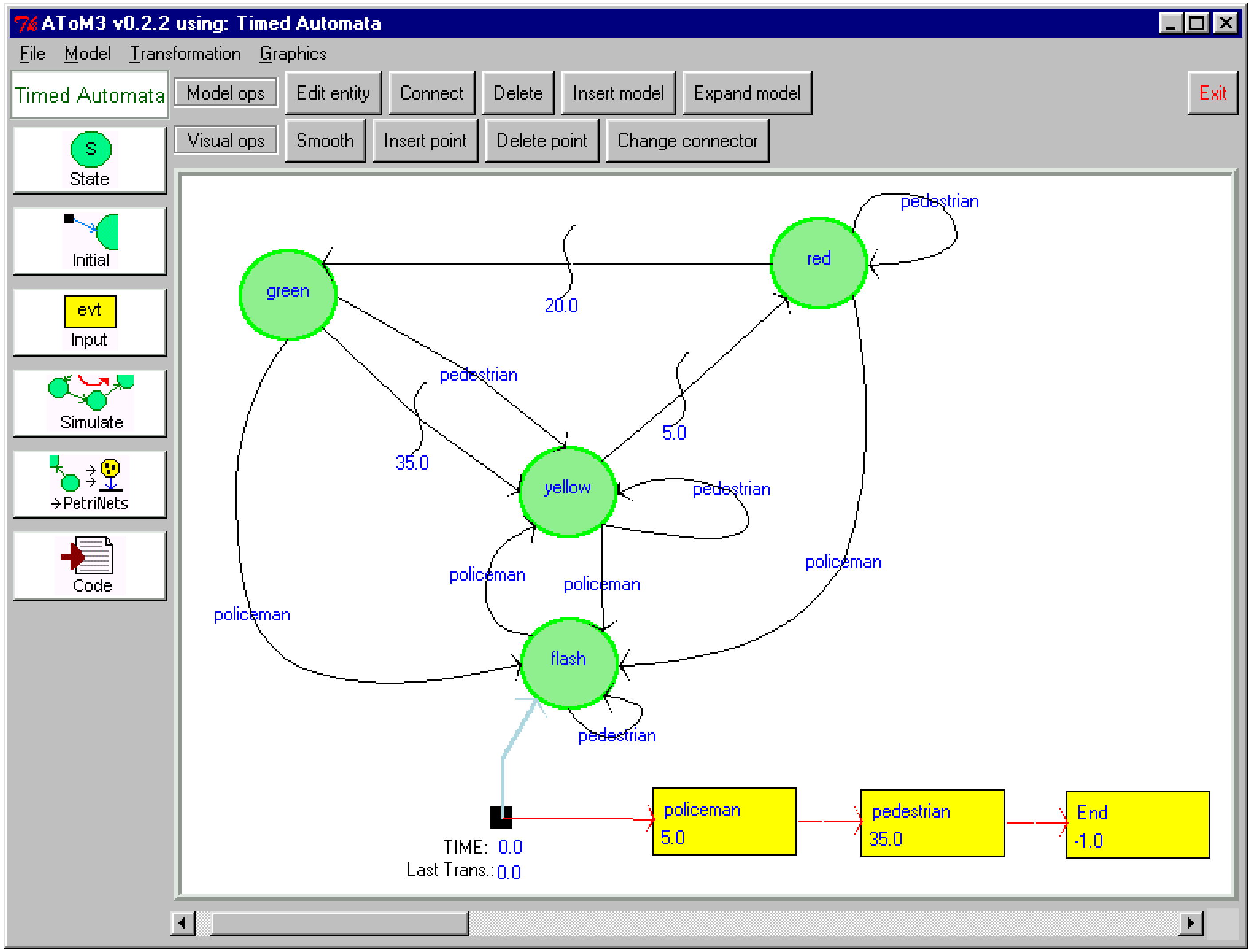
Analysis of Refactoring.



Tom Mens, Gabr
dependencies us
Volume 6(3), pp.. 209—283, Springer.

Simulation

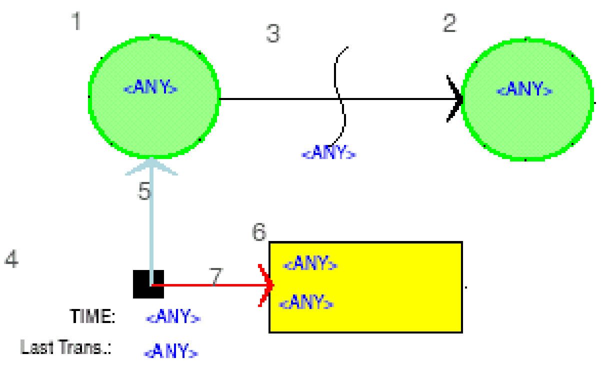
AToM³



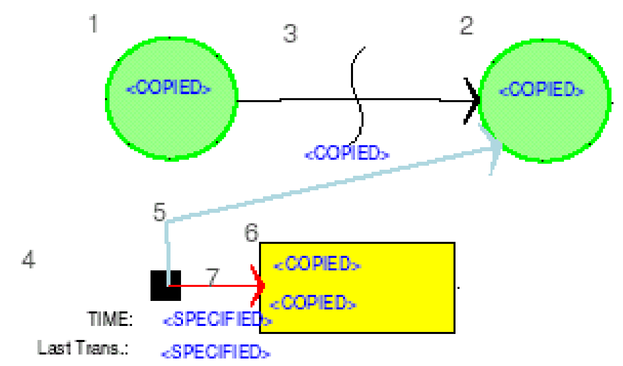
Simulation

AToM^3

TA_MoveTimeTrans Priority 1



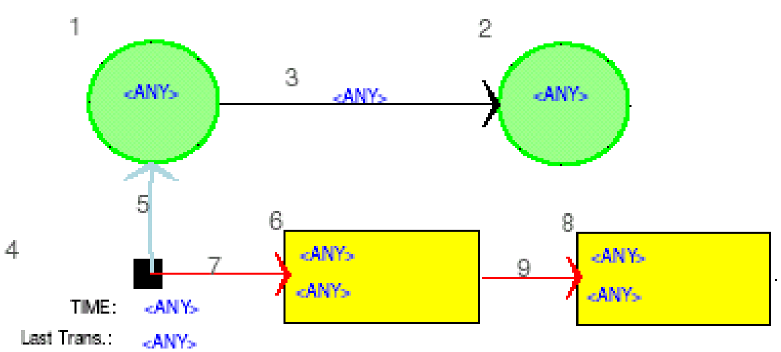
::=



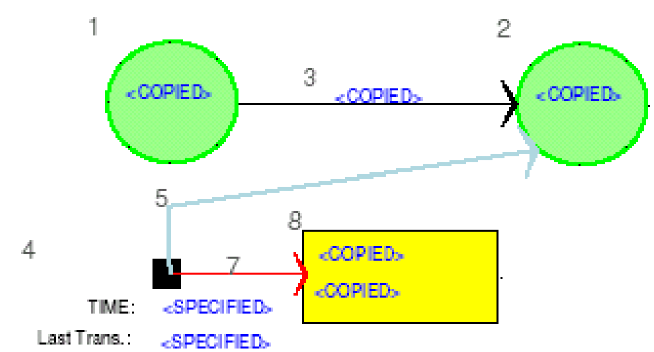
CONDITION: node(6).ArrivalTime > node(4).Time + node(3).Time

