MGS 2024

GT4SE
Graph Transformation
for Software Engineers

Foundations and Applications of Graph Rewriting

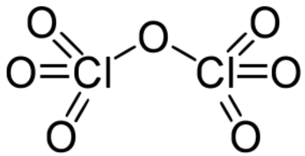
Reiko Heckel

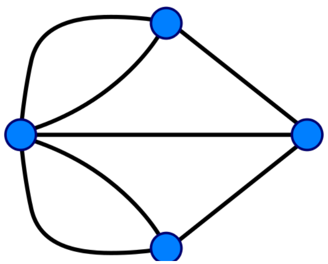
University of Leicester, UK

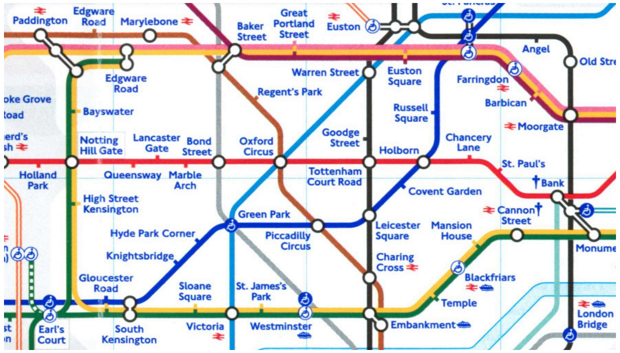
Partly based on book with Gabriele Taentzer

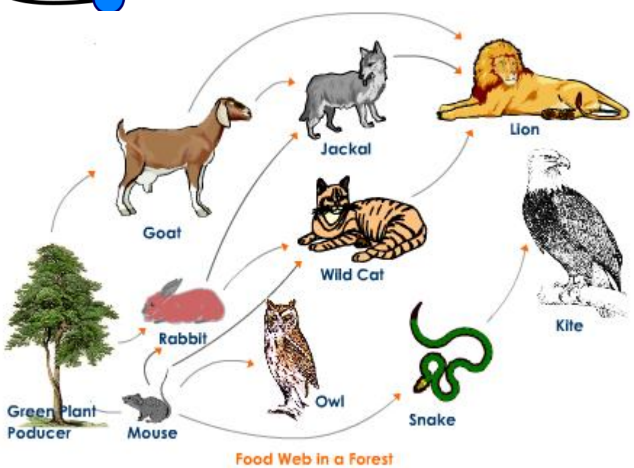
Philipps Universität Marburg, Germany

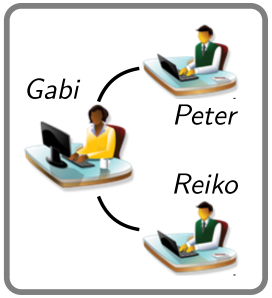
Graphs are everywhere

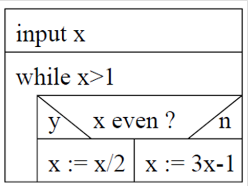


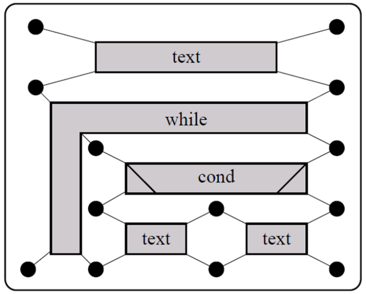


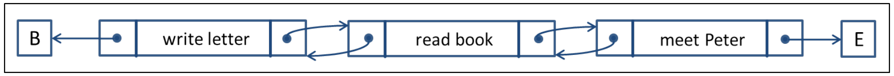


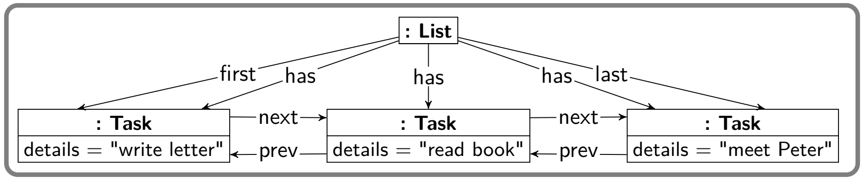


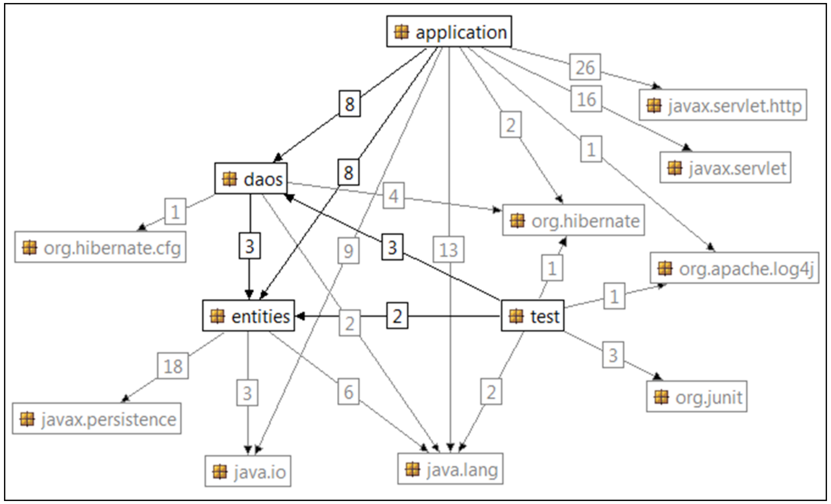












Graphs and Graph Transformations

Graphs represent

- *Class and object structures*
- *Processes and architectures*
- *Diagrams,3D objects*

formally, *visually* and *directly* (little need for encoding)

Graph transformations model

- *Behaviour and reconfiguration*
- *Creation, manipulation, evolution*
- *Operational semantics*

in an *intuitive* rule-based way, with *formal semantics* for *execution* and *analysis*

Outline

Foundations

- Graphs
- Graph Transformations
- Semantics and Control
- Analysis Techniques

Two Types of Applications

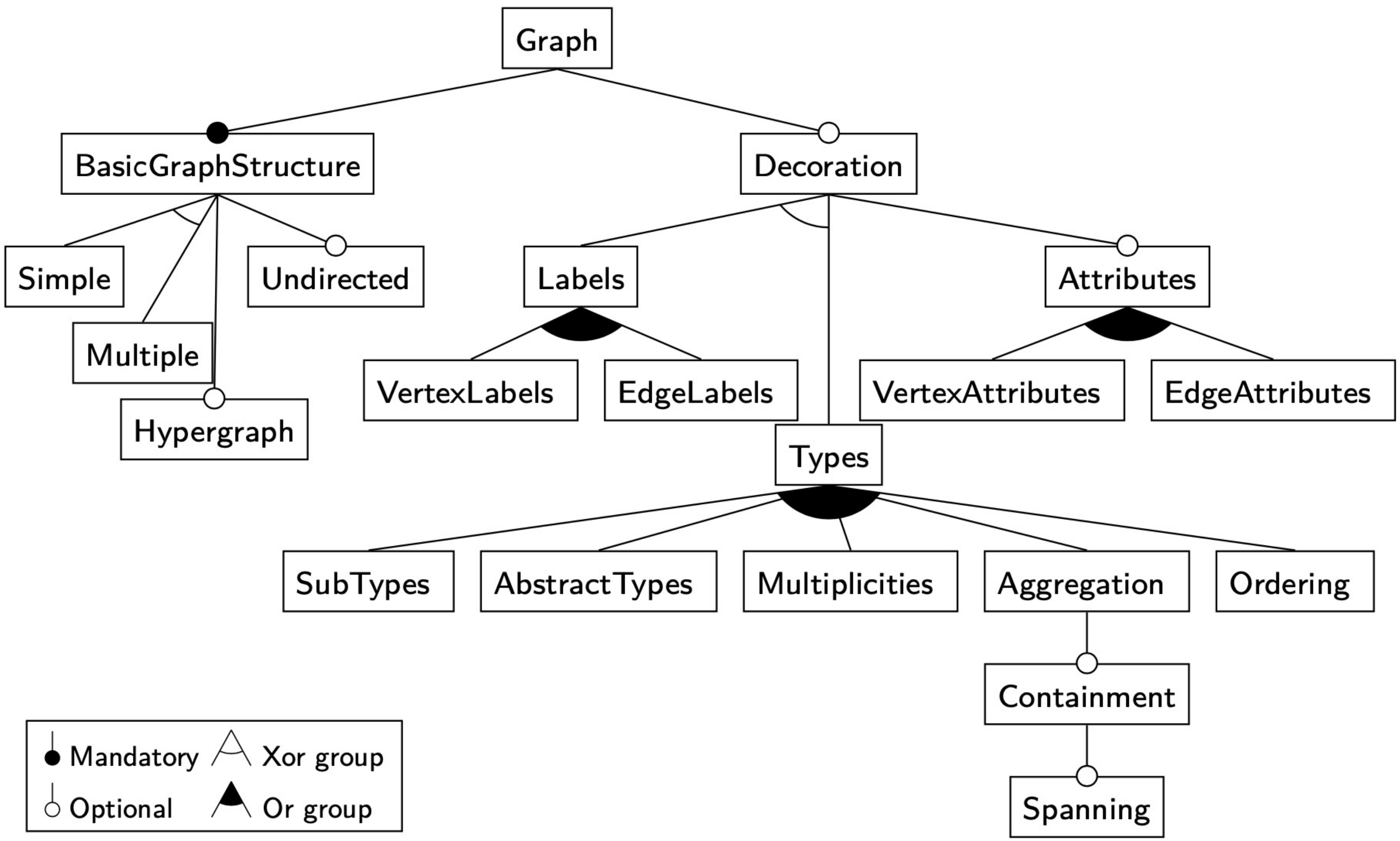
Graph Transformation-Based

- Software Engineering
  ➔ *GTBSE*
- Language Engineering
  ➔ *GTBLE*

Part 1: Foundations

- Graphs
- Graph Transformations
- Semantics and Control
- Analysis

Variants of graphs

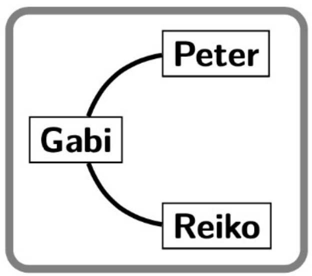


GTBSE or GTBLE?

Directed or undirected
Labelled or typed

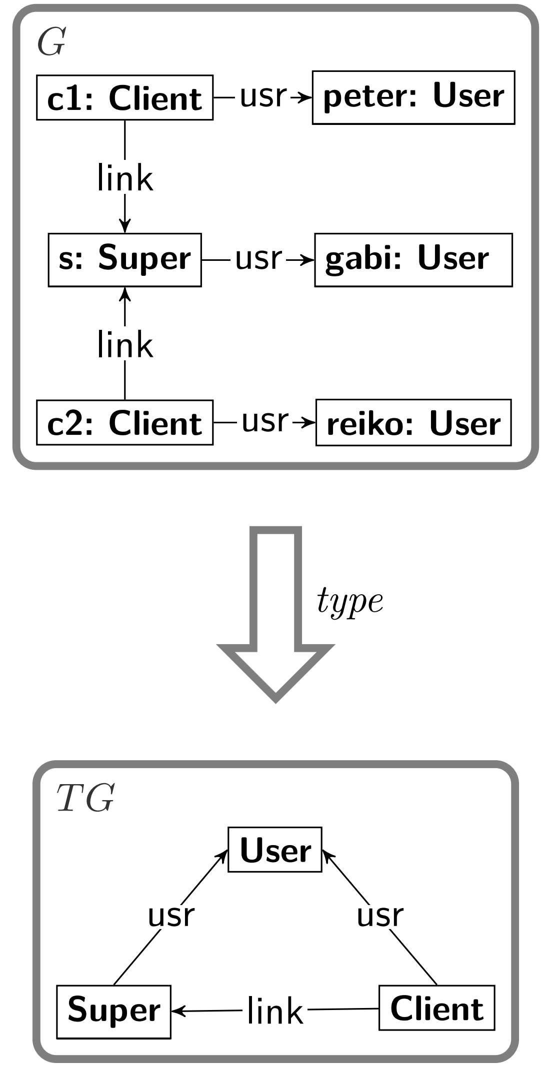
Labelled undirected simple graph

- Nodes set *V*, Edges $E \subseteq \boldsymbol{P}_2(V)$
- Labels *L* = *{Peter, Gabi, Reiko}*
- $lab_V: V \rightarrow L$

