Perspectives on Model Transformation Reuse

Marsha Chechik

June 2, 2016

Integrated Formal Methods (iFM'16)

A Brief and Partial Research History

Model-checking (Xchek) and formal specifications, para-consistent logics

Reasoning about incomplete and inconsistent systems

Modeling and reasoning about variability, product lines

Software compliance analysis

Mid-1990 2000s 2010s now time

Static analysis of programs, state machine specifications

Software model-checking (Yasm, UFO)

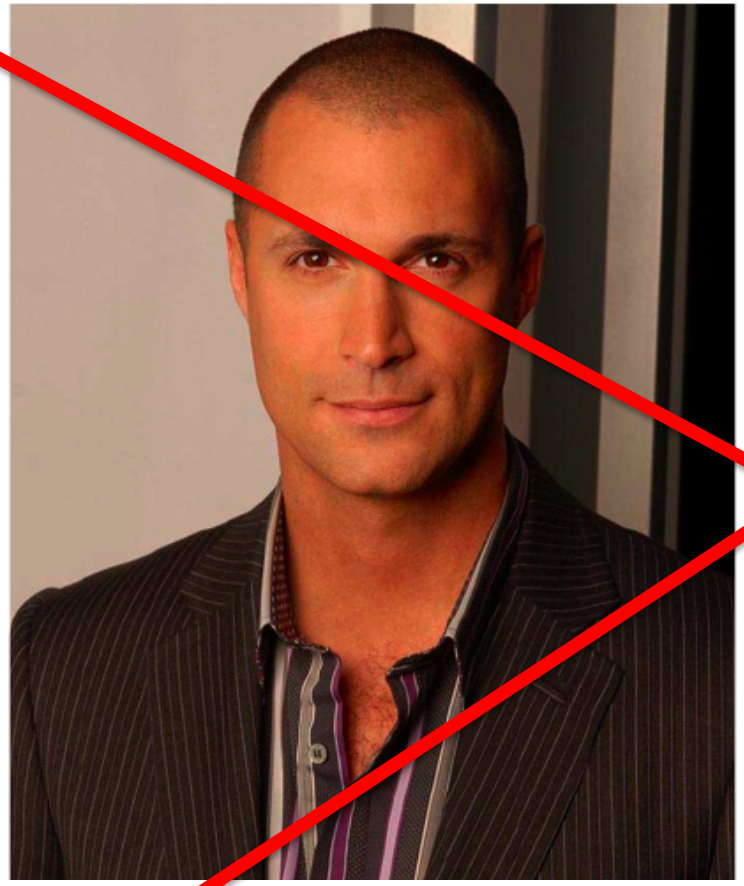
Runtime analysis of web service interactions

Reuse (feature-level)

Reuse (model transformation level)

Perspectives of Model Transformation Reuse

Googling “Model”



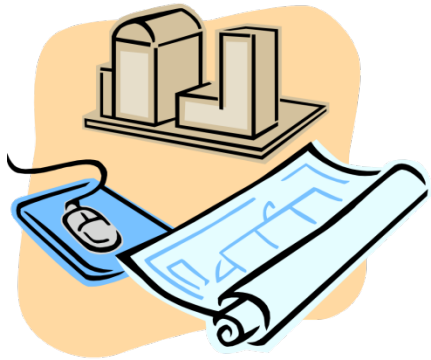
Model: “Purposeful abstraction”

Ideally, property-preserving!!!

Original

Models

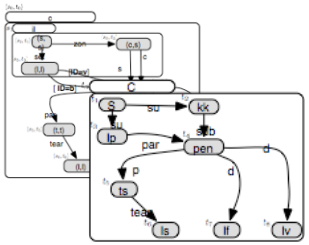
Why Models?



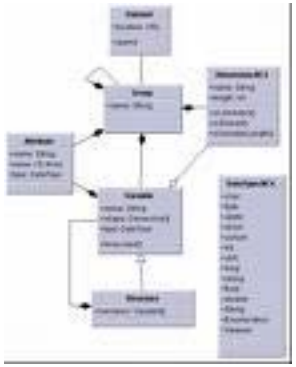
- Traditional Engineering Approach
  - Abstract & Precise
  - Amenable to analysis
  - Complexity: Model << System
- Pre-development and pre-deployment analysis
  - Early detection -> cheaper fixes
- Cost < Benefit

Software Engineering Models

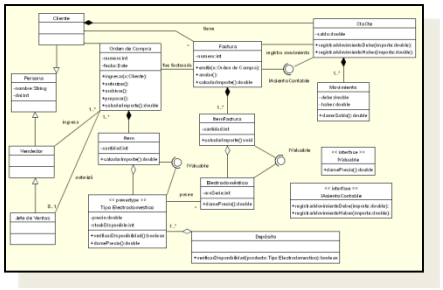
Abstraction → Structure → Purposeful Reasoning



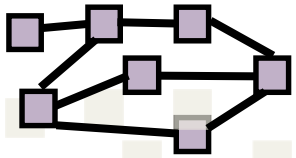
Behaviour



Concepts

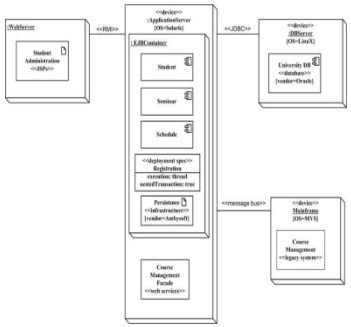


Static Design

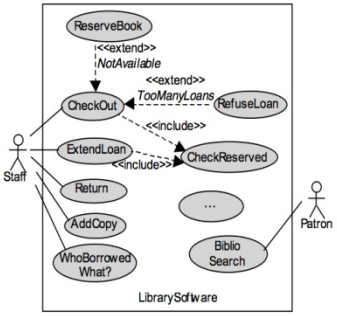


Architecture

Requirements



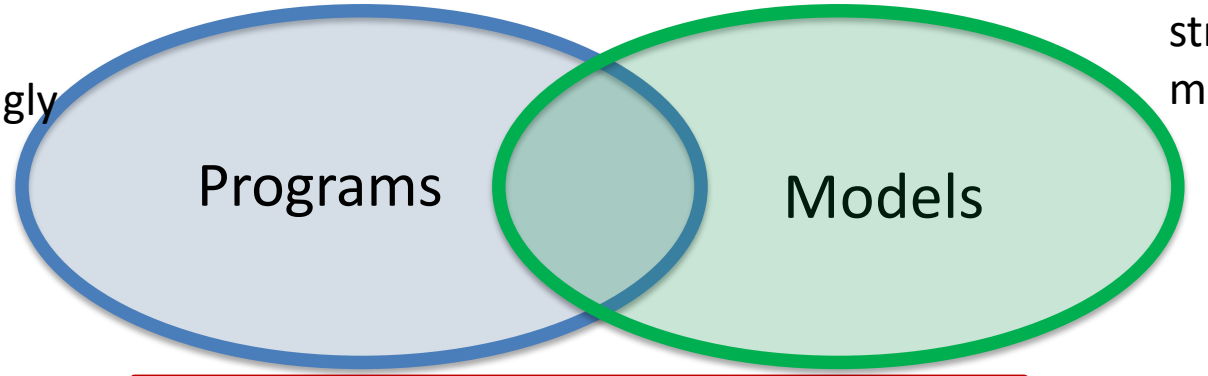
Deployment



Use Cases

Models vs Programs

- Executable
- Present in abundance “in real world”
- Complex data structures and control flow
- Typically strongly typed



- Higher level of abstraction (“for a purpose”)
- Confirms to the strongly-typed meta-model

Goals:
- Quality
- Speed of development
- Understandability

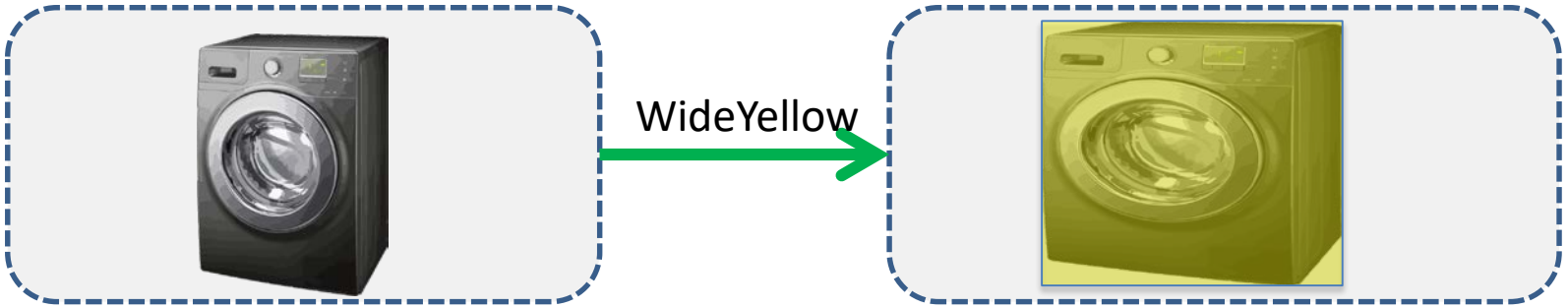
(Formal) Methods:
- Specification
- Analysis

Transformations

* requisite cute picture of animals

Why Transformations

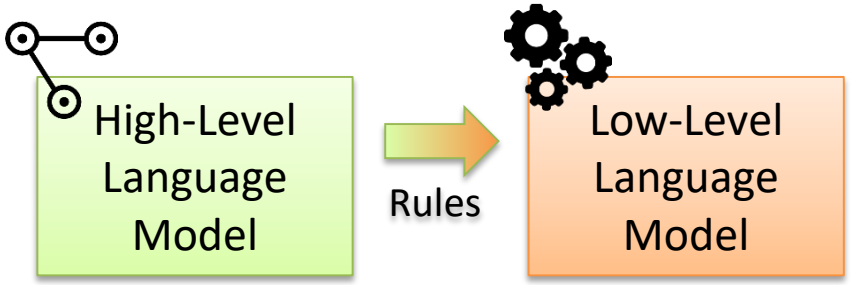
- Use to convert one artifact into another



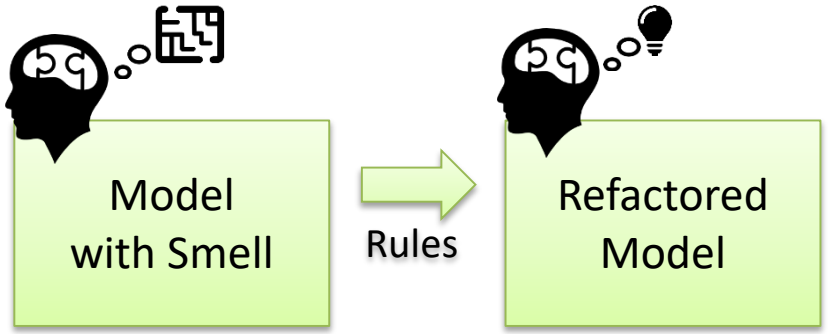
- Automates mundane tasks and ensures quality
- Enables raising level of abstraction in software development

Model Transformations

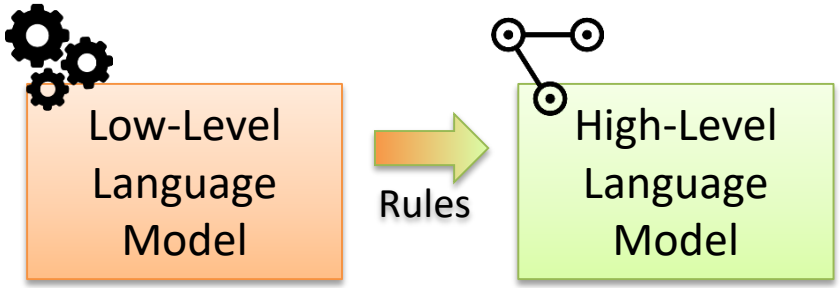
Key enabler of model driven development



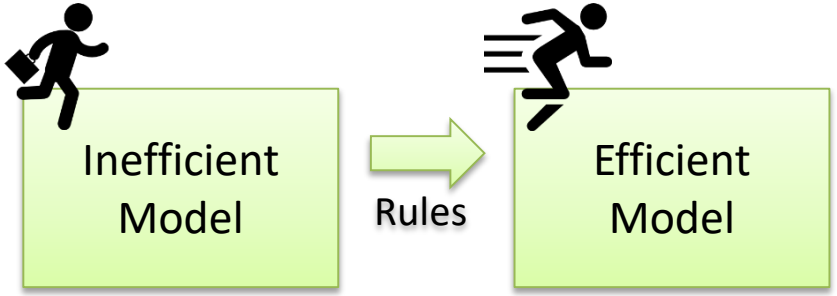
Forward Engineering/Refinement



Refactoring



Reverse Engineering



Performance Optimization

Sendall, Shane, and Wojtek Kozaczynski. *Model Transformation - the Heart and Soul of Model-Driven Software Development*. No. LGL-REPORT-2003-007. 2003.

Characteristics of Model Transformations

In principle, an arbitrary program

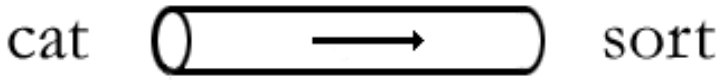
Transformation

In practice:

Strongly typed (by input and output models)

One-step task with specific intent

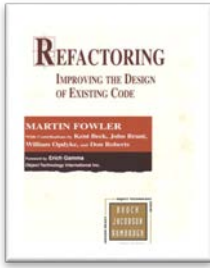
Aimed to be chained together, like Unix pipes



A UNIX pipe provides one-way communication between two processes on the same computer

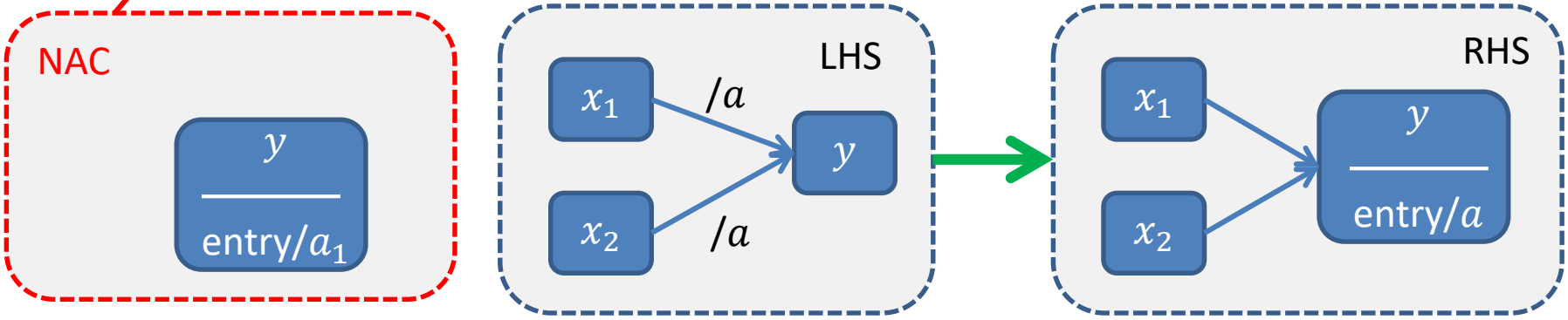
Often implemented in languages which allow easy graph manipulation

Example: State Machine Refactoring Transformation



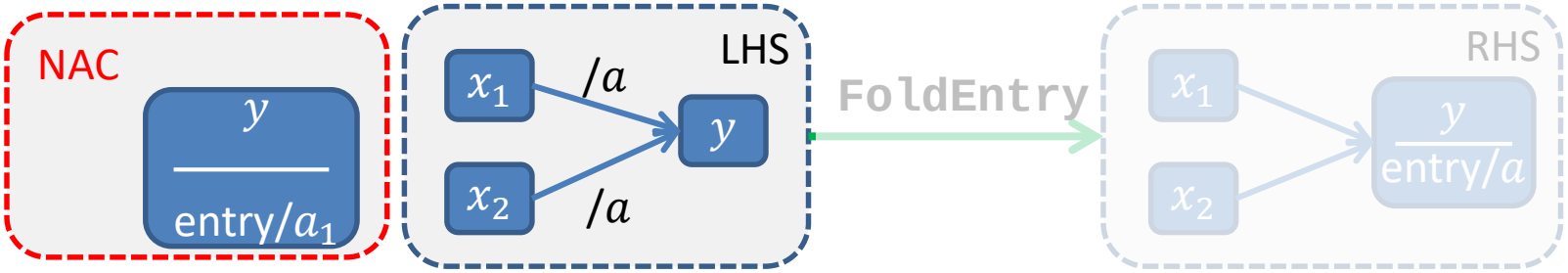
- Transformation **FoldEntry**
  - Simplifies state machine
  - Moves common actions on input transitions to the entry action of the target state
- Usin n rule:

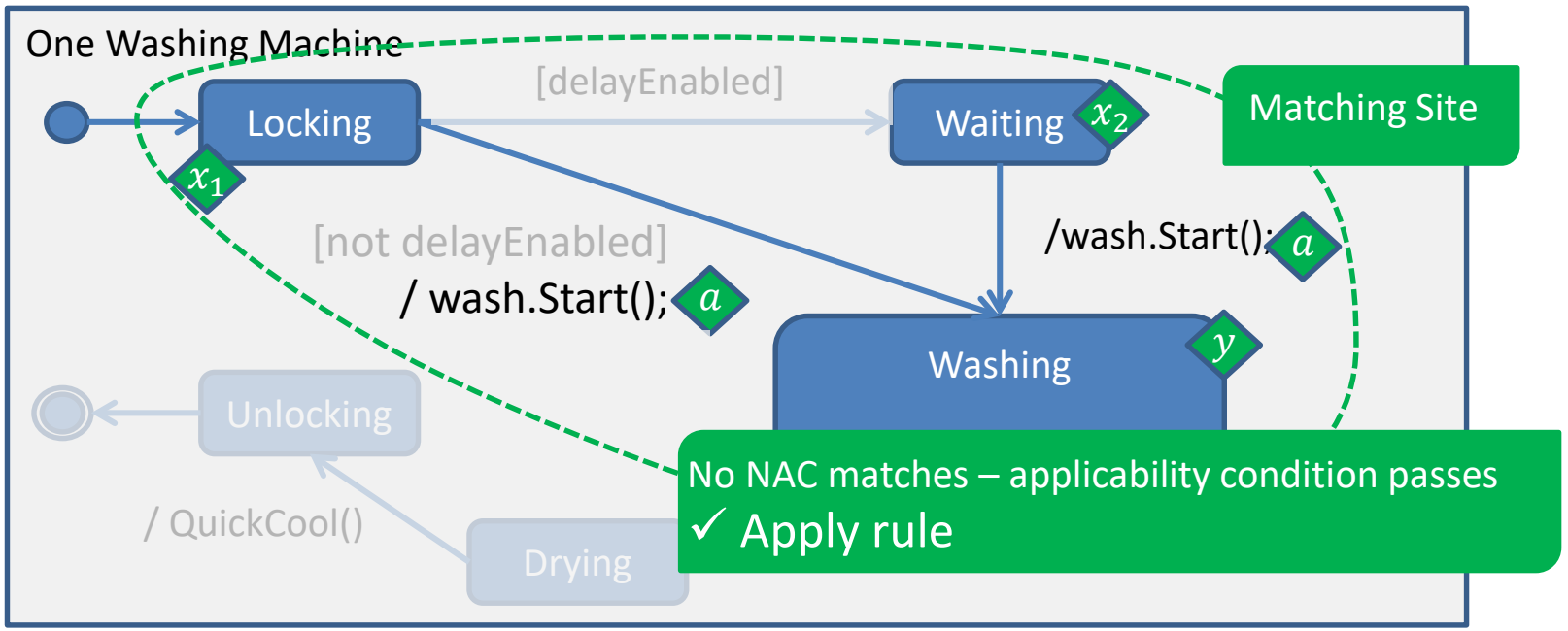
Negative Application Condition



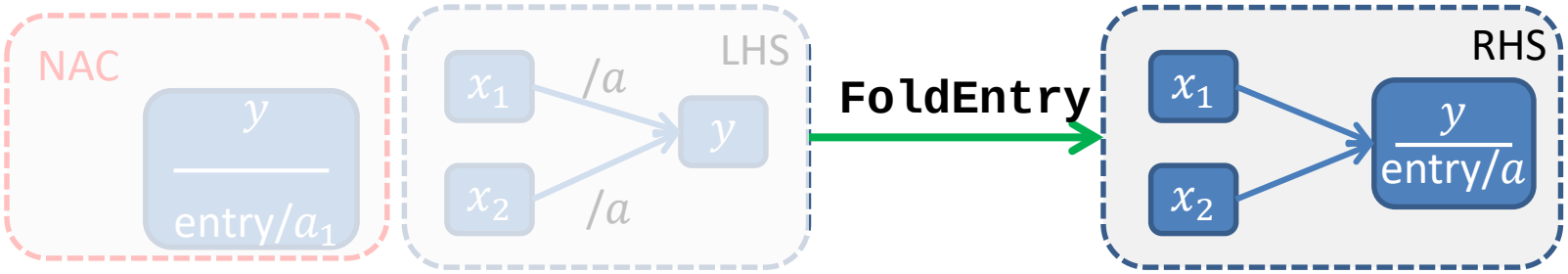
Applicability Condition:
Apply the rule if the LHS matches and no NAC matches

