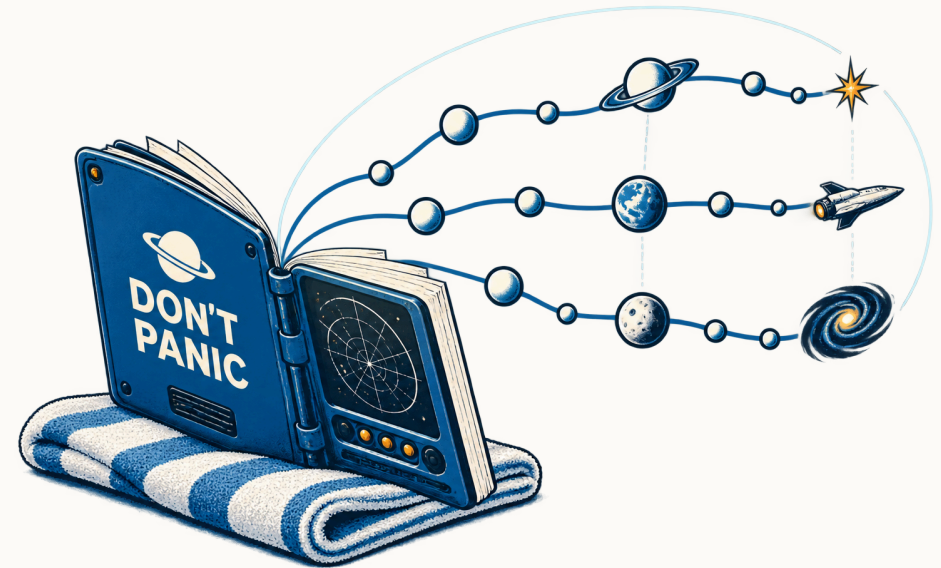


The Hitchhiker's Guide to Hyperproperties



Mishel Carelli, Bernd Finkbeiner, N. Ege Saraç

Kickoff meeting

Welcome



Mishel Carelli
mishel.carelli@cispa.de



Bernd Finkbeiner
finkbeiner@cispa.de



N. Ege Saraç
ege.sarac@cispa.de

Welcome



Mishel Carelli
mishel.carelli@cispa.de



Bernd Finkbeiner
finkbeiner@cispa.de



N. Ege Saraç
ege.sarac@cispa.de

Your turn: name + background + why this seminar

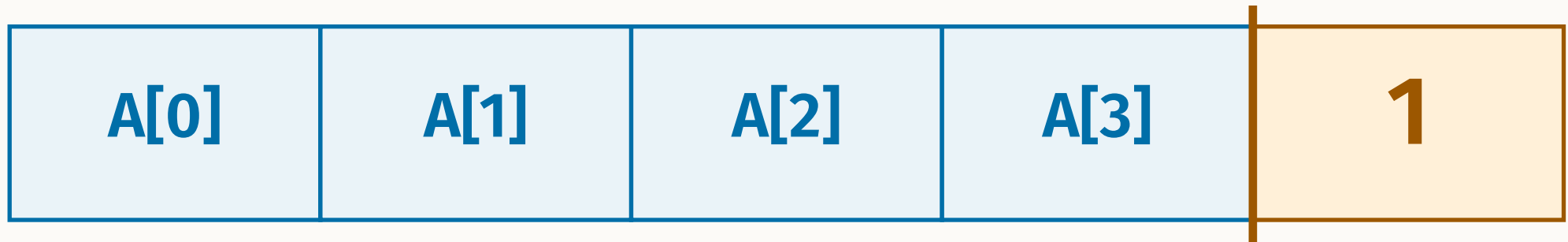
Today's meeting

1. What are **hyperproperties**, and why study them?
2. How will the **seminar** work?
3. Which **paper** would you like to present?

