A Lightweight Approach for Model Checking Variability-Based Graph Transformations

Mitchell Albers[1]

Carlos Diego N. Damasceno[1]

Daniel Strüber[1,2]

1 Radboud University

2 CHALMERS | GÖTEBORGS UNIVERSITET

Agenda

1. Variability-based (VB) graph transformations

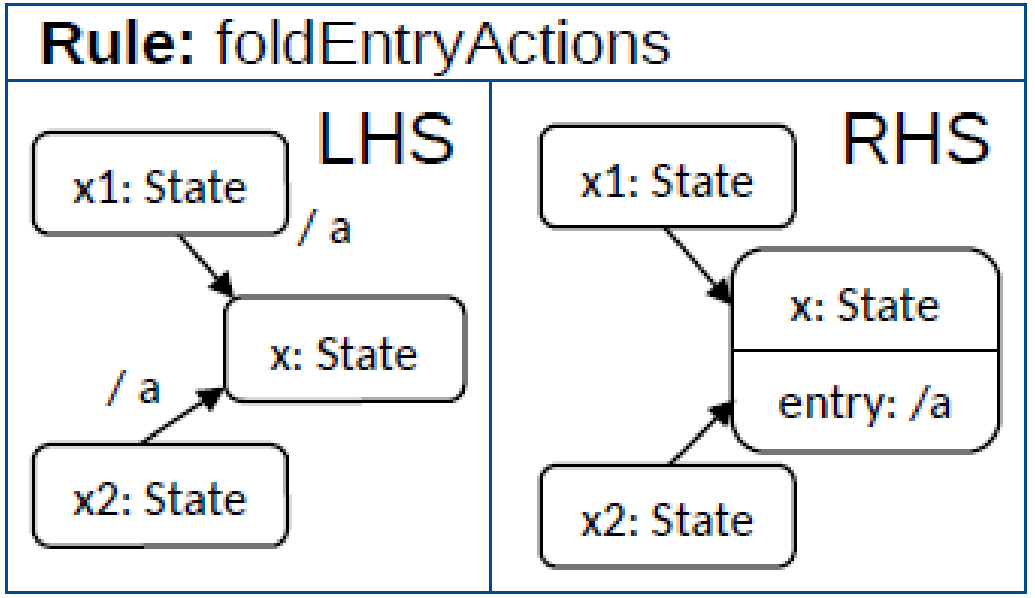
2. Model checking of VB graph transformations
   - Baseline model checking technique: Gryphon
   - Variability-aware encoding
   - Evaluation

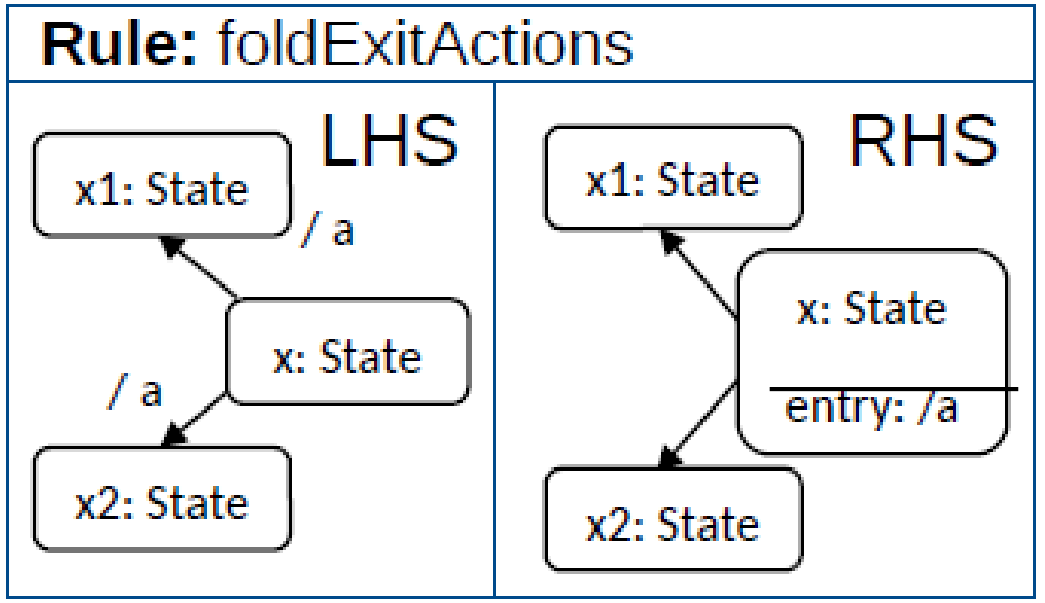
Variability-Based Graph Transformation

Rule variants

- Need for many similar, but different rules

- Often created in "copy and paste" manner
  - Error-prone
  - Problematic for maintenanc
  - Performance bottlenecks



