

# Automated Generation of Domain-Specific Graph Models The CORE-DISC Challenge

Dániel Varró

McGill University, Canada

Dept. of Measurement and Information Systems, BME

MTA-BME Lendület Cyber-Physical Systems Research Group

IncQuery Labs Ltd.

# Outline and Main Contributors

Graphs in Software Tools for Safety-Critical Systems

The Need for Automated Model Generators

Automated Generation of Consistent Graph Models

Automated Synthesis of Diverse Graph Models

## Graph Generator (BME)

- **O. Semeráth**
- **K. Marussy**
- G. Szárnyas
- G. Bergmann
- R. Farkas

## Graph Generator (McGill)

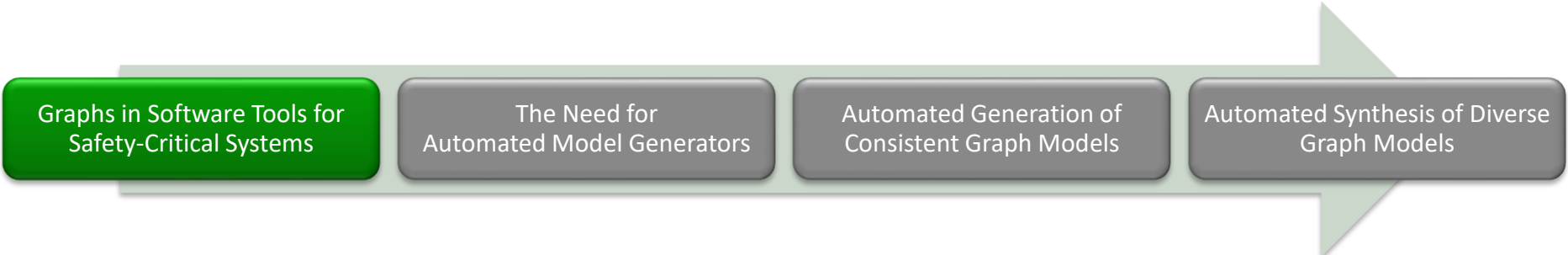
- A. Babikian
- S. Pilarski
- B. Chen
- L. Li
- A. Li
- M. Ding
- V. Gidla

## Lendület

- G. Bergmann
- A. Vörös
- M. Búr
- Cs. Debreceni
- R. Farkas
- B. Graics
- Á. Hajdú
- D. Honfi
- V. Molnár
- A. Nagy
- O. Semeráth
- G. Szárnyas

## VIATRA

- I. Ráth
- Á. Horváth
- Á. Hegedüs
- Z. Ujhelyi
- G. Bergmann
- T. Szabó
- P. Lunk
- D. Segesdi
- I. Dávid
- I. Papp
- A. Nagy
- B. Grill
- D. Harmath



Graphs in Software Tools for  
Safety-Critical Systems

The Need for  
Automated Model Generators

Automated Generation of  
Consistent Graph Models

Automated Synthesis of Diverse  
Graph Models

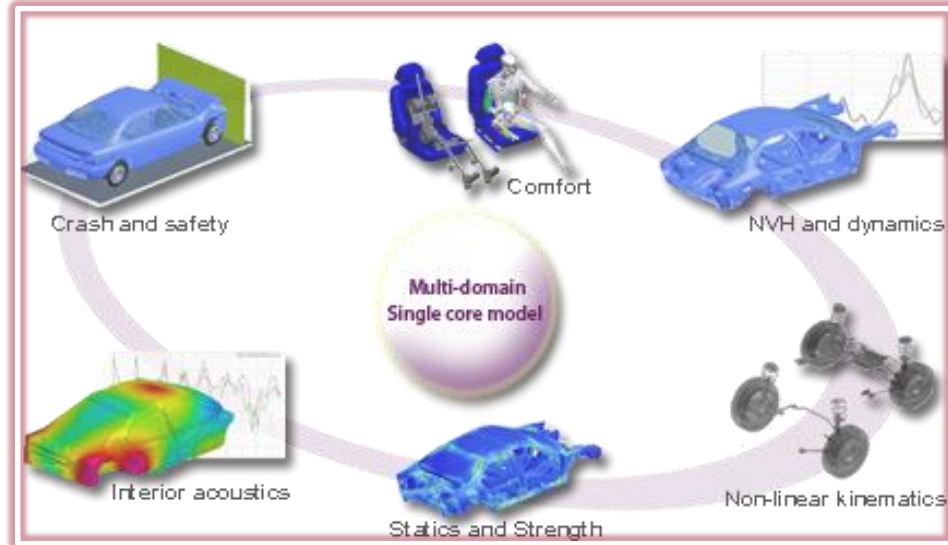
# GRAPHS IN SOFTWARE TOOLS FOR SAFETY-CRITICAL SYSTEMS

# Model-Based Systems Engineering

- Models for engineering of modern cars, aircrafts...
  - Identify problems + optimize the system
    - Early in the design process
    - Without physically building it
    - Multidisciplinary models
    - By simulations + testing + optimizationE.g. virtual crash tests, virtual test scenarios
  - „Model-in-the-loop“
    - Virtual software crashes

Image source:

<http://virtualperformance.esi-group.com/>



# Motivation: Early validation of design rules

## SystemSignalGroup design rule (from AUTOSAR)

### Mapping ISignals to IPDUs

The screenshot displays a software interface for mapping ISignals to IPDUs. The top panel, titled "ISignals", lists several signals and their corresponding IPDU mappings. Below this, the "Position of ISignals in the selected IPDU" section shows a visual representation of the IPDU structure with colored boxes representing different signal groups. At the bottom, a "Problems" window lists four error messages related to the SystemSignalGroup design rule.

ISignals	Signal
B_sigPedalPosition	sigPedalPosition
B_sigSpeedValue	sigSpeedValue
ch_sigEngineTemperature	sigEngineTemperat
ch_sigIgnition	sigIgnition
ch_sigRpm	sigRpm
ch_status	status
ch_status_ccActive	status_ccActive

**Position of ISignals in the selected IPDU**

ch\_status\_ccSpeedU | ch\_status\_ccActive | ch\_status\_ccSp

**Problems**

Description	Resource	Path	Location	Type
✗ ISignal of a grouped System Signal should be mapped to an IPdu along with the ISignal of the System Signal Group	demo_sw.c.ar.xml	/alma	/rootP...	AUTOSAR P...
✗ ISignal of a grouped System Signal should be mapped to an IPdu along with the ISignal of the System Signal Group	demo_sw.c.ar.xml	/alma	/rootP...	AUTOSAR P...
✗ ISignal of a grouped System Signal should be mapped to an IPdu along with the ISignal of the System Signal Group	demo_sw.c.ar.xml	/alma	/rootP...	AUTOSAR P...
✗ Reference iPduTimingSpecification has invalid multiplicity! (Must be in: [1, 1])	demo_sw.c.ar.xml	/alma	/rootP...	AUTOSAR P...

### AUTOSAR:

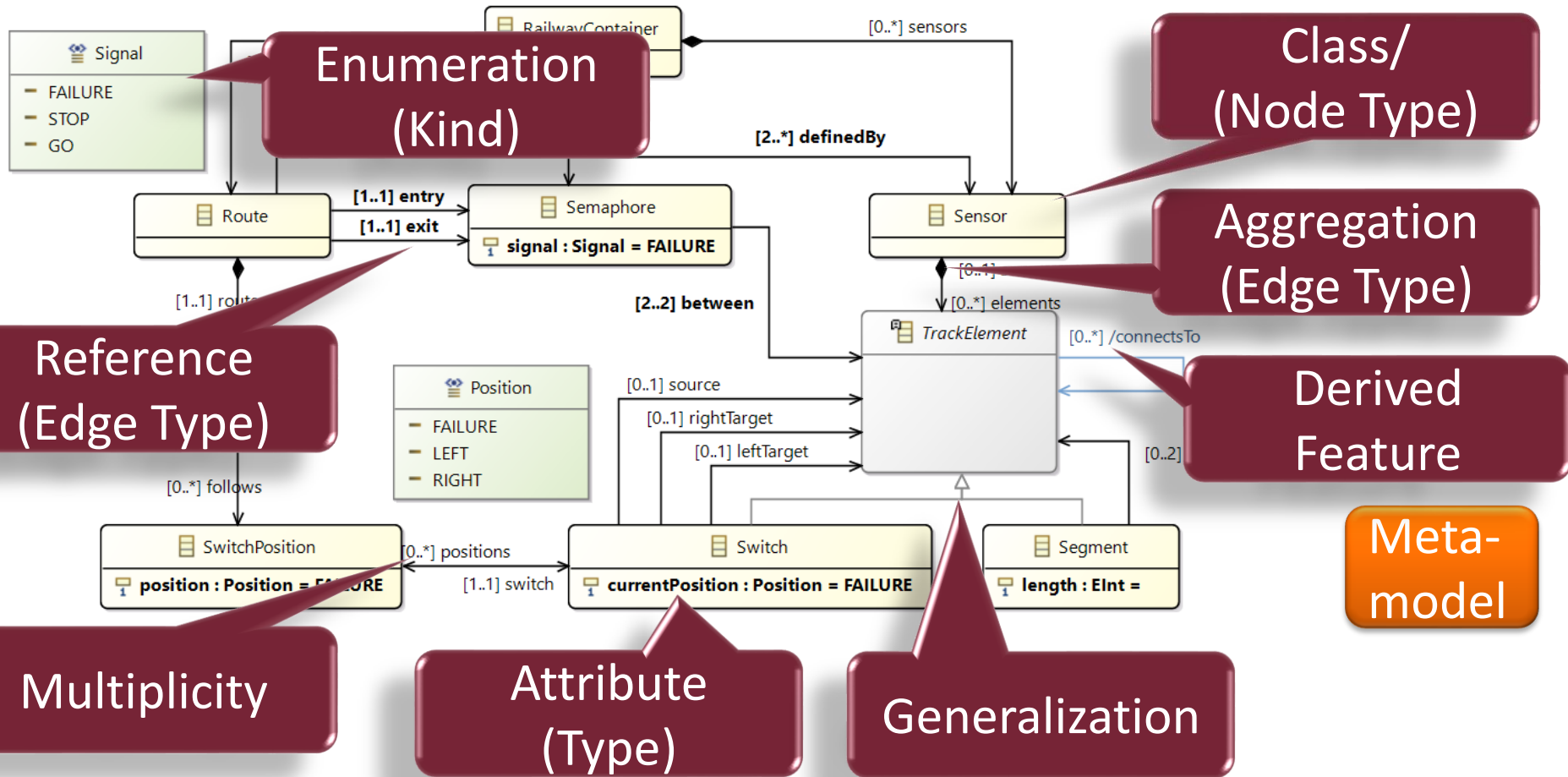
- standardized SW architecture of the automotive industry
  - now supported by modern modeling tools
- Design Rule/Well-formedness constraint:**
- each valid car architecture needs to respect
  - designers are immediately notified if violated