Spectre: Setup

Array A, size = 4

Secret bit



Allowed indices: 0, 1, 2, 3

Address A[4]
outside array A

Spectre: Size check

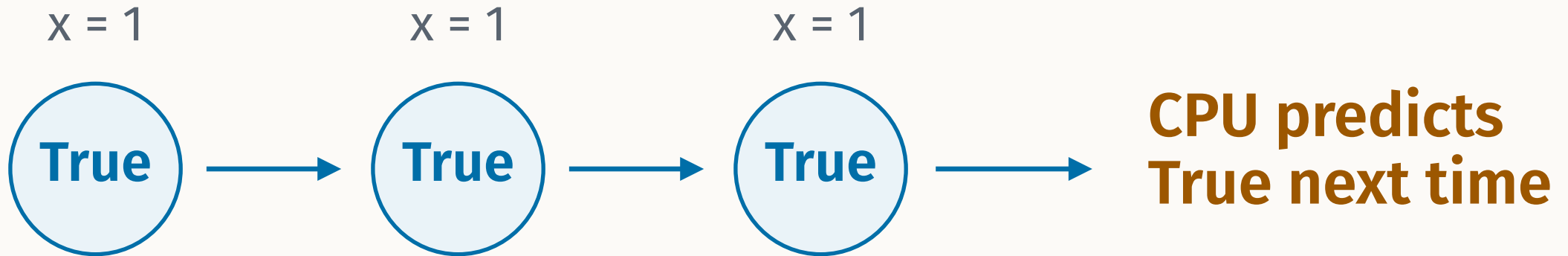
```
if x < size:    //size = 4  
    s = A[x]  
    read B[s]   //array B is public
```

x = 1  **1 < 4: True**  Read A[1]

x = 4  **4 < 4: False**  Reject

Spectre: Speculative execution

```
if x < size:    //size = 4  
    s = A[x]  
    read B[s]    //array B is public
```



Spectre: Speculative execution

```
if x < size:    //size = 4
    s = A[x]
    read B[s]    //array B is public
```



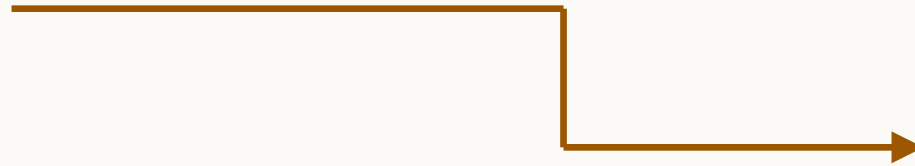
Spectre: Secret leaked?

```
if x < size:      //size = 4  
    s = A[x]  
    read B[s]     //array B is public
```

s = secret

1

s = 1 → read B[1]



B[0]
Not cached

B[1]
Cached

Spectre: Secret leaked?

```
if x < size:    //size = 4
    s = A[x]
    read B[s]   //array B is public
```

`s = secret` **4 < 4: False**

1

`s = 1 → read B[1]`

Discarded

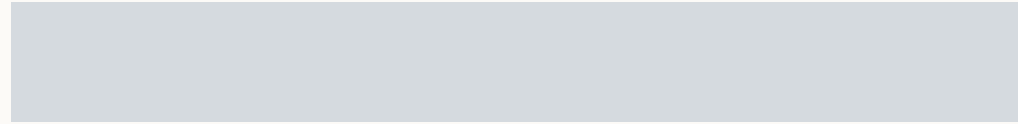
B[0]
Not cached

B[1]
Cached

Spectre: Secret leaked!

Time to read

B[0]



Slow

Read from memory

B[1]



Fast

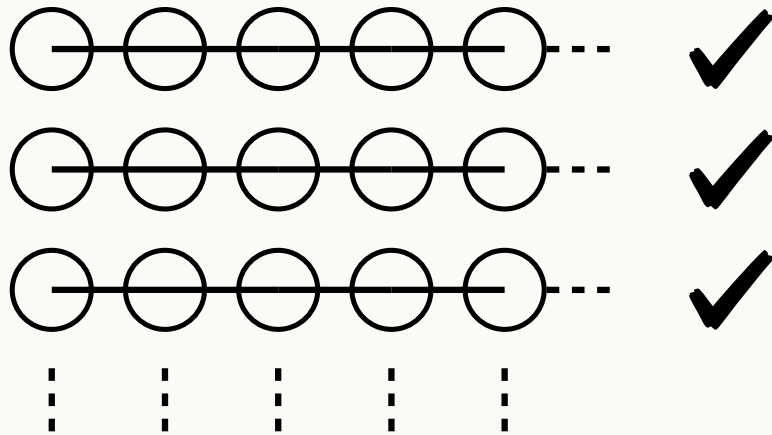
Read from cache

B[1] is fast → secret is **1**

Trace properties vs. hyperproperties

Trace properties

Check each execution separately.