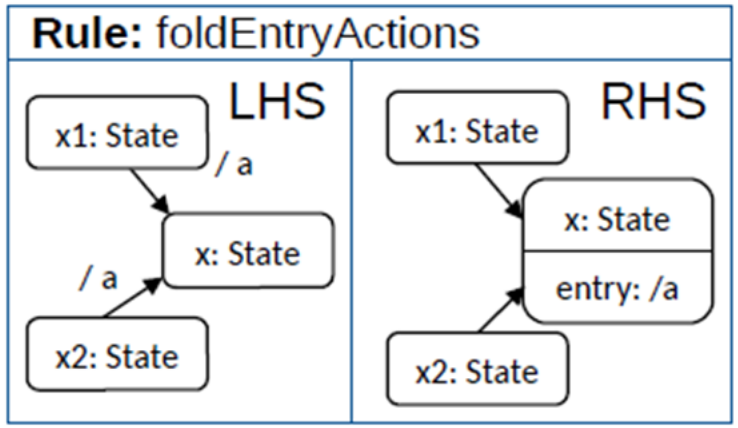


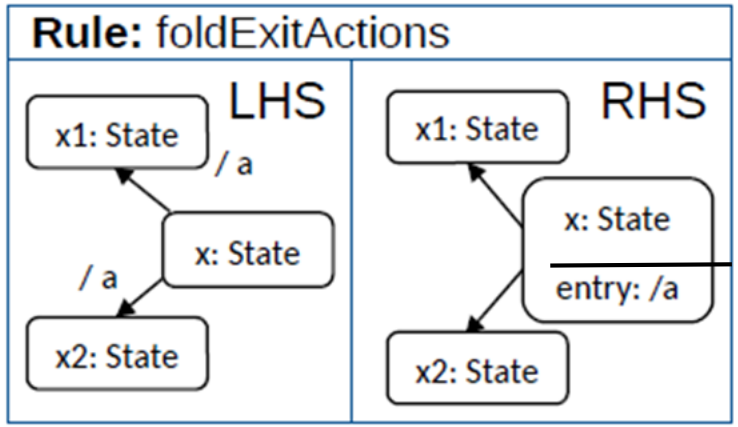
Overall research objective

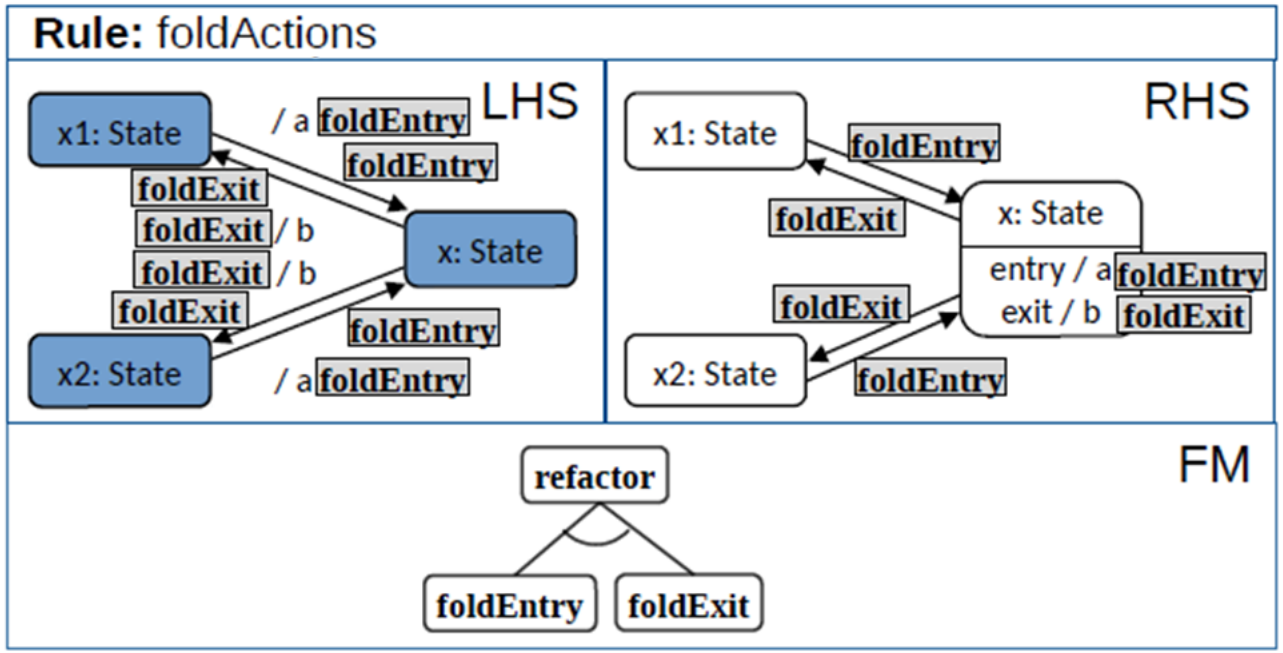
- In realistic scenarios, often 100s of similar rules!
- Address situations in which many rule variants which slow down execution of rule applications as well as analyses
- Objective:

Define an approach to specify and efficiently deal with rules with many variants

Variability-based (VB) rules: represent rule variants as a "rule product line"





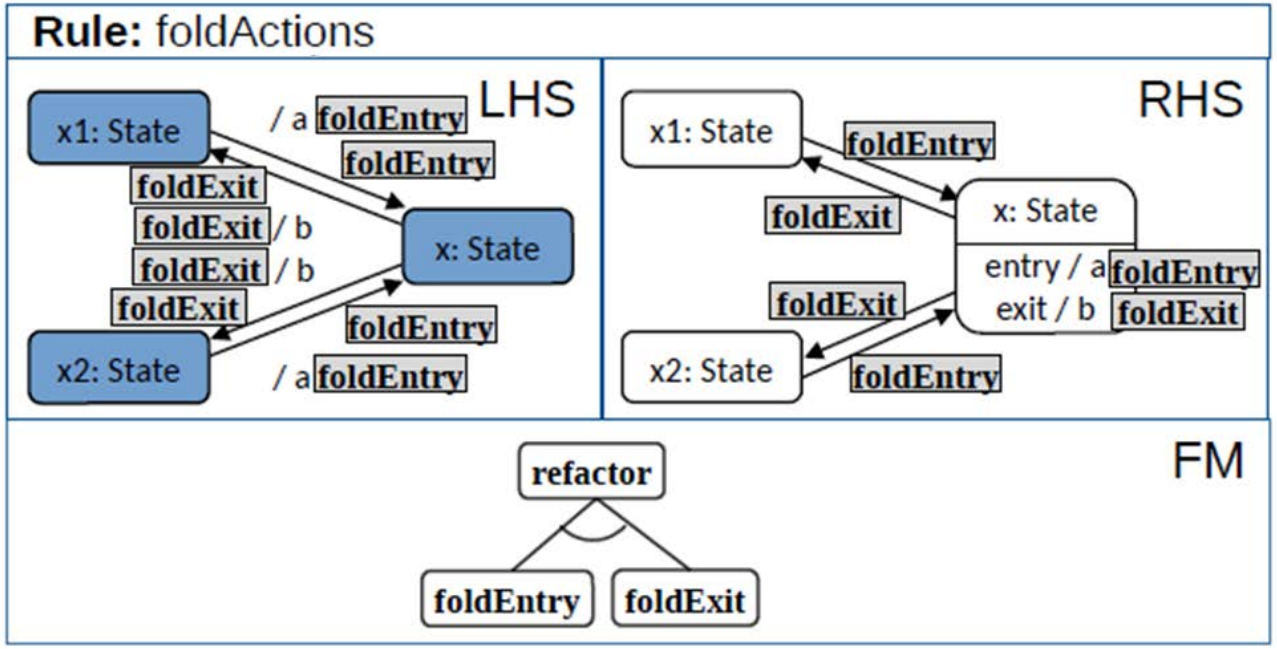


Rules with:

- A *feature model* FM
- Presence conditions (annotations)

Variability-based (VB) rules: represent rule variants as a "rule product line"

- Benefits:
  - **Maintainability**: all variants edited at once
  - **Performance**: can reuse shared parts during analysis (e.g., matching)



