

Seminar: Formal Methods in Software Engineering

Summer Semester 2026 — Kick-off

| 23. April 2026



Organisation

Links

Website

<https://formal.kastel.kit.edu/teaching/Seminar-SS26>

ILIAS

https://ilias.studium.kit.edu/ilias.php?baseClass=ilrepositorygui&cmdNode=xu:m6&cmdClass=ilObjCourseGUI&cmd=view&ref_id=2896584

Registration

WiWi-Portal (see website)

Format

- Individual work only — no groups
- Each student picks **one topic** and **one advisor**
- Students may also **propose their own topic**
- Talk: **20–30 minutes**
- Report: **8 pages, ACM format**
- Reviews submitted via **EasyChair**

Timeline & Grading

Timeline

End of March	Final list of suggested topics
Apr 23, 13–14	First session (today)
April – July	Individual supervision
~End of June	Submit draft report to EasyChair (2 weeks before presentations)
Jul 2 & Jul 9	Student presentations
End of July	Peer reviews on EasyChair
End of semester	Final reports due

Grading

50 %	Final presentation
30 %	Final report
10 %	Peer review
10 %	Discussion

Lenses: Bidirectional Transformations

Description

Lenses formalise bidirectional transformations between data structures. The talk should cover both the classical symmetric lens framework and recent effectful extensions.

Papers

- M. Hoffmann et al. **Symmetric Lenses**. *POPL 2011*.
- Y. Xie et al. **Effectful Lenses**. *POPL 2025*.

Putback-Based Bidirectional Programming

Description

In bidirectional programming, transformations are automatically reversible. This topic focuses on the *putback* approach: only the reverse (“putback”) function is specified, and the forward (“get”) function is derived automatically — contrasting with correct-by-construction approaches where both are written together.

Papers

- H. Ko et al. **Bidirectional Programming in a More Applicative Style**. *PEPM 2016*.
- A recent publication on the same language (e.g. POPL 2025) or an alternative approach for comparison.

Efficient Graph Programming with Rewrite Rules

Description

Graph-based languages let programs operate directly on graph data structures. Recent work on the **GP 2** language addresses performance issues of earlier rule-based approaches.

Papers

- B. Bak, D. Plump. **Fast Rule-Based Graph Programs**. *Sci. Comput. Program.*, 208, 2021.
- A classic graph rewriting paper (to be agreed with advisor)

What Is a Model in Software Engineering?

Description

Models are ubiquitous in software engineering — created, managed, updated, and analysed in various methodologies. Yet there is little agreement on what a model *fundamentally is*. This topic surveys and critically compares foundational perspectives.

Note: some relevant literature is only available in German.

Papers

- T. Kühne. **What Is a Model?** *Dagstuhl Seminar 04101*, 2005.
- H. Stachowiak. **Allgemeine Modelltheorie**, Chapter 3. Springer, 1973.
- Possibly an Isola paper (to be agreed with advisor)

Verification of Graph Transformation Systems

Description

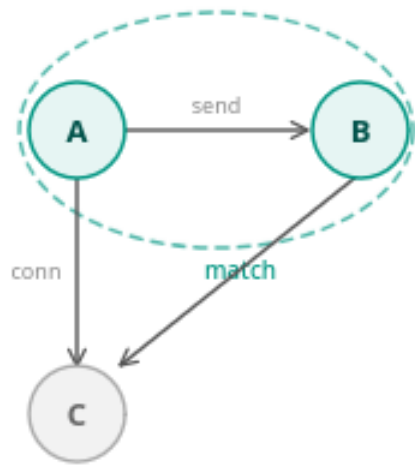
Graph transformation systems model dynamic, rule-driven systems — networks, UML diagrams, distributed protocols — by rewriting graphs via formally defined match-and-replace operations. Verification checks whether desired properties (safety, reachability) hold across all possible rule applications, requiring techniques to handle potentially infinite state spaces.

