UNIVERSITY OF LEICESTER

MGS 2022

GT4SE
Graph Transformation
for Software Engineers

Graph Transformation: Foundations and Applications to Software Engineering

Reiko Heckel

University of Leicester, UK

Based on book with Gabriele Taentzer

Philipps Universität Marburg, Germany

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*

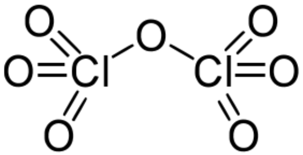
- Language Engineering

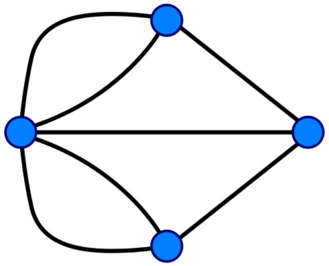
  ➔ *GTBLE*

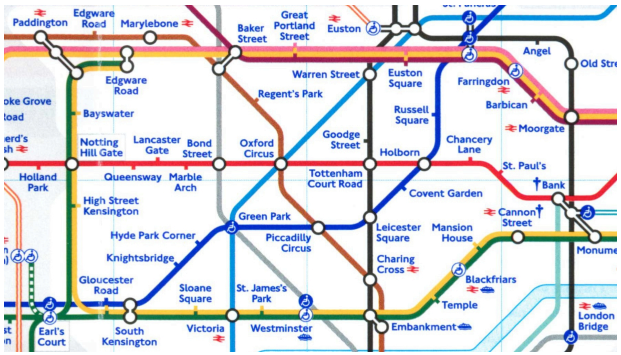
Part 1: Foundations

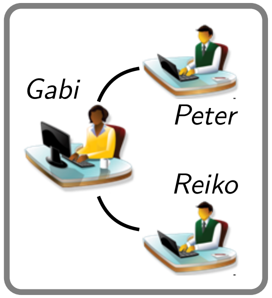
- Graphs
- Graph Transformations
- Semantics and Control
- Analysis

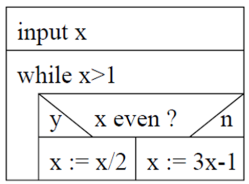
Graphs

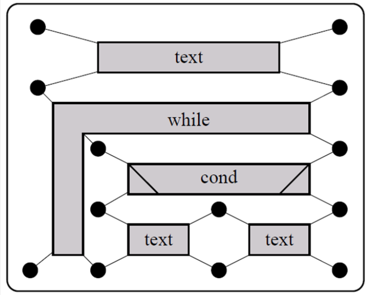

O
O
O
O
O
O
O
Cl
Cl



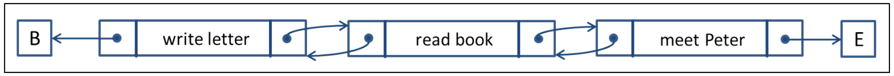

Paddington
Edgware Road
Marylebone
Baker Street
Great Portland Street
Euston
Angel
Old Stre
Edgware Road
Warren Street
Euston Square
Farringdon
Regent's Park
Barbican
Russell Square
Moorgate
Bayswater
Notting Hill Gate
Lancaster Gate
Bond Street
Oxford Circus
Goodge Street
Holborn
Chancery Lane
St. Paul's
Holland Park
Queensway
Marble Arch
Tottenham Court Road
Bank
Covent Garden
High Street Kensington
Green Park
Cannon Street
Hyde Park Corner
Piccadilly Circus
Leicester Square
Mansion House
Monume
Knightsbridge
Charing Cross
Blackfriars
Gloucester Road
Sloane Square
St. James's Park
Temple
London Bridge
Earl's Court
South Kensington
Victoria
Westminster
Embankment

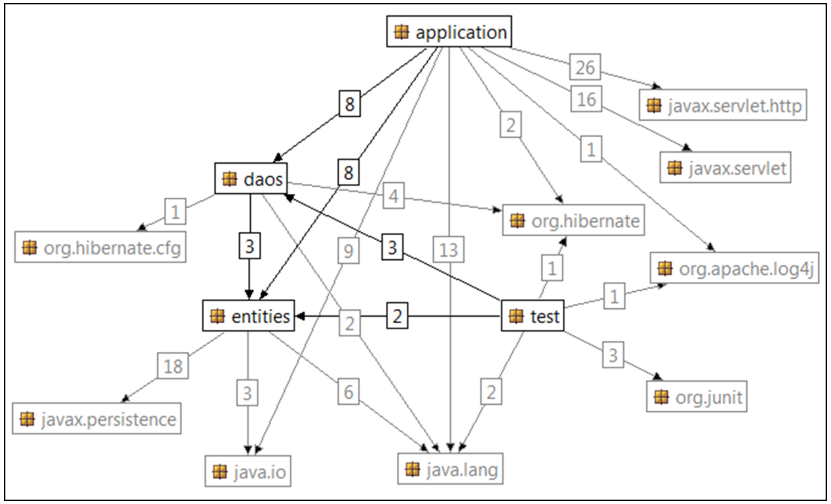

Gabi
Peter
Reiko

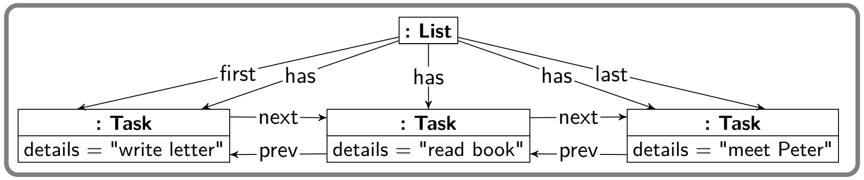

input x
while x>1
y
x even ?
n
x := x/2
x := 3x-1


text
while
cond
text
text

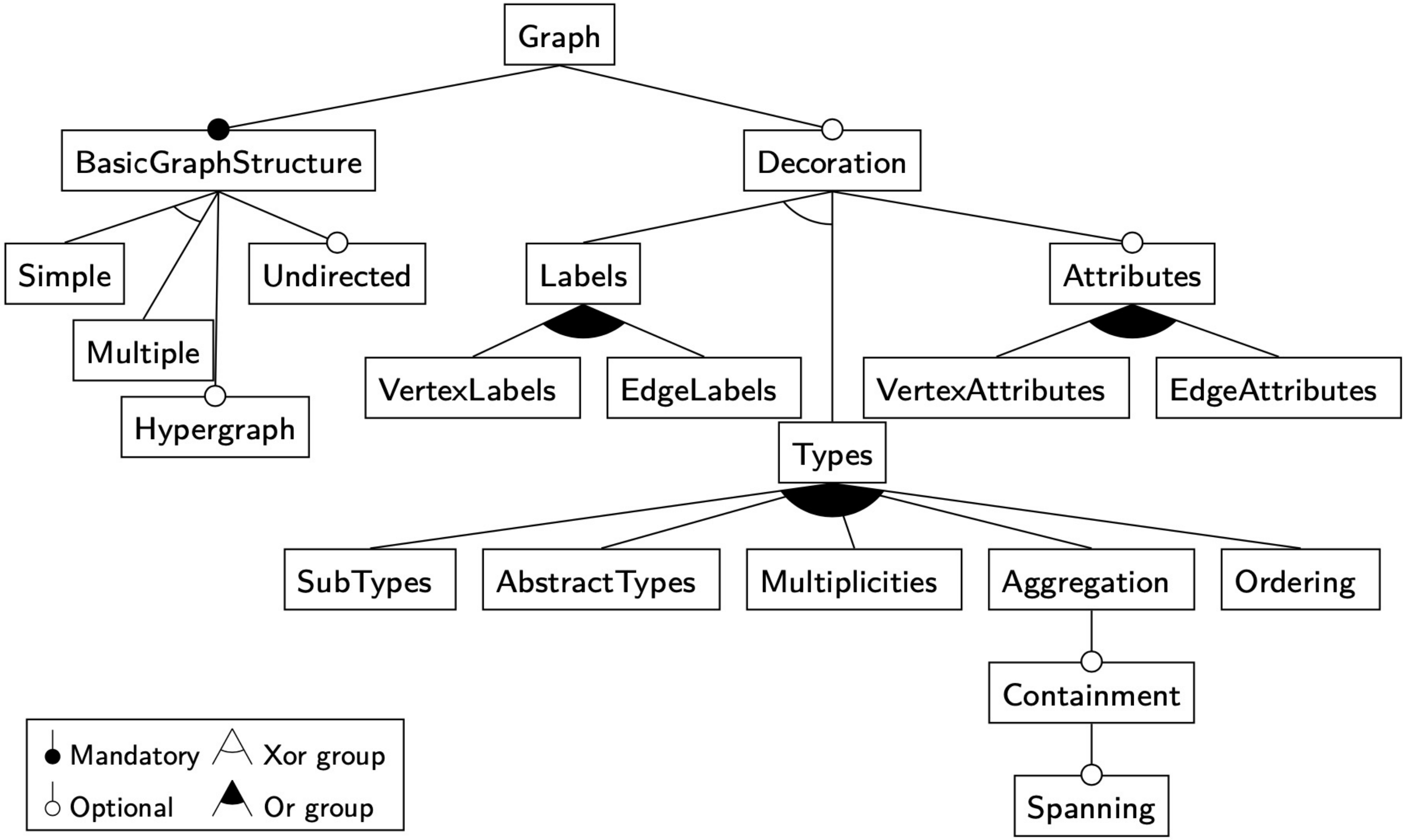
Goat
Jackal
Lion
Wild Cat
Rabbit
Kite
Green Plant
Poducer
Mouse
Owl
Snake
Food Web in a Forest


B
write letter
read book
meet Peter
E


application
javax.servlet.http
javax.servlet
daos
org.hibernate
org.hibernate.cfg
org.apache.log4j
entities
test
javax.persistence
org.junit
java.io
java.lang
26
16
8
2
1
8
4
1
3
9
3
13
1
1
2
2
18
3
6
2
3


: List
first
has
has
has
last
: Task
details = "write letter"
next
prev
: Task
details = "read book"
next
prev
: Task
details = "meet Peter"

Variants of graphs

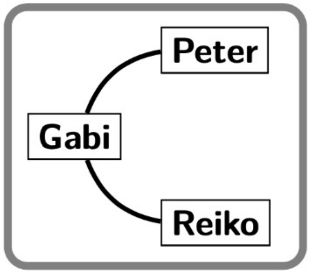


GTBSE or GTBLE?