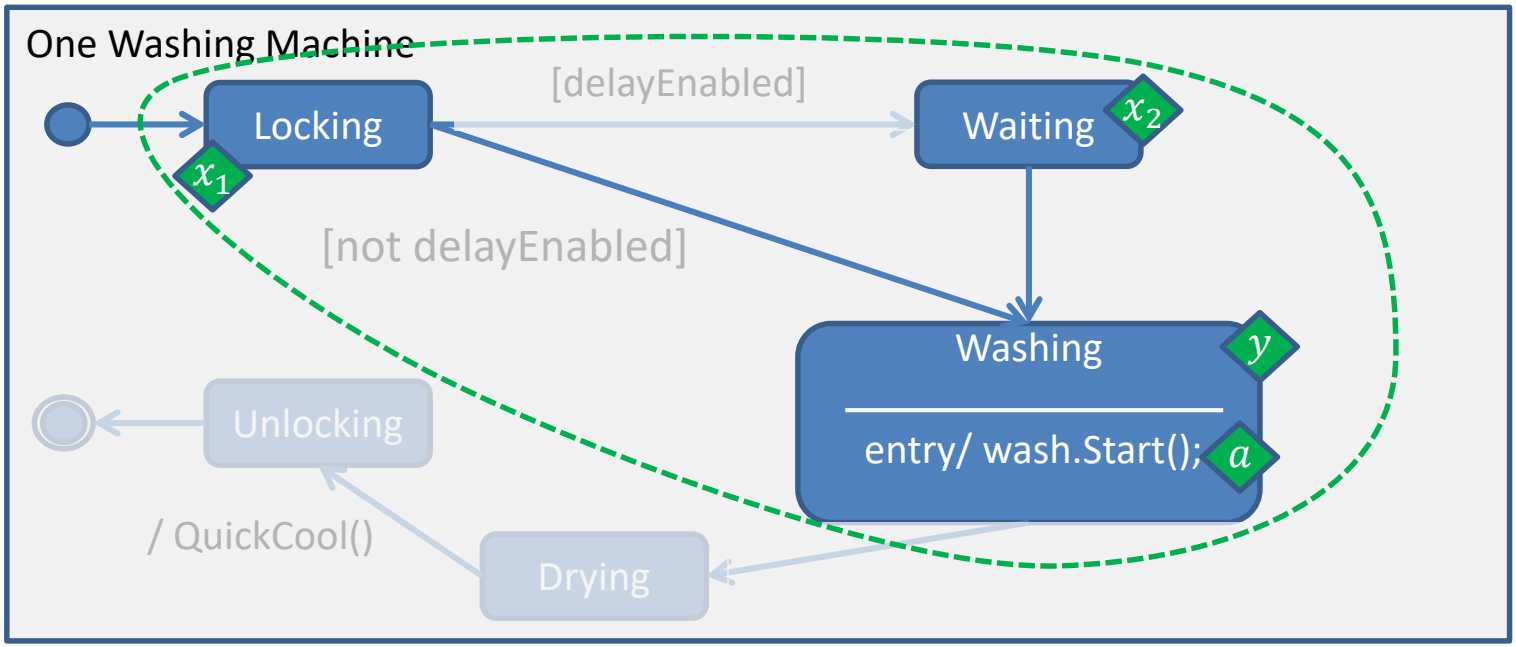
Applying `FoldEntry` – Example 1



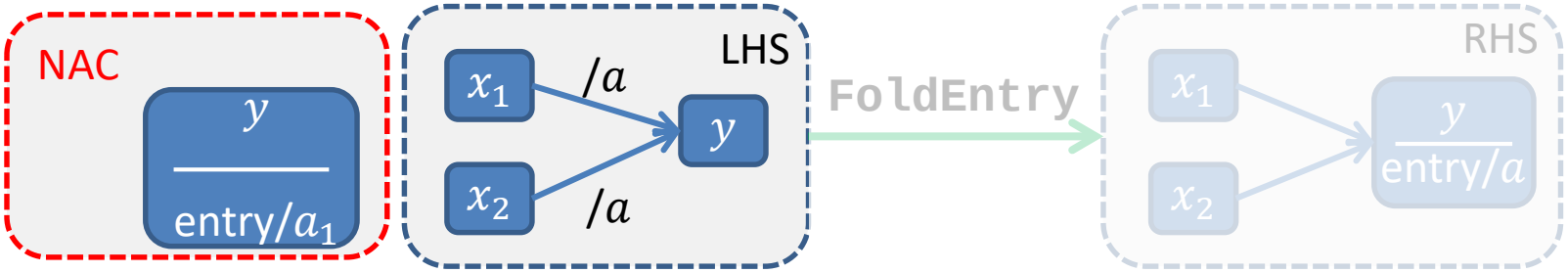


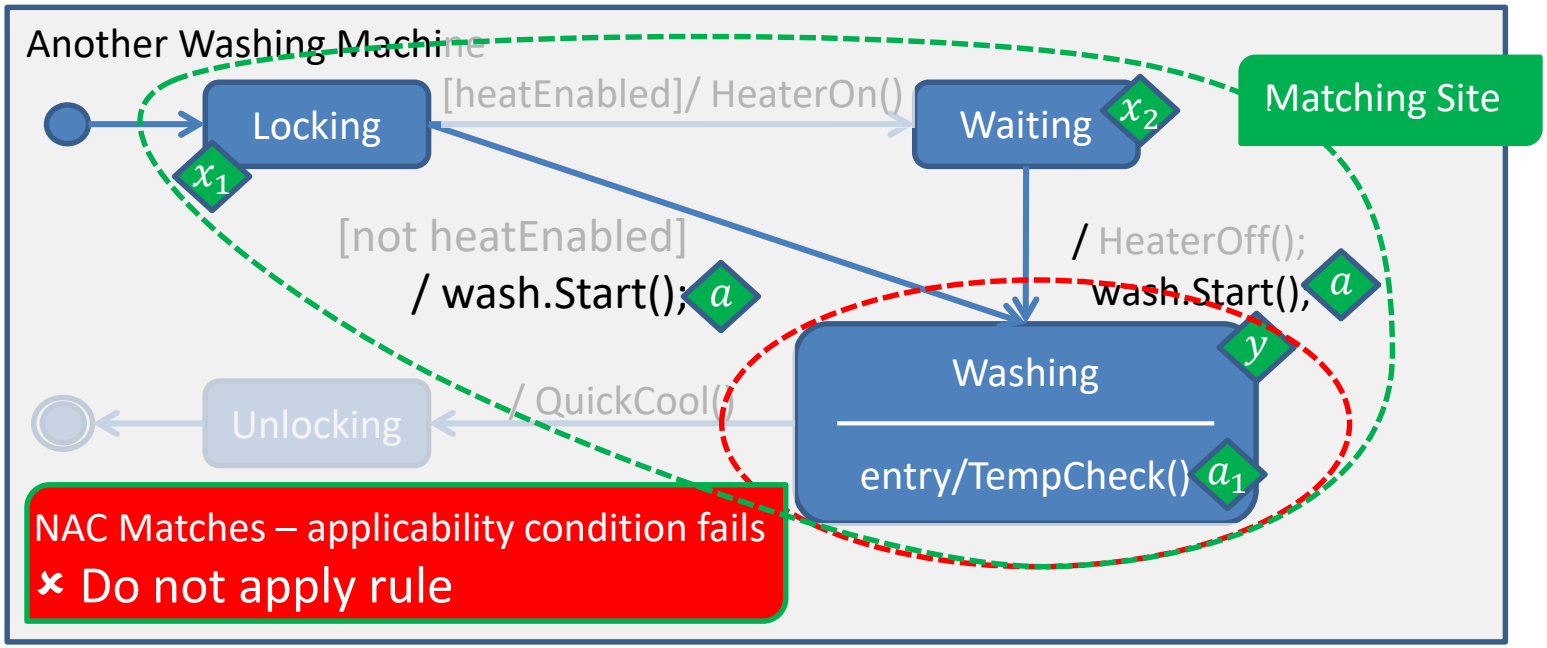
Applying **FoldEntry** – Example 1





Applying `FoldEntry` – Example 2

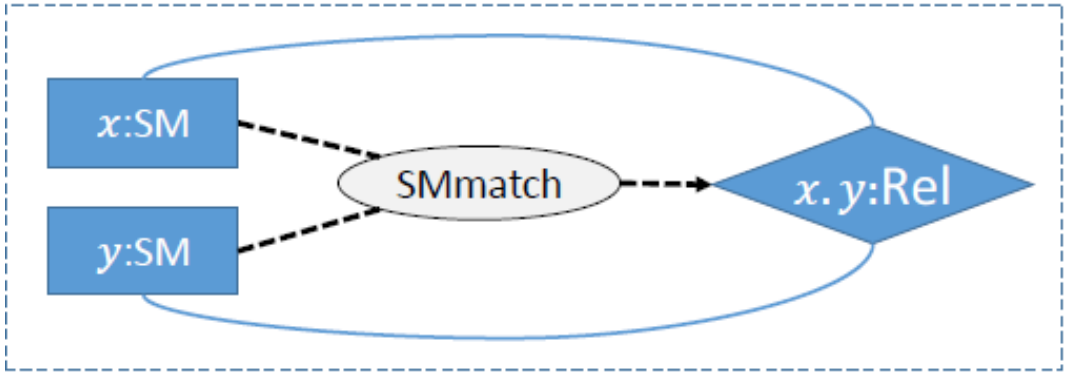




Transformation Signatures

- Single model to single model

- Two state machines to a mapping (or relationship)



Perspectives of Model Transformation

Reuse

Reuse Transformations

- Why?
  - Same reason as program reuse:
    - Reduce effort
    - Accelerate development
    - Increase quality
- Where?
  - Within same process
  - Across processes

How to Reuse?

Transformation reuse

vs

general software reuse

– Adapt, generalize, “reinvent” program reuse techniques
– Create novel modeling-specific approaches

Perspectives of Model Transformation Reuse


MSC
HEREFORD
STEIKHÚS
WHALE MENU
STARTER
Lobster soup with Cognac
MAIN COURSE
Whale peppersteak served with fried vegetables, potato and pepper sauce
DESSERT
Icelandic Skyr Herefordstyle
Price kr. 5.900
PUFFIN MENU
STARTER
Smoked Puffin with fresh salad and raspberry vinegar
MAIN COURSE
Grilled breast of Puffin served with fried vegetables, potato and malt sauce
DESSERT
Icelandic Skyr Herefordstyle
Price kr. 5.900

- Motivation
  - Models and Transformations
  - Why Reuse

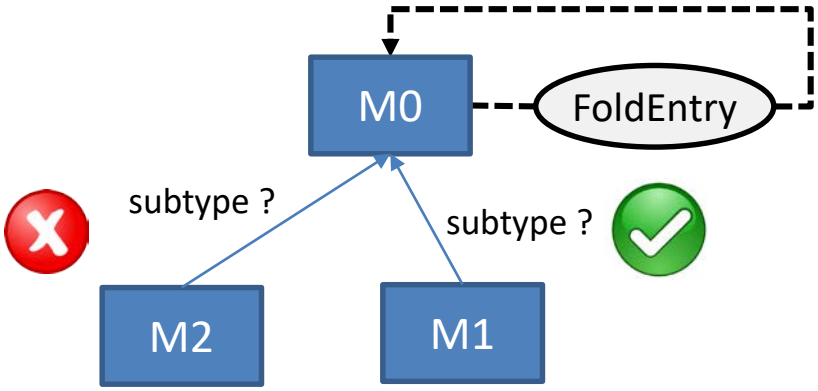
- Transformation Reuse
  - PL adaptations: subtyping and mapping
  - MDE-specific approaches: lifting and aggregating
- Future perspectives

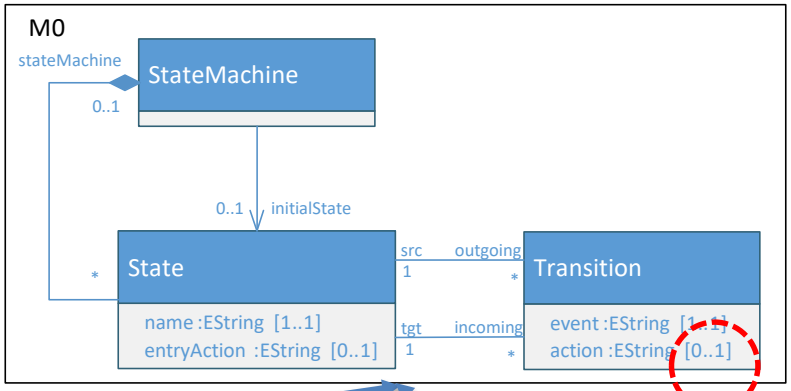
Adaptation: Subtyping

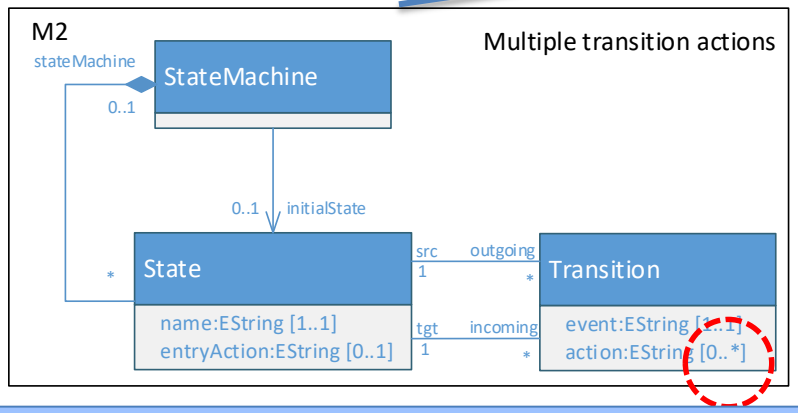
Simple Subtyping: PL

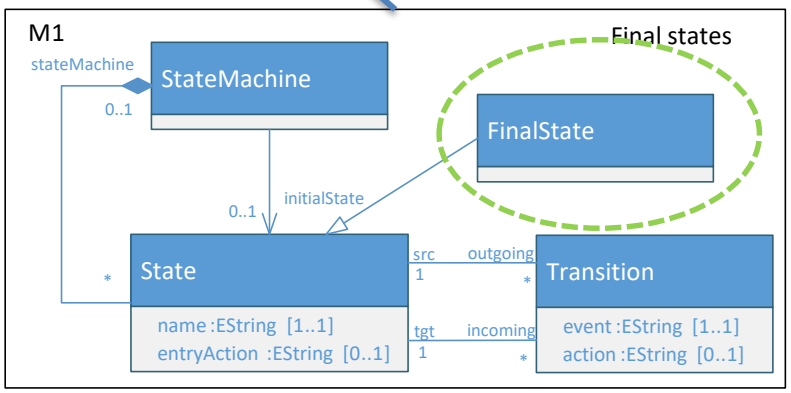
- *Int* is a subtype of *Real* since $[\![Int]\!] \subseteq [\![Real]\!]$
- So function *Name: Real→String* can be applied to Int inputs
  - *Name* (3.14) = "3.14"
  - *Name* (3) = "3"

Simple Model Subtyping









So, FoldEntry can be reused for M1 but not for M2

Coercive Subtyping: PL

Requires an explicit conversion function

Example:

– a conversion function *Int2Str*: *Int* → *String*…

- … allows any *Int*-valued input to be coerced into a *String* value

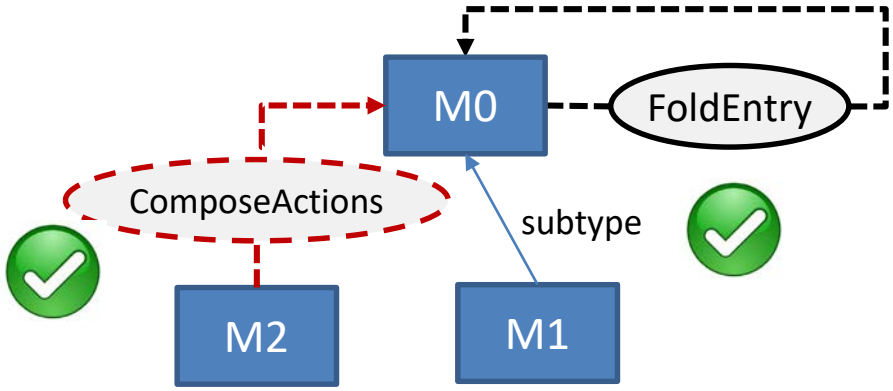
– function *cat*: *String* × *String*→*String*

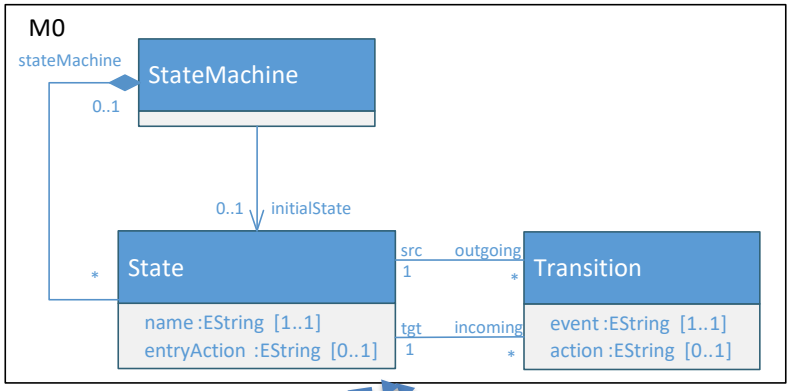
- *cat* (“hello”, “world”) ⟼ “hello world”
- *cat* (“high”, 5)

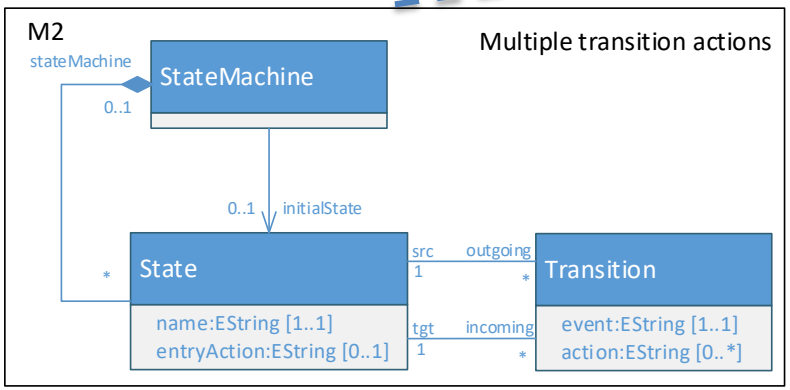
  ⇝ *cat* (“high”, *Int2Str*(5))

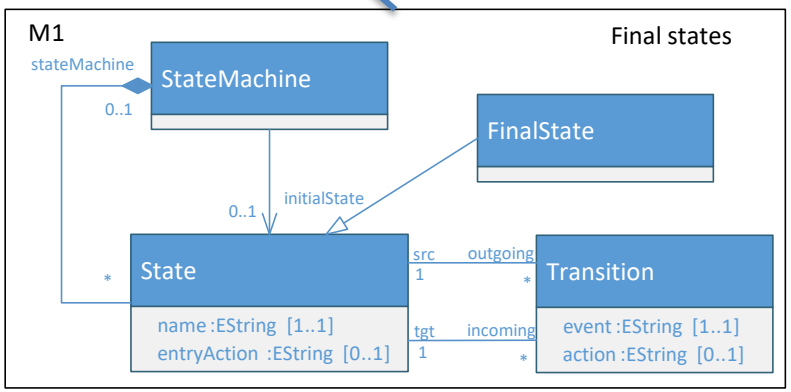
  ⟼ “high 5”

Coercive Model Subtyping









So, FoldEntry can be reused for M1 and M2

Adaptation: Mapping

[MODELS'15]

Programming Language *MAP*

- *Map <F> (L)*

  Apply function *F* to elements of list *L*

- Example:

  Function *Double*: *Int* → *Int*

  Double (2) ⟼ *4*

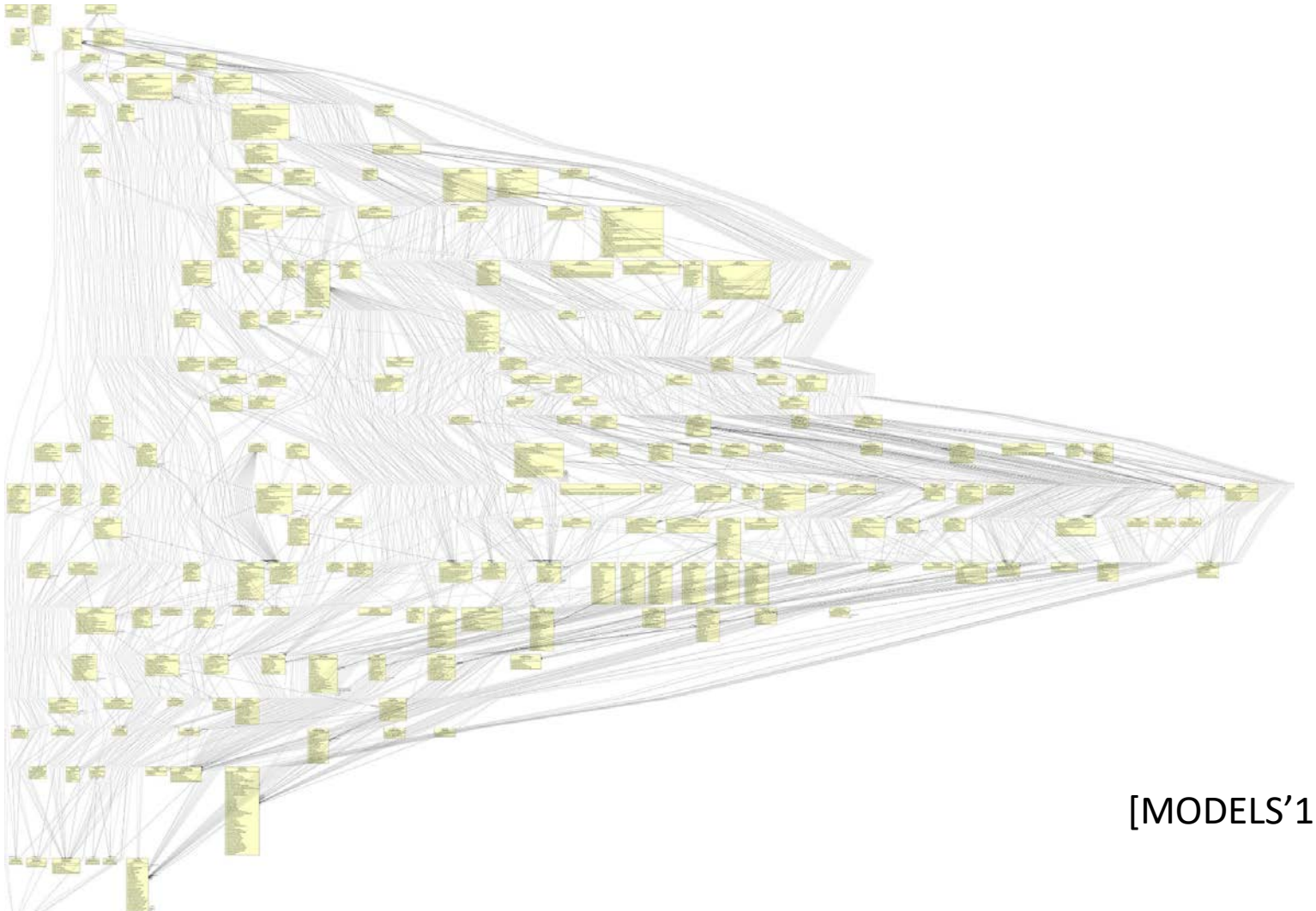
  Map <Double>([1, 2, 3, 4])