TIME += node(3).Time
Last Trans = TIME

TA_MoveTrans Priority 3



::=

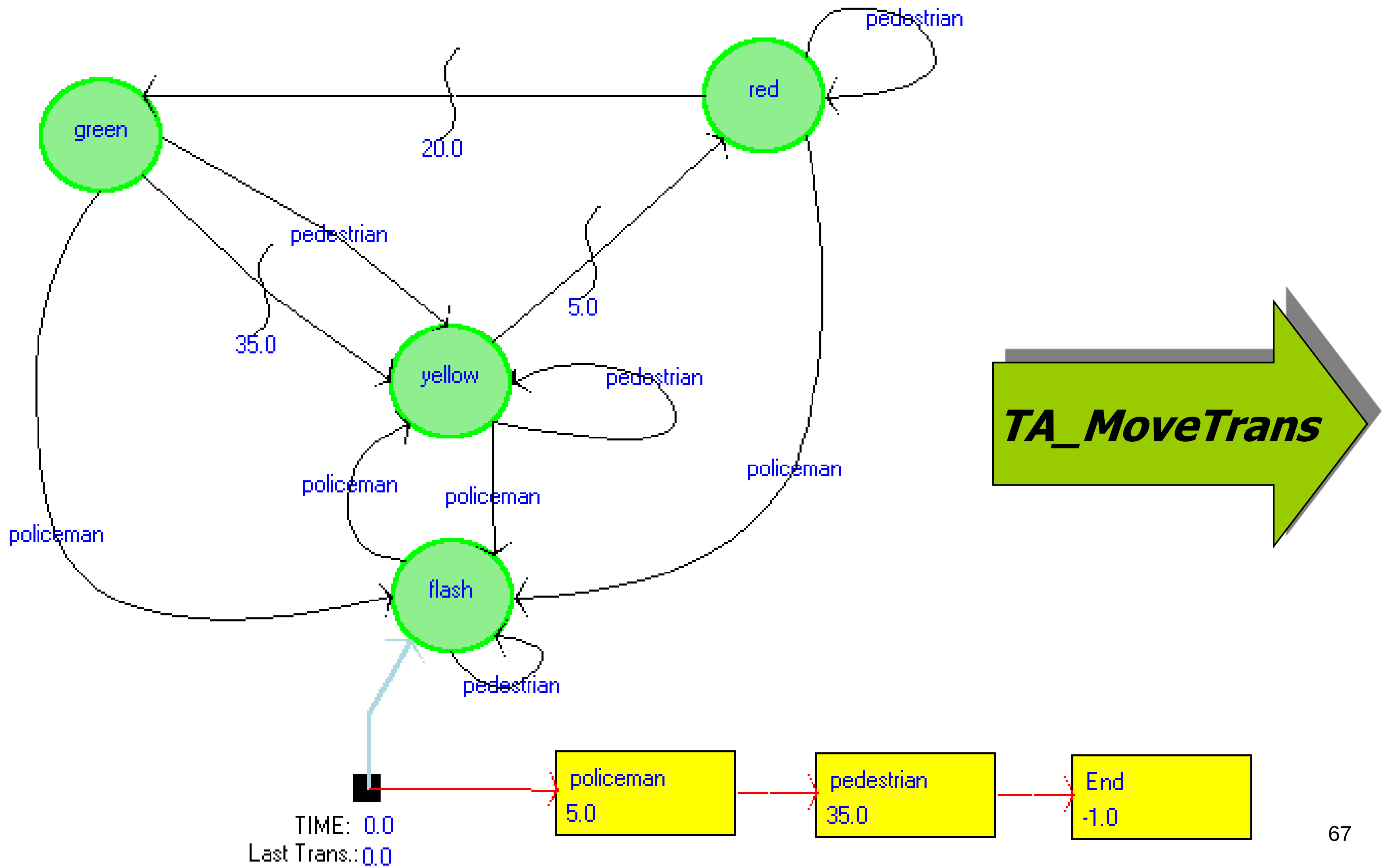


CONDITION: node(6).Value == node(3).Value

TIME = node(6).ArrivalTime
Last Trans = TIME

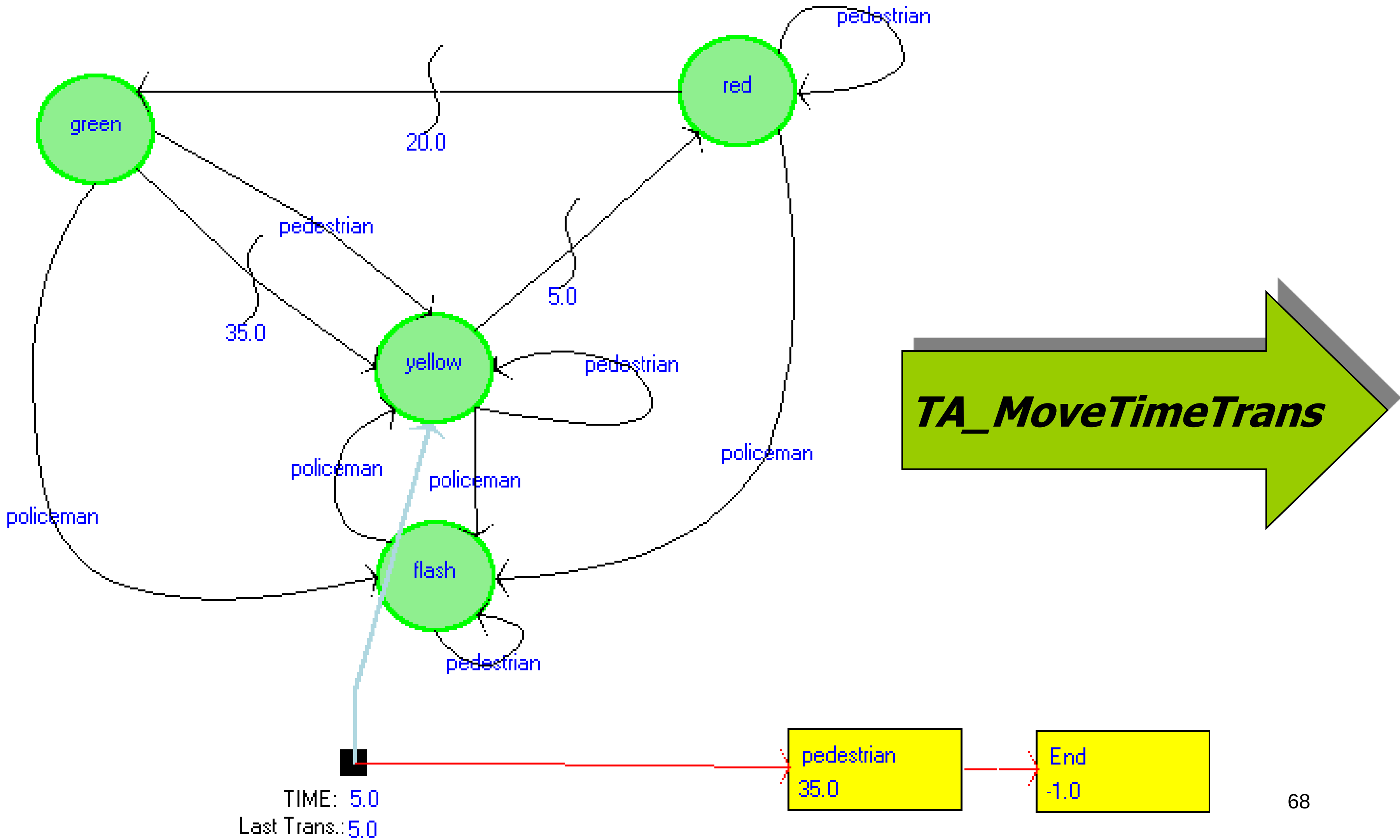
Simulation

AToM³