Directed or undirected Labelled or typed

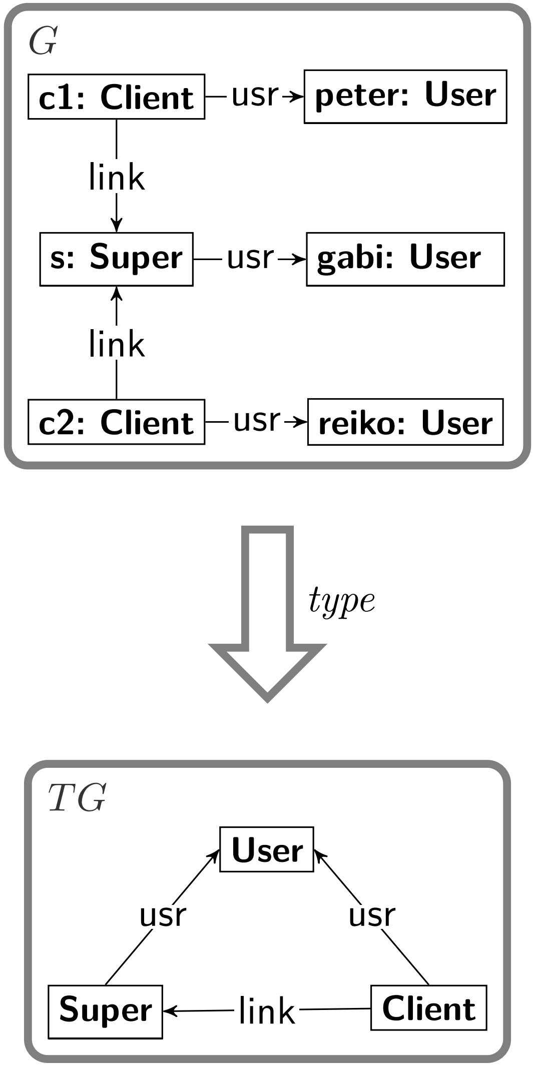
Undirected labelled graph

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

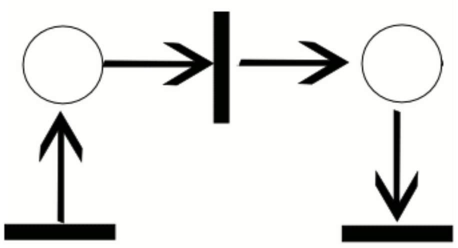


Directed typed graph

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

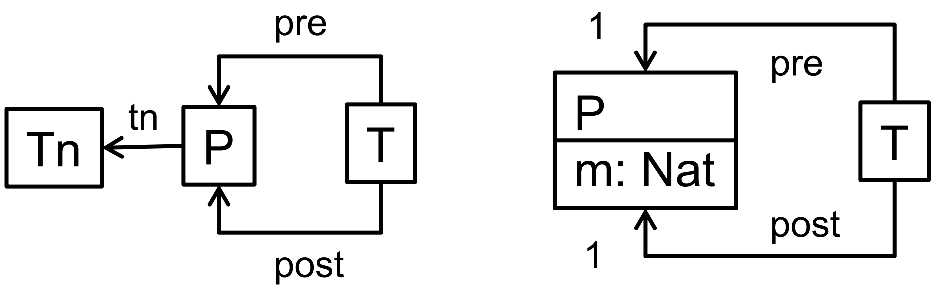


GTBLE: Petri nets



Create

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

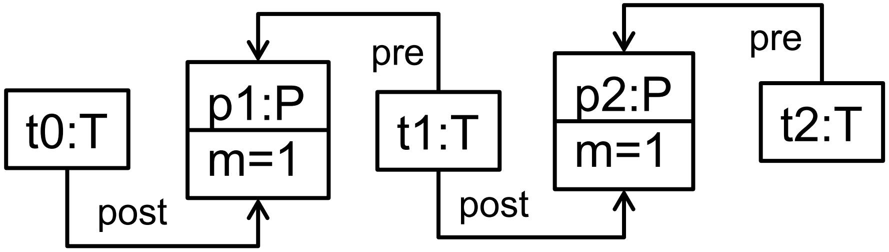


Design decisions

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

Marked graphs (right)

- *add constraints for singleton pre and post sets*

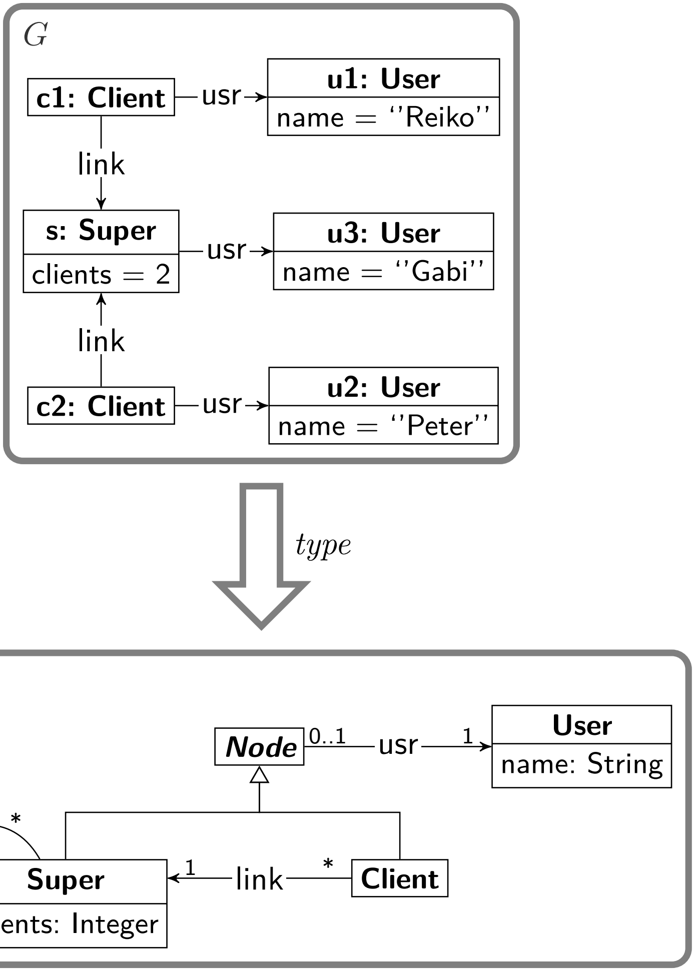


Typed attributed graphs with subtyping

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

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

Part 1: Foundations

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

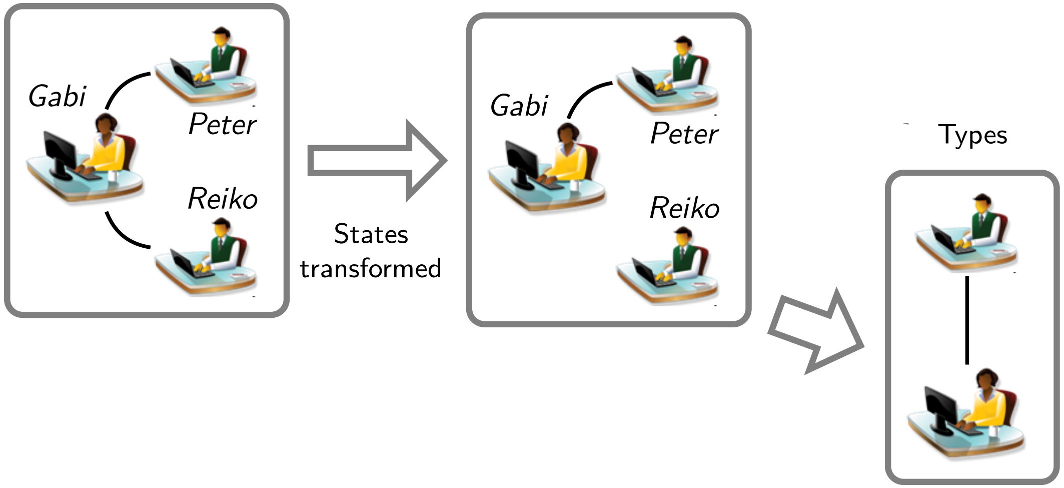
Rules generalize transformations

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

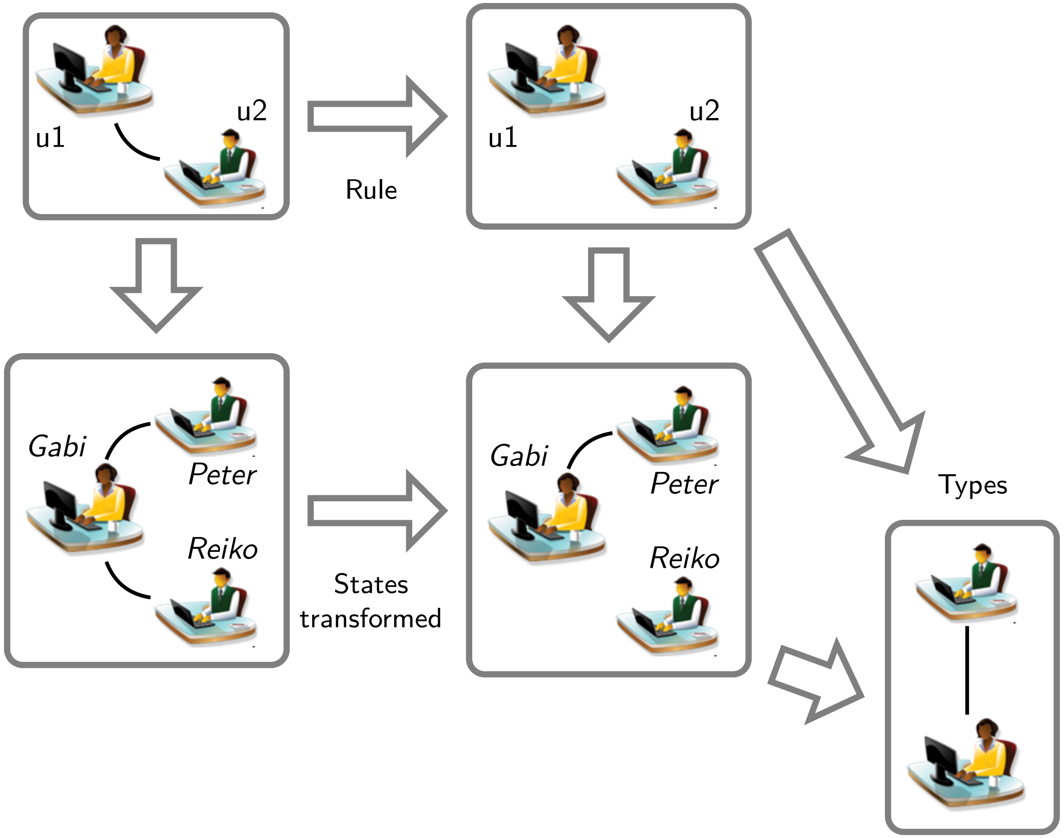
- *What changes, and how?*
- *In which situations?*



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*

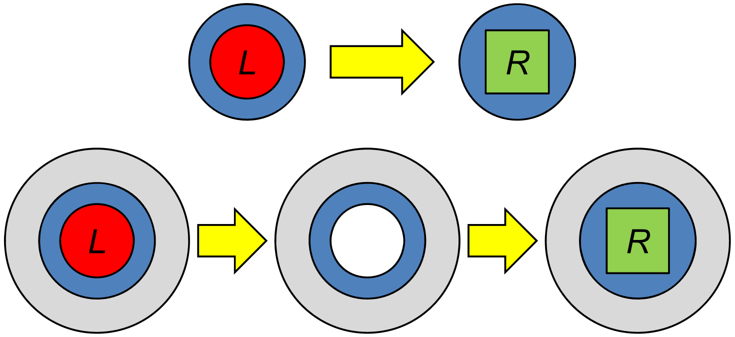


Two ways to embed replacement R graph into context

Gluing approaches

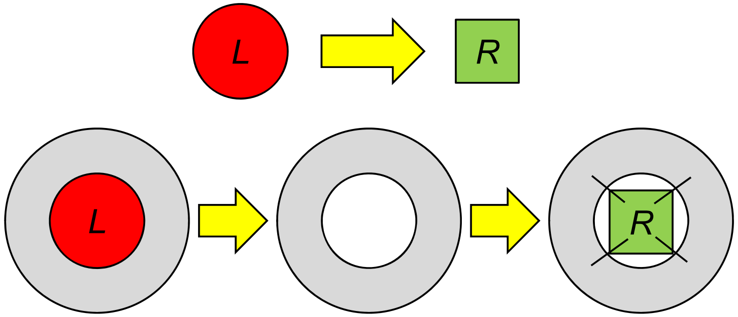
Interface shared between L and R provides gluing points

➔ *our choice*



Connecting approaches

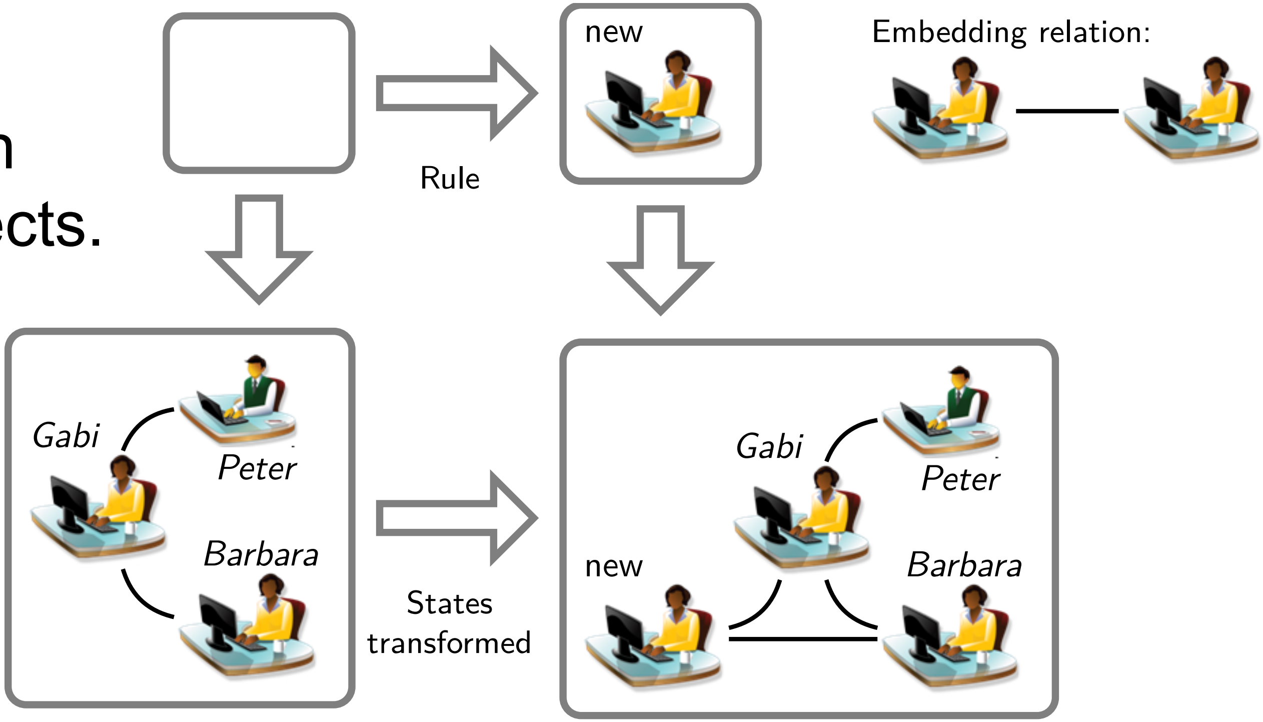
R is linked to context by edges created according to embedding relation



