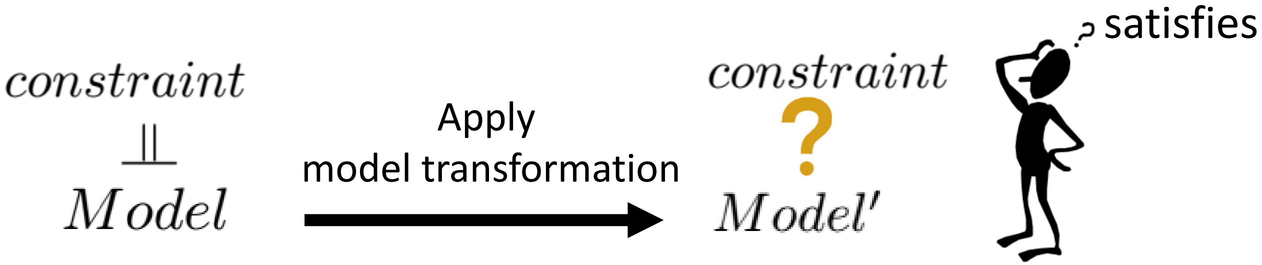
OCL2AC
Automatic Translation of OCL Constraints to Graph Constraints and Application Conditions for Transformation Rules

Nebras Nassar, Jens Kosiol, Thorsten Arendt, and Gabriele Taentzer

Philipps-Universität, Marburg, Germany
GFFT Innovationsförderung GmbH, Bad Vilbel, Germany

June 25, 2018

Introduction



$constraint$

$\models$

$Model$

Apply
model transformation

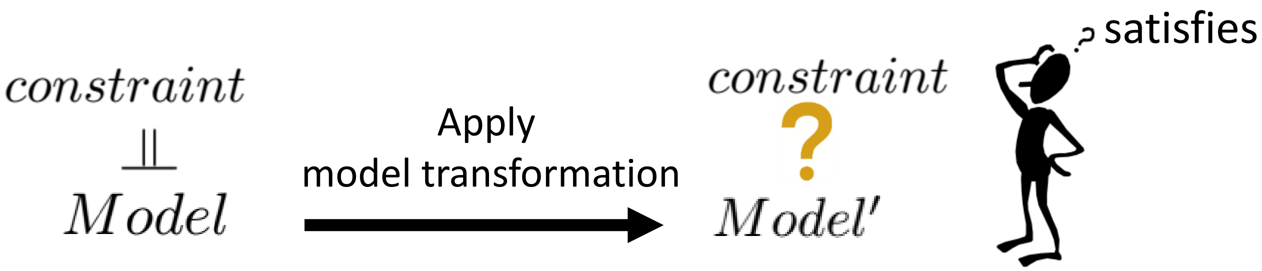
$constraint$

?

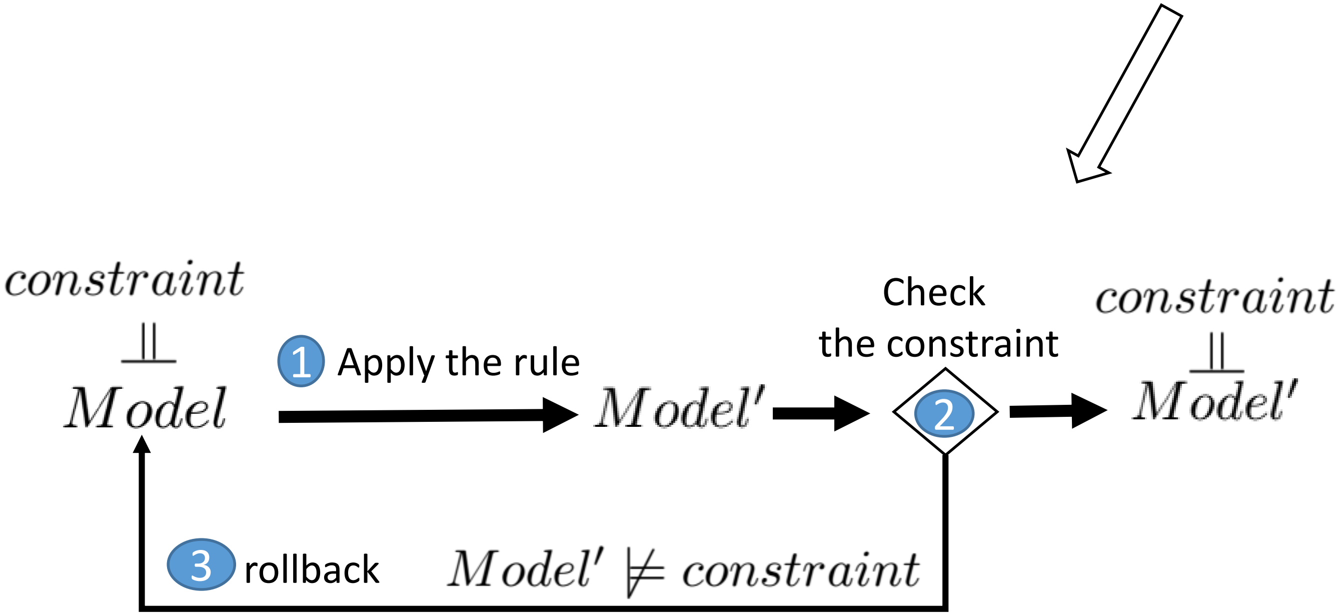
$Model'$

? satisfies

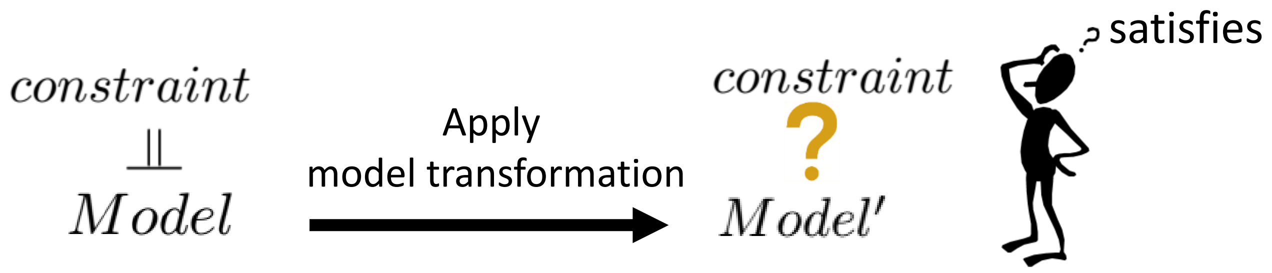
Introduction



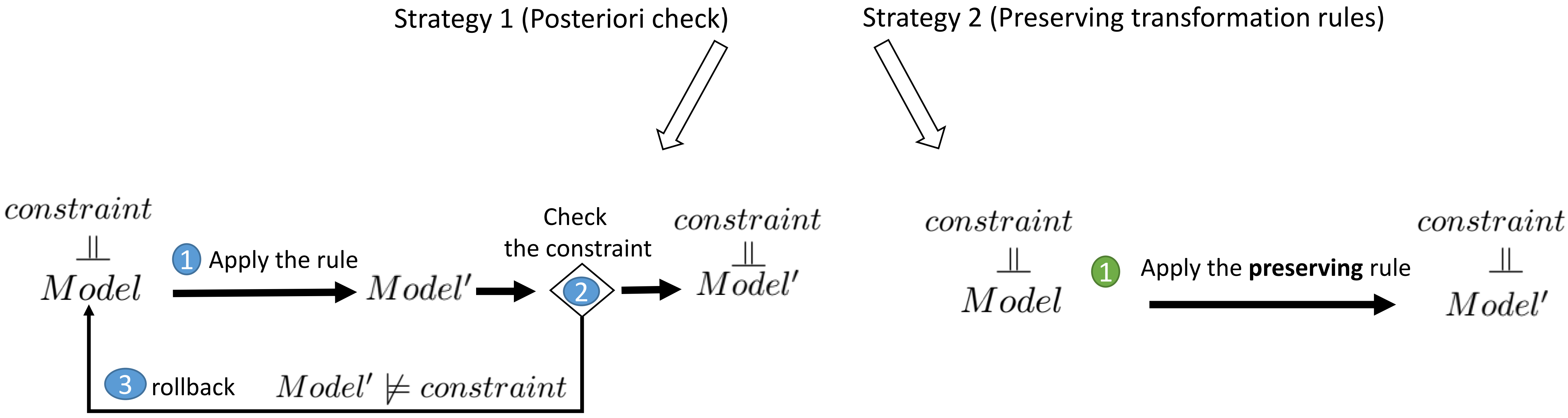
Strategy 1 (Posteriori check)



Introduction



$\begin{array}{c} constraint \\ \rotatebox{90}{$\models$} \\ Model \end{array}$ $\xrightarrow{\text{Apply model transformation}}$ $\begin{array}{c} constraint \\ ? \\ Model' \end{array}$



Strategy 1 (Posteriori check)

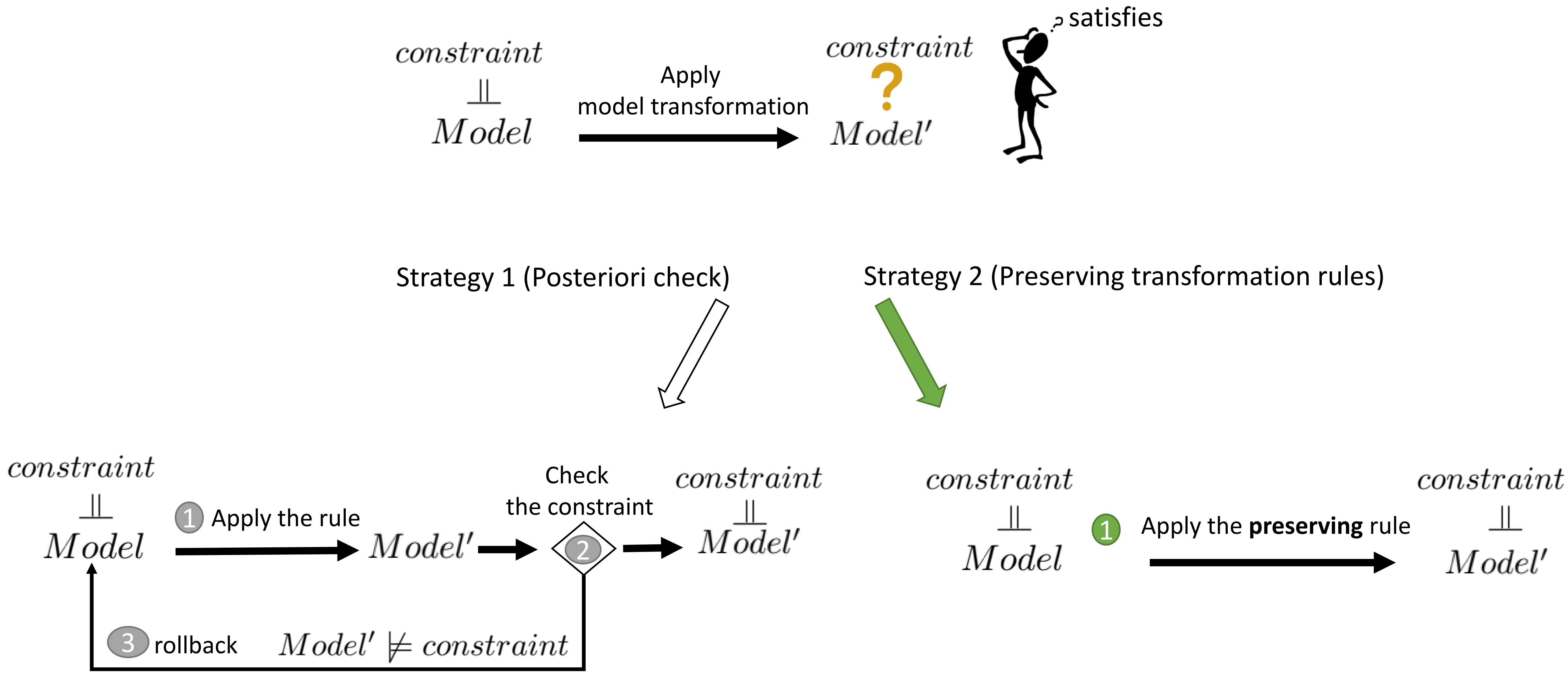
$\begin{array}{c} constraint \\ \rotatebox{90}{$\models$} \\ Model \end{array}$ 1 Apply the rule $\rightarrow$ $Model' \rightarrow$ 2 Check the constraint $\rightarrow$ $\begin{array}{c} constraint \\ \rotatebox{90}{$\models$} \\ Model' \end{array}$

3 rollback $Model' \not\models constraint$

Strategy 2 (Preserving transformation rules)

$\begin{array}{c} constraint \\ \rotatebox{90}{$\models$} \\ Model \end{array}$ 1 Apply the **preserving** rule $\rightarrow$ $\begin{array}{c} constraint \\ \rotatebox{90}{$\models$} \\ Model' \end{array}$

Introduction



$constraint$
$\models$
$Model$

Apply
model transformation

$constraint$
?
$Model'$

satisfies

Strategy 1 (Posteriori check)

Strategy 2 (Preserving transformation rules)

$constraint$
$\models$
$Model$

1 Apply the rule

$Model'$

Check
the constraint

2

$constraint$
$\models$
$Model'$

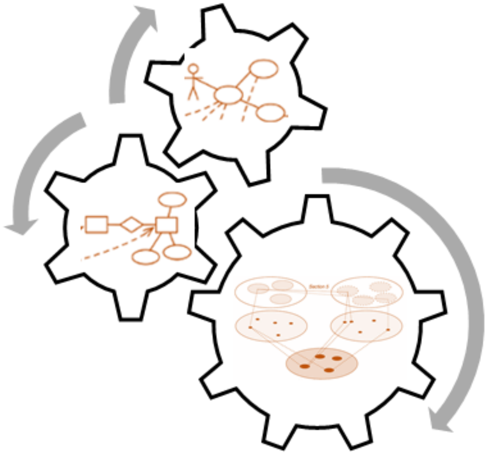
3 rollback $Model' \not\models constraint$

$constraint$
$\models$
$Model$

1 Apply the **preserving** rule

$constraint$
$\models$
$Model'$

Challenge



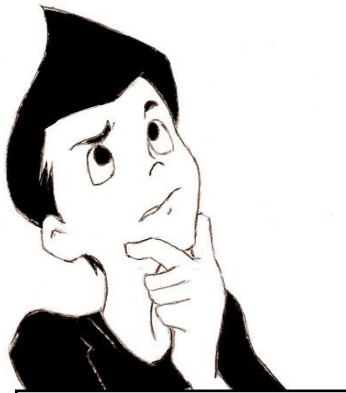
Transformation Rules

Set of Constraints

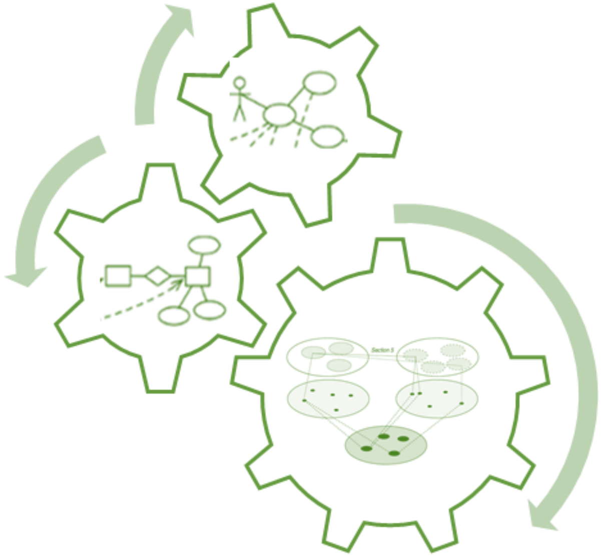
context A **invariant**:
self.b()-> size()>=1;

$\exists (A, \nexists$

$\forall (A, \exists B)$

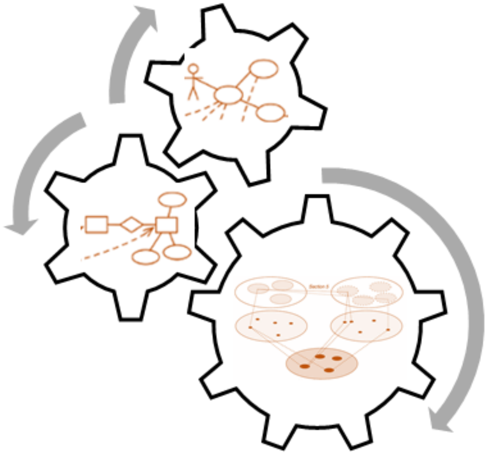


How can we automatically update model transformations to preserve a given set of constraints?