Contact: Shobhit Singh | shobhit.singh@kit.edu

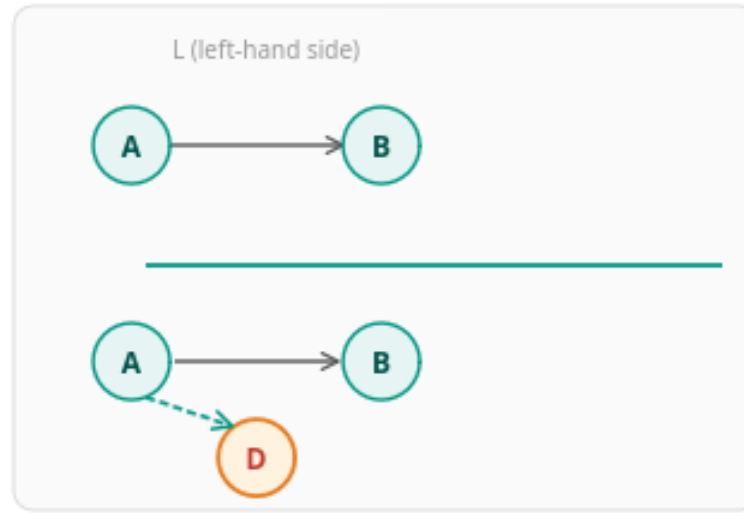
Paper

- B. König, D. Nolte, J. Padberg, A. Rensink. **A Tutorial on Graph Transformation.** *LNCS 10800, Springer, 2018.*

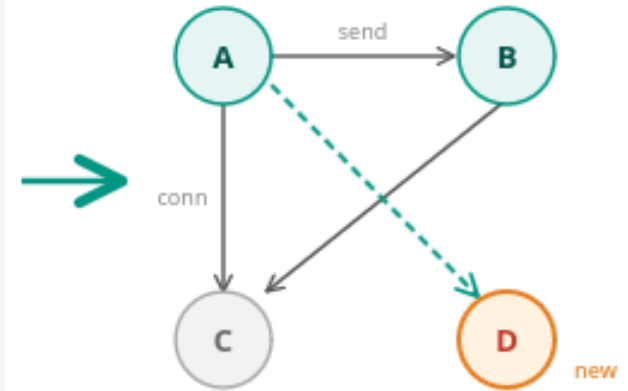
HOST GRAPH G



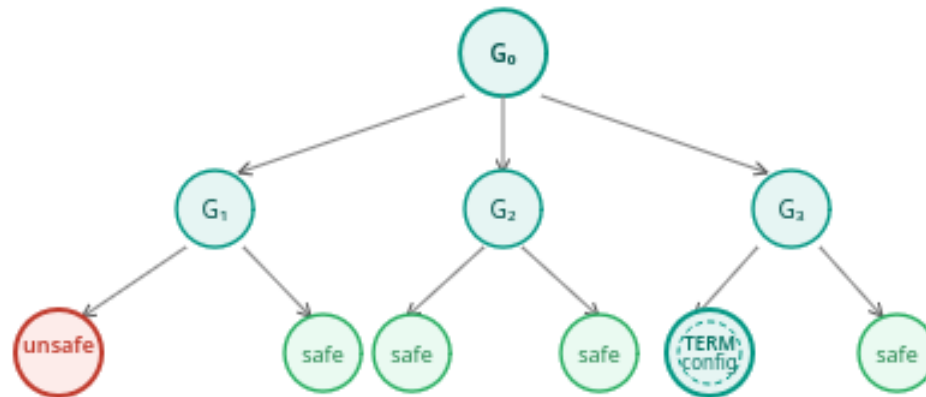
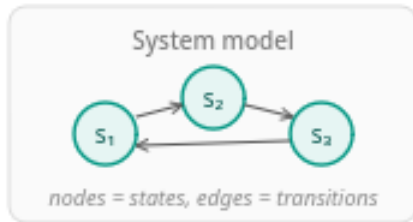
REWRITE RULE



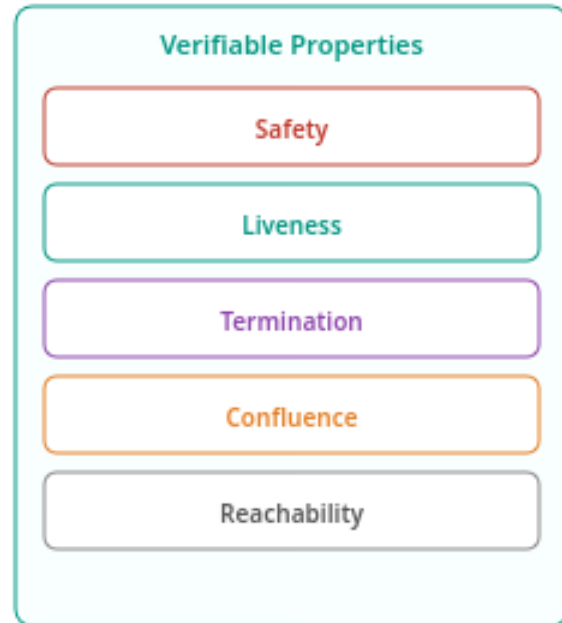
RESULT GRAPH G'