  ⟼*[2, 4, 6, 8]*

[MODELS'15]

Goal: apply *Map* to megamodels

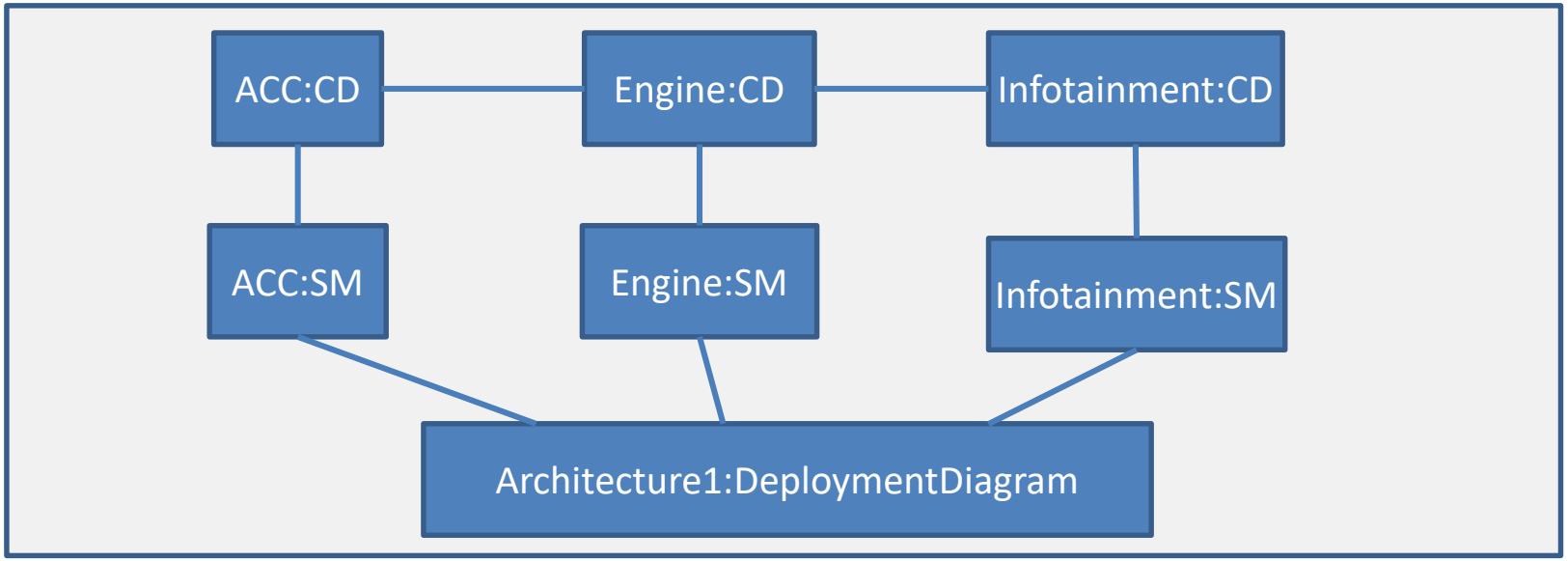
A Megamodel is not just a really big model!



[MODELS'15]

Megamodels

Represent models and their relationships at a high level of abstraction to facilitate model management.



Context: Large software projects ⇒ proliferation of model artifacts

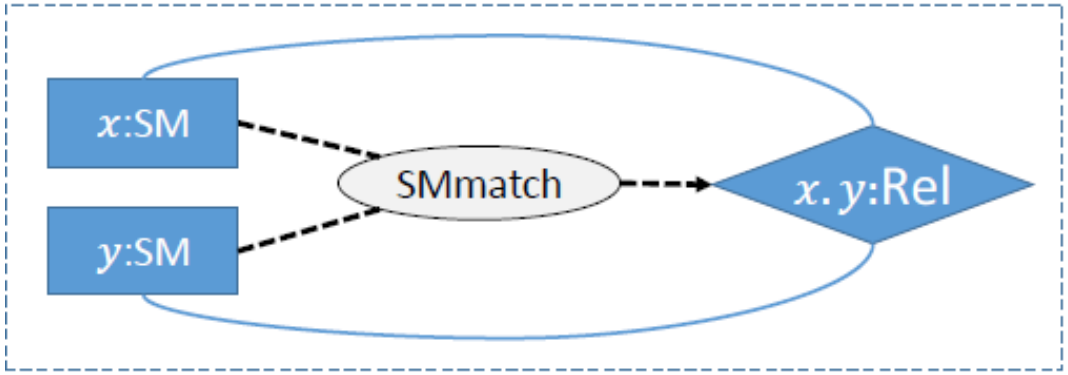
- Need to manage this "accidental complexity"

[MODELS'15]

Map for Megamodels

Key Challenges for *Map*

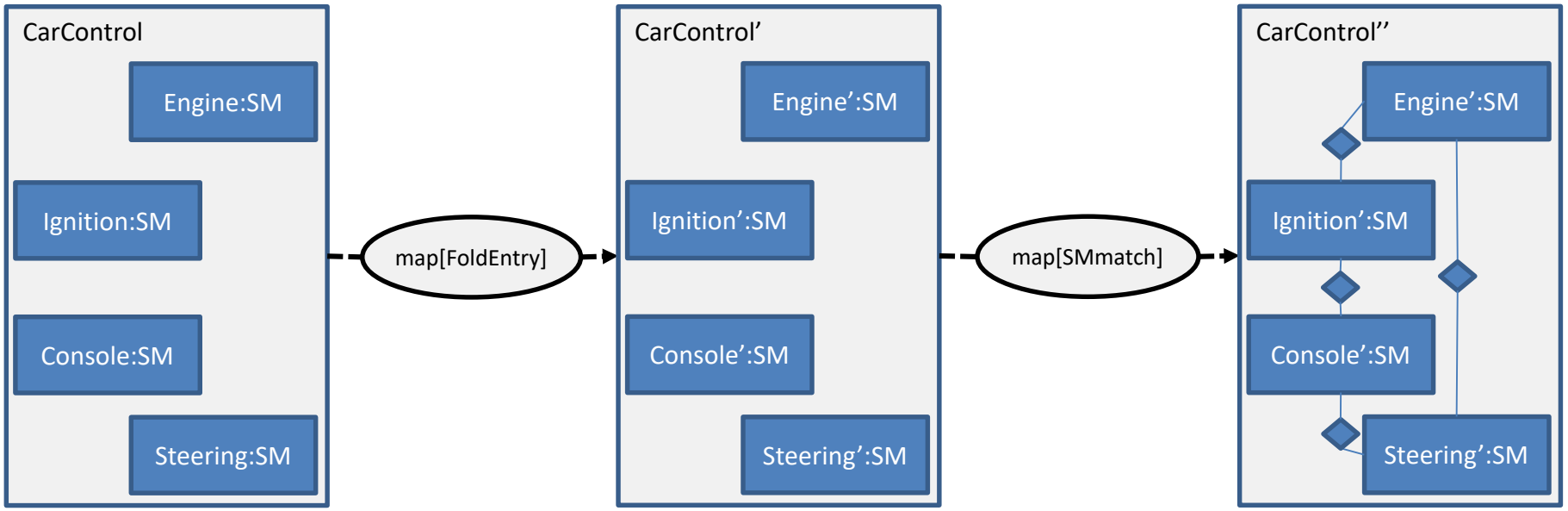
1. Must correctly manipulate entire graphs of related models rather than just sets *of models*.
   - Graph edges (i.e., relationships) have content
2. Must work with transformation signatures
   - Transformations accept graphs of models and relationships as input and output



[MODELS'15]

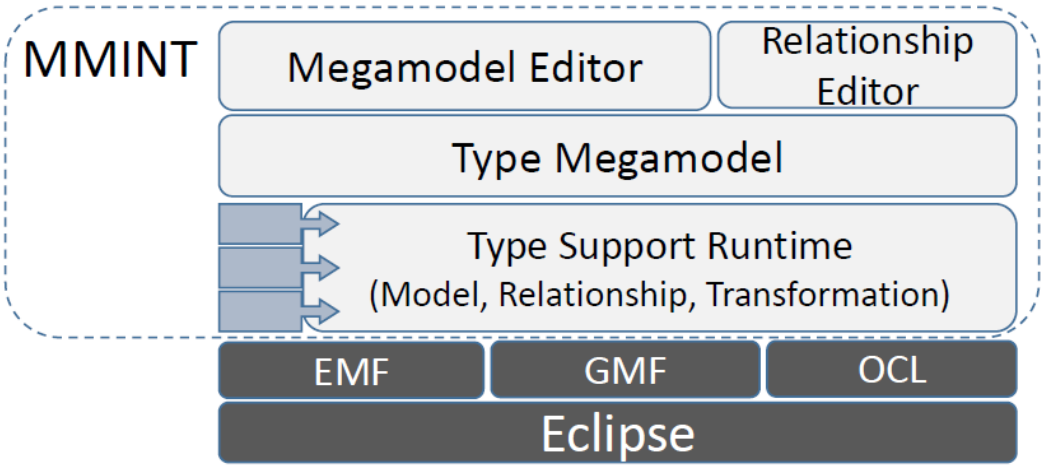
Map Applied to Megamodels

1. First apply **FoldEntry** refactoring
2. Then apply **SMmatch** transformations to find state machine correspondences



[MODELS'15]

Tooling: MMINT



- Model Management INTeractive workbench
  https://github.com/adissandro/MMINT
- Support for strongly typed models:
  - simple and coercive subtyping
  - lazy coherence checking for coercion
  - type downcasting when model conforms to subtype
- Support for megamodels
  - map / reduce /filter

[MODELS15-tool]

Summary of Adaptation Approaches

Subtyping

Mapping

Other

Generic programming

Model concepts

- Motivation
  - Models and Transformations
  - Why Reuse
- Transformation Reuse
  - PL adaptations: subtyping and mapping
  - MDE-specific approaches: lifting and aggregating
- Future perspectives

Novel Approaches: Lifting

lift

[ICSE'14]

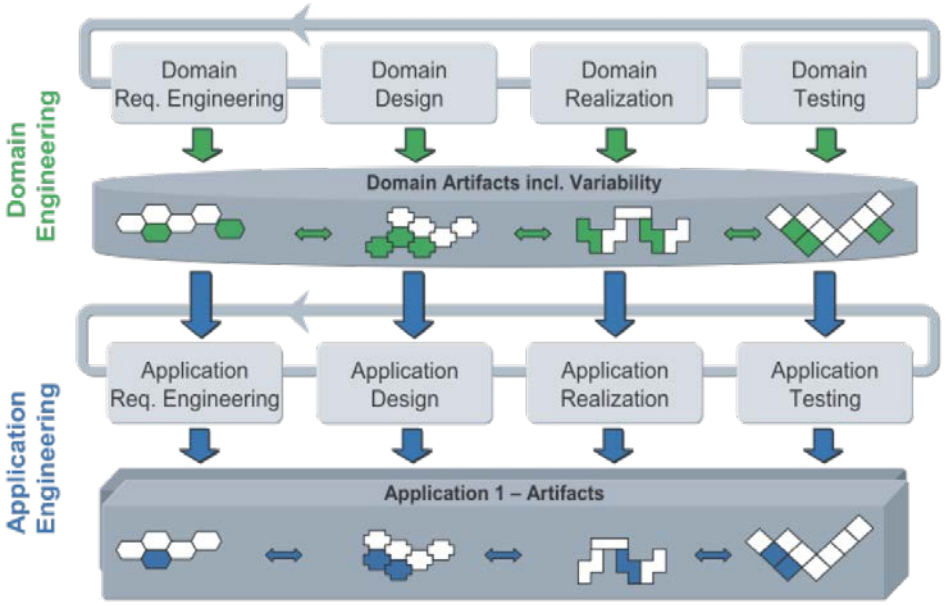
A slight aside: Product Lines

- Goal: Help develop, manage, reuse a large number of similar but different artifact variants (products)
- Example: Washing Machine Co.

[ICSE'14]

Software Product Line Engineering

a discipline that promotes planned and predictive software reuse

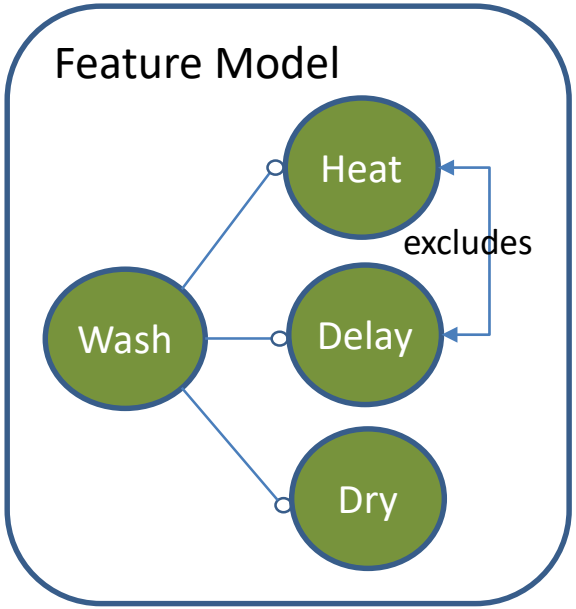


K. Pohl et al., Software Product Line Engineering: Foundations, Principles, and Techniques, 2005

P. C. Clements and L. Northrop, Software Product Lines: Practices and Patterns, 2001

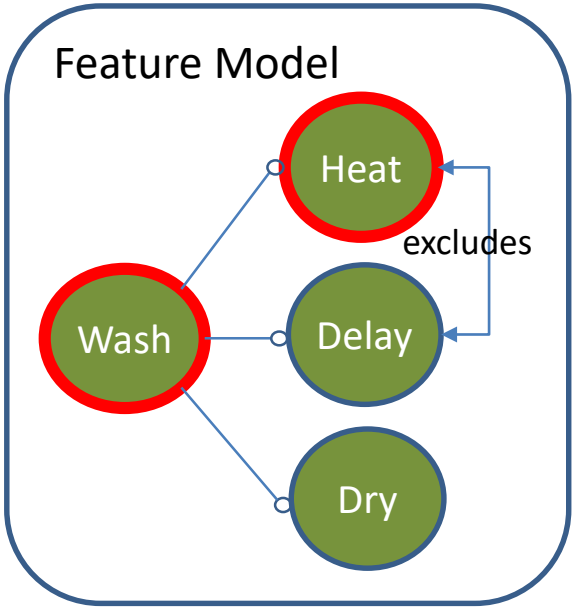
Product Line Structure + Terminology

- Product line (annotative) represented by
  - Domain Model – combined parts from all products, annotated by features (presence conditions). A.k.a. 150% representation.
  - Feature Model – shows possible features and restrictions for product combinations
- Example : Washing Machine Co.