Constraints-preserving Transformation Rules

Contribution



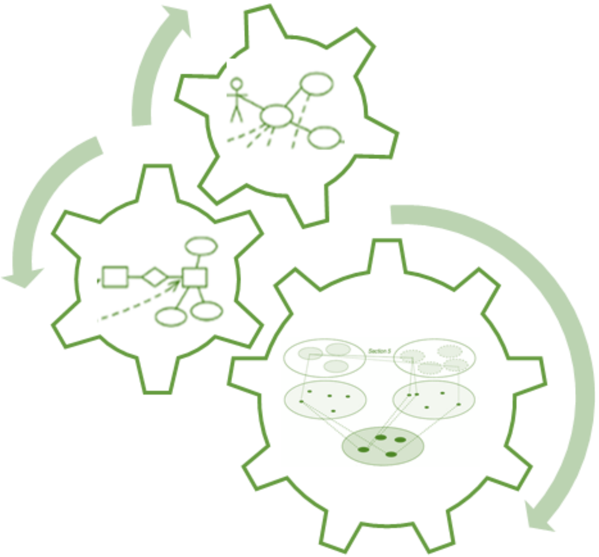
Transformation Rules

Set of Constraints

```
context A invariant:
self.b()-> size()>=1;
```

$\exists\,(A, \nexists$

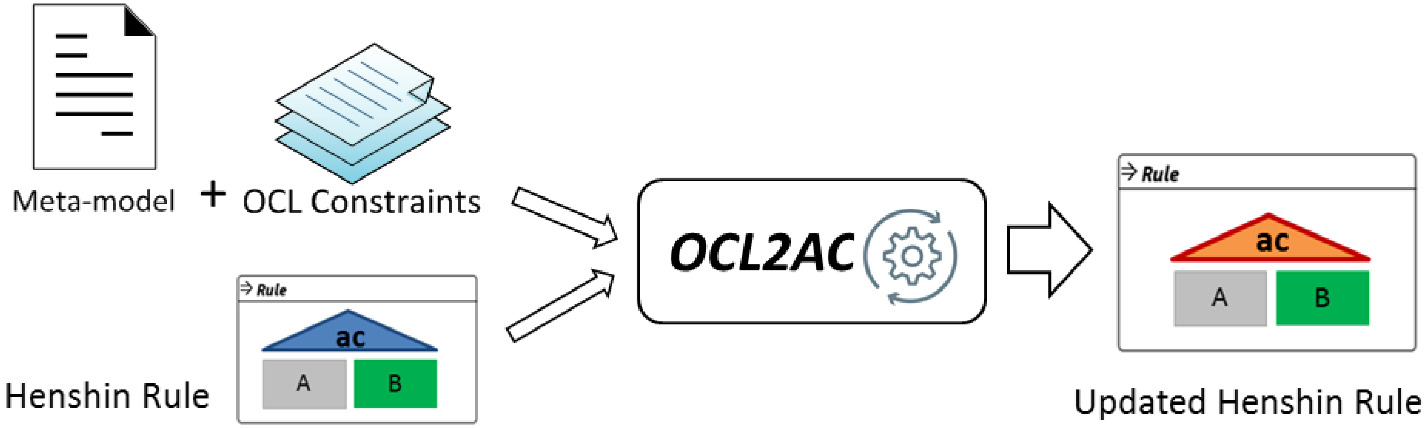
$\forall\,(A, \exists B)$



Constraints-preserving Transformation Rules

Contribution

Based on existing theory [1, 2] we developed a tool, called ***OCL2AC***, which automatically adapts a given rule-based model transformation such that resulting models do not violate a given set of constraints



OCL2AC is an Eclipse plugin which relies on: EMF, OCL and the Henshin language

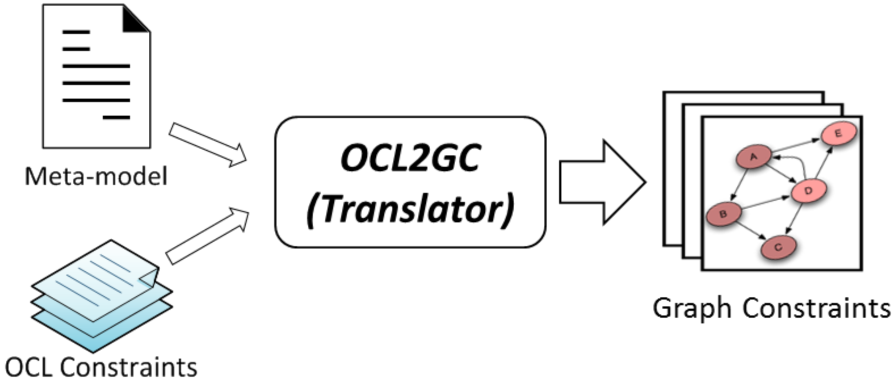
[1] Radke, H., Arendt, T., Becker, J.S., Habel, A., Taentzer, G.: **Translating essential OCL invariants to nested graph constraints for generating instances of metamodels**. Science of Computer Programming 152, 38 - 62 (2018)

[2] Habel, A., Pennemann, K.H.: **Correctness of high-level transformation systems relative to nested conditions**. Mathematical Structures in Computer Science 19, 245-296 (2009)

OCL2AC: Overview

❑ **Two main components:**

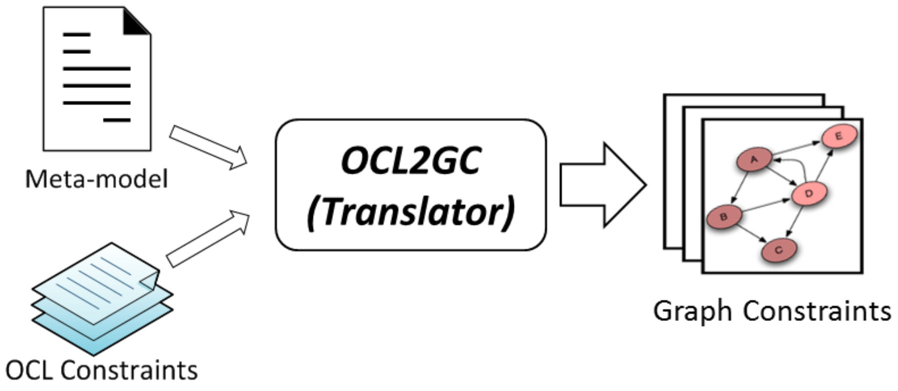
- **OCL2GC: Translate OCL constraints to graph constraints**



OCL2AC: Overview

❑ **Two main components:**

- **OCL2GC: Translate OCL constraints to graph constraints**



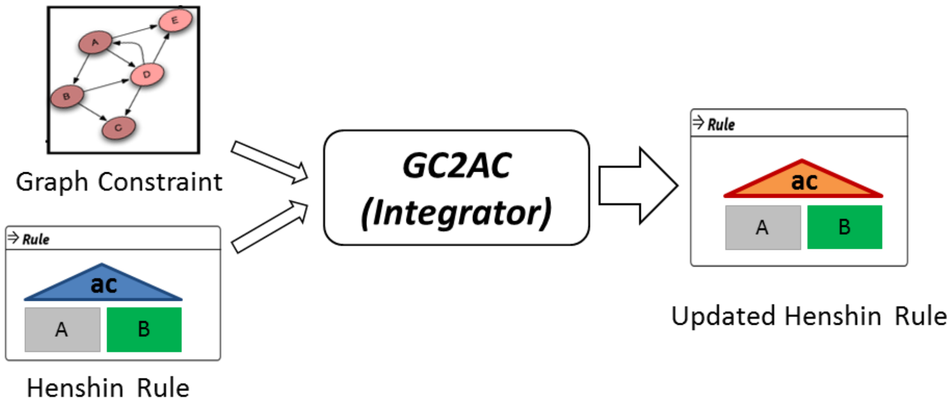
Meta-model

OCL Constraints

OCL2GC (Translator)

Graph Constraints

- **GC2AC: Integrate graph constraints as application conditions**



Graph Constraint

Henshin Rule

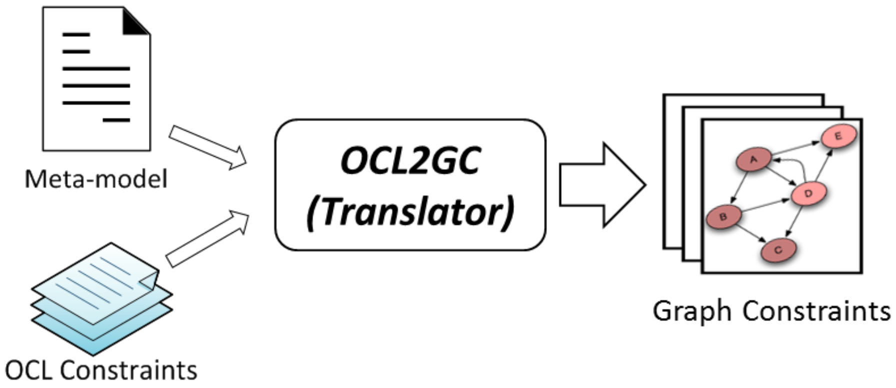
GC2AC (Integrator)

Updated Henshin Rule

OCL2AC: Overview

- **Two main components:**

- **OCL2GC: Translate OCL constraints to graph constraints**



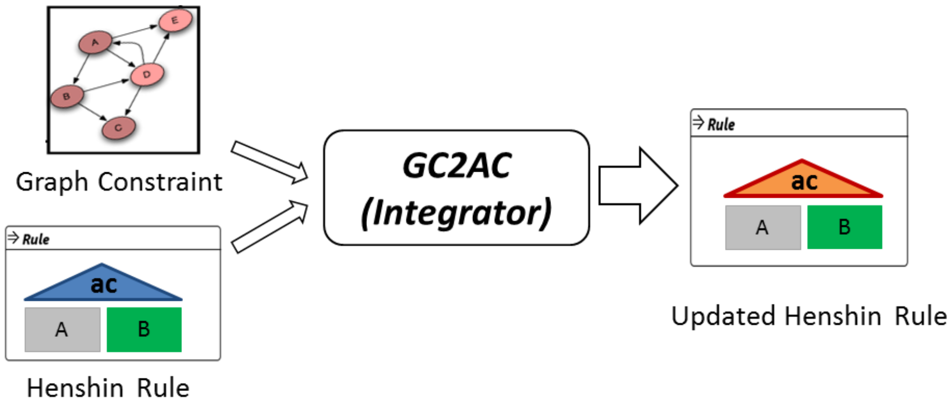
Meta-model

OCL Constraints

OCL2GC (Translator)

Graph Constraints

- **GC2AC: Integrate graph constraints as application conditions**



Graph Constraint

Henshin Rule

GC2AC (Integrator)

Updated Henshin Rule

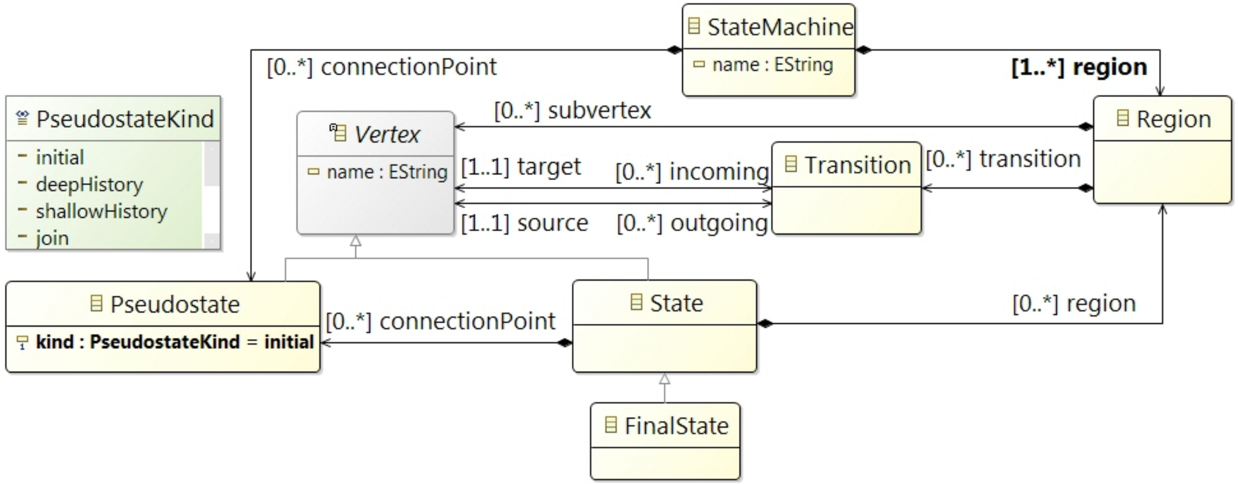
- Each component is designed to be usable on its own (as Eclipse plugins)
- Limitation: The theory beyond the tool considers OCL constraints corresponding to a first-order, two-valued logic and sets as the only collection type

Agenda

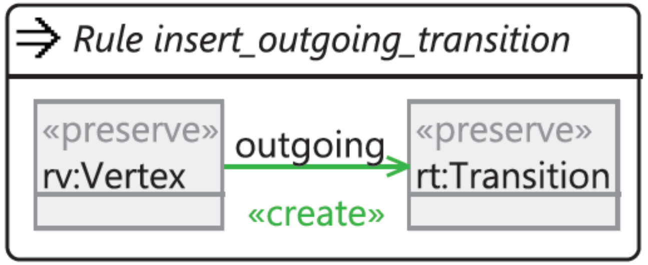
1. Introduction
2. Challenge and contribution
3. OCL2AC overview
4. Agenda
5. Scenario for presenting the tool architecture and functionalities
    - Running example
    - OCL2GC: Translate OCL constraints to graph constraints
    - GC2AC: Integrate graph constraints as application conditions
6. Future work: Simplifications of application conditions
7. Demo
8. Conclusion