VERIFICATION VIA REACHABILITY IN THE STATE SPACE



Verifiable Properties



Program Repair as a Game

Description

Program repair automatically fixes bugs using specifications such as tests or assertions. Framing repair against a formal specification as a two-player game allows the repair strategy to be synthesised to win against all adversarial environment choices.

Papers

- B. Jobstmann, A. Griesmayer, R. Bloem. **Program Repair as a Game**. *CAV 2005*.
- R. Bloem, K. Chatterjee, B. Jobstmann. **Graph Games and Reactive Synthesis**. *Handbook of Model Checking, Springer, 2018*.

Program Repair as a Game

Jobstmann, Griesmayer and Bloem - CAV 2005

BUGGY PROGRAM

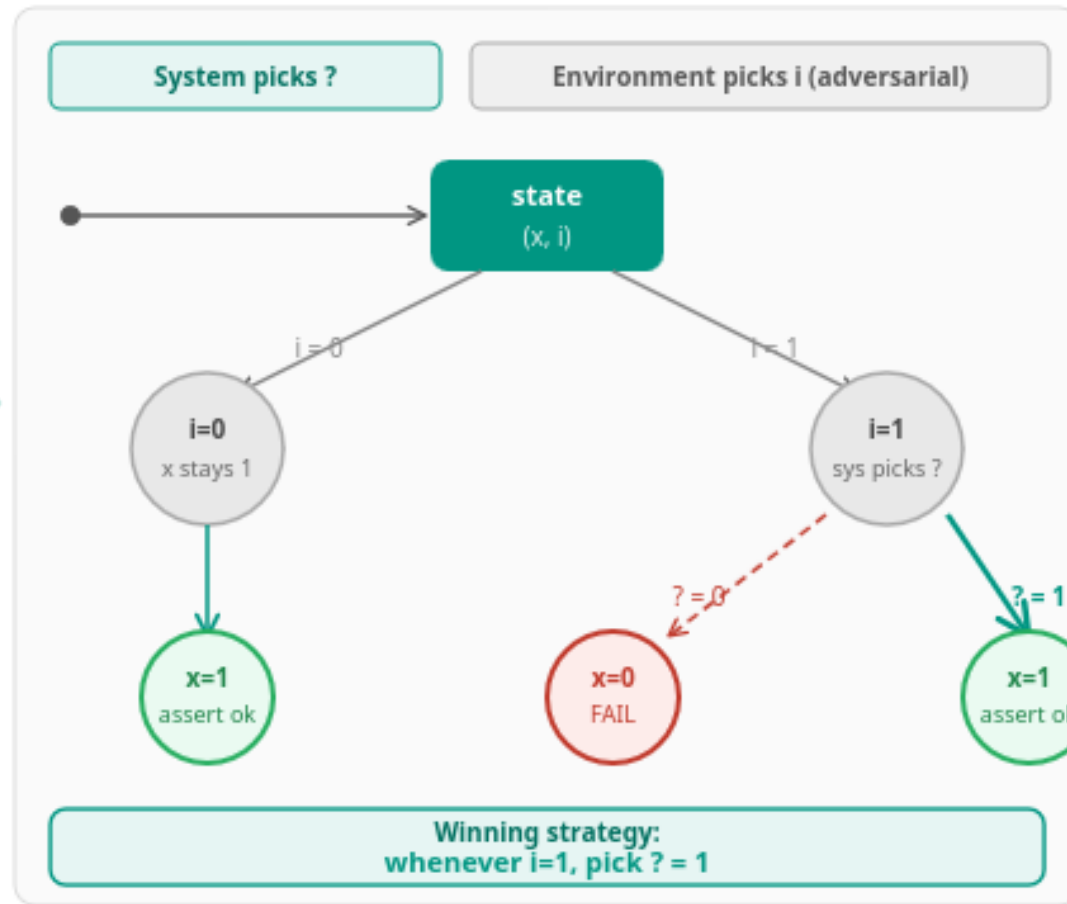
```
int x = 0;
while (true) {
  input i; // env
  if (i == 1)
    x = 0; // BUG
  assert(x == 1);
  // FAILS when i=1
}
```

spec: $G(x == 1)$

Step 1: free the hole

```
if (i == 1)
  x = ? ; // sys
```

2 player buchi game



REPAIRED PROGRAM

```
int x = 0;
while (true) {
  input i;
  if (i == 1)
    x = 1; // FIX
  assert(x == 1);
  // always holds
}
// only suspect line changed
```

$G(x == 1)$ holds

General case

Strategy can depend on state:

$? = f(x, i, \dots)$

Each branch gets a different value \rightarrow multi-line repair.