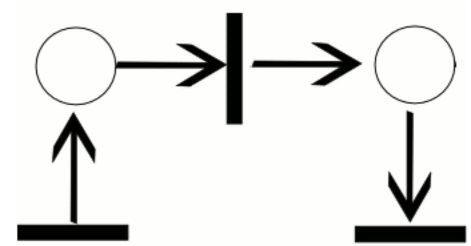


Directed typed graph

- Node, edge labels form type graph *TG*
- Instance graph *G* with homomorphism *type* preserving graph structure

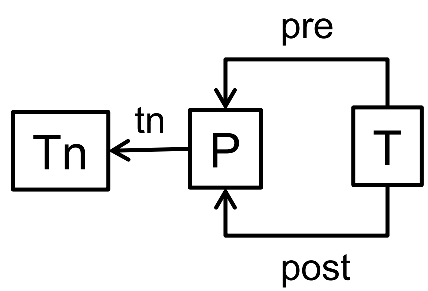


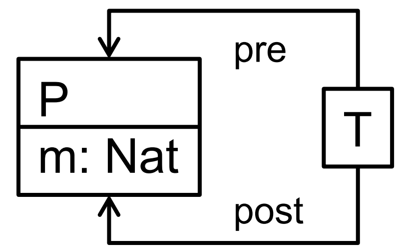
GTBLE: Petri nets



Create

- *Type graph for Petri nets*
- *Instance graph for the example*



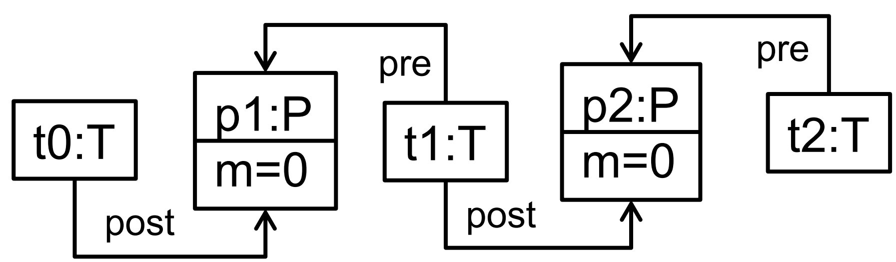


Design decisions

- *Bipartite (or unordered hyper) graphs as typed graphs*
- *Node type Tn to represent tokens (left) vs attribute m for marking (right)*

Marked graphs (right)

- *plus constraints for singleton pre and post sets*

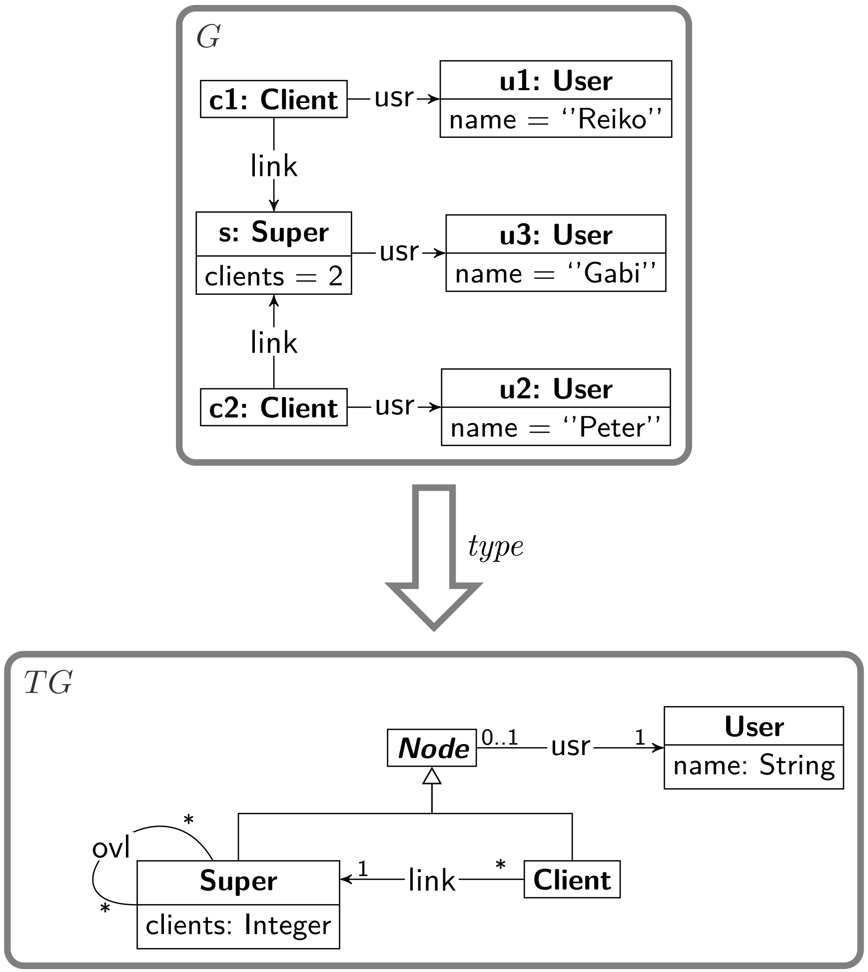


Typed attributed graphs with subtyping

Structured P2P model

- *Client* and *Super Nodes* serve *Users*
- *Super* nodes form overlay network, support limited number of *Clients*

➔ GTBSE



Formalise this, …

Type, instance graphs

- *Graphs and graph homomorphisms*
- *Slice category*

Attributed graphs

- *Attributes for a fixed data algebra*
- *Symbolic attributed graphs*

GT4SE
Graph Transformation
for Software Engineers

Part 1: Foundations

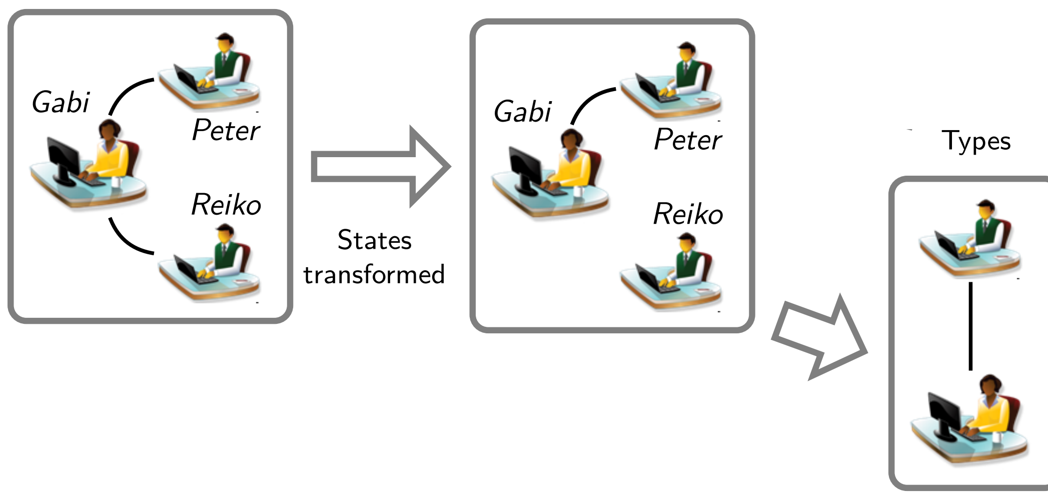
- ✓ Graphs
- Graph Transformations
  - *Basic rules and transformations*
  - *Global application conditions*
  - *Advanced features*
- Semantics and Control
- Analysis

Rules generalize transformations

Specify changes by

- Difference between pre and post graphs
  - ➔ *Deleted: L \ R*
  - ➔ *Created: R \ L*
- Context required for the changes to happen
  - ➔ *Preserved: L ∩ R*

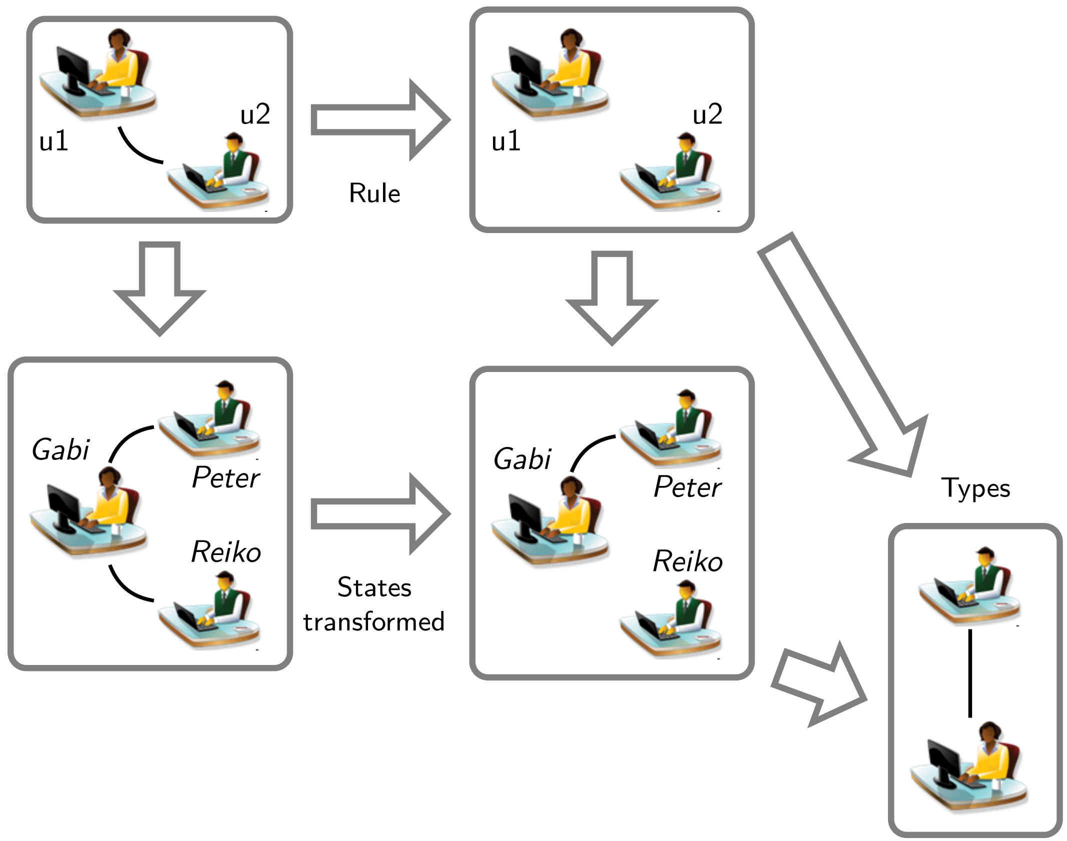
Rule should describe
- *What changes, and how?*
- *In which situations?*



