Graph Abstraction & Model Generation Techniques

Oszkár Semeráth, *Rebeka Farkas*

MŰEGYETEM 1782

Budapest University of Technology and Economics
Department of Measurement and Information Systems
ftsrg Research Group

ftsrg

Main topics of the course

Graph Abstraction Techniques

- Partial Modeling (MAVO)
- Shaping (Neighbourhood, TVLA)

Model generation

- Motivation & Use-Cases
- Requirements & Objectives: the CoReDiSc criteria
- Related approaches
  - Graph Solver
  - Solver-Based Approaches
  - Design-Space Exploration

Summary & Learning Outcomes

Graph Abstraction Techniques

Graph Abstraction

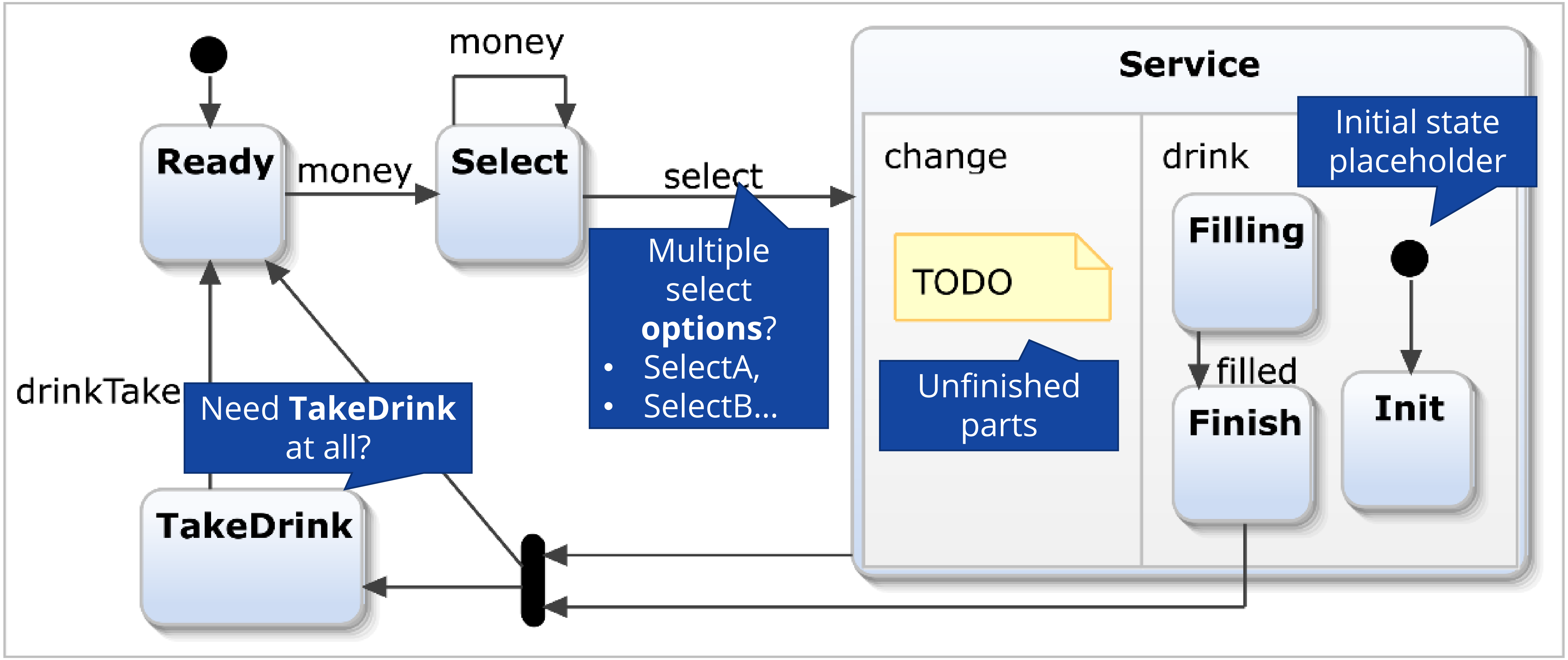
- **Previously:** concrete graph-based models
- **Motivation:** capture a range of potential models
  - Uniform handling of a range of graphs
  - Verification of graph-based systems

- Terminology:
  - $abstraction\colon\ graph \rightarrow abstract graph$
  - Refinement between abstract graphs: $A_1 \sqsubseteq A_2$
  - Concretization of an abstract graph A is G if $abstraction(G) \sqsubseteq A$

- **Goal:** illustrate useful graph abstraction techniques

Partial Modeling

Example: Unfinished models



Motivation

- Early phase of development → high uncertainty in the models
- Editor forces the developer to work with complete models

Missing ⇔ Undecided / Uncertain / Unknown

Model refinement ⇔ Model rewriting

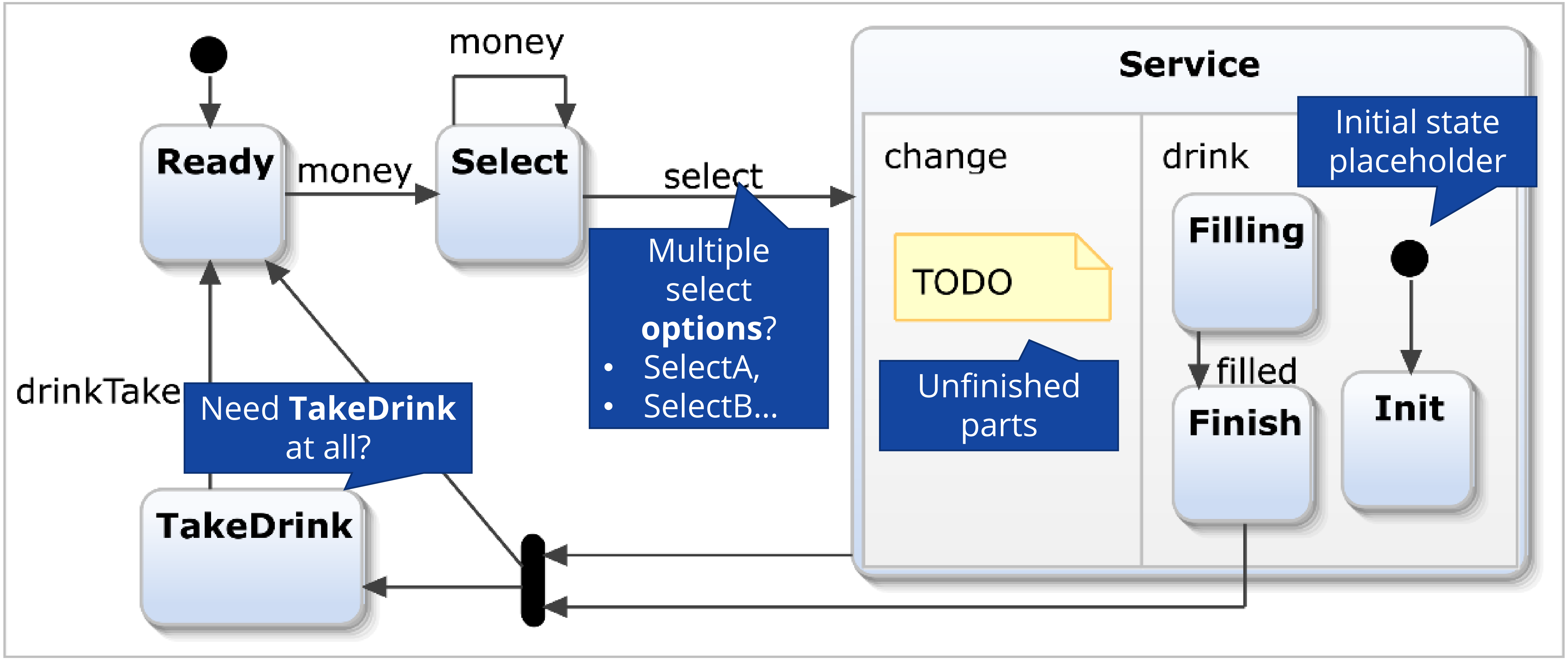
- **Issues:**
  - Forces the developer to make premature decisions
  - No way to list / document design alternatives
  - Editor mixes: invalid ⇔ unfinished
- **Clarify the semantics of missing elements**

Partial Modeling

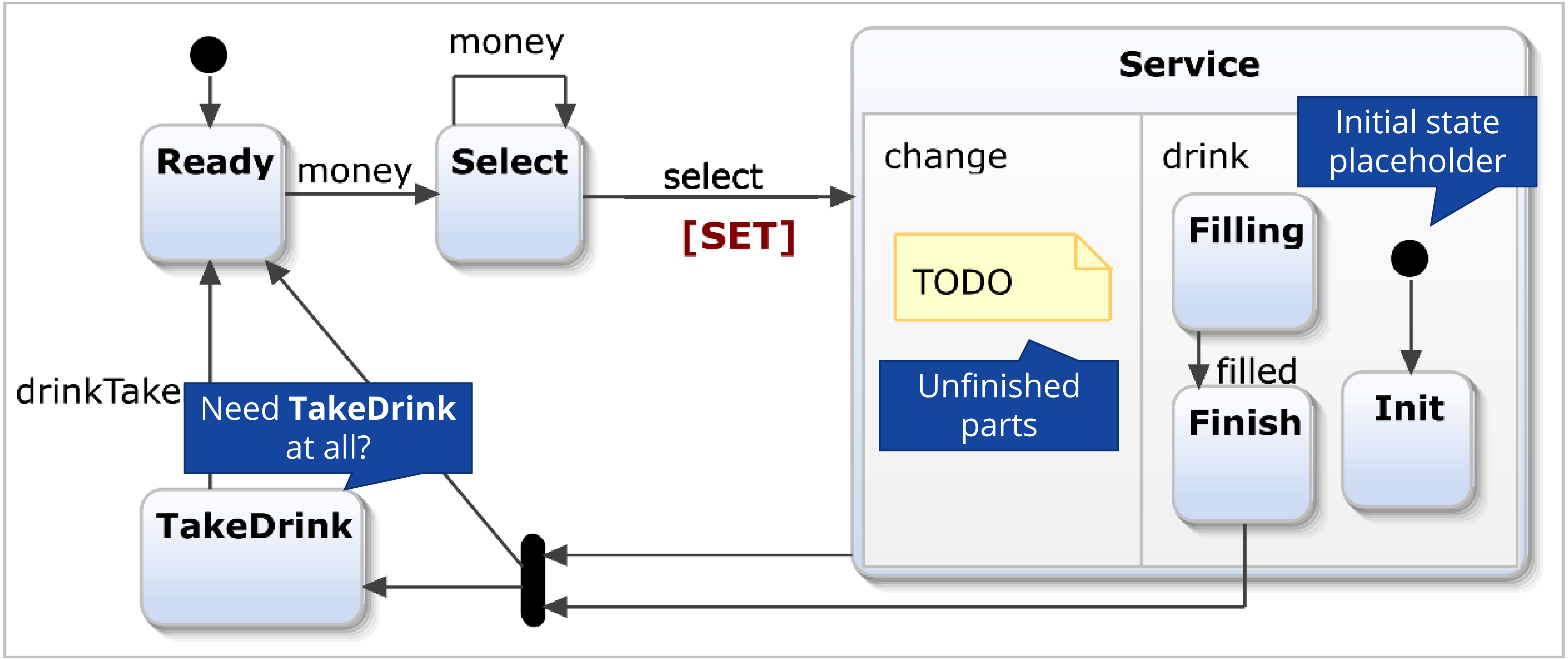
- Generic technique to explicitly represent uncertainty in models
  - Generic: works for every metamodel
  - Explicitly represent: uncertainty = model element
  - In Models: The uncertainty is attached to the models
- **MAVO:** practical way to annotate model with uncertainty
  - **M**ay: elements can be omitted
  - **A**bstract (Set): representing sets of elements
  - **V**ar: elements that can be merged
  - **O**pen: new elements can be added
- Automation: generate alternatives, check all alternatives

Michalis Famelis, Rick Salay, and Marsha Chechik. Partial models: towards modeling and reasoning with uncertainty. In: Proceedings of the 34th International Conference on Software Engineering, pp. 573–583. IEEE Press, 2012.