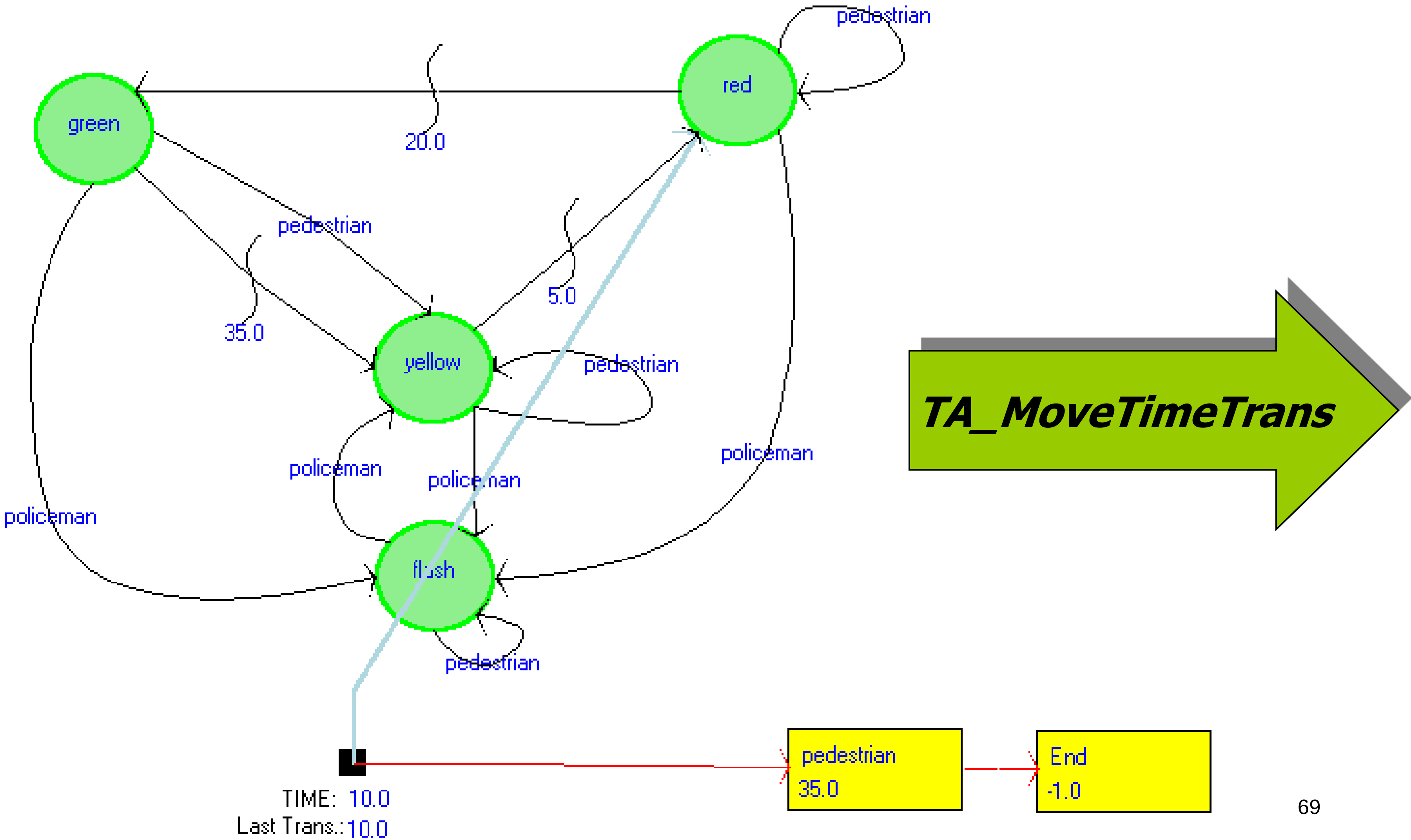
Simulation

AToM³



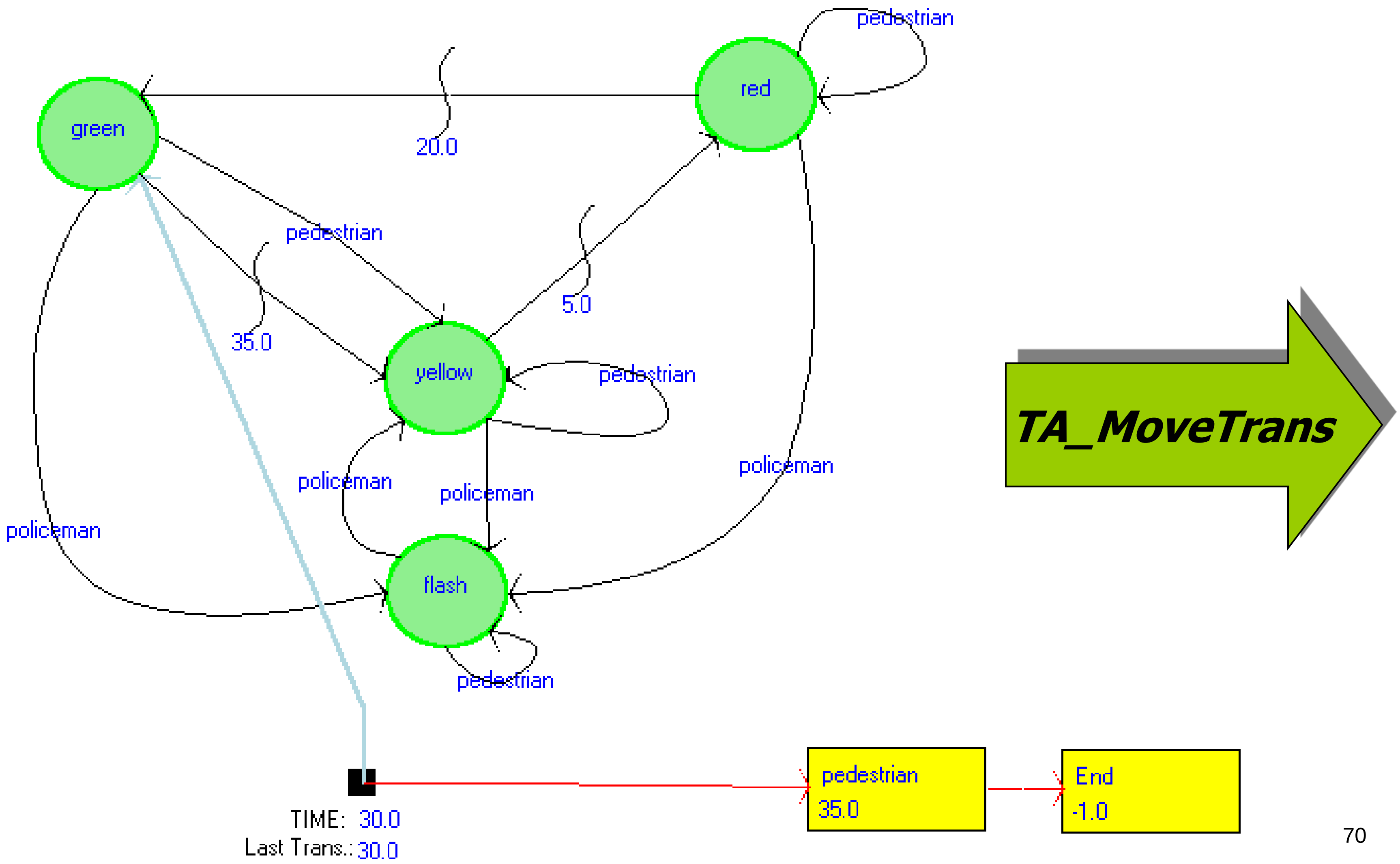
Simulation

ATOM³



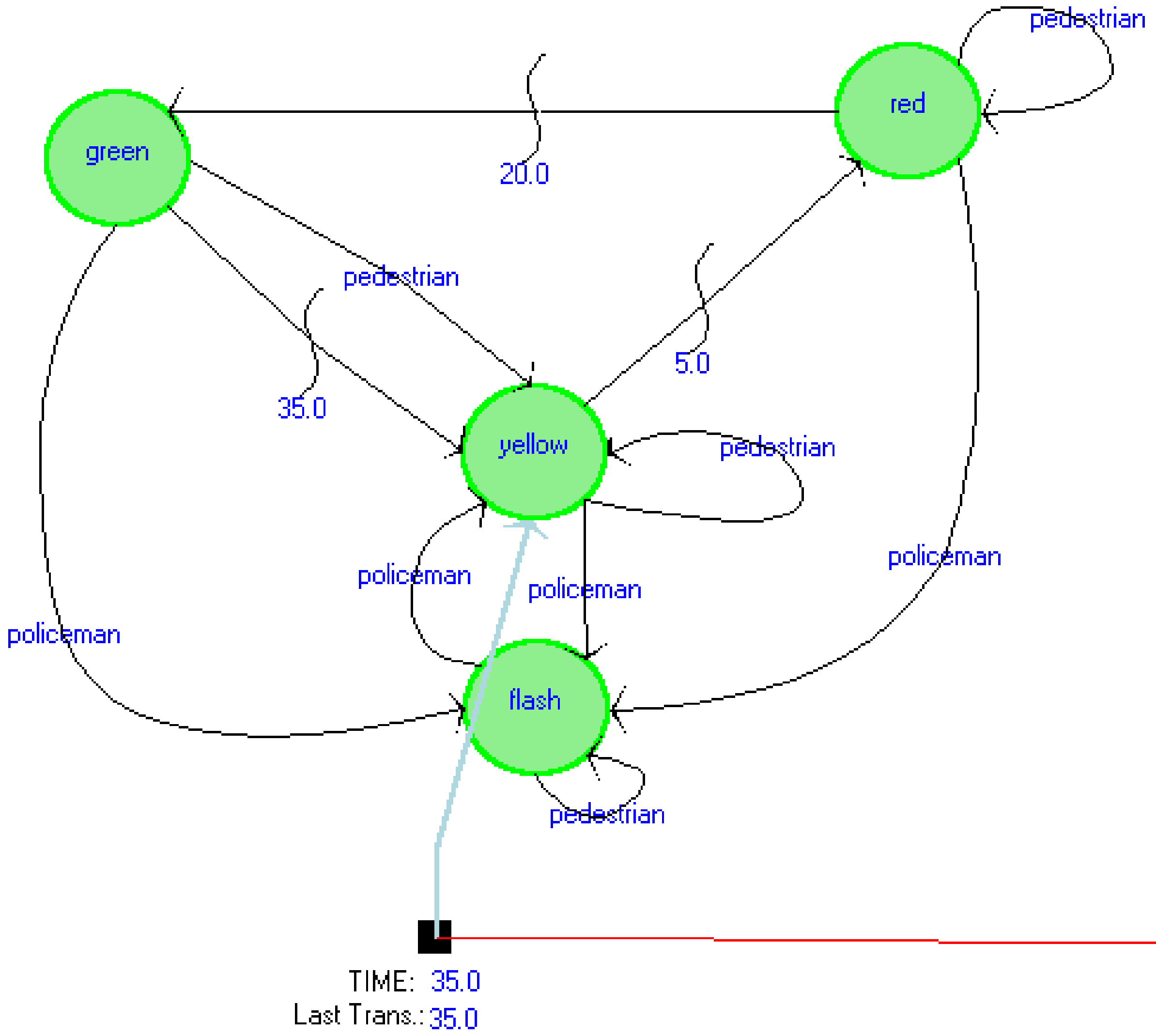
Simulation

AToM³



Simulation

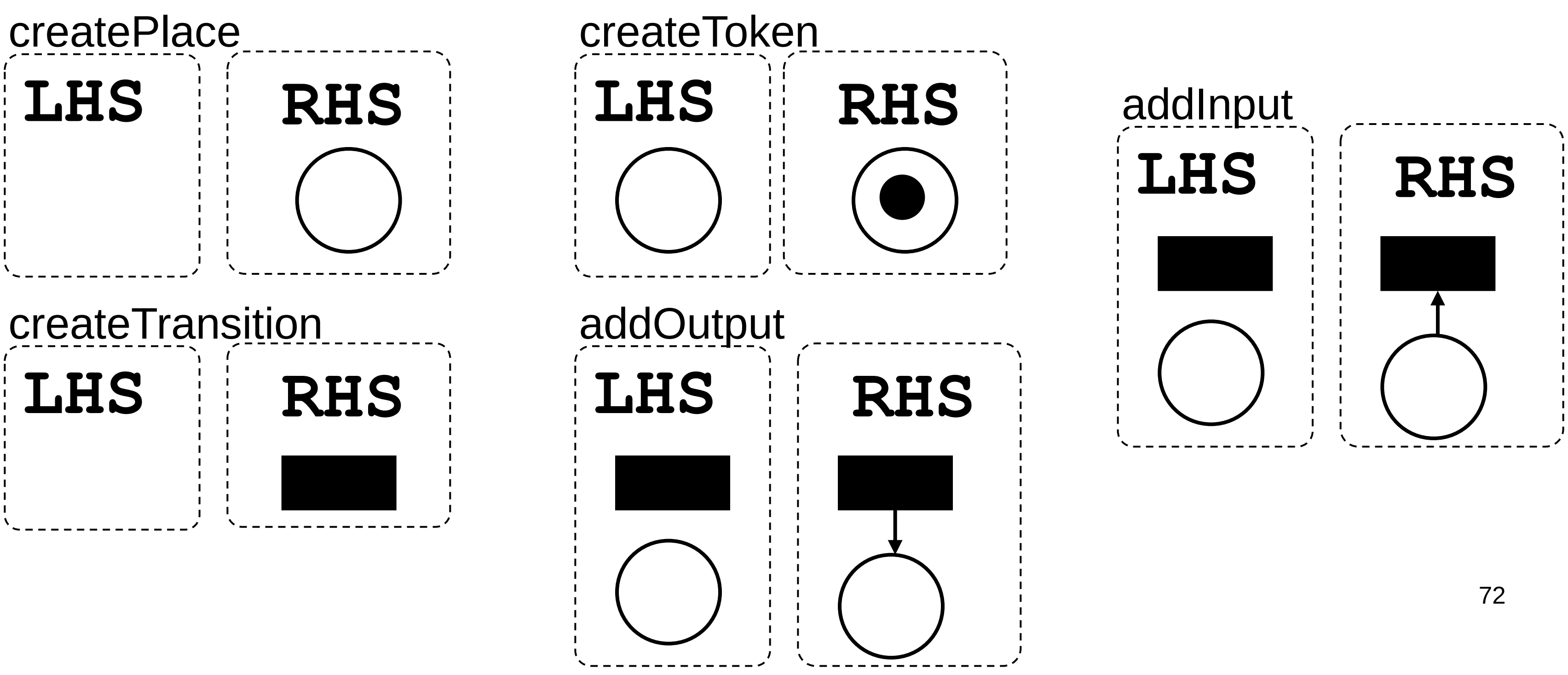
AToM³



TA_MoveTrans
...