Formalizing VB rules

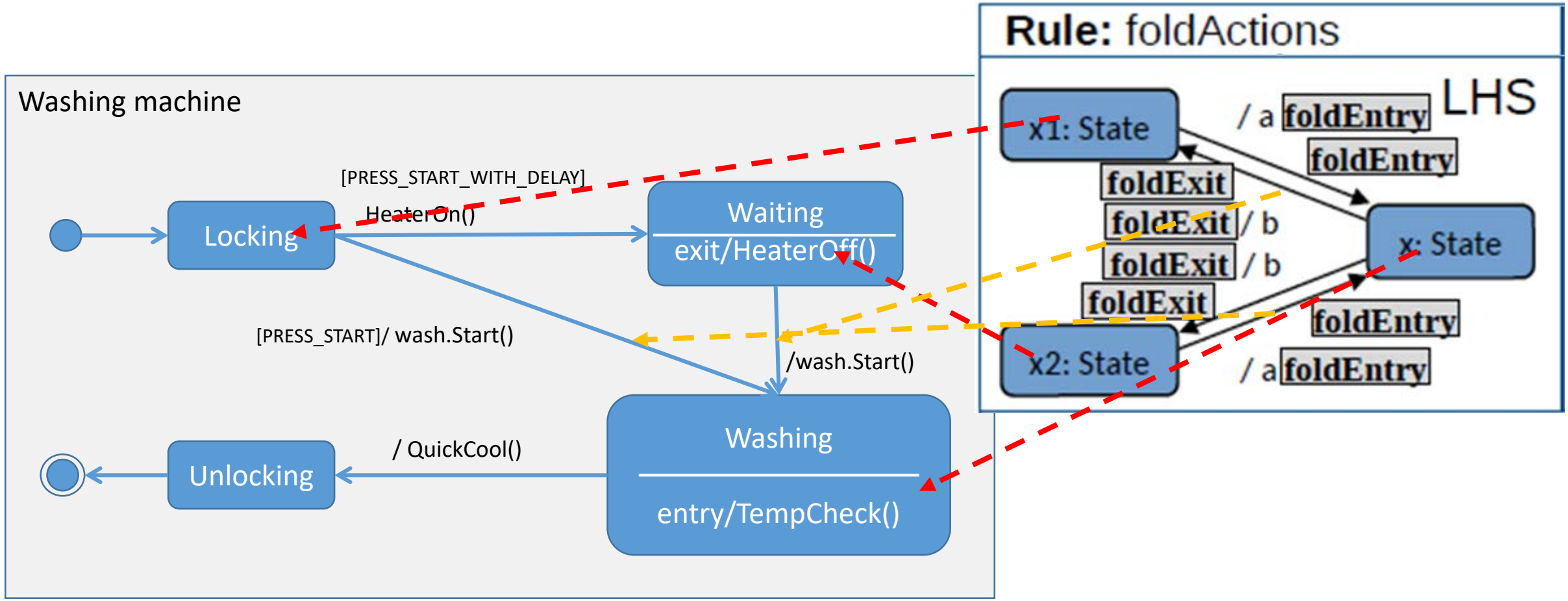
- A **variability-based rule** is a tuple (*r, S, pc, vm*) over a set of atoms *V*
  - Rule *r* — *maximal rule*
  - Set *S* of subrules of *r* — *subrules*
  - Function *pc: S → Bool(V)* — *presence conditions*
  - Formula *vm ∈ Bool(V)* — *feature model*

- A **configuration** is a total function cfg : V → { true, false }, configurations can *satisfy* presence conditions

- Given a configuration cfg, a **variant** is produced by merging all subrules $s \in S$ $s.t.$ cfg satisfies pc(s).

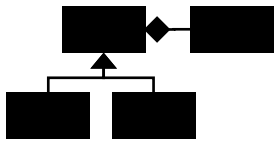
- The **base rule** is the subrule common to all variants.

Efficient application of VB rules



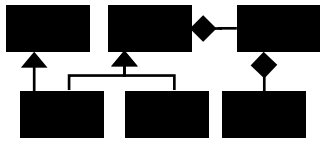
Key idea: during match-finding, first match the rule elements common to base rule and then extend the identified matches

Evaluation



Subject transformation

- Translator from OCL to Nested Graph Constraints
- 54 rules in total
- Subject: 36 rules being applied nondeterministically
- Left-hand sides: Between 9 and 37 graph elements



Benchmark models

- 10 typical OCL constraints + OCL standard library
- Between 1832 and 1854 graph elements

Metrics

- Performance: Execution time (10 runs)

Variability-based rule execution was 4 times as fast as classic rule execution

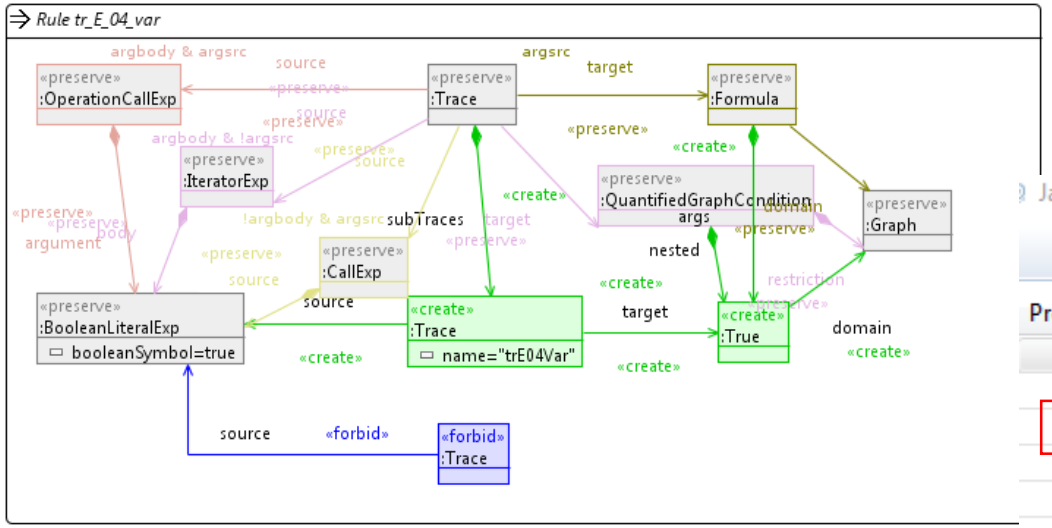
- ocl01-03: observed no difference
- ocl04-09: average speed gain of 3.9

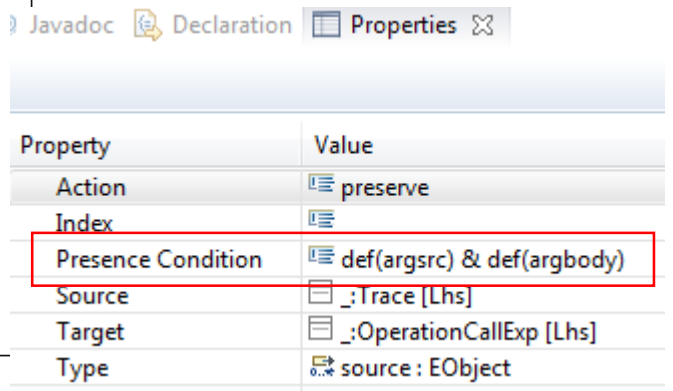
- Number of successful rule applications equal for both rule sets

	time (sec) classic		*time (sec) var.-based*	
model	mean	sd	mean	sd
ocl01	**<.1**	<.1	**<.1**	<.1
ocl02	**<.1**	<.1	**<.1**	<.1
ocl03	**<.1**	<.1	**<.1**	<.1
ocl04	**56.7**	10.6	**14.2**	4.5
ocl05a	**65.1**	9.2	**13.0**	3.4
ocl05b	**96.7**	20.4	**19.7**	4.8
ocl06	**49.0**	13.4	**11.5**	3.9
ocl07	**389.4**	93.4	**78.4**	3.5
ocl08	**191.0**	11.7	**48.4**	12.7
ocl09	**11.6**	2.6	**5.0**	1.5
average	**85.9**	16.1	**19.0**	3.4