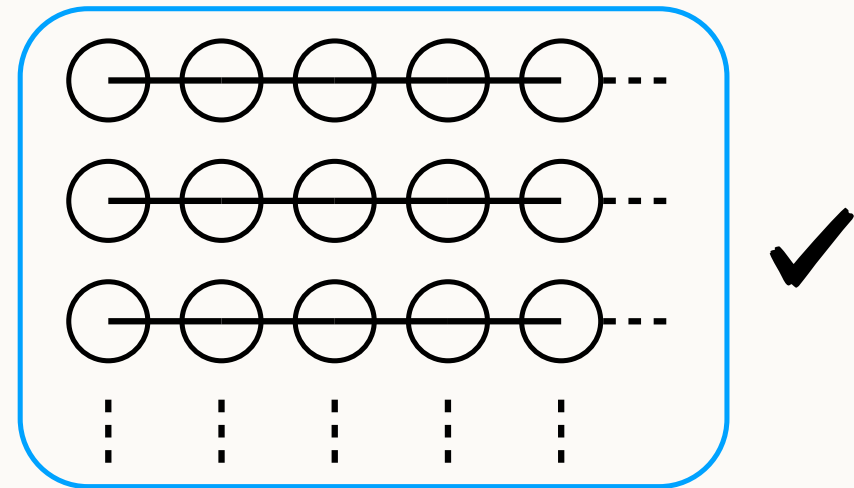


Example: size-check rule

Every read of A the CPU keeps must use a valid index.

Hyperproperties

Check a set of traces together.



Example: confidentiality

The secret must not affect what the attacker can observe.

Learning objectives

By the end of the seminar, you should be able to:

- Distinguish trace properties from hyperproperties using examples.
- Formulate and interpret simple hyperproperties in HyperLTL.
- Explain how automata-based HyperLTL model checking handles multiple traces and quantifiers.
- Explain a paper's contribution and its connection to the seminar's foundations.
- Evaluate a paper's assumptions, evidence, and limitations in the context of related work.
- Communicate technical ideas clearly in presentations, writing, and scientific discussion.

Seminar's structure



Group phase

- Two groups of 3-4 students.
- Read your group's readings.
- Prepare a presentation on your readings.
- Give one **joint presentation (~60 min)**.
- Lead a **discussion (~30 min)** using **exercises and quiz questions prepared in advance**.
- Participate in both group discussions.

Suggestions for the group phase

- Build one coherent talk together.
- Explain the definitions and main ideas that everyone will need later.
- Use examples to make the formal concepts concrete.
- Use exercises and quizzes to check understanding.

Individual phase

- One paper per student.
- Read your paper and prepare your presentation on it.
- Give a **presentation (~30 min)**.
- Lead a **discussion (~15 min)** on the paper's contributions and limitations.
- Participate in all individual discussions.
- Write a **summary (~5 pages)** placing your paper in the context of the seminar.

Suggestions for the individual phase

The presentation

- Explain the problem and why it matters.
- Use an example to explain the main contribution.
- Discuss assumptions, evidence, and limitations.

The summary

- Explain the contribution in your own words.
- Connect your paper to the foundations and other talks.

What we expect from you

- Listen actively to all presentations.
- Ask questions and contribute ideas to discussions.
- Give constructive feedback to your peers.
- Seek feedback (both from us and your peers).

Grading

- Group presentation – 40%
- Individual presentation – 40%
- Individual summary – 20%

Use of generative AI

- You may use AI as a study aid in any way that deepens your understanding of the topics.
- Do the readings yourself and double-check AI's answers.
- Prepare your presentation and summary yourself.
- In your slides and summary, state how you used AI tools.
- If you are unsure whether a use is allowed, ask us.

Further reading: [TUM ProLehre: Use AI wisely](#)

Group A: Hyperproperties, Hyperlogics

1. Hyperproperties

Clarkson, Schneider. Journal of Computer Security 2010.

2. Temporal Logics for Hyperproperties

Clarkson, Finkbeiner, Koleini, Micinski, Rabe, Sánchez. POST 2014.

3. Meltdown, Spectre, and why hardware can be correct yet insecure (for motivation)

Myers. Blog post, 2018.