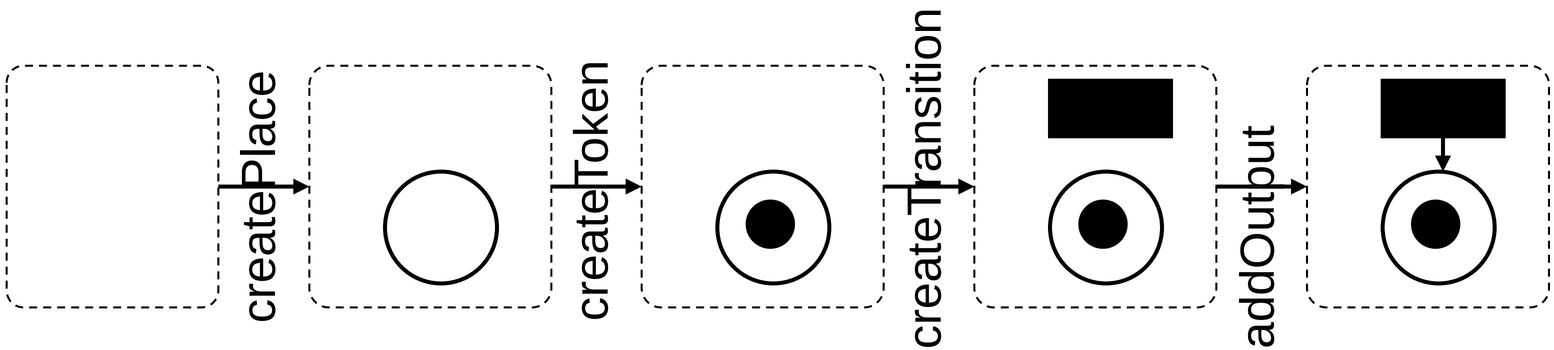
Generation of Languages

- Grammars can also be used as an alternative to meta-models to describe modelling languages.
  - Similar to programming language definition through Chomsky grammars.
- Example: Defining the language of Petri nets.