### Challenge:

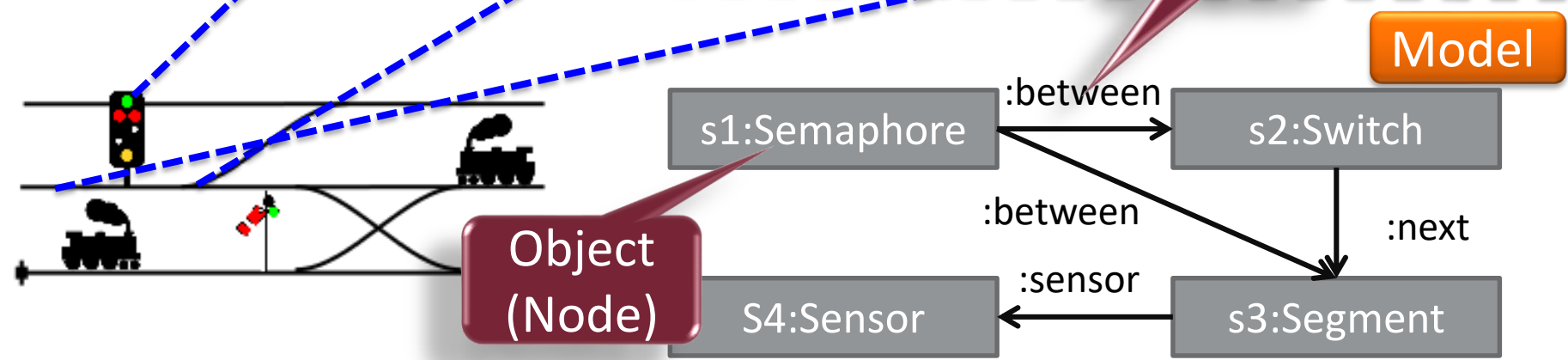
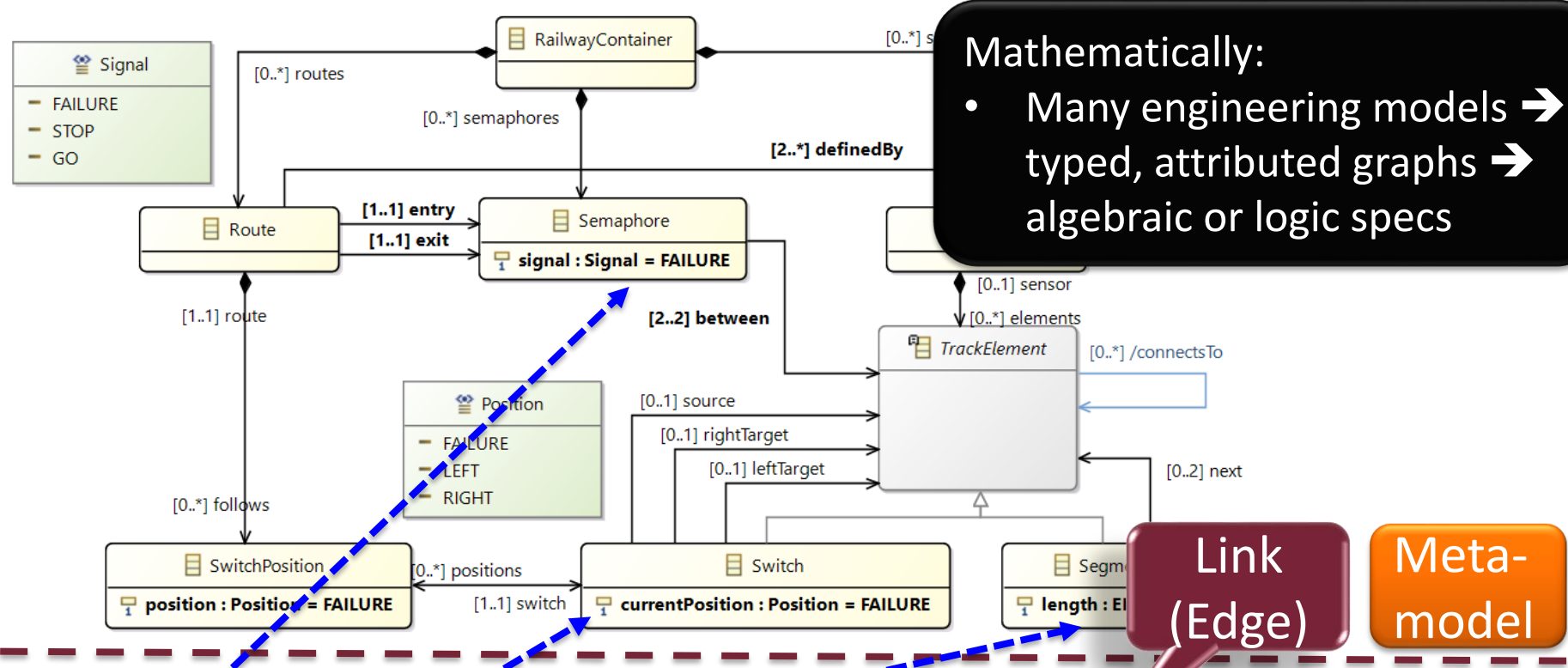
- >500 design rules in AUTOSAR tools
- >1 million elements in AUTOSAR models
- models constantly evolve by designers

Similar challenges at: Thales, NASA JPL, CEA, Ericsson, ThyssenKrupp ...

# Metamodels and (Instance) Models

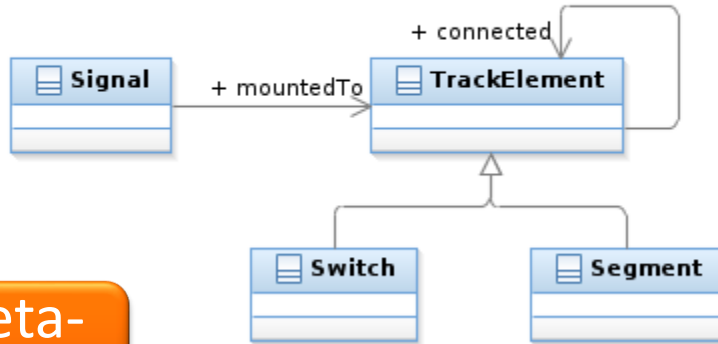


# Metamodels and (Instance) Models



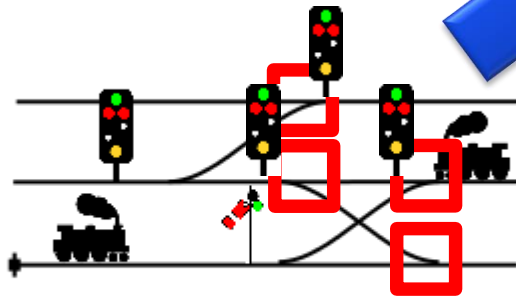
# Validation of Well-formedness Constraints

## Domain-specific modeling languages



Meta-model

Model



Query

```
pattern switchWOSignal(sw) {
  Switch(sw);
  neg find switchHasSignal(sw);
}
```

```
pattern switchHasSignal(sw) {
  Switch(sw);
  Signal(sig);
  Signal.mountedTo(sig, sw);
}
```

Modify

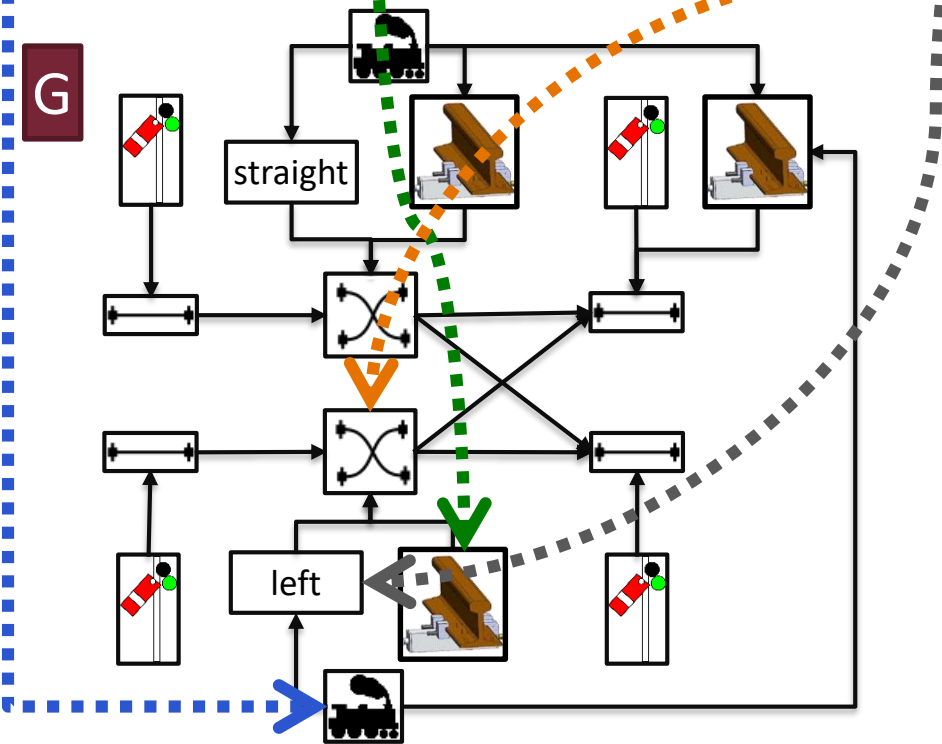
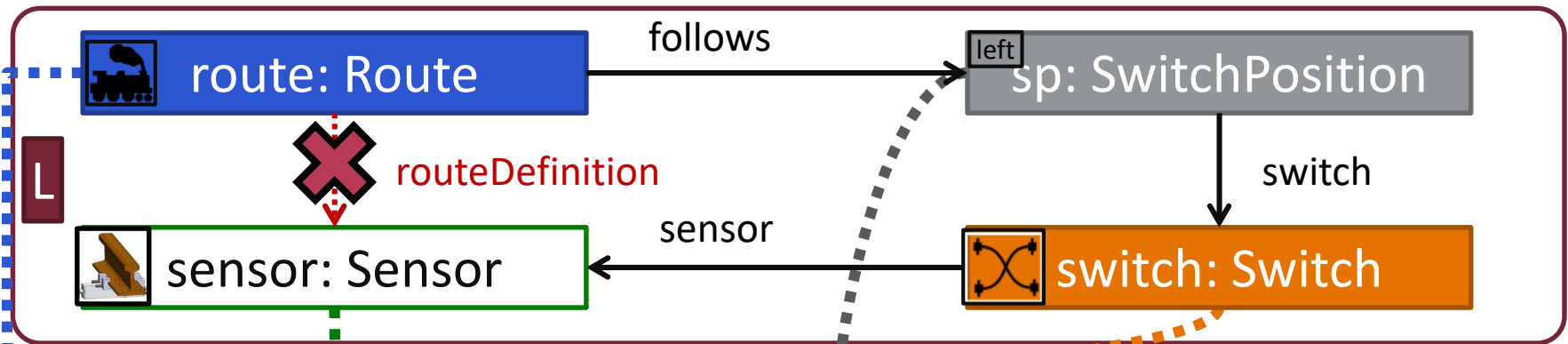
Result



User



# Graph Pattern Matching for Queries

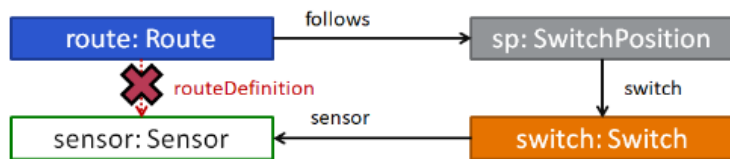


## Match:

- $m: L \rightarrow G$   
(graph morphism)
- CSP:
  - Variables: Nodes of L
  - Constraints: Edges of L
  - Domain values: G
- Complexity:  $|G|^{|L|}$

All sensors with switch that belongs to a route must directly be linked to the same route.