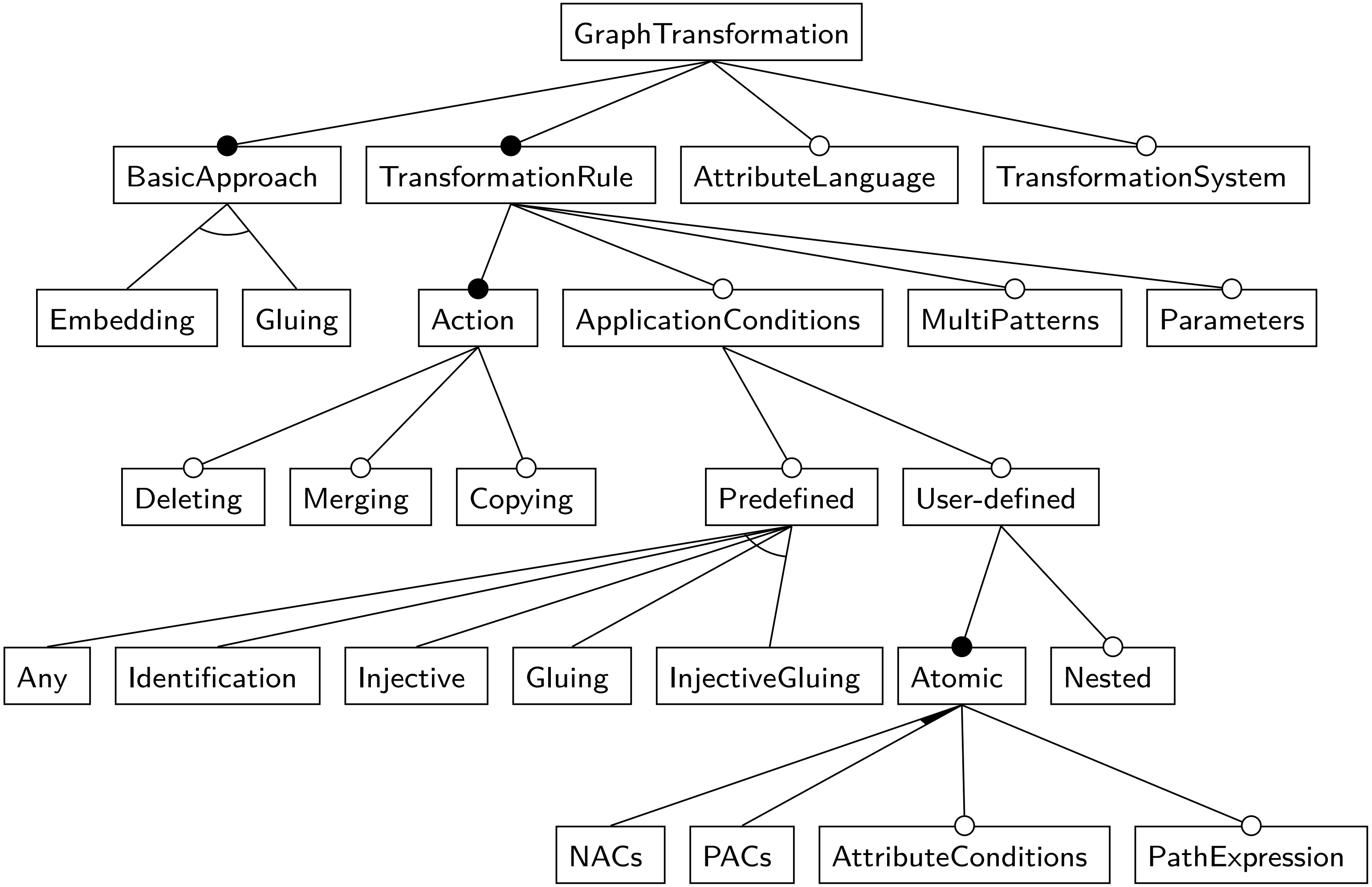
Example: Connecting Approach

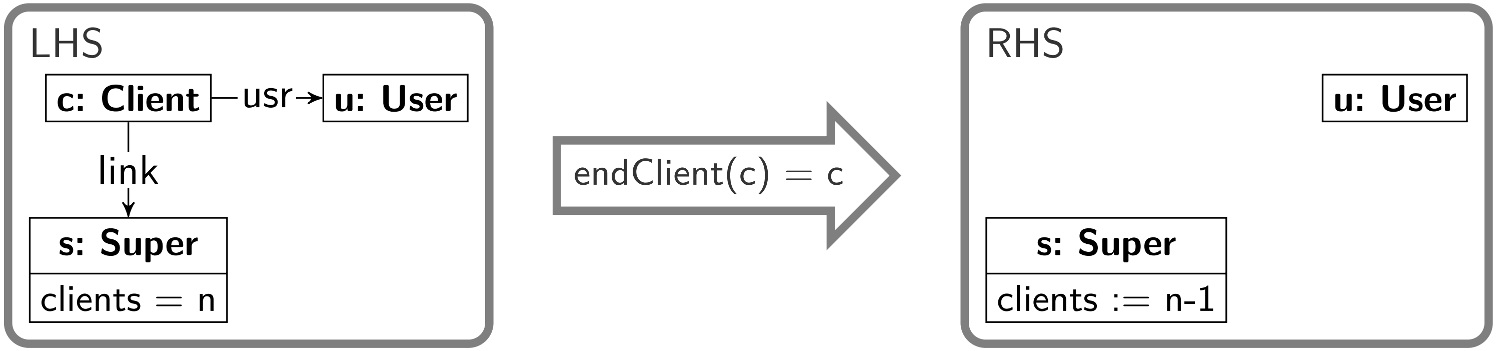
More powerful, but mathematically less tractable.

Simulated by gluing approach using multi objects.



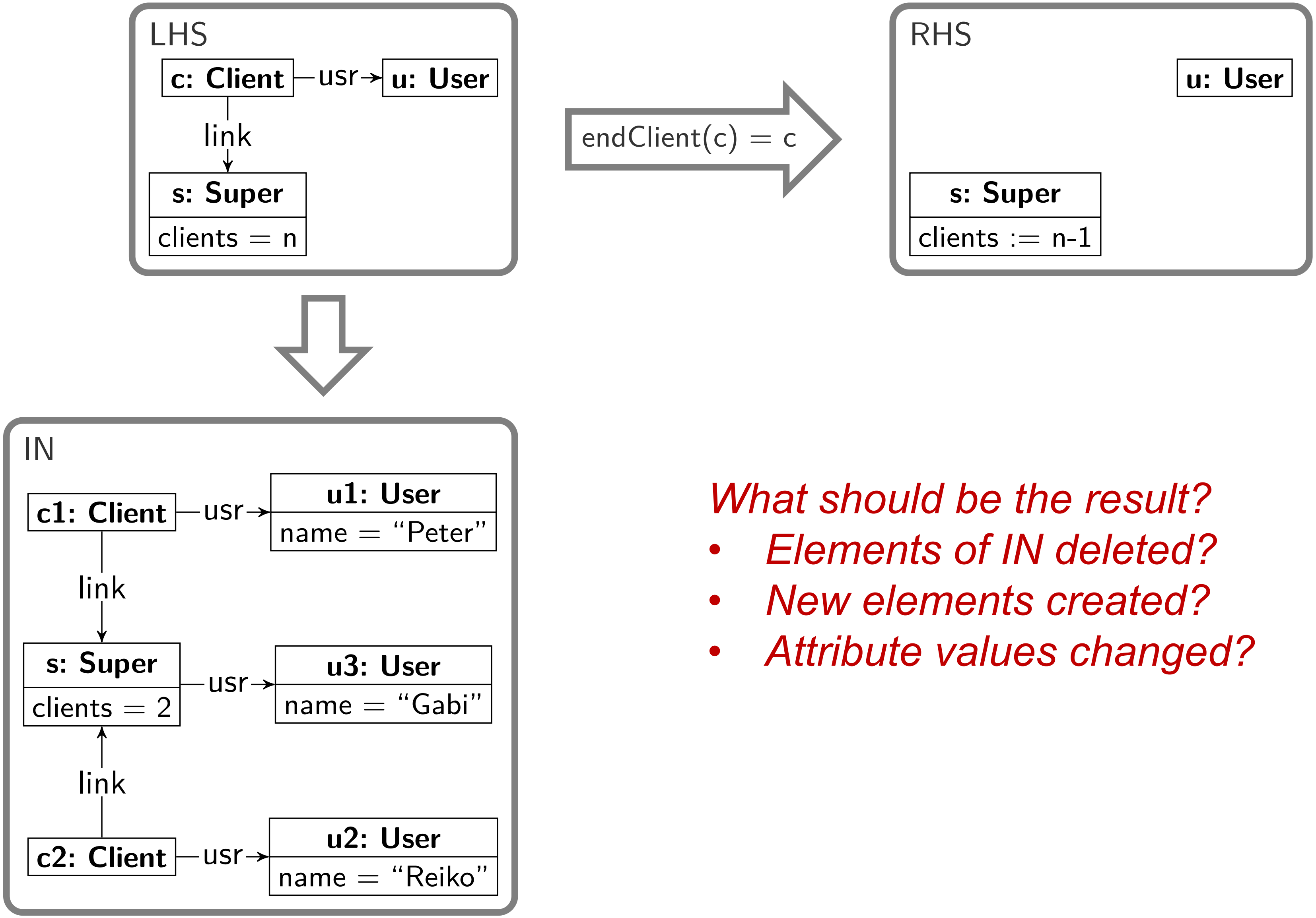
Graph Transformation Concepts





Rule Features

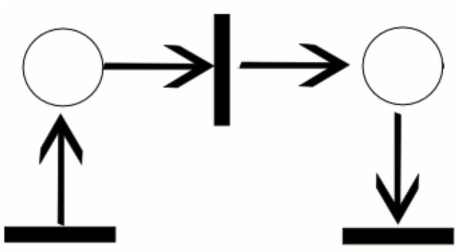
- Left- and right-hand side
- UML-like object notation
  - *c:Client, u: User, s: Super*
- Attribute conditions
  - *clients = n*
- Attribute assignments
  - *clients := n-1*
- Rule signature
  - *endClient(c: Client) : Client*
- Rule label
  - *endClient(c) = c*



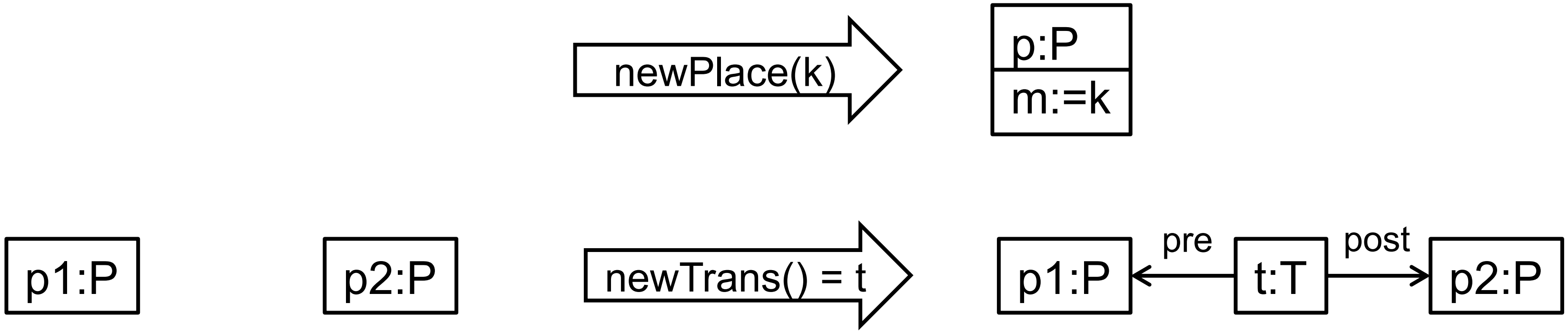
What should be the result?