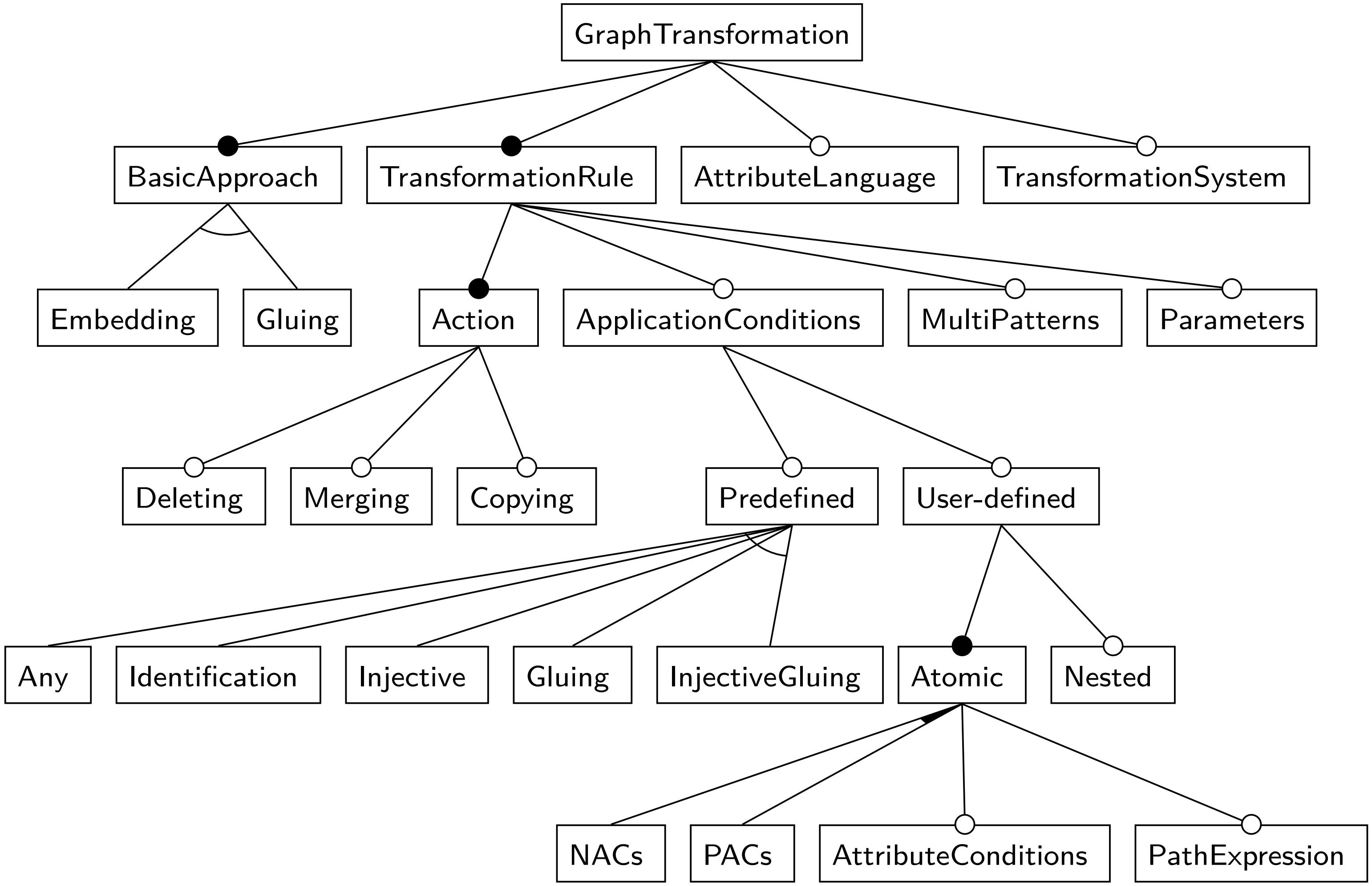
Rules generalize transformations

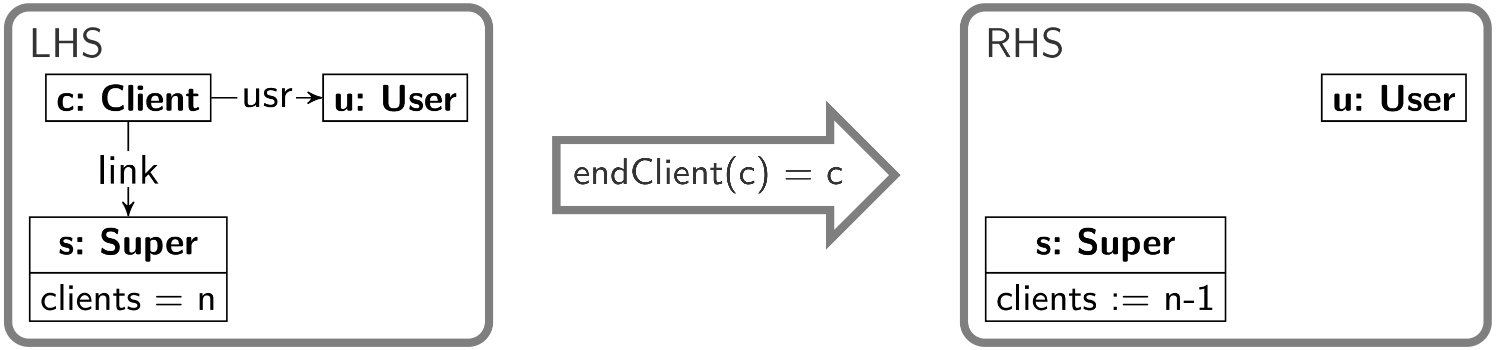
Specify changes by

- Difference between pre and post graphs
  - ➔ *Deleted: L \ R*
  - ➔ *Created: R \ L*
- Context required for the changes to happen
  - ➔ *Preserved: L ∩ R*



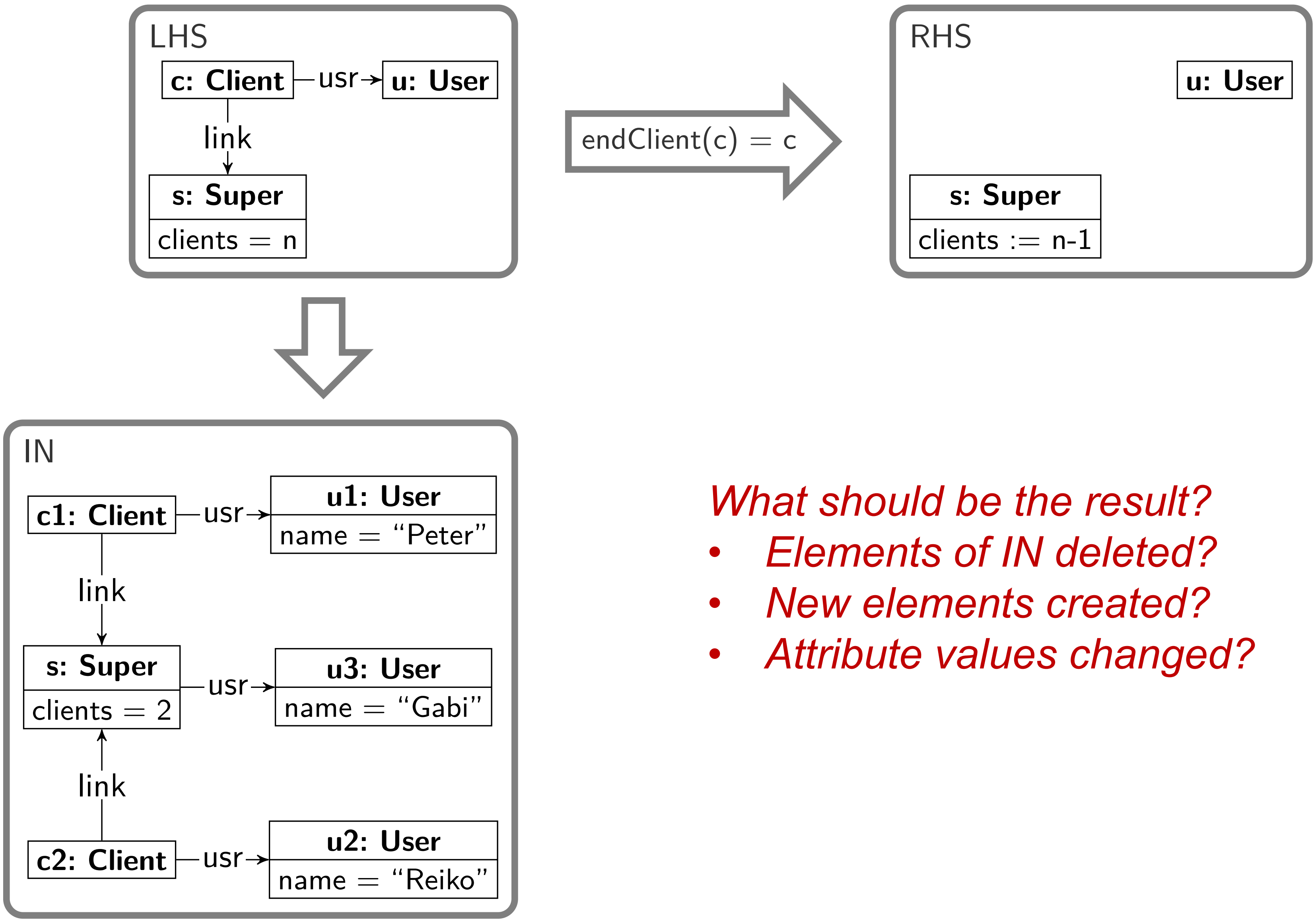
Graph Transformation Concepts





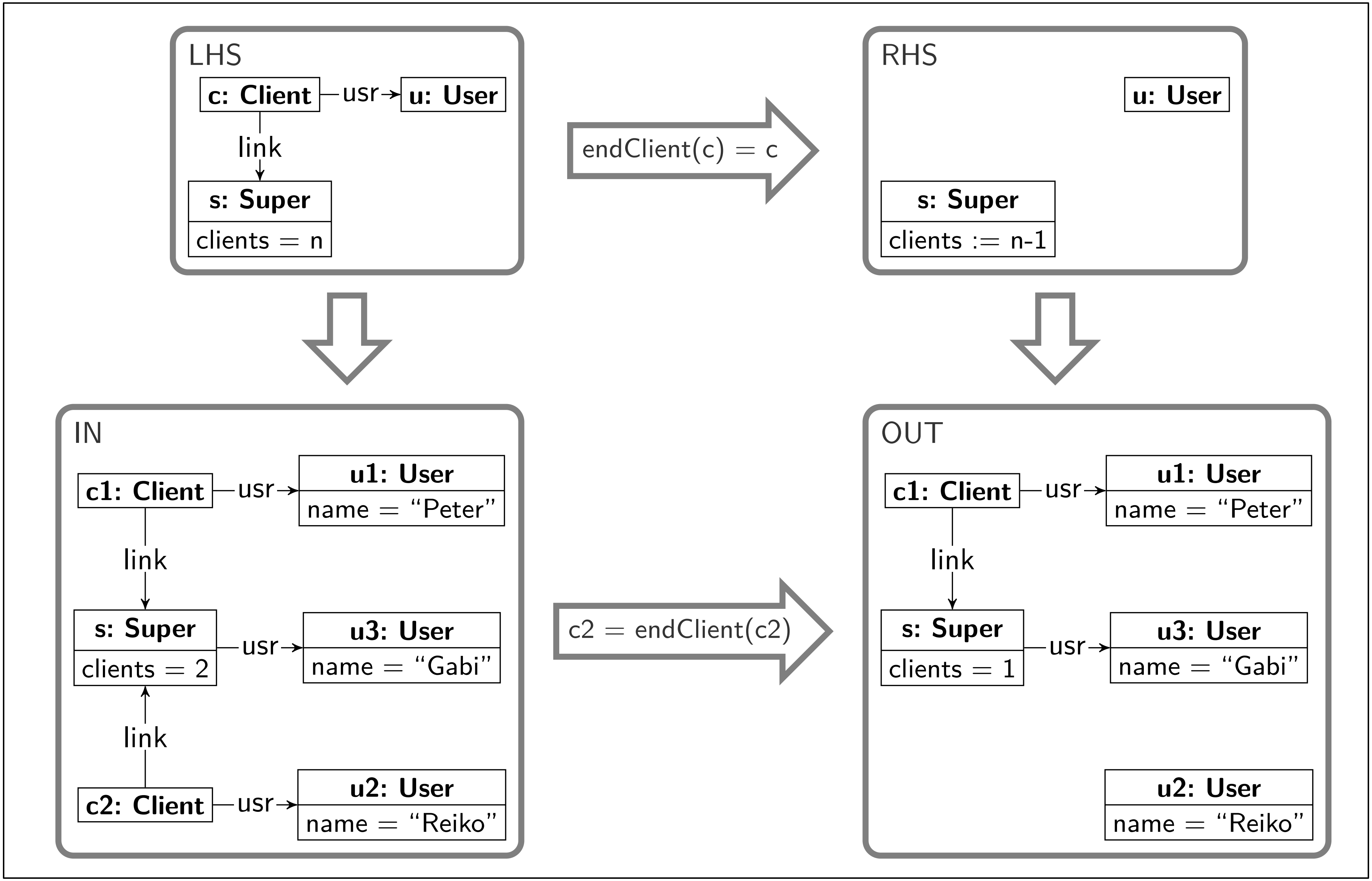
Rule Features

- Left- and right-hand side
- UML-like object notation
  - *c:Client, u: User, s: Super*
- Attribute conditions
  - *clients = n*
- Attribute assignments
  - *clients := n-1*
- Rule signature
  - *endClient(c: Client) : Client*
- Rule label
  - *endClient(c) = c*

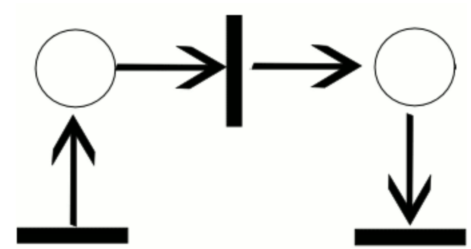


What should be the result?