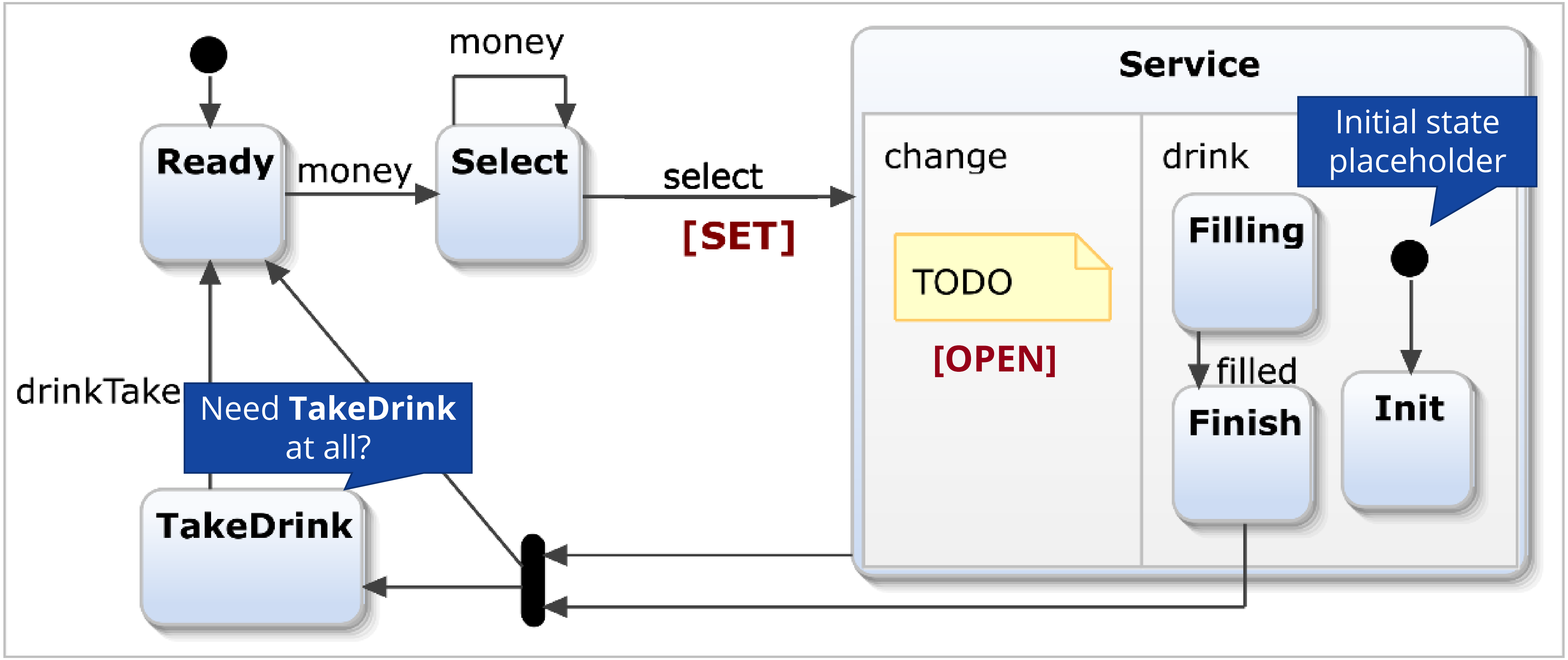
Example: Unfinished models with MAVO



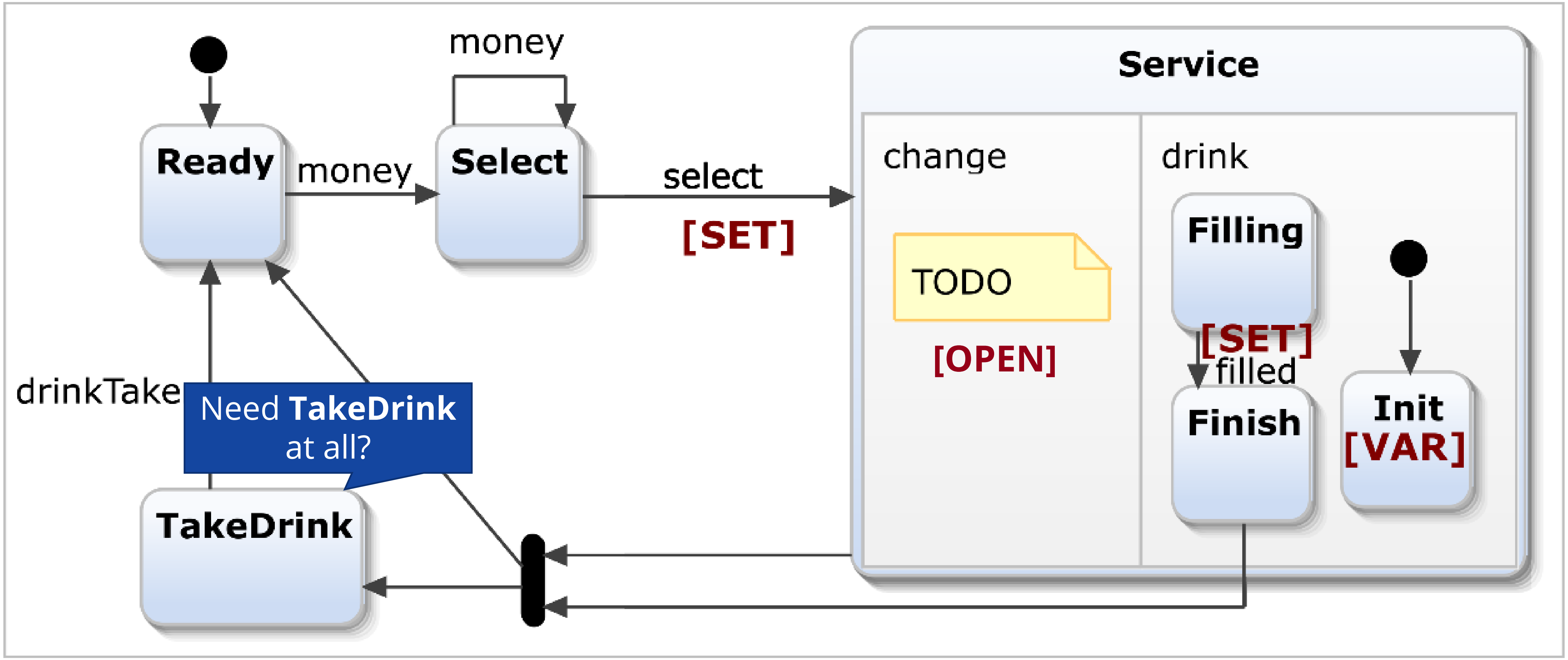
Example: Unfinished models with MAVO



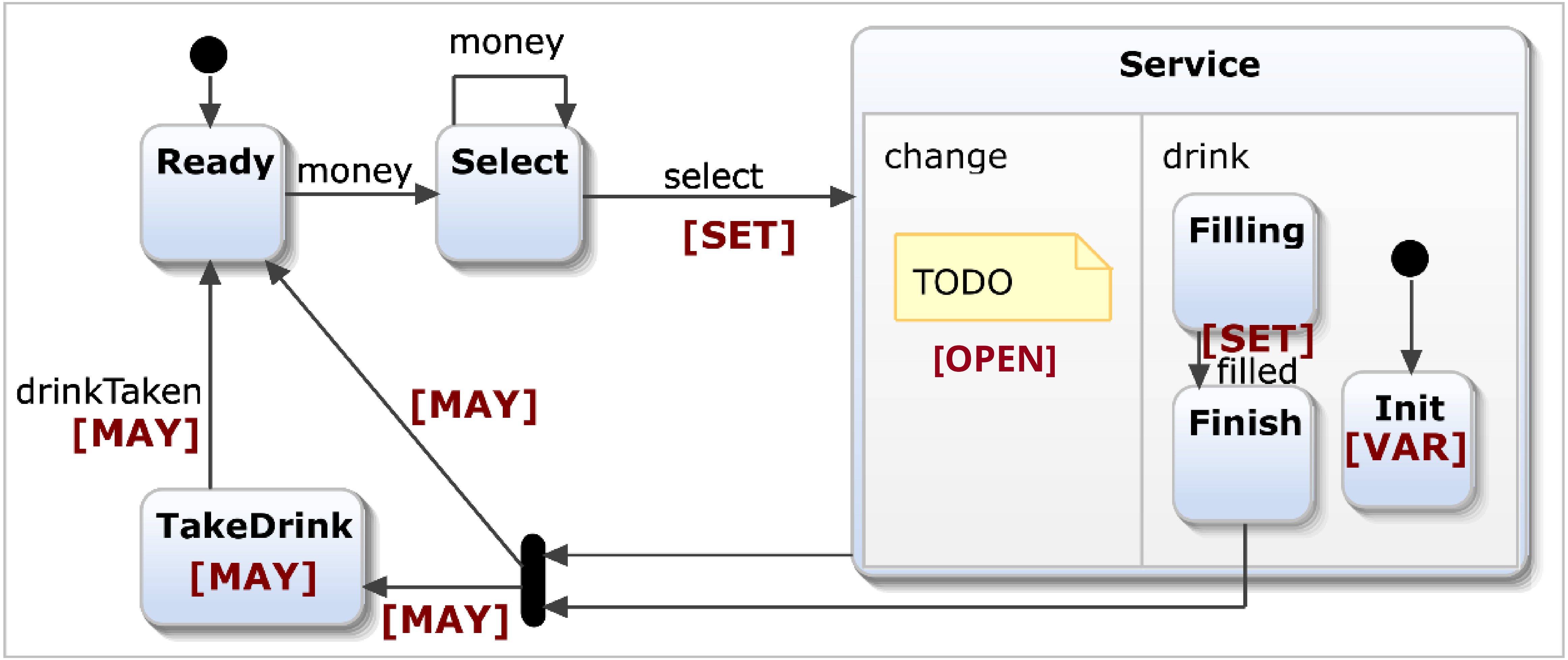
Example: Unfinished models with MAVO



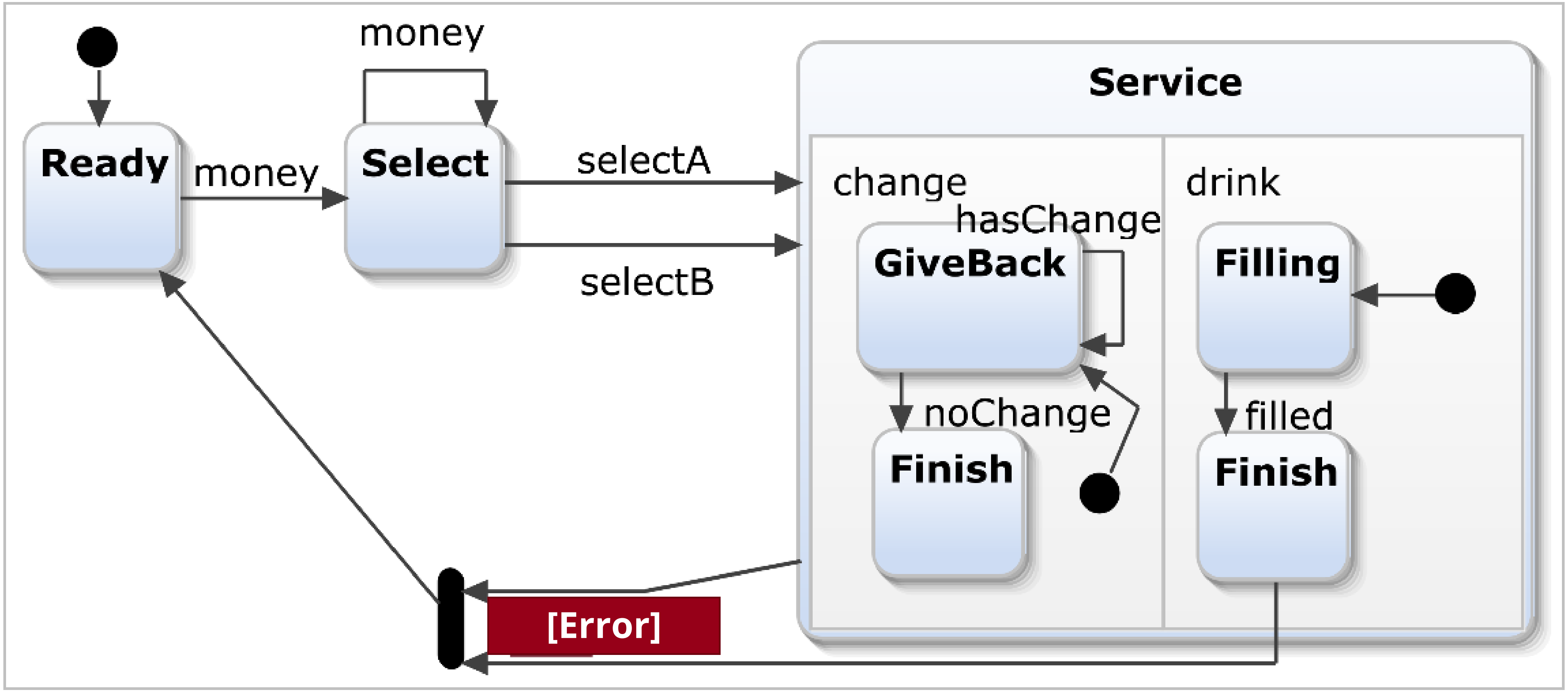
Example: Unfinished models with MAVO



Example: Unfinished models with MAVO



Example: Example concretization



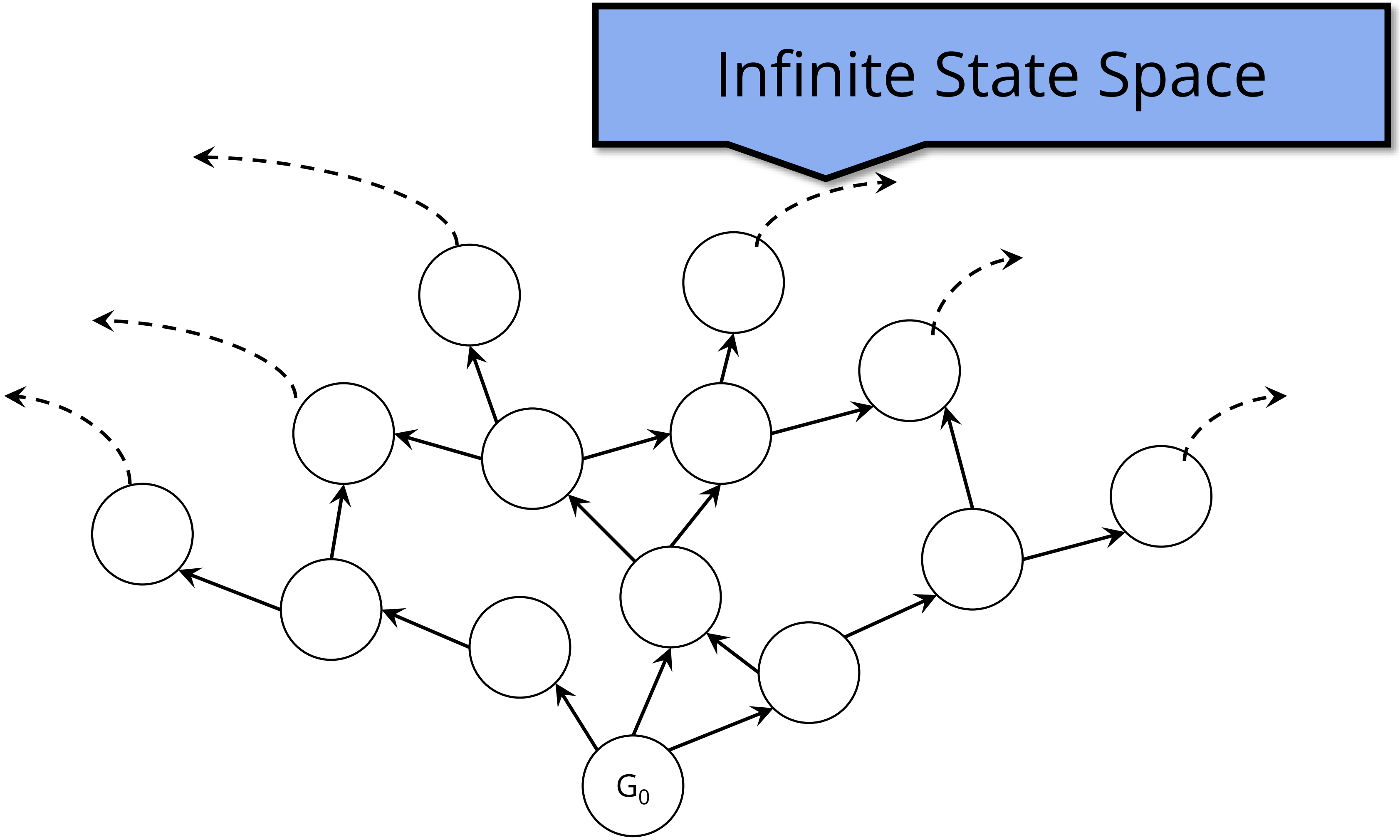
Partial Modeling Summary

- Partial modeling captures the uncertainty of models
- 1 partial model = set of complete model
- MAVO: framework for uncertainty annotation + tooling

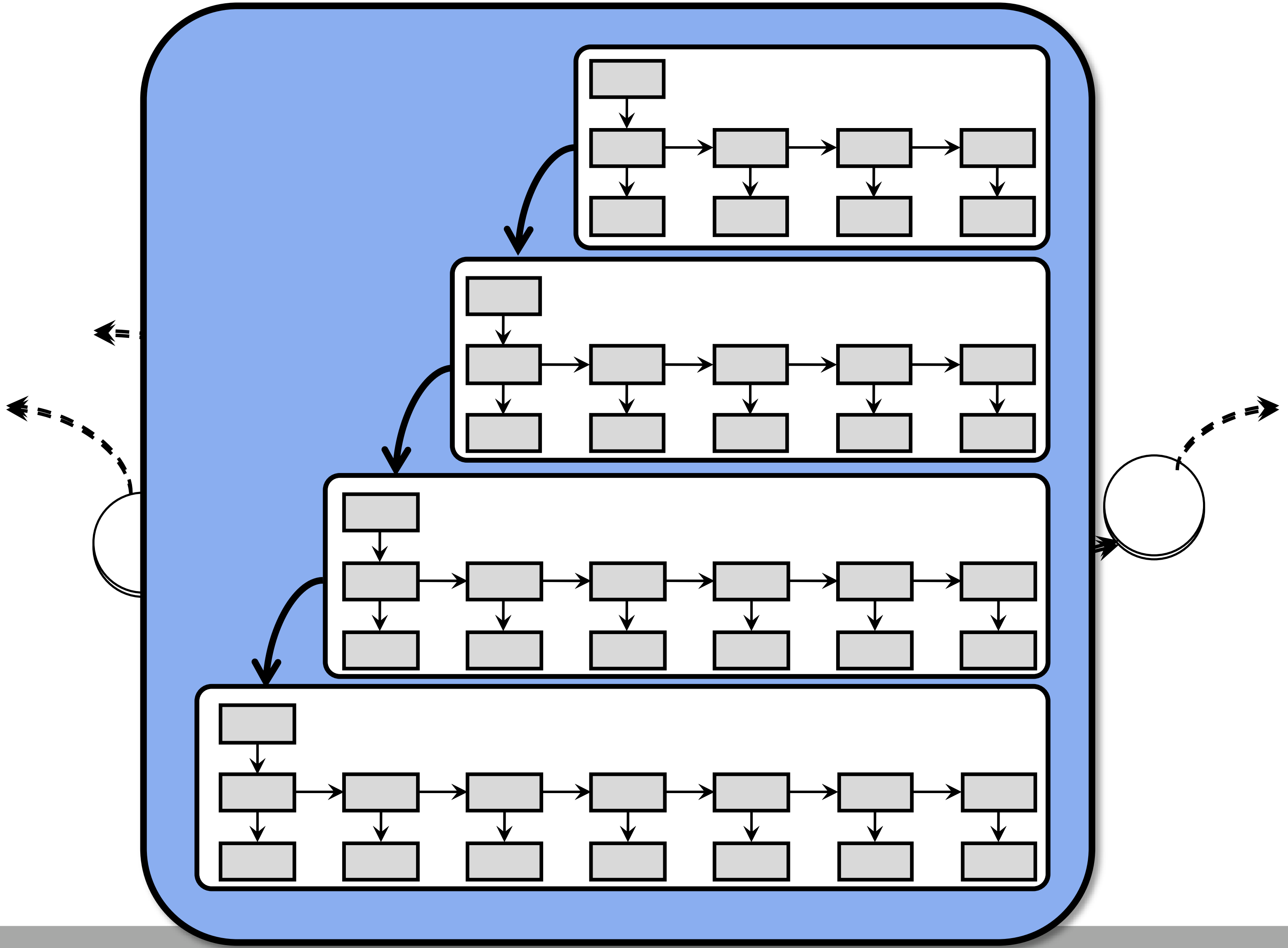
- Semantics of missing vs unfinished

Shaping Approaches

Motivation: Checking graph-based systems

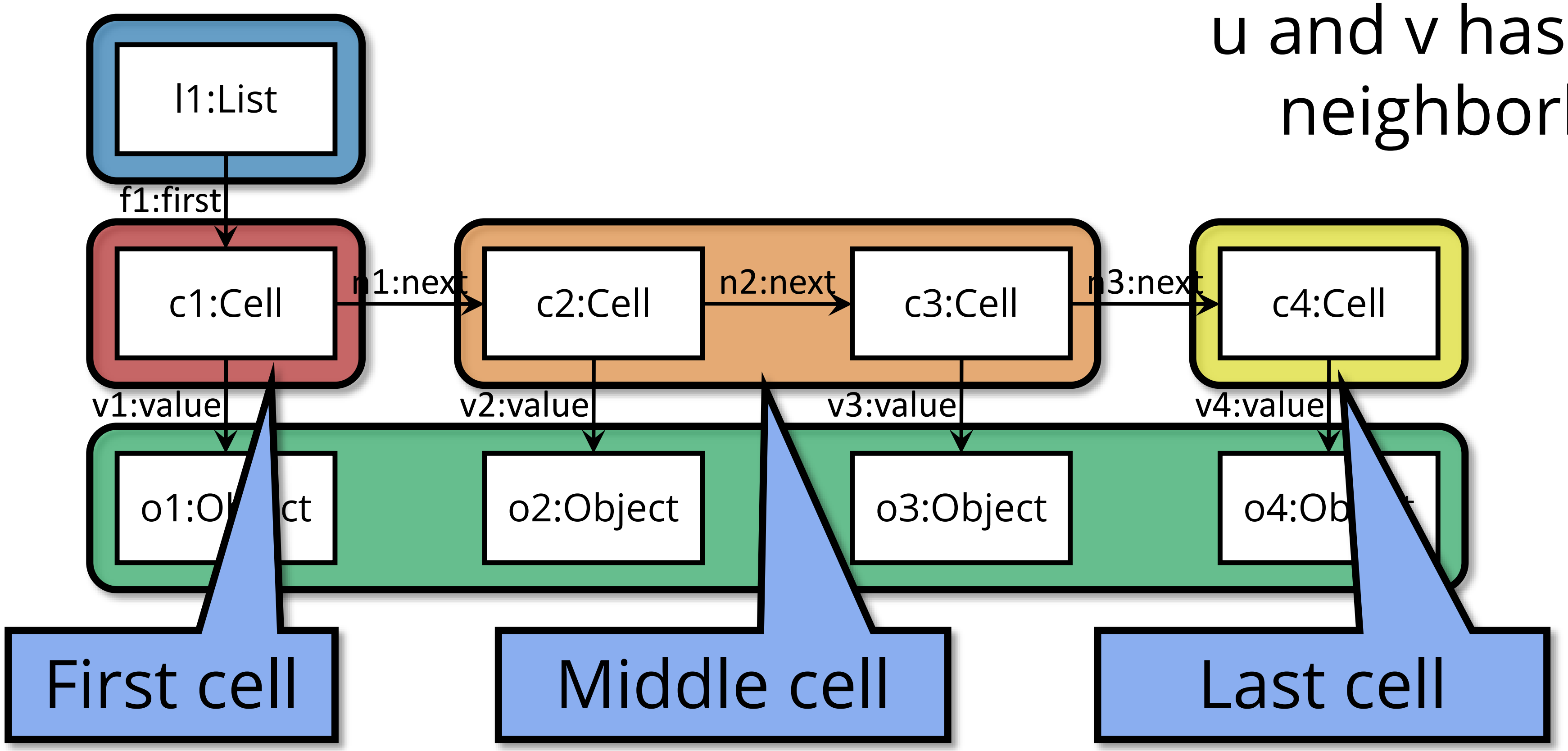


Motivation: Checking graph-based systems



Shaping

- **Goal:** Collect similar graphs together
- Similar graphs behave similarly



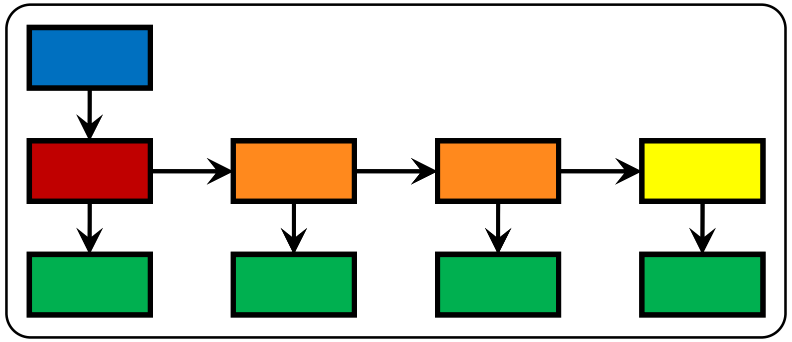
Neighborhood Equivalence:

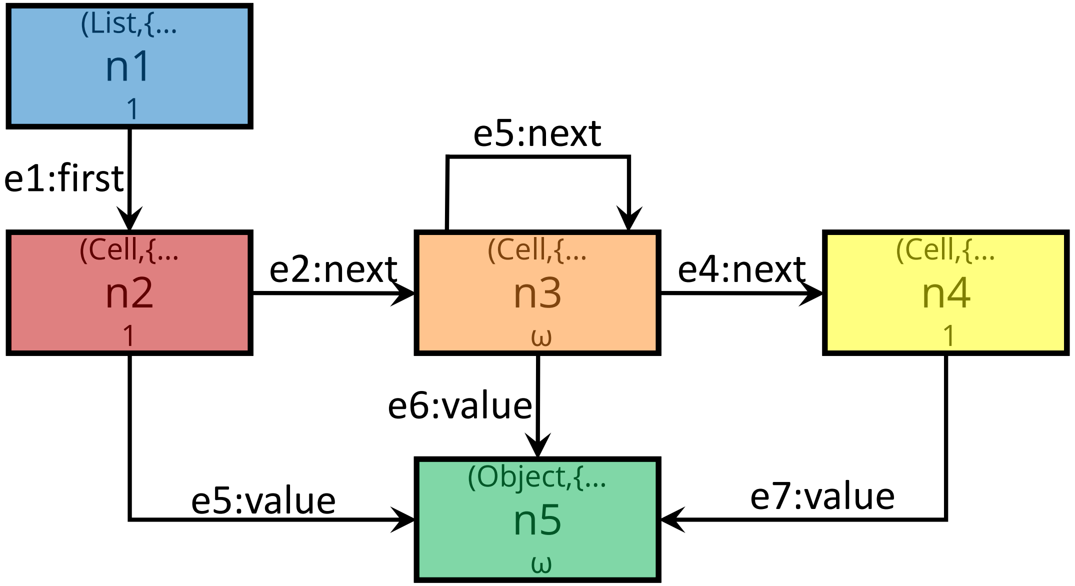
u~v

↔

u and v has similar neighborhood

Finite number of equivalence classes





Size = 4,5,6,...
Same shape

Shaping summary

- Similar graphs collected together
- Uniform analysis of on the representation of similar graphs
- One of the few method to analyze infinite GT systems

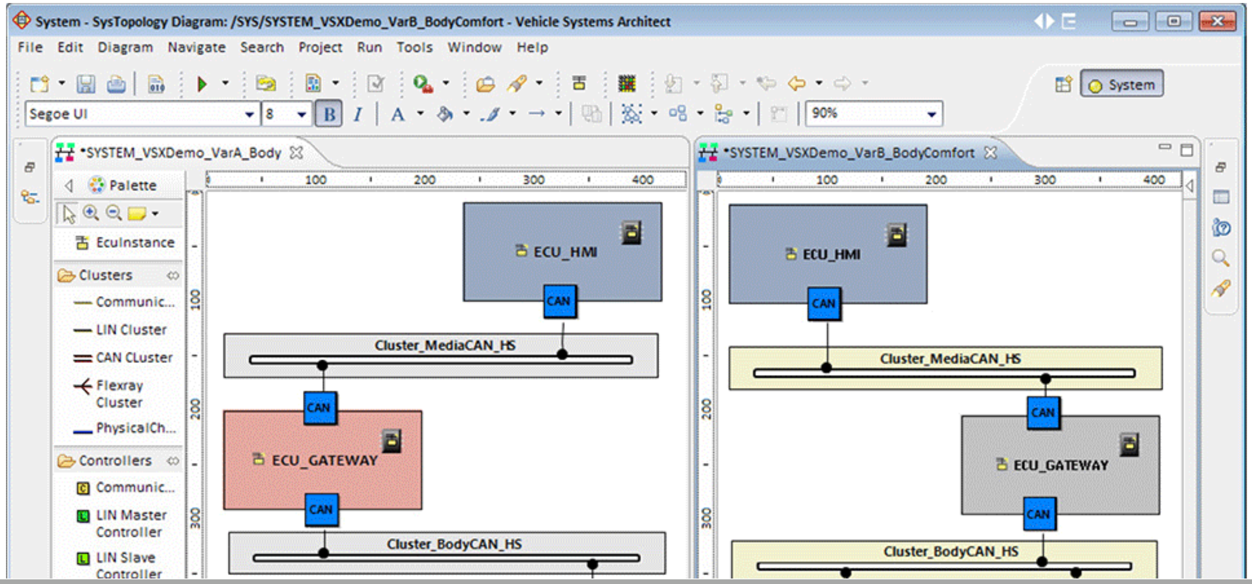
- Model $\sqsubseteq$ Shape $\sqsubseteq$ Metamodel

Model Generation

Motivation: Why to Generate Graph Models?

Tool Qualification

- Design of avionics / automotive systems
- Can you trust the tools?
- Safety standards (DO-178C) require systematic testing with guaranteed coverage



Source: https://www.mentor.com/embedded-software/automotive/autosar

Motivation: Why to Generate Graph Models?

Tool Qualification

- Design of avionics / automotive systems
- Can you trust the tools?
- Safety standards (DO-178C) require systematic testing with guaranteed coverage

Validating Intelligent CPS Components

- AI Safety: How to check components driven by AI?
  - Treat AI component as black box
  - Generate test contexts
  - E.g. ICSE'18 paper from L. Briand's groups

https://medium.com/self-driving-cars/beginners-guide-to-self-driving-vehicles-9e9003e790b8

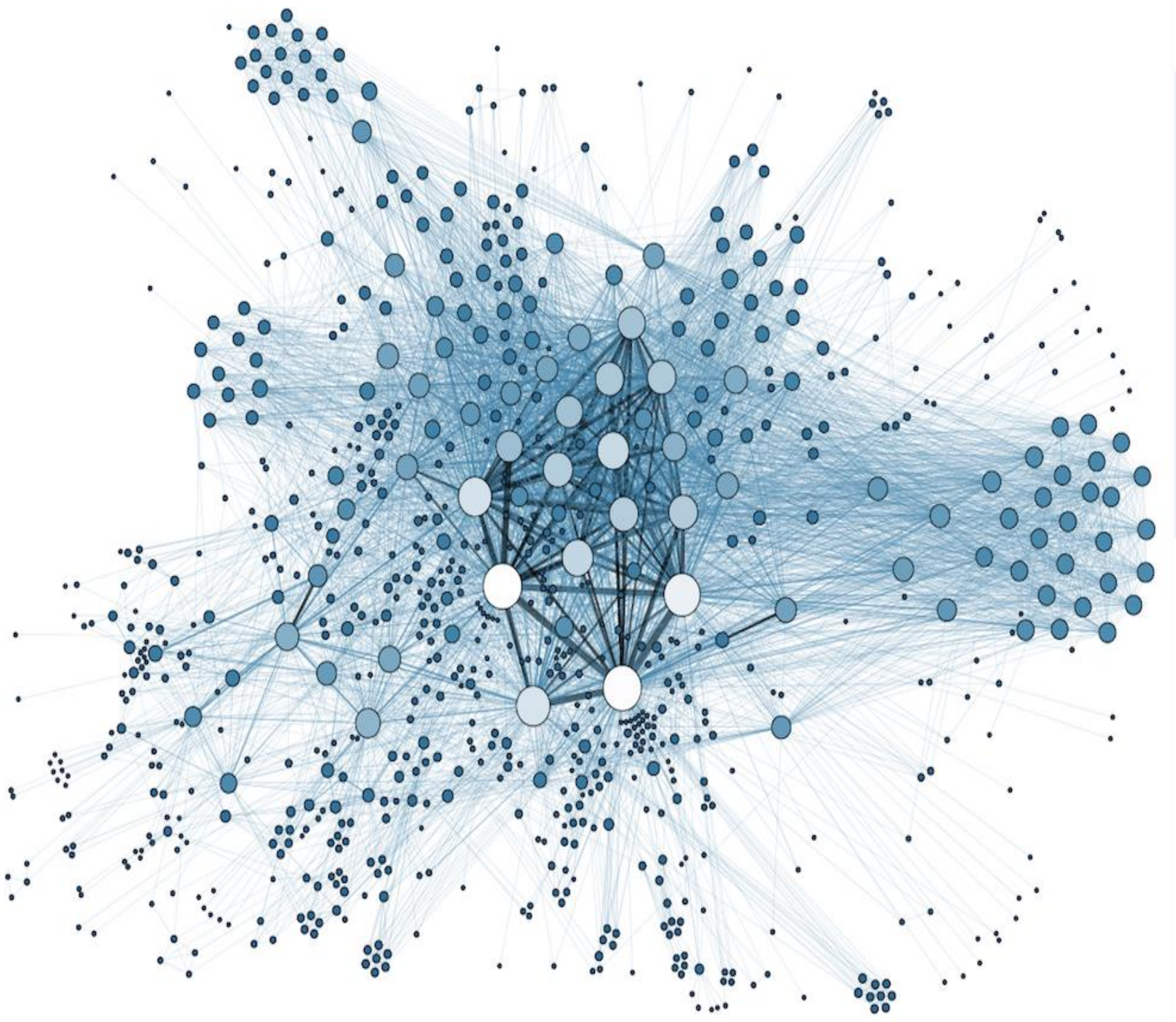
Motivation: Why to Generate Graph Models?

Tool Qualification

- Design of avionics / automotive systems
- Can you trust the tools?
- Safety standards (DO-178C) require systematic testing with guaranteed coverage

Graph Database Benchmarks

- How to check that algorithms for graph DBs will behave well for real data?
- Real data has IP restrictions and never shown
- Real data is well-formed – random data is not



Source: https://neo4j.com/blog/other-graph-database-technologies/

Motivation: Why to Generate Graph Models?

Tool Qualification

- Design of avionics / automotive systems
- Can you trust the tools?
- Safety standards (DO-178C) require systematic testing with guaranteed coverage

Validating Intelligent CPS Components

- AI Safety: How to check components driven by AI?
  - Treat AI component as black box
  - Generate test contexts
  - E.g. ICSE’18 paper from L. Briand’s groups

https://modeling-languages.com/presence-of-modelig-papers-at-icse-is-purely-anecdotal-once-again/

Empirical Evaluation of Modeling Papers

- Real models are either confidential or too small
- Is your case study relevant & scalable?
- Existing generators are ad hoc and domain-specific

Motivation: Why to Generate Graph Models?

Tool Qualification

- Design of avionics / automotive systems
- Can you trust the tools?
- Safety standards (DO-178C) require systematic testing with guaranteed coverage

Validating Intelligent CPS Components

- AI Safety: How to check components driven by AI?
  - Treat AI component as black box
  - Generate test contexts
  - E.g. ICSE'18 paper from L. Briand's groups

Graph Model Generator

Graph Database Benchmarks

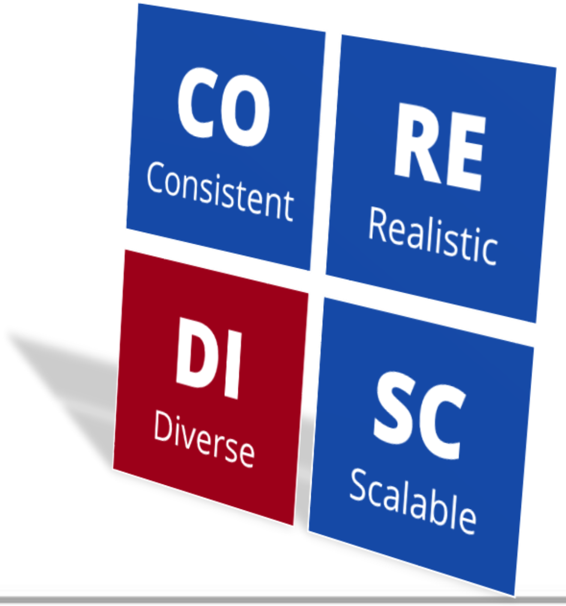
- How to check that algorithms for graph DBs will behave well for real data?
- Real data has IP restrictions and never shown
- Real data is well-formed – random data is not

Empirical Evaluation of Modeling Papers

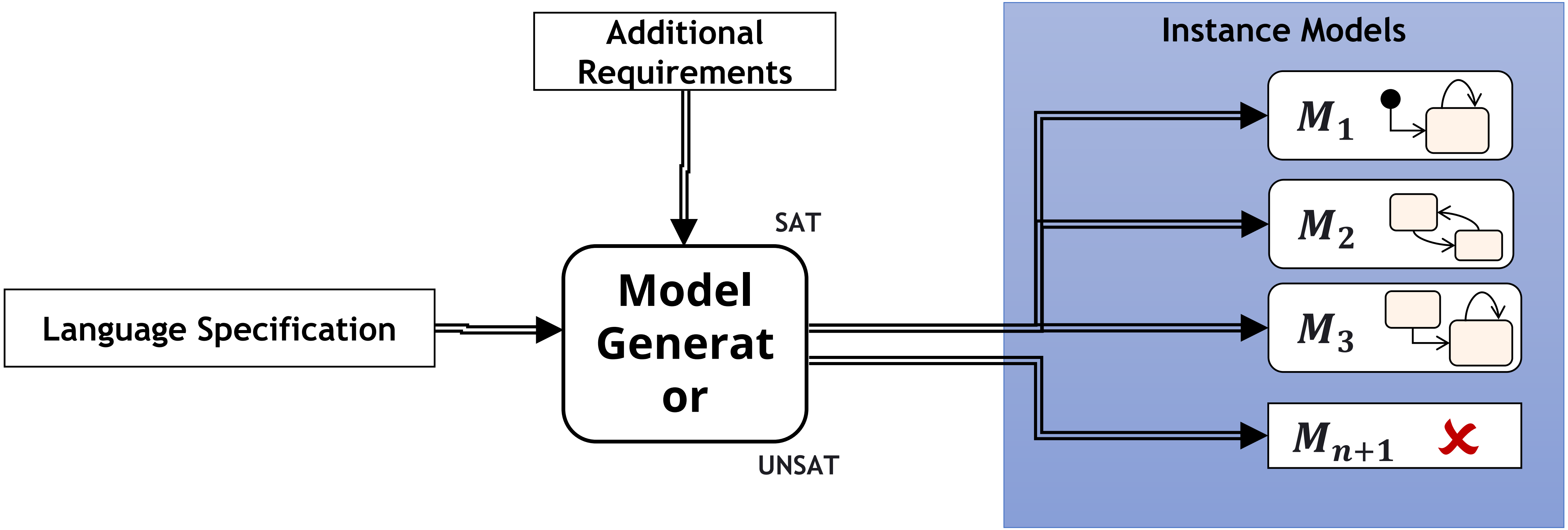
- Real models are either confidential or too small
- Is your case study relevant & scalable?
- Existing generators are ad hoc and domain-specific