Running Example

- Meta-model: A simple Statechart



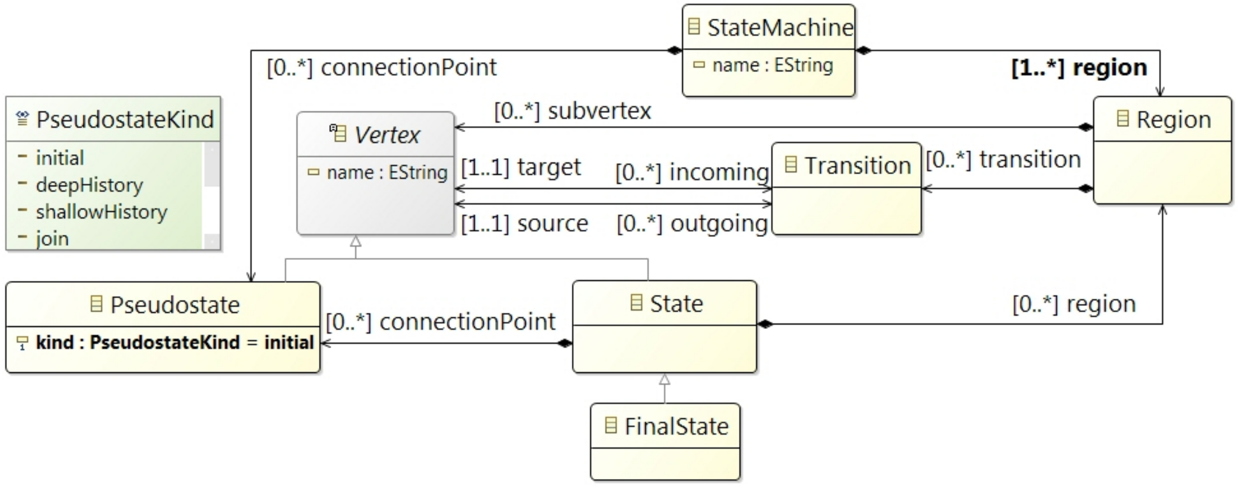
- Editing rules



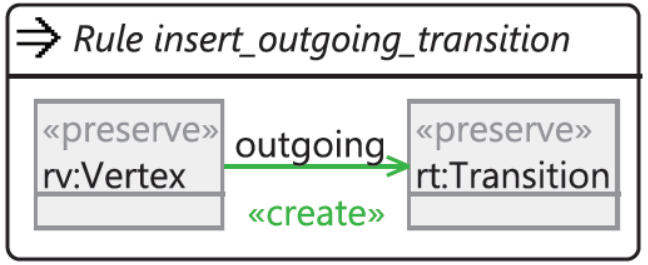
Henshin rule: Insert_outgoing_transition

Running Example

- ❑ Meta-model: A simple Statechart

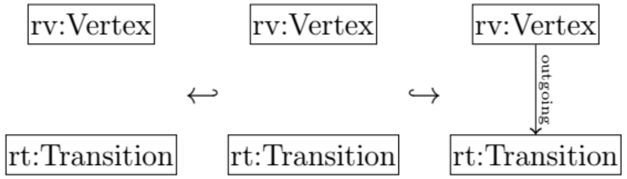


- ❑ Editing rules



Henshin rule: Insert_outgoing_transition

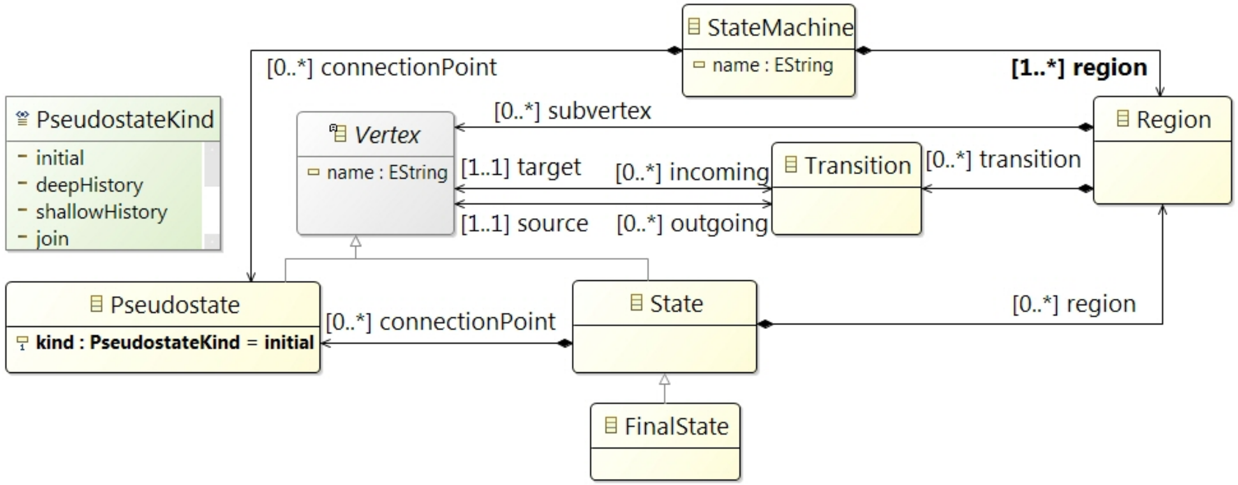
$$L \hookleftarrow K \hookrightarrow R$$



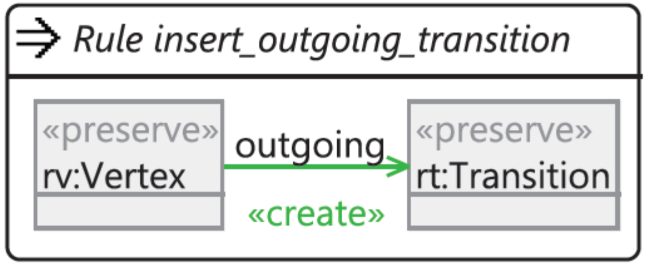
Insert_outgoing_transition rule (Formally)

Running Example

- Meta-model: A simple Statechart

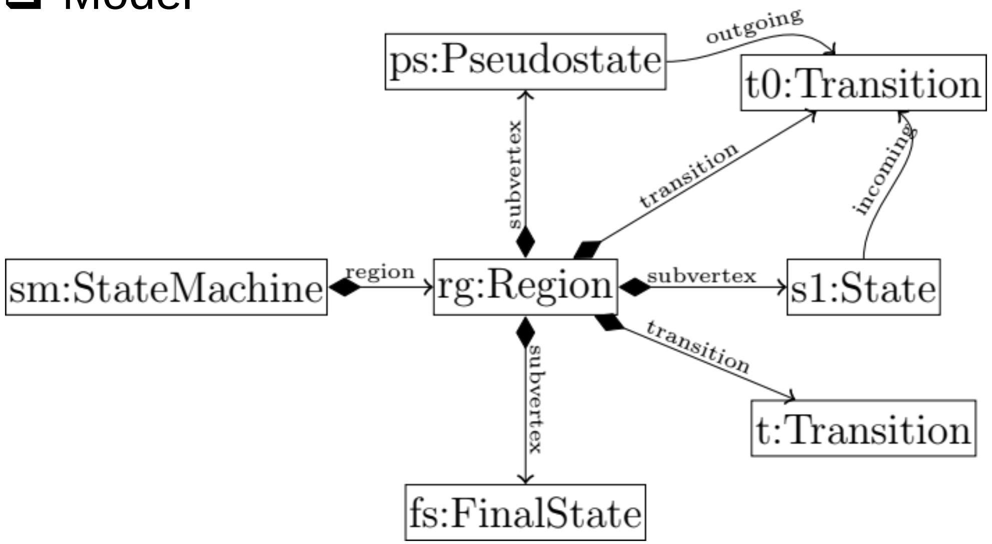


- Editing rules



Henshin rule: Insert_outgoing_transition

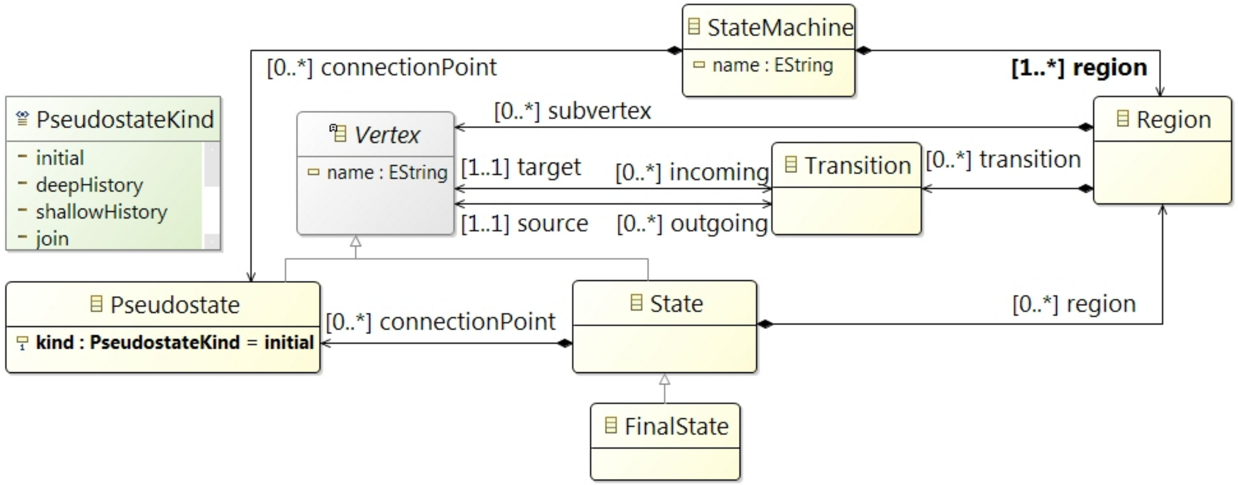
- Model



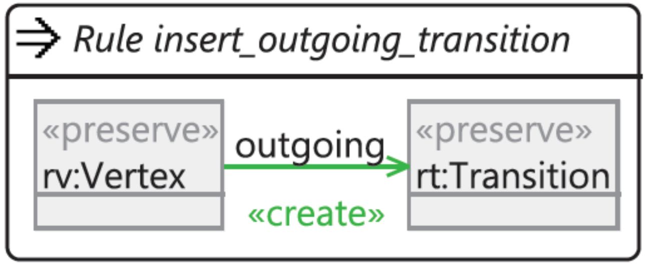
The abstract syntax of the state machine

Running Example

- ❑ Meta-model: A simple Statechart

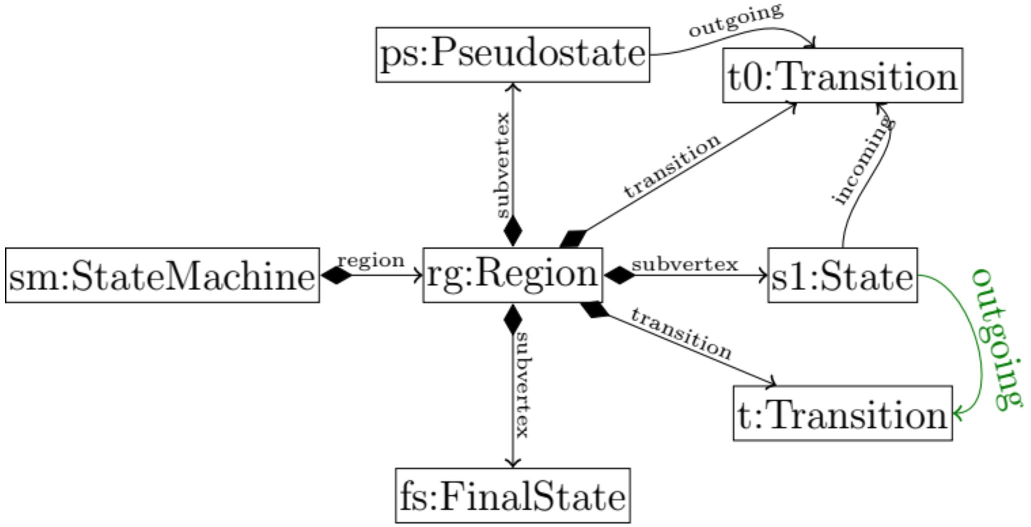


- ❑ Editing rules



Henshin rule: Insert_outgoing_transition

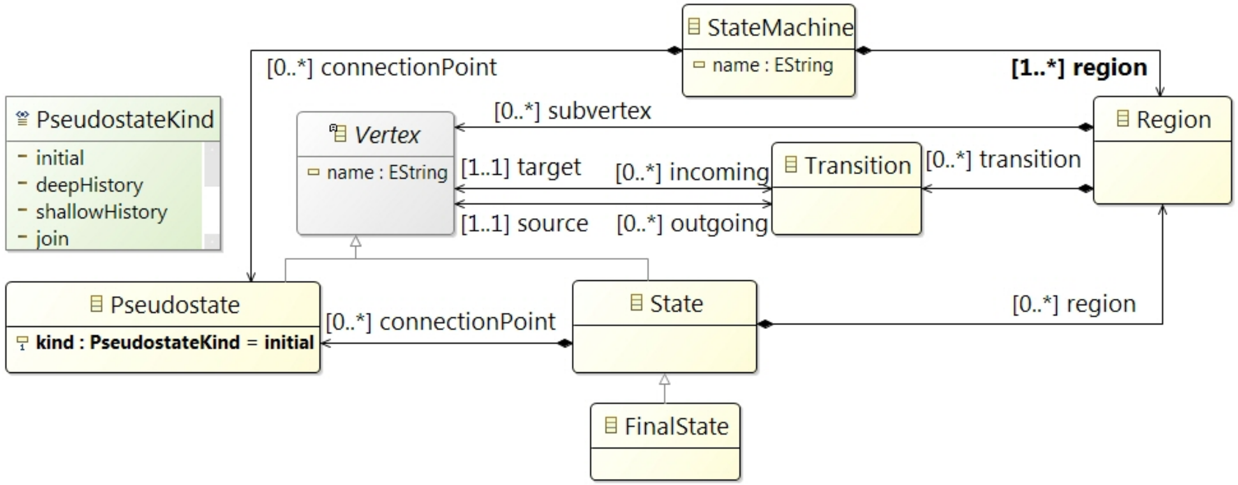
- ❑ Model



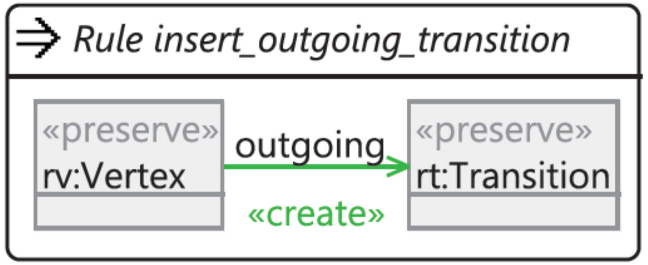
The abstract syntax of the state machine