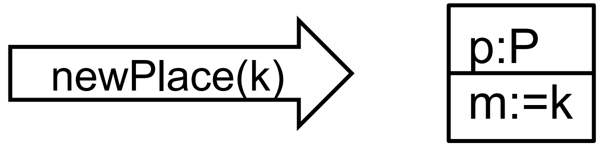
- *Elements of IN deleted?*
- *New elements created?*
- *Attribute values changed?*

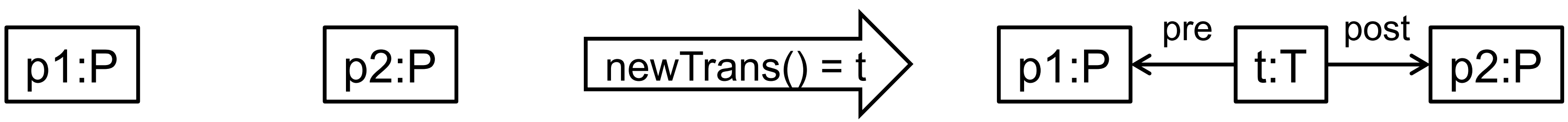

LHS
c: Client
usr
u: User
link
s: Super
clients = n
endClient(c) = c
RHS
u: User
s: Super
clients := n-1
IN
c1: Client
usr
u1: User
name = "Peter"
link
s: Super
clients = 2
usr
u3: User
name = "Gabi"
link
c2: Client
usr
u2: User
name = "Reiko"
c2 = endClient(c2)
OUT
c1: Client
usr
u1: User
name = "Peter"
link
s: Super
clients = 1
usr
u3: User
name = "Gabi"
u2: User
name = "Reiko"

GTBLE: Rules for Nets

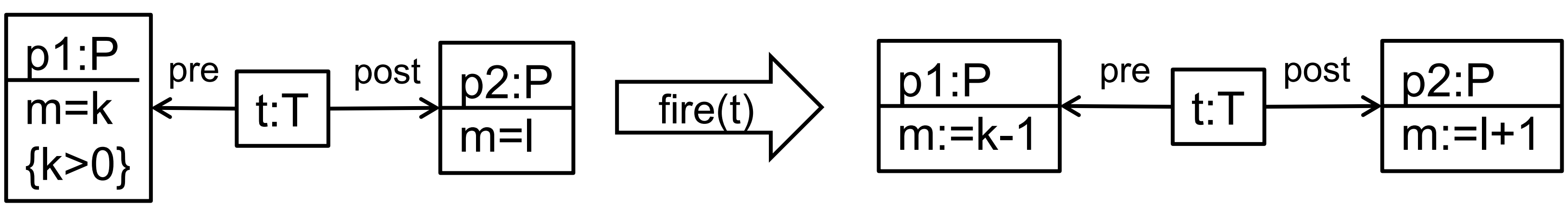


Rules to create all marked graphs (nets as attributed directed multi graphs)





Firing rule for marked graphs



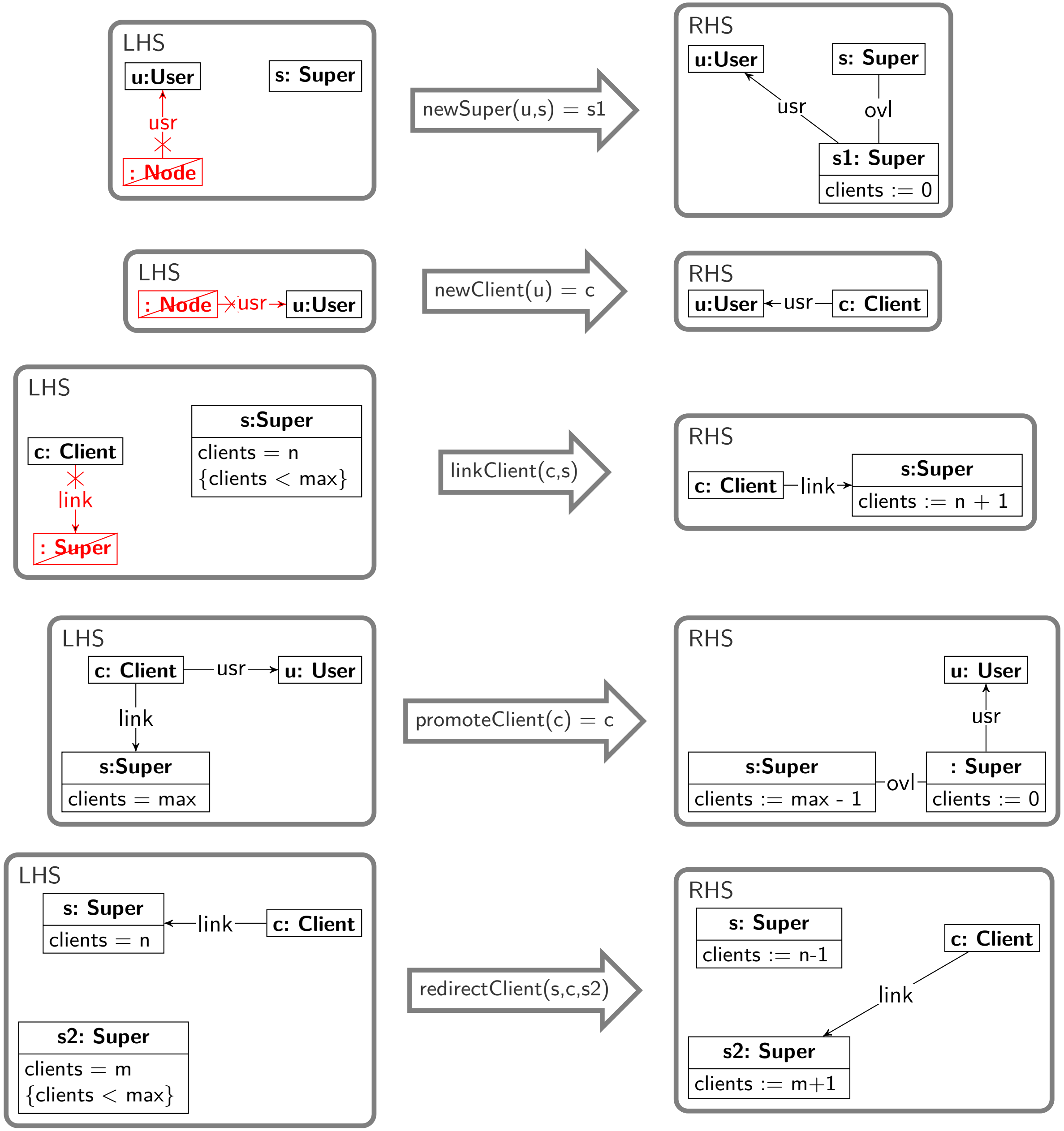
GTBSE: P2P Rules

Network formation

- *Node creation and promotion*
- *Linking and redirecting clients*

Rule Concepts

- *Creation*
- *Deletion*
- *Parameters*
- *NACs*
- *Attribute conditions*



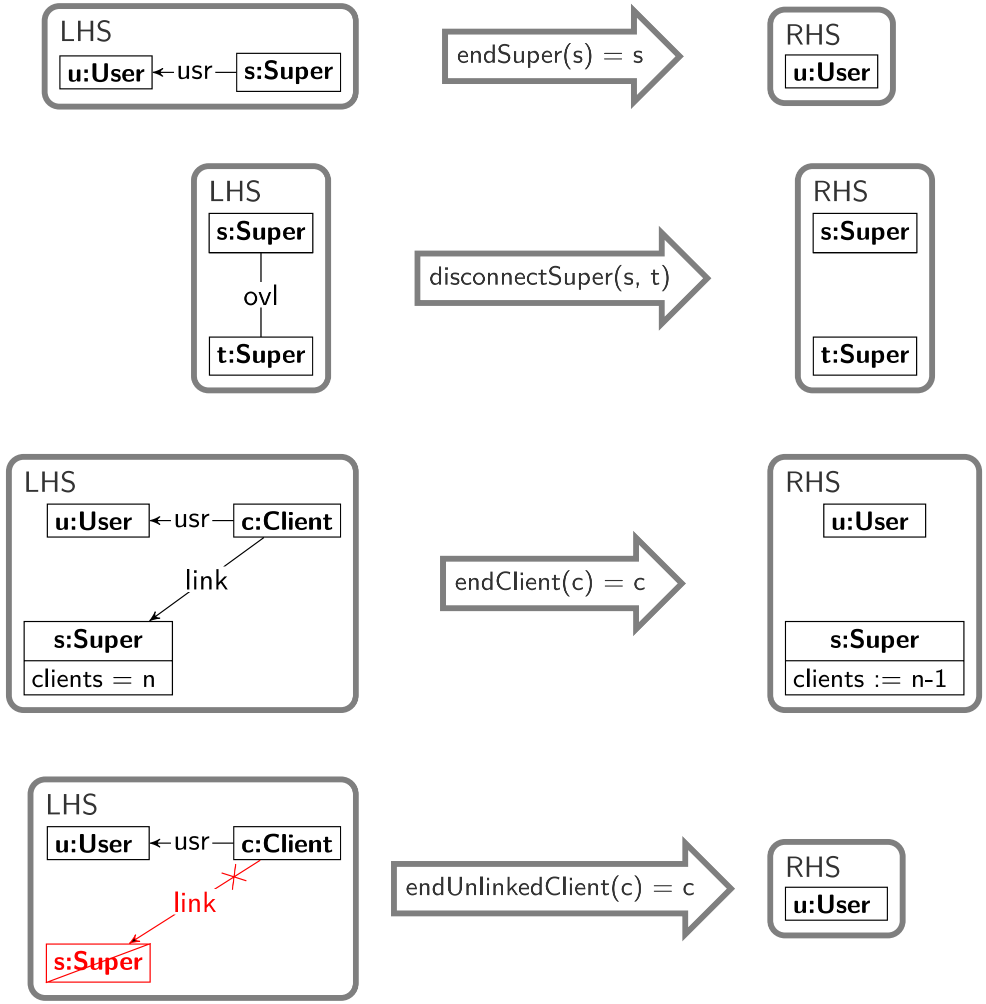
GTBSE: P2P Rules

Network deconstruction

- *Disconnect nodes*
- *Terminate nodes*

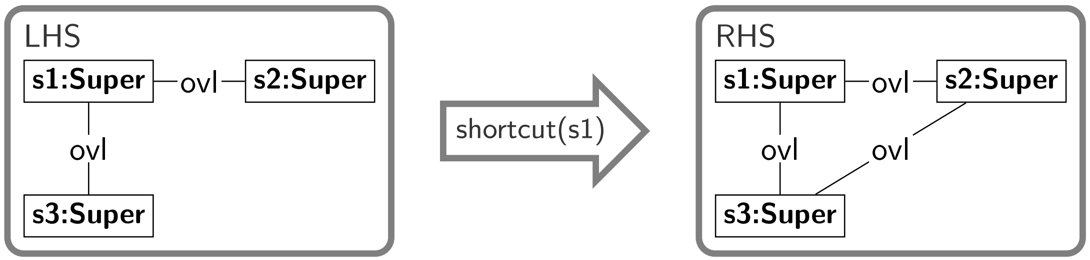
Rule Concepts

- *Creation*
- *Deletion*
- *Parameters*
- *NACs*
- *Attribute conditions*



Global application conditions

Create shortcuts for redundancy:



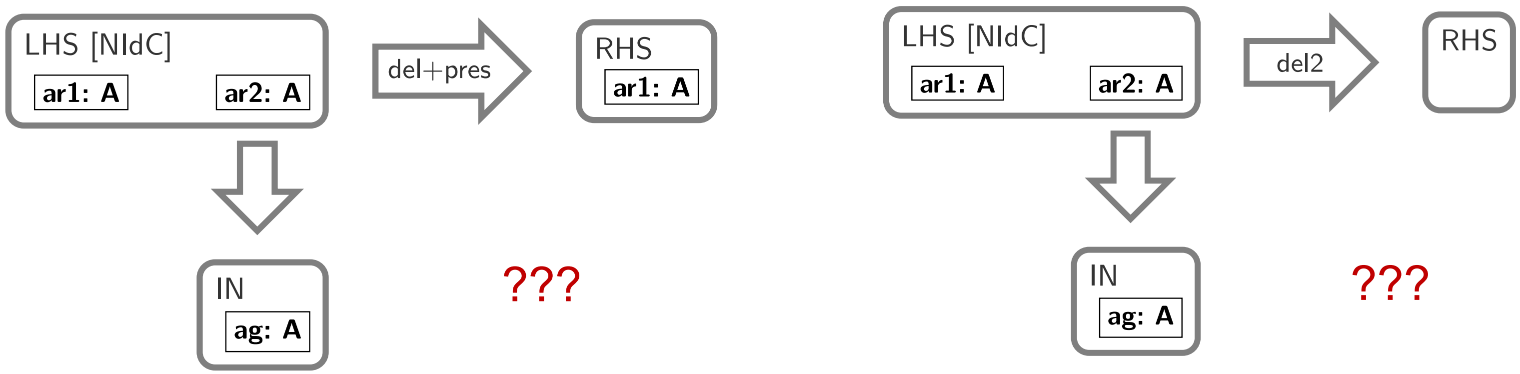
What could go wrong here?