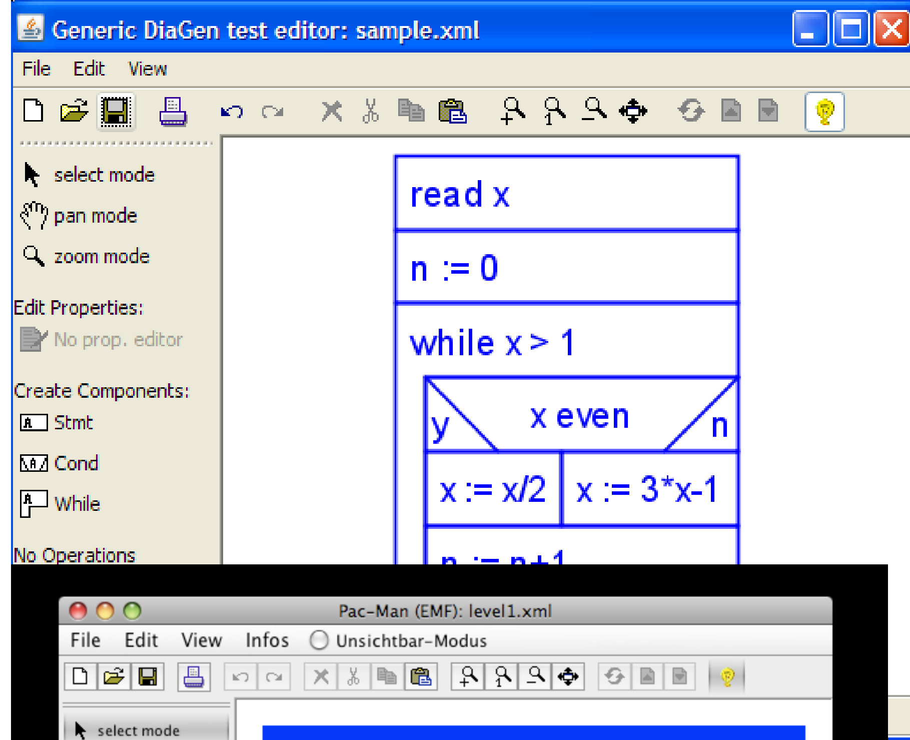
Generation of Languages

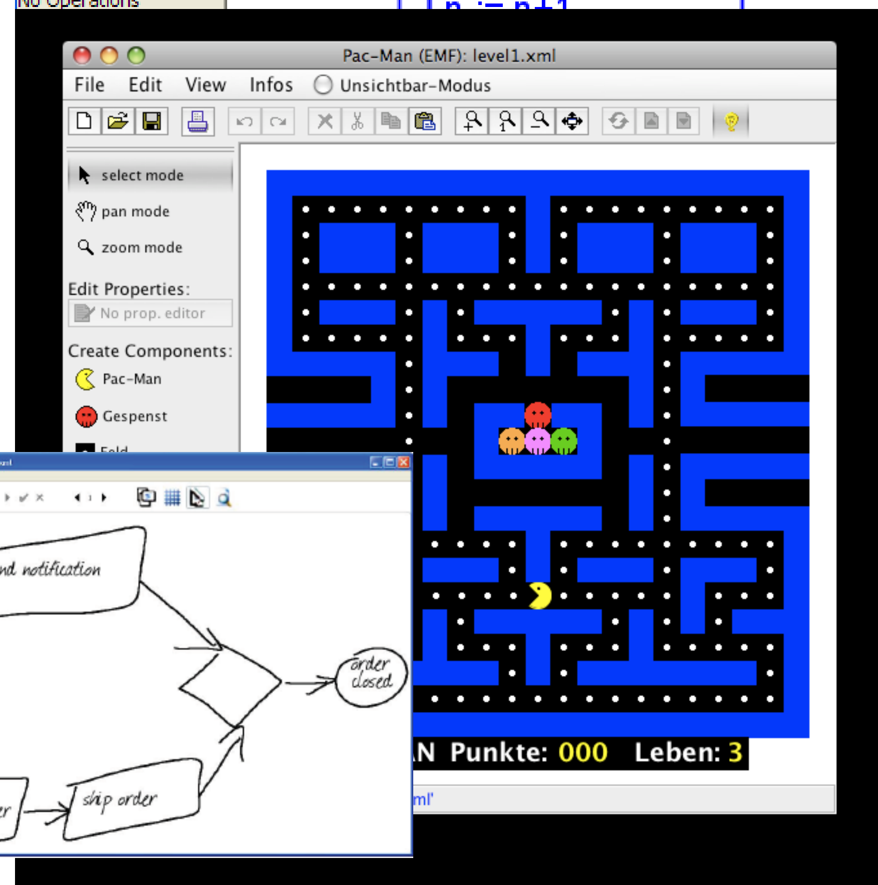
- Derivation example:

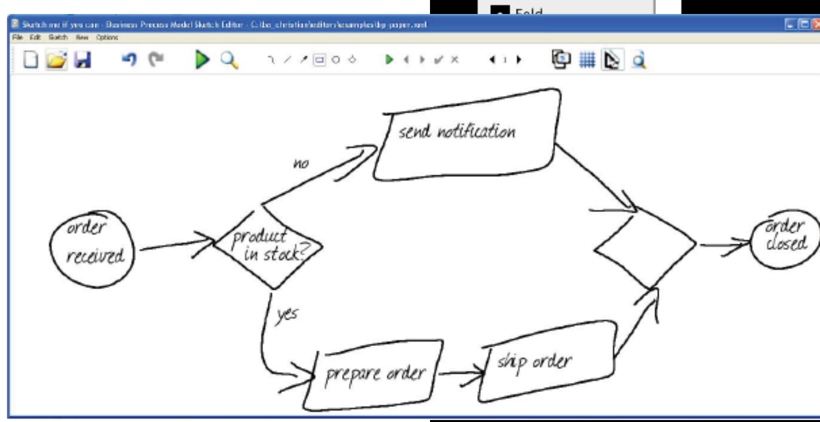


- The semantics of the grammar SEM(PN) are all possible reachable models: the language of Petri nets.
- For recognition, one uses parsing, applying rules backwards trying to reduce the model to the empty graph.
- Also useful to generate random, synthetic models (e.g., for testing model transformations, code generators, etc)

- DiaGen/DiaMeta. (https://www.unibw.de/inf2/diagen/diagen-diameta). First version in 1993.
- Based on hypergraph grammars.
- Sketching.
- Mark Minas (Munich).



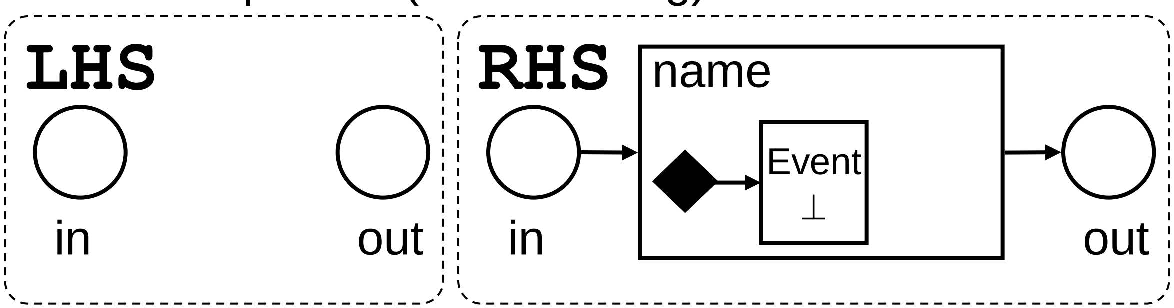




Complex Editing Commads

- Express by means of rules complex editing commands.
- The match in which the rule has to be applied can be partially given by the user by selecting elements in the model.
- Assign the execution of such rule to a button in the GUI.

createComponent (name: String)

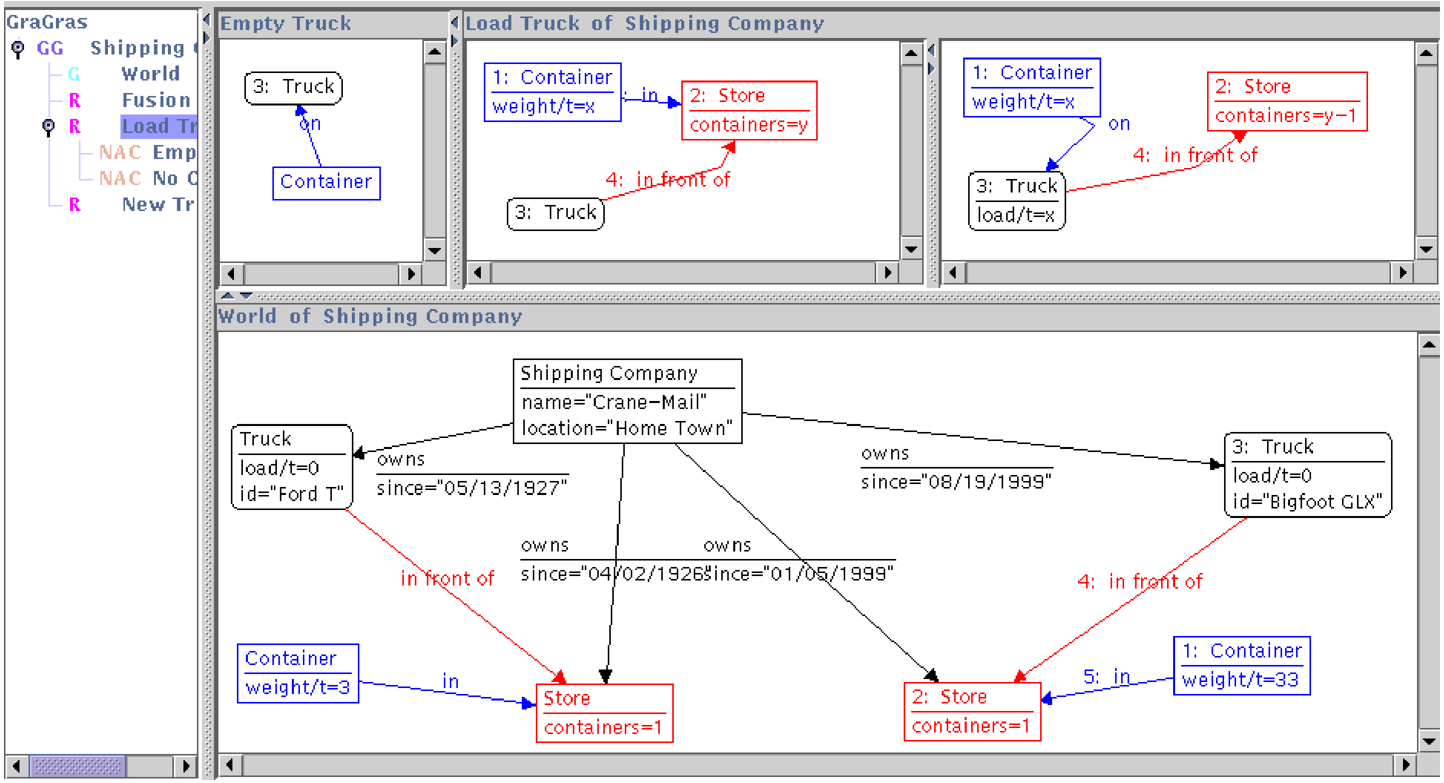


Index

- Introduction.
- Overview of Graph Transformation.
- Exercise: Playing tic-tac-toe.
- Control Language.
- Applications.
- **Tools.** 
- Conclusions.