Requirements & Objectives

the CoReDiSc model



Setup of an ideal model generator



Properties of Model Generators: Consistent

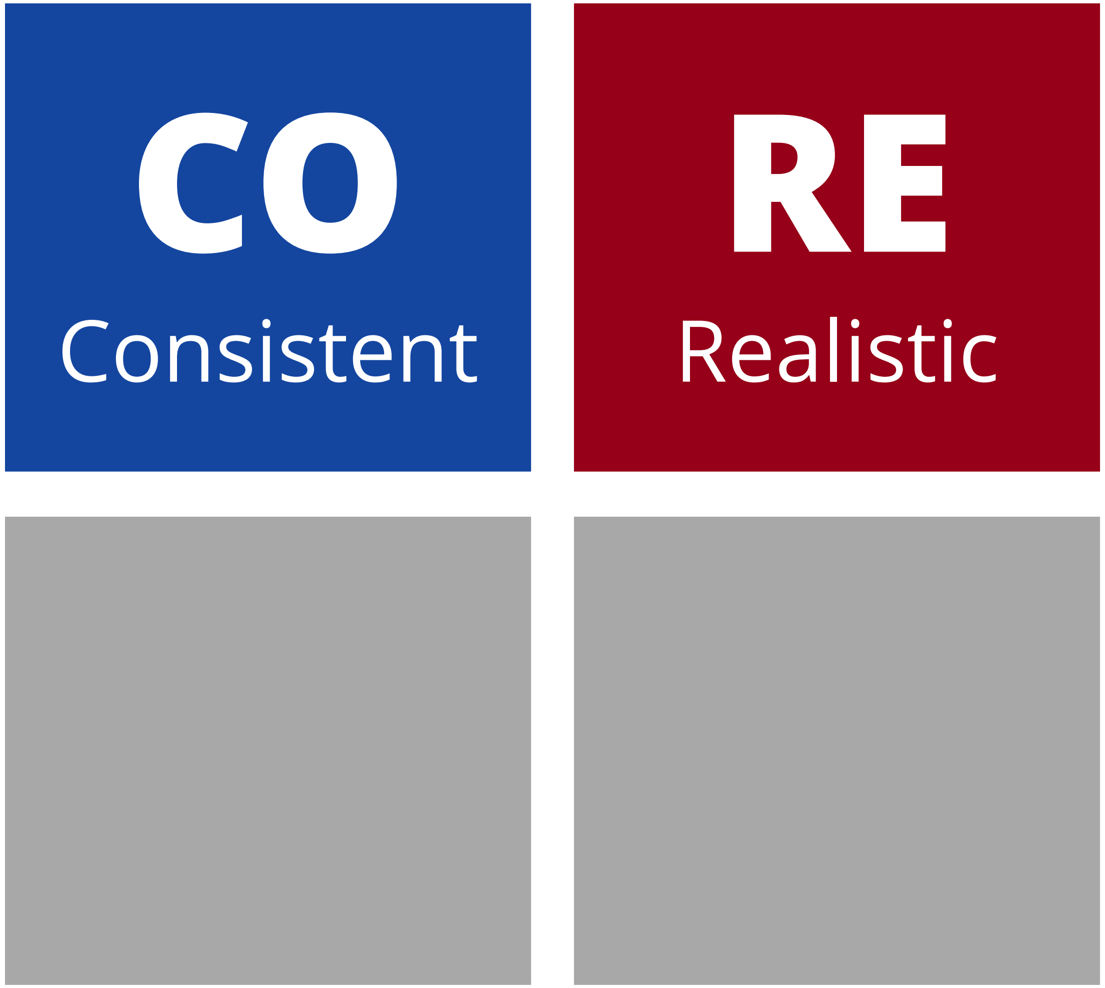
- All (well-formedness) constraints are satisfied
- All (and only) consistent models are derived

CO

Consistent

Properties of Model Generators: Realistic

- Cannot be distinguished from a real model (By removing text+values and evaluating graph metrics)
- Set of generated models is close to real ones



Properties of Model Generators: Diverse

- Models are not symmetric

- The distance between any pairs of models is large

  E.g. all equivalence classes are covered

CO Consistent

RE Realistic

DI Diverse

Diversity as a requirement for testing

- Test case diversity
  - Test selection: similar test cases find similar errors
  - Test coverage: similar test cases cover the same code
- Methodologies
  - Equivalence partitioning, boundary-value analysis, etc.
  - Rely on *similarity*, *difference*, *distance*

  →Straightforward for simple structures, eg. Numbers

- What about models?

How to measure model **diversity**?

Diversity as a requirement for testing

- Test case diversity
  - Test selection: similar test cases find similar errors
  - Test coverage: similar test cases cover the same code
- Methodologies
  - Equivalence partitioning, boundary-value analysis, etc.
  - Rely on *similarity*, *difference*, *distance*

  →Straightforward for simple structures, eg. Numbers

- What about models?

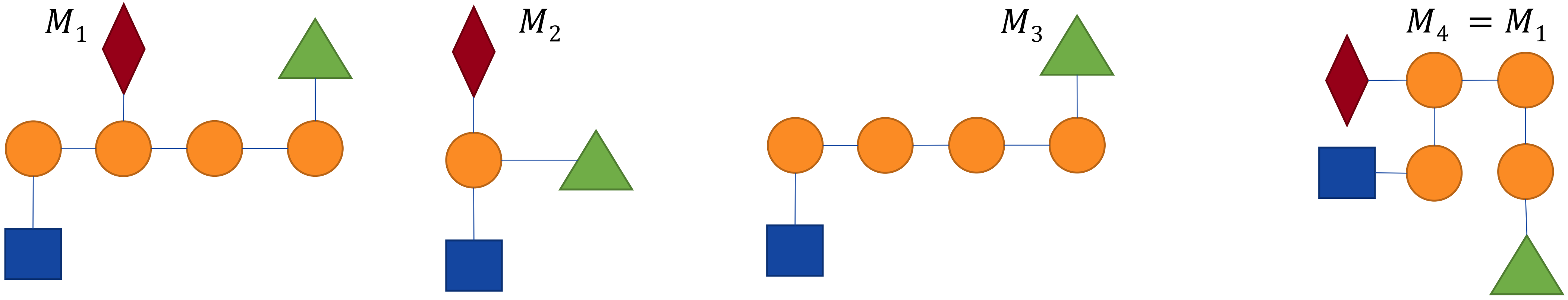
How to measure model **diversity**?

Properties of Model Generators: Scalable

- In size: ability to generate huge graphs
- In quantity: generation time of next model does not grow

CO Consistent	**RE** Realistic
DI Diverse	**SC** Scalable

Model similarity and diversity



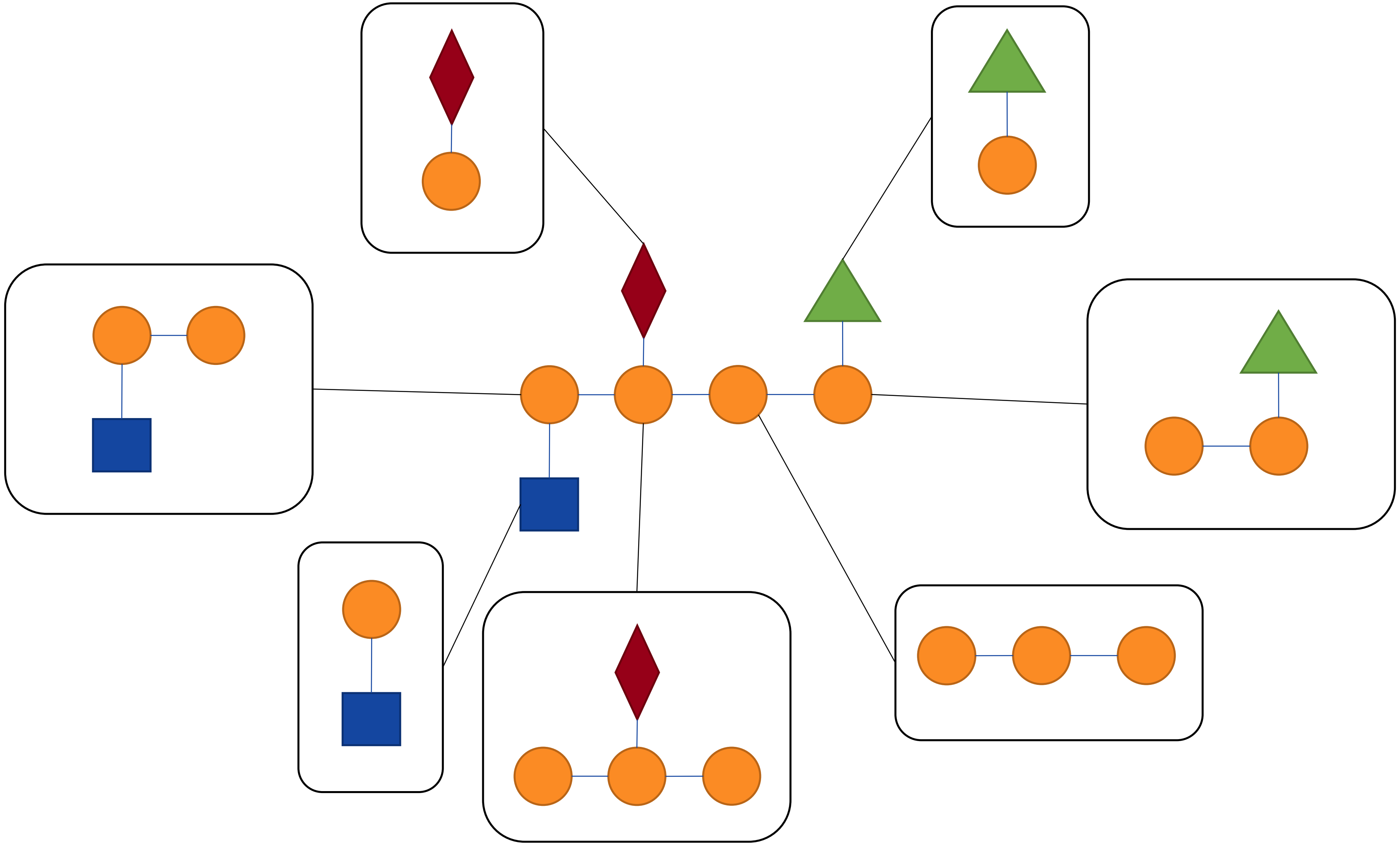
Which of these graphs…

- …are similar?
- …are equivalent?
- …should be selected as test cases?

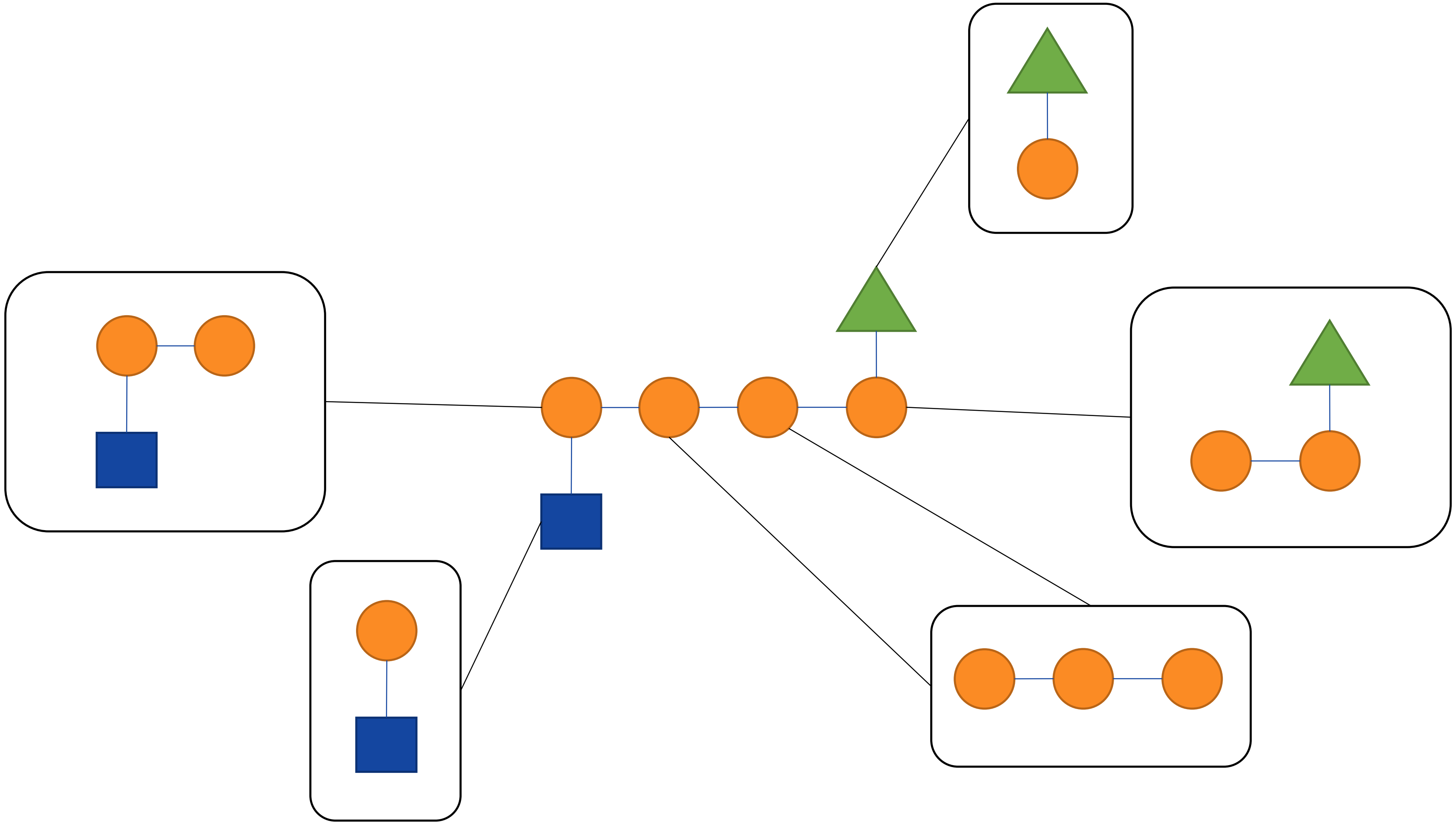
How to automate the process?

Solution: neighbourhood shapes

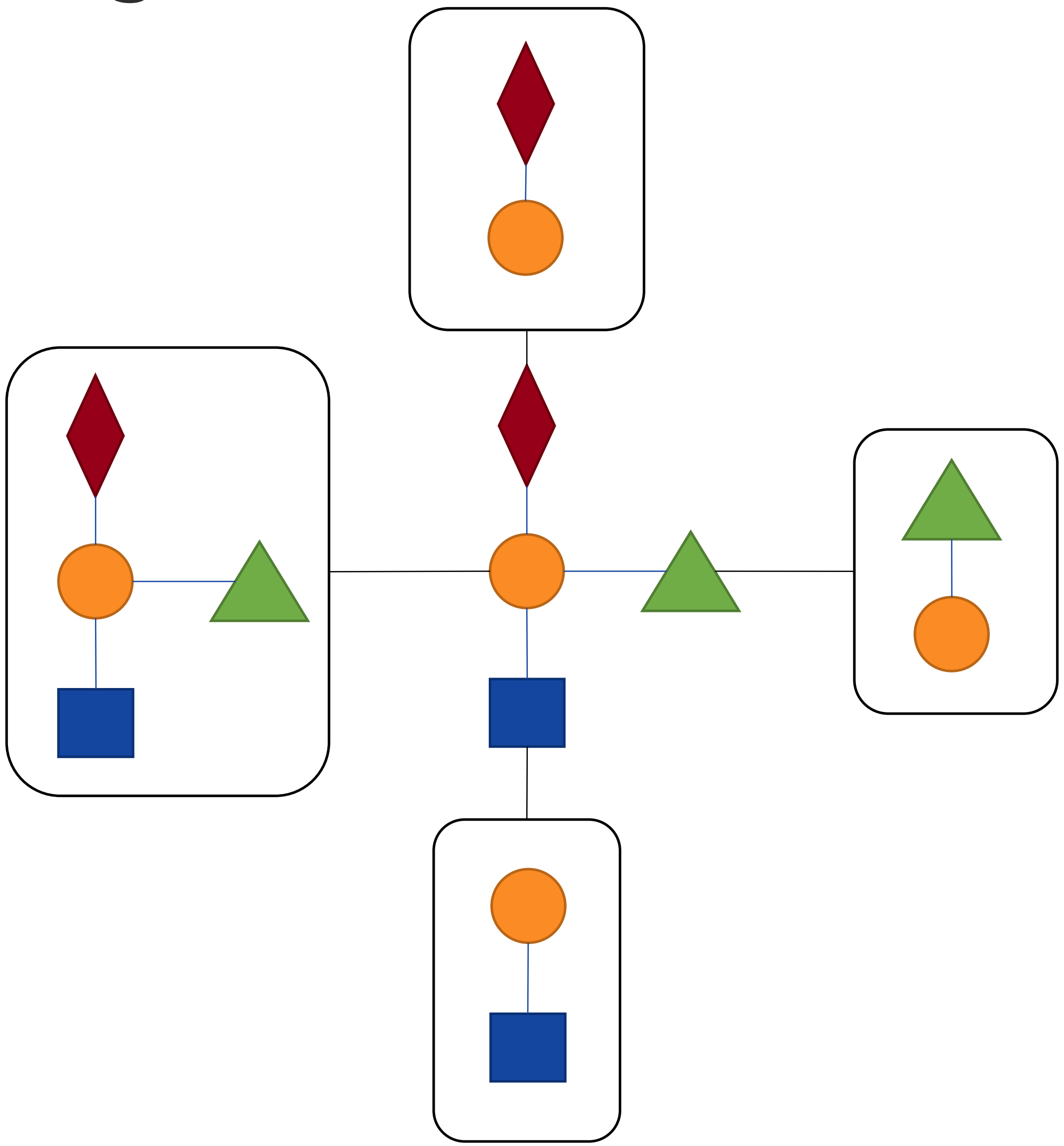
Shapes in M_1



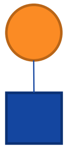
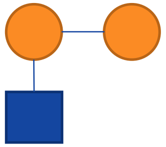
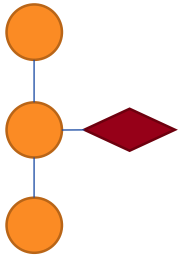
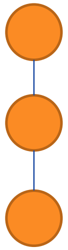
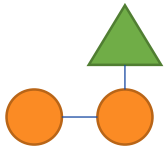
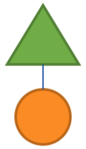
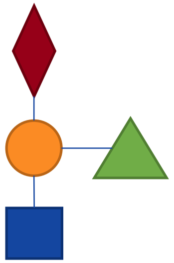
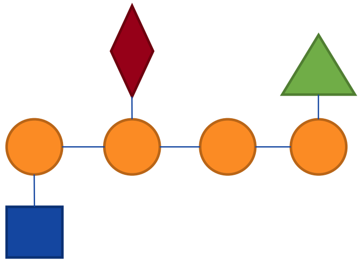
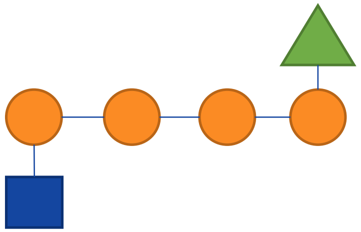
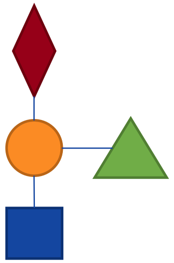
Shapes in M_2