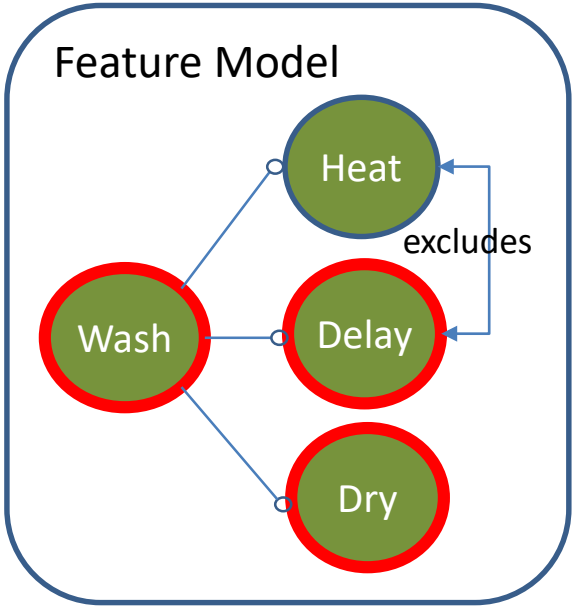
Product Line Configuration – Example 1

- +Heat product
  - Feature configuration: {Wash, Heat}

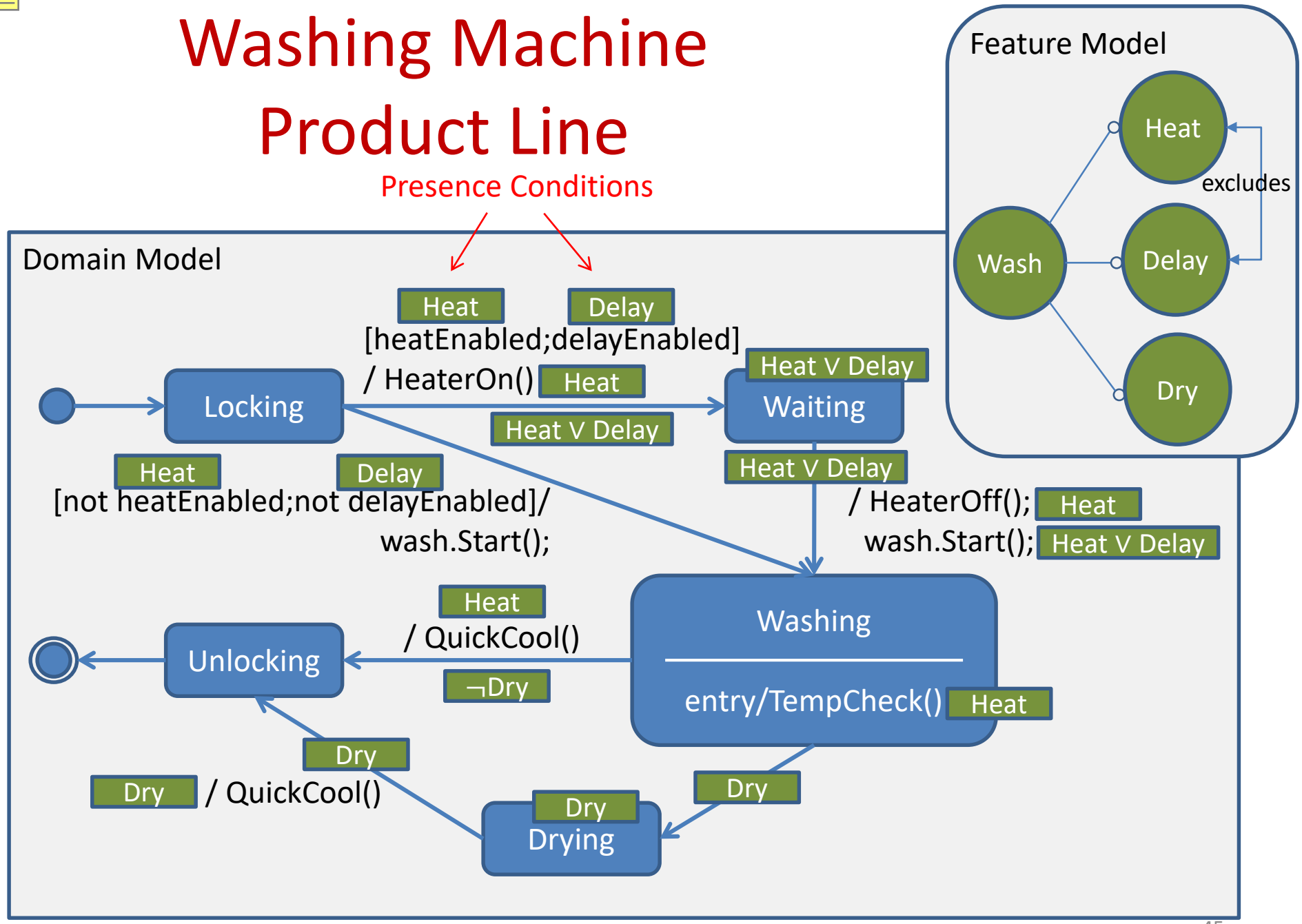


Product Line Configuration – Example 2

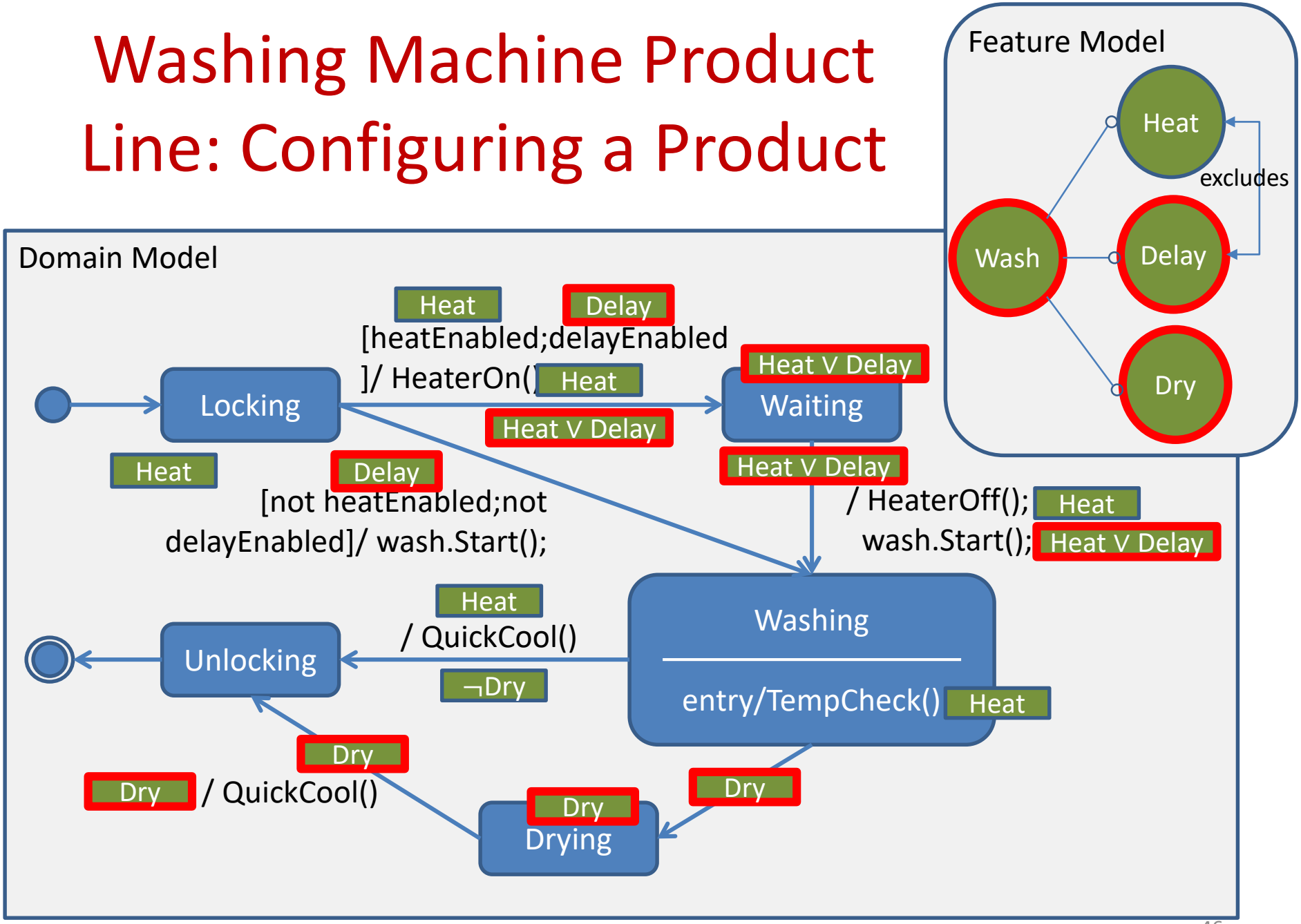
- +Dry/Delay product
  - Feature configuration: {Wash, Dry, Delay}



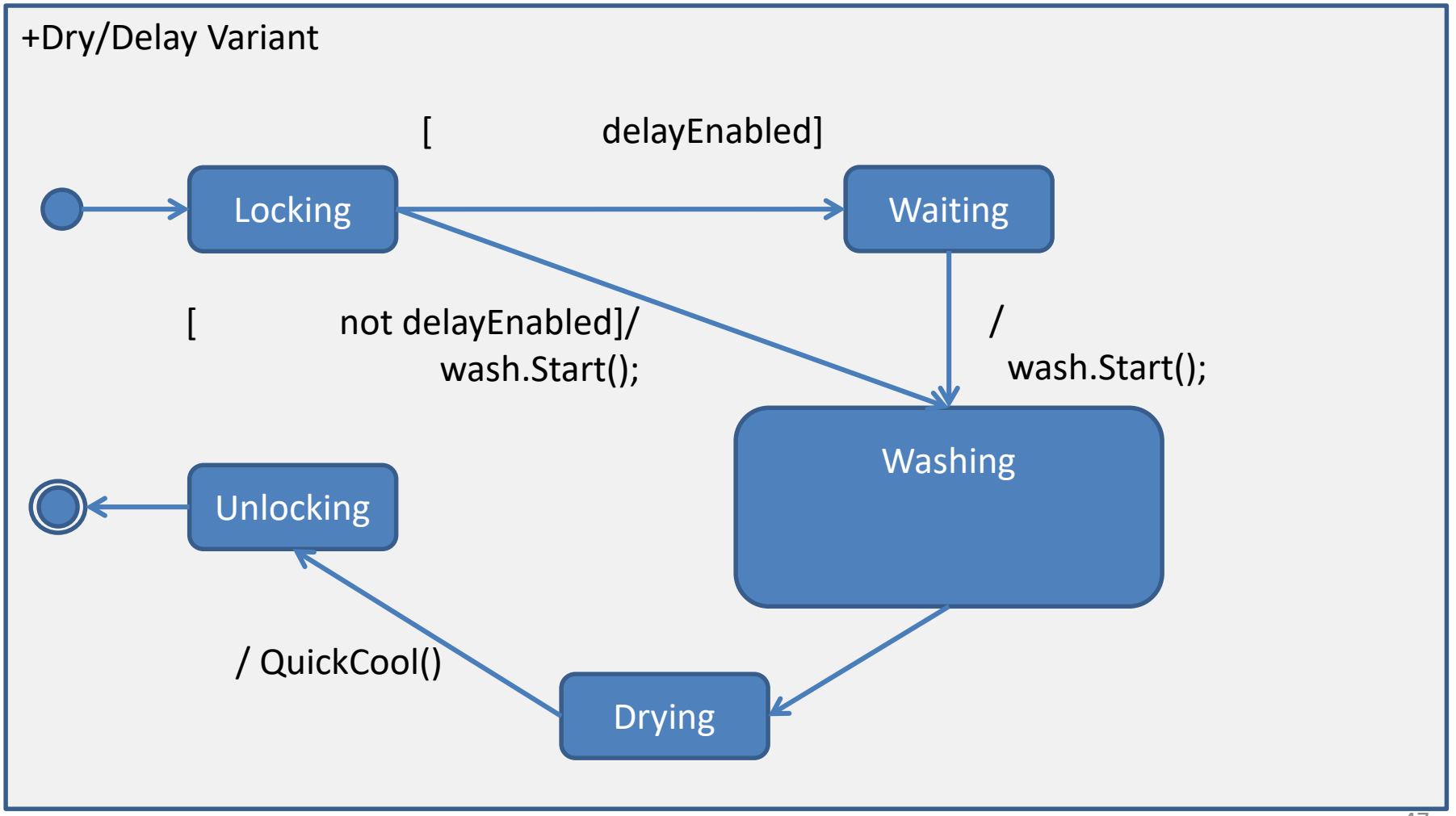
Washing Machine Product Line



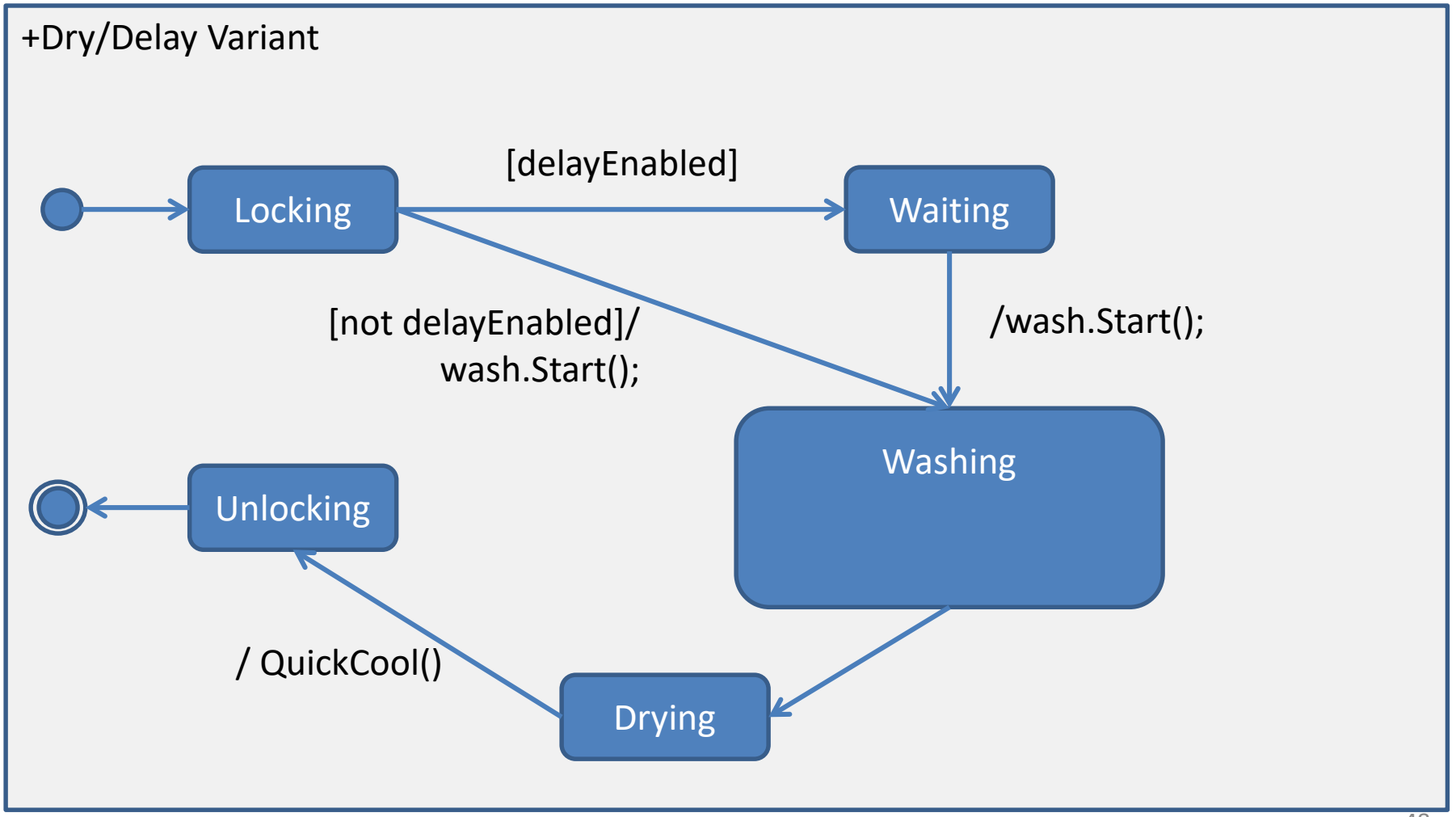
Washing Machine Product Line: Configuring a Product



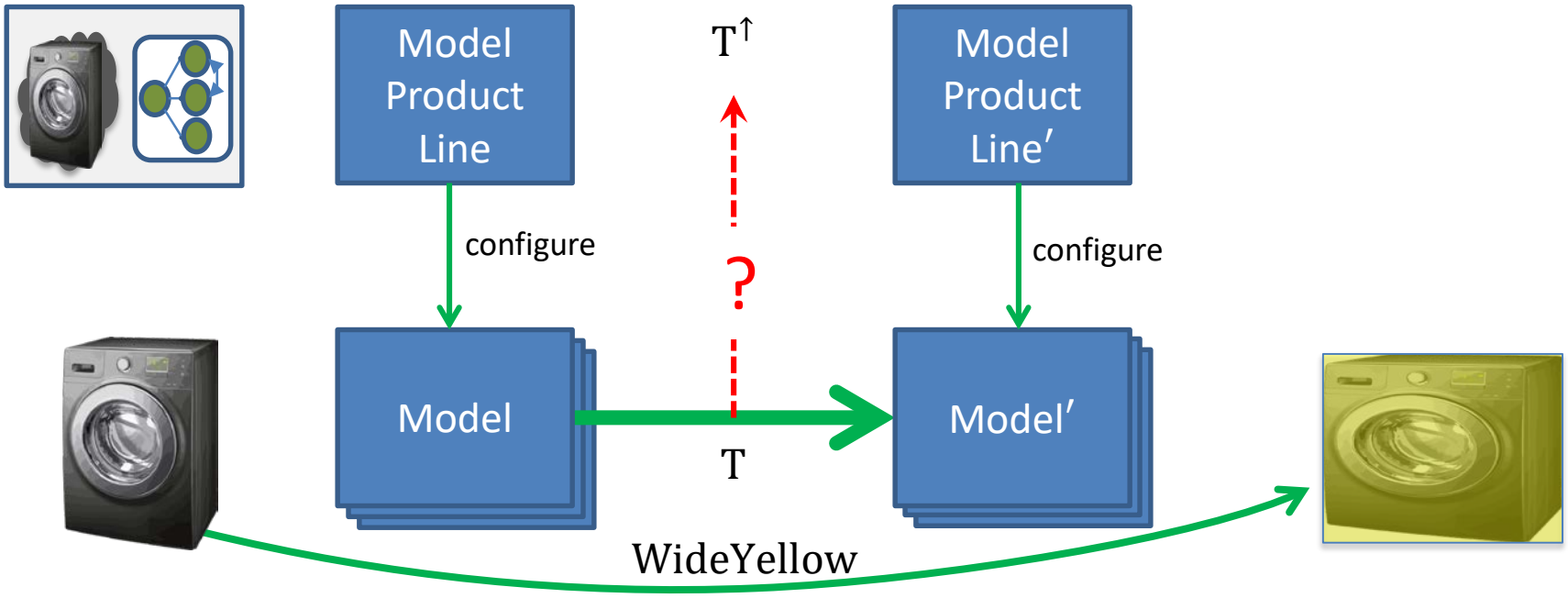
Result: +Dry/Delay State Machine



Result: +Dry/Delay state machine



Our Goal: Reuse Transformation Defined for Products for Entire PLs



- Key problem: Transformations written for models cannot be used directly with product lines of models
- Ideally we should lift them to product lines - but how?

[ICSE'14]

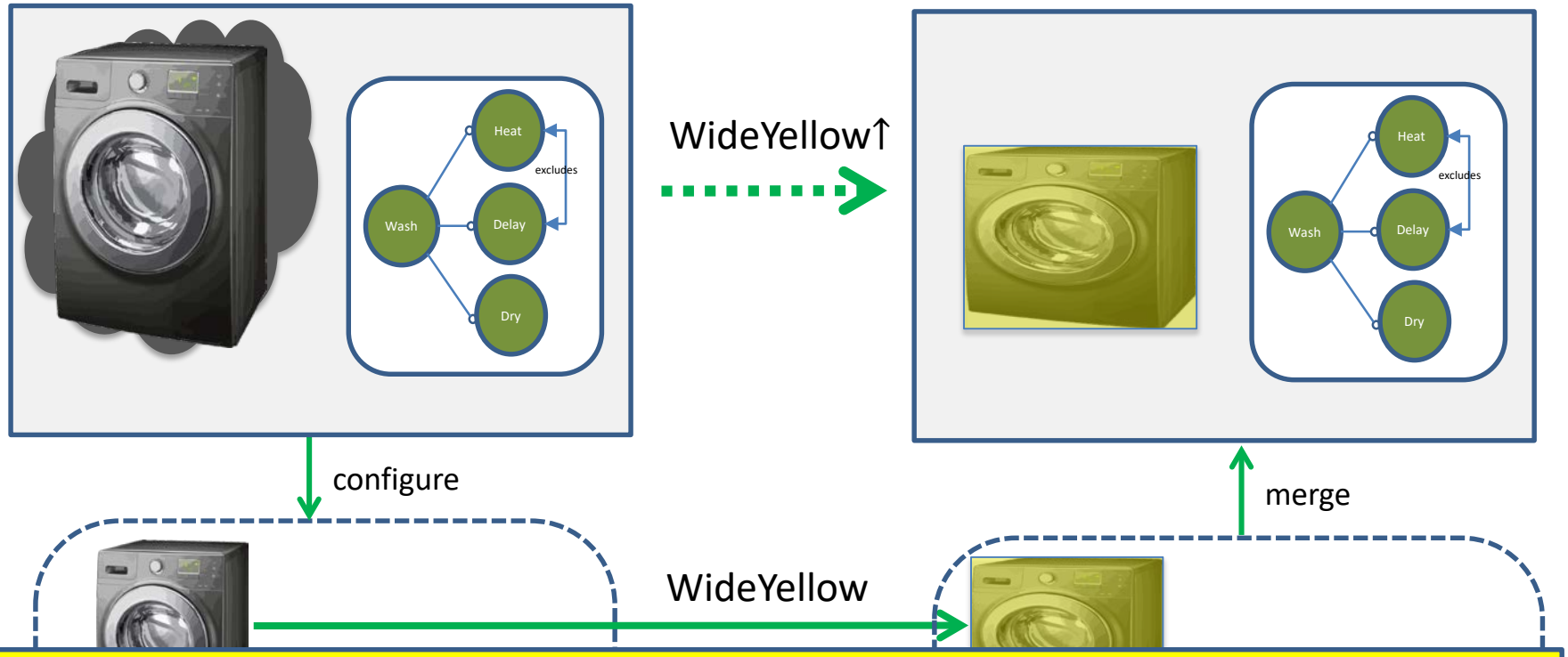
Idea 1 – Avoid Lifting Transformation

configure

WideYellow

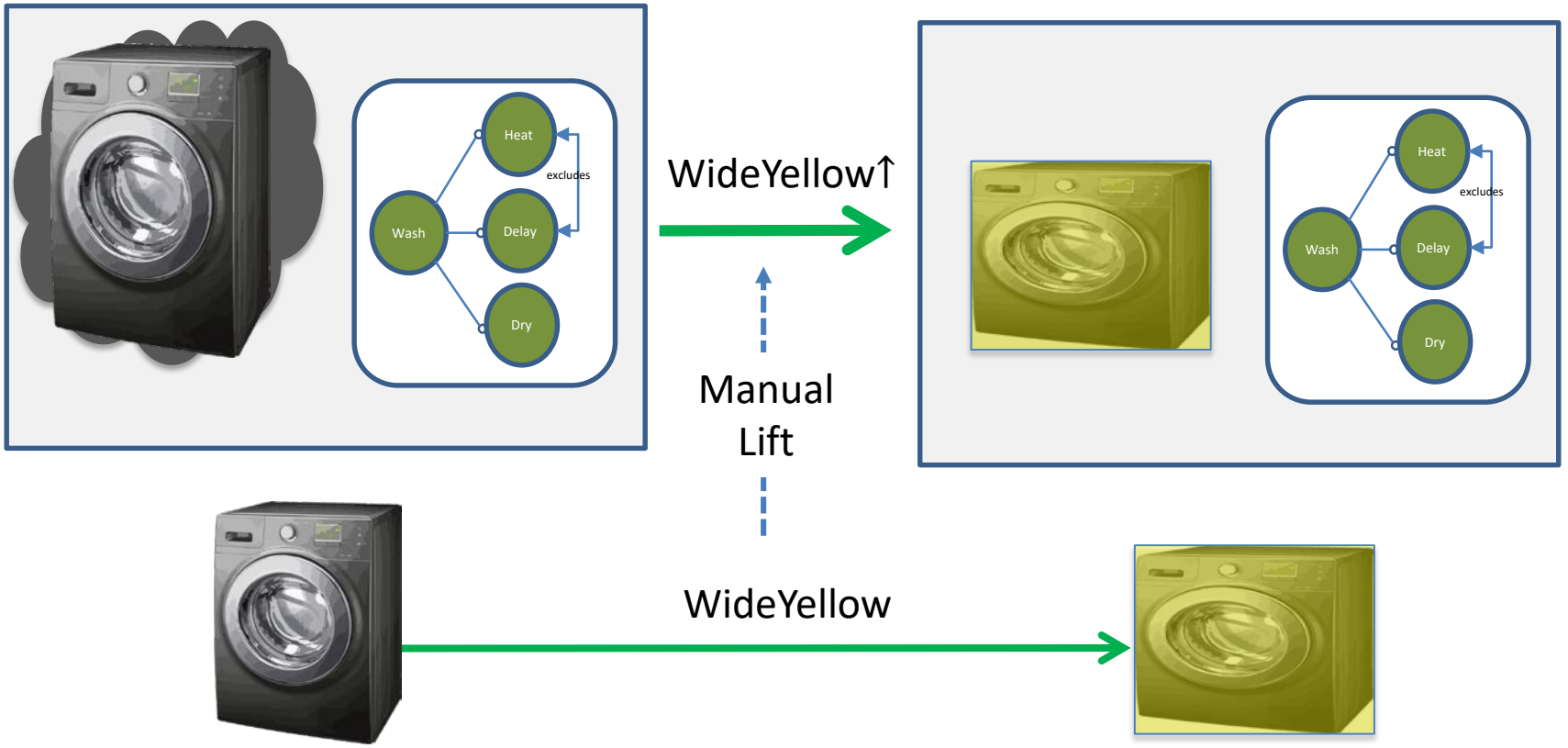
- Problems:
  - Must keep track of transformations to apply
  - Can't do analysis of transformation's effect on product line
  - No reuse!