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

GTBLE: Rules for Nets



Rules to create marked graphs



newPlace(k)

p:P
m:=k

p1:P

p2:P

newTrans() = t

p1:P

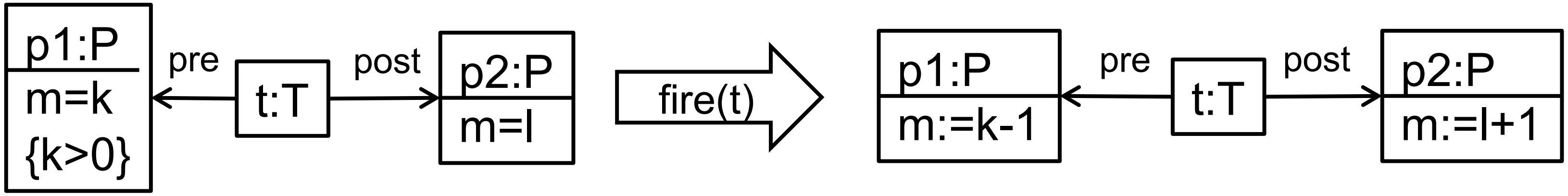
pre

t:T

post

p2:P

Firing rule for marked graphs



p1:P
m=k
{k>0}

pre

t:T

post

p2:P
m=l

fire(t)

p1:P
m:=k-1

pre

t:T

post

p2:P
m:=l+1

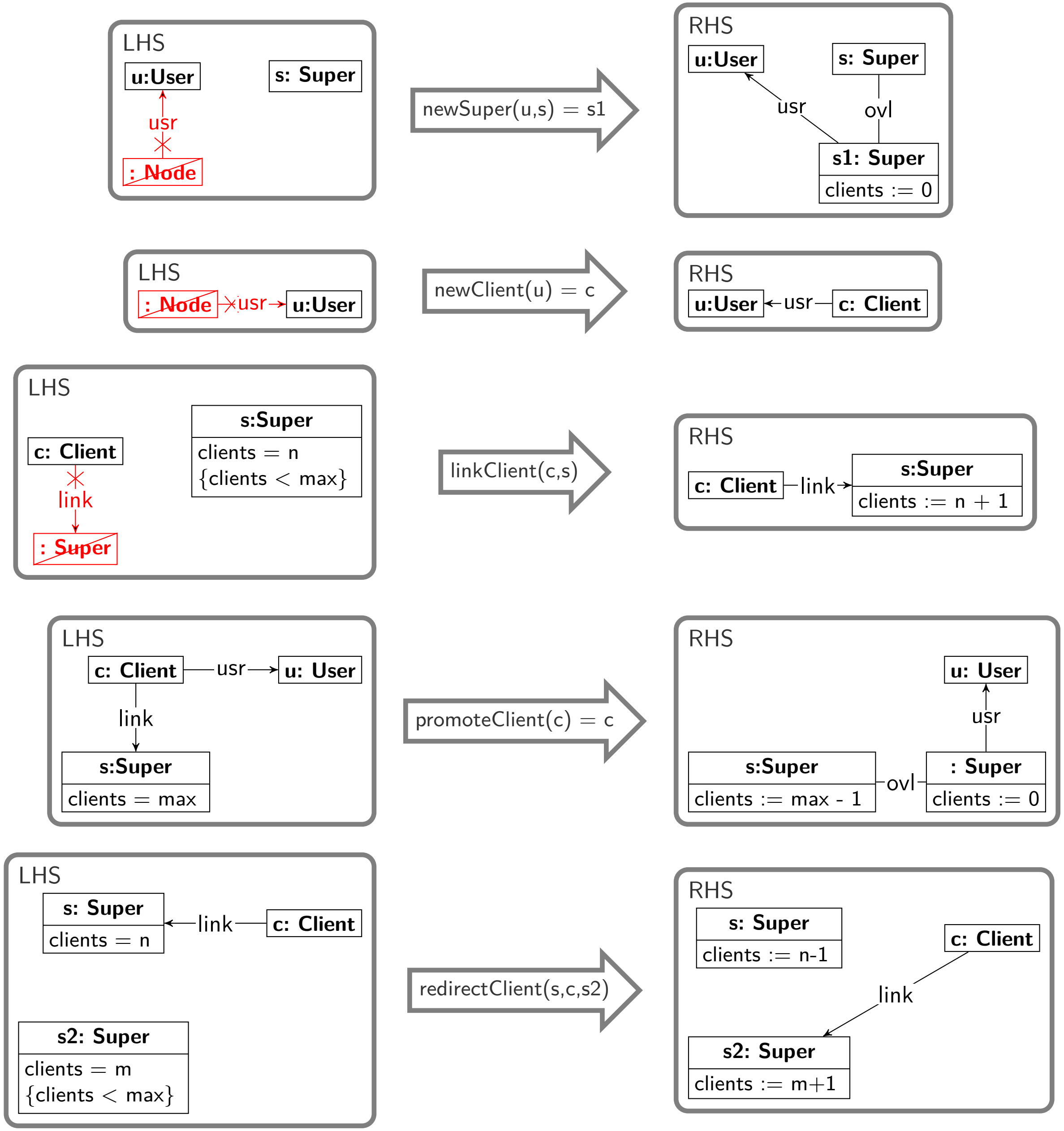
GTBSE: P2P Rules

Network formation

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

Rule Concepts

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



LHS
u:User
s: Super
usr
~~: Node~~
newSuper(u,s) = s1
RHS
u:User
s: Super
usr
ovl
s1: Super
clients := 0

LHS
~~: Node~~
usr
u:User
newClient(u) = c
RHS
u:User
usr
c: Client

LHS
c: Client
s:Super
clients = n
{clients < max}
link
~~: Super~~
linkClient(c,s)
RHS
c: Client
link
s:Super
clients := n + 1

LHS
c: Client
usr
u: User
link
s:Super
clients = max
promoteClient(c) = c
RHS
u: User
usr
s:Super
clients := max - 1
ovl
: Super
clients := 0

LHS
s: Super
clients = n
link
c: Client
s2: Super
clients = m
{clients < max}
redirectClient(s,c,s2)
RHS
s: Super
clients := n-1
c: Client
link
s2: Super
clients := m+1

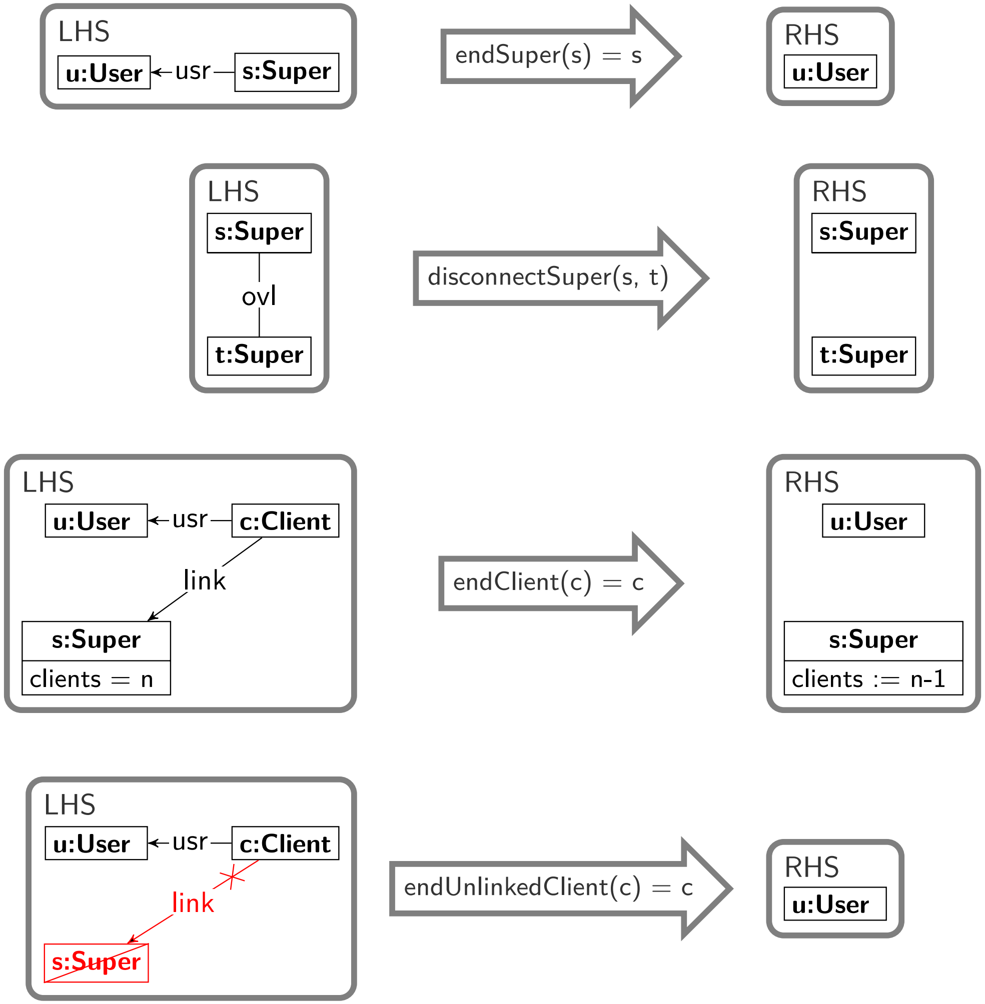
GTBSE: P2P Rules

Network deconstruction

- *Disconnect nodes*
- *Terminate nodes*

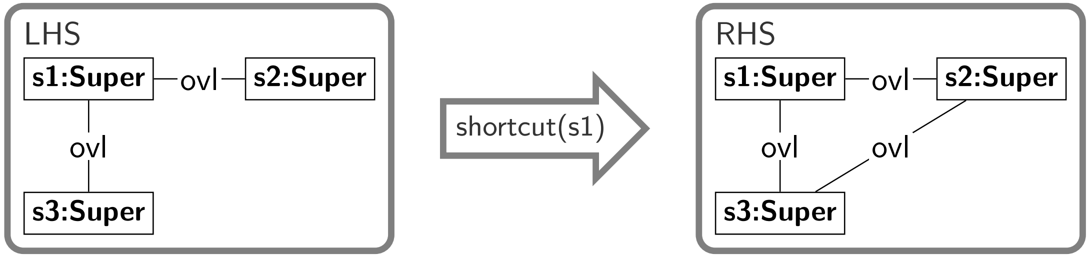
Rule Concepts

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



Global application conditions

What could possibly be wrong with this rule?

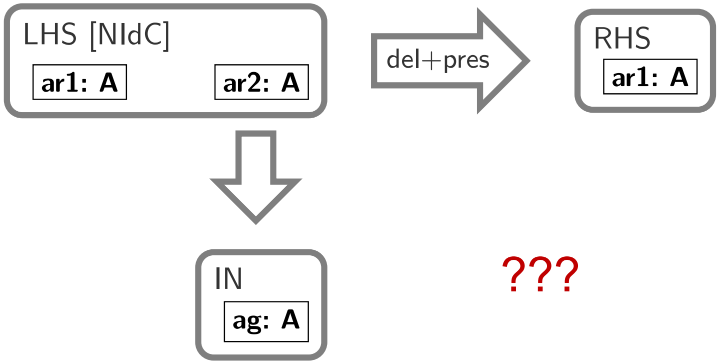


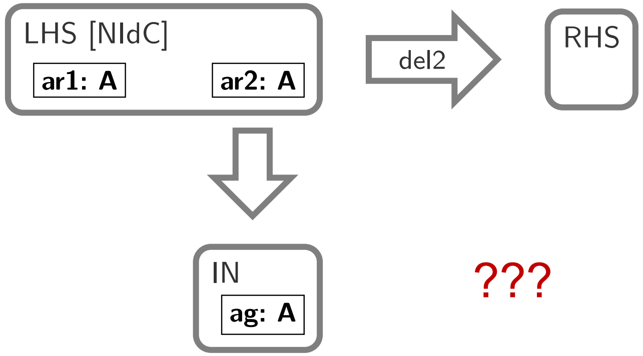
➔ With a match mapping *s2* and *s3* to the same node in the input graph, the rule would create a loop.