Shapes in M_3

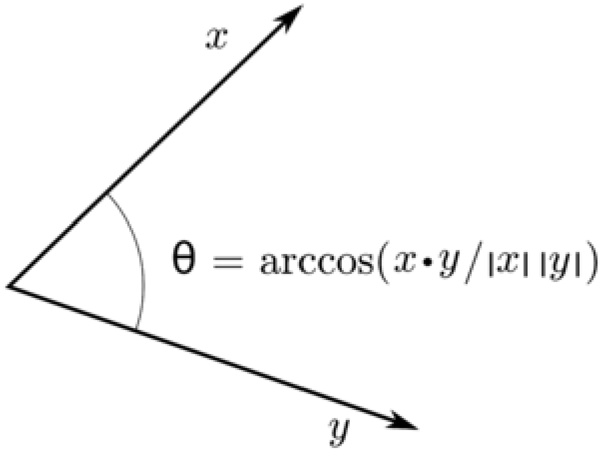


Similarity basis: Shape vectors

									vector
	1	1	1	1	1	1	1	0	11111110
	1	1	0	0	2	1	1	0	11002110
	1	0	0	1	0	0	1	1	10010011

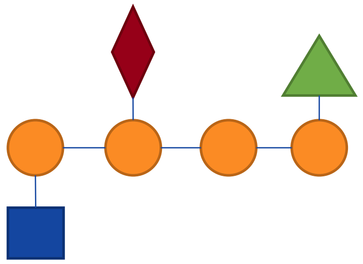
Cosine similarity

- Well known similarity measure between vectors
- The cosine of the enclosed angle
  - Nonnegative vectors → $\theta < 90°$
  - Smaller the angle larger the *cosine similarity*
- Computed from euclidean product

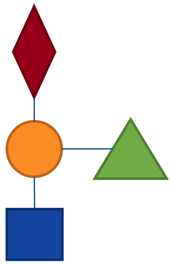


- Advantages:
  - Not affected by model size (as opposed to vector difference)
  - Sensitive to neighbourhood distribution

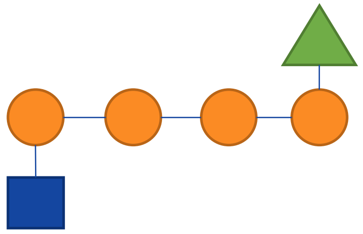
Cosine similarity of models



$v(M_1)=(1,1,1,1,1,1,1,0)$



$v(M_2)=(1,0,0,1,0,0,1,1)$



$v(M_3)=(1,1,0,0,2,1,1,0)$

- Applying cosine similarity to shape vectors
  - M_1 vs M_2: 0.567 – *different*
  - M_1 vs M_3: 0.802 – *similar*
  - M_2 vs M_3: 0.353 – *very different*

→First two models are similar

If M_1 is *real*, M_2 is *realistic*, M_3 is not.

Model diversity basis: neighbourhood sets

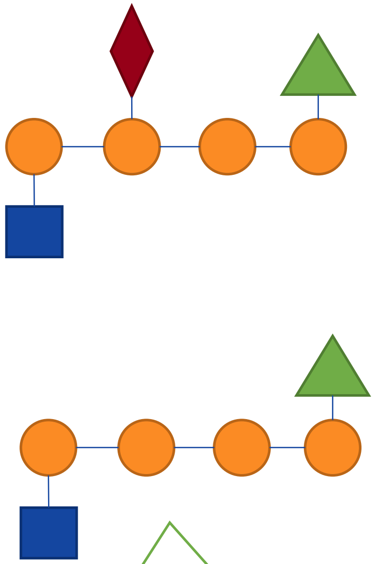
- Idea: Similar neighbourhoods cover similar parts of code

→Goal: differentiate between models by neighbourhoods

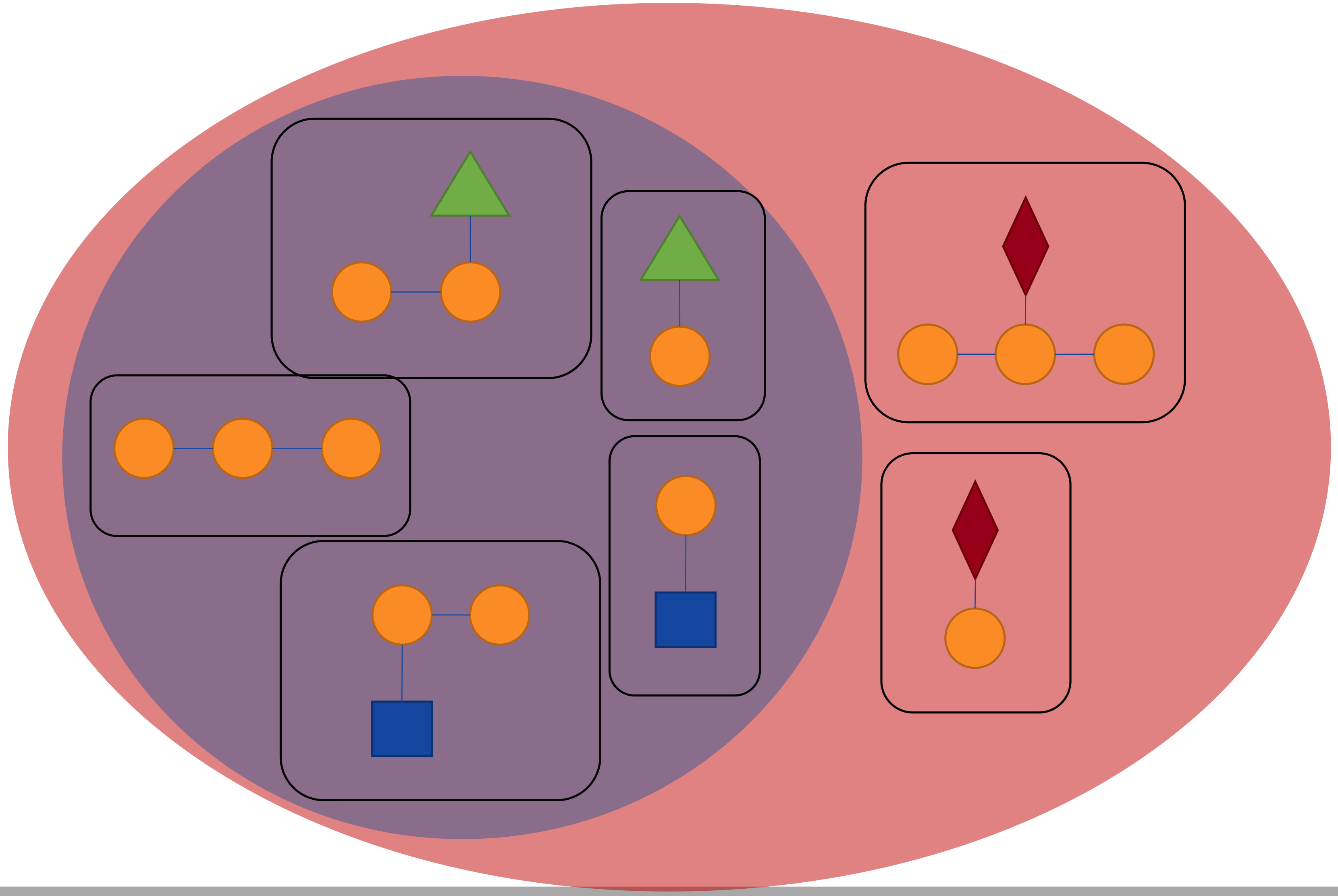
Advantage: not affected by shape distribution, model size

- *External diversity*: How big is the difference between two models?
  - Selection of diverse test cases
  - Symmetric difference of model neighbourhood sets
- *Internal diversity*: How effective is a model?
  - Test coverage vs input size
  - #neighbourhoods/#objects