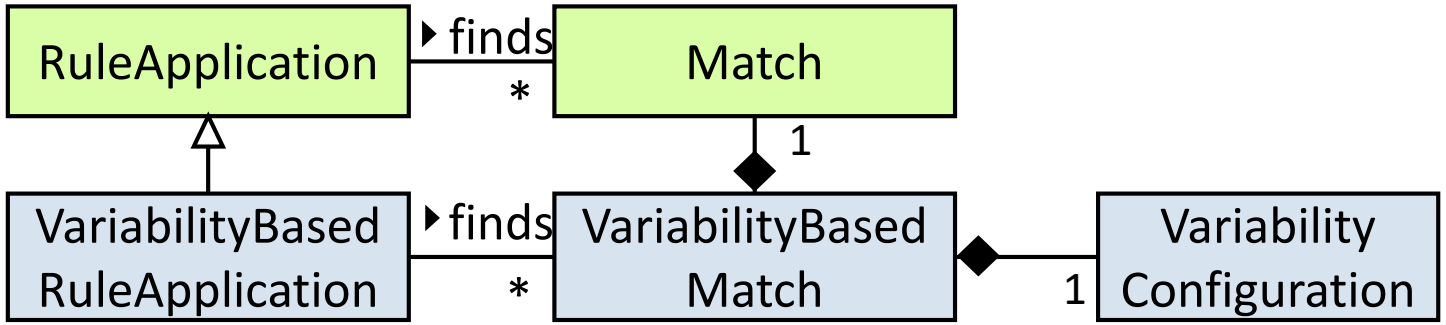
Tool support in Henshin

- To **specify** variability-based rules, we extended the Henshin editor



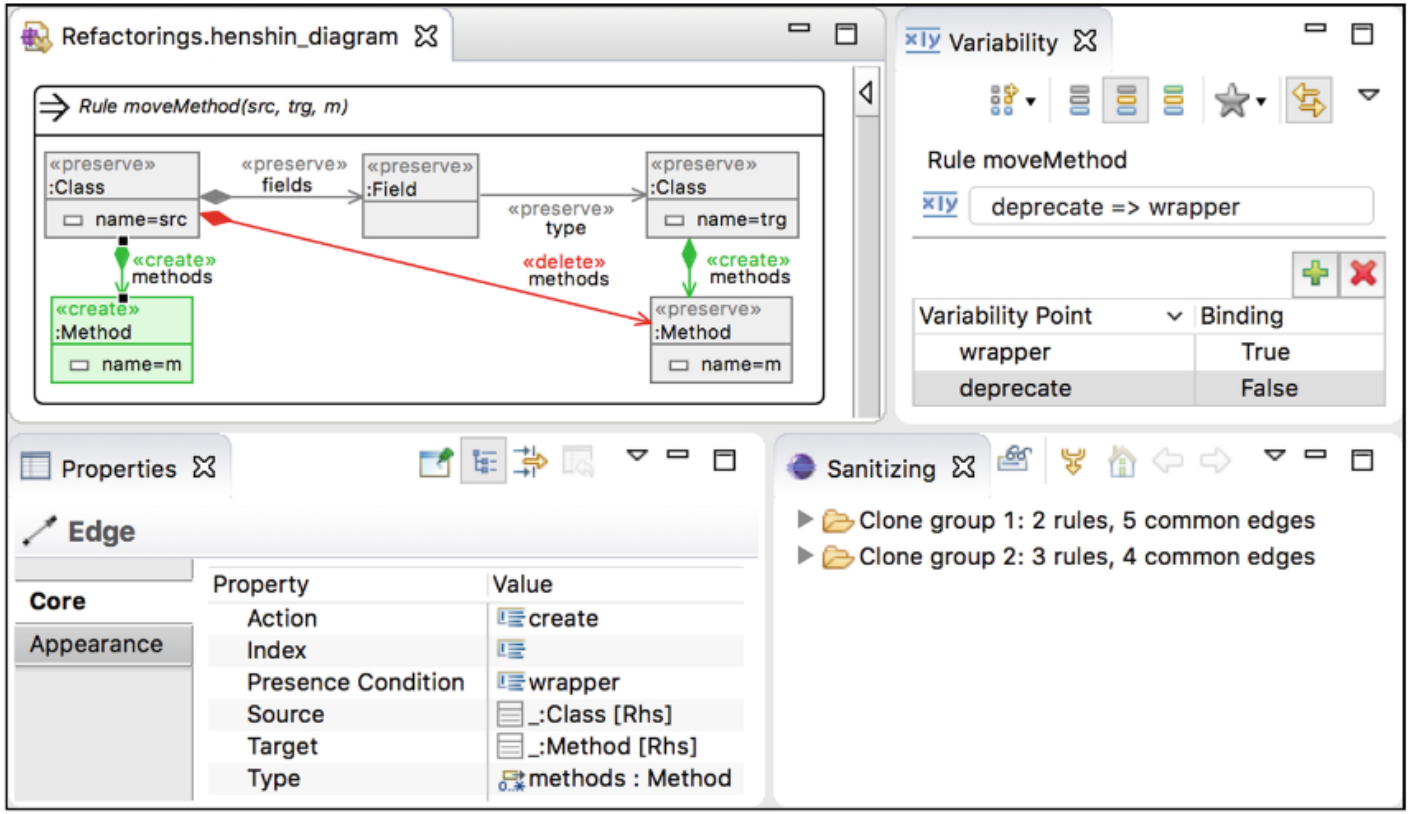


- To **apply** variabilited-based rules, we extended the Henshin interpreter API



Tool support in Henshin

- **Problem**: working with presence conditions can be awkward
  - Rules become big, editing boolean formulas by hand is not intuitive
- **Solution**: Advanced tool support; filtering and smart editing functions



Paper:

Daniel Strüber, Stefan Schulz: A Tool Environment for Managing Families of Model Transformation Rules. ICGT 2016: 89-101.

Model Checking of VB Graph Transformation

Research objective of this work

- **Context**: analysis of graph transformations, specifically, model checking
  - MC typically prone to state space explosion
  - Rule variants can add one level of combinatorial explosion

- Objective:

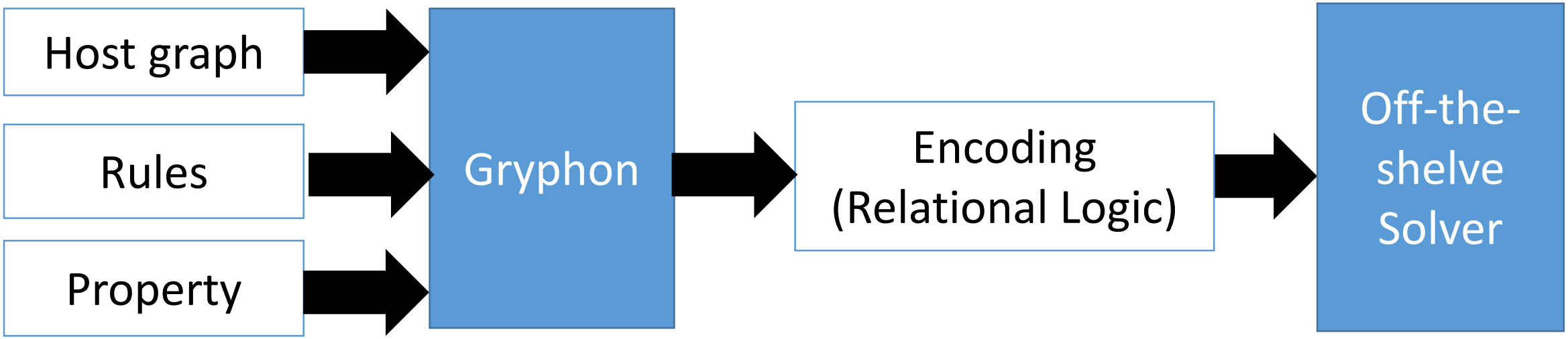
Address variability to make model checking of graph transformations more efficient