# Which of the following first-order logic formula captures the constraint specified by the graph pattern?



$r, \exists sp, \exists sw, \exists se: Route(r) \wedge follows(r, sp) \wedge$   
 $SwitchPosition(sp) \wedge switch(sp, sw) \wedge$   
 $Switch(sw) \wedge sensor(sp, se) \wedge Sensor(se) \wedge$   
 $routeDef(r, se)$

**A**

$r, \exists sp, \exists sw, \exists se: Route(r) \wedge follows(r, sp) \wedge$   
 $SwitchPosition(sp) \wedge switch(sp, sw) \wedge$   
 $Switch(sw) \wedge sensor(sp, se) \wedge Sensor(se) \wedge$   
 $\neg routeDef(r, se)$

**B**

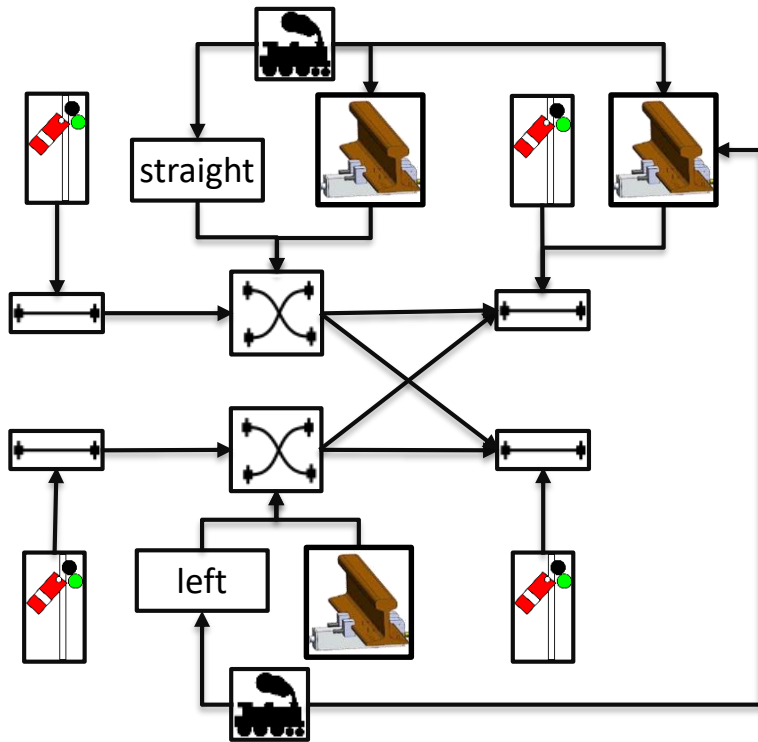
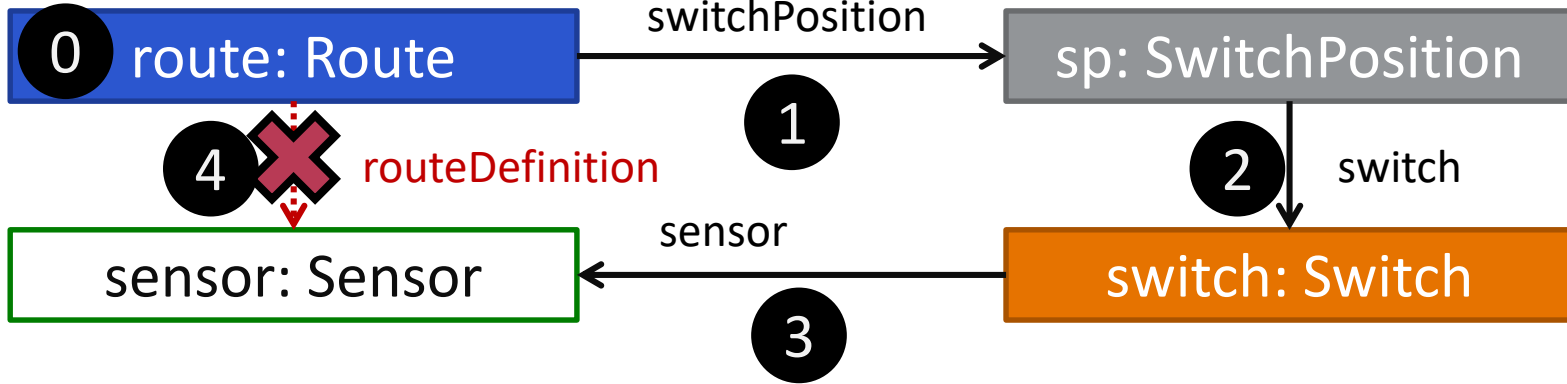
$r, \forall sp, \forall sw, \forall se: Route(r) \wedge follows(r, sp) \wedge$   
 $SwitchPosition(sp) \wedge switch(sp, sw) \wedge$   
 $Switch(sw) \wedge sensor(sp, se) \wedge Sensor(se) \Rightarrow$   
 $\neg routeDef(r, se)$

**C**

$r, \forall sp, \forall sw, \forall se: Route(r) \wedge follows(r, sp) \wedge$   
 $SwitchPosition(sp) \wedge switch(sp, sw) \wedge$   
 $Switch(sw) \wedge sensor(sp, se) \wedge Sensor(se) \Rightarrow$   
 $routeDef(r, se)$

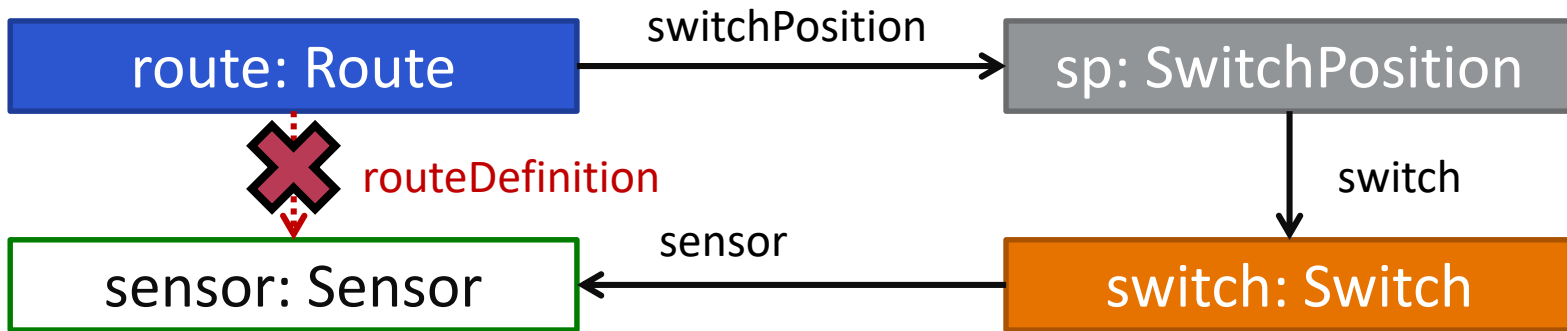
**D**

# Graph Pattern Matching (Local Search)

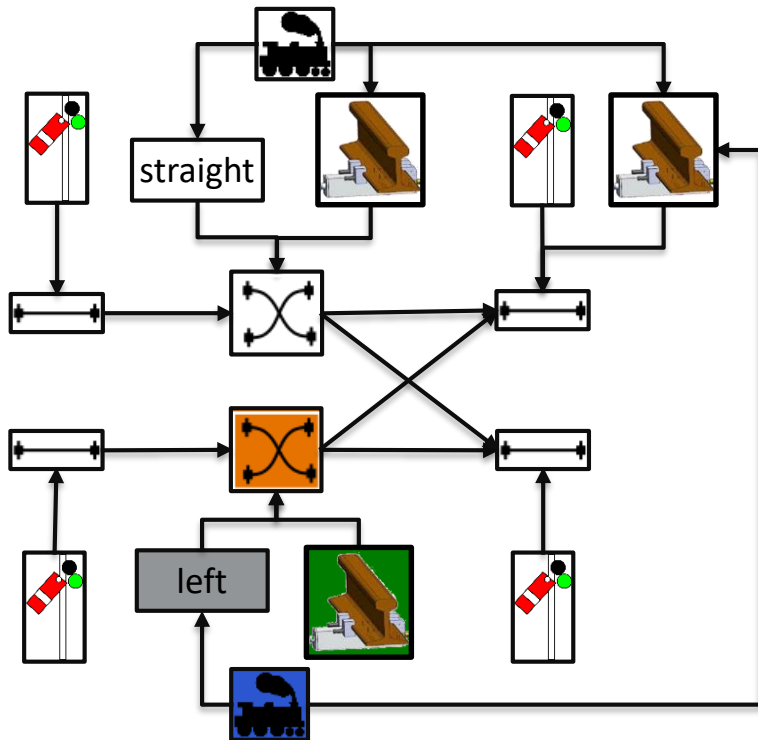


- Search Plan:
  - Select the first node to be matched
  - Define an ordering on graph pattern edges
- Search is restarted from scratch each time

# Incremental Graph Pattern Matching



route	sp	switch	sensor
r1	sp1	sw1	



- Main idea: More space to less time
  - Cache matches of patterns
  - Instantly retrieve match (if valid)
  - Update caches upon model changes
  - Notify about relevant changes
- Approaches:
  - TREAT, LEAPS, RETE, ...
  - Tools: VIATRA, GROOVE, MoTE, TCore

# VIATRA: Open Source Software Project @Eclipse.org

How to improve scalability of modeling tools?