Running Example

- ❑ Meta-model: A simple Statechart

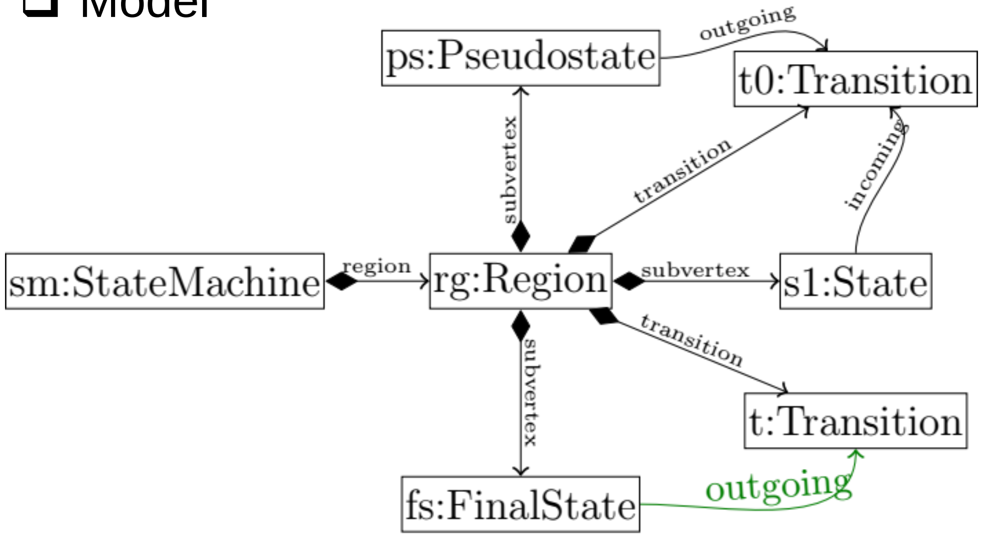


- ❑ Editing rules



Henshin rule: Insert_outgoing_transition

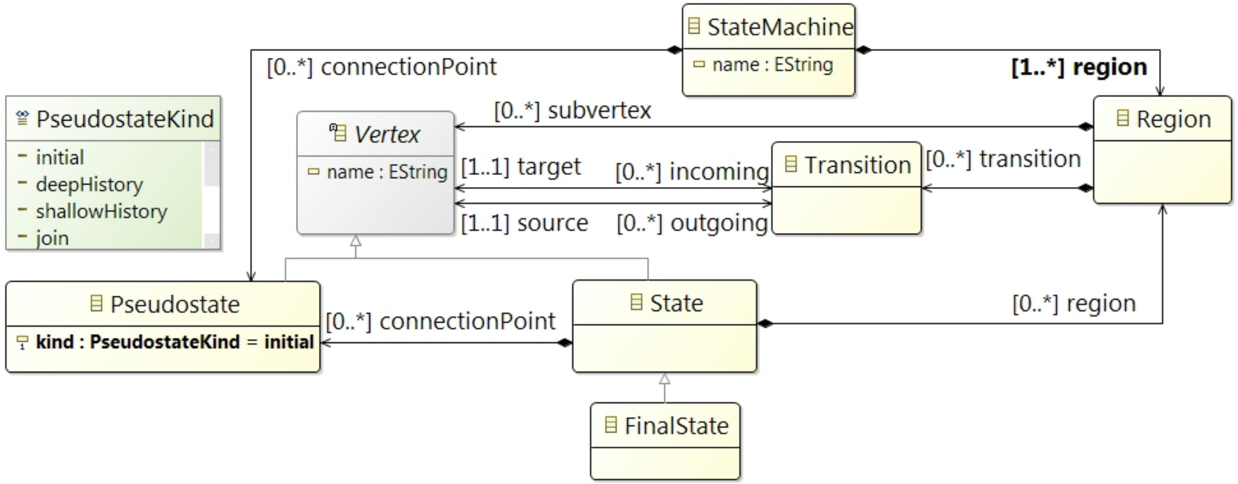
- ❑ Model



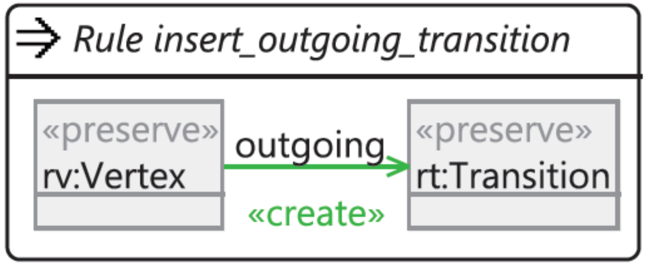
The abstract syntax of the state machine

Running Example

- ❑ Meta-model: A simple Statechart



- ❑ Editing rules



Henshin rule: Insert_outgoing_transition

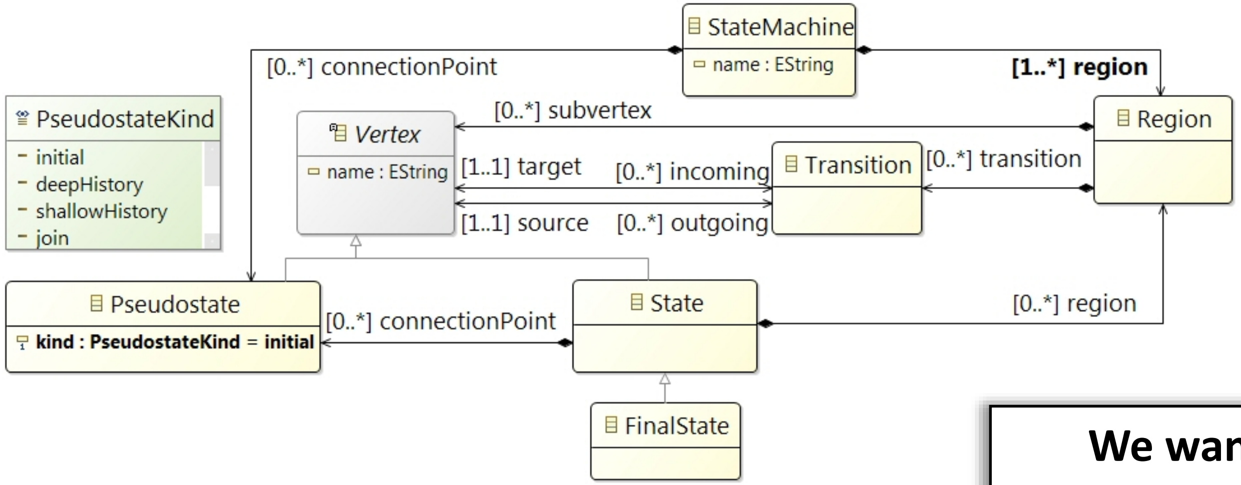
- ❑ Constraint: **A FinalState has no outgoing transition**.

OCL specification

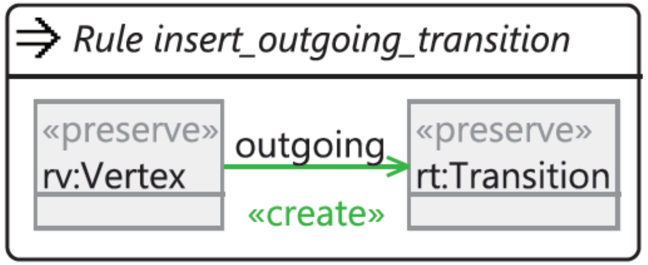
```
context FinalState inv no-outgoing-transitions:
self.outgoing -> isEmpty();
```

Running Example

- ❑ Meta-model: A simple Statechart



- ❑ Editing rules



Henshin rule: Insert_outgoing_transition

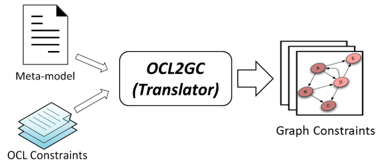
We want to adapt the rule to preserve the given constraint using OCL2AC

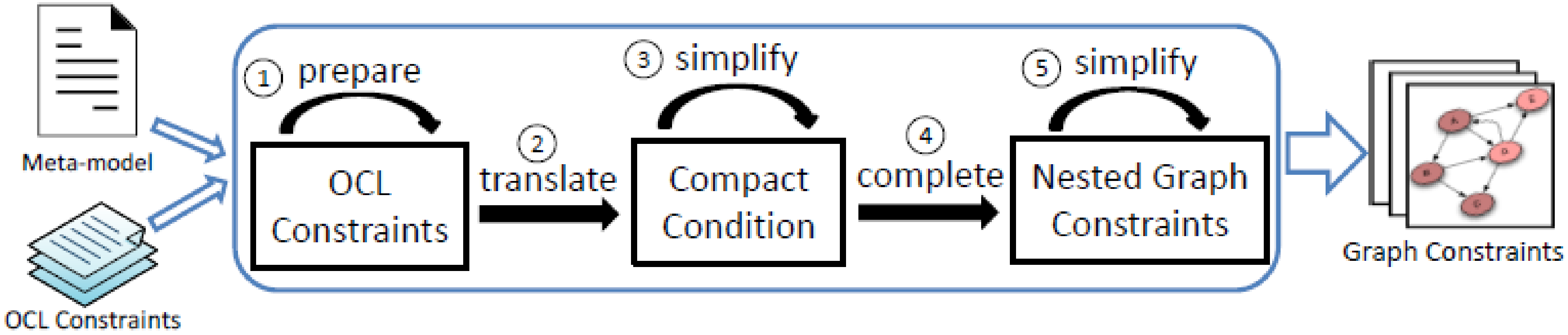
- ❑ Constraint: A FinalState has no outgoing transition.

OCL specification

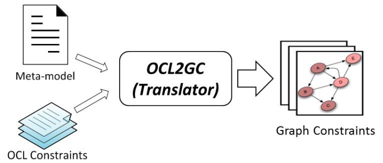
```
context FinalState inv no-outgoing-transitions:
self.outgoing -> isEmpty();
```

OCL2GC: Translate OCL Constraints to Graph Constraints

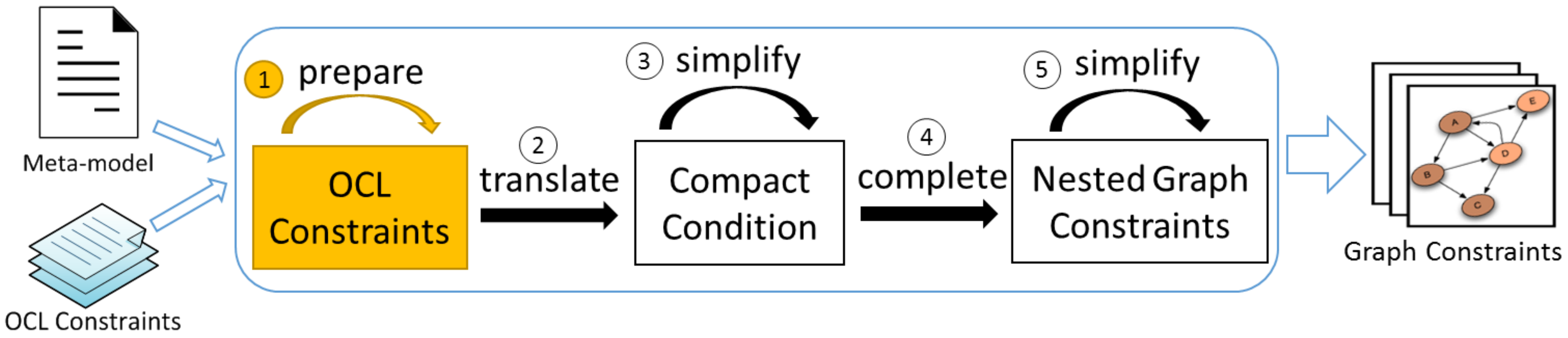




```
context FinalState inv no-outgoing-transitions:
self.outgoing -> isEmpty();
```



OCL2GC: Translate OCL Constraints to Graph Constraints

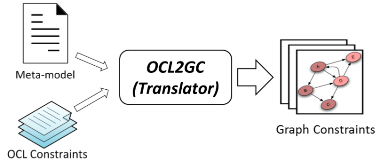


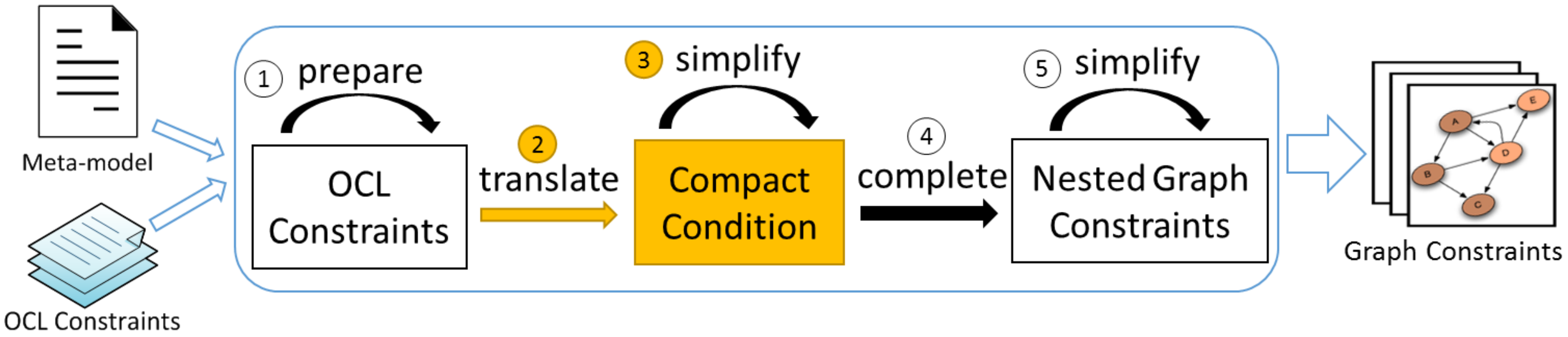
```
context FinalState inv no-outgoing-transitions:
self.outgoing -> isEmpty();
```

⬇ ①

```
context FinalState inv no-outgoing-transitions:
not (self.outgoing -> size()>=1);
```

OCL2GC: Translate OCL Constraints to Graph Constraints





```
context FinalState inv no-outgoing-transitions:
self.outgoing -> isEmpty();
```

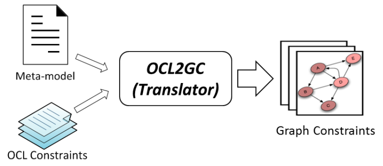
⬇ ①

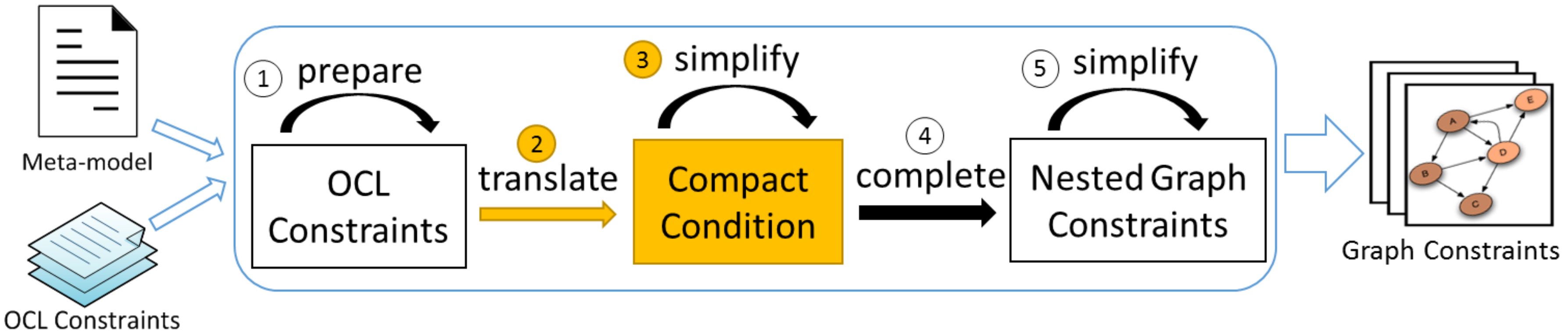
```
context FinalState inv no-outgoing-transitions:
not (self.outgoing -> size()>=1);
```