Gryphon: a symbolic model checking technique

- **Model checking setup:** Given a GTS and a host graph (= initial state), check whether a (potentially bad) state is reachable
  - Properties of form $\Diamond \phi$, where φ is a graph constraint
  - Bounded universe (no arbitrary node deletion / creation supported)
- **Key idea**: encode GTS, graph and property into a lower-level encoding that can be solved using an available solver

Gabmeyer, S., & Seidl, M. (2016). Lightweight Symbolic Verification of Graph Transformation Systems with Off-the-Shelf Hardware Model Checkers. In B. K. Aichernig & C. A. Furia (Eds.), *Tests and Proofs* (Vol. 9762, pp. 94–111). Springer International Publishing. https://doi.org/10.1007/978-3-319-41135-4_6

Gryphon setup



Gryphon's encoding: variables

- Generate relational variables based on type graph $G_T = (V_T,\ E_T)$
- $relgen : V_T \cup E_T \rightarrow Rel$
  - Generate a unary relational variable $r \in Rel$ for each node in V_T
  - Generate a binary relational variable $r \in Rel$ between the source and target node for each relation in E_T
- Include generated relational variables into a bounded universe U
  - Consisting of a sequence of uninterpreted atoms $\mathbb{A}$ (derived from host graph)
- Assign (upper) bounds to relational variables $\sqcup : Rel \rightarrow \mathcal{P}(\mathbb{A})$

Gryphon's encoding: formulae

- For each rule, generate a formula of the form

$$\mathrm{F}_t := Pre(L, Nac, R) \Longrightarrow Post(L, R)$$

- With *Pre* and *Post* being conjunctions of relational formulas mimicking matching and modification, respectively

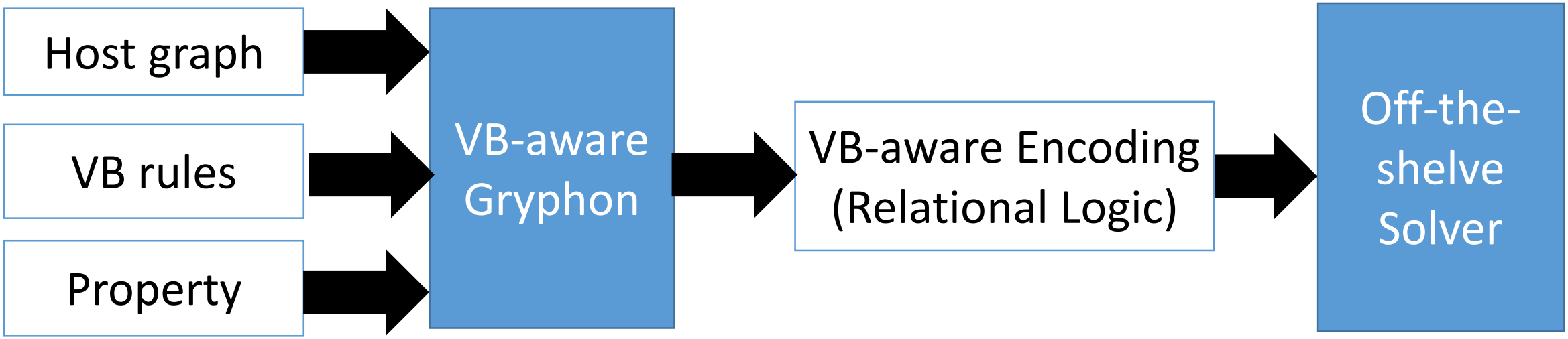
Gryphon's encoding: formulae

$$\underbrace{\exists \mathsf{a1} : \mathsf{A}, \exists \mathsf{a2} : \mathsf{A}', \exists \mathsf{b} : \mathsf{B}, \exists \mathsf{c} : \mathsf{C}}_{\text{LHS,RHS nodes}}, \underbrace{\neg \exists \mathsf{d} : \mathsf{D}}_{\text{NAC}} \mid$$

$$\underbrace{\mathit{match}(\mathsf{a1}, \mathsf{a2}, \mathsf{b}, \mathsf{c}, \mathsf{d})}_{\text{match constraints}} \wedge \underbrace{\mathit{inj}(\mathsf{a1}, \mathsf{b}, \mathsf{c}, \mathsf{d})}_{\substack{\text{injectivity} \\ \text{constraints}}} \Longrightarrow$$

$$\underbrace{\mathsf{A}' = \mathsf{A} - \mathsf{a1} + \mathsf{a2} \wedge \mathsf{B}' = \mathsf{B} - \mathsf{b}}_{\text{modification constraints}} \wedge \underbrace{\mathsf{C}' = \mathsf{C} \wedge \mathsf{D}' = \mathsf{D} \wedge \mathsf{E}' = \mathsf{E}}_{\text{non-modification constraints}}$$

Contribution: making Gryphon variability-aware



VB-aware encoding: variables

- Generate a unary relational (feature) variable for each feature $f \in \mathcal{F}$
  $f_{relgen} : \mathcal{F} \rightarrow Rel$

- Set bounds for relational feature variables $r = f_{relgen}(\mathcal{F})$ to boolean values $\sqcup\,(r) = \{true, false\}$