Injective matches: Each element in the left-hand side is mapped to a separate element in the input graph.

Global application conditions: Identification condition

Allowing for non-injective matches, what happens here?



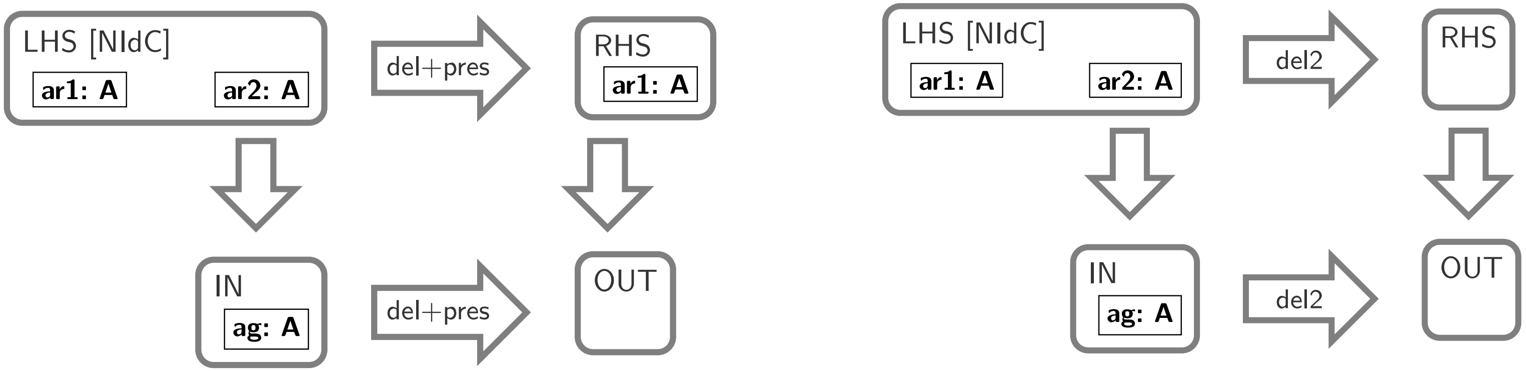


Identification condition: elements to be deleted are kept apart from

other elements and each other

Global application conditions: Identification condition

Allowing for non-injective matches, what happens here?

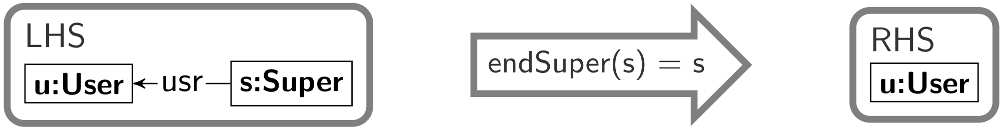


Identification condition: elements to be deleted are kept apart from

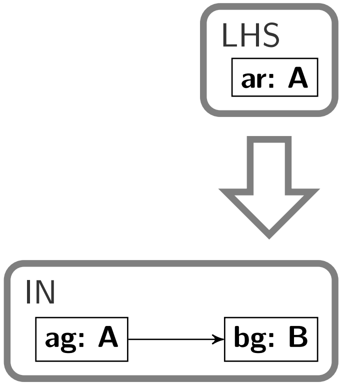
other elements and each other

Global application conditions: Dangling condition

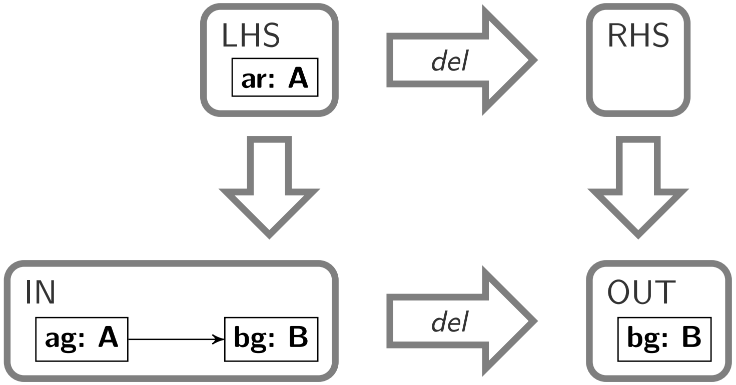
Deletion in unknown contexts



What happens to any connections of s with other Super nodes?



With dangling condition, rule not applicable here!



Global application conditions: Overview

Conservative

$$\begin{tikzcd}
\mathbf{Inj}ective\mathbf{G}luing\mathbf{C}ond \arrow[r] \arrow[d] & \mathbf{G}luing\mathbf{C}ond \arrow[r] \arrow[d] & \mathbf{D}angling\mathbf{C}ond \arrow[dd] \\
\mathbf{Inj}ectivity\mathbf{C}ond \arrow[r] \arrow[d] & \mathbf{Id}entification\mathbf{C}ond \arrow[dl, leftrightarrow] \arrow[d] \arrow[dr] & \\
\mathbf{N}o\mathbf{Inj}ectivity\mathbf{C}ond & \mathbf{N}o\mathbf{Id}entification\mathbf{C}ond \arrow[r, leftrightarrow] & \mathbf{A}ny
\end{tikzcd}$$

Radical

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*

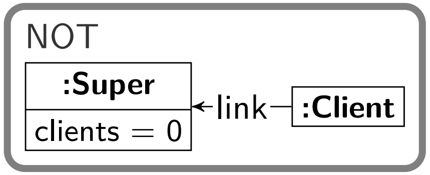
Advanced features: Graph constraints

“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



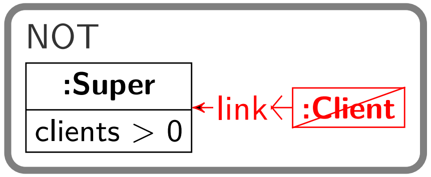
NOT

:Super

clients = 0

link

:Client



NOT

:Super

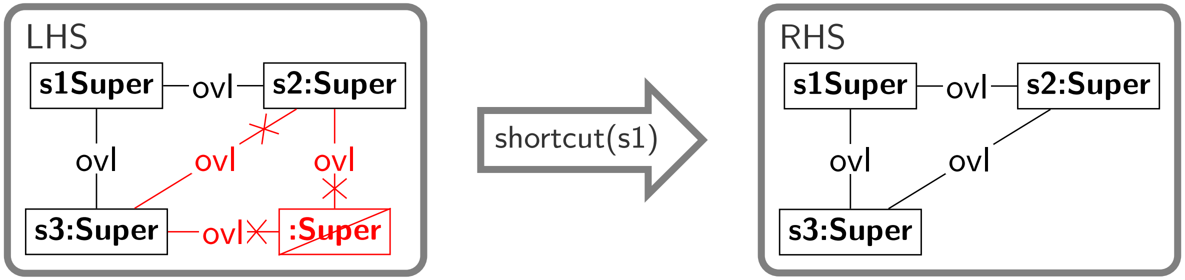
clients > 0

link

~~:Client~~

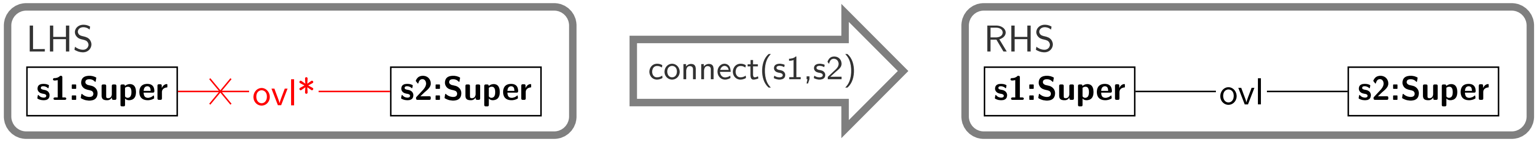
Advanced features: Negative application conditions

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



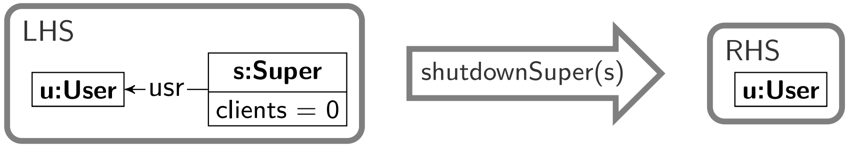
Advanced features: Path Expressions

“Apply connect only if there is no path of *ovl* edges between *s1* and *s2*.”

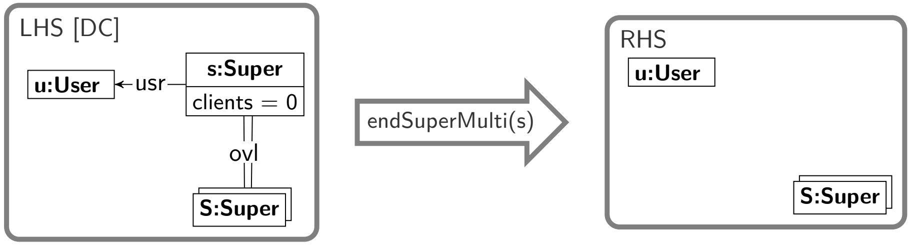


Advanced features: Multi Objects and Patterns

With the dangling condition, we cannot delete a *Super* node without explicitly deleting all links to **all its peers**.

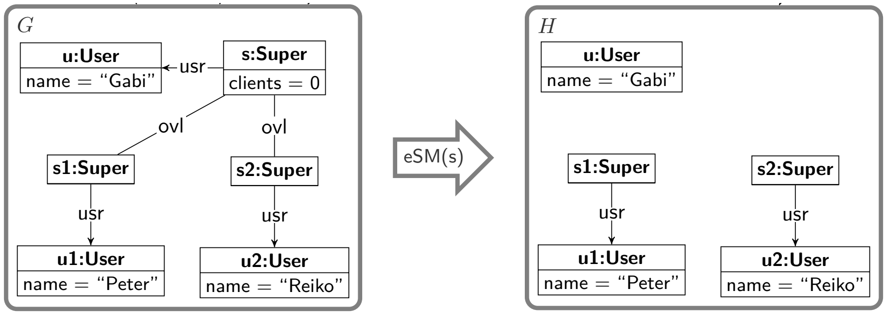


This is possible using a rule with multi object.



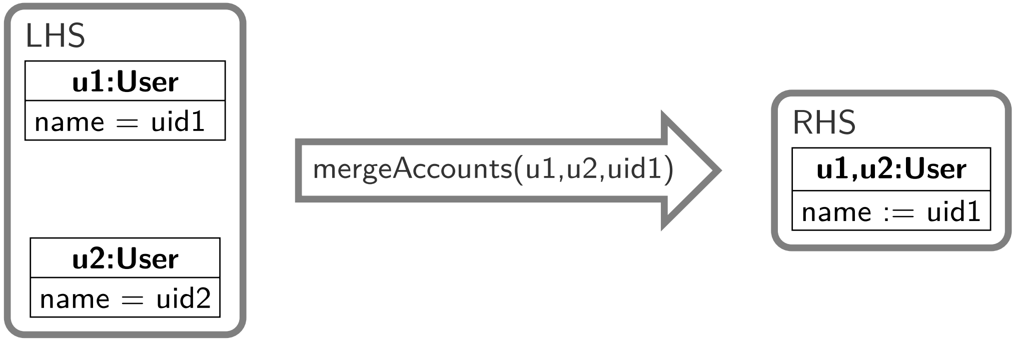
S:Super matches the set of all super nodes linked to s:

{s1, s2}

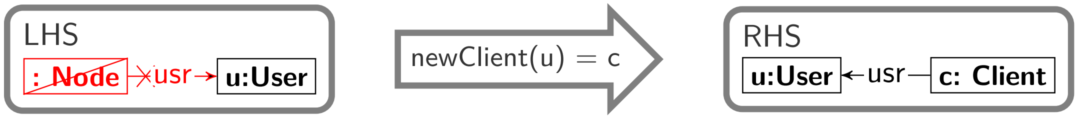


Advanced features: Merging

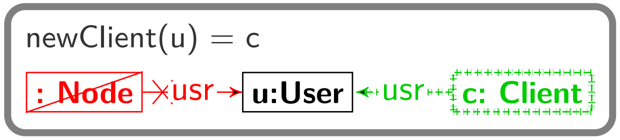
“Merge two user *accounts* into one.”