➔ *With a match mapping s2 and s3 to the same node in the input graph, the rule would create a useless self edge (loop).*

Injective matches: Each element in LHS is mapped to a separate element in the input graph.

Global application conditions: Identification condition

Allowing for non-injective matches, what could happen here?



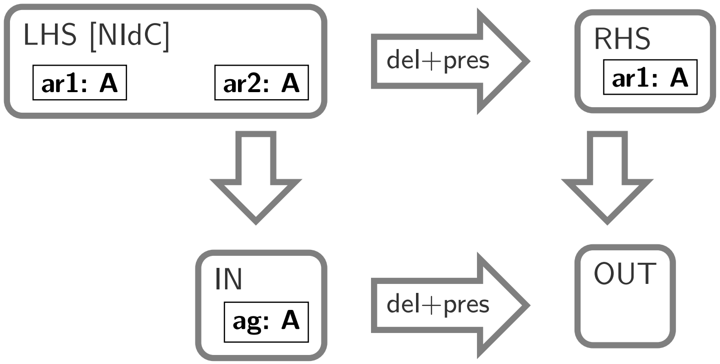
To prevent this (while allowing non-injective matches) the **identification condition** states that all deleted elements must be different from

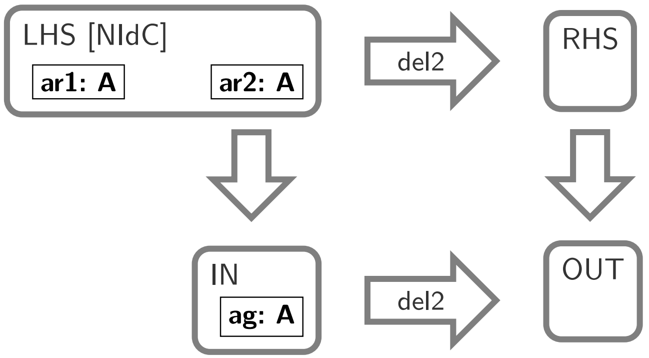
(1) other elements

(2) each other

Global application conditions: Identification condition

Allowing for non-injective matches, what could happen here?





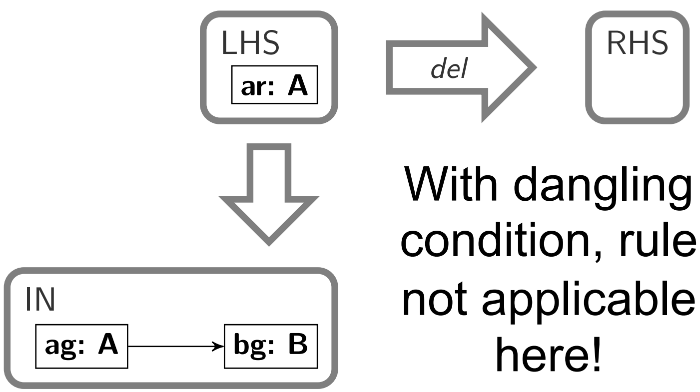
Identification condition: elements to be deleted are kept apart from

other elements and each other

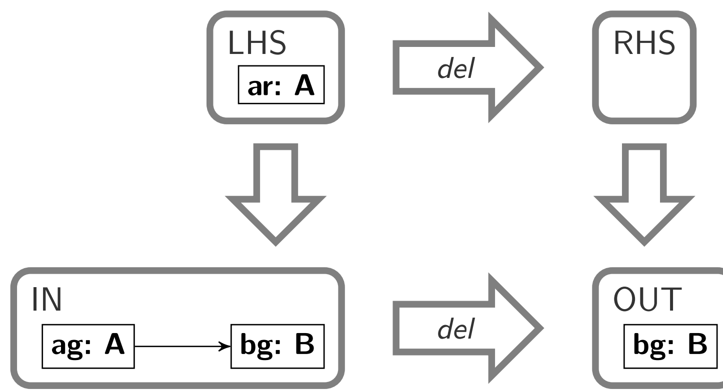
Global application conditions: Dangling condition

Deletion in unknown contexts

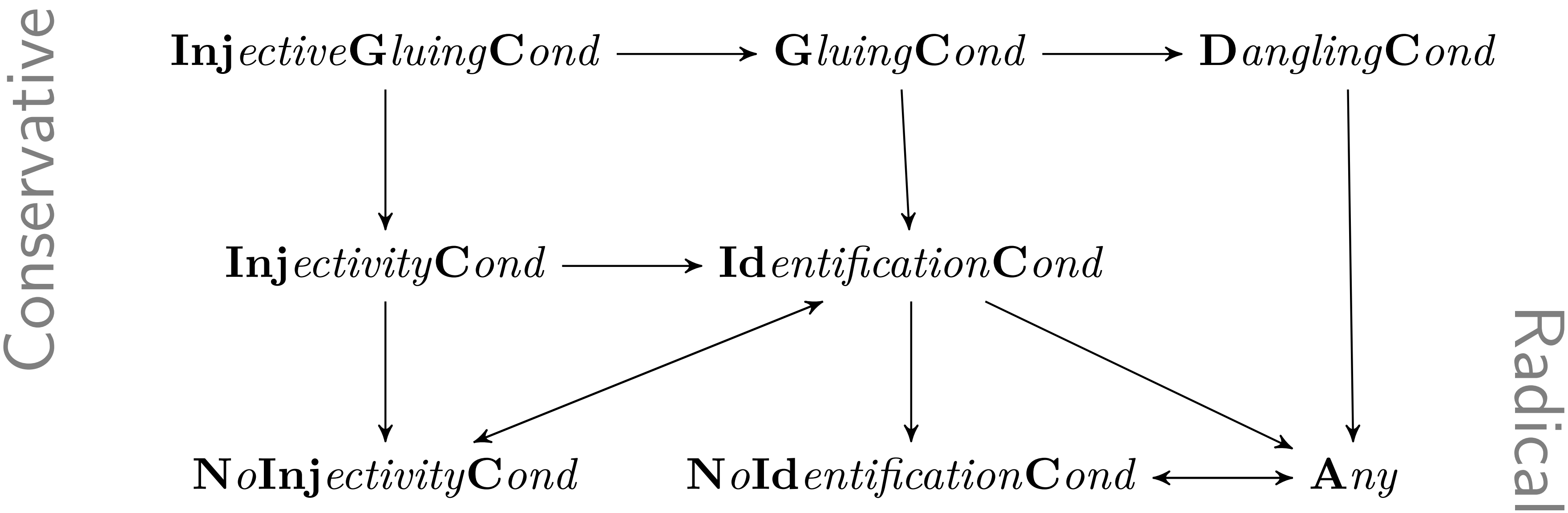
What happens to connections of s with other Super nodes?



With dangling condition, rule not applicable here!



Global application conditions: Overview



Formalise this, …

Type, instance graphs

- *Graphs and graph homomorphisms*
- *Slice category*

Attributed graphs

- *Graphs related to fixed data algebra*
- *Symbolic attributed graphs*

Rules, transformations

- *Rules as spans or partial morphisms*
- *DPO, SPO, SqPO*

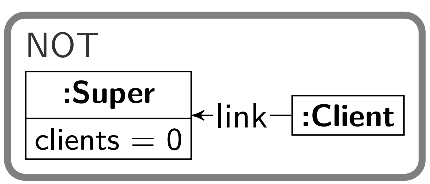
Advanced features: Graph constraints

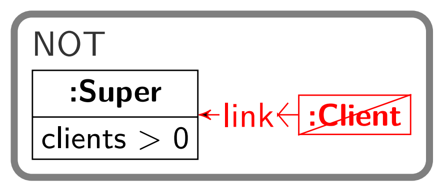
“The *clients* attribute of a *Super* node is 0 iff there are no *Client* nodes connected to it.”

Logically

self.clients=0 iff self.link->isEmpty()

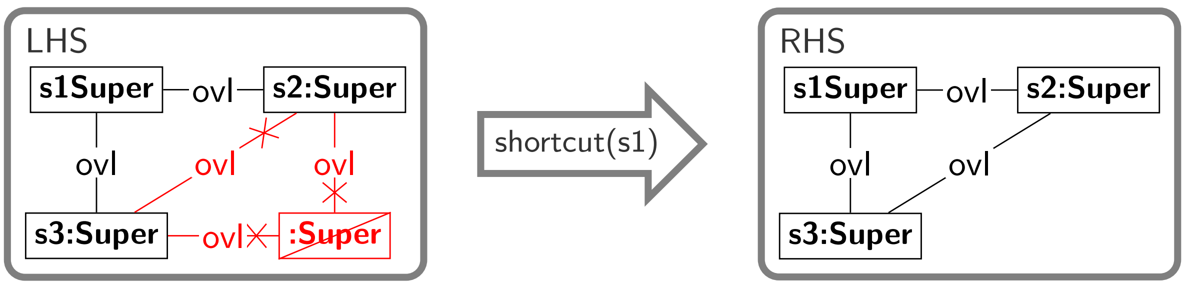
or graphically





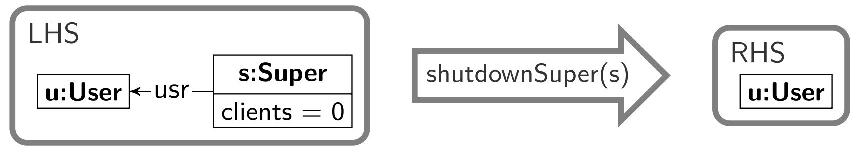
Advanced features: Negative application conditions

"Apply shortcut rule only if nodes *s2* and *s3* are not connected, neither directly nor via a 3rd node."



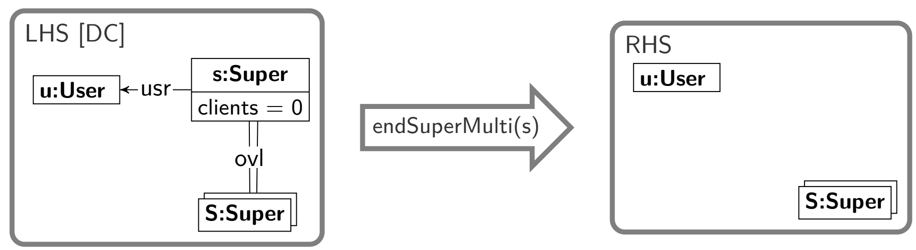
Advanced features: Multi Objects and Patterns

With the dangling condition, we cannot delete a *Super* node

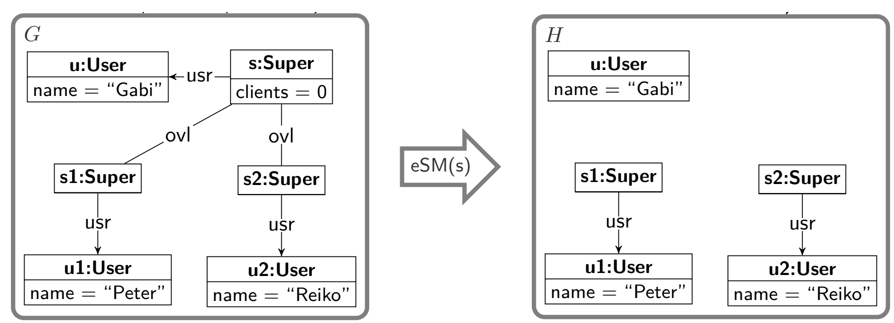


without explicitly deleting the links to **all its peers**.

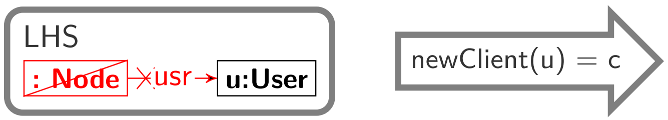
This is achieved by a rule with multi object.



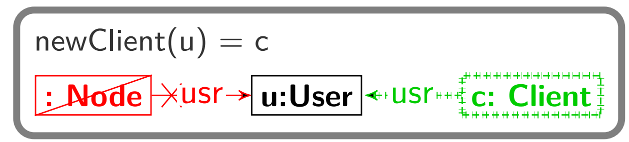
S:Super matches the set of all super nodes {s1, s2} linked to s by ovl edges, which all are deleted.