Idea 2: Configure All Products and Merge



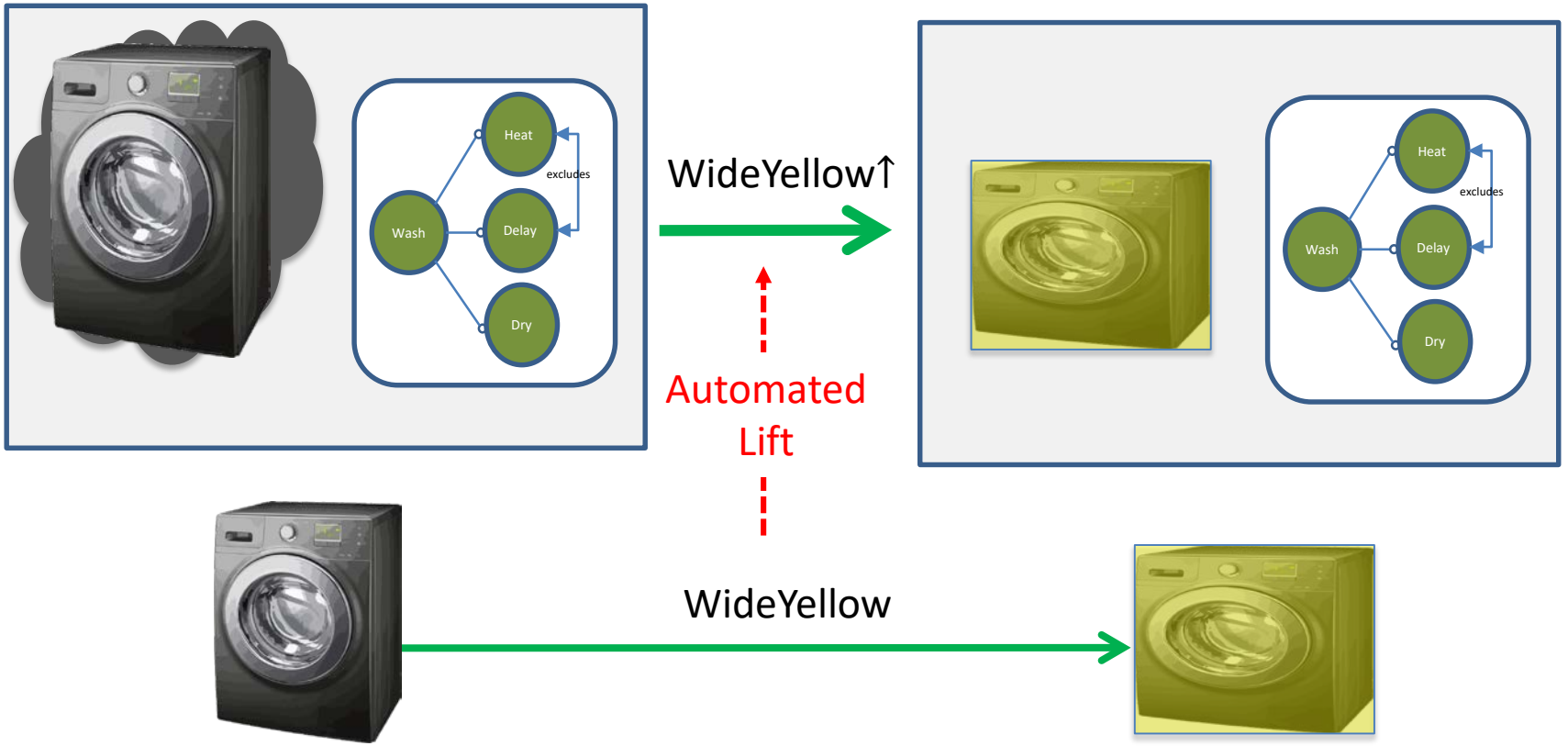
- Problems:
  - Expensive: may be many products!
  - Merge is non-trivial
  - Still not (much) reuse

Idea 3: Manually Lift by Re-developing Transformation



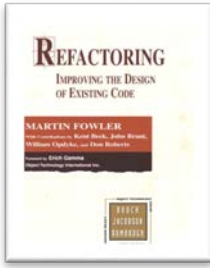
- Problems:
  - Requires extensive effort
  - Error-prone

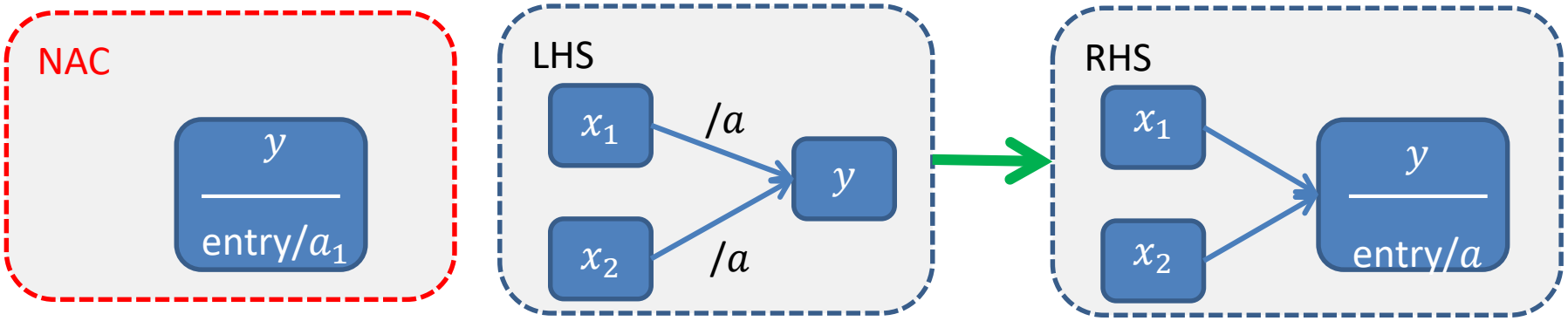
Idea: Automate the Lift



- Benefits
  - Low cost
  - Eliminates manual effort
  - Guarantees correctness

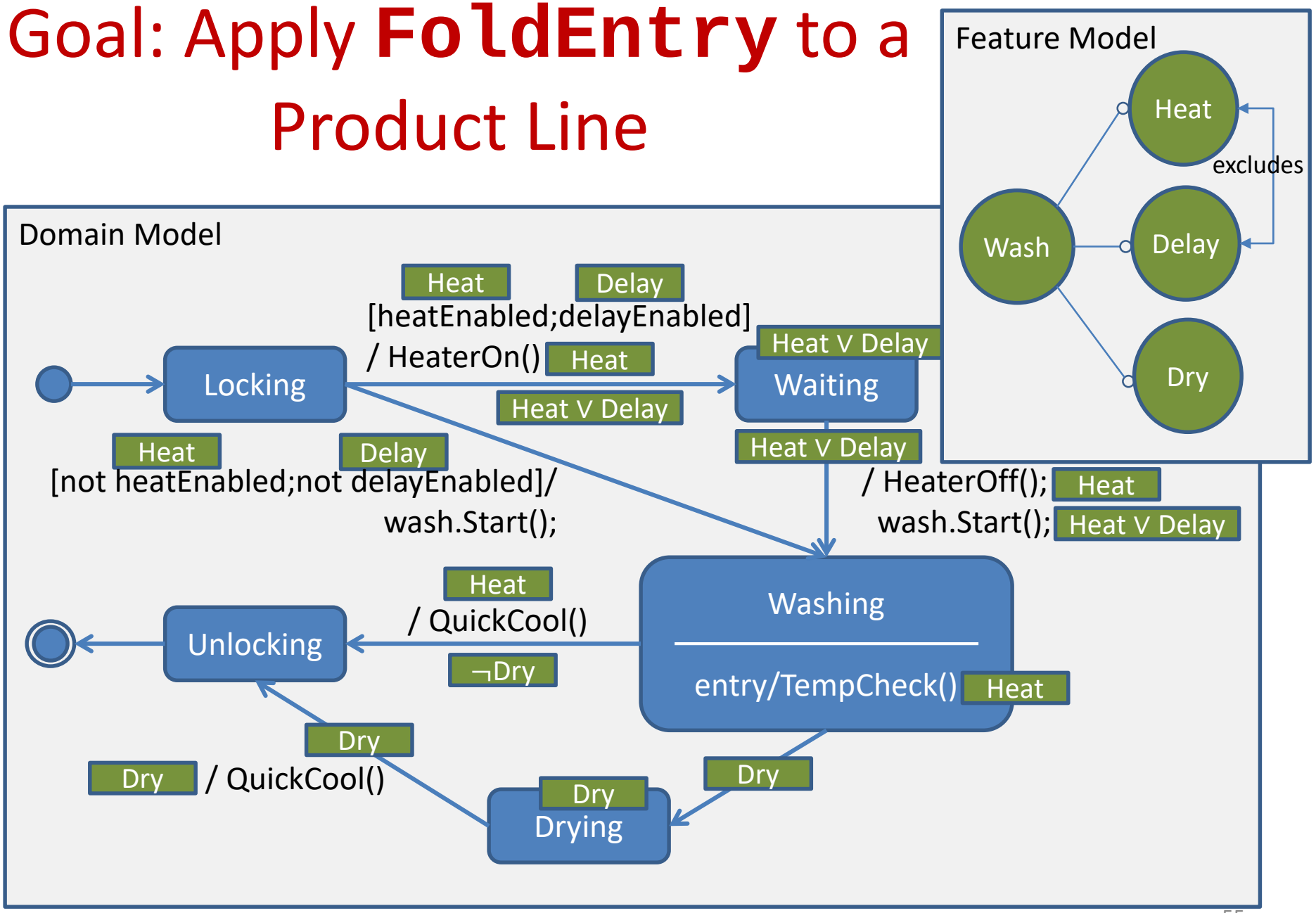
`FoldEntry` Transformation



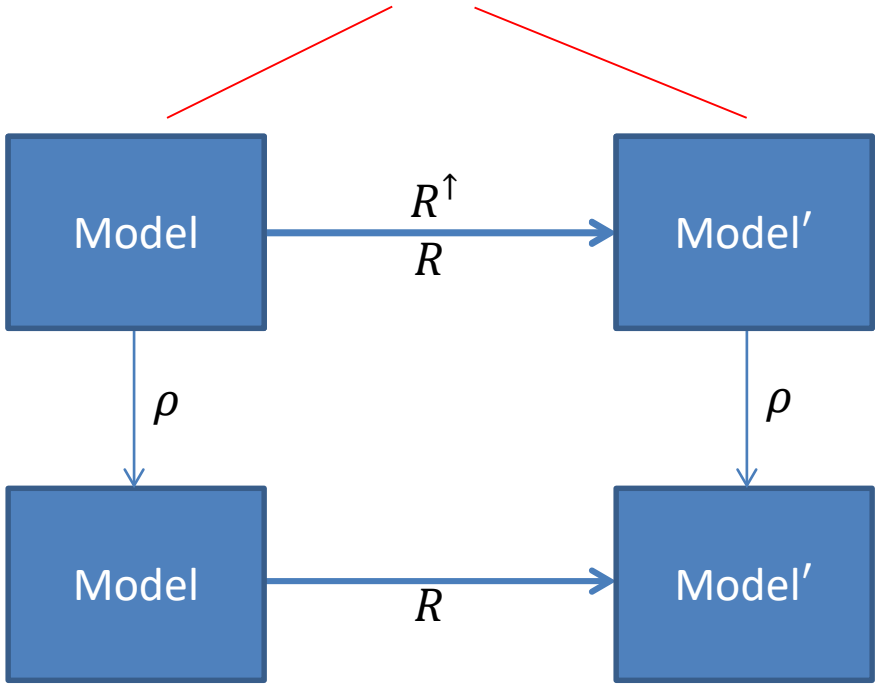


[ICSE'14]

Goal: Apply **FoldEntry** to a Product Line



Correctness Criteria



This does not prevent from...

… changing the domain model, or

… changing the feature model, or

… reducing the number of products

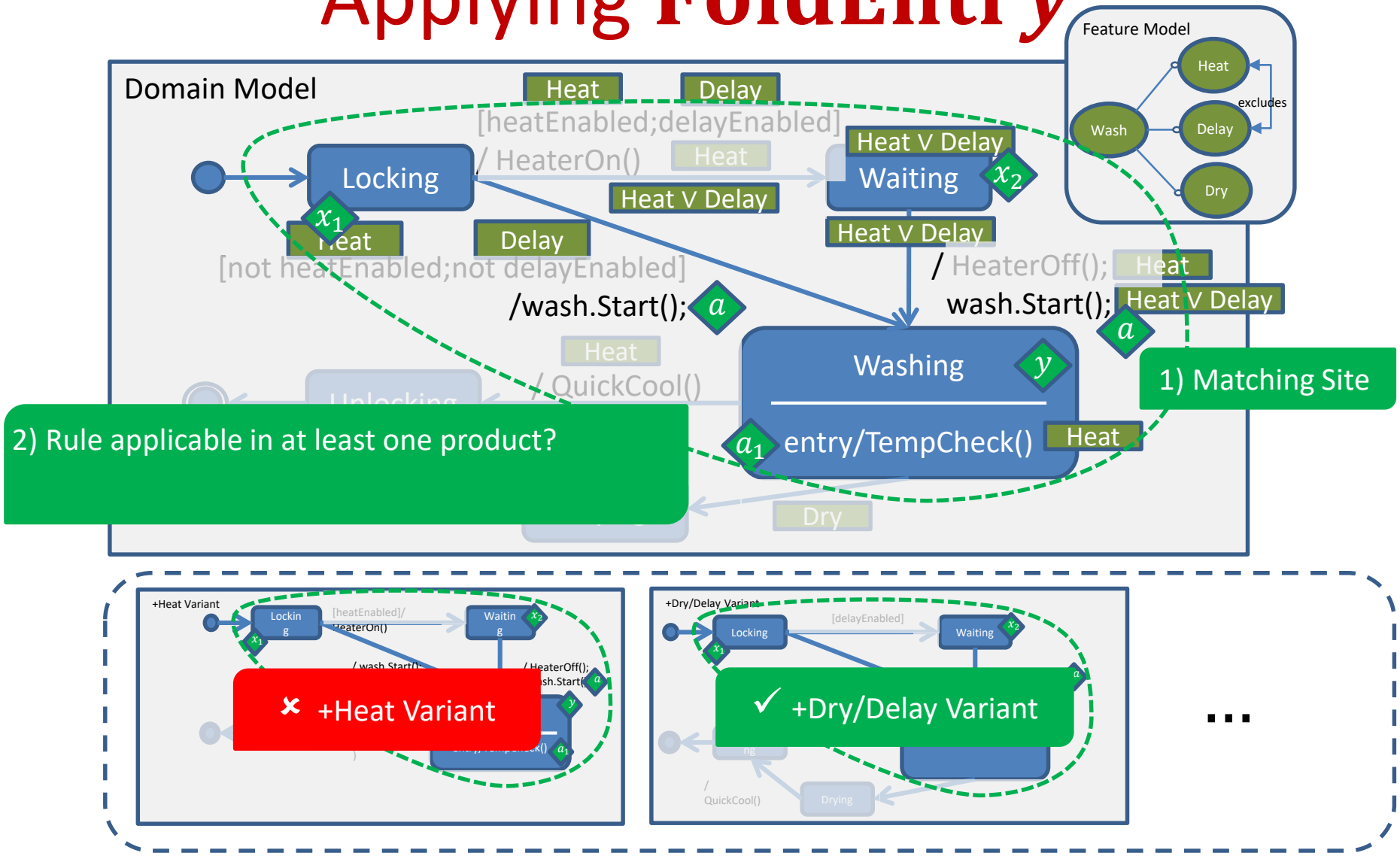
[ICSE'14]

Lifting Algorithm Sketch

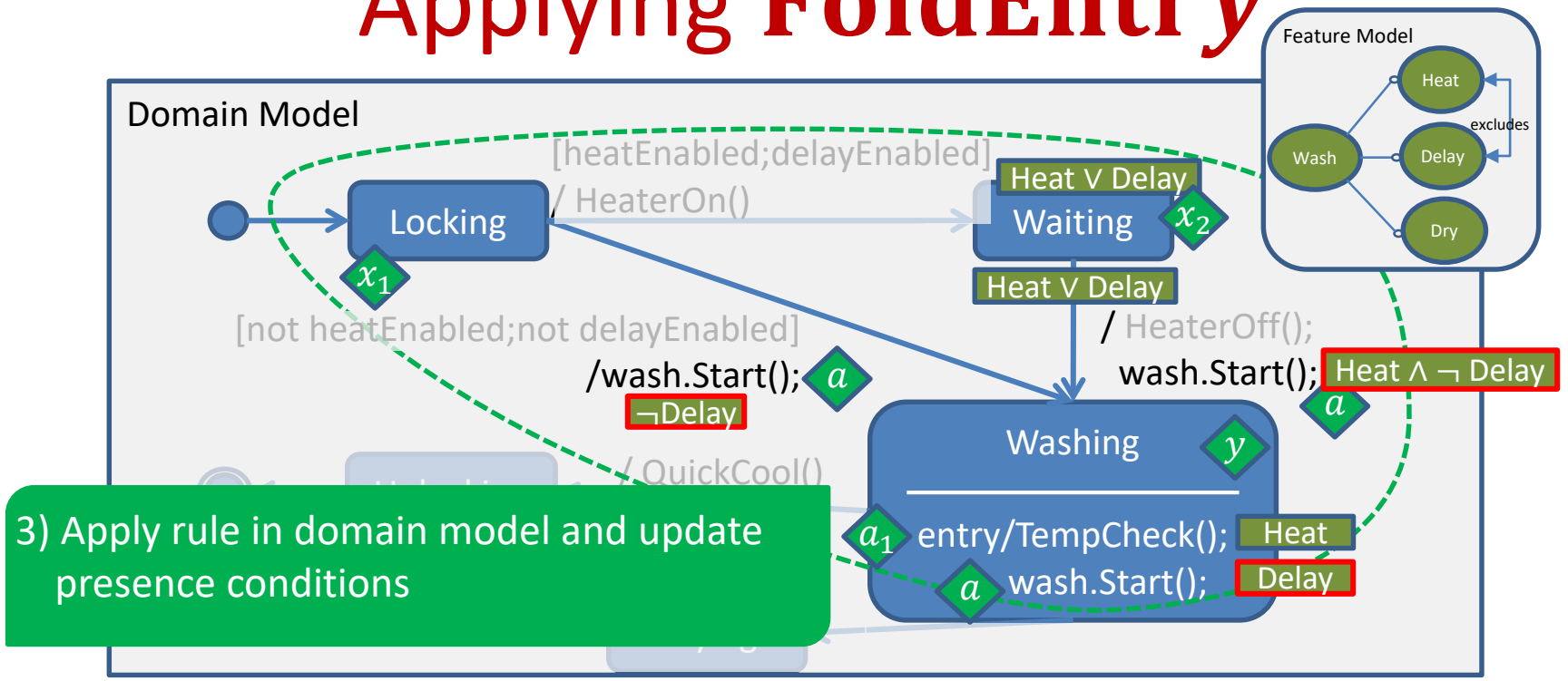
1. Find matching sites in the domain model
2. Reinterpret rule applicability condition
   - Rule must be applicable in at least one product
     - requires a SAT check
3. Reinterpret how to apply the rule
   - Modify domain model and presence conditions so rule effect only occurs in applicable products

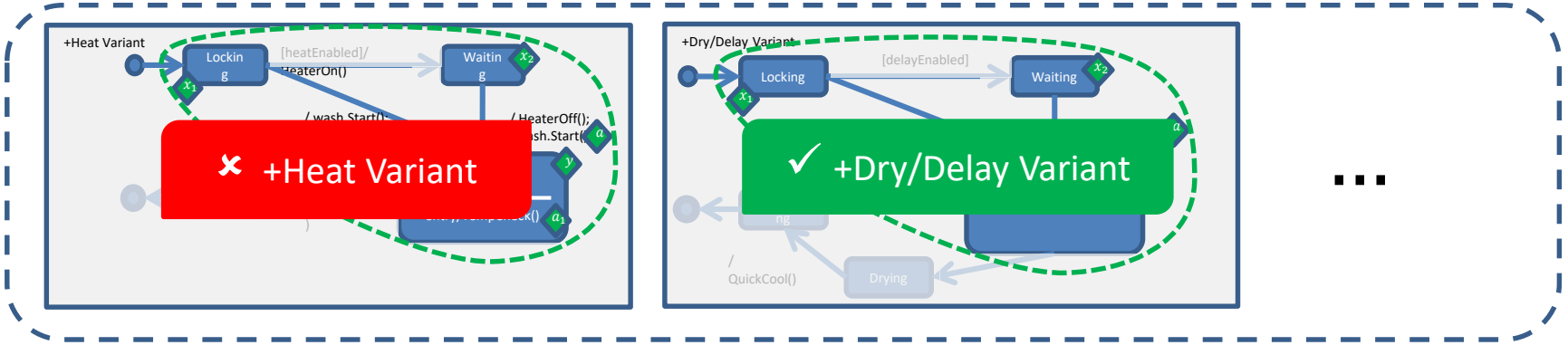
[ICSE'14]