## VIATRA



- Incremental graph query engine
  - Declarative language
  - Incremental graph queries
  - Highly scalable
- Easy integration into tools
  - On-the-fly validation
  - Derived features
  - Custom views
  - Traceability



## An Eclipse project

- Reactive model transformation framework
  - Event-based + reactive execution
  - Internal DSL over Xtend
  - Scalable M2M & M2T
- High-level features
  - Complex event processing
  - Design space exploration
  - Reactive transformations

<http://eclipse.org/viatra>

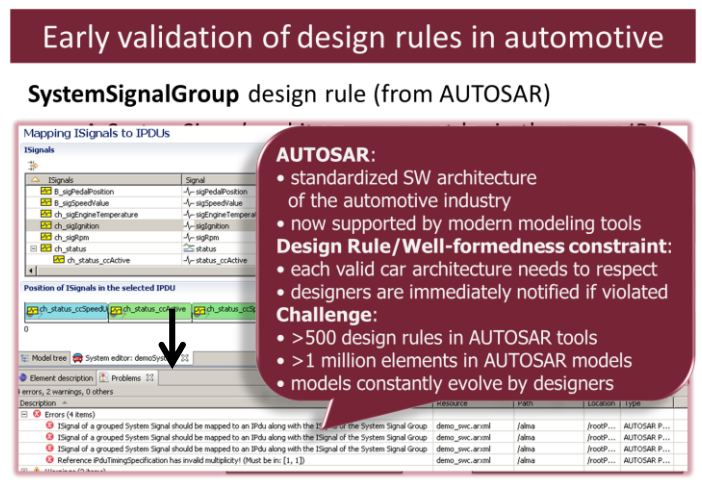
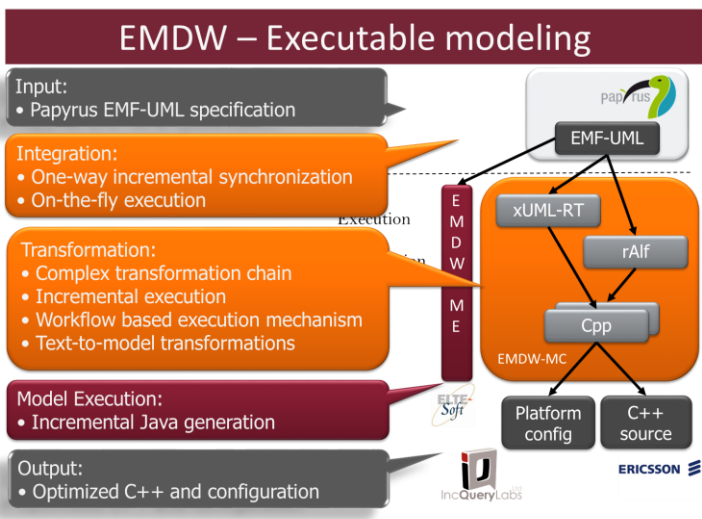
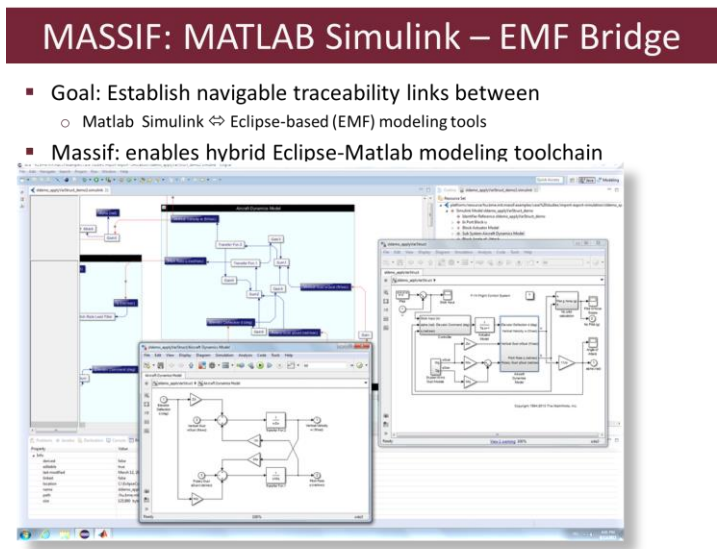
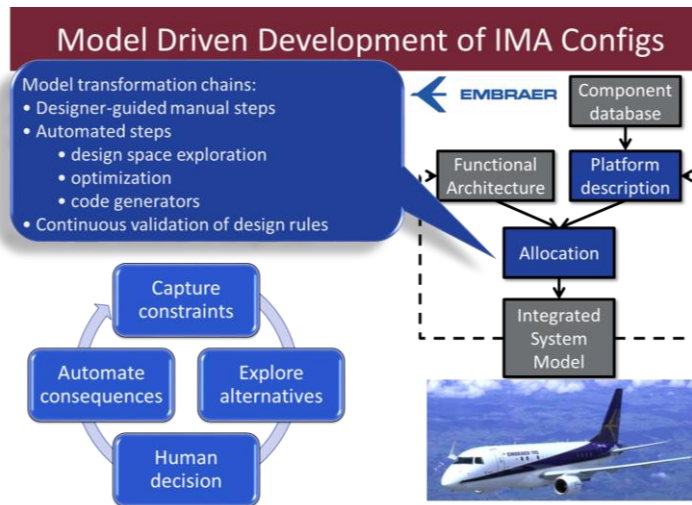
Tool integration with:  
Papyrus UML, Sirius, RMF,  
Capella, ARTOP, mbeddr



Official Eclipse project  
3 Project leads  
10+ Eclipse committers

Industrial use at: Thales, CEA, ThyssenKrupp, Ericsson, Embraer, NASA, CERN, ...

# Industrial Applications of VIATRA



Graphs in Software Tools for  
Safety-Critical Systems

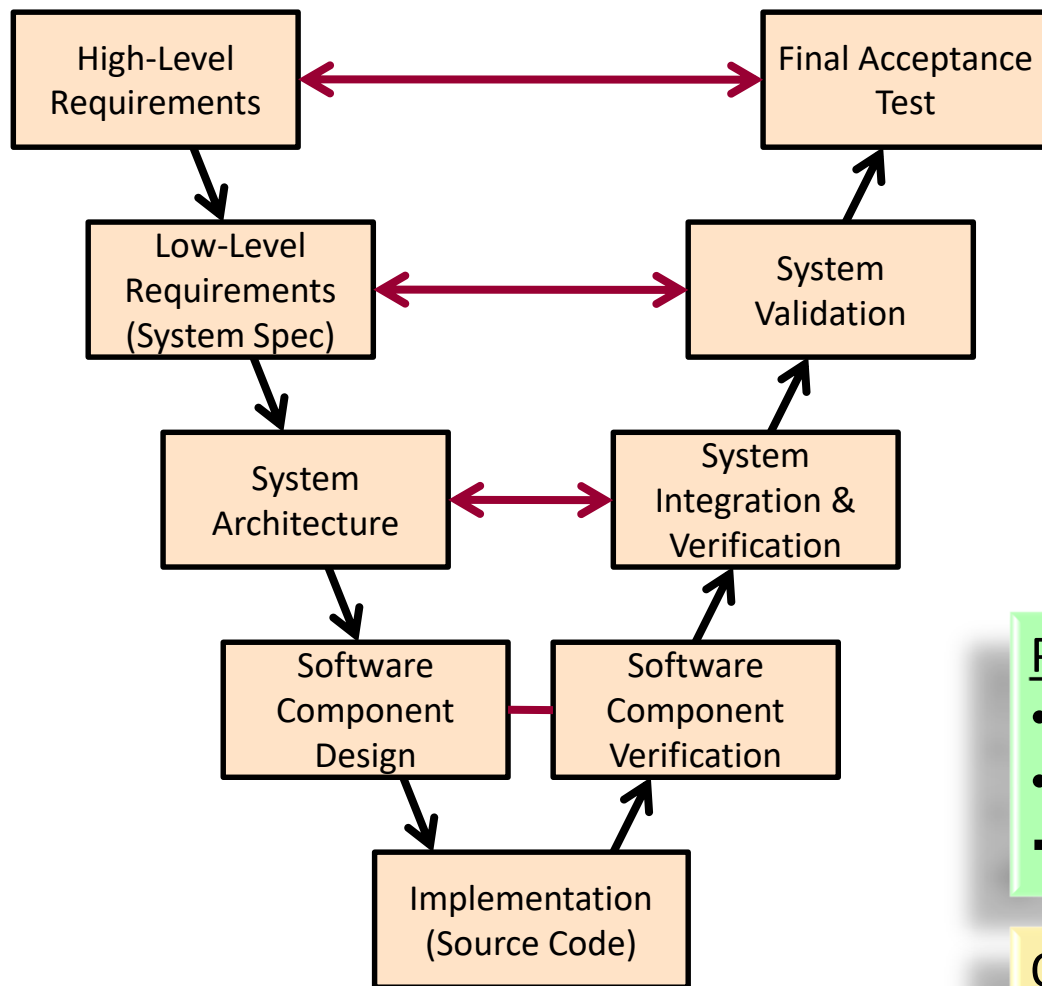
The Need for  
Automated Model Generators

Automated Generation of  
Consistent Graph Models

Automated Synthesis of Diverse  
Graph Models

# THE NEED FOR AUTOMATED GRAPH MODEL GENERATORS

# How to Validate Software Tools?



Development tools:

- input → output deterministically
- introduce new errors

Verification tools:

- fail to detect errors

## Promises of Tool Qualification

- reduce development + V&V cost
- increase quality and productivity
- ➔ reduce certification costs

## Obstacles for Tool Qualification

- ➔ extreme qualification costs
- complex V&V for graph models?
- graph models as test input?

How to systematically test development and verification tools used for critical systems?



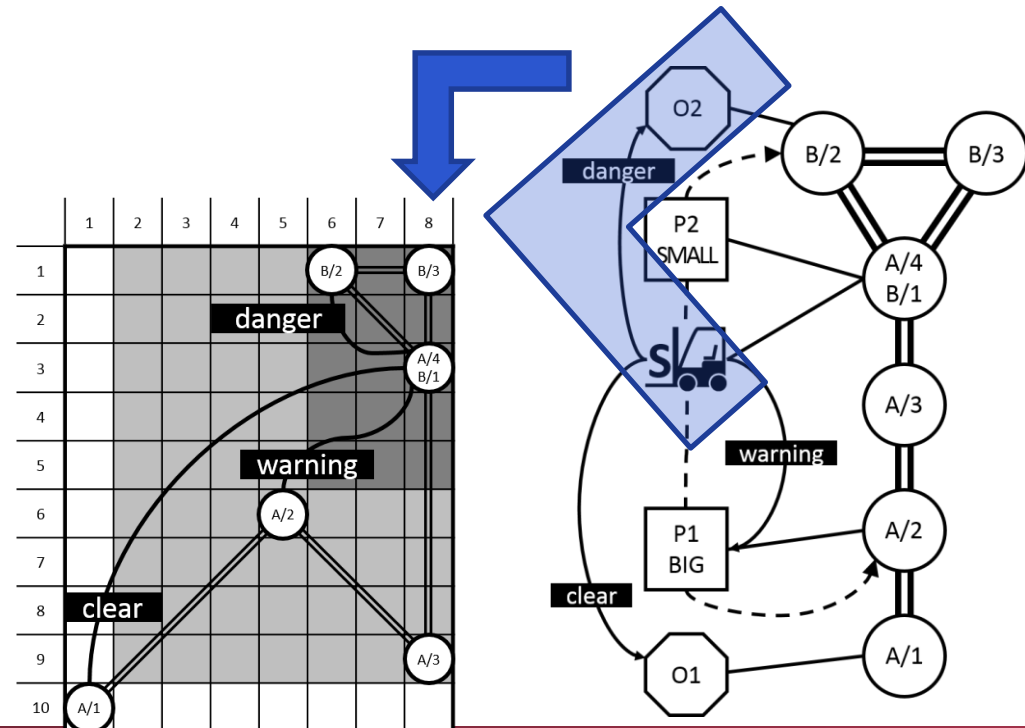
# Automated Model Generation of Test Context

- R3COP & R5COP EU projects:  
Testing of autonomous robots
  - Generate diverse test context (with various obstacles)
  - Test: Navigable by autonomous robot?