AGG

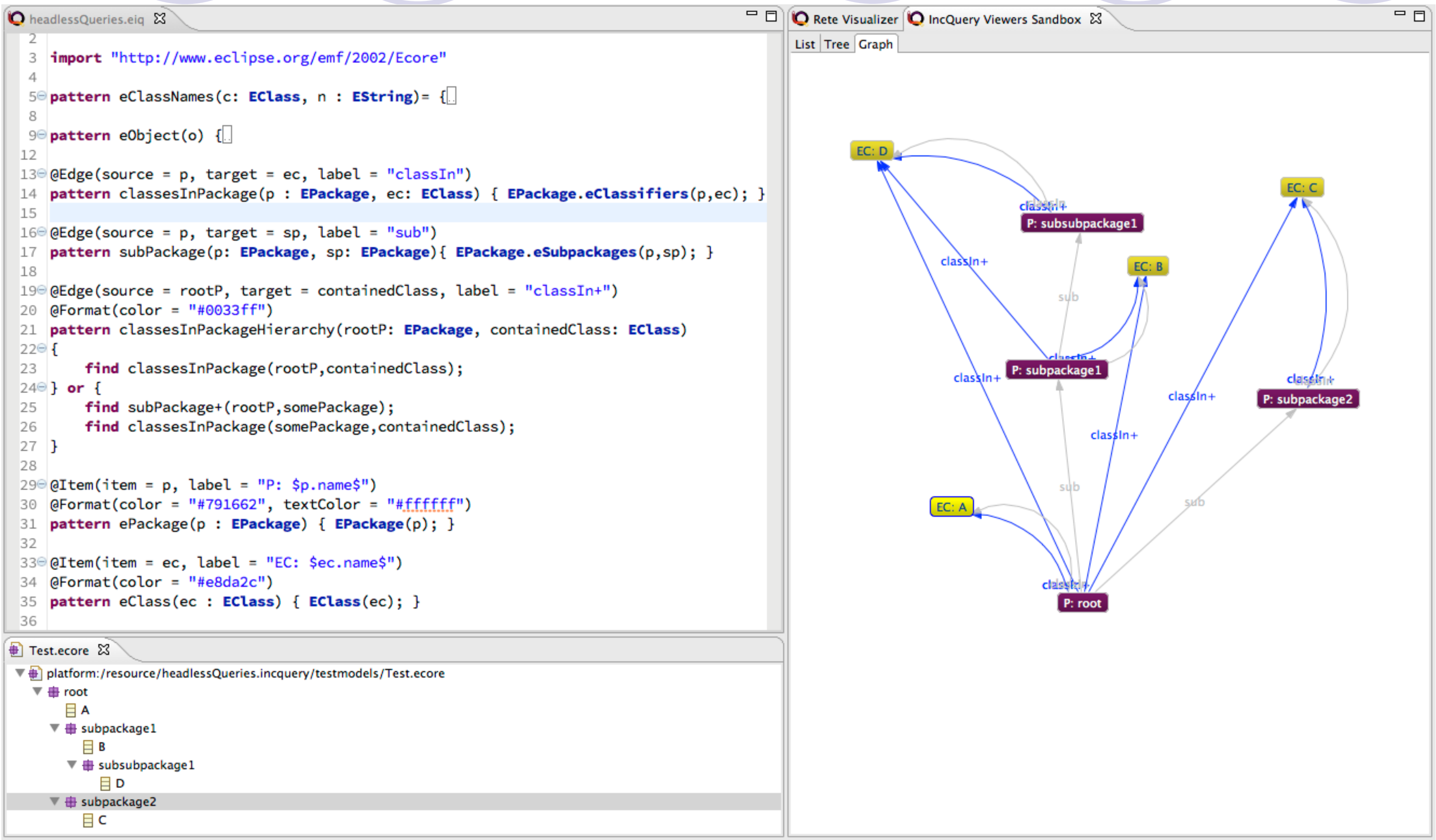
http://tfs.cs.tu-berlin.de/agg/



- Definition and execution of grammars.
- Type Graph.
- Analysis (Critical Pairs – confluence)

VIATRA

https://www.eclipse.org/viatra/



•Definition and execution of queries and (reactive) transformations.

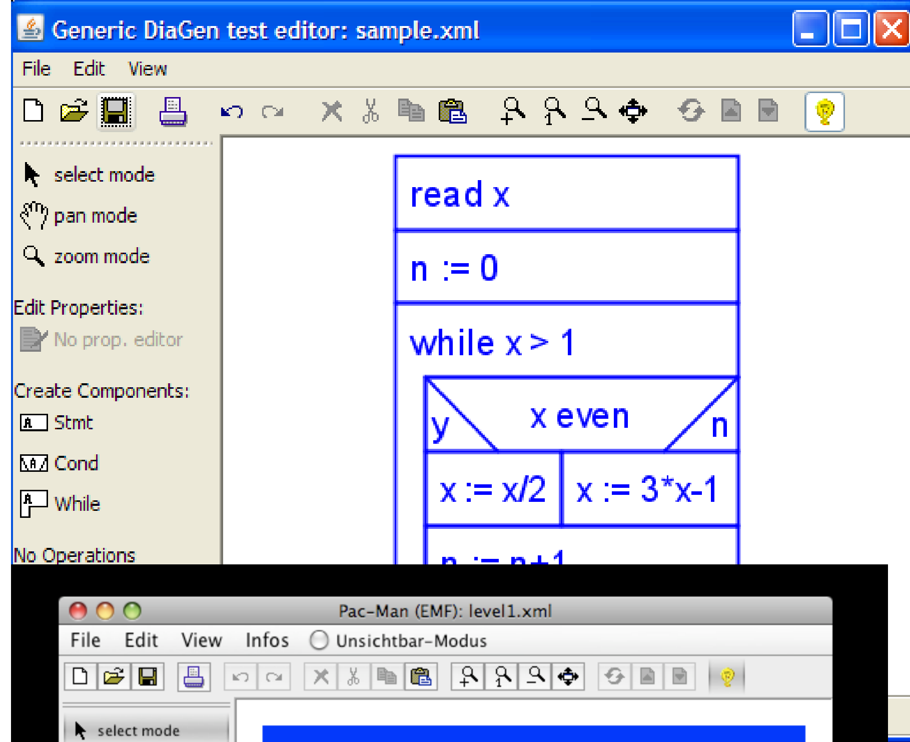
•Models and meta-models

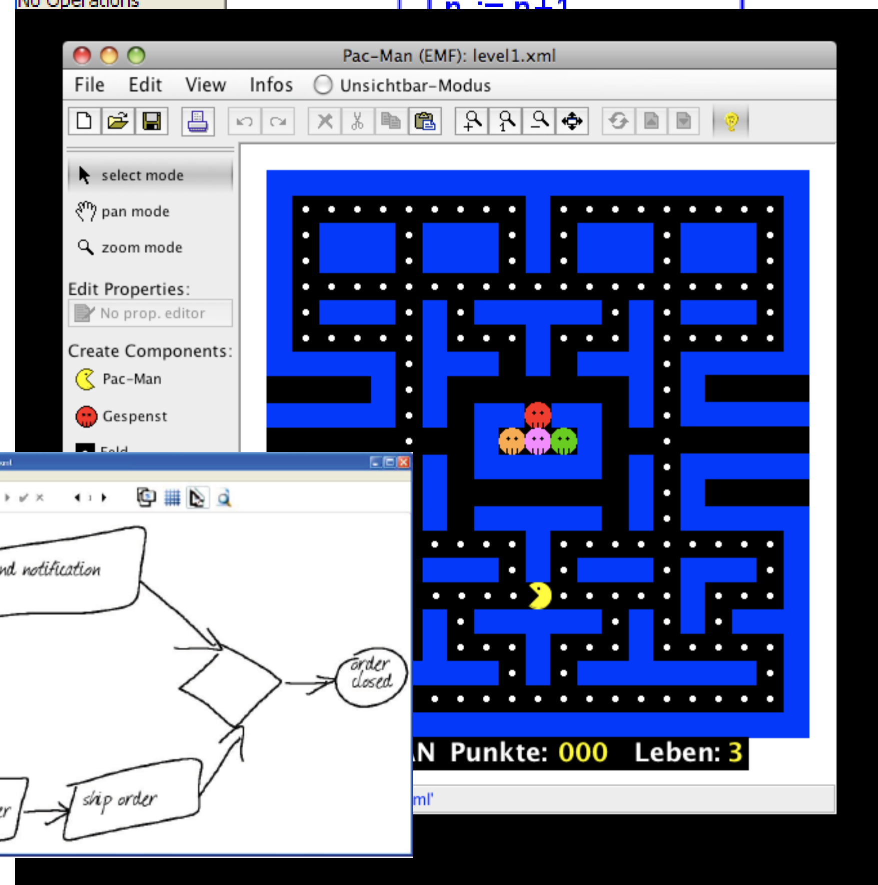
•Scalability

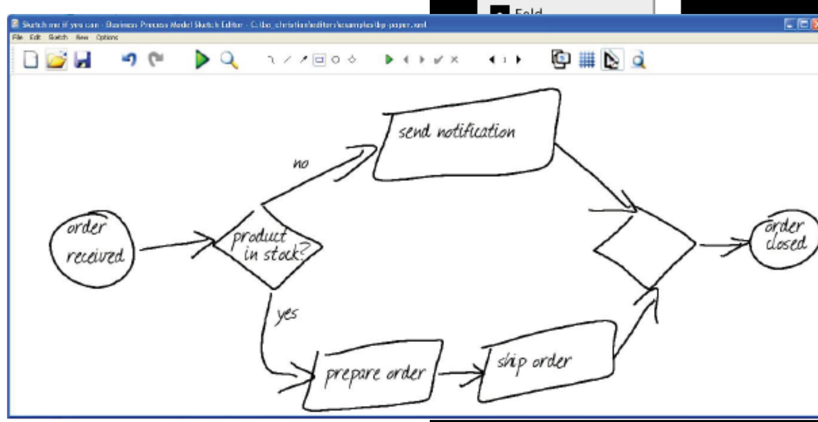
Tools

- DiaGen/DiaMeta. (https://www.unibw.de/inf2/diagen/diagen-diameta). First version in 1993.
- Based on hypergraph grammars.
- Sketching.
- Mark Minas

(Munich).