- Generate a relational formula for each presence condition

  $pc_{relgen} : Bool(\mathscr{F}) \rightarrow Bool(Rel)$

VB-aware encoding: formulae

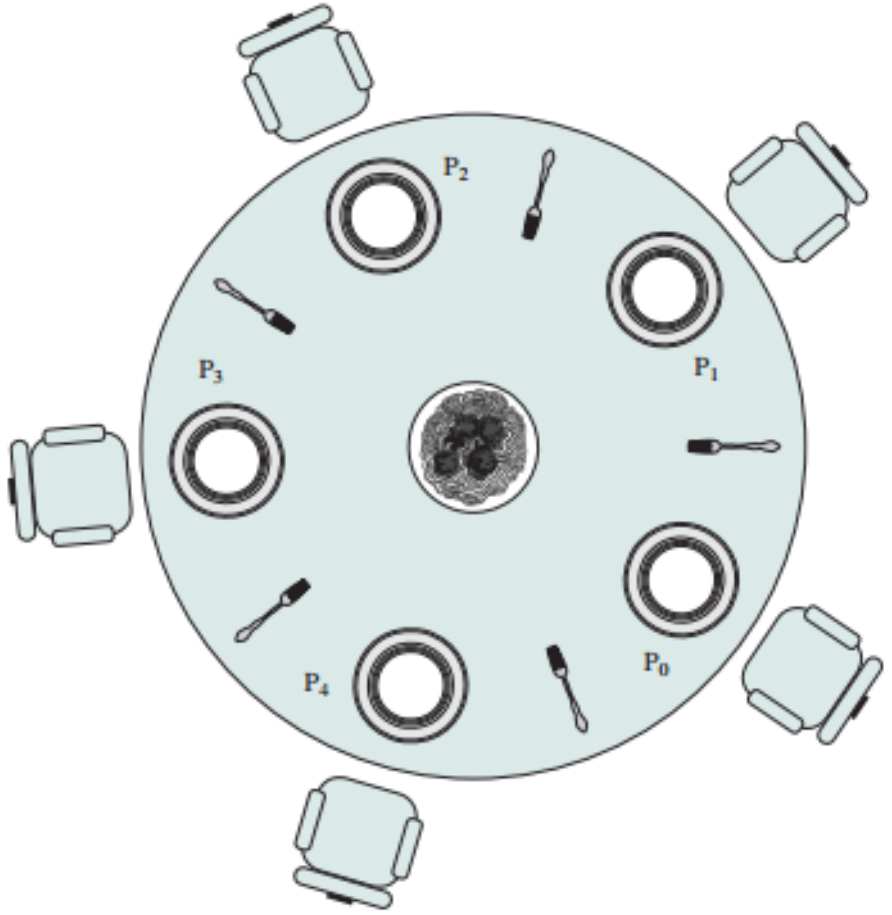
- **Key idea:** make match constraints depend on presence condition
  - Whenever the presence condition can be met, then the actual matching can also be done

- Our scope: edge annotations
  - Based on encoding of edges between nodes c and d as $c \rightarrow d \subseteq C_{ref}$

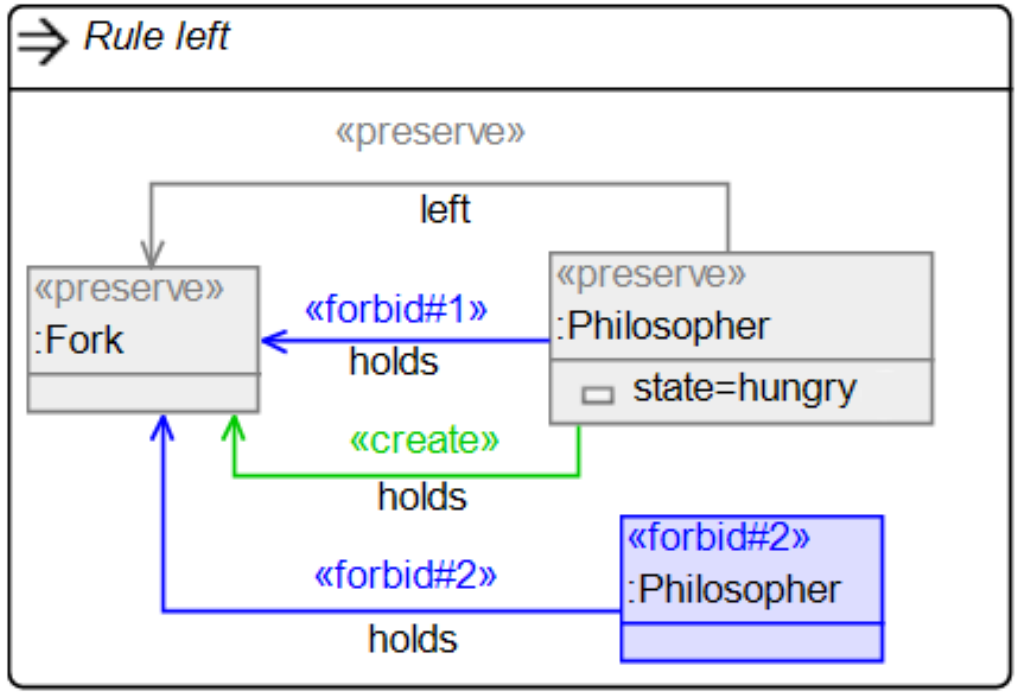
- Add presence condition pc as an implication of the actual matching:

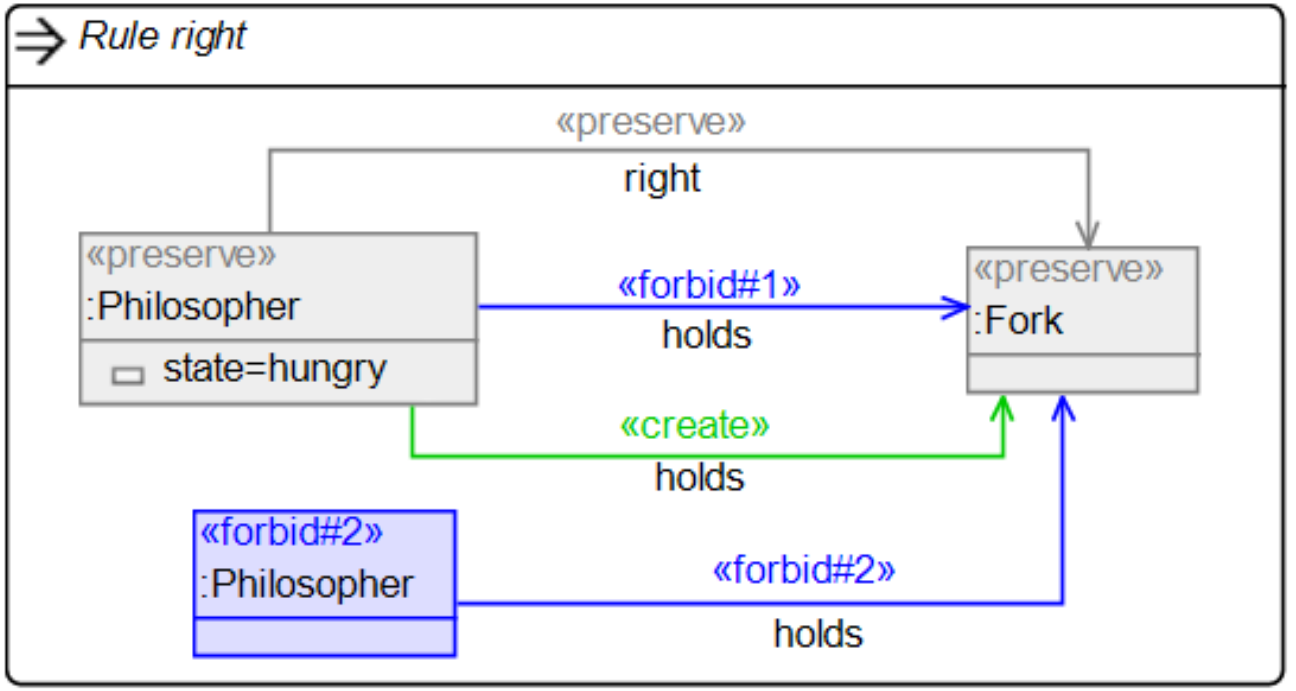
$$pc \rightarrow ((c \rightarrow d) \subseteq C_{ref})$$