Model Repair

Description

Model repair restores consistency between multiple system models. Traditional approaches modify elements via predefined rules; recent work uses semantic properties and constraint solving, with formal guarantees about termination and absence of side effects.

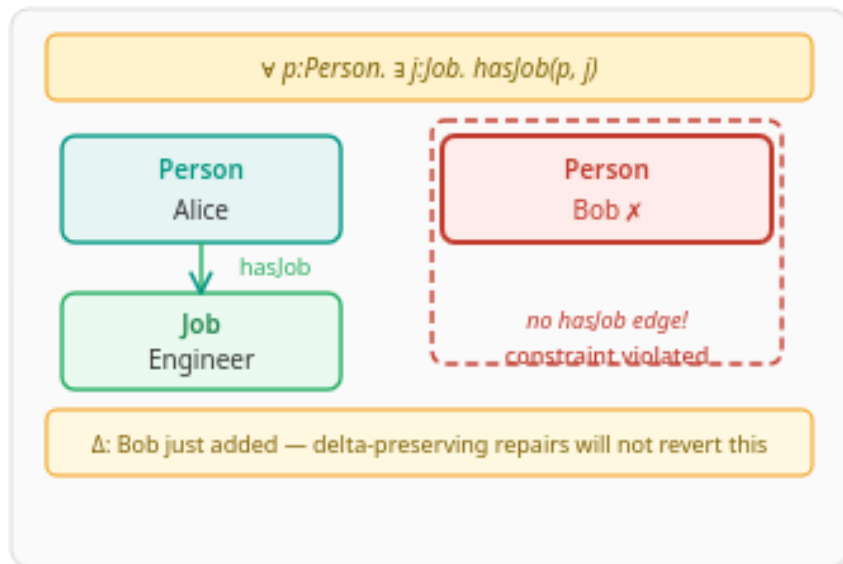
Papers

- C. Schneider, L. Lambers, F. Orejas. **A Logic-Based Incremental Approach to Graph Repair.** *FASE 2019.*
- C. Sandmann, A. Habel. **Rule-Based Graph Repair.** *GCM 2019, EPTCS 309.*
- M. Lauer, J. Kosiol, G. Taentzer. **Empowering Model Repair: A Rule-Based Approach to Graph Repair without Side Effects.** *MoDELS 2023.*