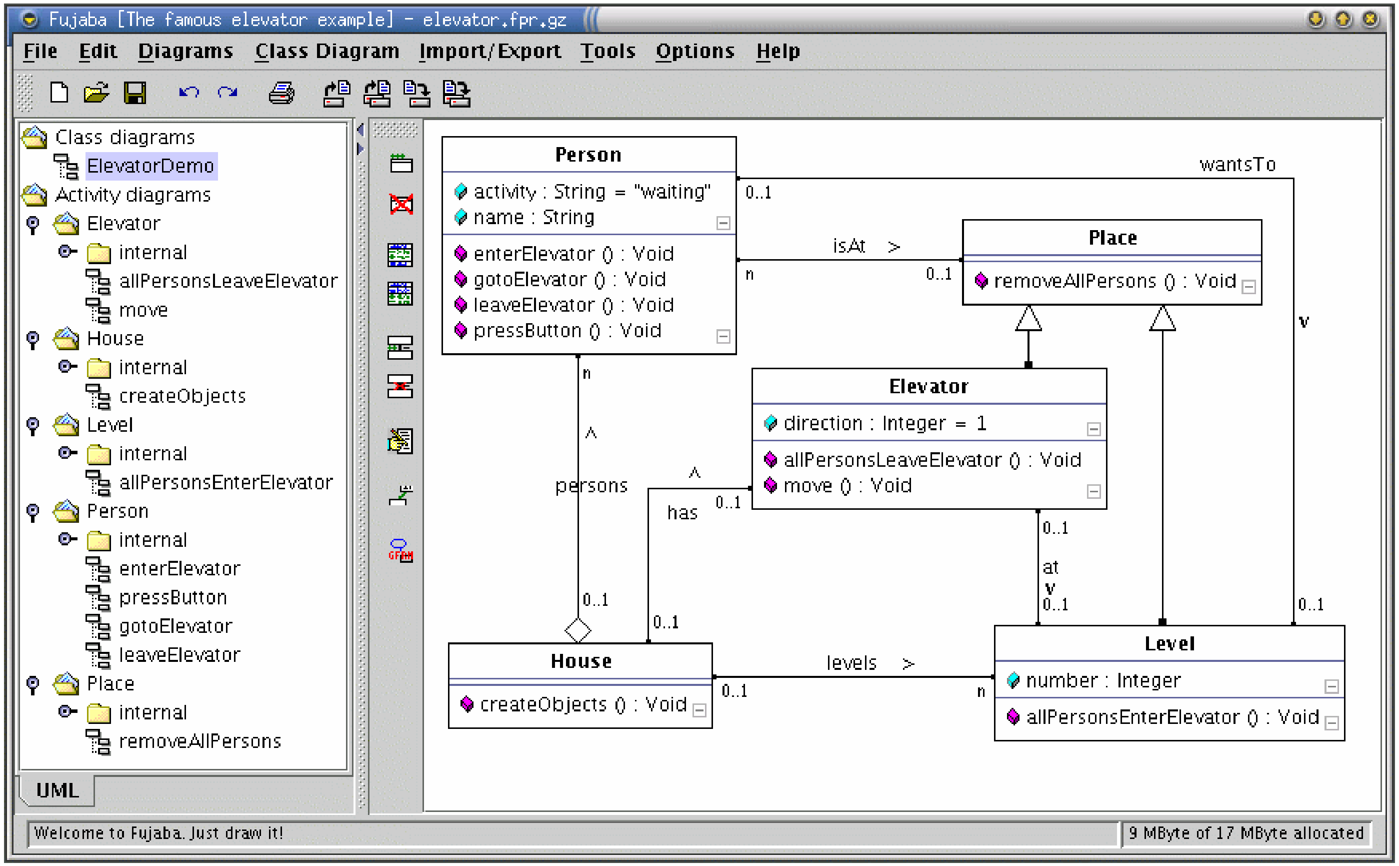






FUJABA

https://github.com/fujaba

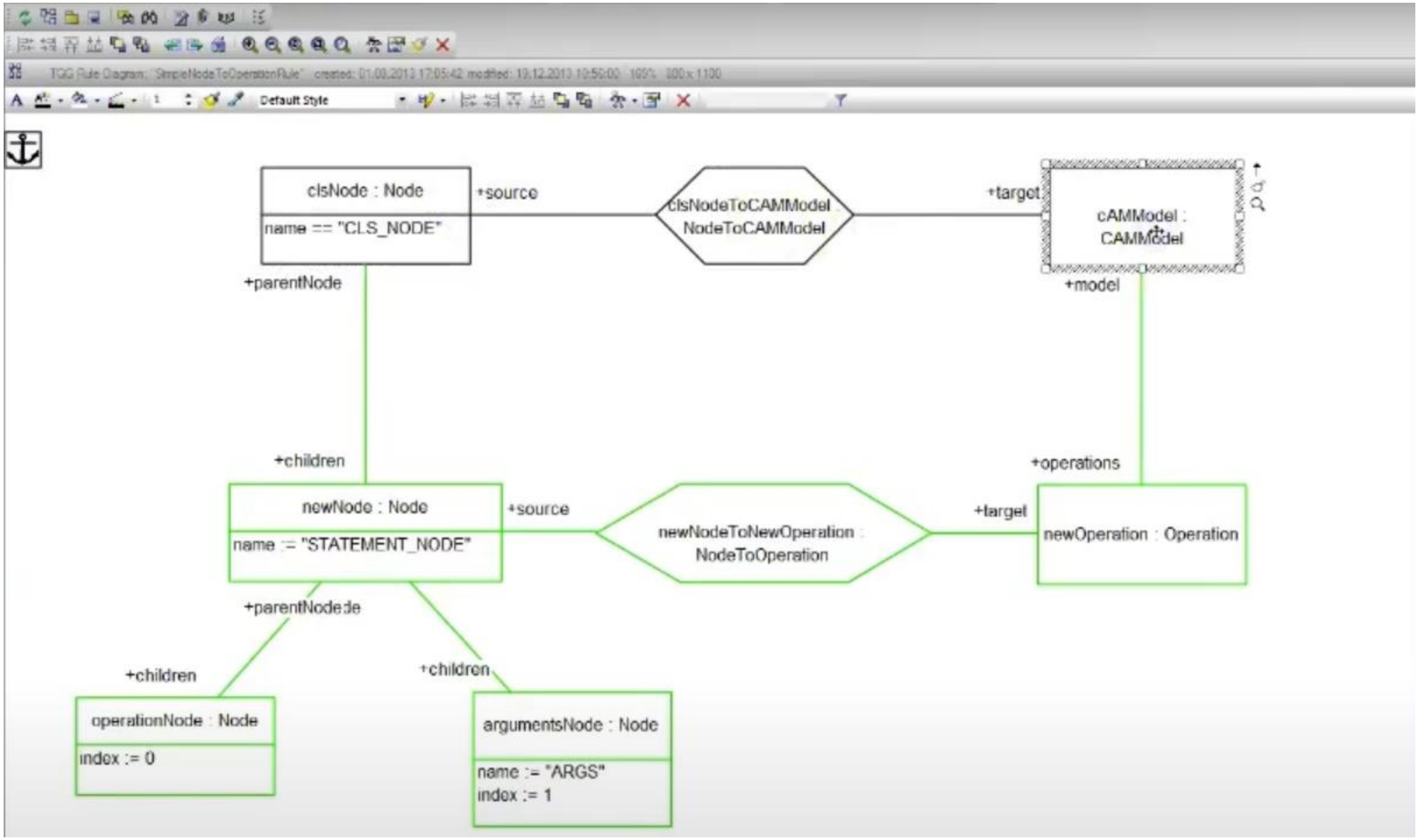


•Story diagrams (Activity diagrams+rules)

•From UML to Java and back.