Evaluation case: dining philosophers

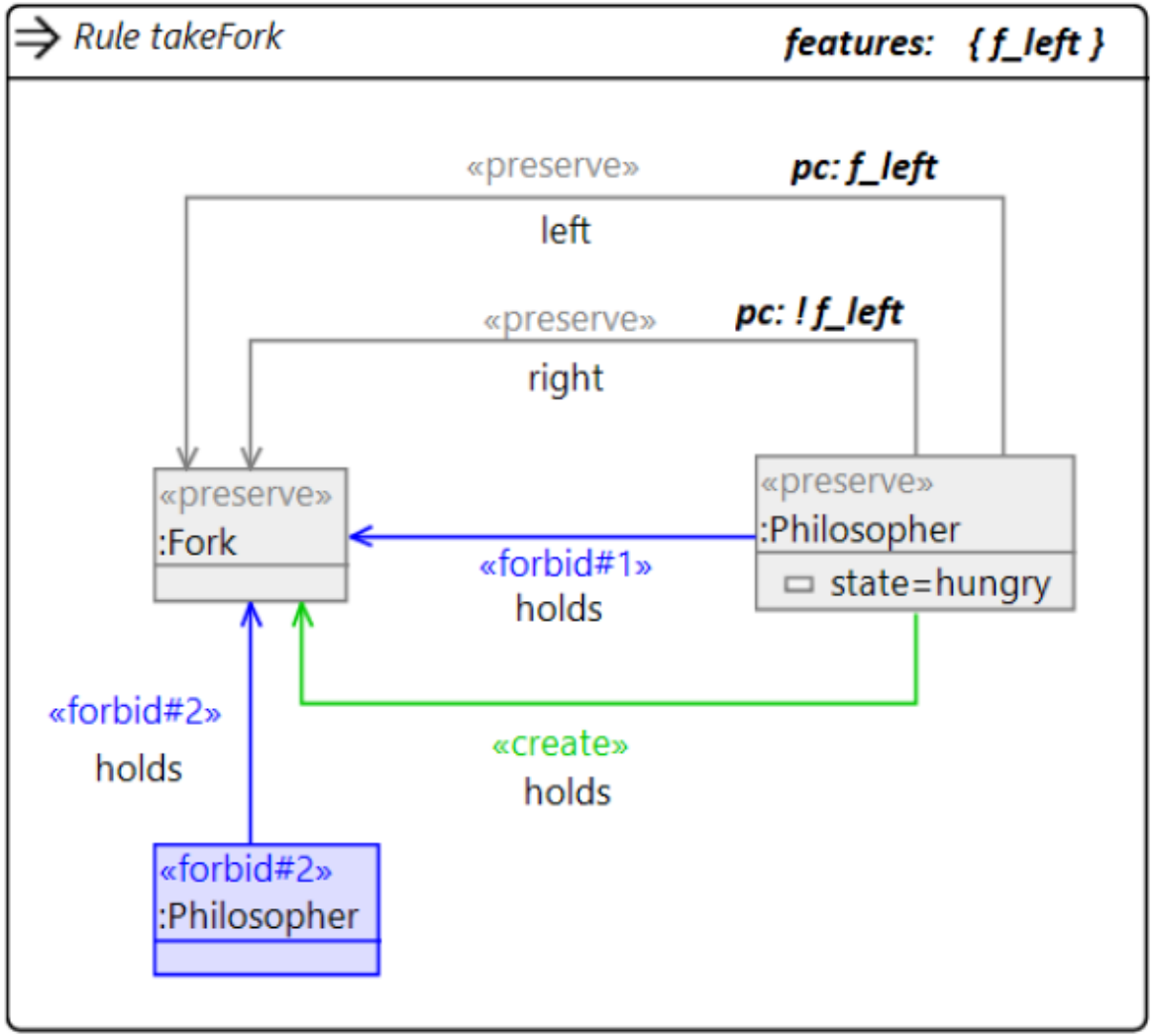


Evaluation case: dining philosophers

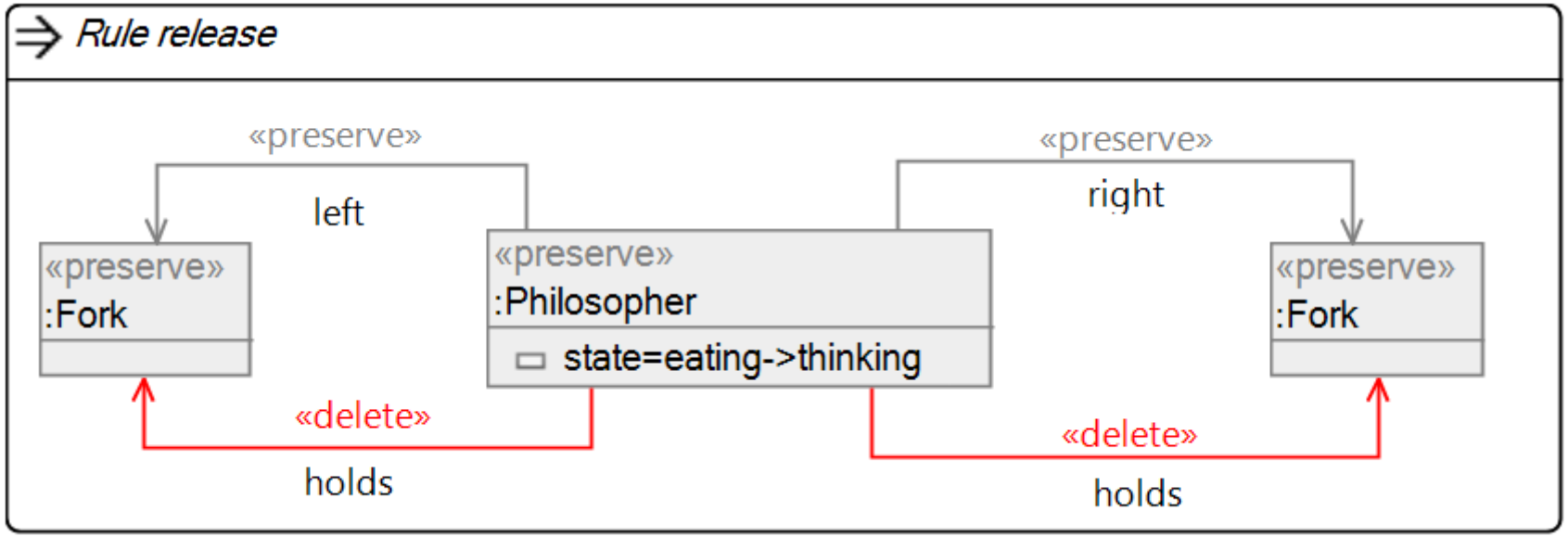


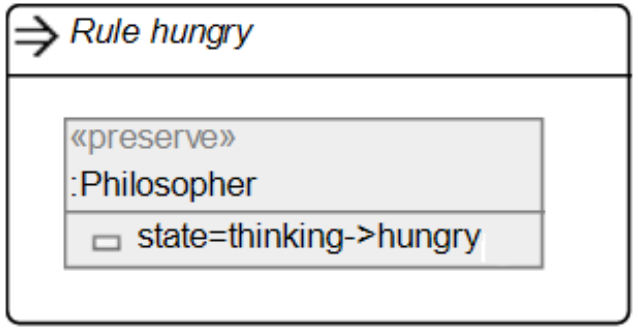


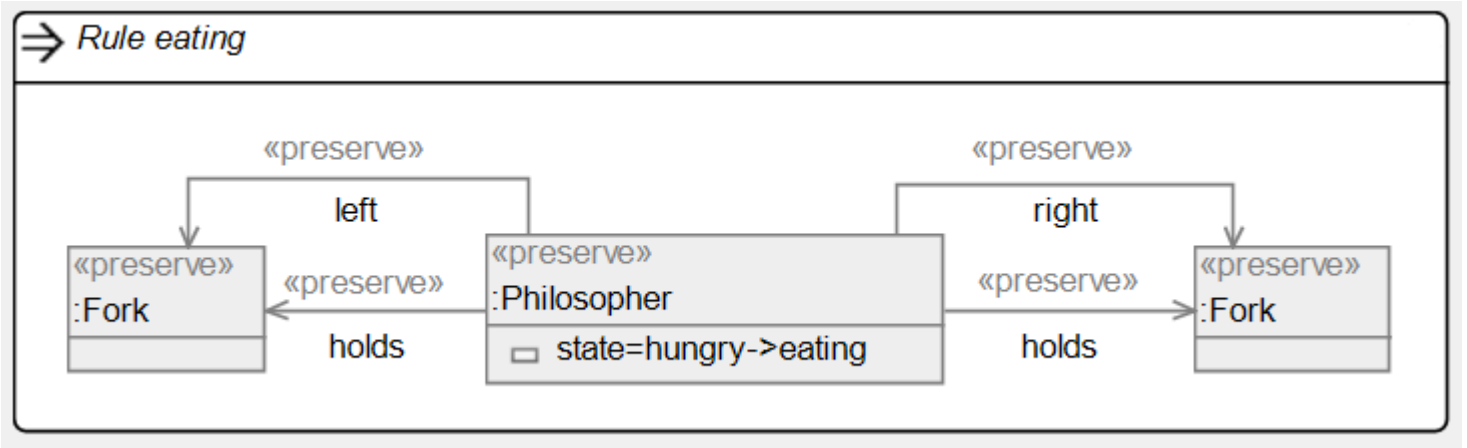
Evaluation case: dining philosophers



Evaluation case: dining philosophers



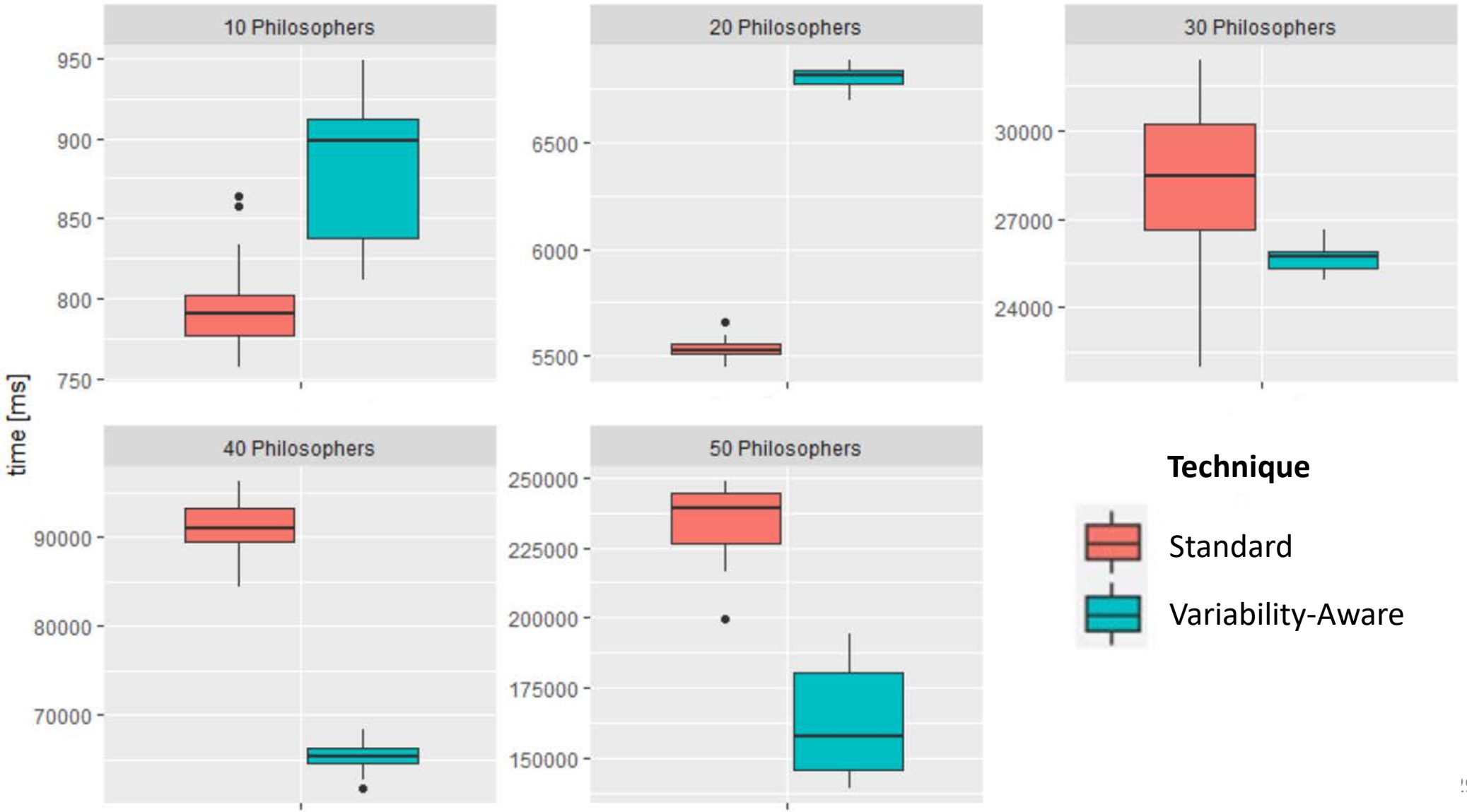




Performance evaluation: setup

- Evaluated reachability property $F\ eating$
- Compared standard vs. VB-aware Gryphon
  - Standard: 5 rules + input graph + property
  - VB-aware: 3 standard rules + 1 VB-aware rule + property
- Considered input graph with 10, 20, 30, 40, 50 philosophers
- Measures execution times for 30 runs each

Performance evaluation: results



Conclusion

- Clear performance improvements on input graphs and rules, especially larger scenarios (up to 45% faster)
- Shows potential for model checking graph transformations
- Performance gains are expected to scale

Limitations and future work

- Main limitation: currently only edge creation and deletion supported
- More rigorous soundness and performance argumentation
- More exhausitive empirical evaluation
- Addressing other model checking techniques and paradigms

Thank you!