⬇ ②+③

$$\forall\left(\boxed{\text{self:FinalState}},\ \nexists\left(\boxed{\text{self:FinalState}} \xrightarrow{\text{outgoing}} \boxed{\text{var27:Transition}}\right)\right)$$

OCL2GC: Translate OCL Constraints to Graph Constraints





context FinalState **inv** no-outgoing-transitions:
self.outgoing -> isEmpty();

⬇ ①

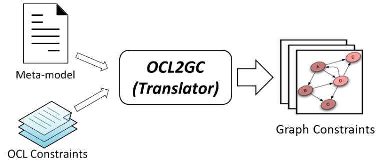
context FinalState **inv** no-outgoing-transitions:
not (**self**.outgoing -> size()>=1);

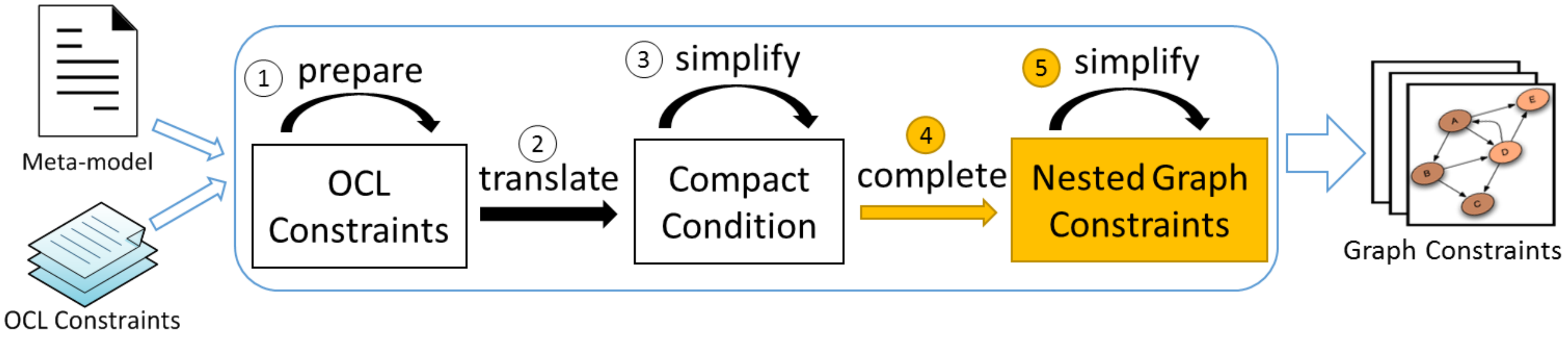
⬇ ②+③

$$\forall\left(\boxed{\text{self:FinalState}},\ \nexists\left(\boxed{\text{self:FinalState}} \xrightarrow{\text{outgoing}} \boxed{\text{var27:Transition}}\right)\right)$$

OCL constraint (snippet)	Graph pattern (snippet)
`context T inv:`	$\forall\ \boxed{\text{self:T}}$
`v.b` $\langle op \rangle$ `c`	$\begin{array}{\|c\|}\hline \text{v:T} \\ \hline \text{b } \langle \text{op} \rangle \text{ c} \\ \hline \end{array}$
`v.role`	$\exists\ \boxed{\text{v:T}} \xrightarrow{\text{role}} \boxed{\text{v':T'}}$
$\langle nav1 \rangle$`->union(`$\langle nav2 \rangle$`)`	$tr(\langle nav1 \rangle) \vee tr(\langle nav2 \rangle)$
$\langle nav \rangle$`->size() >= n`	$\exists(\boxed{\text{v}_1\text{:T}}\ \ldots\ \boxed{\text{v}_n\text{:T}}, \bigwedge_{i=1}^{n} tr(\langle nav(\text{v}_i) \rangle))$

OCL2GC: Translate OCL Constraints to Graph Constraints





```
context FinalState inv no-outgoing-transitions:
self.outgoing -> isEmpty();
```

⬇ ①

```
context FinalState inv no-outgoing-transitions:
not (self.outgoing -> size()>=1);
```

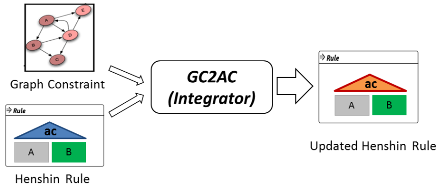
⬇ ②+③

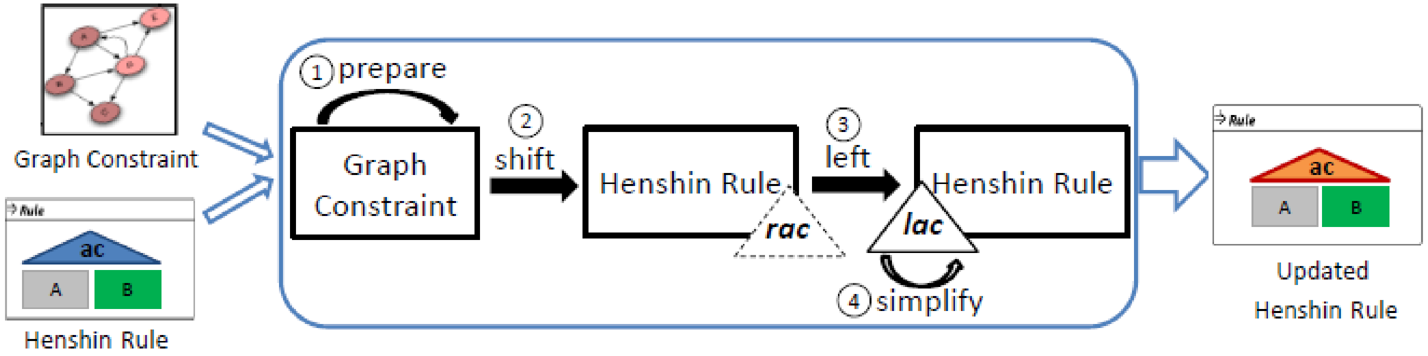
$$\forall\left(\boxed{\text{self:FinalState}},\ \nexists\left(\boxed{\text{self:FinalState}} \xrightarrow{\text{outgoing}} \boxed{\text{var27:Transition}}\right)\right)$$

4-5 ➡

$$\forall\left(\emptyset \hookrightarrow \boxed{\text{self:FinalState}},\ \nexists\left(\boxed{\text{self:FinalState}} \hookrightarrow \boxed{\text{self:FinalState}} \xrightarrow{\text{outgoing}} \boxed{\text{var27:Transition}},\ \mathit{true}\right)\right)$$

GC2AC: Integrate Graph Constraints as Left Application Conditions





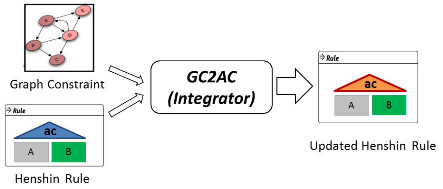
- **Graph constraint (no-outgoing-transitions)**

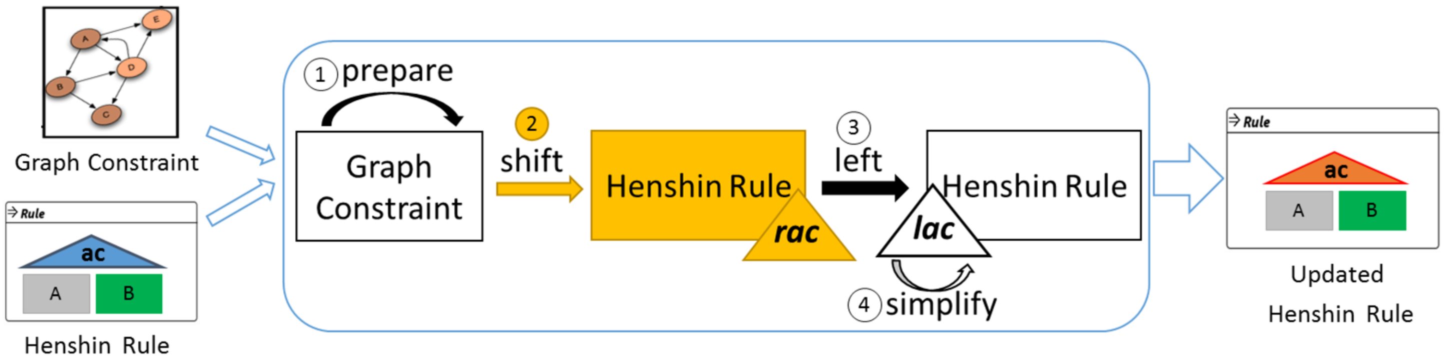
$$\forall\Big(\emptyset \hookrightarrow \boxed{\text{self:FinalState}}, \nexists\Big(\boxed{\text{self:FinalState}} \hookrightarrow \boxed{\text{self:FinalState}} \xrightarrow{\text{outgoing}} \boxed{\text{var27:Transition}}, \mathit{true}\Big)\Big)$$

- **Henshin rule (insert_outgoing_transition)**

$$\begin{array}{c}\boxed{\text{rv:Vertex}}\\ \\ \\ \boxed{\text{rt:Transition}}\end{array} \hookleftarrow \begin{array}{c}\boxed{\text{rv:Vertex}}\\ \\ \\ \boxed{\text{rt:Transition}}\end{array} \hookrightarrow \begin{array}{c}\boxed{\text{rv:Vertex}}\\ \Big\downarrow{\scriptstyle \text{outgoing}}\\ \boxed{\text{rt:Transition}}\end{array}$$

GC2AC: Integrate Graph Constraints as Left Application Conditions





- **Graph constraint**

$$\forall\left(\emptyset \hookrightarrow \boxed{\text{self:FinalState}}, \nexists\left(\boxed{\text{self:FinalState}} \hookrightarrow \boxed{\text{self:FinalState}} \xrightarrow{\text{outgoing}} \boxed{\text{var27:Transition}}, true\right)\right)$$

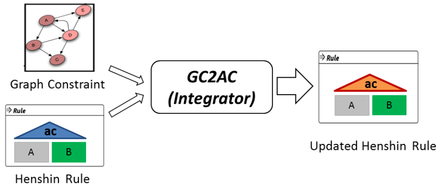
> Shift: For considering all possible ways in which a graph constraint could be satisfied after rule application

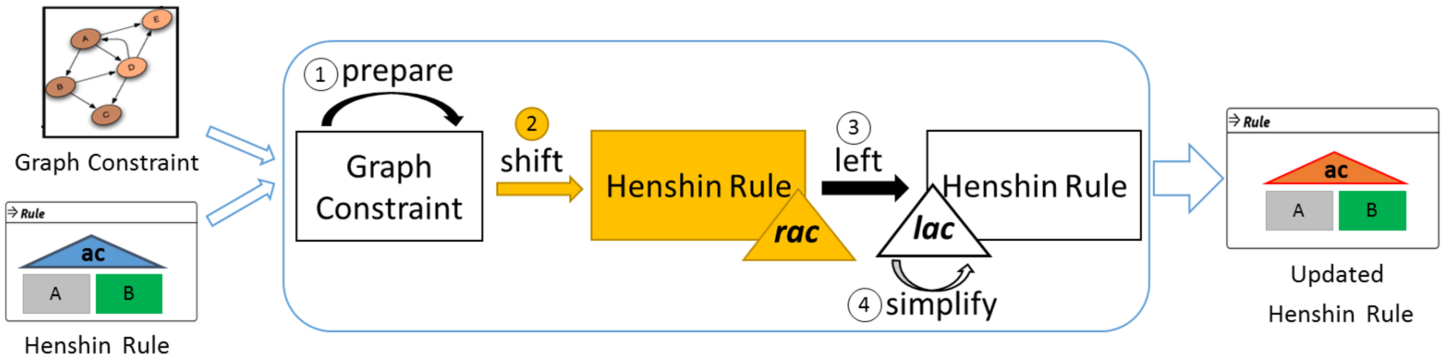
- **Right application condition (RAC)**

- **Henshin rule (insert_outgoing_transition)**

$$\begin{tikzcd} \boxed{\text{rv:Vertex}} & & \boxed{\text{rv:Vertex}} & & \boxed{\text{rv:Vertex}} \arrow[dd, "\text{outgoing}"] \\ & \hookleftarrow & & \hookrightarrow & \\ \boxed{\text{rt:Transition}} & & \boxed{\text{rt:Transition}} & & \boxed{\text{rt:Transition}} \end{tikzcd}$$

GC2AC: Integrate Graph Constraints as Left Application Conditions





- **Graph constraint**

$$\forall\Big(\emptyset \hookrightarrow \boxed{\text{self:FinalState}},\ \nexists\Big(\boxed{\text{self:FinalState}} \hookrightarrow \boxed{\text{self:FinalState}} \xrightarrow{\text{outgoing}} \boxed{\text{var27:Transition}},\ true\Big)\Big)$$

overlapping the RHS along graph constraint

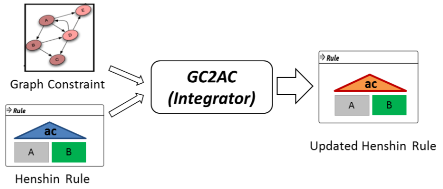
- **Henshin rule**

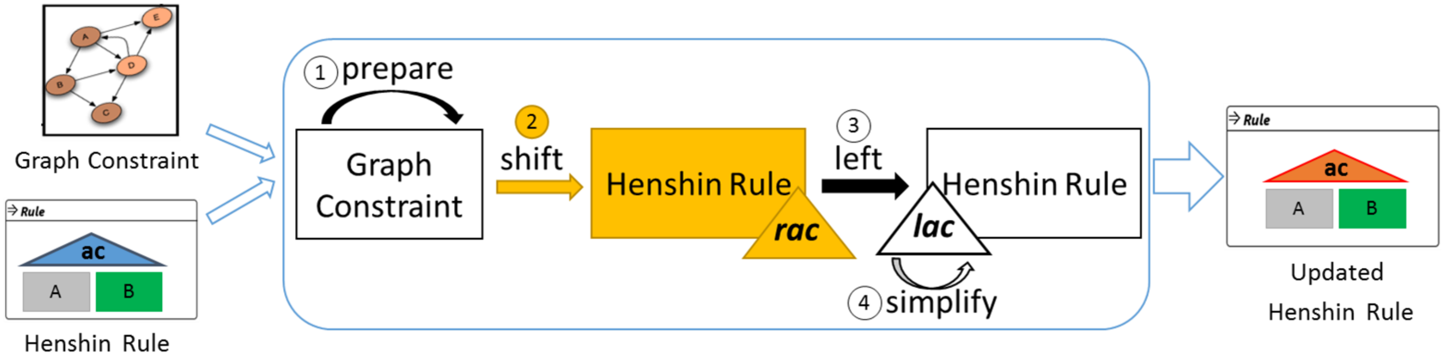
$$\begin{tikzcd}
\boxed{\text{rv:Vertex}} & & \boxed{\text{rv:Vertex}} & & \boxed{\text{rv:Vertex}} \arrow[dd, "\text{outgoing}"] \\
 & \hookleftarrow & & \hookrightarrow & \\
\boxed{\text{rt:Transition}} & & \boxed{\text{rt:Transition}} & & \boxed{\text{rt:Transition}}
\end{tikzcd}$$

> Shift: For considering all possible ways in which a graph constraint could be satisfied after rule application