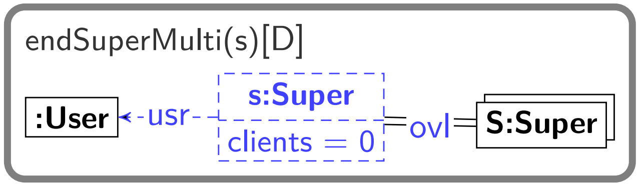
Integrated Rule Notation

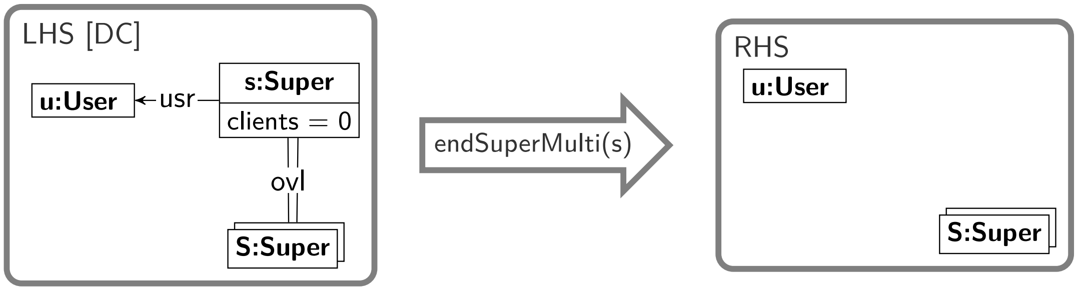


- Integrate left- and right-hand side into a single graph

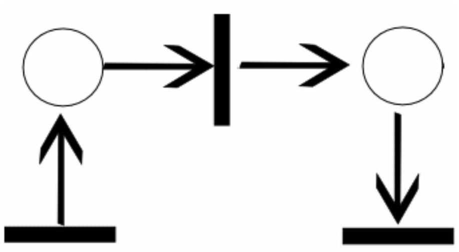


- Use colours and labels to distinguish different roles

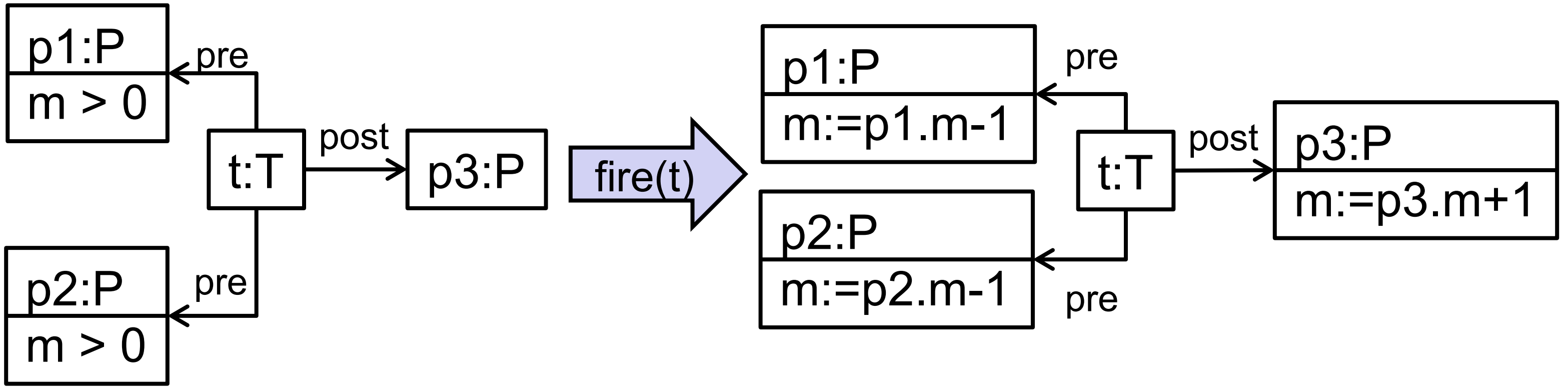




GTBLE: Firing rules for general Petri nets

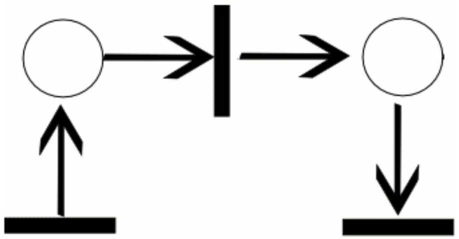


Rule for 2:1 transitions (2 places pre, 1 place post set)

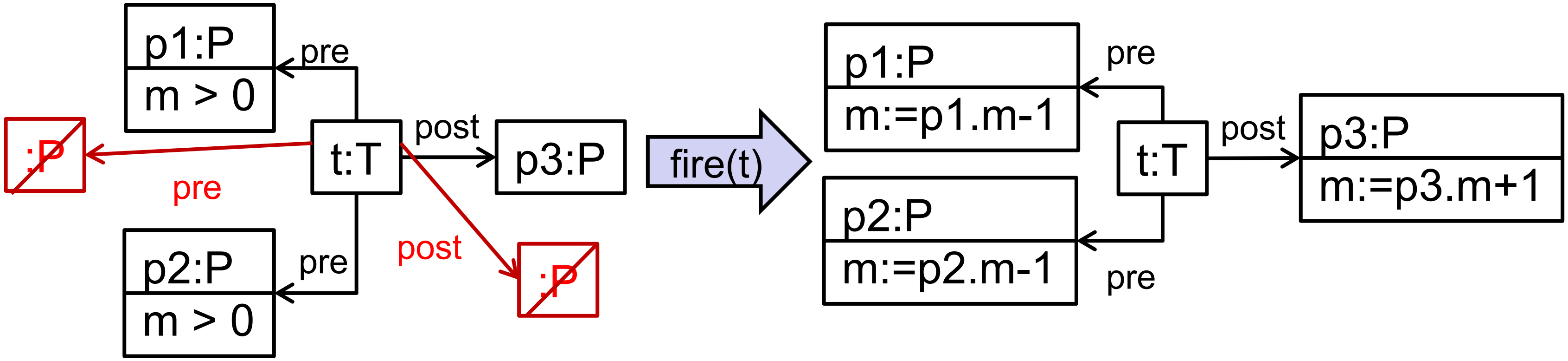


How to ensure this is only applicable to 2:1 transitions?

GTBLE: Firing rules for general Petri nets



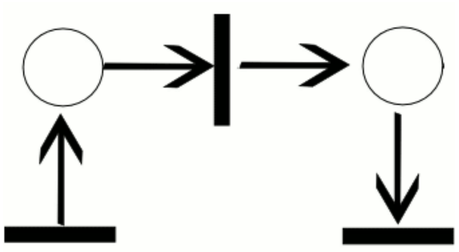
Rule for 2:1 transitions (2 places pre, 1 place post set)



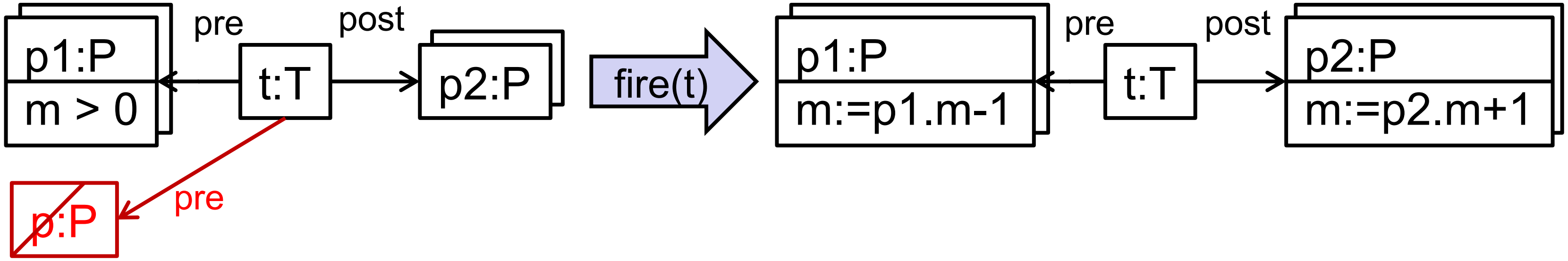
How to ensure this is only applicable to 2:1 transitions?

➔ 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 set 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

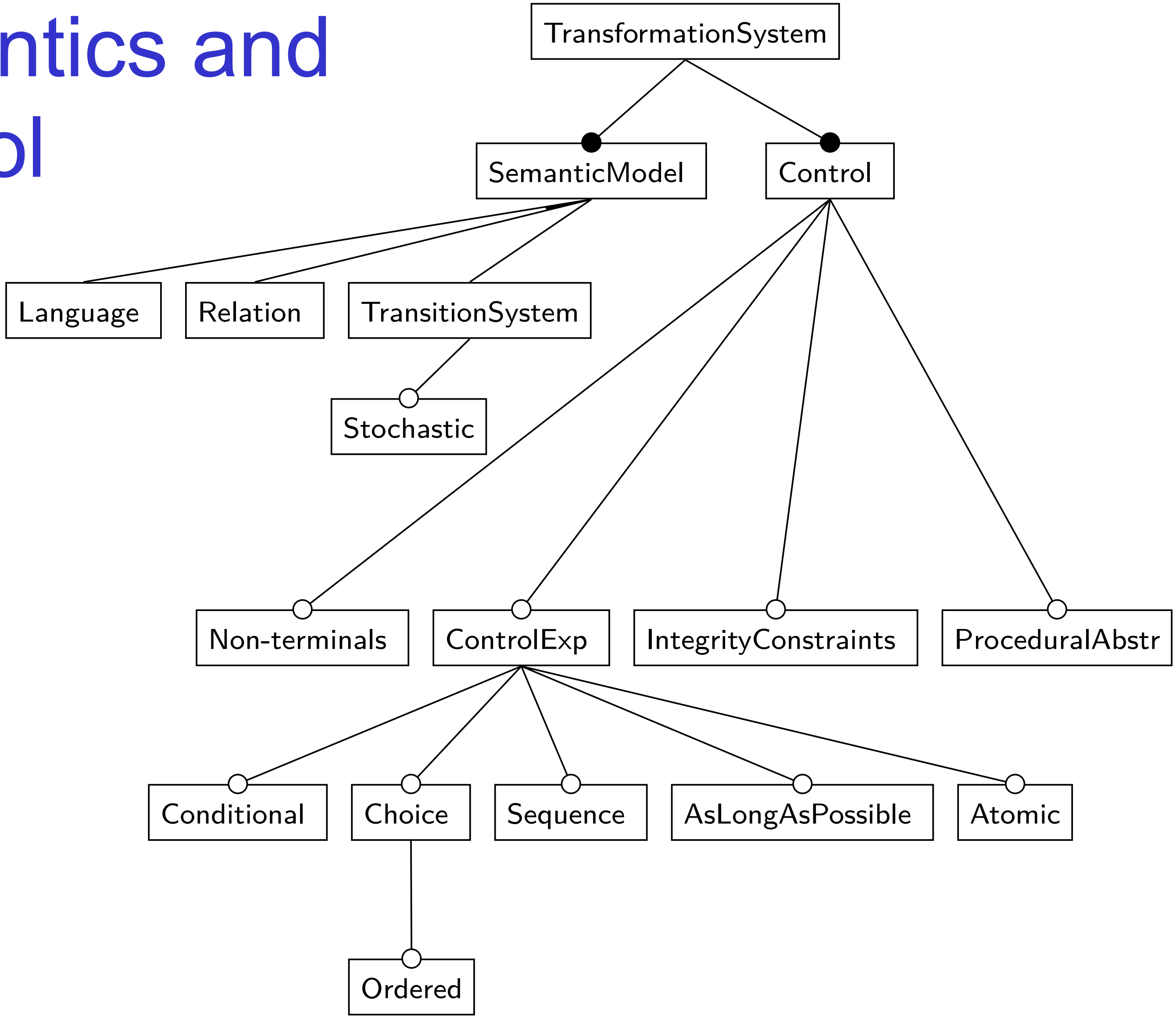
- ➔ *Forbidden matches*
- ➔ *Amalgamation*