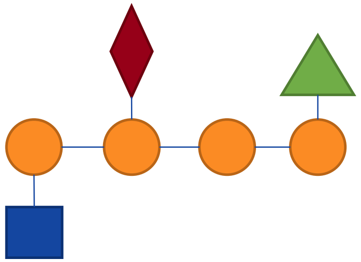
Shape difference: M_1 vs M_2

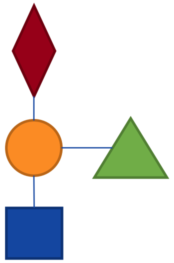


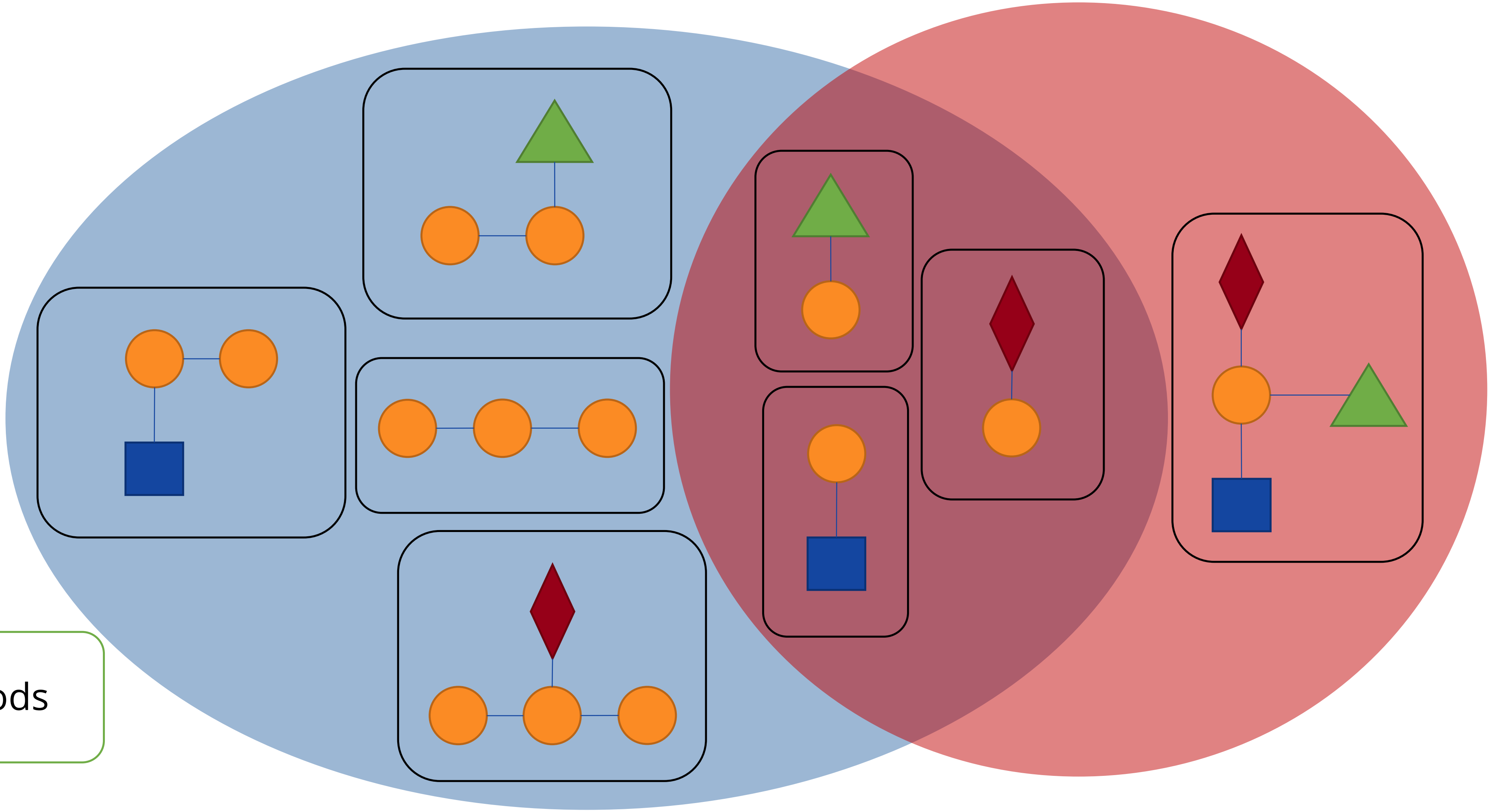
Similar:
5 neighbourhoods
in common



Shape difference: M_1 vs M_3







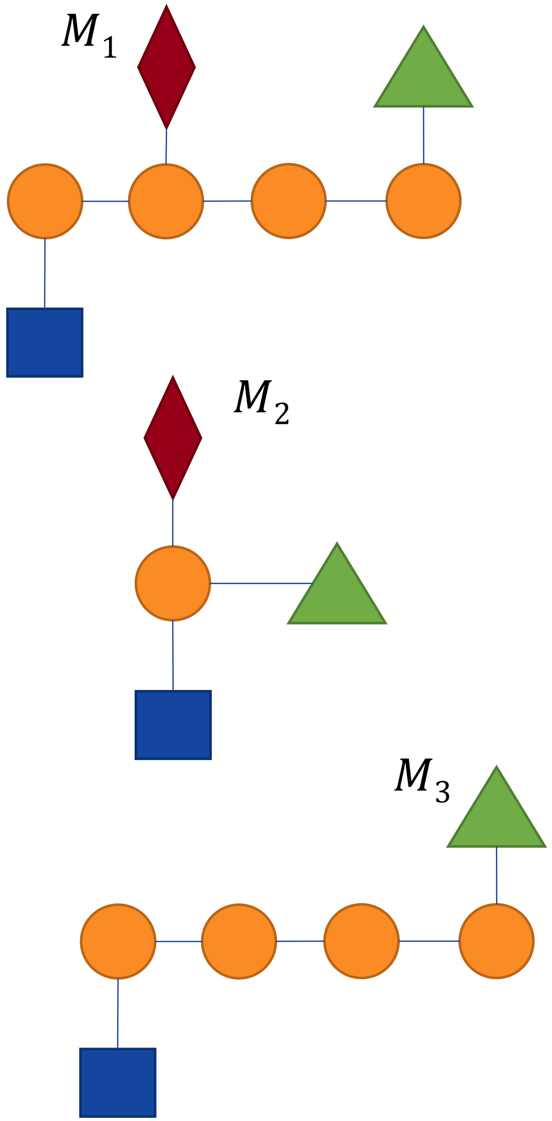
Different:
3 neighbourhoods
in common

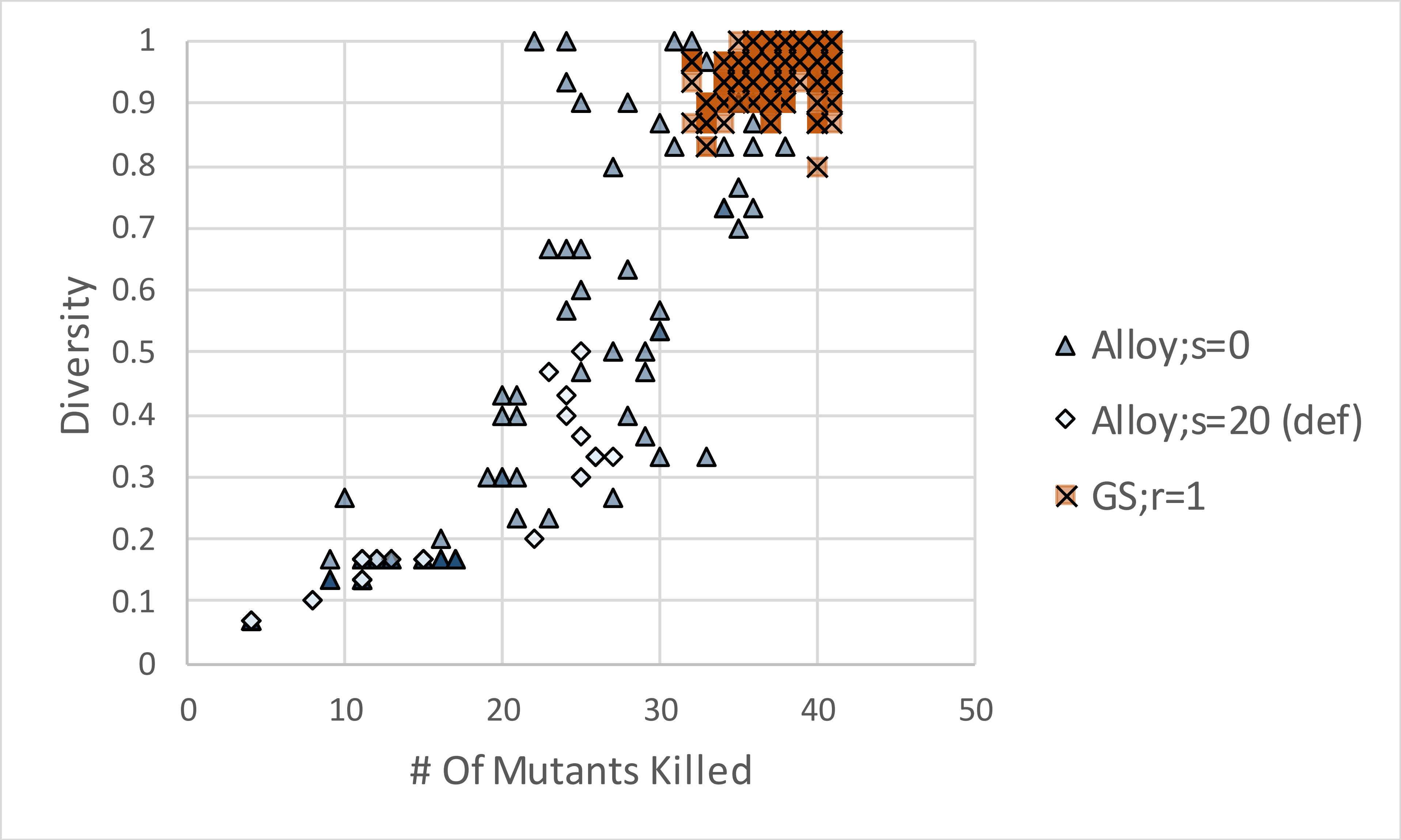
Internal and external model diversity

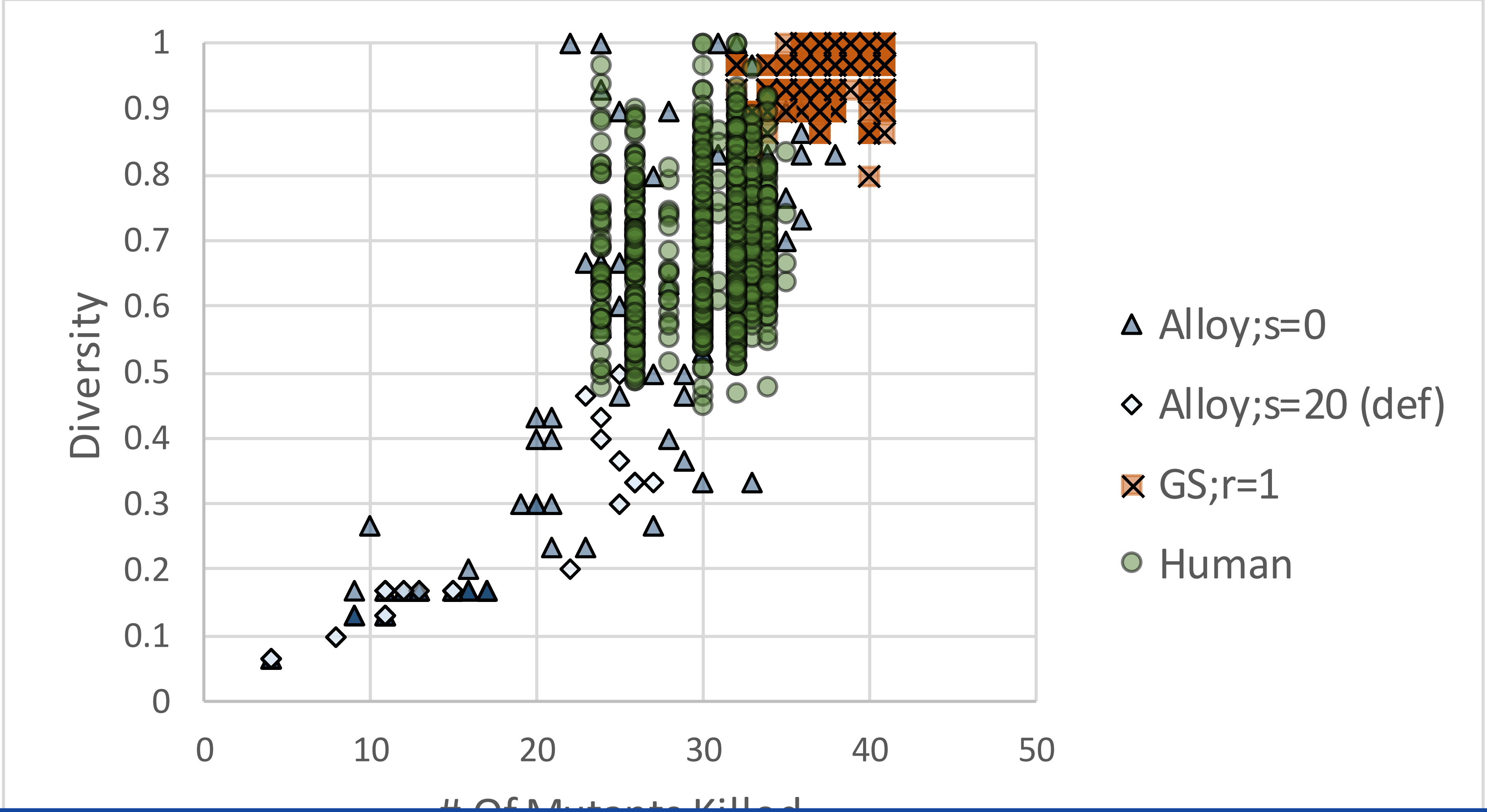
- Internal diversity:
  - M_1: 7/7=1
  - M_2: 4/4=1
  - M_3: 5/6=0.8333

- External diversity:
  - M_1 vs M_2: 5 neighbourhoods only in one model – *different*
  - M_1 vs M_3: 2 neighbourhoods only in one model – *similar*
  - M_2 vs M_3: 5 neighbourhoods only in one model – *different*

→ M_1 and M_2 should be the selected test cases




Diversity
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
10
20
30
40
50
Of Mutants Killed
Alloy;s=0
Alloy;s=20 (def)
GS;r=1



Alloy (def) < Alloy (s=0) < Human < GS

Correlation between **Diversity** and **Mutation Score** in Alloy+GS+Human

Properties of Model Generators: Scalable

- In size: ability to generate huge graphs
- In quantity: generation time of next model does not grow

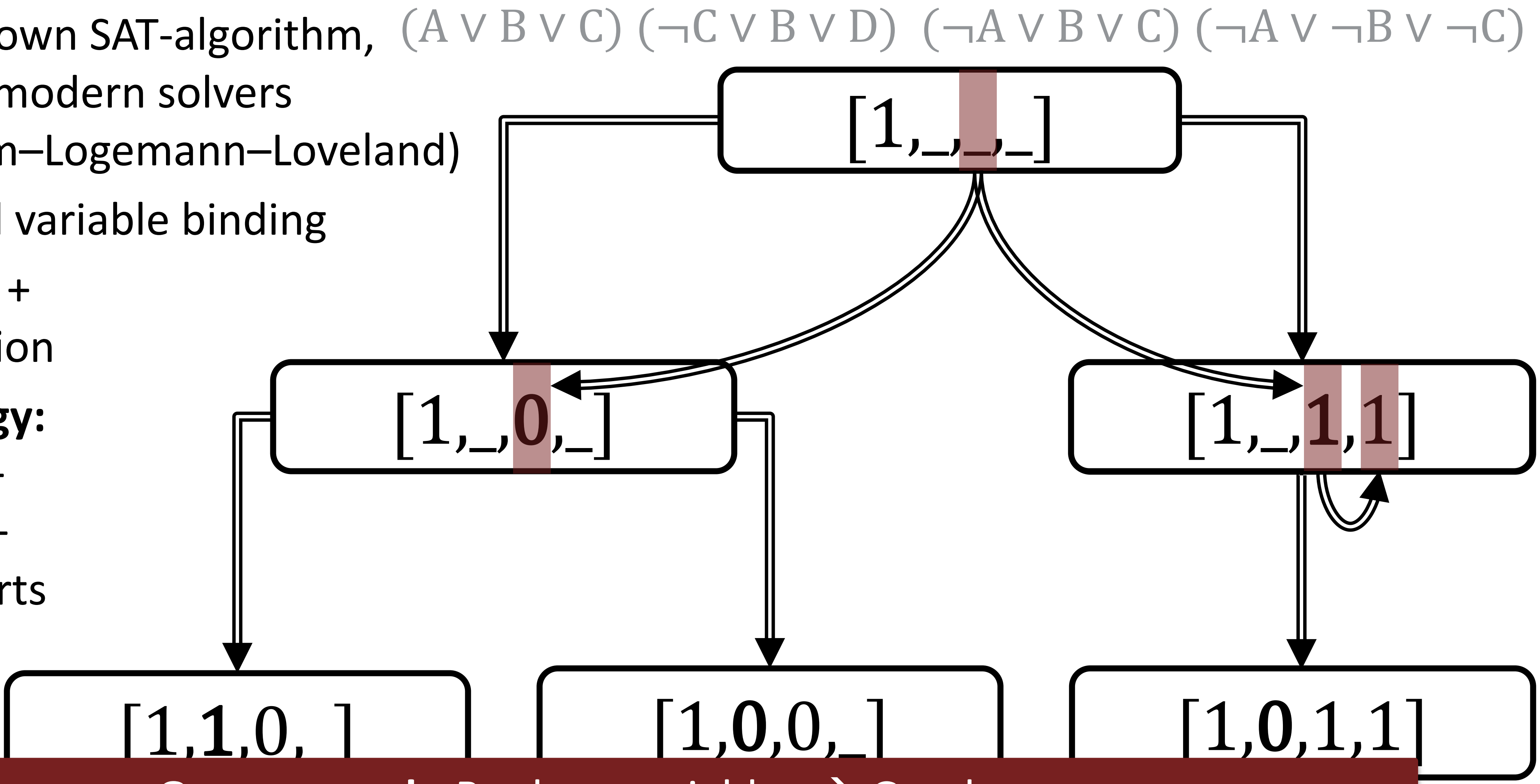
CO Consistent	**RE** Realistic
DI Diverse	**SC** Scalable

Partial Modeling for Model Generation

3-Valued Models

SAT Solver Overview: DPLL Algorithm

- **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

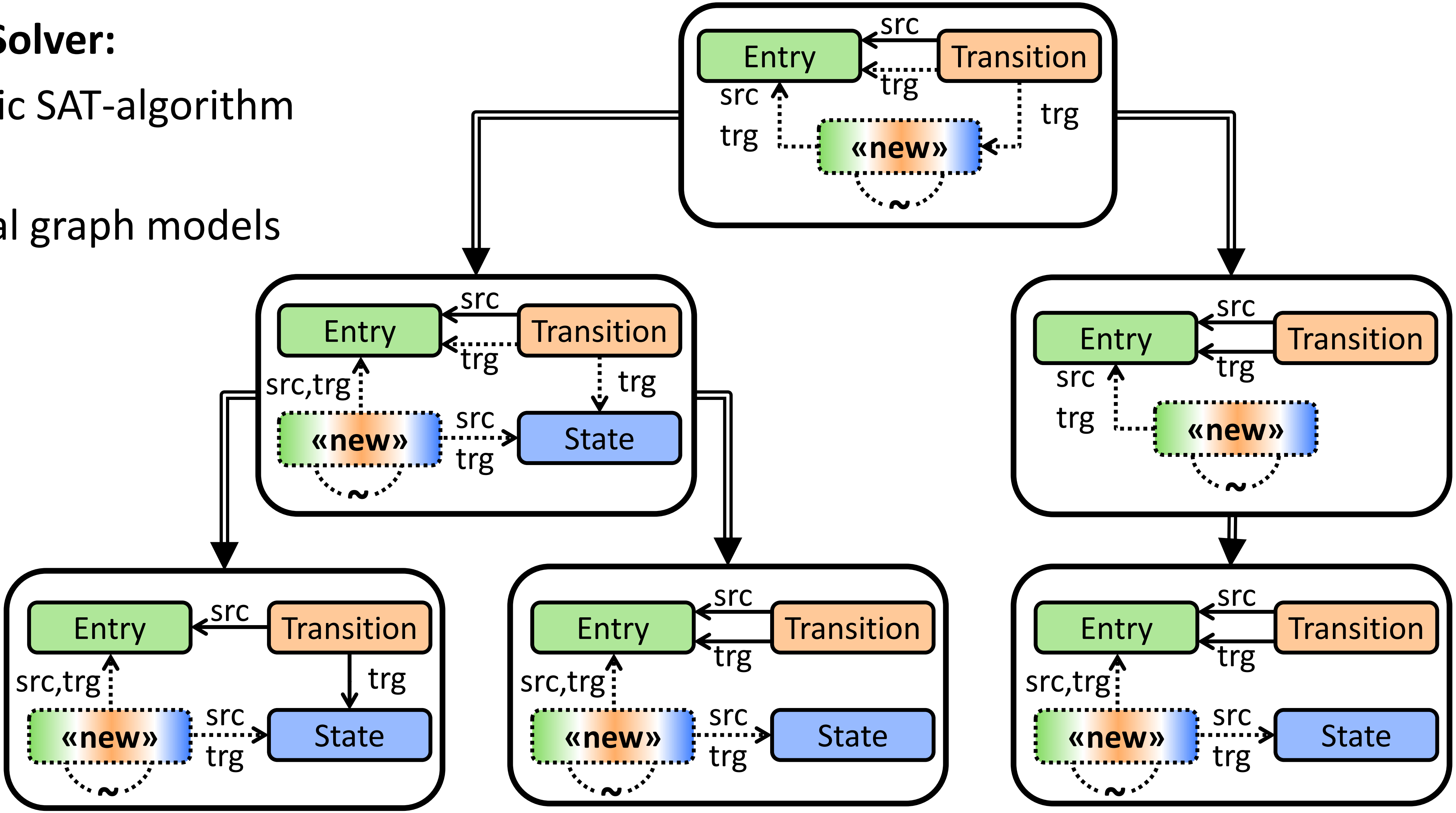


Our approach: Boolean variables → Graphs

Graph Solver Overview: 3-Valued Partial Models as States

Graph Solver:

1) Based on classic SAT-algorithm
2) Refinement of 3-valued partial graph models

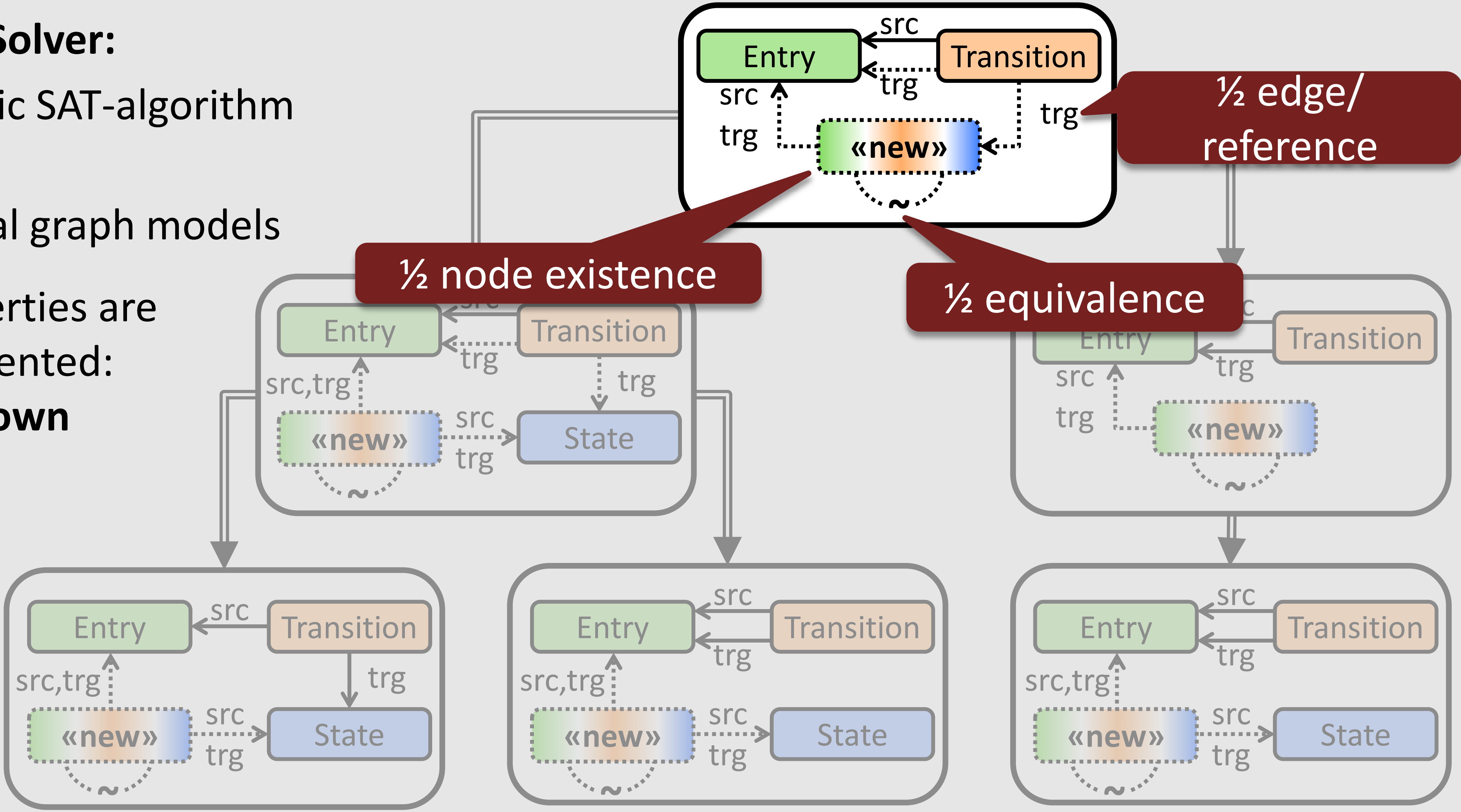


Graph Solver Overview: 3-Valued Partial Models as States

Graph Solver:

1) Based on classic SAT-algorithm
2) Refinement of 3-valued partial graph models

- Uncertain properties are explicitly represented: 1 | 0 | ½: **Unknown**

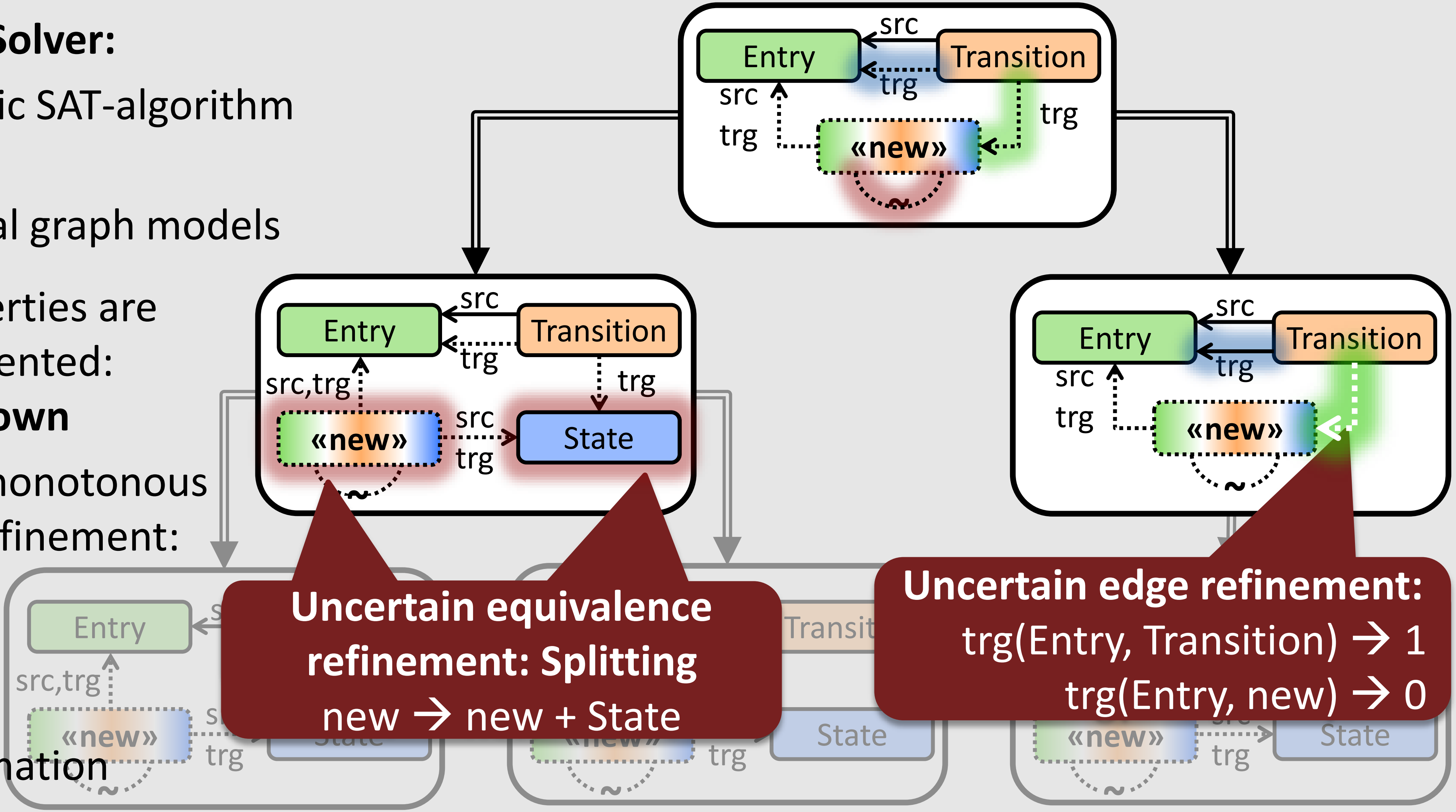


Graph Solver Overview: Partial Model Refinement

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

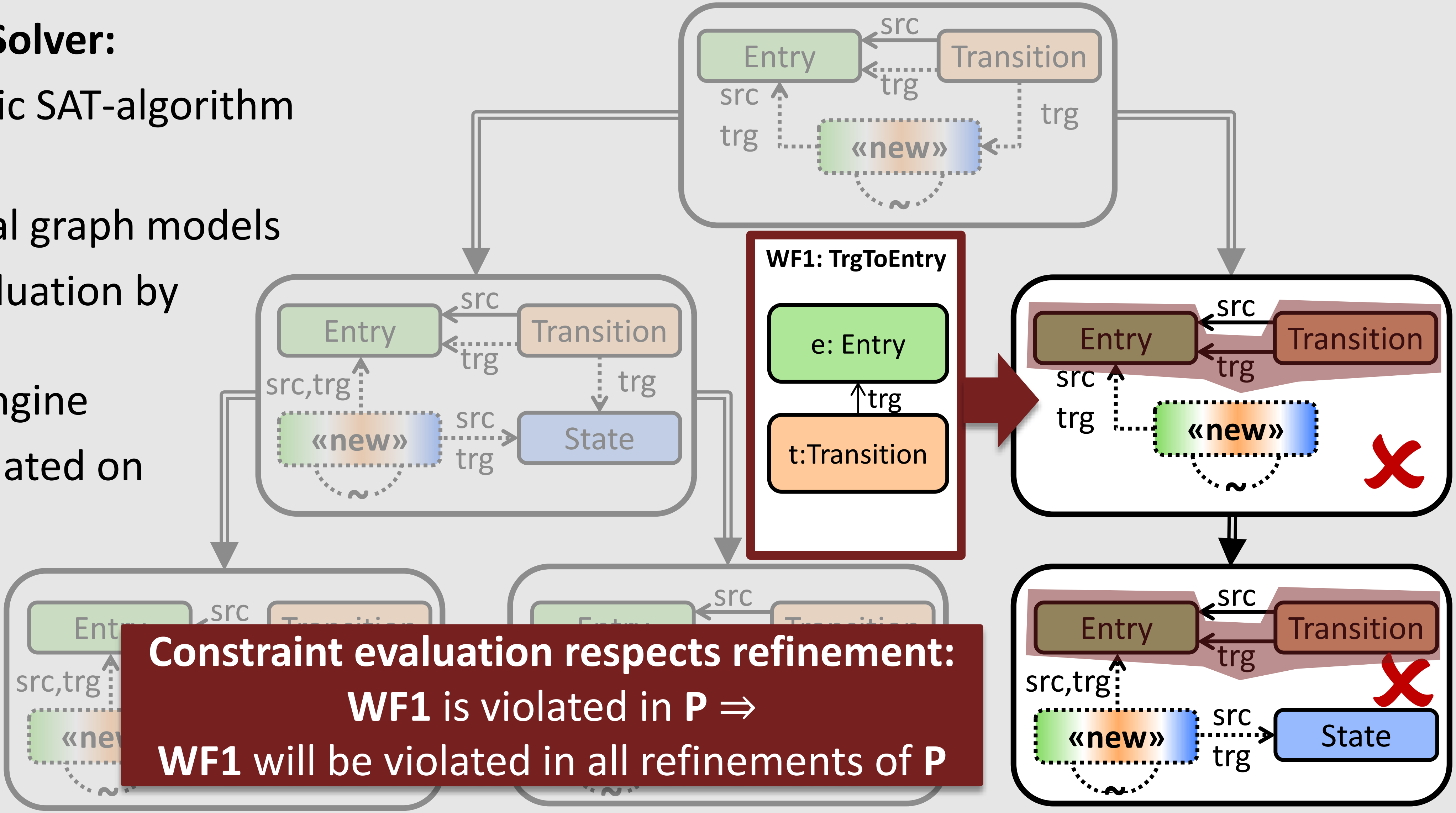


Graph Solver: Approximated Constraint Evaluation

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



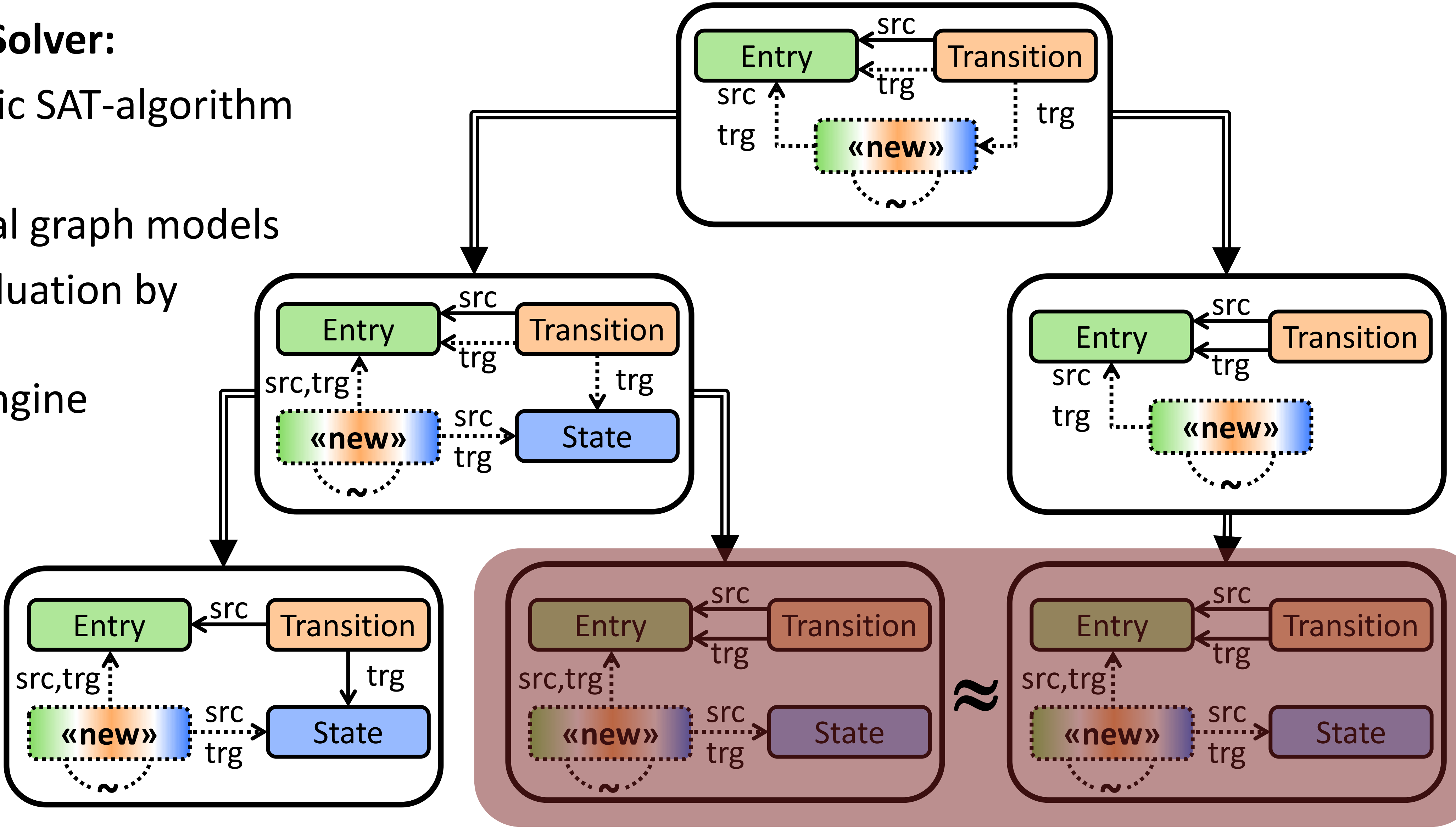
Constraint evaluation respects refinement:
WF1 is violated in **P** ⇒
WF1 will be violated in all refinements of **P**

Graph Solver Overview: 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

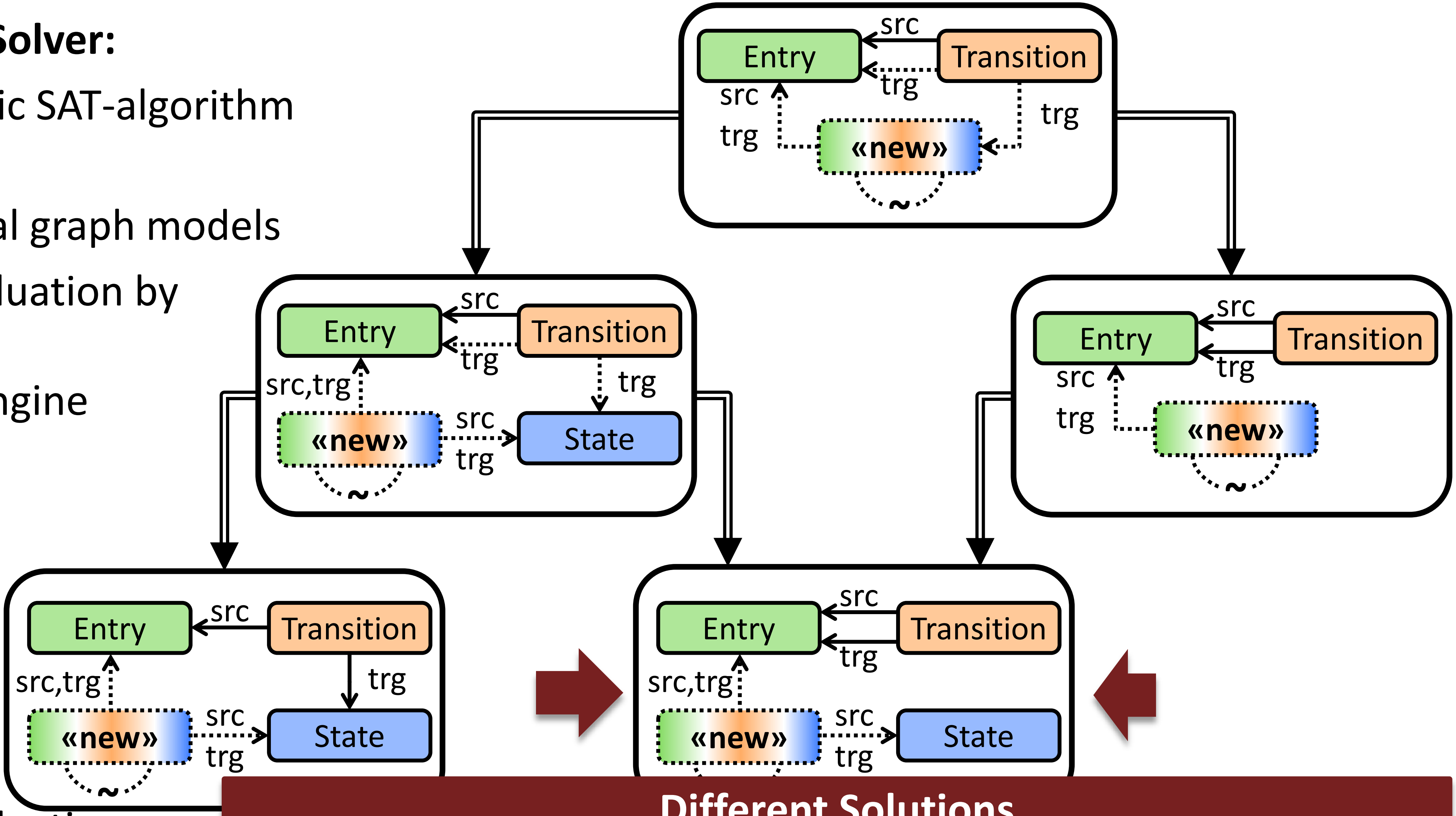


Graph Solver Overview: Equivalence Partitioning

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

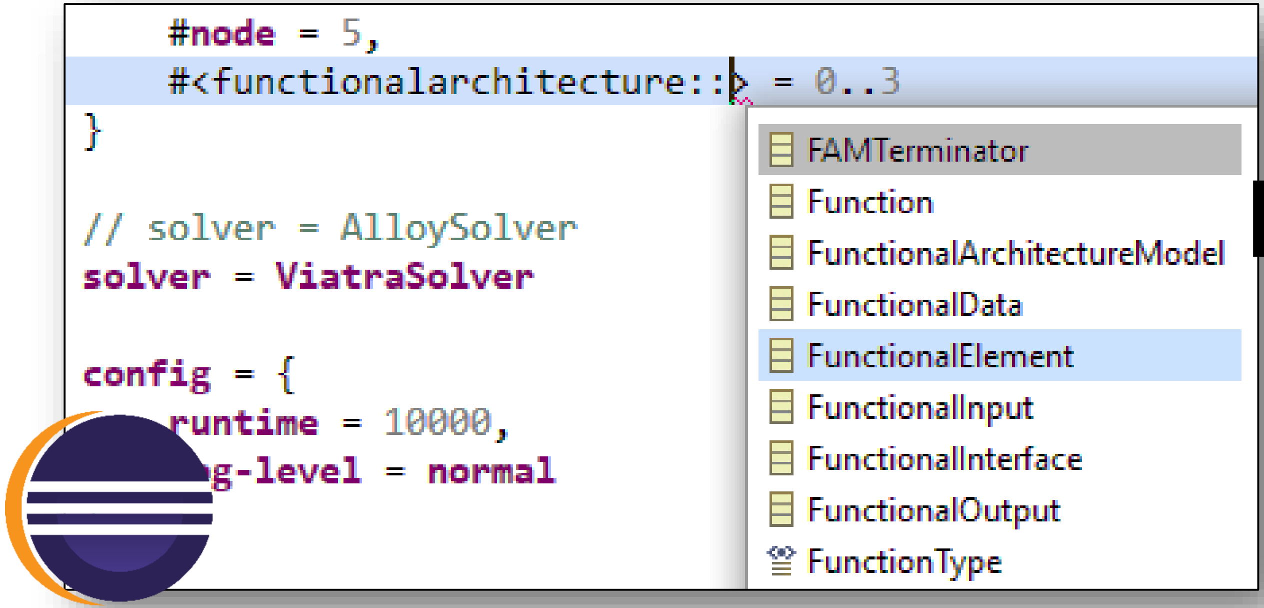
- State encoding
- Partial order reduction