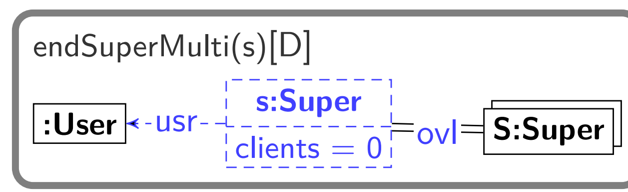
Integrated Rule Notation

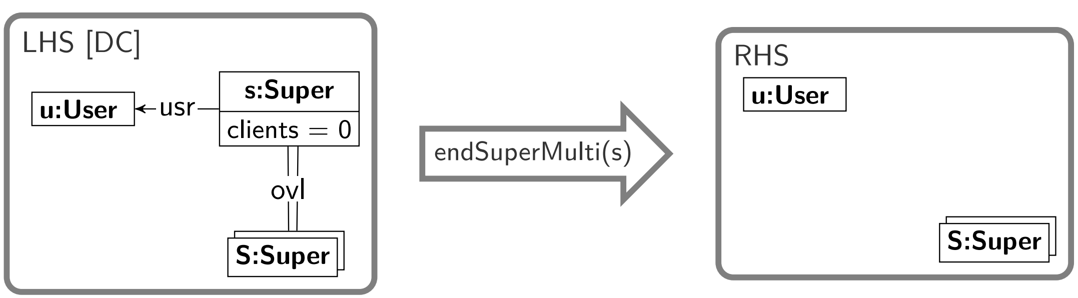


- Integrate left- and right-hand side into a single graph

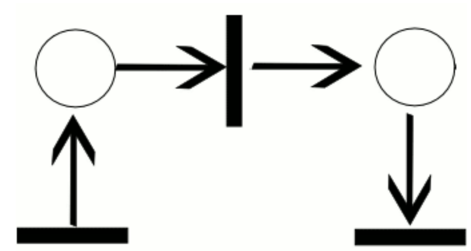


- Use colours and labels to distinguish different roles

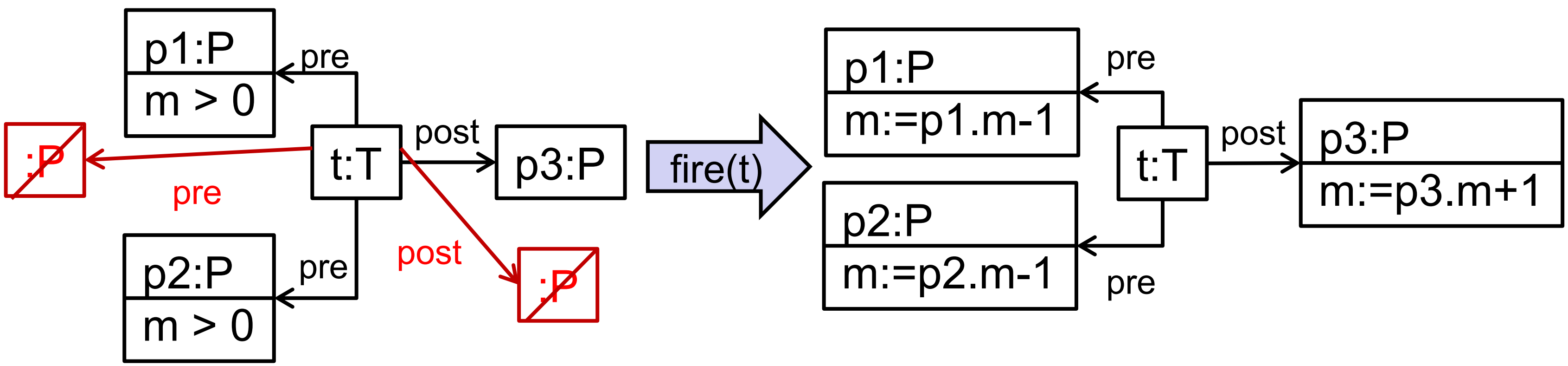




GTBLE: Firing rules for general Petri nets



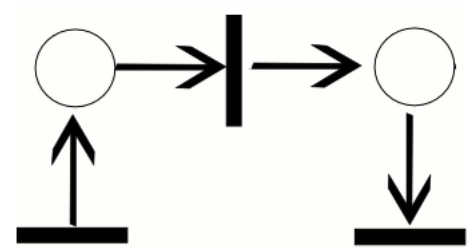
Rule for 2:1 transitions (2 places pre, 1 place post set)



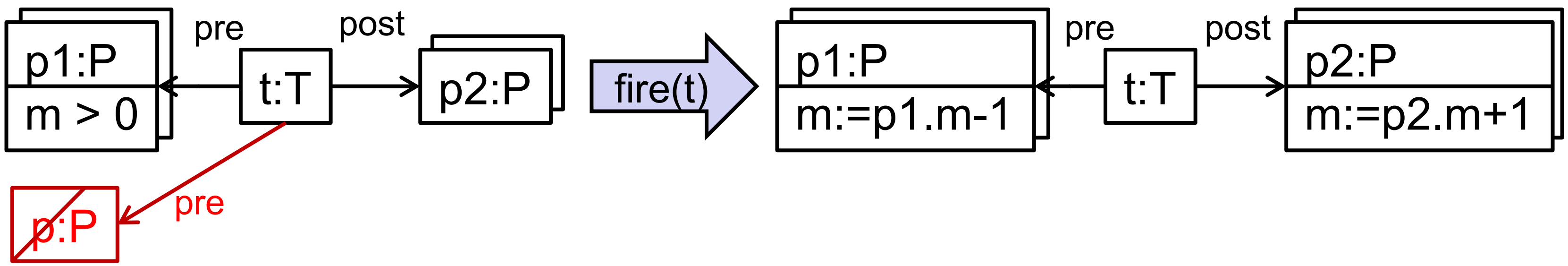
How to ensure this is only applicable to 2:1 transitions?

➔ Only matches if there are exactly 2 pre and 1 post places.

GTBLE: Firing rule for general Petri nets



Firing rule for arbitrary transitions



Matches sets of

- {p1:P | p1.m > 0 and p1 in *pre* set of t}
  - *NAC ensures there is no p:P in pre(t) not in that set*
- {p2:P | p2 in *post* set of t}

Updates attributes m of all matched P nodes

Formalise this, …

Type, instance graphs

- *Graphs and graph homomorphisms*
- *Slice category*

Attributed graphs

- *Graphs related to fixed data algebra*
- *Symbolic attributed graphs*

Rules, transformations

- *Rules as spans or partial morphisms*
- *DPO, SPO, SqPO*

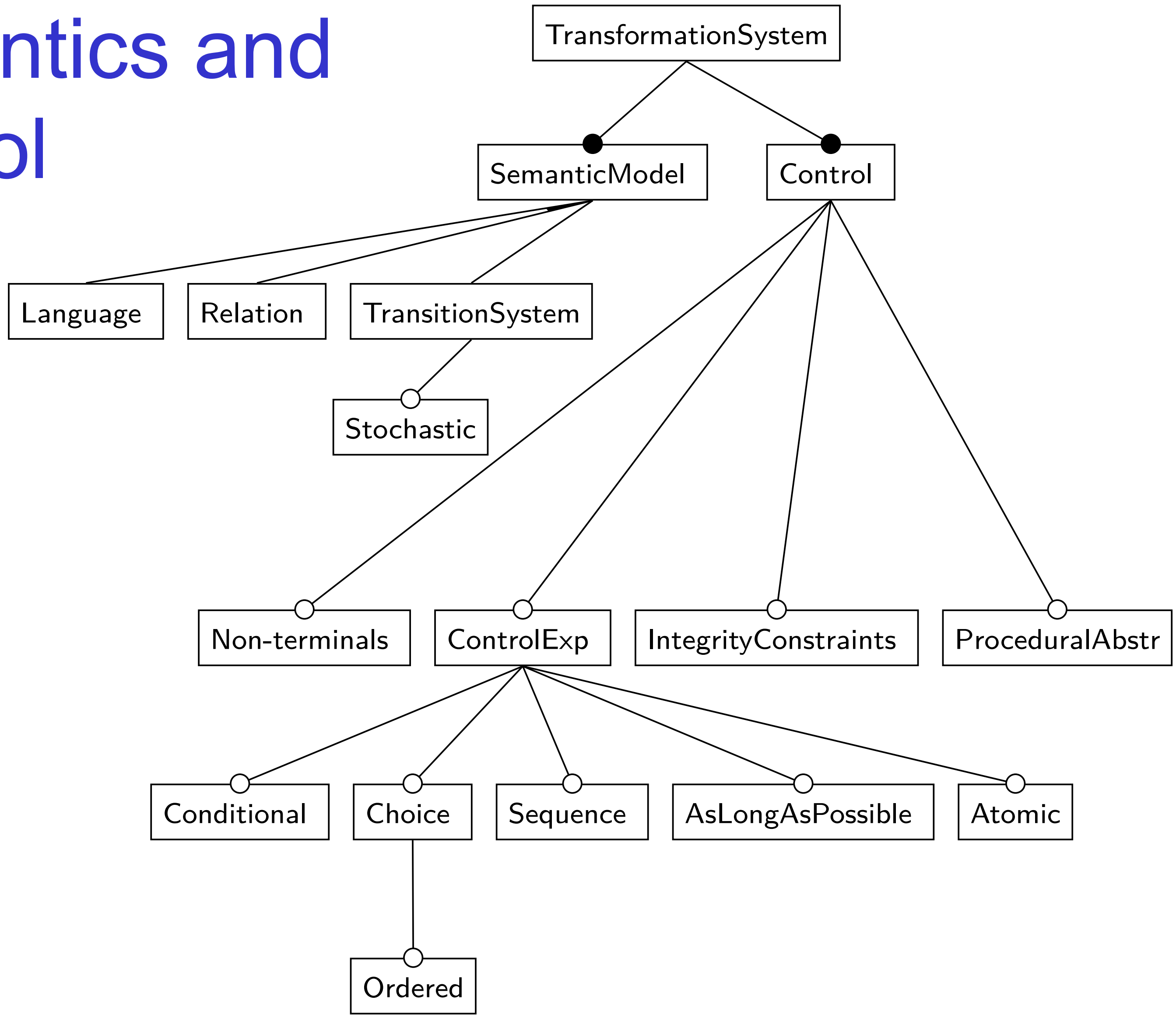
NACs, multi objects, patterns and rules

- *Forbidden matches*
- *Amalgamation*

Part 1: Foundations

- ✓ Graphs
- ✓ Graph Transformations
- Semantics and Control
  - *Graph transformation systems, grammars, and derivations*
  - *Graph languages, relations and transition systems*
  - *Rule-based programming: textual and diagrammatic*
- Analysis

Semantics and Control



Transformation systems, grammars, and semantic models

Graph transformation system $GTS = (TG, R)$

Graph grammar $GG = (TG, R, G_0)$

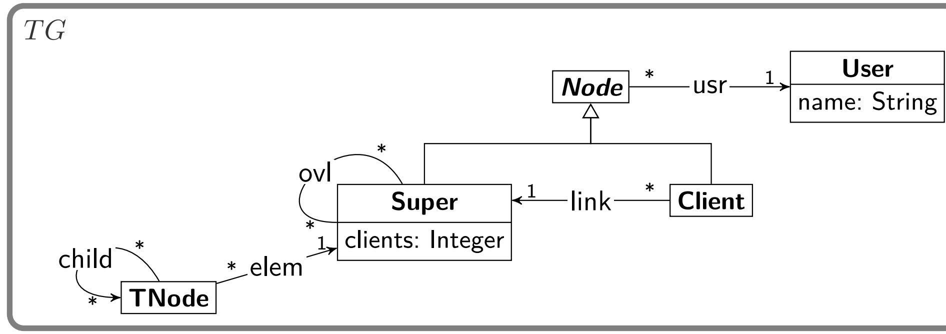
E.g. type graph and rules of the P2P model with start graph as initial network configuration

Semantic Models

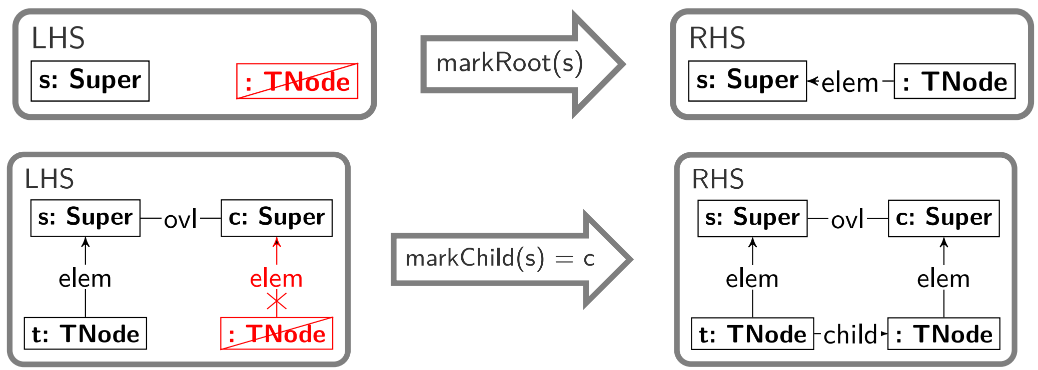
- Language: set of all graphs reachable from G_0
- Relation: mapping from input to output graphs
- Transition system: reachable graphs as states, transformation steps as transitions