GT4SE
Graph Transformation
for Software Engineers

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
- Henshin Tool Demo

Semantics and Control



Transformation systems, grammars, and semantic models

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

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

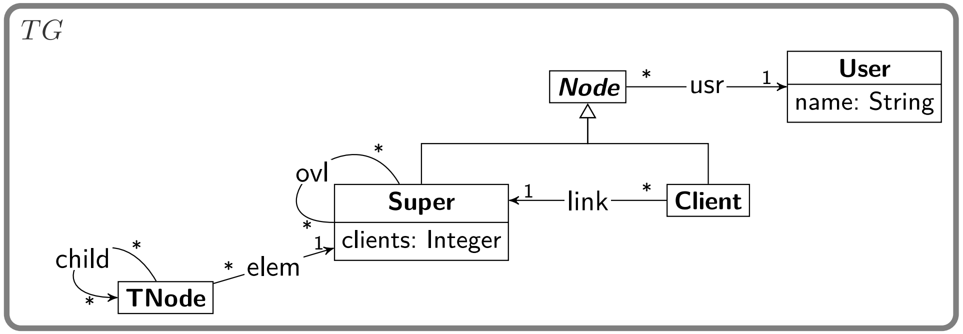
E.g. type graph and rules of the P2P model with start graph as initial network configuration

Semantic Models

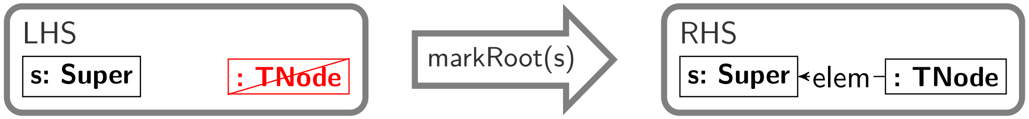
- Language: set of all graphs reachable from G_0
- Relation: mapping from input to output graphs
- Transition system: reachable graphs as states, transformation steps as transitions

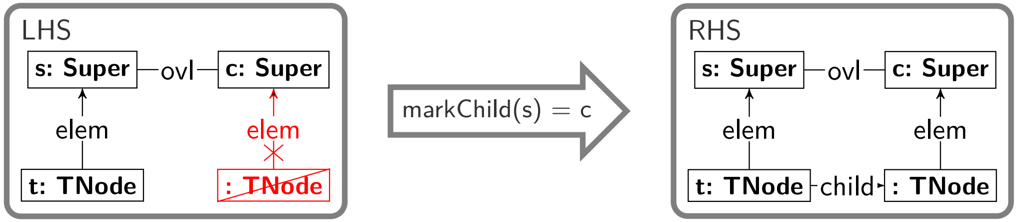
Relation: Compute Spanning Tree of Super Nodes

Extended type graph



Rules to mark root and children of marked nodes



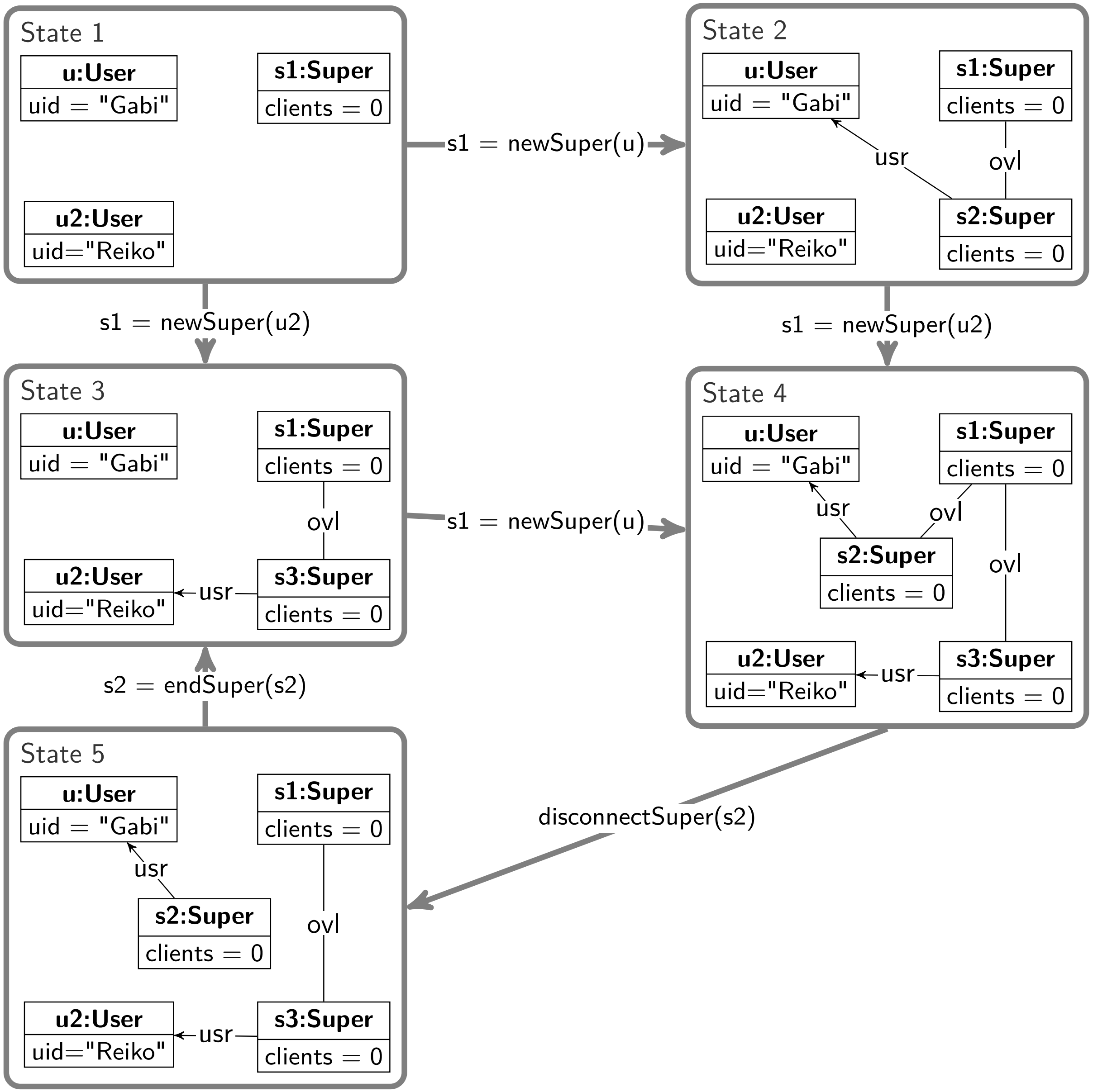


Initial set *I*: all unmarked instances graph of *TG*

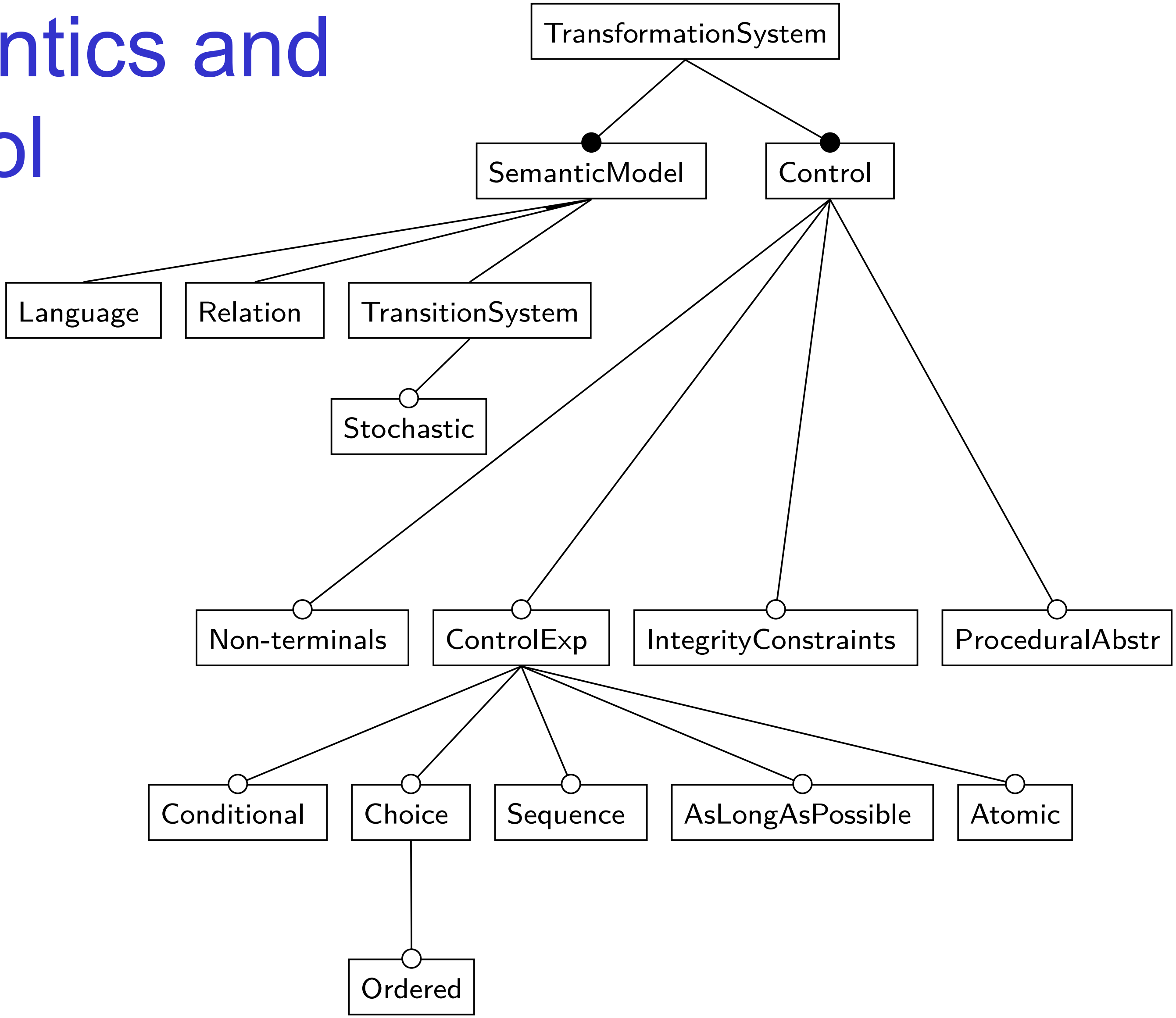
Final set *F*: all graphs where all *Super* nodes are marked

P2P LTS (partial)

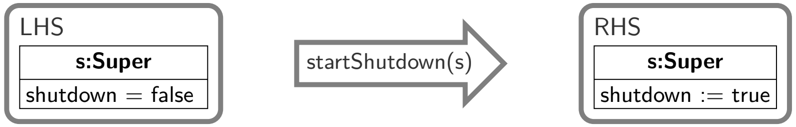
- Graphs as states
- Steps as transitions
- Rule names with args as labels