## Related challenge:

How to generate test scenarios for safety assurance of autonomous vehicles?



# Towards Automated Graph Generators

How to automatically synthesize graph models which are...

## COnsistent

- Correct: All (well-formedness) constraints are satisfied
- Complete: All (and only) consistent models are derived

ICSE'18

ICSE'19

## REalistic

- Cannot be distinguished from a real graph model
- By removing text+values and evaluating graph metrics

MODELS'16

ICSE'20

## Diverse

- Structural diversity within a single graph
- Large distance between any pair of graph models

FASE'18

STTT

## SCalable

- In size: the size of the graph grows
- In quantity: generation time of next graph is stable

Software Tools for  
Safety-Critical Systems

The Need for  
Automated Model Generators

Automated Generation of  
Consistent Graph Models

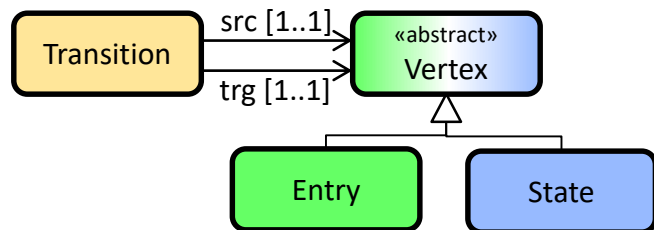
Automated Synthesis of Diverse  
Graph Models

# AUTOMATED GENERATION OF CONSISTENT GRAPH MODELS

# Input of Model Generation: Domain Specification

- Capella (Thales)
- Artop (AUTOSAR)
- **Yakindu (Itemis)**
- MagicDraw (NoMagic)
- Papyrus

## Domain Specification: Metamodel + Constraints

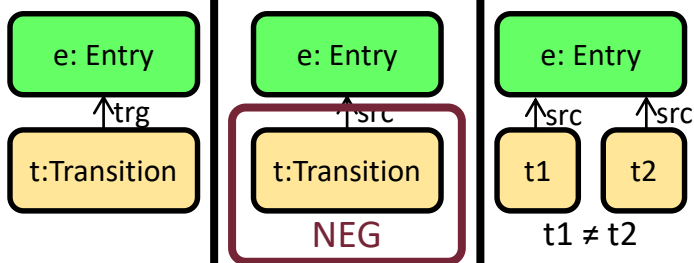


### Metamodel

- Concepts + Relations
- Basic graph structure
- **Statechart:** Transitions, Entries, States, source, target

### Well-formedness constraints

WF1: TrgToEntry    WF2: NoSrcFromEntry    WF3: MultipleSrc

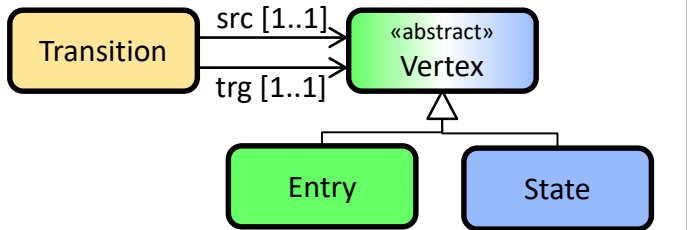


- First order graph predicates (Graph Patterns / OCL)
- Entry is invalid if:
  - **WF1:** Has incoming transition
  - **WF2:** Has no outgoing transition
  - **WF3:** Multiple outgoing transitions

# Output of Consistent Model Generation: Models

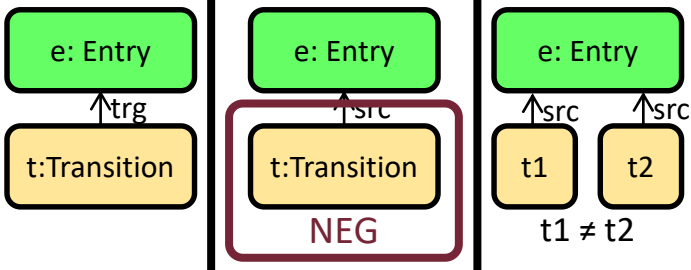
## Domain Specification

### Metamodel



### Well-formedness constraints

WF1: TrgToEntry WF2: NoSrcFromEntry WF3: MultipleSrc

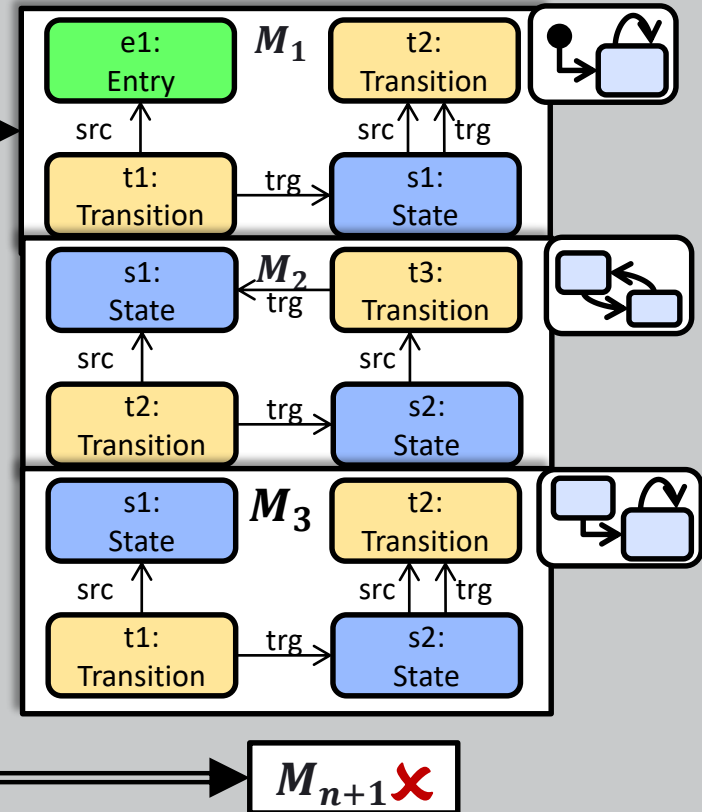


## Model Generator

SAT

UNSAT

## Instance Models



# Model Generation Setup: Logic Solver

Language Specification

Metamodel

## Challenge 1:

Existing solver-based techniques are unable to generate models for industrial modeling languages

- **Alloy (MIT):** 50-100 objects
- **Z3 (Microsoft):** 10-12 objects

**Approach:** create a logic solver that operates natively over graphs

Model Generator

Logic Solver

Alloy: SAT (MIT)  
Z3: SMT (Microsoft)  
Graph solver

Instance Models

e1:  
Entry

M<sub>1</sub>

t2:  
Transition



## Challenge 2:

Logic solvers tend to create similar, highly symmetric models (Copy-Paste)

- No diversity guarantee
- Biased sampling

src

t1:  
Transition

trg

src

s2:  
State

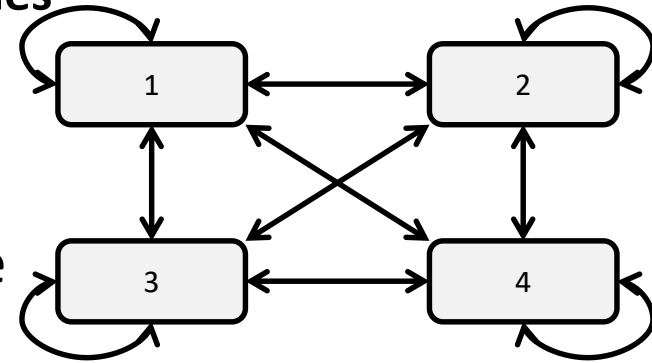
trg

M<sub>n+1</sub> ✗

# Challenges in Model Generation

- Representing graphs as predicates introduce a **large number of variables**

- 1 variable for each **object** and **type** (Transition, Vertex Entry, State)
- 1 variable for each **object pair** and **reference type** (src, trg)
- For 100 objects: more than **20k Boolean variables**



- Quantified well-formedness constraints are unfolded into **complex FOL constraints**

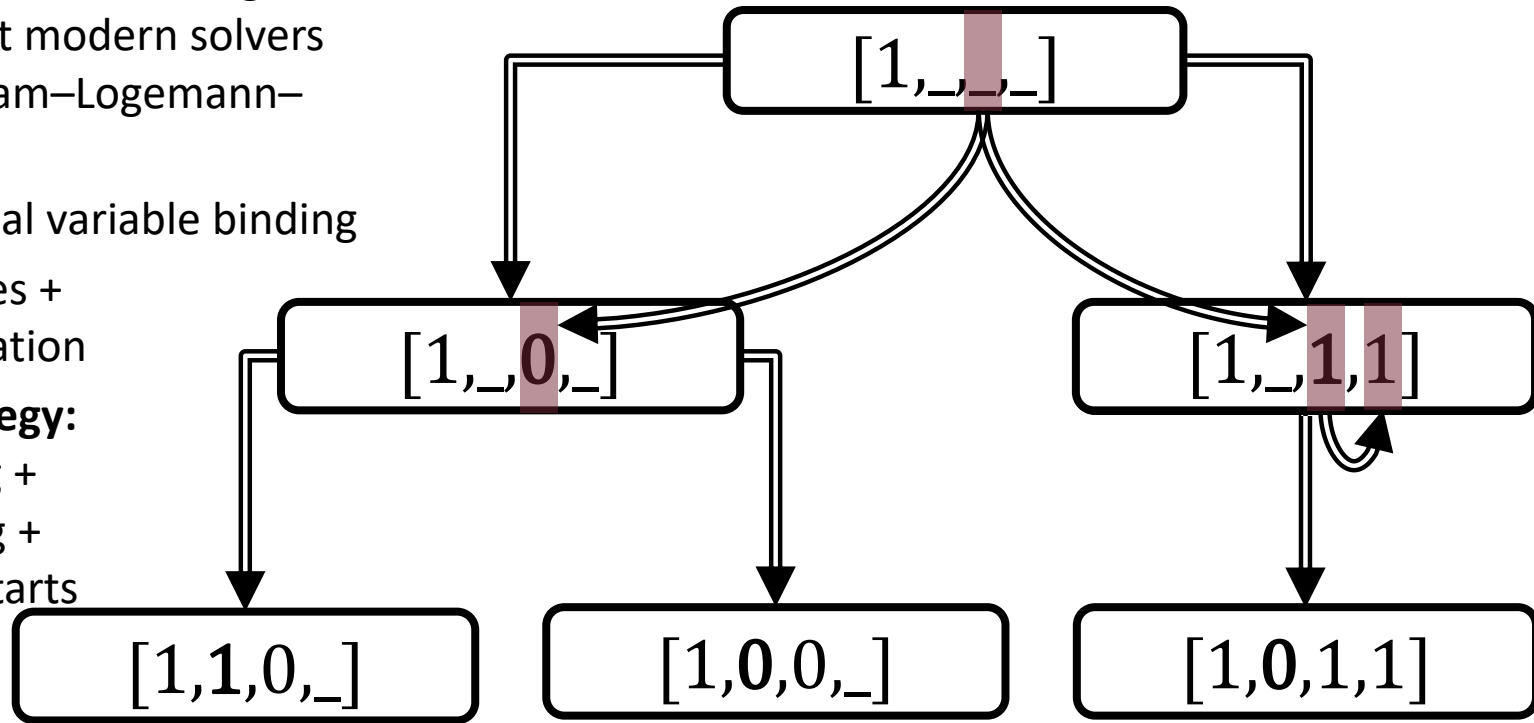
- Each quantified variable is checked for each object
- $\neg \exists e, t: \text{Entry}(e) \wedge \text{Transition}(t) \wedge \text{src}(t, e) \rightarrow$   
**30k atomic expressions** for 100 objects