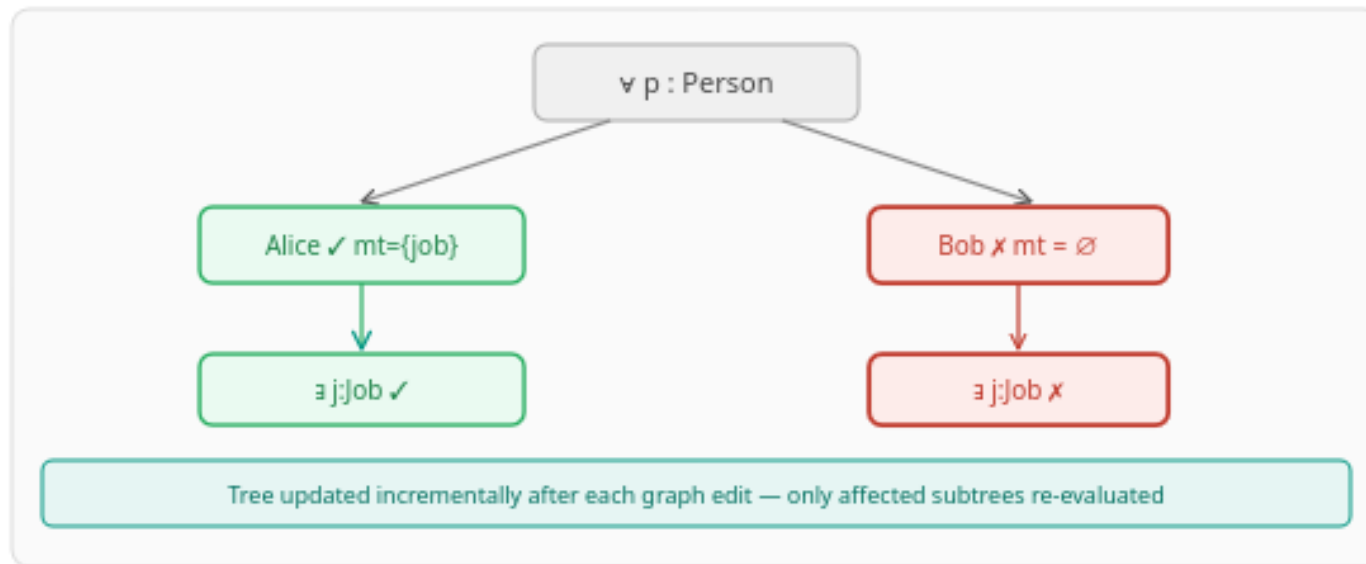
Model Repair via Graph Repair

Schneider, Lambers & Orejas — Incremental Logic-based Graph Repair (FASE 2019)

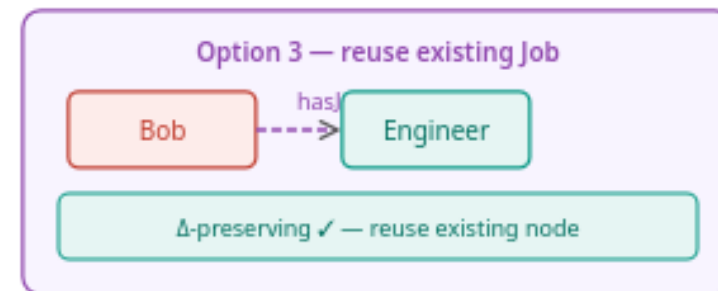
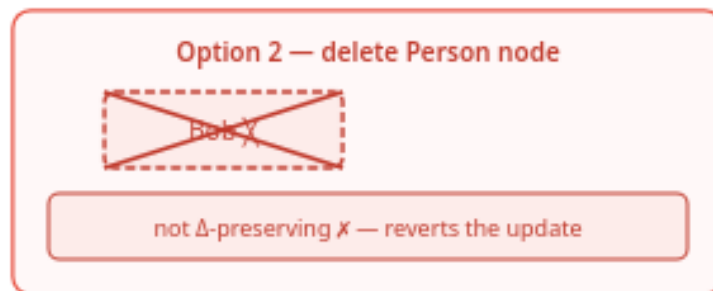
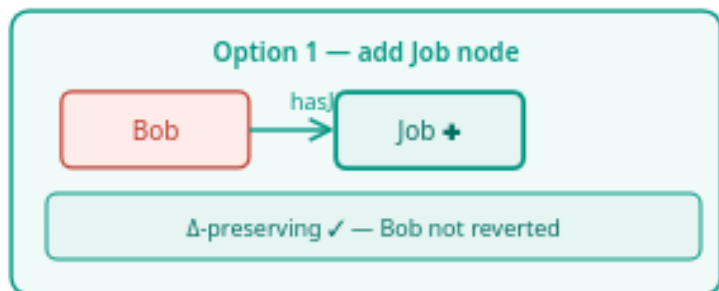
INCONSISTENT GRAPH G



SATISFACTION TREE



LEAST-CHANGING REPAIRS



Syntax Guided Program Synthesis

Description

Program synthesis automatically constructs a program satisfying a formal specification. Syntax-guided synthesis (SyGuS) constrains the search to expressions defined by a grammar, combining logical specifications with syntactic templates to make the problem tractable. So we constrain how the program is supposed to look i.e. syntax.

Paper

- R. Alur et al. **Syntax-Guided Synthesis**. *FMCAD 2013*.