Applying **FoldEntry**↑



Applying **FoldEntry**↑



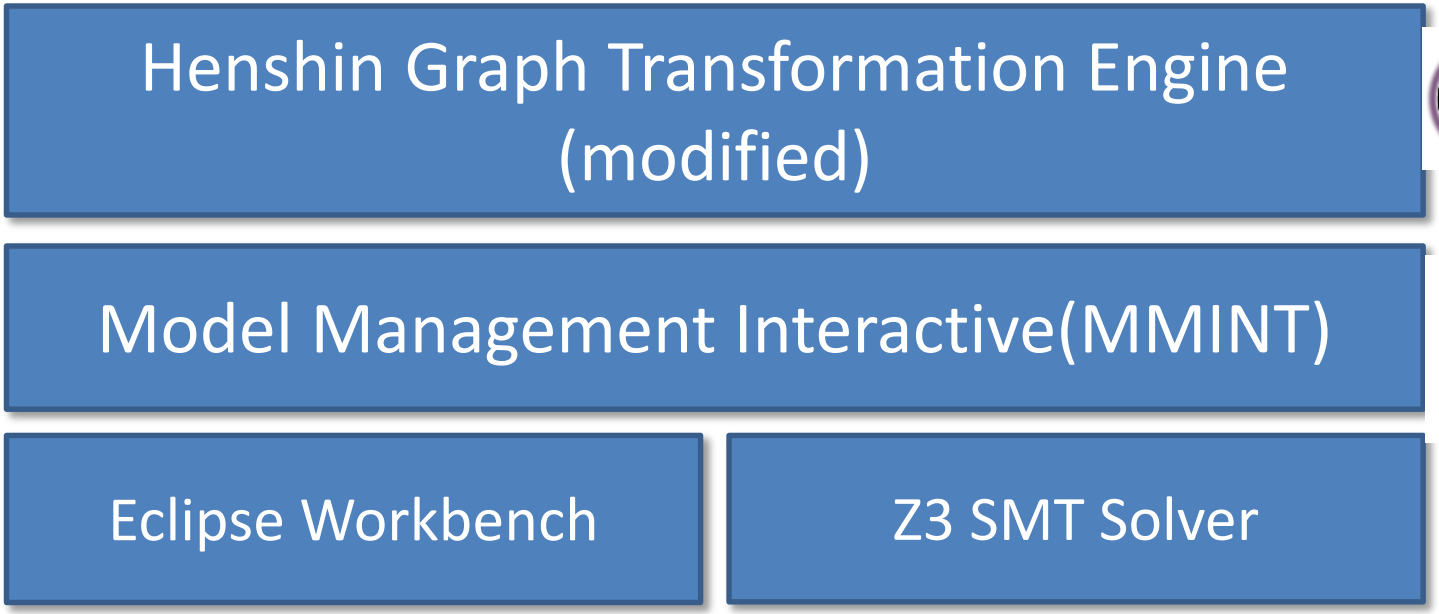


Properties of Lifting Algorithm

- Correctness
  - Lifting satisfies the correctness condition
- Termination
  - Lifting preserves rule set termination
- Confluence
  - Lifting preserves rule set confluence …
    - … up to product line equivalence

[ICSE'14]

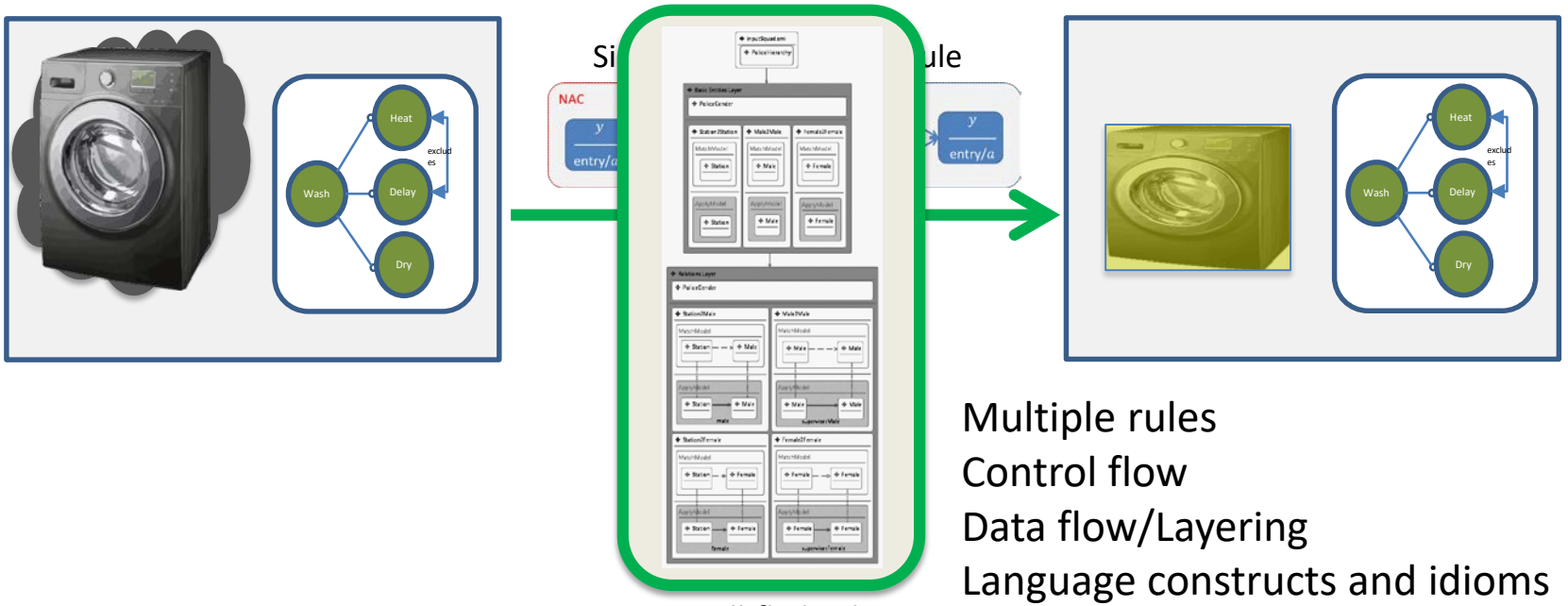
Prototype Implementation



- Modified the Henshin [Arendt et al.] graph transformation engine to …
  - … use the lifting semantics for rule execution
  - … use Z3 [Microsoft] for SAT checks via MMINT

[ICSE'14]

Lifting Complete Transformation Languages

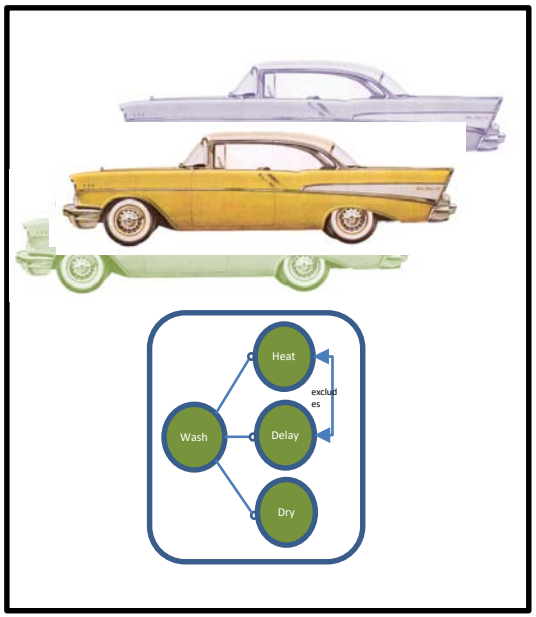


Full-fledged transformation

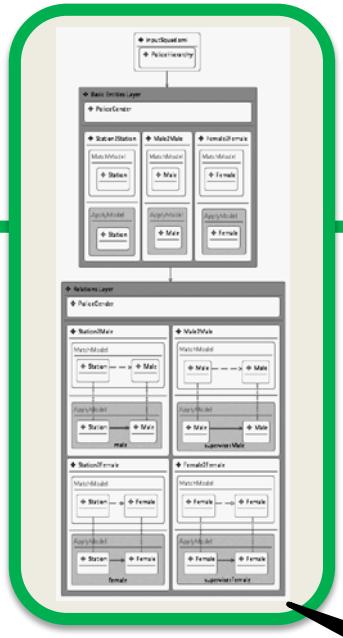
Multiple rules
Control flow
Data flow/Layering
Language constructs and idioms
Language-specific matching semantics

[ICMT'15]

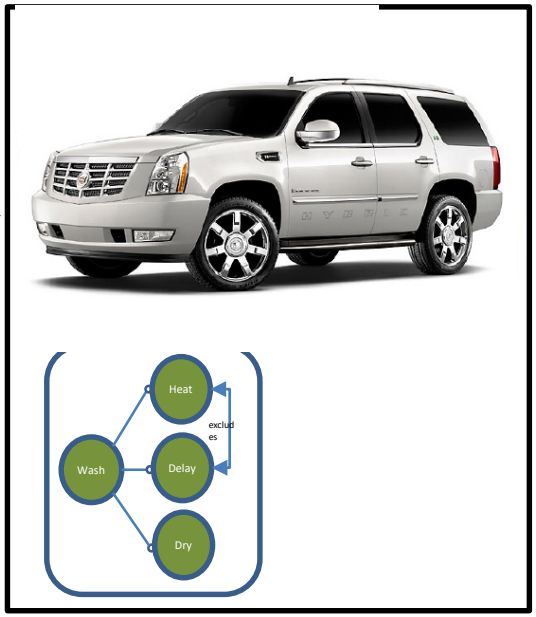
Lifting Complete Transformation Languages



Proprietary GM Metamodel



GM-to-AUTOSAR
[Selim et al., SoSyM'15]



AUTOSAR

DSLTrans
[Lucio et al., SLE'10]

[ICMT'15]

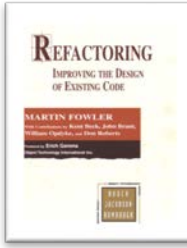
Novel Approaches: Aggregating

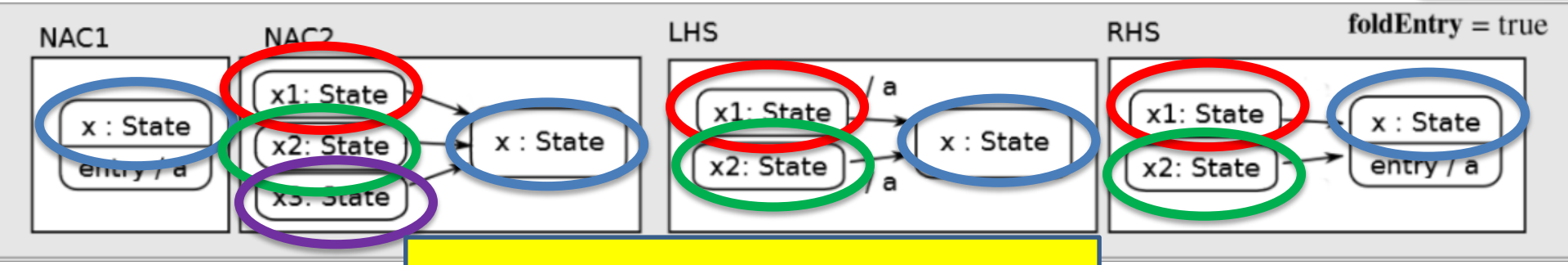
Capture and leverage variability in the transformation itself

1. Reuse transformation fragments to create transformations with variability
2. Use variability-based transformations to reuse intermediate execution artifacts

Some Similar Transformations

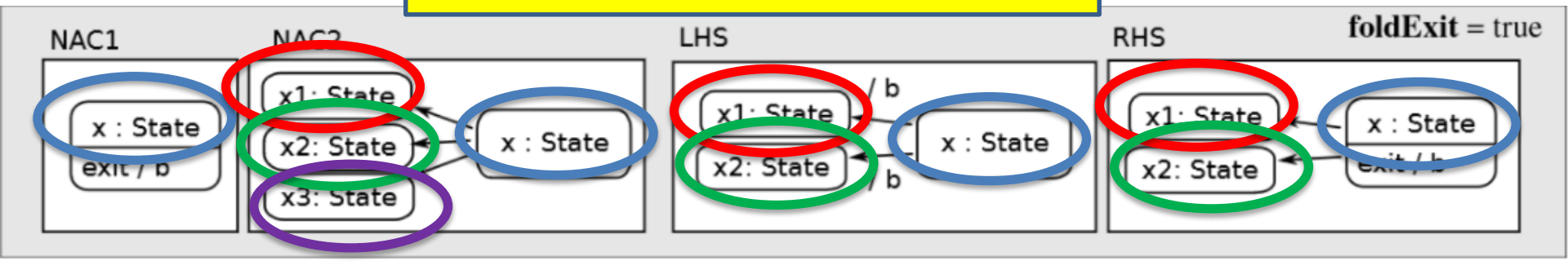
Large transformation systems often have similar but slightly different rules





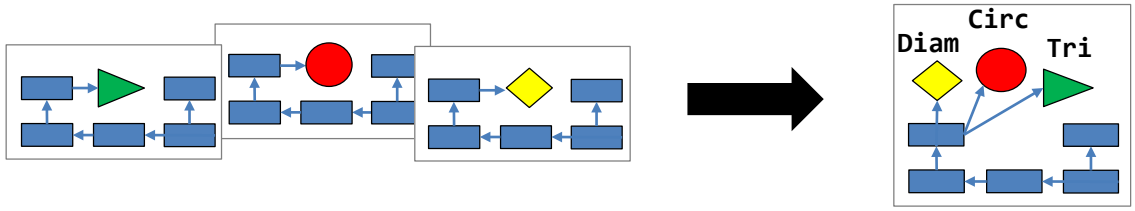
Problems:

- Hard to read / reuse / modify
- Issues with performance



[FASE'16]

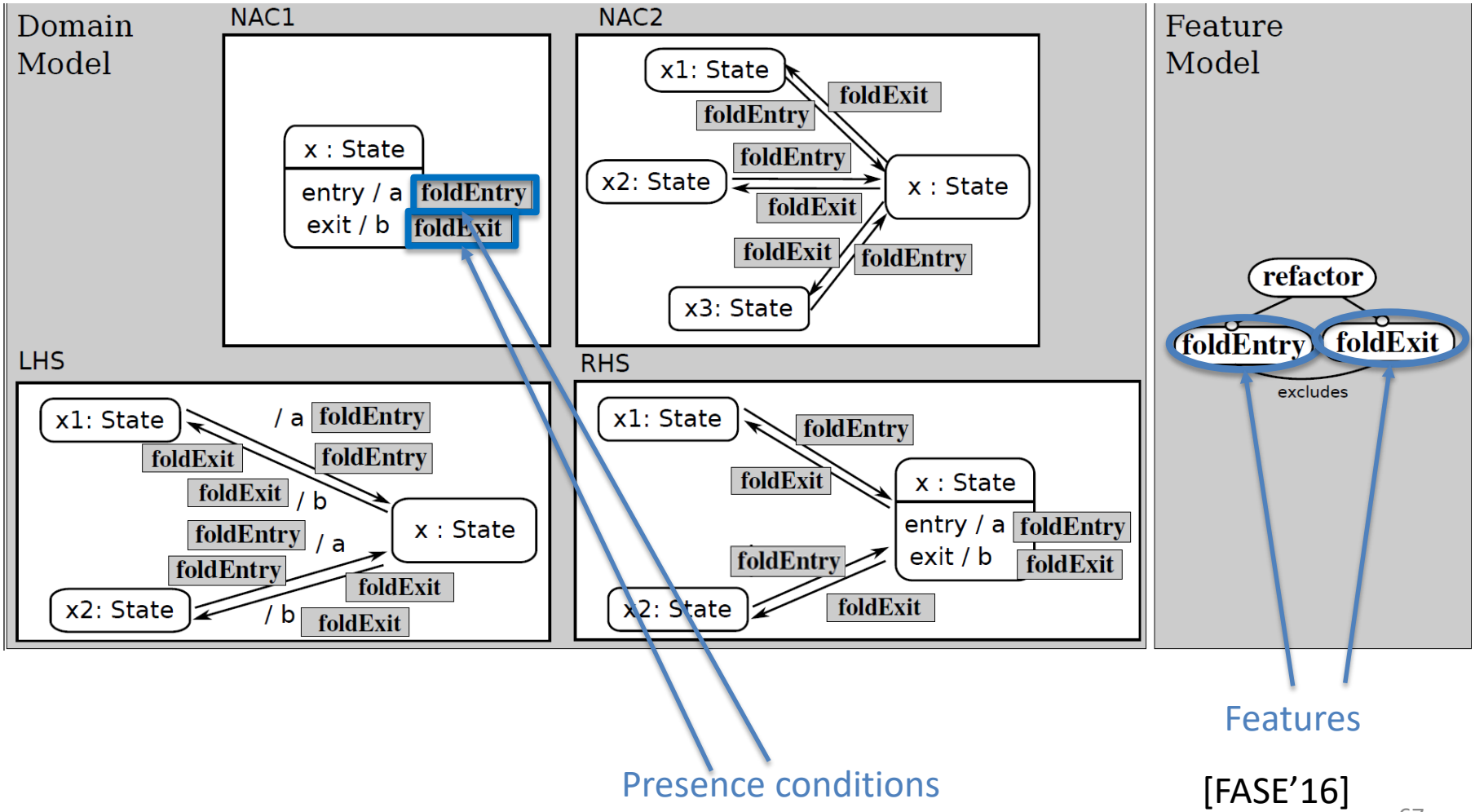
RuleMerger: From similar to variability-based rules



- Merges similar rules to produce a “150% rule”
  - Rule with variability
  - Configuration yields original rules
- Uses clone detection and clustering techniques
- Enables compact specification with improved performance

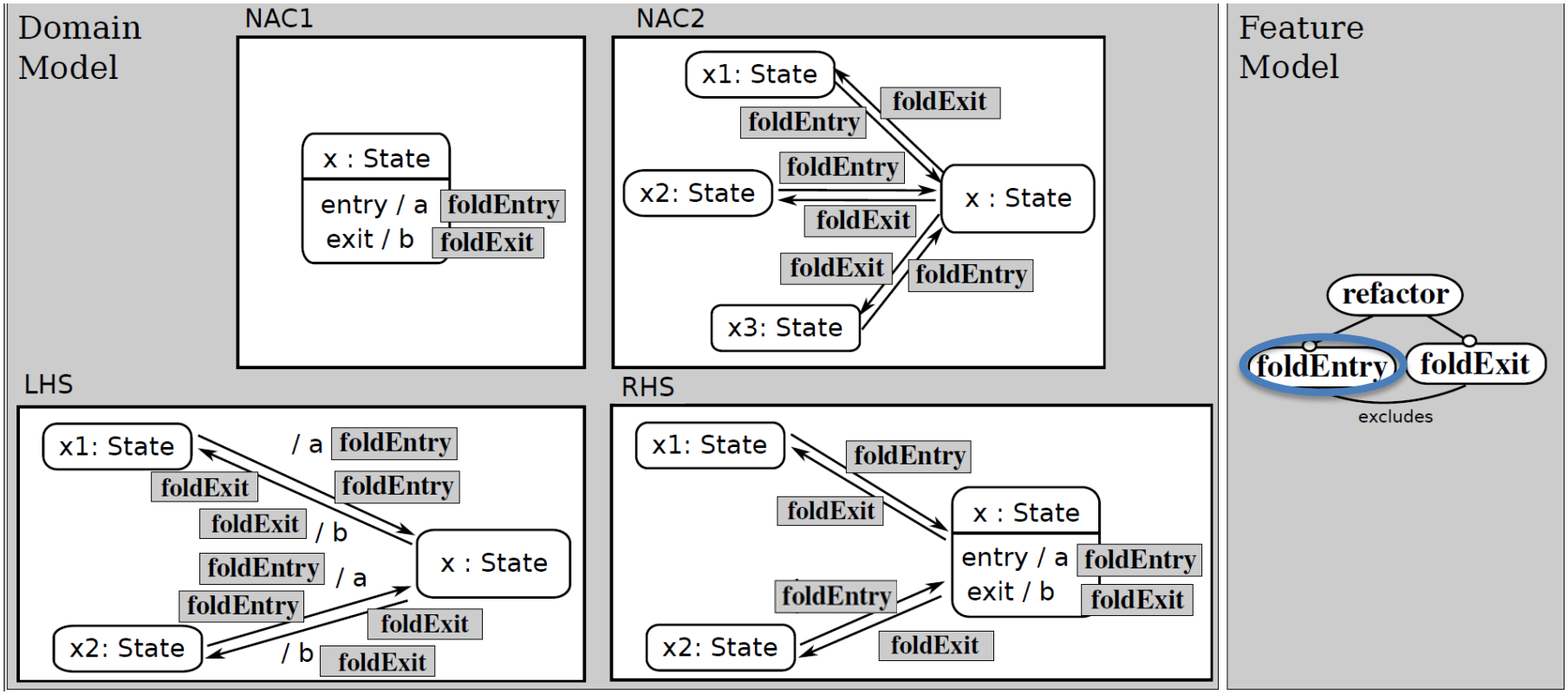
[FASE’16]

Identify commonalities, unify variabilities: **FoldLabel**

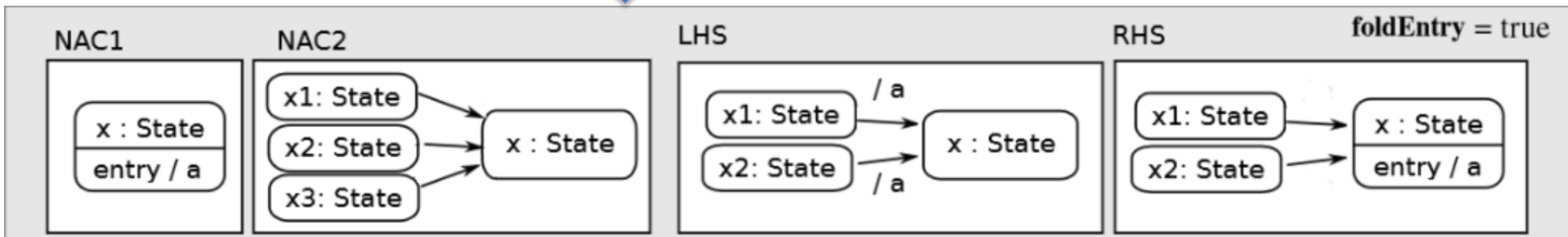


[FASE'16]