- Existing solvers fail to generate graphs with more than 100 nodes

# SAT Solver Overview: DPLL Algorithm

$$(A \vee B \vee C) \wedge (\neg C \vee B \vee D) \wedge (\neg A \vee B \vee C) \wedge (\neg A \vee \neg B \vee \neg C)$$

- **DPLL:** Well-known SAT-algorithm, basis of most modern solvers (Davis–Putnam–Logemann–Loveland)
- Refines partial variable binding
- Decision rules + Unit propagation
- **Search Strategy:** Backtracking + Backjumping + Random restarts



**Our approach: Boolean variables → Graphs**

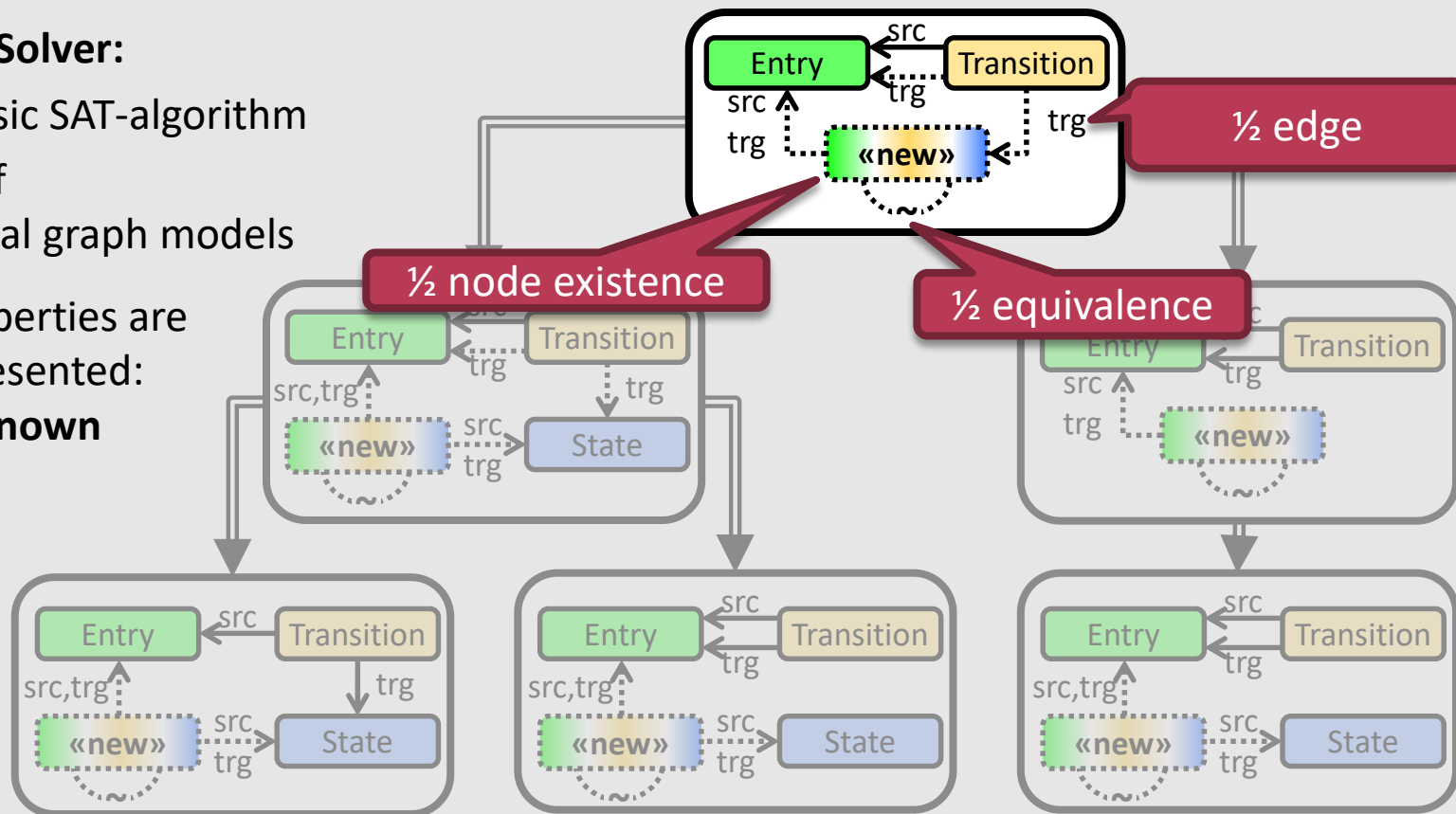




# 3-Valued Partial Models as States

## Graph Solver:

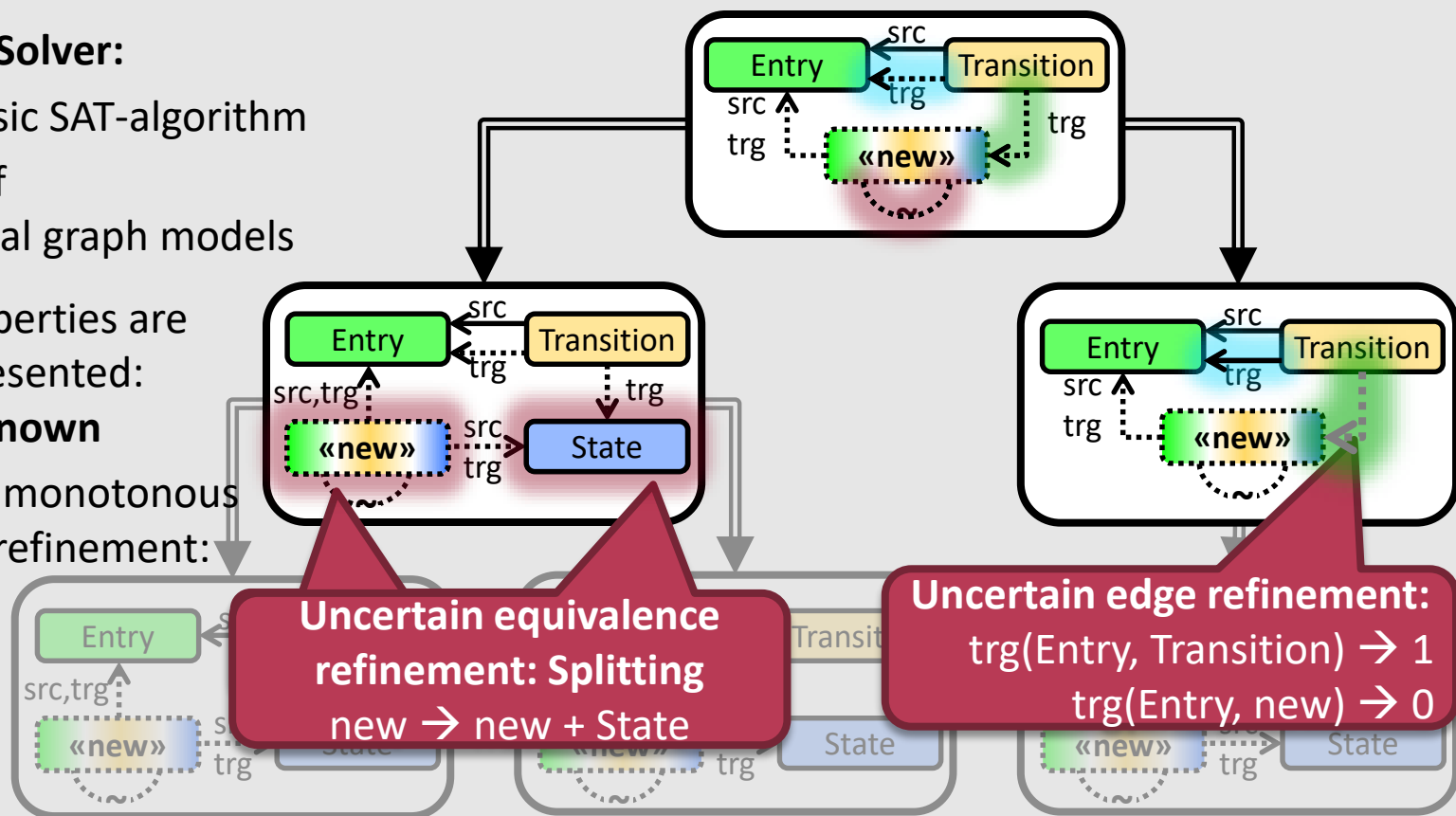
- 1) Based on classic SAT-algorithm
  - 2) Refinement of 3-valued partial graph models
- Uncertain properties are explicitly represented:  
1 | 0 |  $\frac{1}{2}$ : **Unknown**