eMOFLON

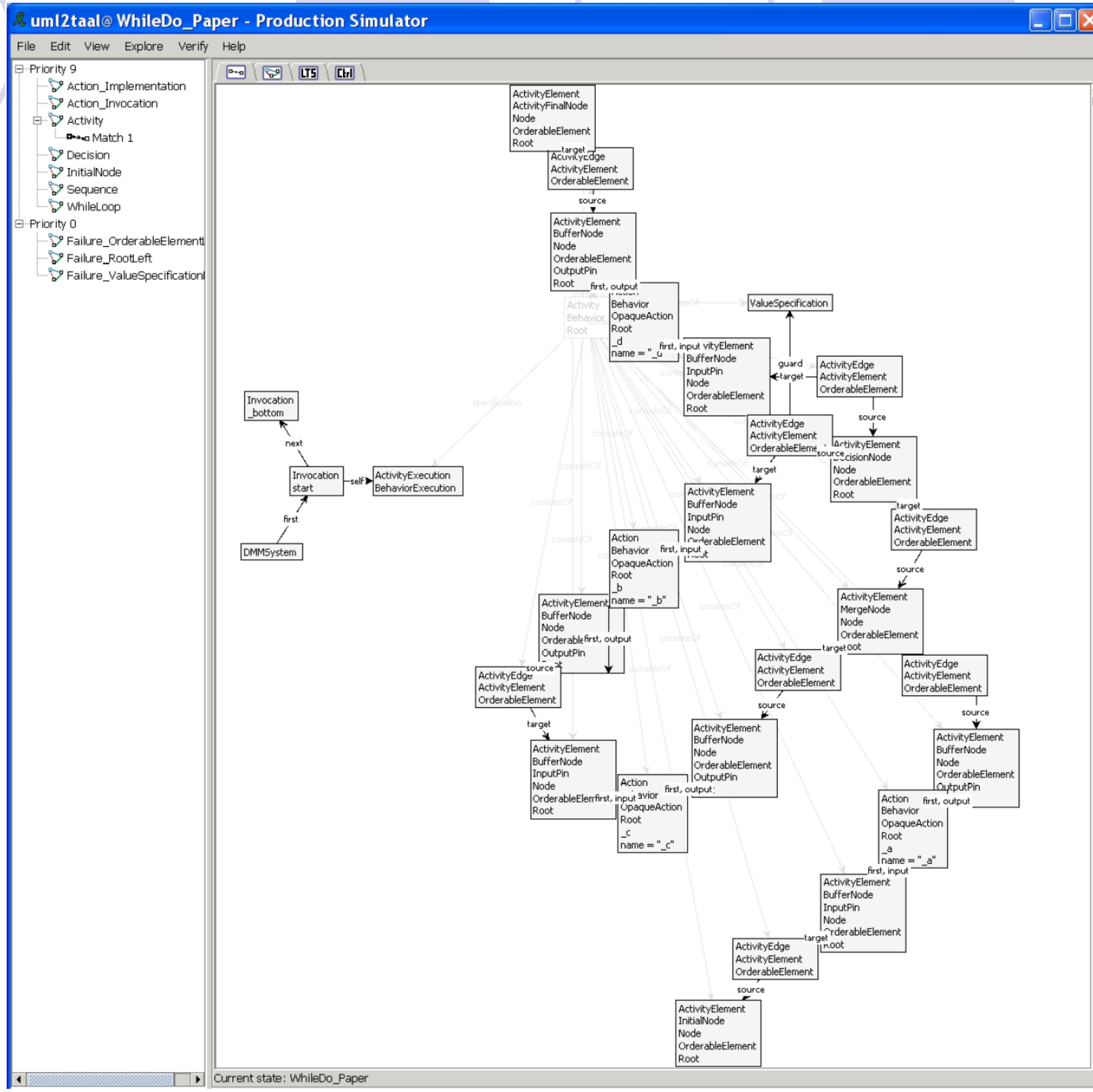
https://emoflon.org/



- Rules for Model to Model transformation (Triple graph grammars)
- Within Enterprise Architect

GROOVE

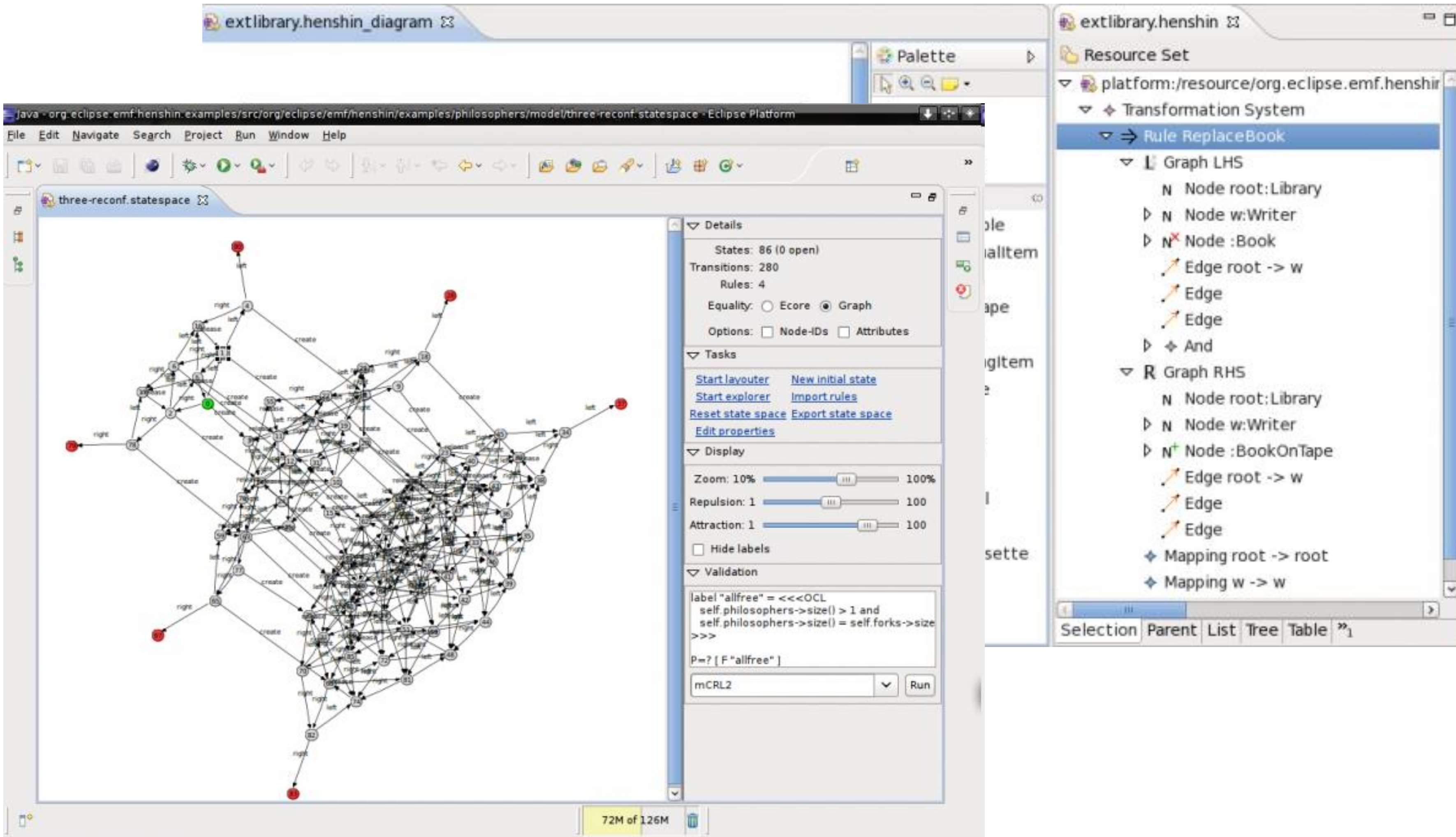
http://groove.cs.utwente.nl/



- Quantified Rules.
- Model Checking.

Henshin

http://www.eclipse.org/modeling/emft/henshin/



- State Space Analysis for Verification.

Index

- Introduction.
- Overview of Graph Transformation.
- Exercise: Playing tic-tac-toe.
- Control Language.
- Applications.
- Tools.
- **Conclusions.**

Conclusions

- An overview of Graph Transformation.
  - Rule, application conditions.
  - Grammar, derivation.
  - Dangling edges, non-injectivity.
  - Control languages for rule execution.
- Based on pre- and post-conditions.
- Use the language of the DSVL.

Remaining Issues

- Some words on formalizations.
  - Analysis.
- Specifying invariants:
  - Graph constraints and application conditions.
- Use of grammars to specify model-to-model transformations.

Bibliography.

- Handbook of Graph Grammars and Computing by Graph Transformation. 3 Vols. 1997. World Scientific.

- Ehrig, H., Ehrig, K., Prange, U., Taentzer, G. 2006. “Fundamentals of Algebraic Graph Transformation”.Springer.