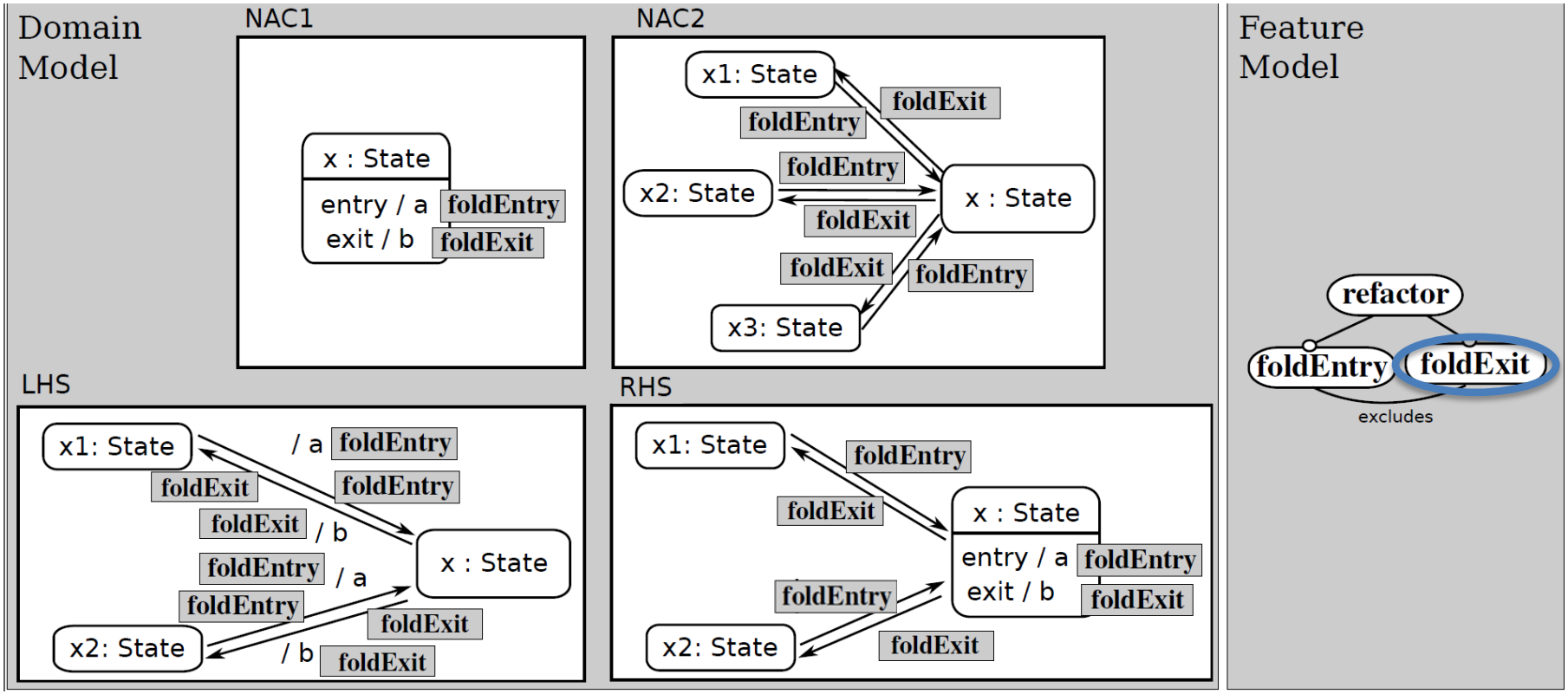
Select foldEntry to obtain **FoldEntry** rule



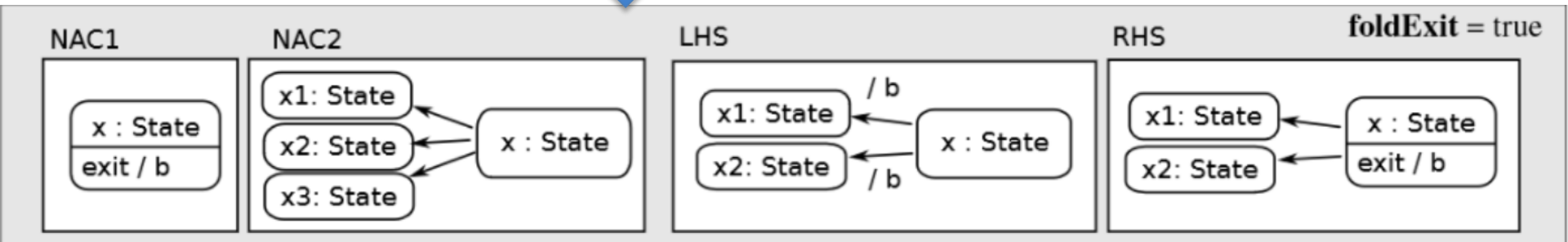
Configuring



Select foldExit to obtain **`FoldExit`** rule



Configuring



Transformations with Variability

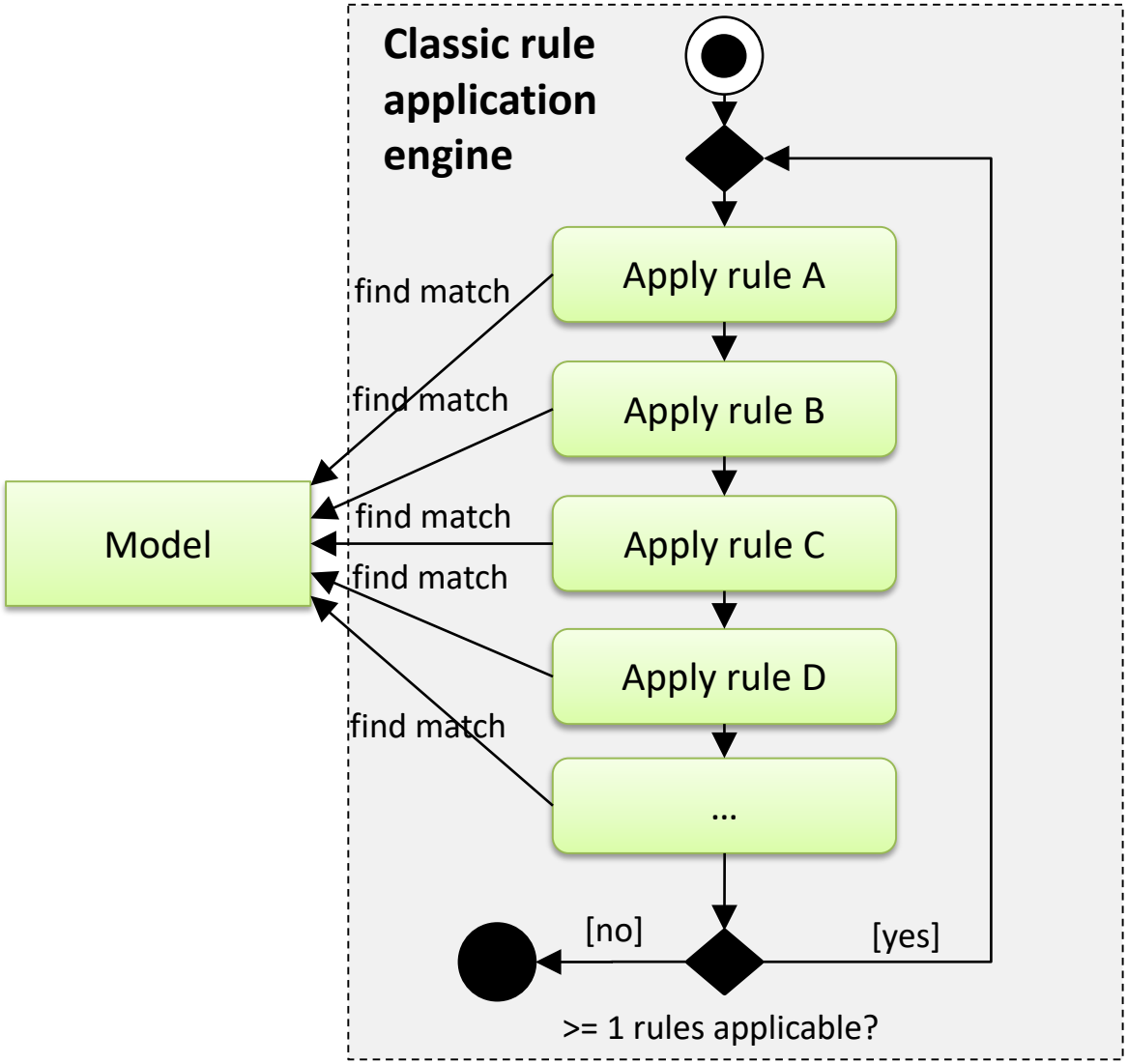
- More compact
- Easier to maintain
- Significantly better performance (see later)

Novel Approaches: Aggregating

Capture and leverage variability in the transformation itself

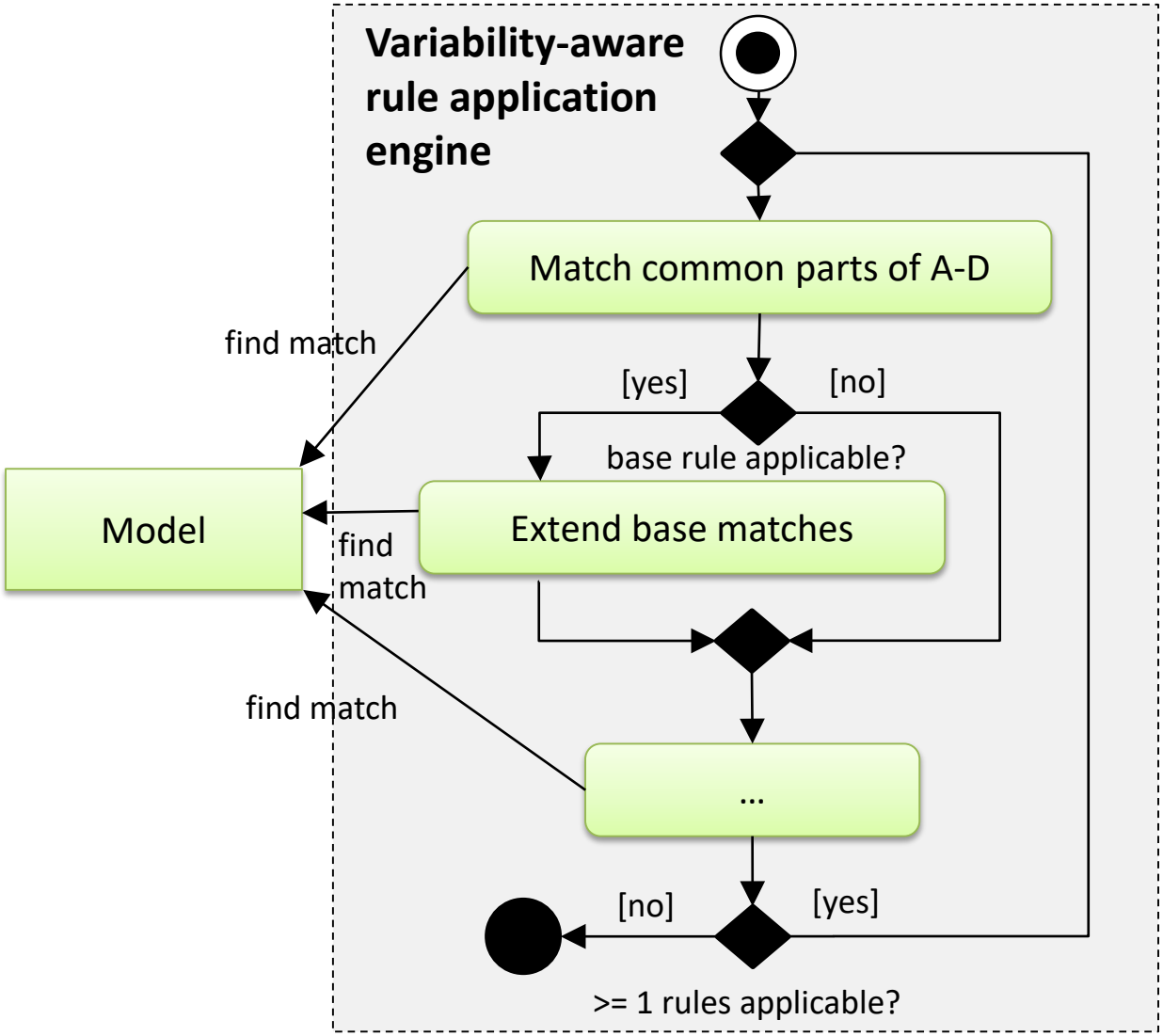
1. Reuse transformation fragments to create transformations with variability
2. Use variability-based transformations to reuse intermediate execution artifacts

Implicit Variability Is Bad for Performance



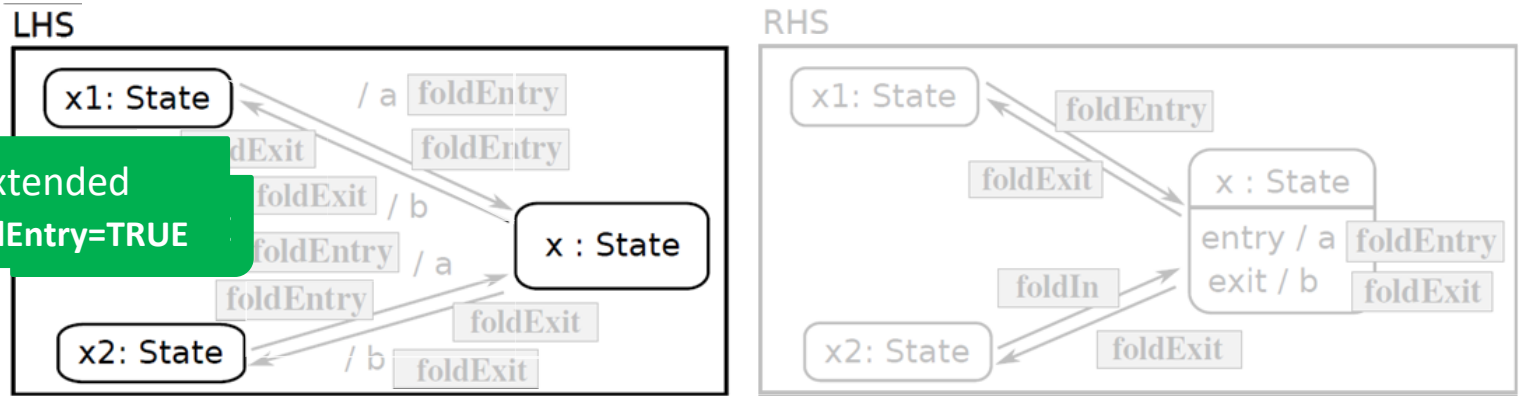
[FASE'15]

Goal: Consider Variability During Rule Application

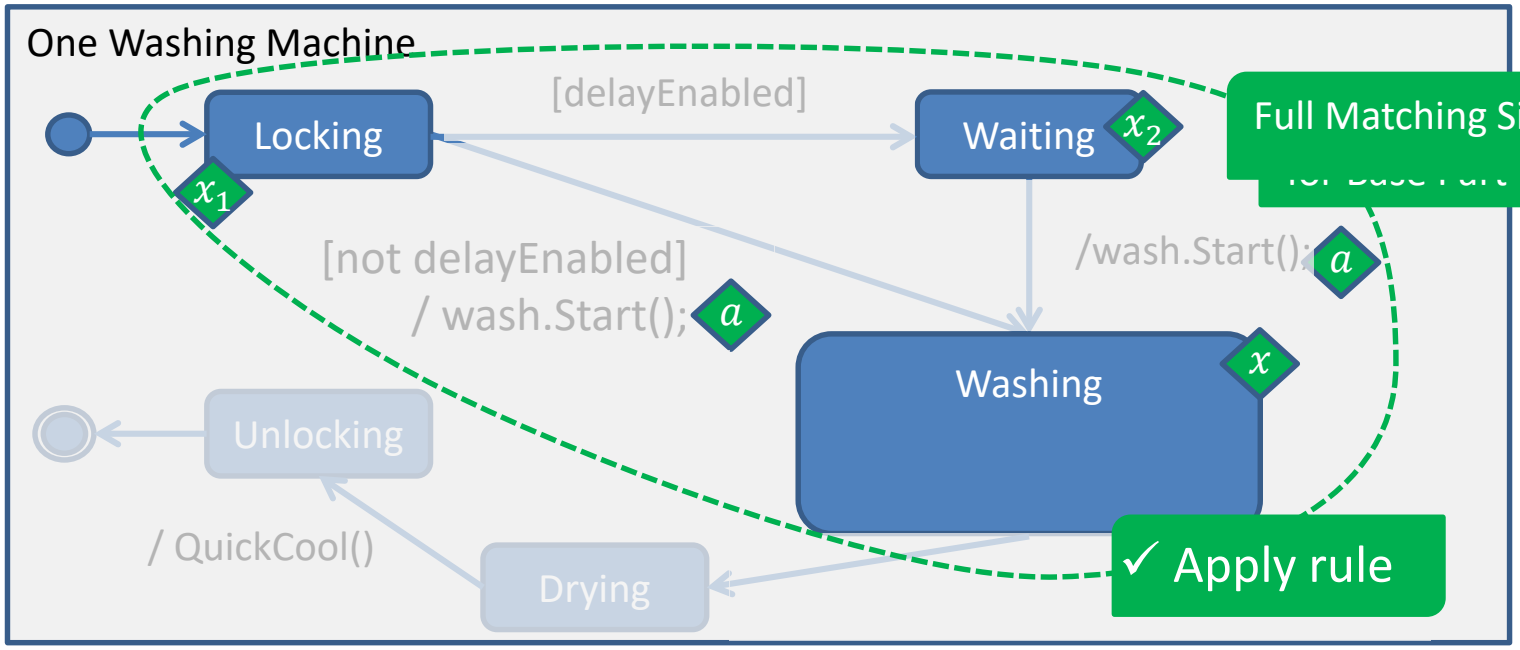


[FASE'15]