Hyperproperties

Michael R. Clarkson^{*} and Fred B. Schneider

Department of Computer Science, Cornell University, Ithaca, NY, USA

E-mails: {clarkson, fbs}@cs.cornell.edu

Trace properties, which have long been used for reasoning about systems, are sets of execution traces. Hyperproperties, introduced here, are sets of trace properties. Hyperproperties can express security policies, such as secure information flow and service level agreements, that trace properties cannot. Safety and liveness are generalized to hyperproperties, and every hyperproperty is shown to be the intersection of a safety hyperproperty and a liveness hyperproperty. A verification technique for safety hyperproperties is given and is shown to generalize prior techniques for verifying secure information flow. Refinement is shown to be applicable with safety hyperproperties. A formalized characterization of hyperproperties is

Temporal Logics for Hyperproperties

Michael R. Clarkson¹, Bernd Finkbeiner², Masoud Koleini¹,
Kristopher K. Micinski³, Markus N. Rabe², and César Sánchez⁴

¹George Washington University; ²Universität des Saarlandes;
³University of Maryland, College Park; ⁴IMDEA Software Institute

Abstract. Two new logics for verification of hyperproperties are proposed. Hyperproperties characterize security policies, such as noninterference, as a property of sets of computation paths. Standard temporal logics such as LTL, CTL, and CTL* can refer only to a single path at a time, hence cannot express many hyperproperties of interest. The logics proposed here, HyperLTL and HyperCTL*, add explicit and simultane-

Group B: HyperLTL Model Checking

1. Principles of Model Checking

Baier, Katoen. MIT Press 2008.

Selected sections:

§4.3 Automata on Infinite Words

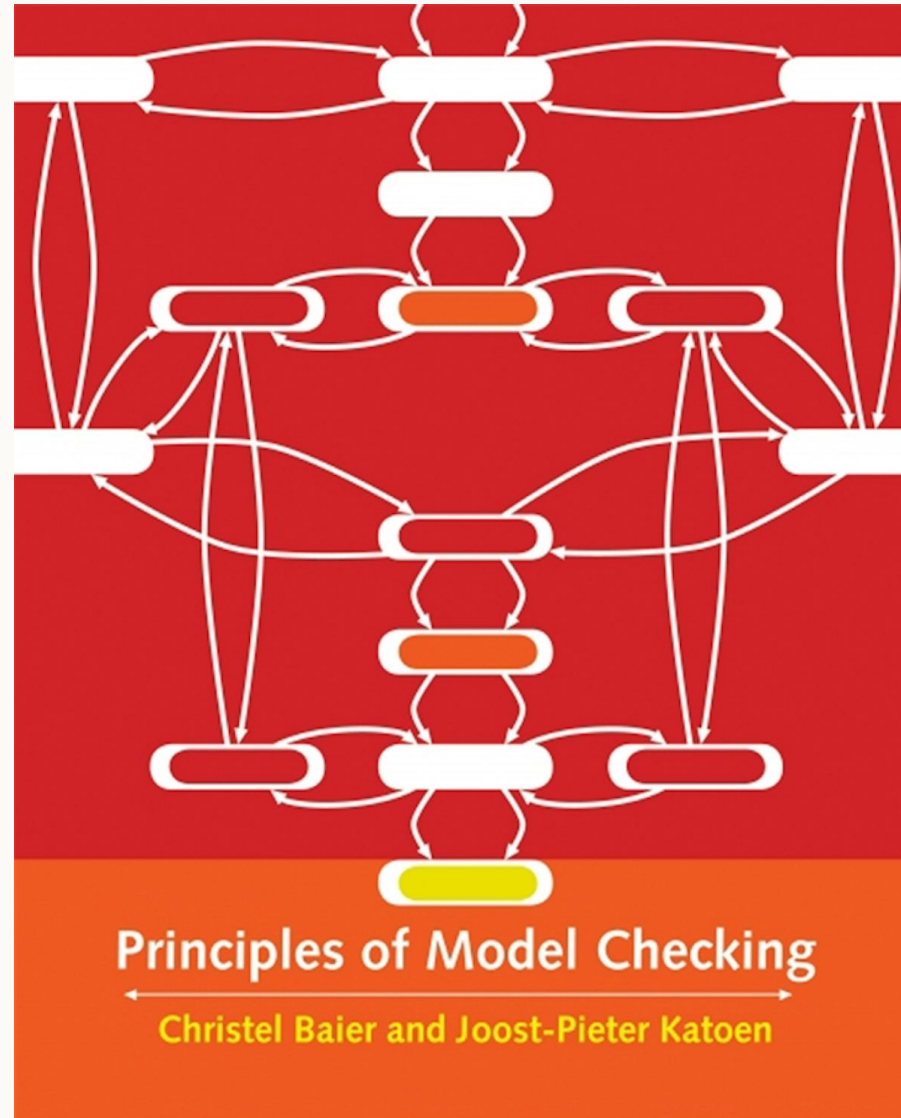
§4.4 Model-Checking Omega-Regular Properties