# Partial Model Refinement

## Graph Solver:

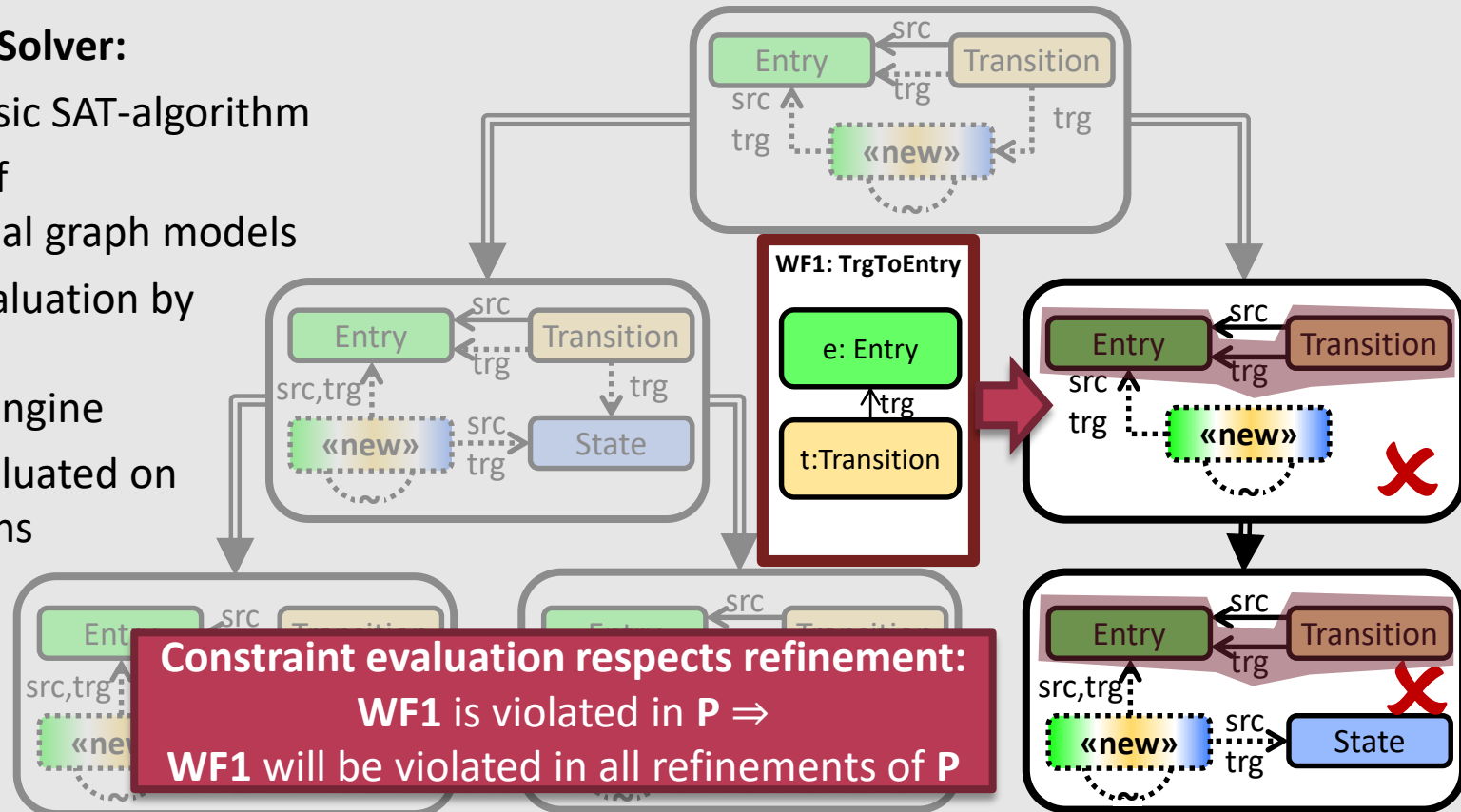
- 1) Based on classic SAT-algorithm
  - 2) Refinement of 3-valued partial graph models
- Uncertain properties are explicitly represented:  
1 | 0 | ½: **Unknown**
  - Generation as monotonous partial model refinement:  
½ → 1|0
  - Decision + Unit prop. → Graph Transformation



# Approximated Constraint Evaluation

## Graph Solver:

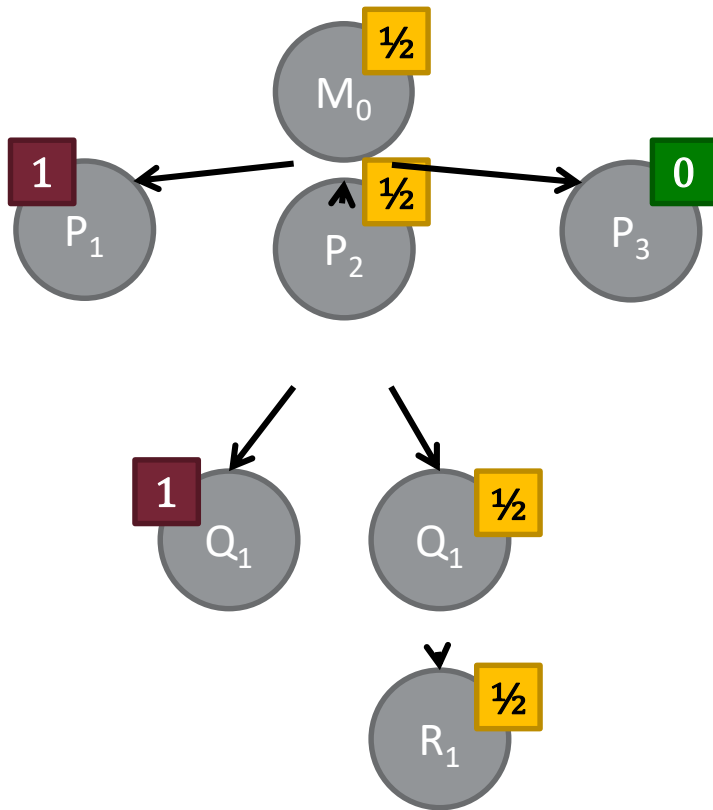
- 1) Based on classic SAT-algorithm
  - 2) Refinement of 3-valued partial graph models
  - 3) Constraint evaluation by incremental graph query engine
- Constraint evaluated on partial solutions
  - Monotonous reasoning
  - Incremental constraint reevaluation



**Approximated constraint evaluation is monotonous (despite the use of negation)**

# Overview of Refinement and Approximation

- Overview of refinement & constraint approximation

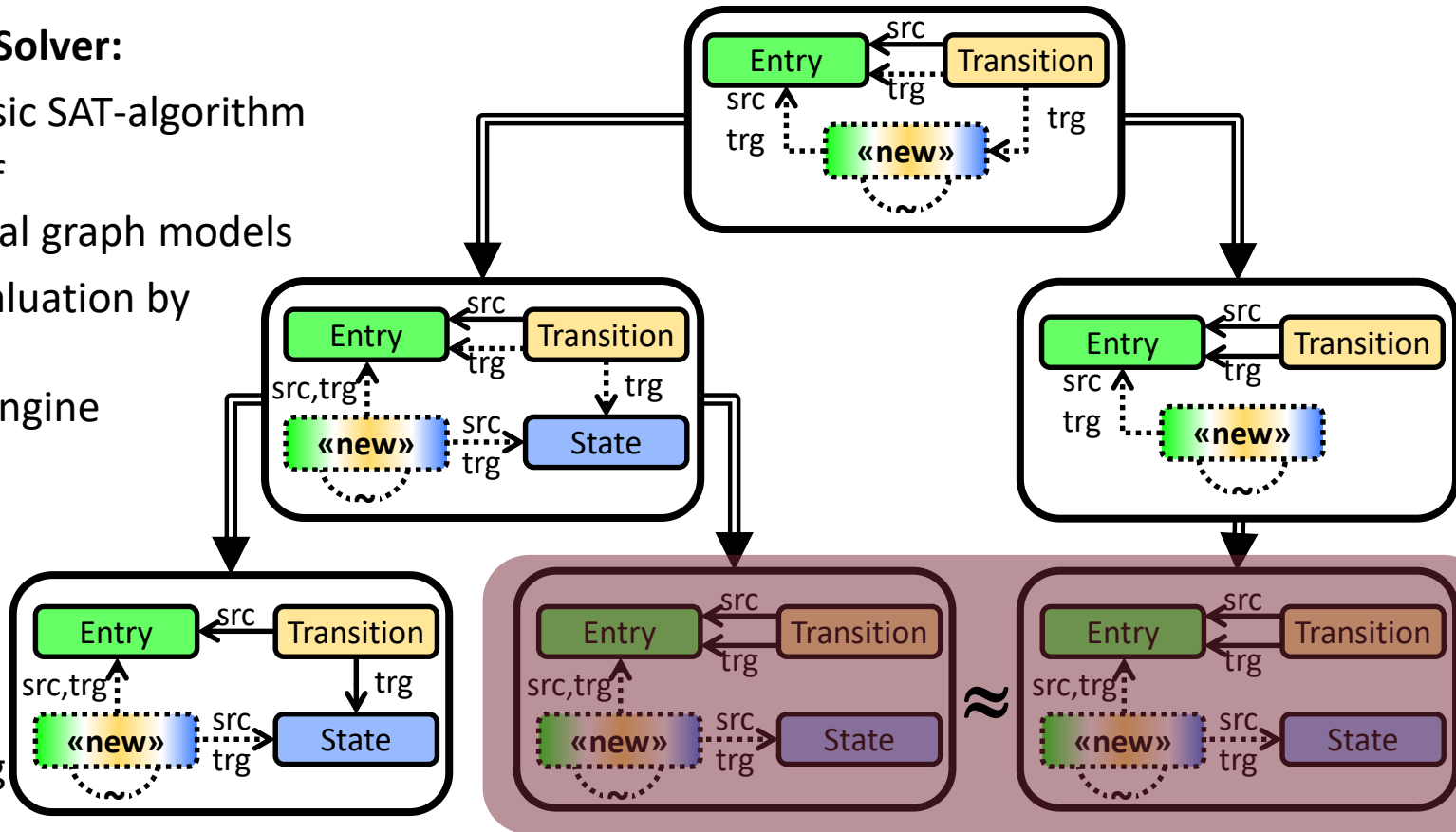


$\varphi[P]$	$\varphi[Q]$ (with $P \succ Q$ )	Action
1	1	Inconsistent, backtrack
0	0	Consistent
$\frac{1}{2}$	1	Inconsistent, backtrack
$\frac{1}{2}$	0	Consistent (corrected)
$\frac{1}{2}$	$\frac{1}{2}$	Unknown

# Equivalence Partitioning

## Graph Solver:

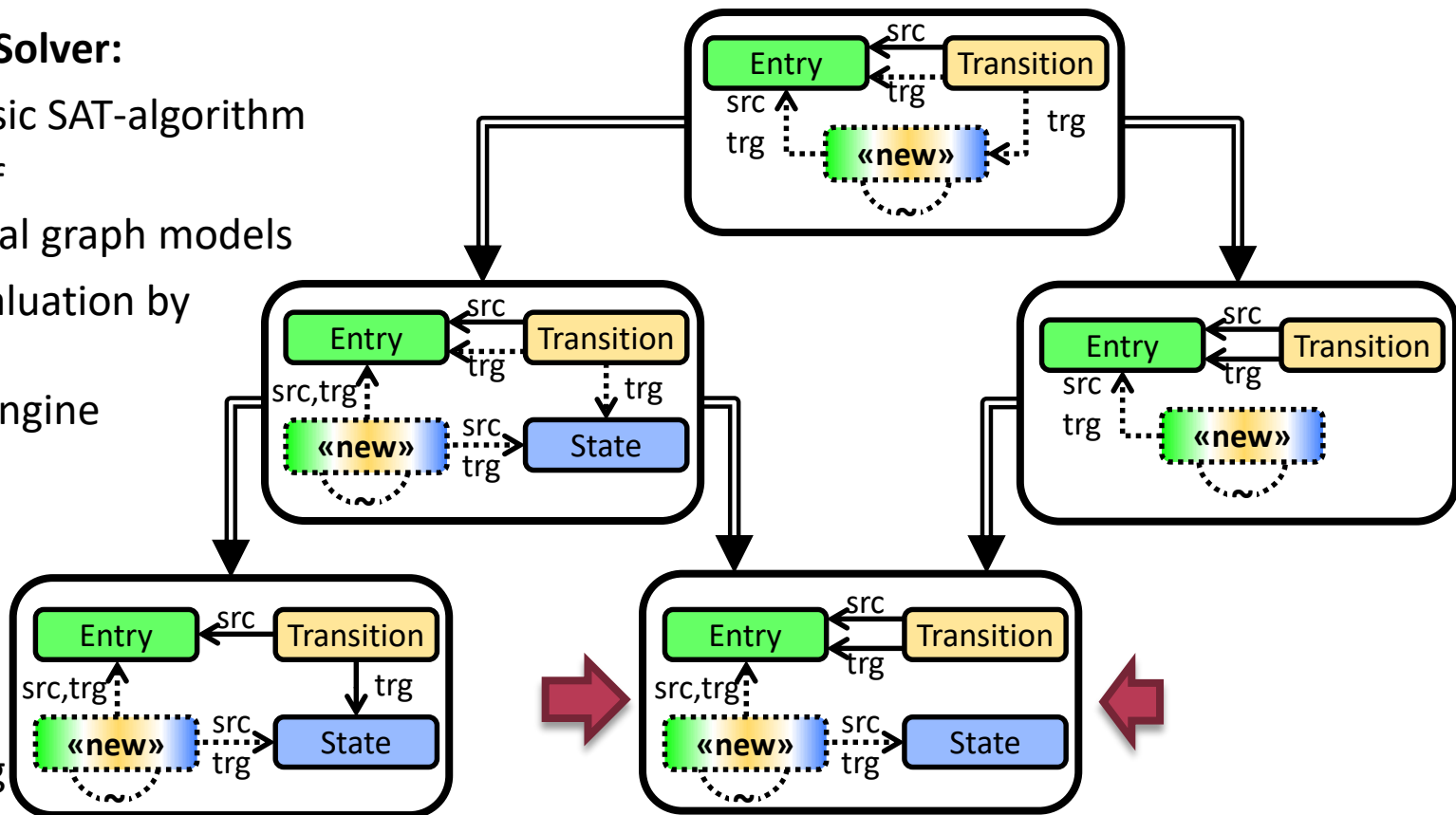
- 1) Based on classic SAT-algorithm
  - 2) Refinement of 3-valued partial graph models
  - 3) Constraint evaluation by incremental graph query engine
  - 4) Equivalence detection by graph isomorphism
- State encoding