Syntax-Guided Synthesis (SyGuS)

Rajeev Alur et al. — FMCAD 2013

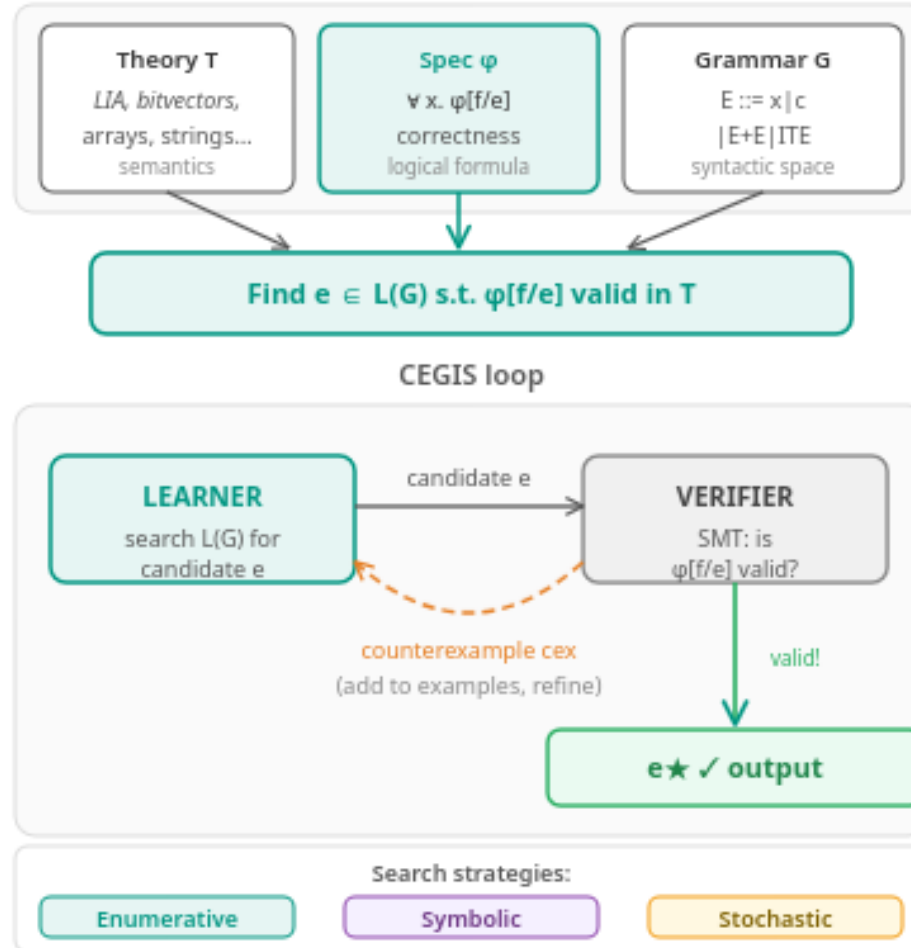
FLASH FILL EXAMPLE

Input (A)	Output (B)
Smith, John	John Smith
Jones, Mary	Mary Jones
Brown, Tom	Tom Brown ←

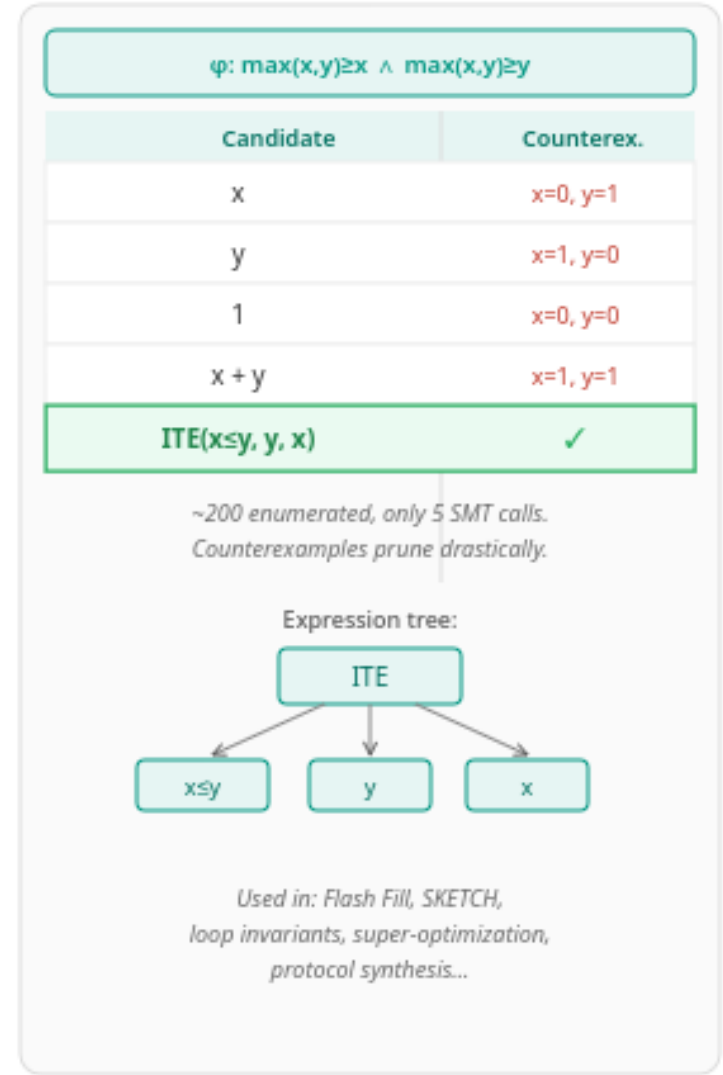
```
// synthesised:
f(s) = Concat(after(",",s),
             ", ", before(",",s))
```