Relation: Compute Spanning Tree of Super Nodes

Extended type graph



Rules to mark root and children of marked nodes

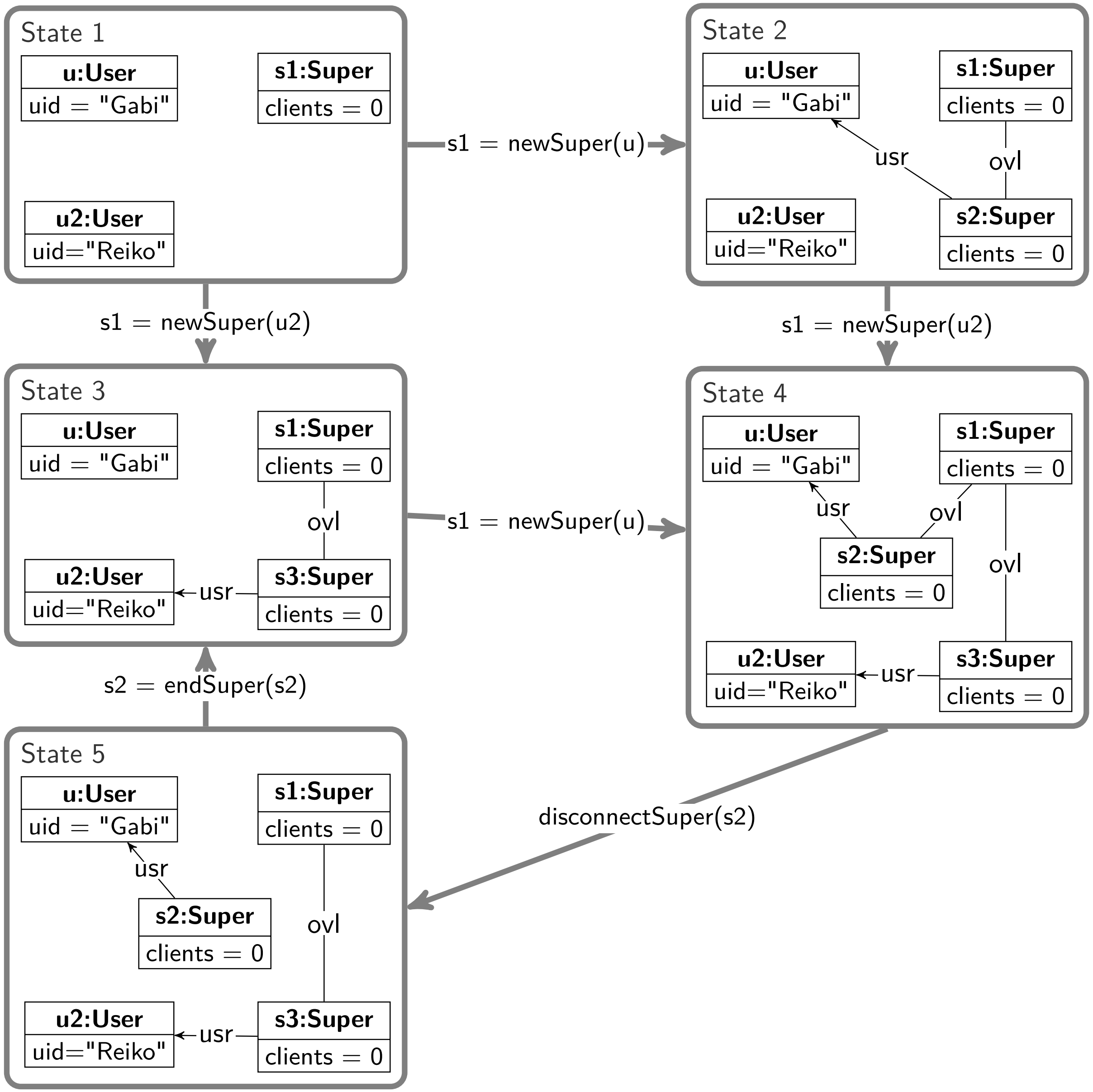


Initial set *I*: all unmarked instances graph of *TG*

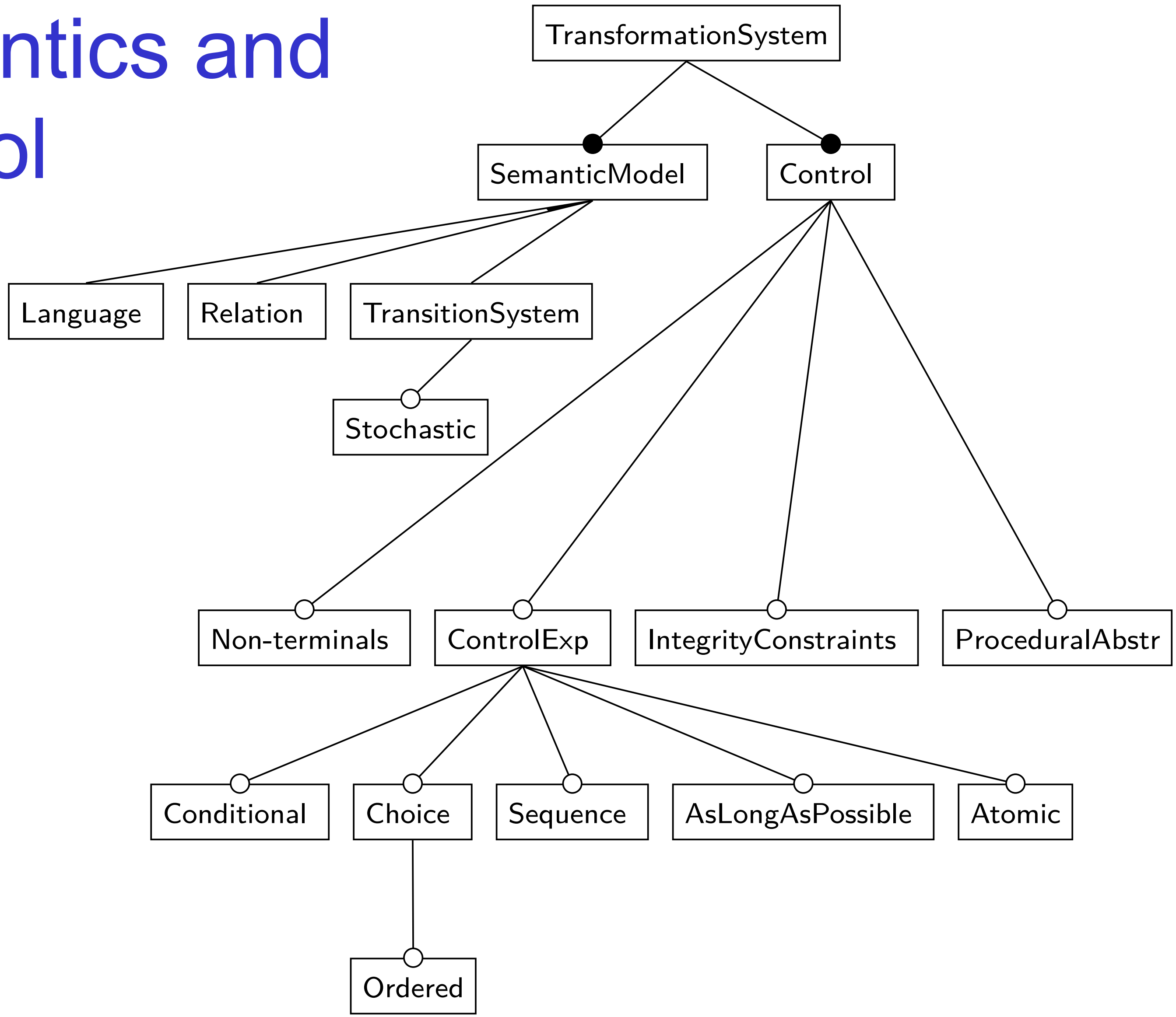
Final set *F*: all graphs where all *Super* nodes are marked

P2P LTS (partial)

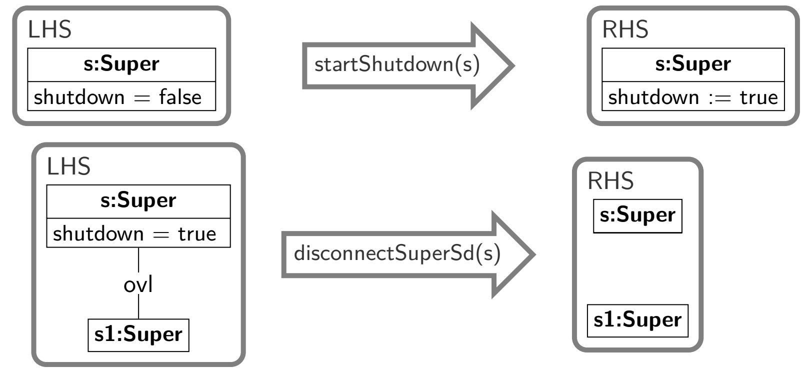
- Graphs as states
- Steps as transitions
- Rule names with args as labels