# Equivalence Partitioning

## Graph Solver:

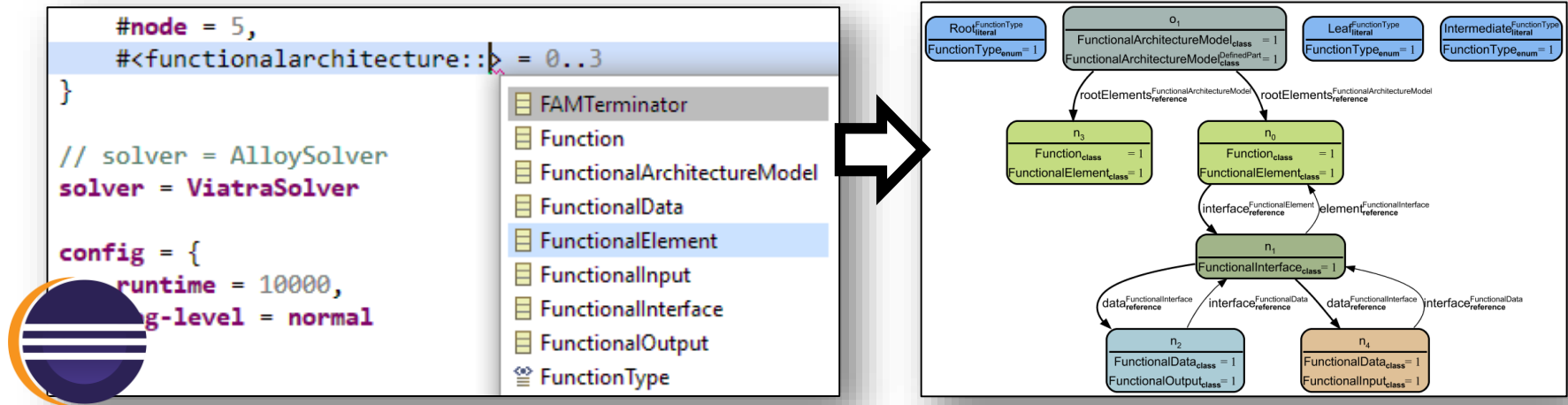
- 1) Based on classic SAT-algorithm
  - 2) Refinement of 3-valued partial graph models
  - 3) Constraint evaluation by incremental graph query engine
  - 4) Equivalence detection by graph isomorphism
- State encoding
  - Partial order reduction




Different Solutions

# VIATRA Solver: An Open Source Tool

- Standard EMF as input and output | Configuration language | Visualization



- Incremental Query Engine:
  - Constraint language: VIATRA Query 
  - Internally uses: Incremental constraint reevaluation, DPLL as VIATRA DSE
- Open source: [github.com/viatra/VIATRA-Generator](https://github.com/viatra/VIATRA-Generator)



# Scalability Measurements

## Maximal model size

	Largest model (#Objects)		
	Graph Solver	Sat4J	MiniSat
FAM+WF	6250	58	61
FAM-WF	7000	87	92
Yak+WF	1000	–	–
Yak-WF	7250	86	90
FS	4750	87	89
Ecore	2000	38	41

FAM: Industrial, Avionics

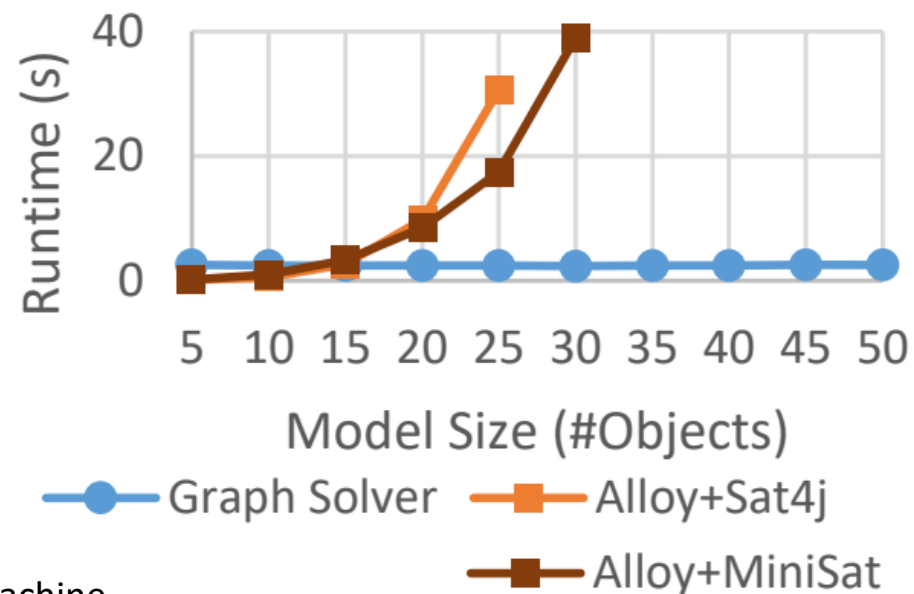
Yakindu: Industrial, Statemachine

FS: File System example of Alloy

Ecore: Metamodelling language

5 min timeout

## Example comparison (FAM)



Our solver generates ~two orders of magnitude larger models

Graphs in Software Tools for  
Safety-Critical Systems

The Need for  
Automated Model Generators

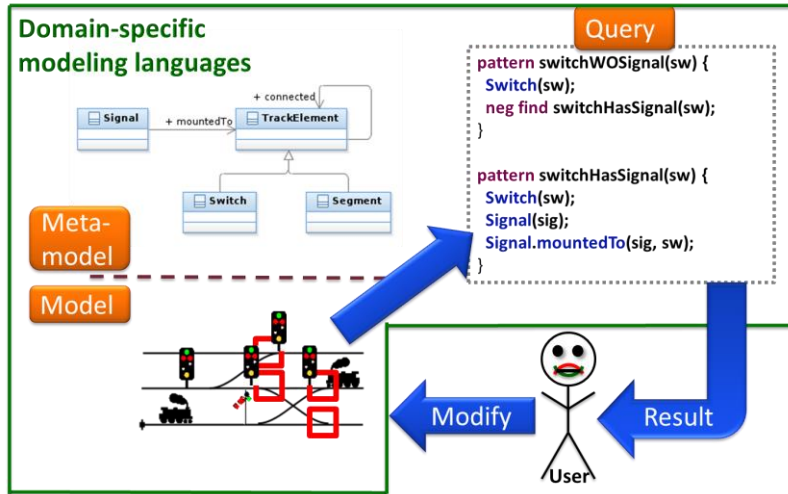
Automated Generation of  
Consistent Graph Models

Automated Synthesis of Diverse  
Graph Models

# GENERATION OF DIVERSE GRAPH MODELS

# Conclusions

## Validation of Well-formedness Constraints



## Towards Automated Graph Generators

How to automatically synthesize graph models which are...

**Consistent**

- Correct: All (well-formedness) constraints are satisfied
- Complete: All (and only) consistent models are derived

ICSE'18 ICSE'19

**Realistic**

- Cannot be distinguished from a real graph model
- By removing text+values and evaluating graph metrics

MODELS'16 ICSE'20

**Diverse**

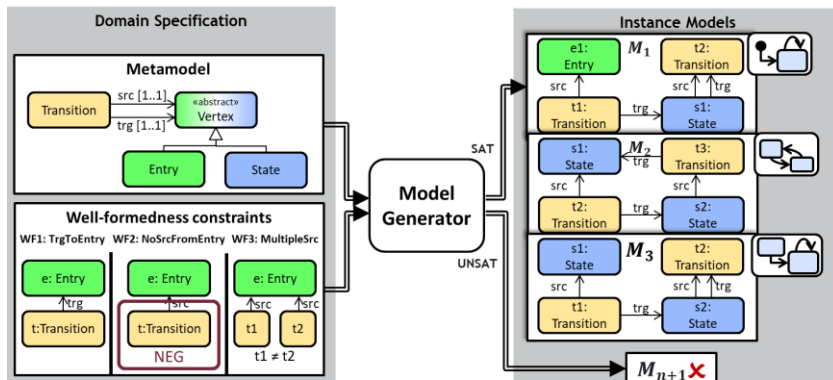
- Structural diversity within a single graph
- Large distance between any pair of graph models

FASE'18 STTT

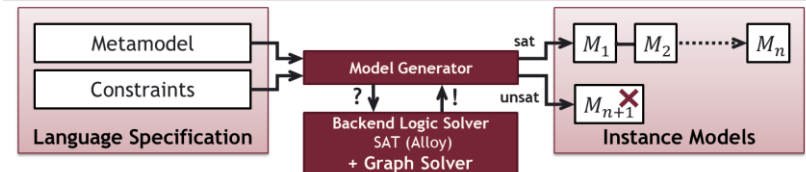
**Scalable**

- In size: the size of the graph grows
- In quantity: generation time of next graph is stable

## Output of Consistent Model Generation: Models



## Generation of Diverse Models



### Testing Challenge:

- High coverage = Diverse set of graph models
- **C1**: Lack of diversity metrics for graph models
- **C2**: Logic solvers provide poor quality test suites
- **S1**: Introduce diversity metrics for graphs
- **S2**: Guide solvers to generate diverse test suites

# THANKS FOR YOUR ATTENTION

**Links to tools:**

VIATRA: <https://www.eclipse.org/viatra/>

VIATRA Generator: <https://github.com/viatra/VIATRA-Generator>