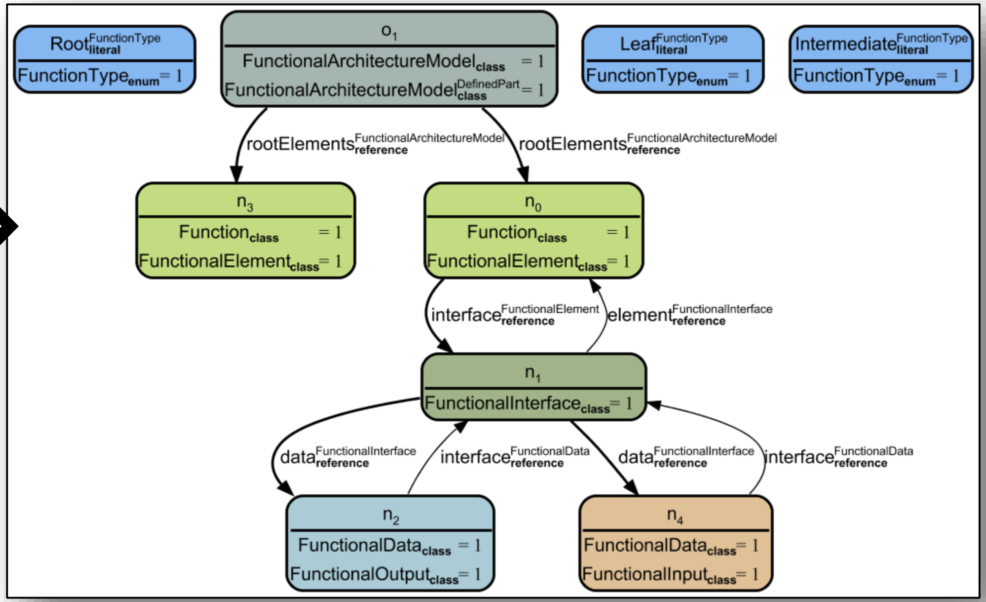


Different Solutions

VIATRA Solver: An Open Source Implementation

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





- Incremental Query Engine: VIATRA
  - Constraint language: VIATRA Query
  - Internally uses: Incremental constraint reevaluation, DPLL as VIATRA DSE
- Open source: 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

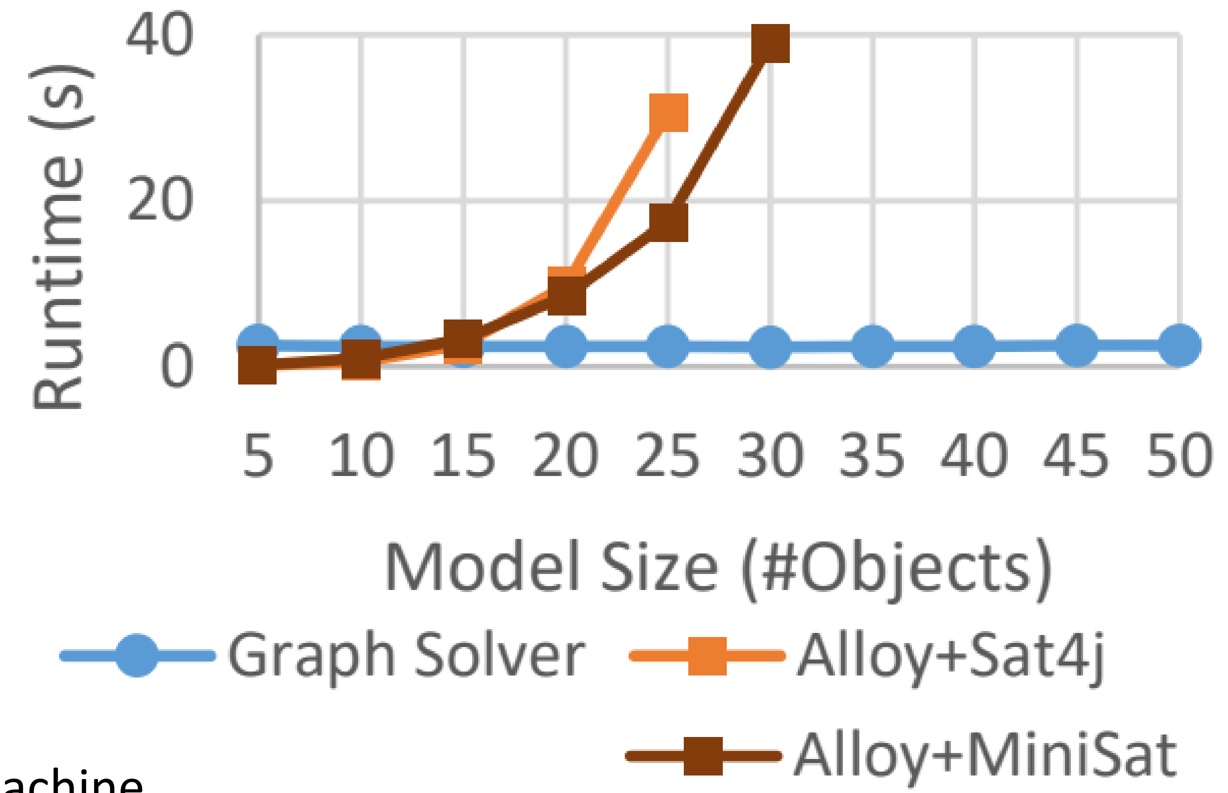
Yakindu: Industrial, Statemachine

FS: File System example of Alloy

Ecore: Metamodelling language

5 min timeout

Example comparison (FAM)

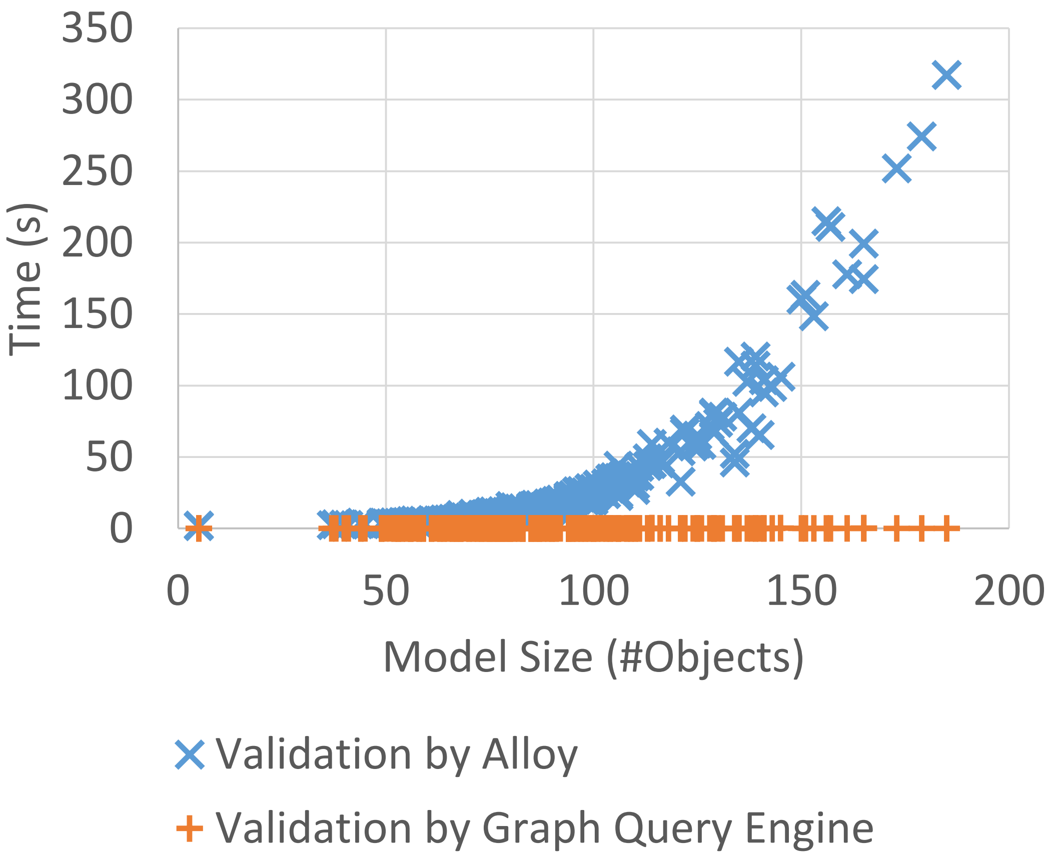


Our solver generates ~two orders of magnitude larger models

Additional Findings

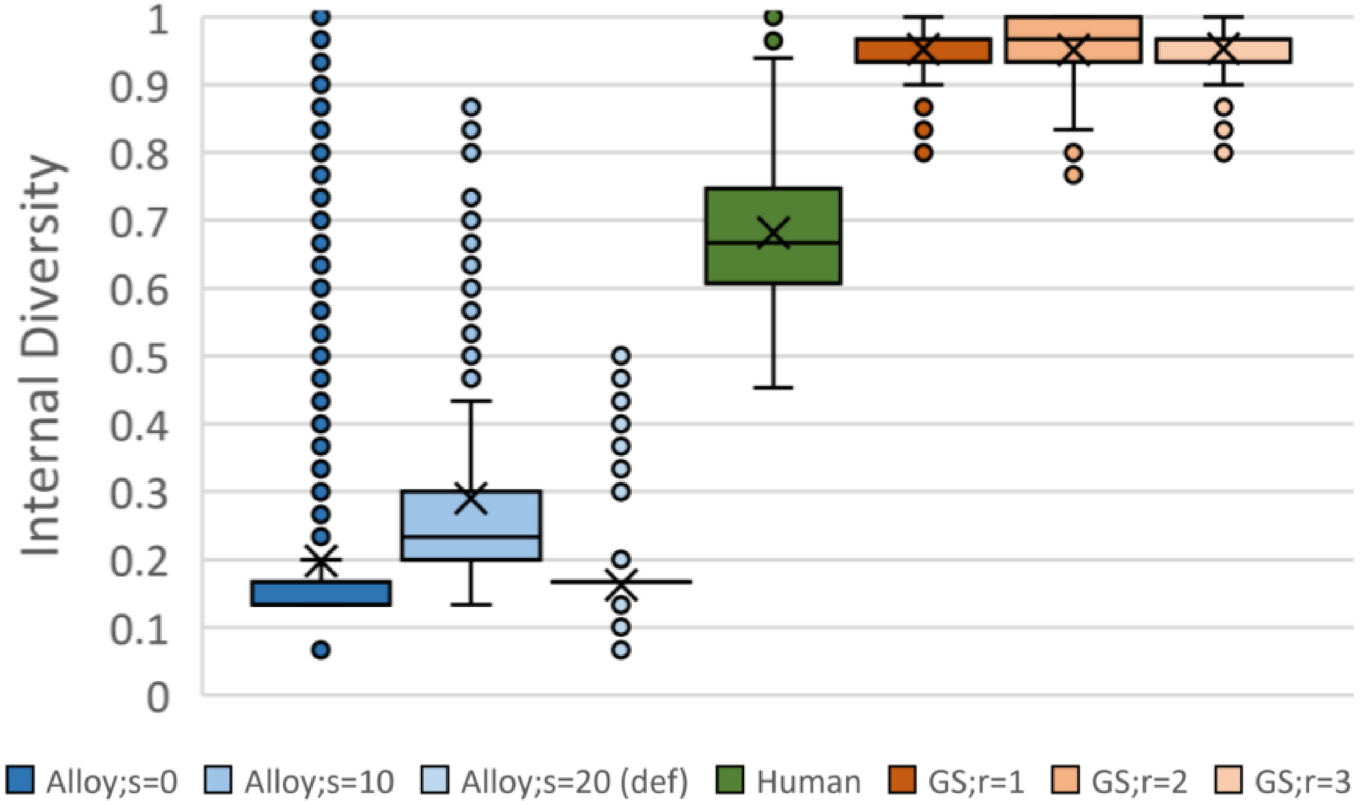
Constraint evaluation on complete graphs:

Query Engine vs Alloy



Diversity of graph models:

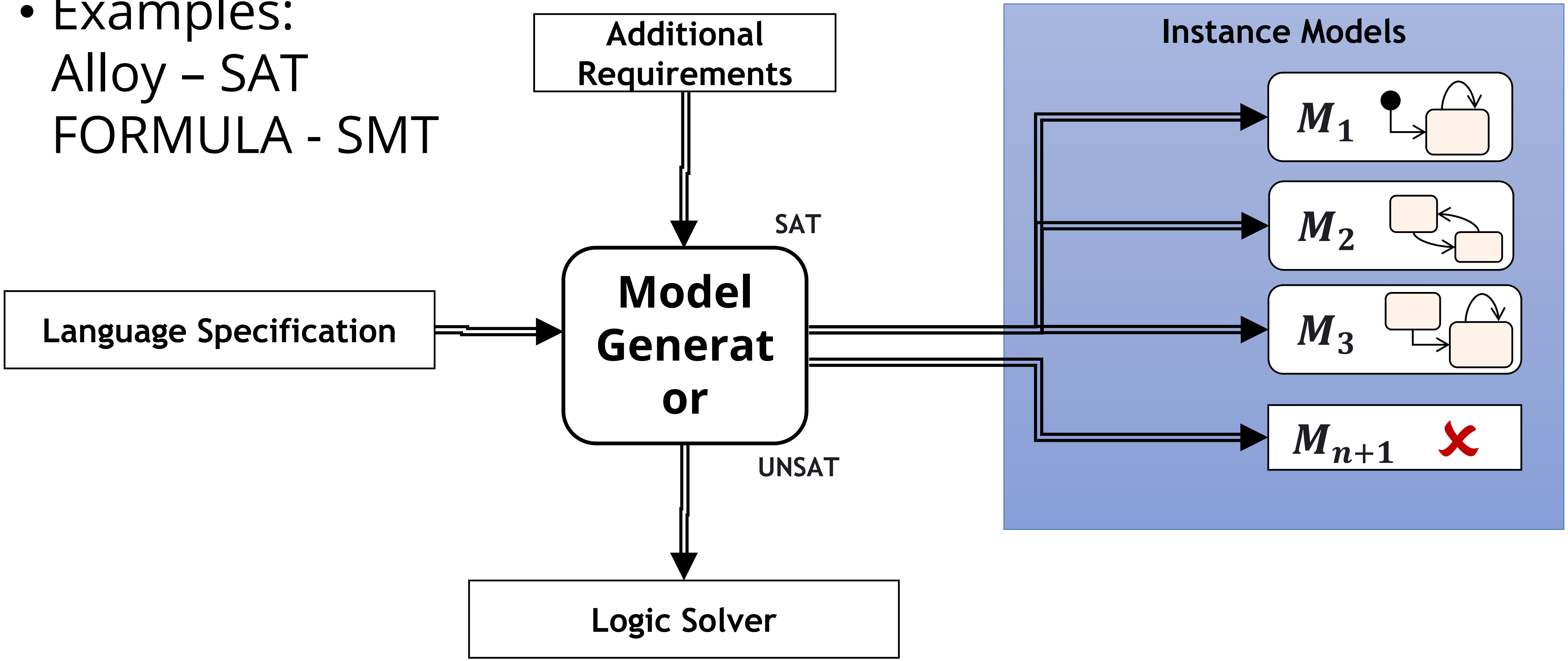
Alloy vs Human vs Graph Solver



Related Approaches

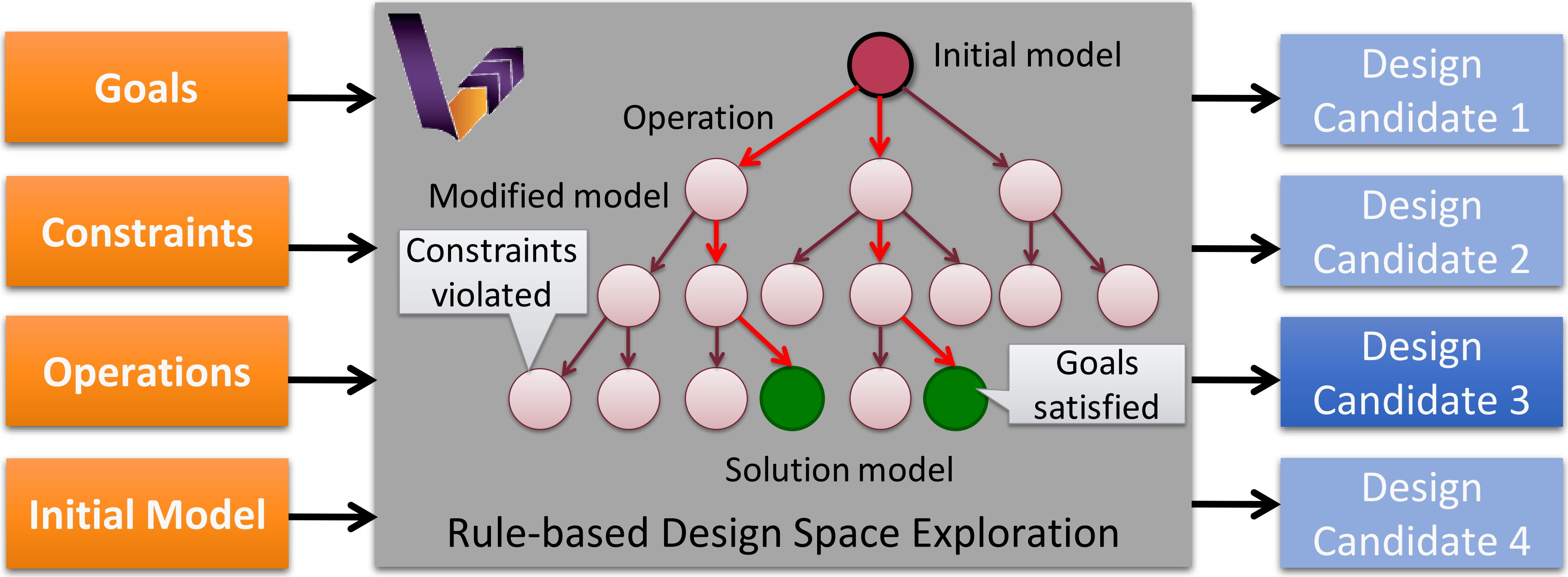
Solver-based model generators

- Examples:
  Alloy – SAT
  FORMULA - SMT



Rule-based Design Space exploration

ASE 11
ASE 14



Heuristics
- Approximate distance from a solution
- Guided or multi-objective optimization
- Backtracking / backjumping

Summary & Learning outcomes

Learning summary

- Model abstraction technique
- Model generation challenges
- Model generation ⬄ abstraction refinement