- **Right application condition (RAC)**

GC2AC: Integrate Graph Constraints as Left Application Conditions





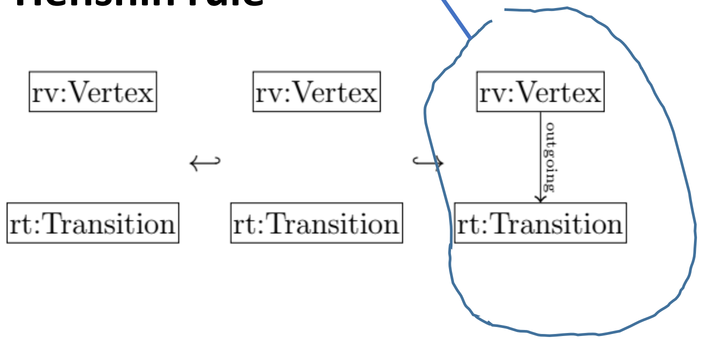
- **Graph constraint**

$$\forall\Big(\emptyset \hookrightarrow \boxed{\text{self:FinalState}},\ \nexists\Big(\boxed{\text{self:FinalState}} \hookrightarrow \boxed{\text{self:FinalState}} \xrightarrow{\text{outgoing}} \boxed{\text{var27:Transition}},\ true\Big)\Big)$$

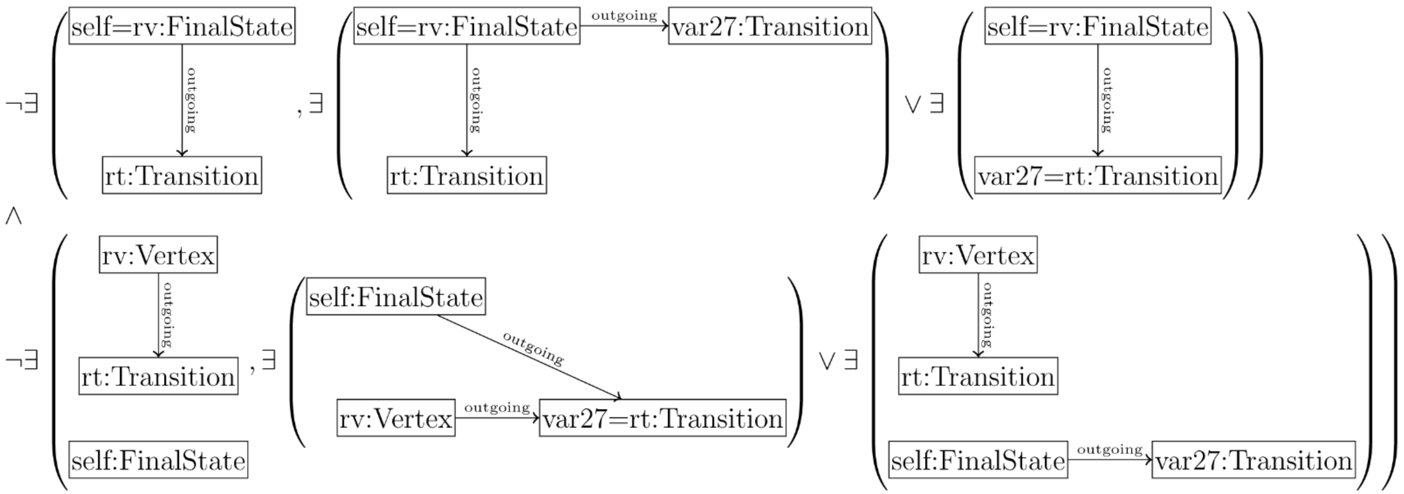
Shift: For considering all possible ways in which a graph constraint could be satisfied after rule application

overlapping the RHS along graph constraint

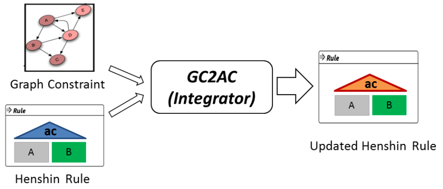
- **Henshin rule**

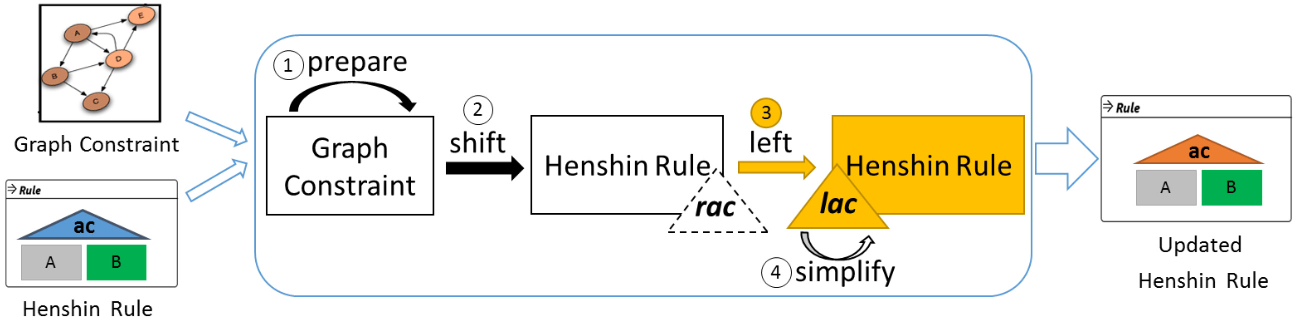


- **Right application condition (RAC)**

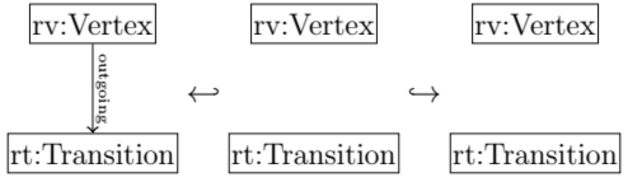


GC2AC: Integrate Graph Constraints as Left Application Conditions

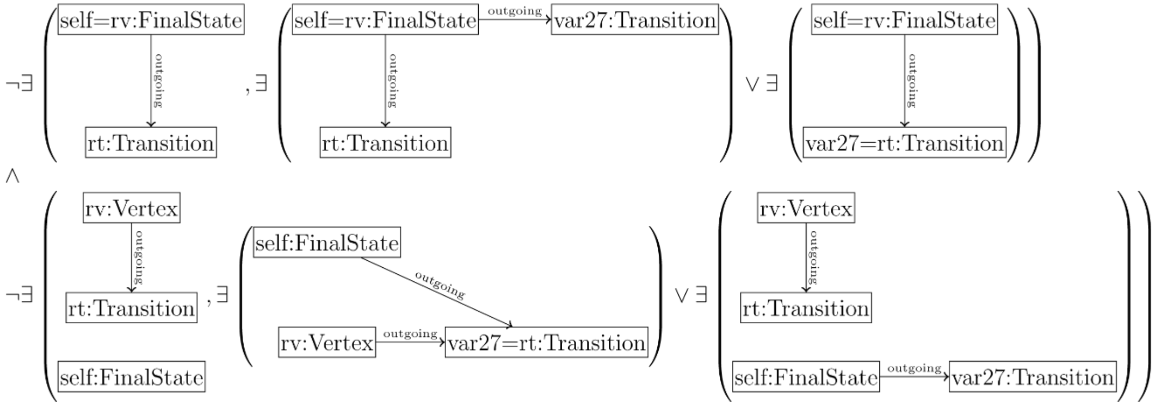




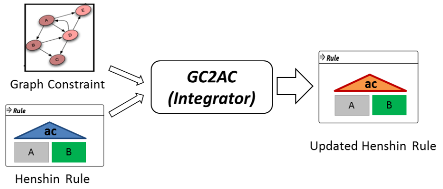
- **Inverse rule (delete-outgoing-transition)**

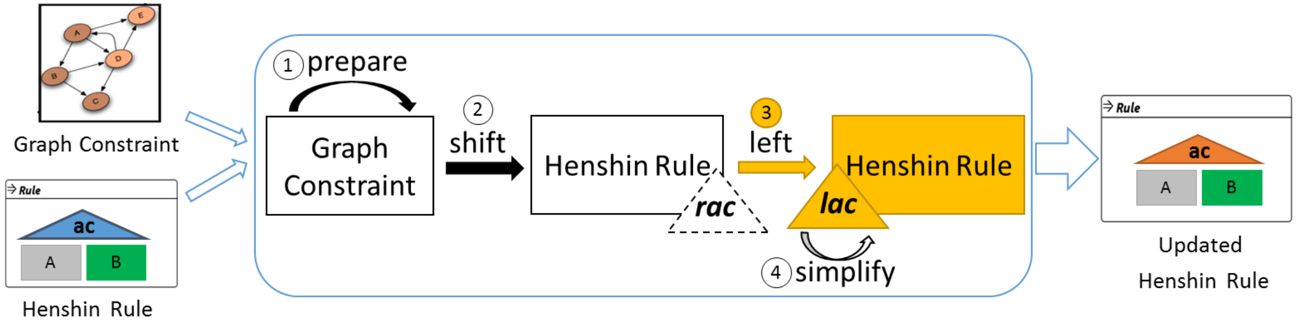


- **Right application condition (RAC)**

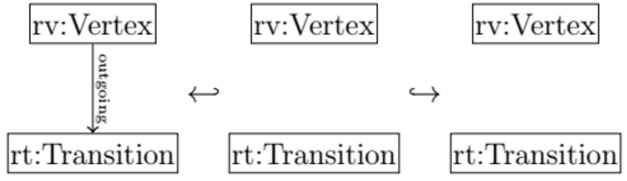


GC2AC: Integrate Graph Constraints as Left Application Conditions





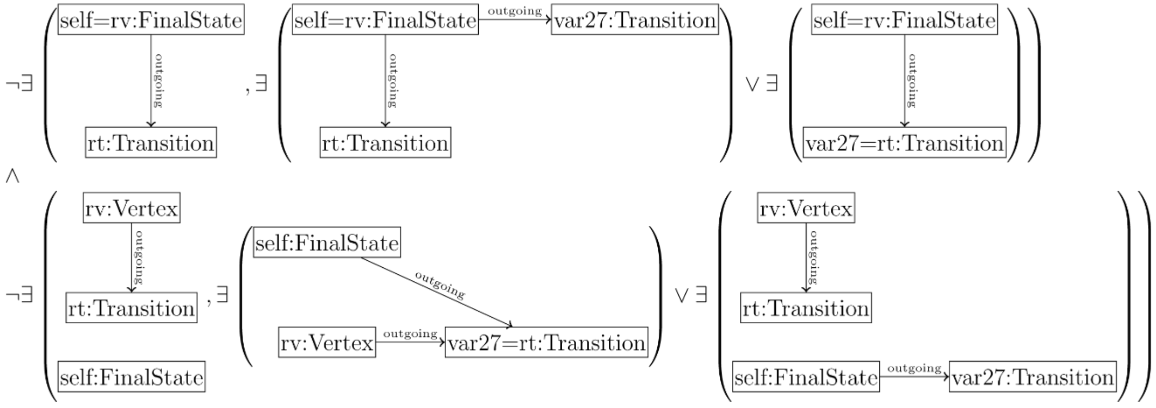
- **Inverse rule (delete-outgoing-transition)**



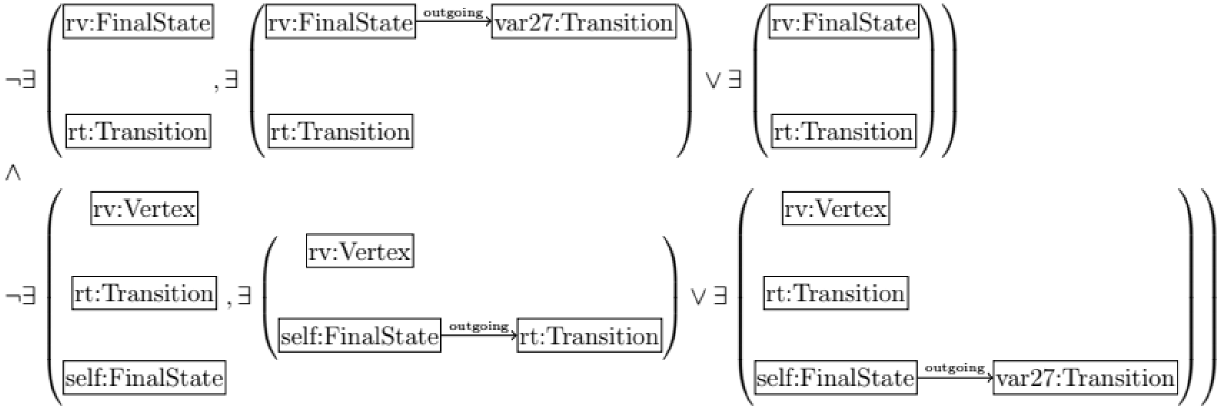
Apply it along RAC

lefting

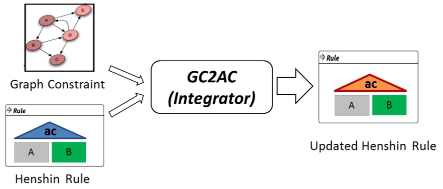
- **Right application condition (RAC)**

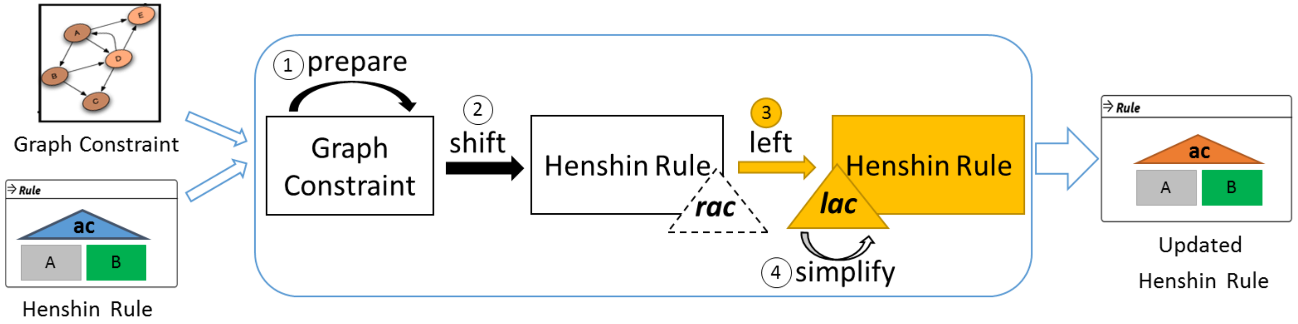


- **Left application condition (LAC = AC)**

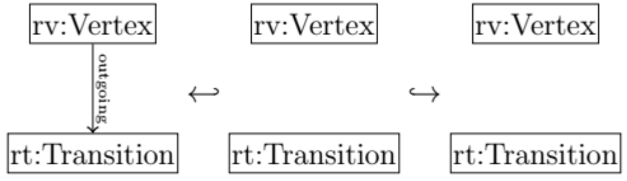


GC2AC: Integrate Graph Constraints as Left Application Conditions





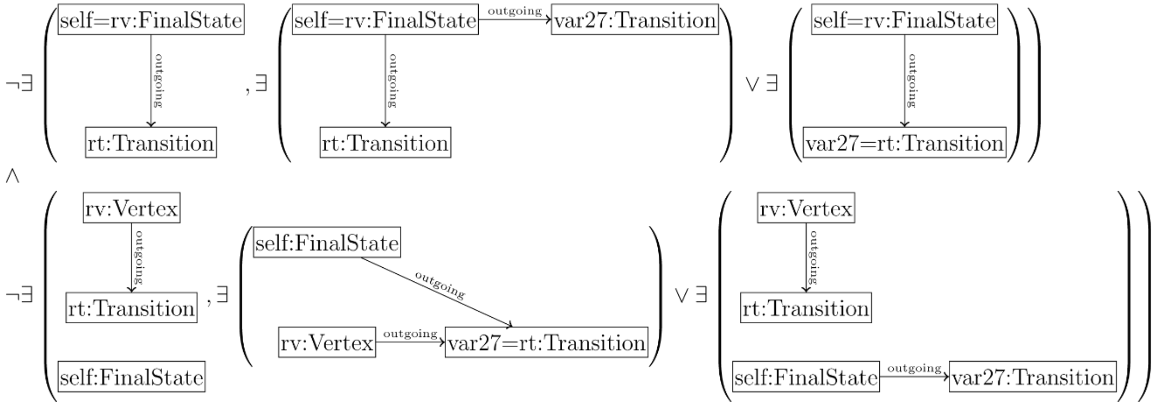
- **Inverse rule (delete-outgoing-transition)**