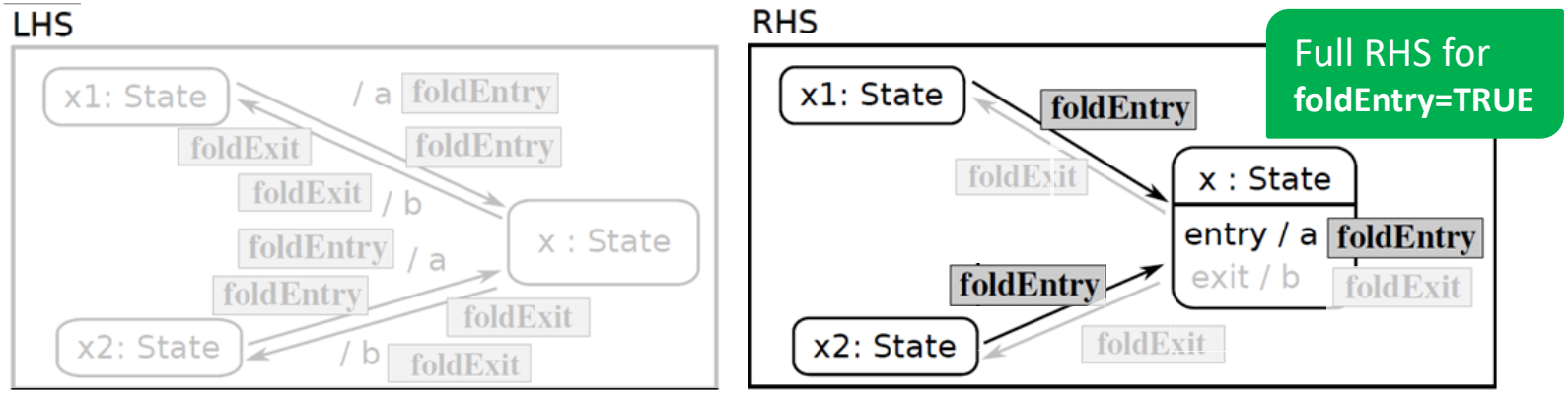
Applying **FoldLabel** – Example

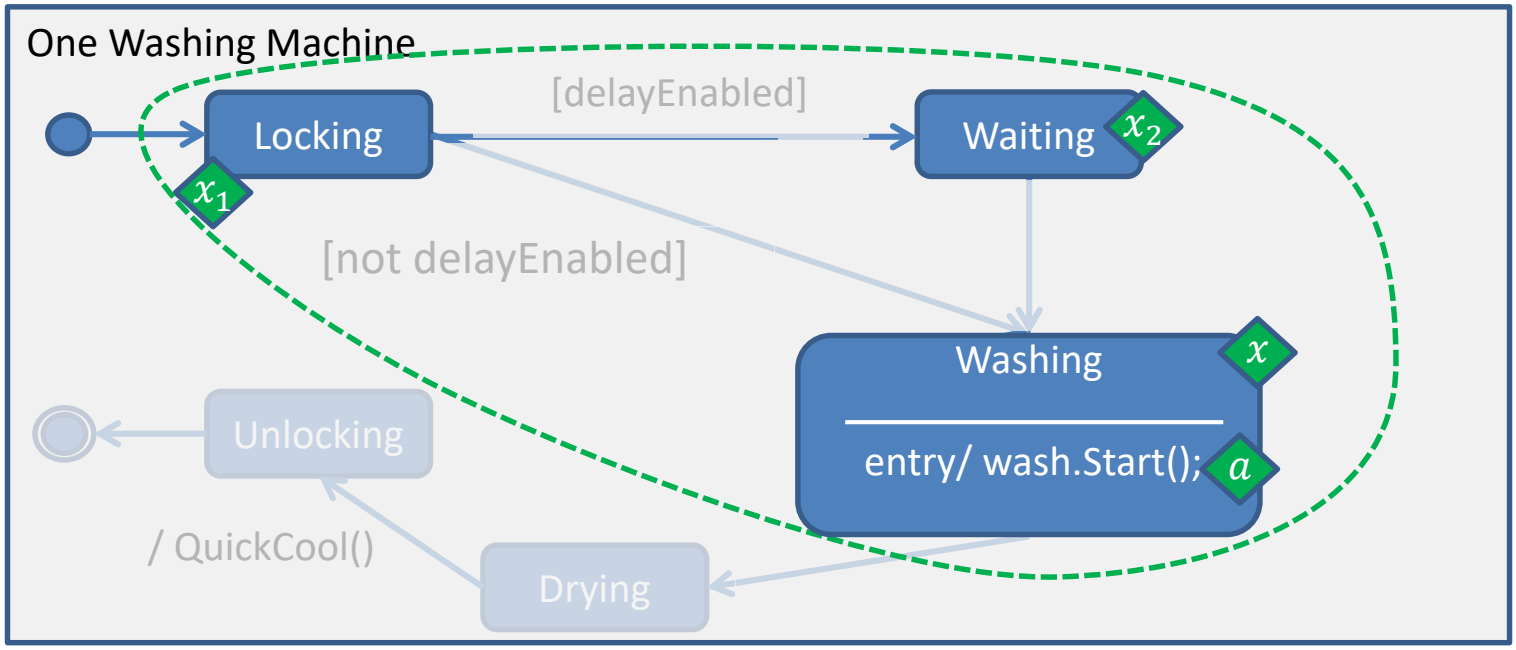


LHS is extended with **foldEntry=TRUE**



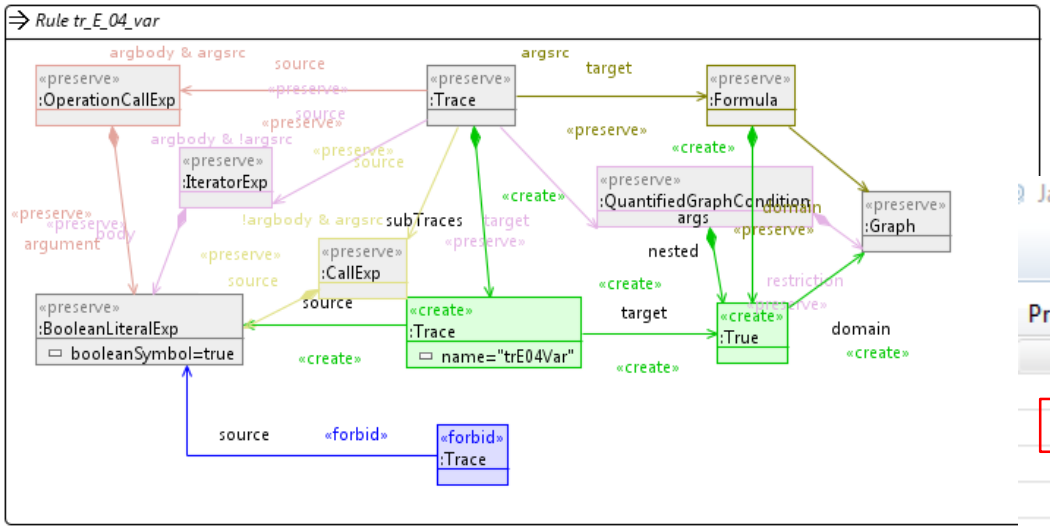
Applying **FoldLabel**– Example

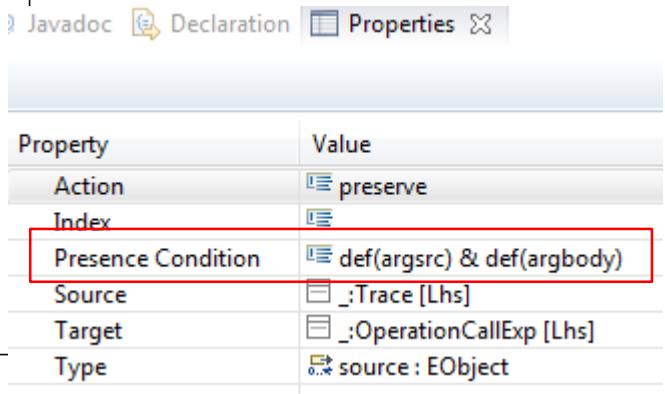




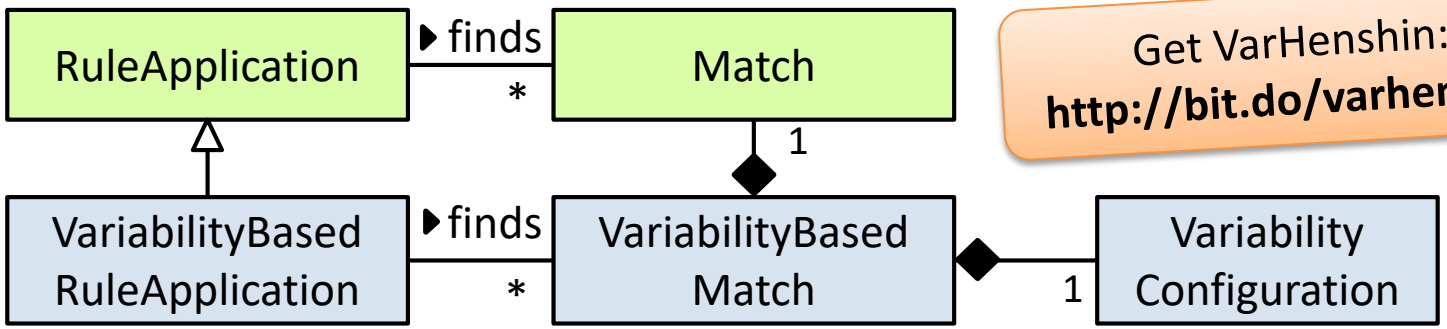
Tool Support: VarHenshin

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





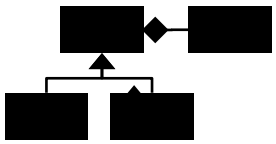
- To **apply** variability-based rules, extended the Henshin interpreter API



Get VarHenshin:
http://bit.do/varhenshin

[FASE'15]

Evaluation



Materials

- 3 rule sets from different domains
  - Edit operation recognition [Bürdek 2015]
  - Model constraint translation [Arendt 2014]
  - Transformation benchmark [Varró 2006]

Set-up

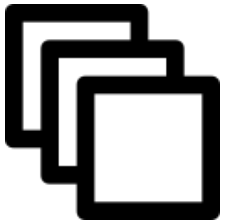
- Measured performance, scalability, and compactness
  - Input parameters optimized for performance

[FASE'15, FASE'16]

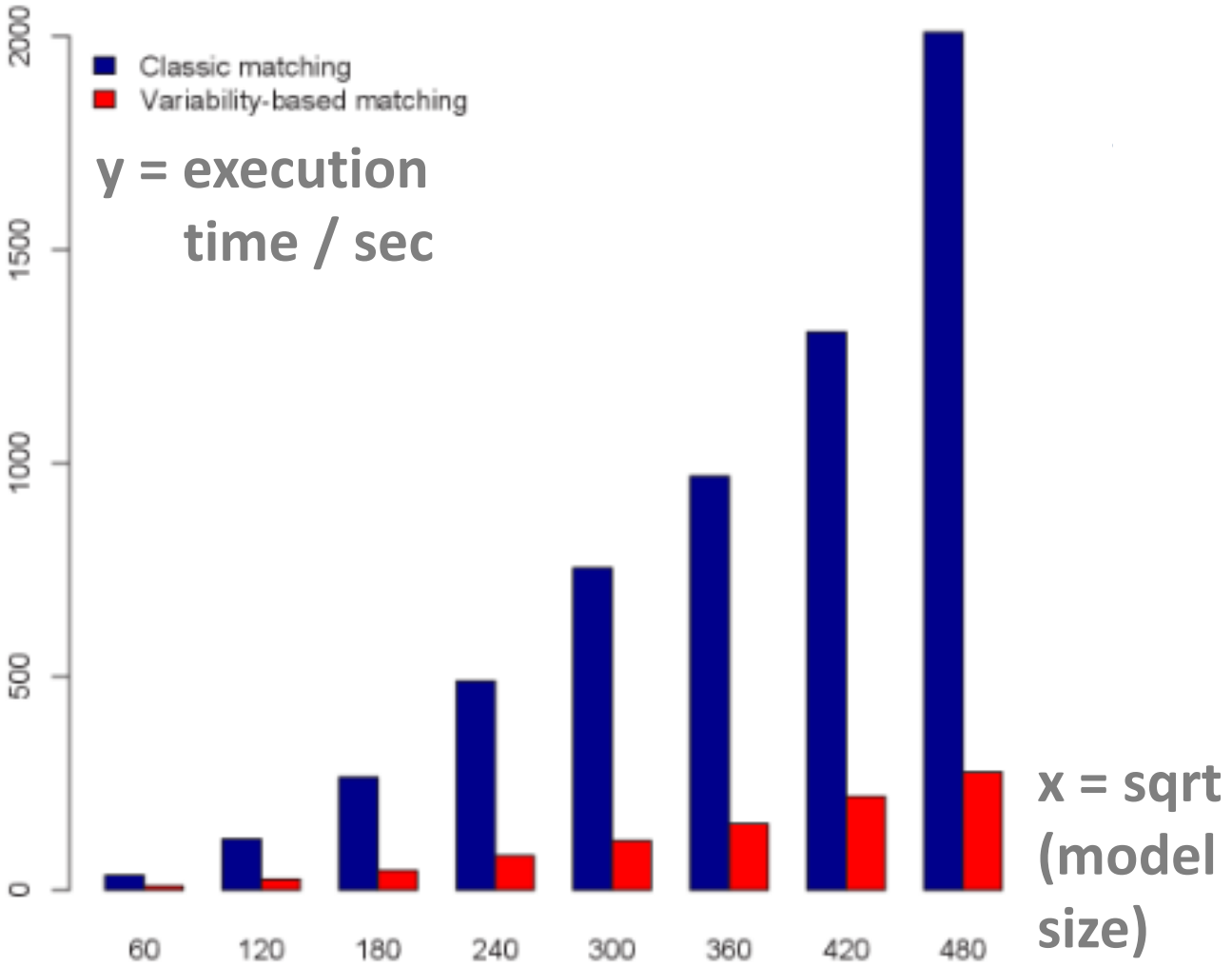
Merged Rules Improve Performance!!!

Performance

Scalability



Compactness



[FASE'15, FASE'16]

Novel Approaches: Summary

Lifting transformations

From individual products to product line

Aggregating

Reuse transformation fragments

Reuse intermediate execution artifacts

- Other approaches
  - Transformation composition (by chaining and weaving)
  - Transformation reuse across families of related domain-specific languages [DeLara et. al., SOSYM'15]

- Motivation
  - Models and Transformations
  - Why Reuse
- Transformation Reuse
  - PL adaptations: subtyping and mapping
  - MDE-specific approaches: lifting and aggregating
- Future perspectives
- Coffee

Some Future Perspectives

- Transformation Intent
- Applying MDE techniques to programs

Transformation Intent

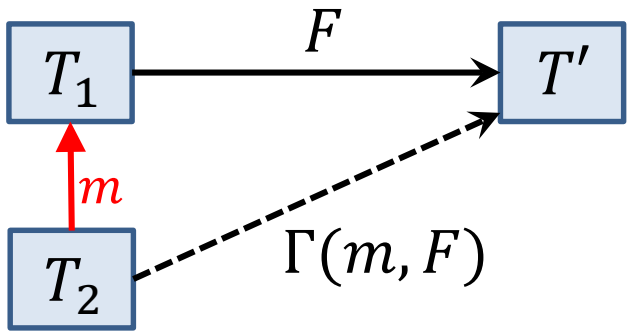
Recall:

> Transformations are aimed to accomplish a one-step task with specific intent

So reuse objective is to preserve this intent

- Subtyping: same intent of trans on subtype inputs
- Mapping: same intent of trans for collections
- Lifting: same intent of trans for product lines
- Aggregation: same intent of fragment in each transformation

General Intent Preservation



F: a transformation
m: type mapping
Ω: set of transformations of interest

Transformation reuse mechanism Γ …

… constructs new trans $\Gamma(m, F)$ given type mapping m: $T_2 \Rrightarrow T_1$

Is sound for set of transformations Ω iff

$\forall F \in \Omega \cdot \Gamma(m, F)$ has same intent of F

Is complete for set of transformations Ω iff

$\forall F, F' \in \Omega \cdot (F'$ has same intent as $F) \Rightarrow \exists m \cdot F' = \Gamma(m, F)$

Current research: how to check/guarantee Γ is sound and/or complete for Ω?

[AMT'15, ICMT'16]

Adapting MDE Techniques to Programs (lifting)

A terrific body of work by Christen Kaestner on reinterpreting various code analyses – one at a time – on 150% code models (with #ifdefs)

Problem:

Given an analysis method on programs, reinterpret (lift) it on 150% representations of programs, together with proofs of correctness (that the method gives correct analysis on each variant)

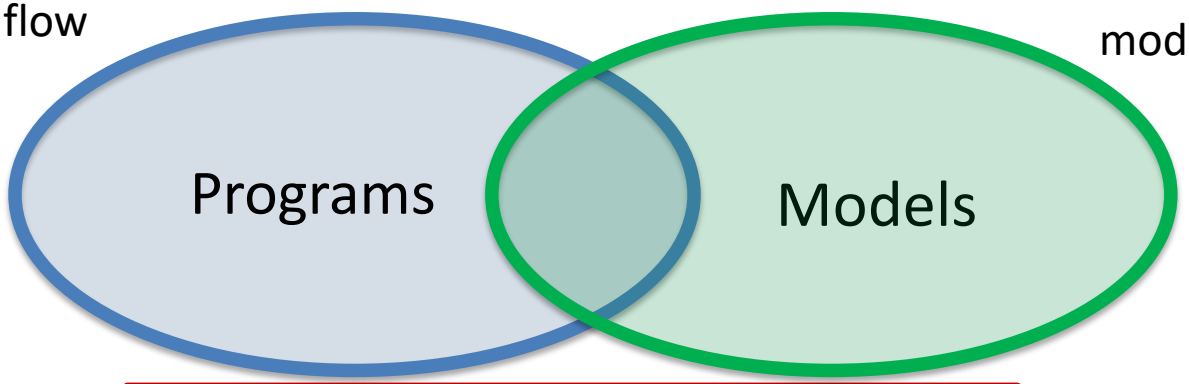
Current work:

Trying to lift analysis behind UFO [CAV'12] - a combination of over- and under-approximation

A parting thought

- Executable
- Present in abundance “in real world”
- Complex data structures and control flow

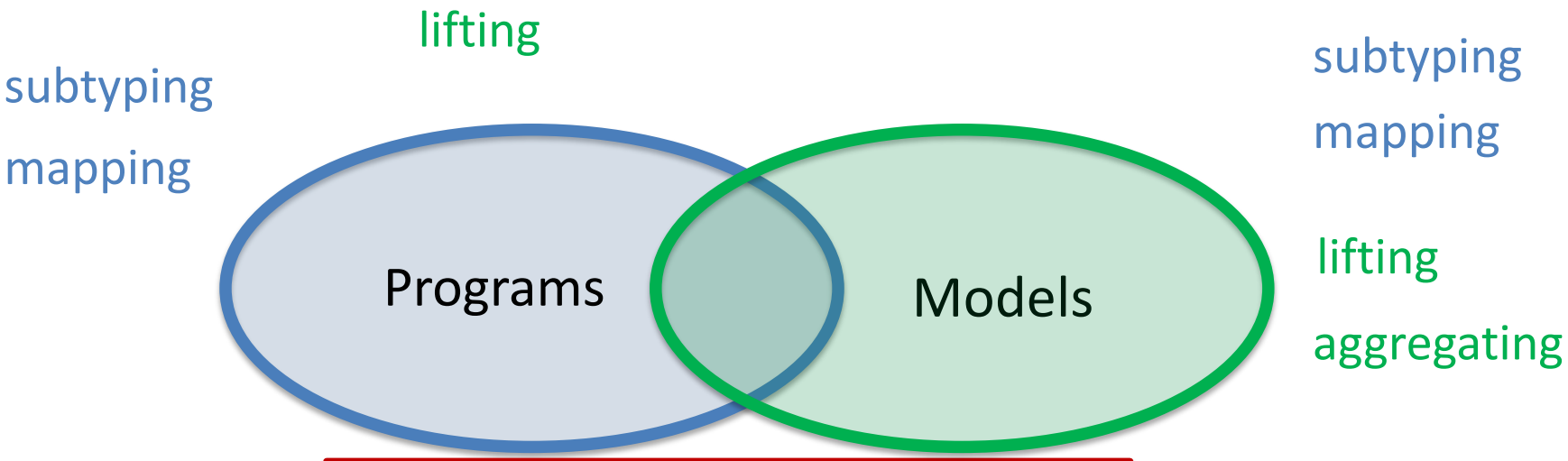
- Higher level of abstraction (“for a purpose”)
- Presence of meta-models



Goals:
- Quality
- Speed of development
- Understandability

(Formal) Methods:
- Specification
- Analysis

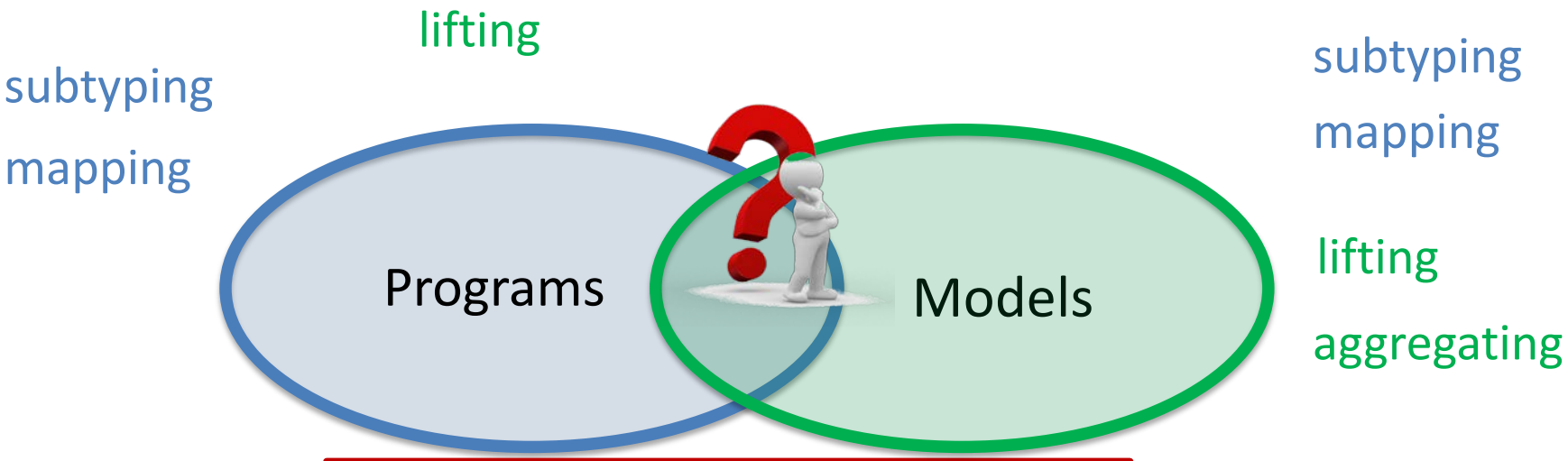
Perspectives on Model Transformation Reuse



Goals:
- Quality
- Speed of development
- Understandability

(Formal) Methods:
- Specification
- Analysis

A parting thought: Synergy



Goals:
- Quality
- Speed of development
- Understandability

(Formal) Methods:
- Specification
- Analysis

Acknowledgements

Many thanks for colleagues in Toronto…





…and elsewhere in the world

- Gehan Selim




References

[ICMT'15] M. Famelis, L. Lúcio, G. Selim, A. Di Sandro, R. Salay, M. Chechik, J. R Cordy, J. Dingel, H. Vangheluwe, and Ramesh S. Migrating Automotive Product Lines: a Case Study, ICMT'15: 82-97.

[MODELS15Tool] A. Di Sandro, M. Famelis, R. Salay, S. Kokaly, M. Chechik. MMINT: A Graphical Tool for Interactive Model Management: MODELS 2015 Demos.

[ICSE14] Rick Salay, Michalis Famelis, Julia Rubin, Alessio Di Sandro, Marsha Chechik: Lifting model transformations to product lines. ICSE 2014: 117-128

[FASE15] Daniel Strüber, Julia Rubin, Marsha Chechik, Gabriele Taentzer: A Variability-Based Approach to Reusable and Efficient Model Transformations. FASE 2015: 283-298

[FASE16] Daniel Strüber, Julia Rubin, Thorsten Arendt, Marsha Chechik, Gabriele Taentzer, Jennifer Ploger: RuleMerger: Automatic Construction of Variability-Based Model Transformation Rules. FASE 2016: 122-140.

[ICMT'16] Rick Salay, Steffen Zschaler, Marsha Chechik: Correct Reuse of Transformations is Hard to Guarantee. ICMT'16. To appear.

[AMT'15] Rick Salay, Steffen Zschaler, Marsha Chechik: Transformation Reuse: What is the Intent? AMT@MoDELS 2015: 7-15

[MODELS15] Rick Salay, Sahar Kokaly, Alessio Di Sandro, Marsha Chechik: Enriching Megamodel Management with Collection-Based Operators. MODELS 2015: 236-245

[CAV12] Aws Albarghouthi, Yi Li, Arie Gurfinkel, Marsha Chechik: UFO: A Framework for Abstraction- and Interpolation-Based Software Verification. CAV 2012: 672-678