Semantics and Control

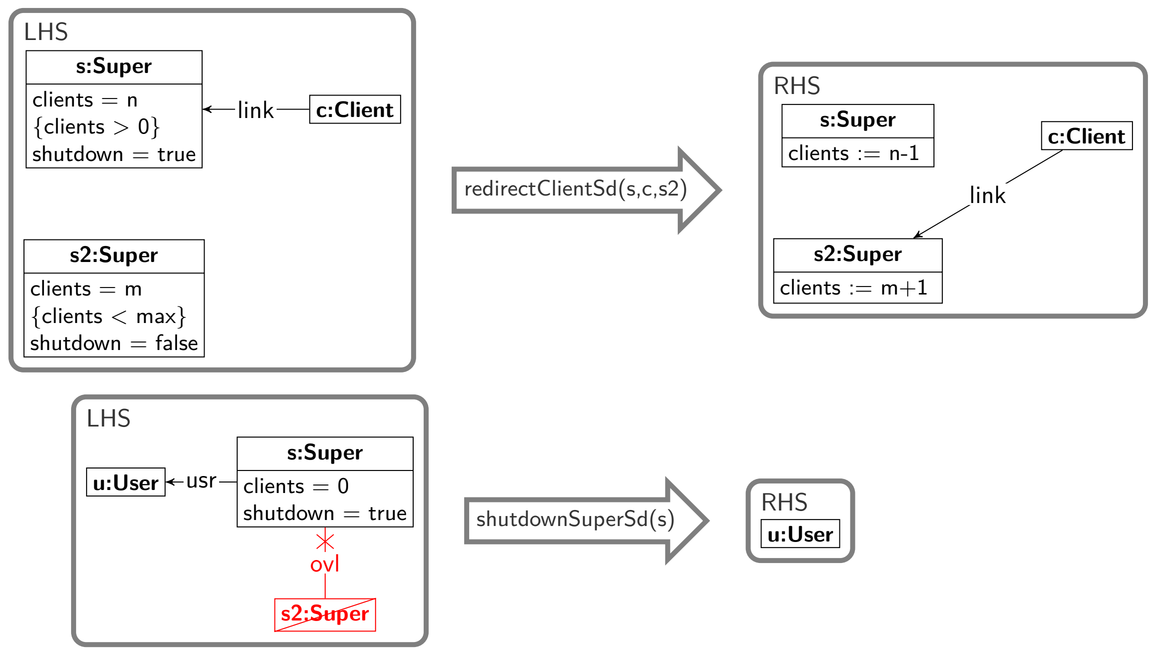


Rule-based programming: textual

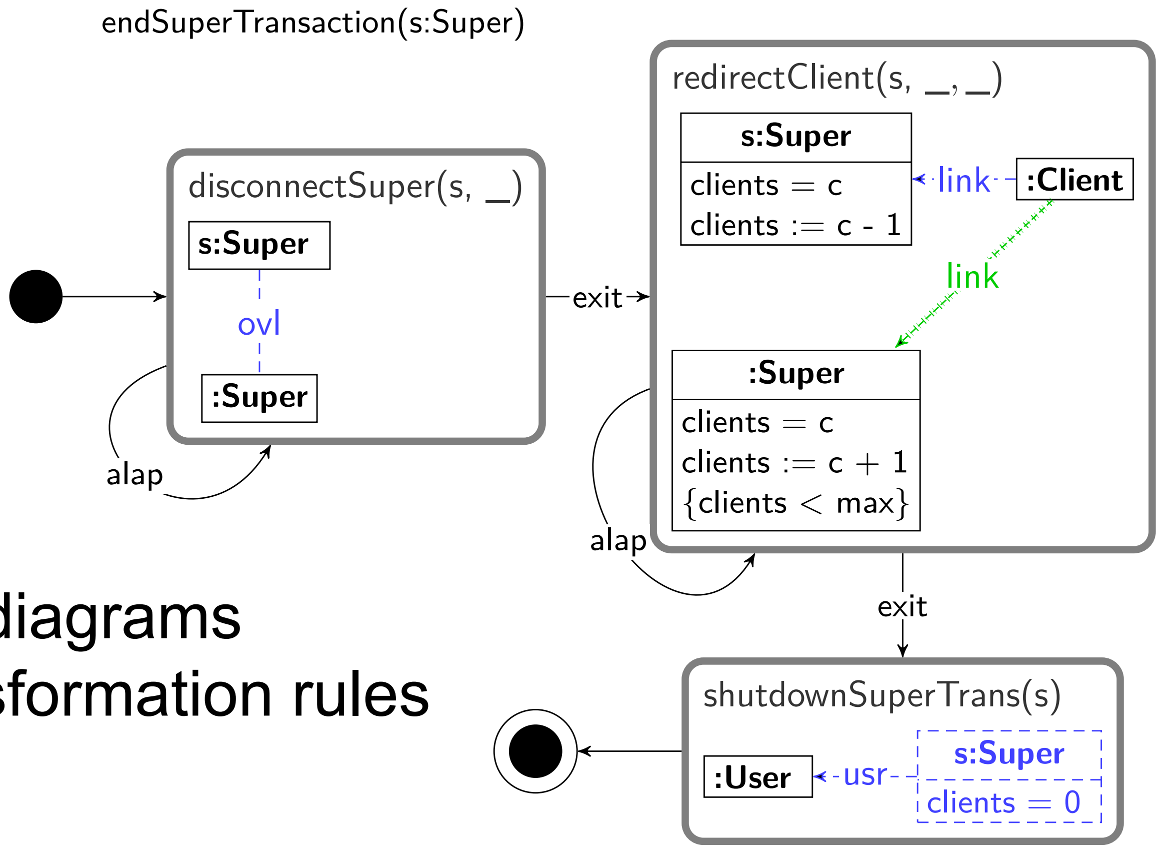


Control expressions,
Procedural abstraction, …

```
unit endSuper(s1: Super) {
    atomic {
        alap {
            disconnectSuper(s1,_)
        }
        alap {
            redirectClient (s1,_ ,_)
        }
        shutdownSuper(s1)
    }
}
```



Rule-based programming: diagrammatic



Using activity diagrams
to control transformation rules

Part 1: Foundations

- ✓ Graphs
- ✓ Graph Transformations
- ✓ Semantics and Control
- Analysis
  - *Properties: analysis problems*
  - *Techniques: solutions*
  - *Mapping problems to solutions*

Properties

Languages

- membership
- inclusion
- instance generation
- non-ambiguity

Relations

- functionality (uniqueness)
- totality
- injectivity
- surjectivity
- correctness

Trans Systems

- reachability
- invariants
- deadlocks
- planning, optimisation
- temporal, pre-post properties
- termination
- confluence

Techniques

Static

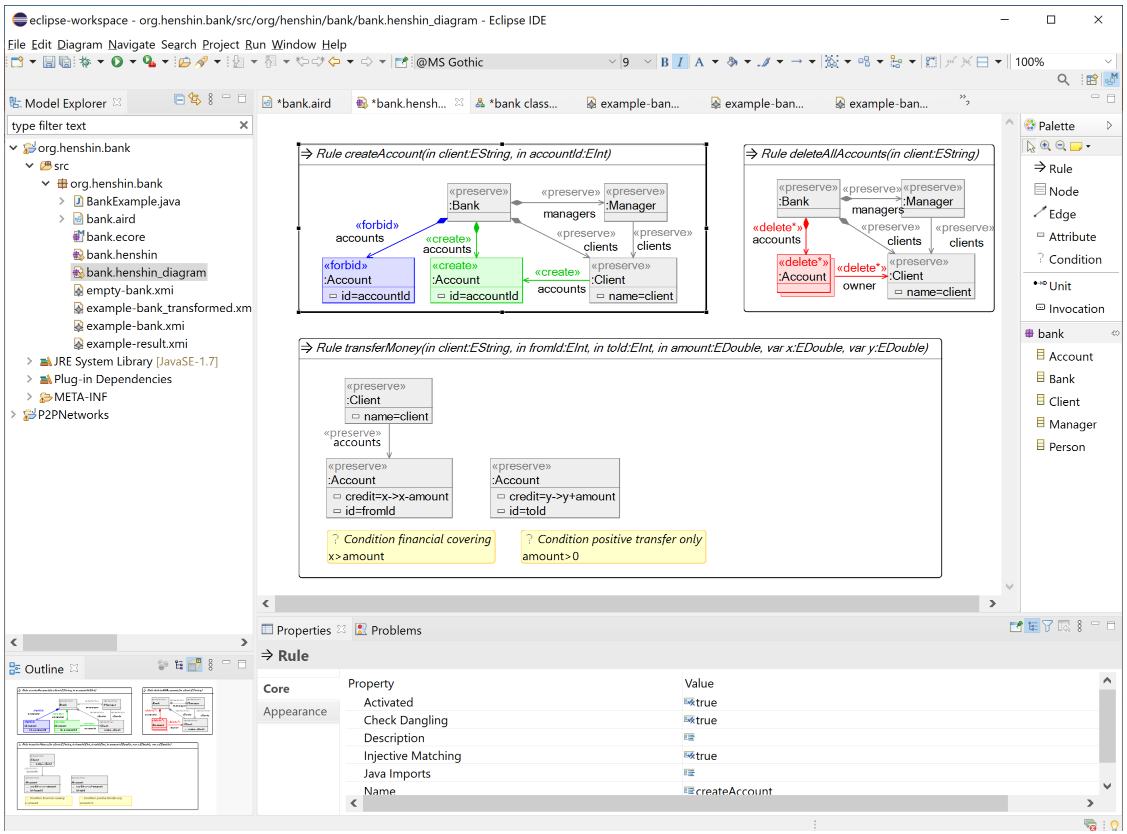
- Conflict and Dependency Analysis
- Termination Analysis
- Constraint Verification and Enforcement

Dynamic

- Model checking
- Graph parsing

	Conflict and dependency analysis	Termination analysis	Constraint verification	Model checking	Graph parsing
Language					
Membership				X	X
Inclusion			X	X	X
Instance generation	X		X	X	
Non-ambiguity	X	X		X	
Relation					
Functional behaviour	X	X		X	X
Totality		X		X	
Injectivity	X	X		X	X
Surjectivity		X		X	
Correctness	X		X	X	
Transition system					
Reachability				X	X
Invariants			X	X	
Deadlocks				X	
Planning & optimisation				X	
Temporal properties			X	X	
Termination		X		X	
Confluence	X	X		X	

Tools: Henshin



https://www.eclipse.org/henshin/

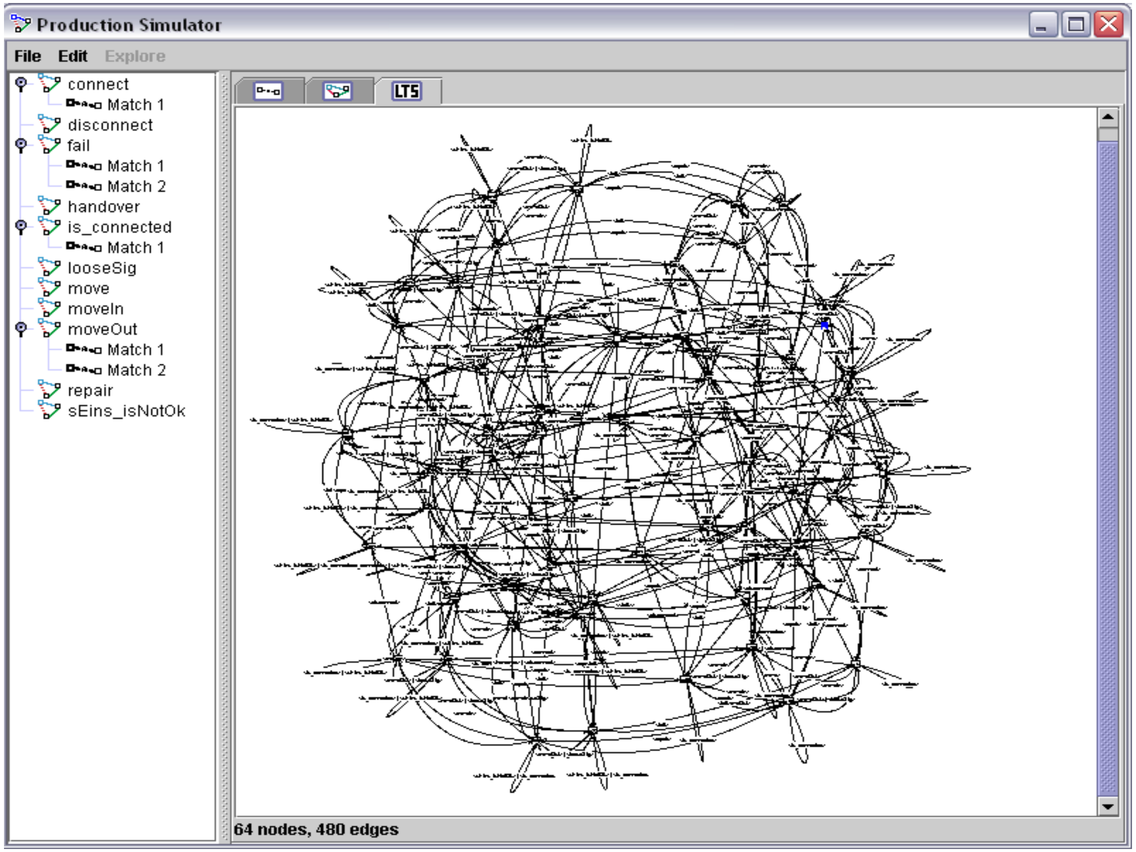
EMF model transformation

Conflict and dependency analysis

Constraint verification

Model checking by translation

Tools: Groove



https://groove.ewi.utwente.nl

Graph transformation

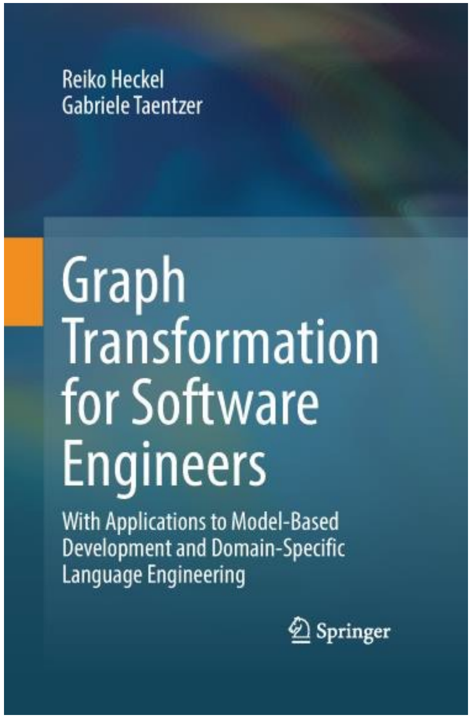
Visualisation of rules and matches

Native model checking in CTL and LTL

GT4SE
Graph Transformation
for Software Engineers

Part 1: Foundations

- ✓ Graphs
- ✓ Graph Transformations
- ✓ Semantics and Control
- ✓ Analysis



Further Reading

GT4SE
Graph Transformation for Software Engineers

www.graph-transformation-for-software-engineers.org

- Author copy
- Exercises
- Slides

Part I – Foundations of Graph Transformation

1. Graphs for Modelling and Specification
2. Graph Transformation Concepts
3. Beyond Individual Rules: Usage Scenarios and Control Structures
4. Analysis and Improvement of Graph Transformation Systems

Part II – Graph Transformation in Software Engineering

5. Detecting Inconsistent Requirements in a Use-Case-Driven Approach
6. Service Specification and Matching
7. Model-Based Testing
8. Reverse Engineering: Inferring Visual Contracts from Java Programs
9. Stochastic Analysis of Dynamic Software Architectures
10. Advanced Modelling-Language Definition: Integrating Metamodelling with Graph Transformation
11. Improving Models and Understanding Model Changes
12. Translating and Synchronising Models