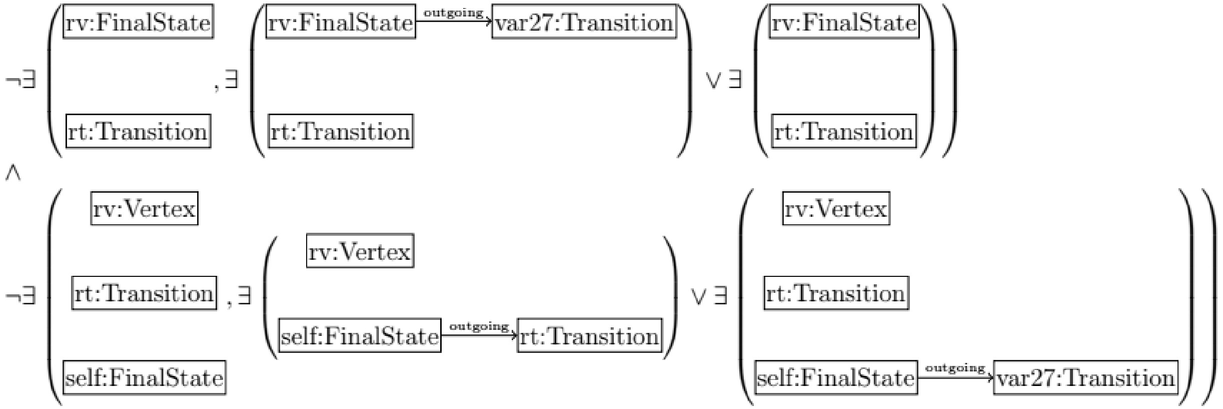
Apply it along RAC

lefting

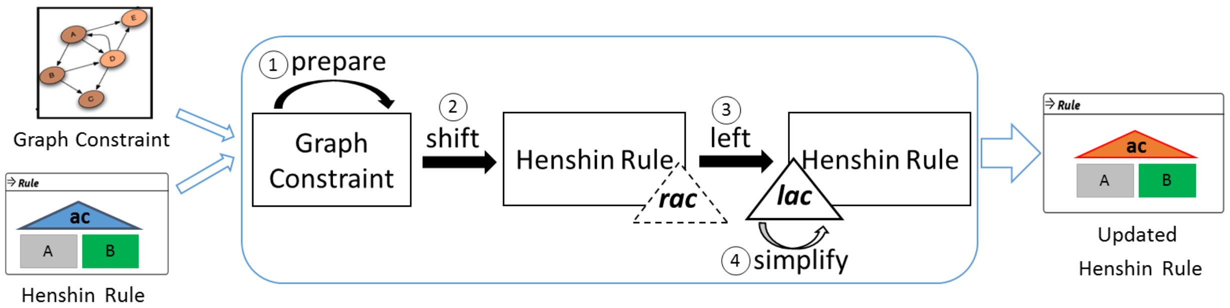
- **Right application condition (RAC)**



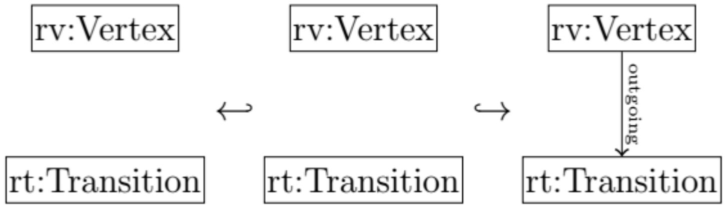
- **Left application condition (LAC = AC)**



GC2AC: Integrate Graph Constraints as Left Application Conditions



Rule insert_outgoing_transition

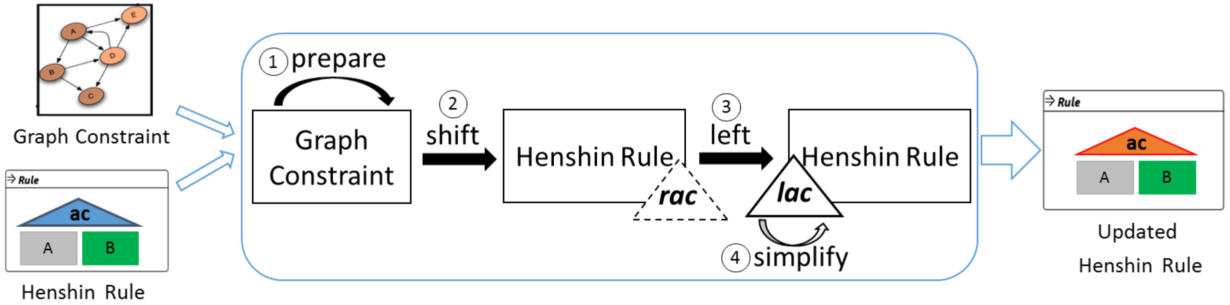


+

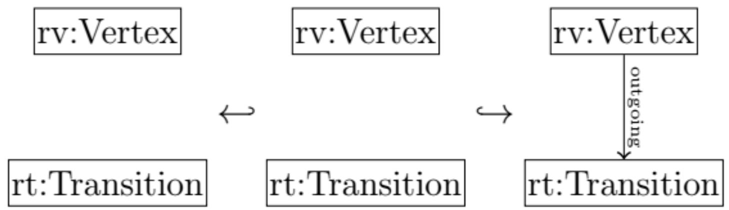
Left application condition (LAC or AC)

$$
\begin{array}{l}
\neg\exists\left(\begin{array}{c}\boxed{\text{rv:FinalState}}\\ \\ \boxed{\text{rt:Transition}}\end{array}, \exists\left(\begin{array}{l}\boxed{\text{rv:FinalState}}\xrightarrow{\text{outgoing}}\boxed{\text{var27:Transition}}\\ \\ \boxed{\text{rt:Transition}}\end{array}\right)\vee\exists\left(\begin{array}{c}\boxed{\text{rv:FinalState}}\\ \\ \boxed{\text{rt:Transition}}\end{array}\right)\right)\\
\wedge\\
\neg\exists\left(\begin{array}{c}\boxed{\text{rv:Vertex}}\\ \\ \boxed{\text{rt:Transition}}\\ \\ \boxed{\text{self:FinalState}}\end{array}, \exists\left(\begin{array}{l}\boxed{\text{rv:Vertex}}\\ \\ \boxed{\text{self:FinalState}}\xrightarrow{\text{outgoing}}\boxed{\text{rt:Transition}}\end{array}\right)\vee\exists\left(\begin{array}{l}\boxed{\text{rv:Vertex}}\\ \\ \boxed{\text{rt:Transition}}\\ \\ \boxed{\text{self:FinalState}}\xrightarrow{\text{outgoing}}\boxed{\text{var27:Transition}}\end{array}\right)\right)
\end{array}
$$

GC2AC: Integrate Graph Constraints as Left Application Conditions



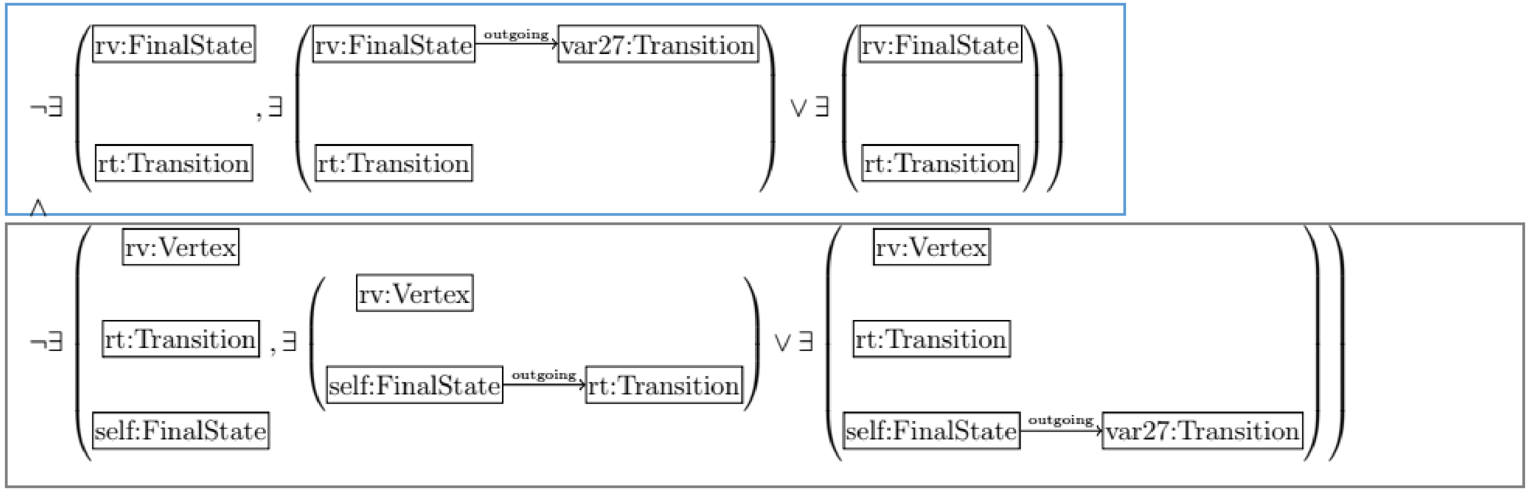
Rule insert_outgoing_transition



+

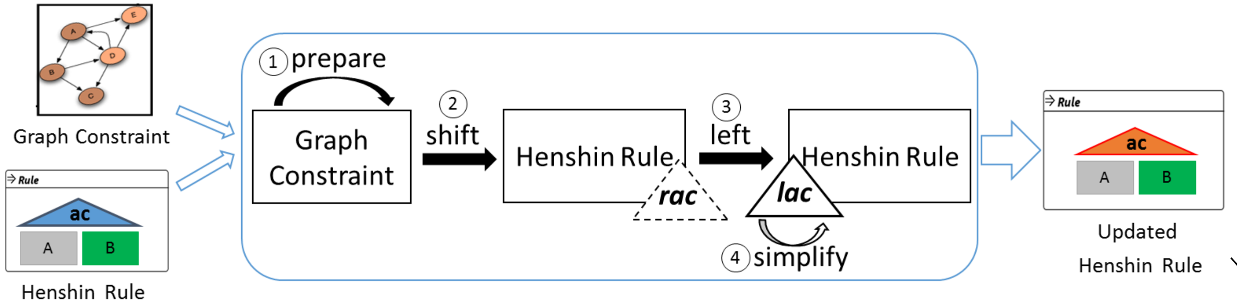
Left application condition (LAC or AC)

Forbids the rule node *rv:Vertex* being matched to a *FinalState*



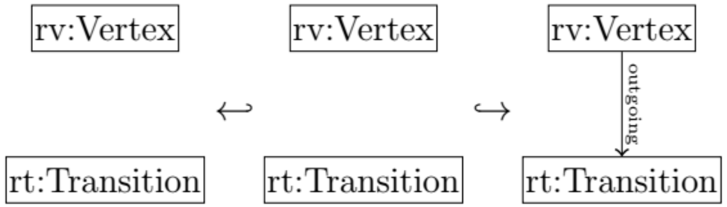
Requires that the rule is matched to consistent models only

Future Work: Simplifications of Application Conditions



- Working with valid instance models

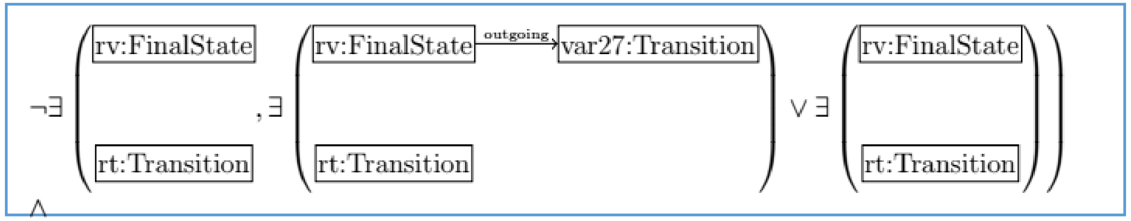
Rule insert_outgoing_transition

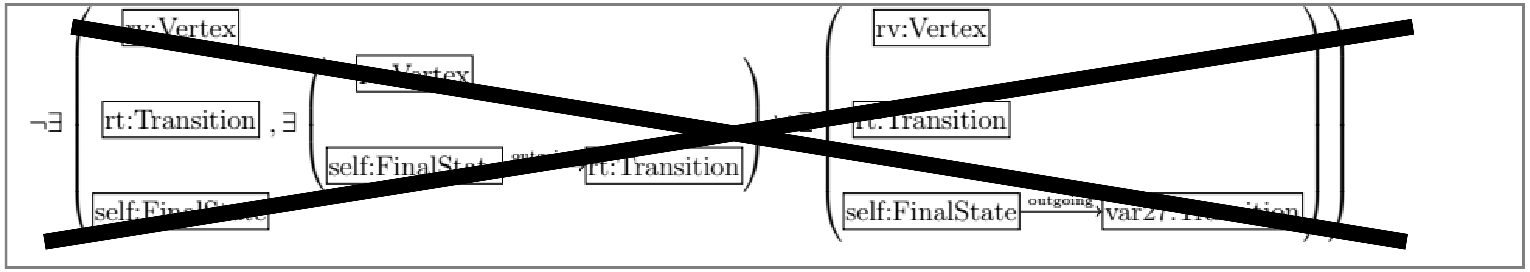


+

Left application condition (LAC or AC)

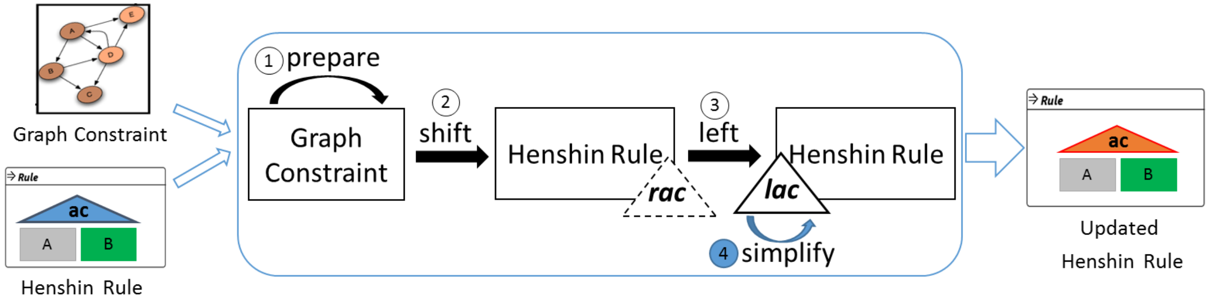
Forbids the rule node *rv:Vertex* being matched to a *FinalState*