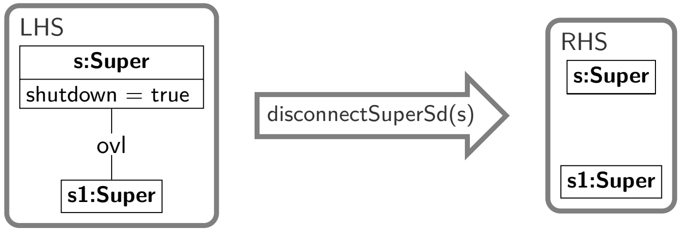
Semantics and Control



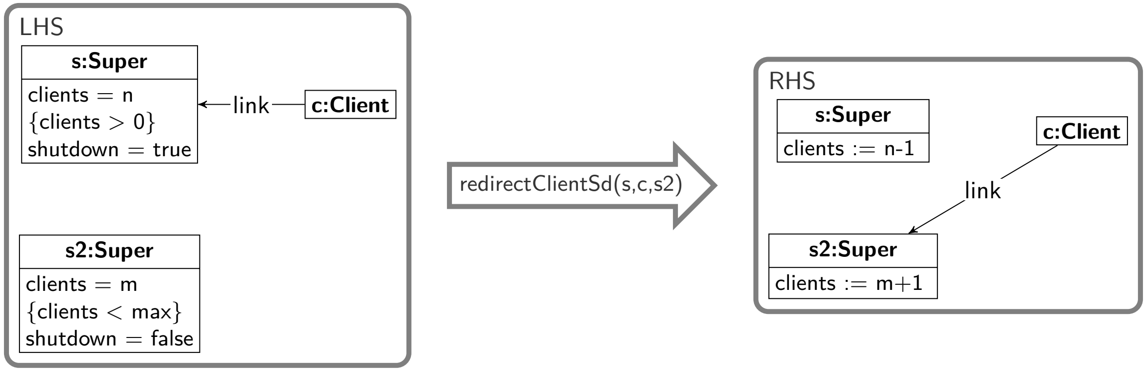
Rule-based programming: textual

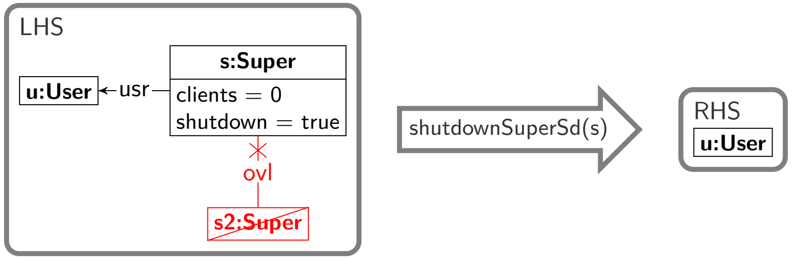


Control expressions,
Procedural abstraction, …

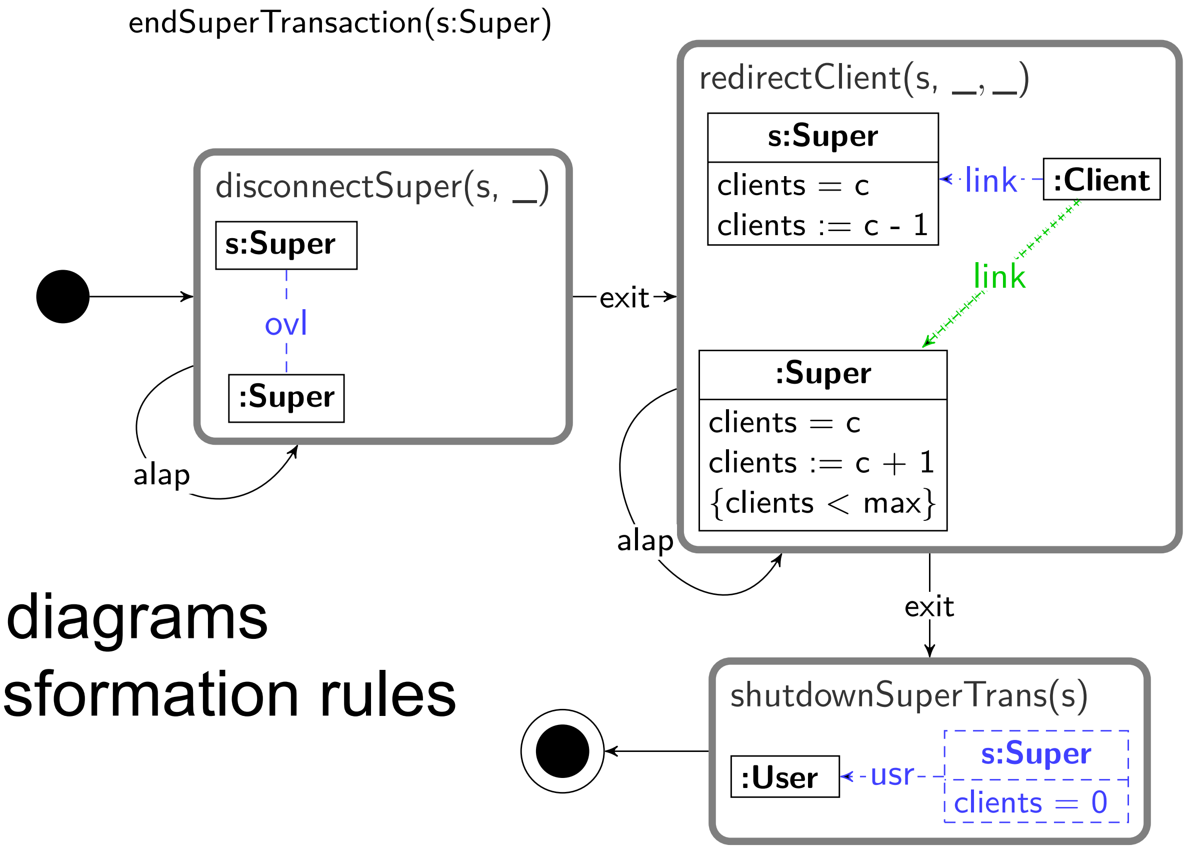


```
unit endSuper(s1: Super) {
    atomic {
        alap {
            disconnectSuper(s1,_)
        }
        alap {
             redirectClient (s1,_ ,_)
        }
        shutdownSuper(s1)
    }
}
```





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*
- Henshin Tool Demo

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

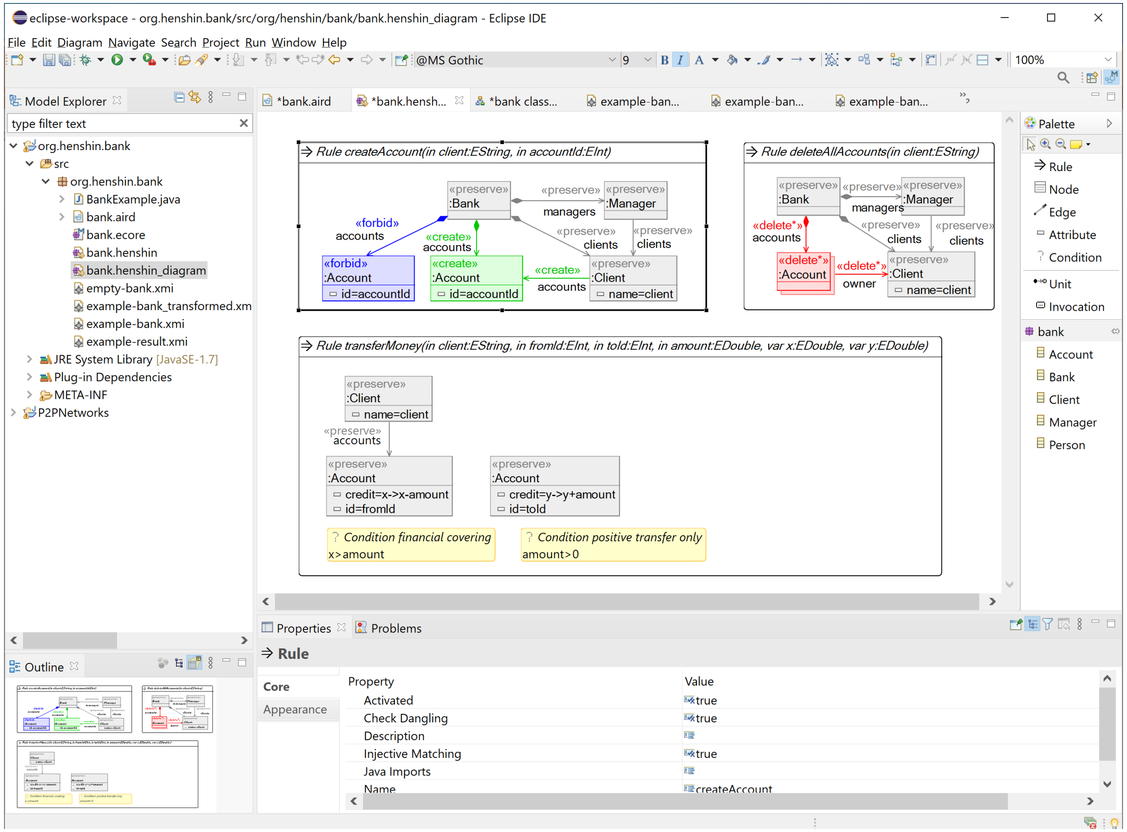
- Conflict and Dependency Analysis
- Termination Analysis
- Constraint Verification and Enforcement

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



Property	Value
Activated	true
Check Dangling	true
Description	
Injective Matching	true
Java Imports	
Name	createAccount

EMF model transformation

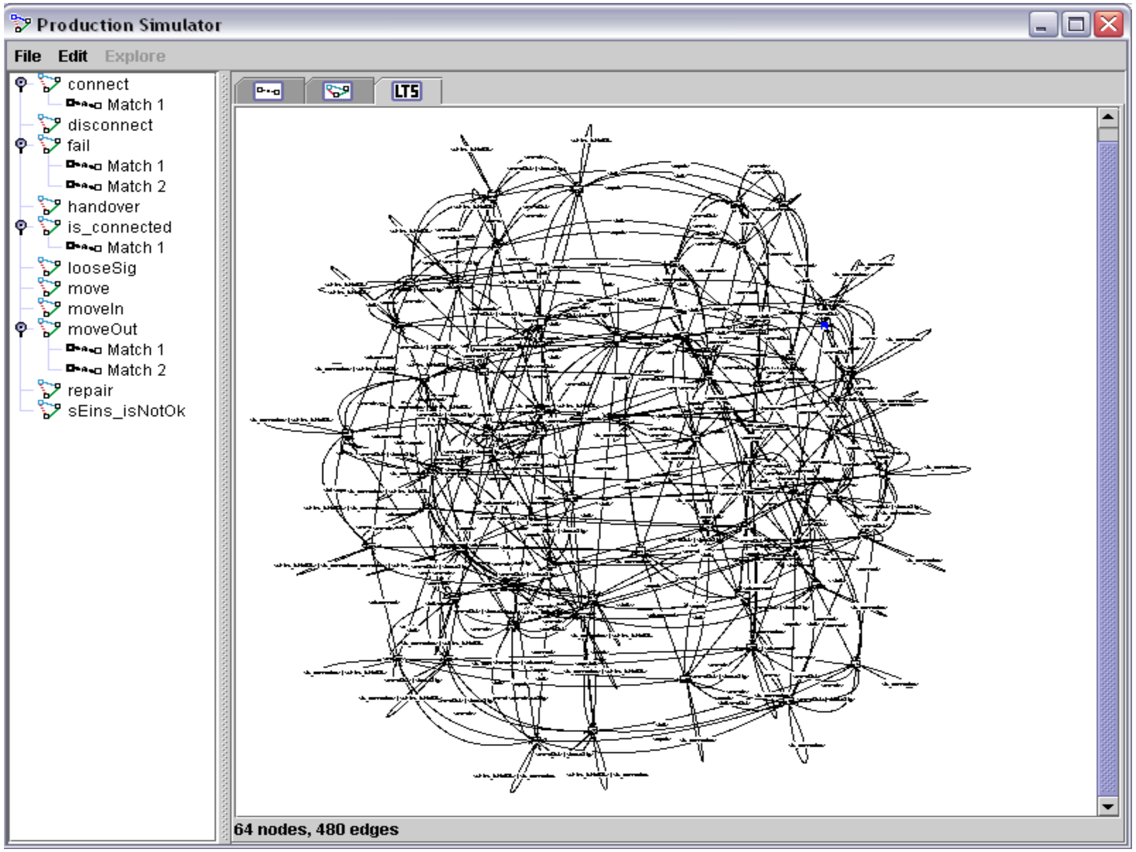
Conflict and dependency analysis

Constraint verification

Model checking by translation

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

Tools: Groove



64 nodes, 480 edges

Graph transformation

Visualisation of rules and matches

Native model checking in CTL and LTL

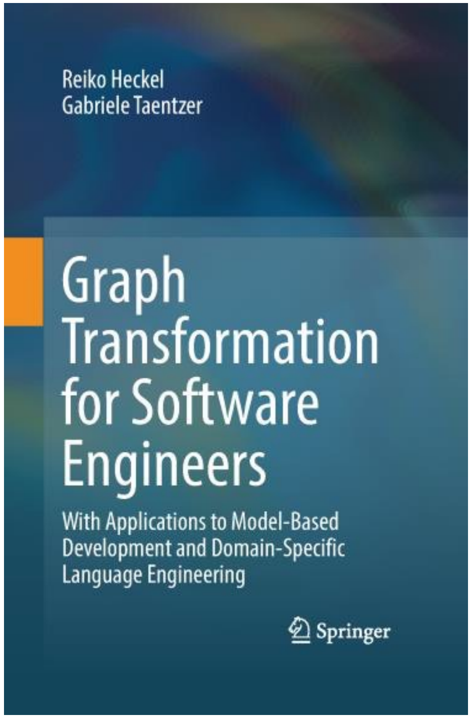
https://groove.ewi.utwente.nl

Part 1: Foundations

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

Further Reading

GT4SE
Graph Transformation
for Software Engineers



Reiko Heckel
Gabriele Taentzer

Graph Transformation for Software Engineers

With Applications to Model-Based Development and Domain-Specific Language Engineering

Springer

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