User gives examples;
SyGuS finds the program.
Pure search — no ML/LLM.

THE SyGuS PROBLEM



ENUMERATIVE RUN: $\max(x,y)$



Modular Specification Language

Description

Investigate and understand approaches to constructing a modular specification language for complex systems, covering both semantic foundations (build systems) and language product-line techniques.

Contact: Tianhai Liu | Room 227 | tianhai.liu@kit.edu

Papers

- A. Mokhov, N. Mitchell, S. Peyton Jones. **Build Systems à la Carte: Theory and Practice.** *J. Funct. Program.* 30:e11, 2020.
- J. de Lara, E. Guerra, P. Bottoni. **Modular Language Product Lines: A Graph Transformation Approach.** *MODELS 2022.*

Change Impact Analysis

Description

Investigate and understand change-aware incremental approaches that increase the scalability of formal verification.

In particular, assuming the original system is consistent with respect to a given system property verified by a set of analyses, when a component changes and the impact propagates to other parts of the system, is it really necessary to rerun **all** analyses to ensure that the updated system remains consistent? What analysis results are still valid? What is not and requires re-execution?

Contact: Tianhai Liu | Room 227 | tianhai.liu@kit.edu

Papers

- S. Autexier, C. Lüth. **Adding Change Impact Analysis to the Formal Verification of C Programs.** *IFM 2010.*
- K. Johnson, R. Calinescu, S. Kikuchi. **An Incremental Verification Framework for Component-Based Software Systems.** *CBSE 2013.*
- H. Guissouma, M. Schindewolf, E. Sax. **ICARUS – Incremental Design and Verification of Software Updates in Safety-Critical Product Lines.** *SEAA 2021.*

BMC for Stochastic Systems (LLM Outputs)

Description

Apply bounded model checking (BMC) to stochastic systems, specifically to formally verify outputs of Large Language Models using probabilistic temporal logic (PCTL).

⇒ Verifying LLMs using BMC.

Contact: Tianhai Liu | Room 227 | tianhai.liu@kit.edu

Paper

- D. Gross, H. Spieker, A. Gottlieb. **Bounded PCTL Model Checking of Large Language Model Outputs.** *ICTAI 2025.*

LLMs + BMC for Loop Reasoning

Description

Combine Large Language Models with bounded model checking to address the loop reasoning bottleneck in BMC via neuro-symbolic loop invariant inference.

⇒ Increasing the scalability of BMC using LLMs.

Contact: Tianhai Liu | Room 227 | tianhai.liu@kit.edu

Paper

- G. Wu, W. Cao, Y. Yao, H. Wei, T. Chen, X. Ma. **LLM Meets Bounded Model Checking: Neuro-Symbolic Loop Invariant Inference.** *ASE 2024.*

Accelerating BMC via Interpolation Summaries

Description

Learn reusable summaries of program parts using interpolation to accelerate bounded model checking and improve scalability across multiple verification runs.

⇒ Increasing the scalability of BMC using static software analysis techniques.

Contact: Tianhai Liu | Room 227 | tianhai.liu@kit.edu

Paper

- M. Solanki, P. Chatterjee, A. Lal, S. Roy. **Accelerated Bounded Model Checking Using Interpolation Based Summaries.** *TACAS 2024.*

Next Steps

What to do now

1. Read through the topic list and the suggested papers
2. Contact your preferred **advisor** to discuss the topic
3. **Register** via WiWi-Portal if not done yet.