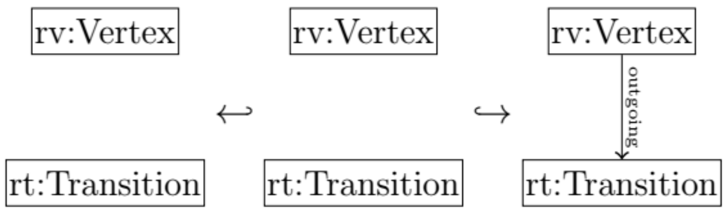


Requires that the rule is matched to consistent models only

Future Work: Simplifications of Application Conditions



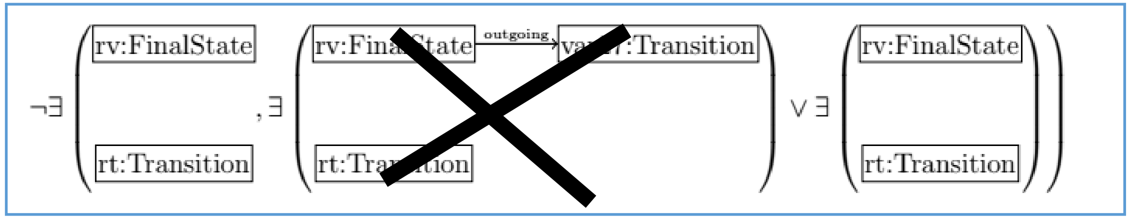
Rule insert_outgoing_transition



+

Left application condition (LAC or AC)

Forbids the rule node *rv:Vertex* being matched to a *FinalState*

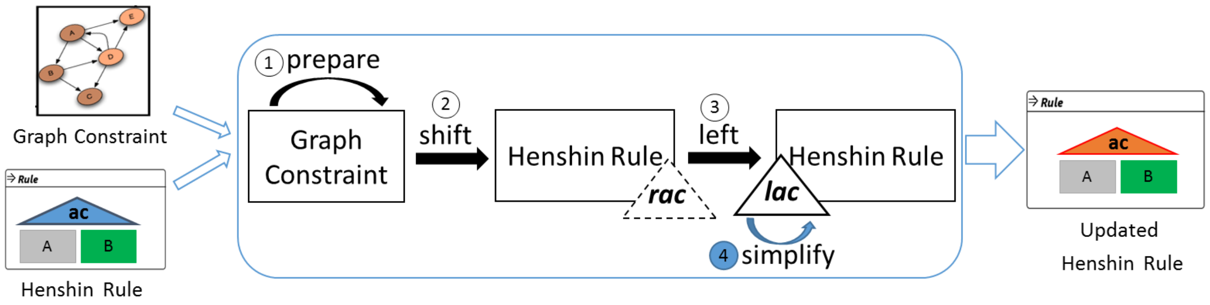


- Simplifications of application conditions
  - Eliminating unnecessary graphs. E.g.:

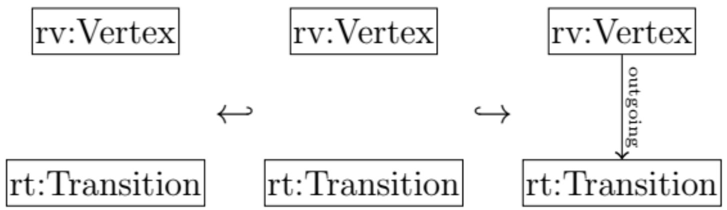
$$\exists\, g1 \vee \exists\, g2 \equiv \exists\, g1$$

| g1 is a subgraph of g2

Future Work: Simplifications of Application Conditions



Rule insert_outgoing_transition



+

Left application condition (LAC or AC)

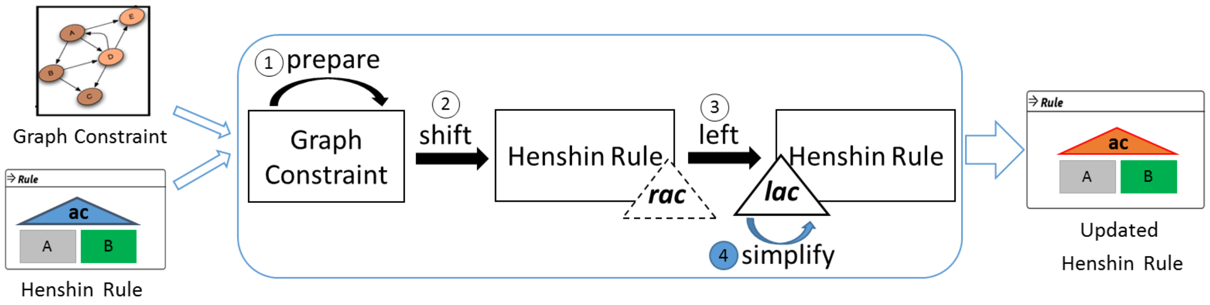
Forbids the rule node *rv:Vertex* being matched to a *FinalState*

$$\neg\exists\left(\begin{array}{c}\boxed{\text{rv:FinalState}}\\ \\ \boxed{\text{rt:Transition}}\end{array}, \exists\left(\begin{array}{c}\boxed{\text{rv:FinalState}}\\ \\ \boxed{\text{rt:Transition}}\end{array}\right)\right)$$

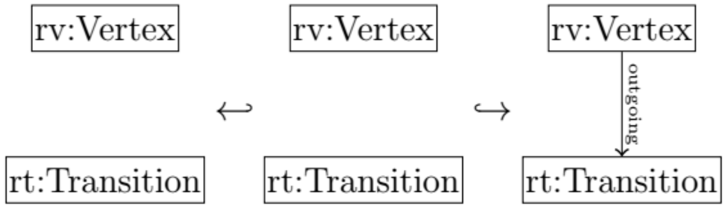
- Simplifications of application conditions
  - $\nexists\,(A, \exists\, C) \equiv \nexists\, C$

    | A is a subgraph of C

Future Work: Simplifications of Application Conditions



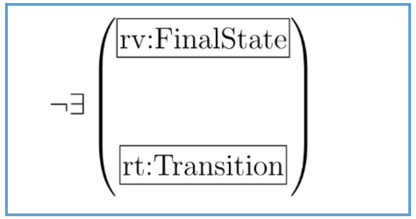
Rule insert_outgoing_transition



+

Left application condition (LAC or AC)

Forbids the rule node *rv:Vertex* being matched to a *FinalState*



Demo

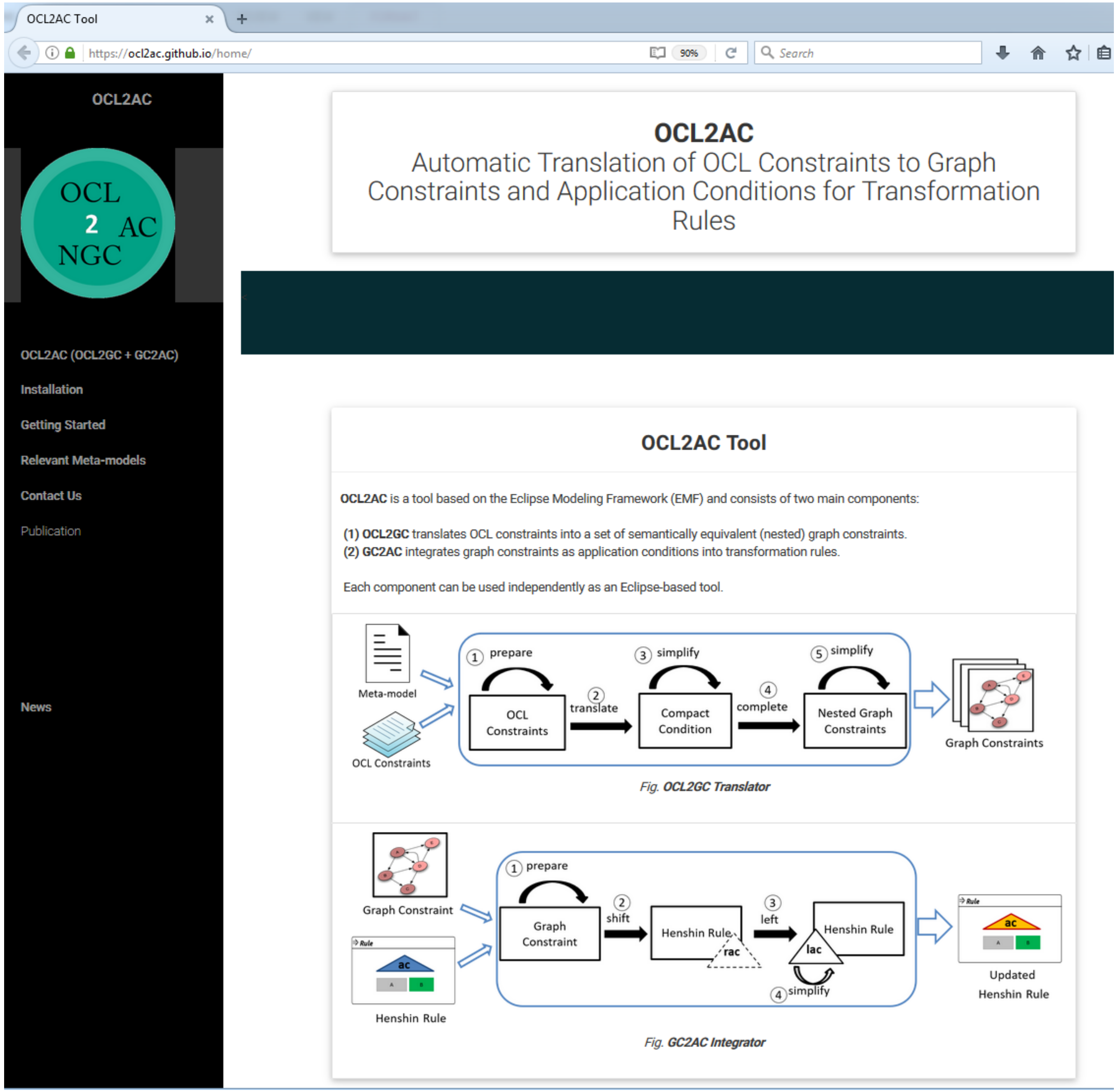
Demo

https://www.youtube.com/watch?v=75qXZboIVVg

Tooling: Webpage

Webpage on GitHub: [*https://ocl2ac.github.io/home/*](https://ocl2ac.github.io/home/)

- **Installation**
- **Getting Started**
- **Relevant Meta-models**



OCL2AC

Automatic Translation of OCL Constraints to Graph Constraints and Application Conditions for Transformation Rules

OCL2AC Tool

OCL2AC is a tool based on the Eclipse Modeling Framework (EMF) and consists of two main components:

(1) OCL2GC translates OCL constraints into a set of semantically equivalent (nested) graph constraints.
(2) GC2AC integrates graph constraints as application conditions into transformation rules.

Each component can be used independently as an Eclipse-based tool.

*Fig. **OCL2GC Translator***

*Fig. **GC2AC Integrator***

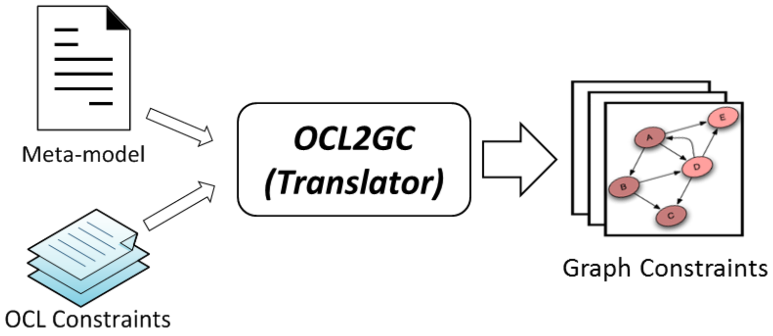
Conclusion

OCL2AC webpage on GitHub: https://ocl2ac.github.io/home/

- ❑ ***OCL2AC*** automatically updates model transformations to preserve a given set of constraints

Two main components as Eclipse plugins:

- **OCL2GC: Translate OCL constraints to graph constraints**



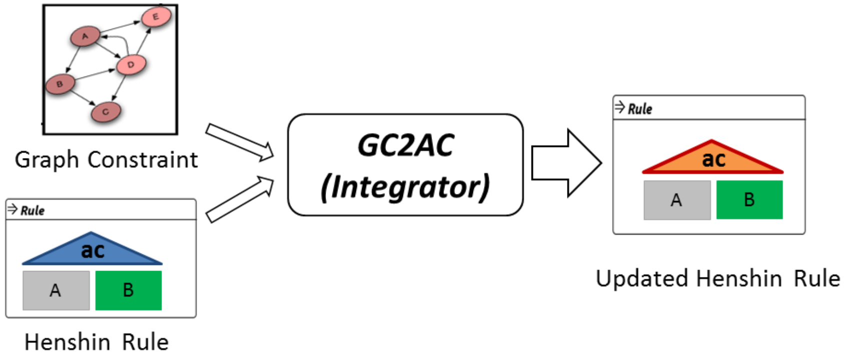
Meta-model

OCL Constraints

OCL2GC (Translator)

Graph Constraints

- **GC2AC: Integrate graph constraints as application conditions**



Graph Constraint

Henshin Rule

GC2AC (Integrator)

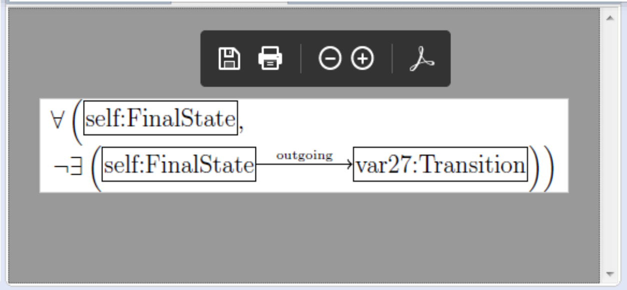
Updated Henshin Rule

OCL2GC

OCL constraints

context FinalState inv no-outgoing-transitions:
self.outgoing -> isEmpty();

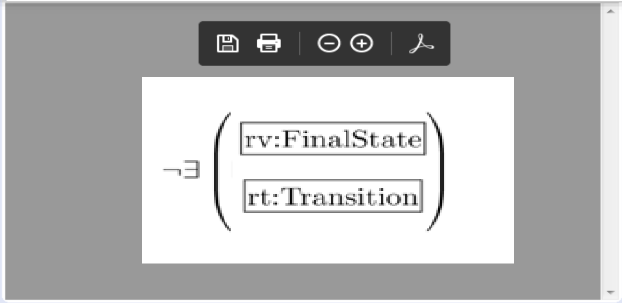
Graph constraints



$\forall\left(\boxed{\text{self:FinalState}},\ \neg\exists\left(\boxed{\text{self:FinalState}} \xrightarrow{\text{outgoing}} \boxed{\text{var27:Transition}}\right)\right)$

GC2AC

Application conditions



$\neg\exists\left(\begin{matrix}\boxed{\text{rv:FinalState}}\\ \boxed{\text{rt:Transition}}\end{matrix}\right)$