§5.1 Linear Temporal Logic (except §5.1.6)

§5.2 Automata-Based LTL Model Checking (opening material and §5.2.1)

2. Algorithms for Model Checking HyperLTL and HyperCTL*

Finkbeiner, Rabe, Sánchez. CAV 2015.



Algorithms for Model Checking HyperLTL and HyperCTL*

Bernd Finkbeiner¹, Markus N. Rabe¹, and César Sánchez²

¹Saarland University, ²IMDEA Software Institute

Abstract. We present an automata-based algorithm for checking finite state systems for hyperproperties specified in HyperLTL and HyperCTL*. For the alternation-free fragments of HyperLTL and HyperCTL* the automaton construction allows us to leverage existing model checking technology. Along several case studies, we demonstrate that the approach enables the verification of real hardware designs for properties that could not be checked before. We study information flow properties of an I2C

Research papers

1. **AutoHyper: Explicit-State Model Checking for HyperLTL**
Beutner, Finkbeiner. TACAS 2023.
2. **Prophecy Variables for Hyperproperty Verification**
Beutner, Finkbeiner. CSF 2022.
3. **Finding $\forall\exists$ Hyperbugs using Symbolic Execution**
Correnson, Nießen, Finkbeiner, Weissenbacher. OOPSLA 2024.
4. **Reasoning About Quality in Hyperproperties**
Graepler, Monmege, Talbot. CSL 2026.
5. **Hyperproperties of Real-Valued Signals**
Nguyen, Kapinski, Jin, Deshmukh, Johnson. MEMOCODE 2017.
6. **A formalization of Heisenbugs and their causes in terms of hyperproperties**
Sallinger, Weissenbacher, Zuleger. Software and Systems Modeling 2026.
7. **Hypertesting of Programs: Theoretical Foundation and Automated Test Generation**
Pasqua, Ceccato, Tonella. ICSE 2024.

AutoHyper: Explicit-State Model Checking for HyperLTL

Raven Beutner^(✉)  and Bernd Finkbeiner 

CISPA Helmholtz Center for Information Security, Germany
`{raven.beutner,finkbeiner}@cispa.de`

Abstract. HyperLTL is a temporal logic that can express hyperproperties, i.e., properties that relate multiple execution traces of a system. Such properties are becoming increasingly important and naturally occur, e.g., in information-flow control, robustness, mutation testing, path planning, and causality checking. Thus far, complete model checking tools for HyperLTL have been limited to alternation-free formulas, i.e.,

Prophecy Variables for Hyperproperty Verification

Raven Beutner
CISPA Helmholtz Center for
Information Security
Germany

Bernd Finkbeiner
CISPA Helmholtz Center for
Information Security
Germany

Abstract—Temporal logics for hyperproperties like HyperLTL use trace quantifiers to express properties that relate multiple system runs. In practice, the verification of such specifications is mostly limited to formulas without quantifier alternation, where verification can be reduced to checking a trace property over the self-composition of the system. Quantifier alternations like $\forall\pi.\exists\pi'.\phi$, can either be solved by complementation or with an interpretation as a two-person game between a $\forall$ -player, who incrementally constructs the trace π , and an $\exists$ -player, who constructs π' in such a way that π and π' together satisfy ϕ . The game-based approach is significantly cheaper but incomplete because the $\exists$ -player does not know the future moves of the $\forall$ -

systems where the simple variant of NI is violated when the nondeterminism influences the low-security output.

In this paper, we study the verification of HyperLTL, i.e., the question of whether a given system satisfies a given property. For HyperLTL, the structure of the quantifier prefix has direct implications on the complexity of the verification problem. For our example properties, the fundamental difference (w.r.t. verification) between NI and GNI, is that NI uses only universal quantification over traces (we say NI is alternation-free) whereas GNI involves a quantifier alternation. Verification of

Finding $\forall\exists$ Hyperbugs using Symbolic Execution

ARTHUR CORRENSON*, CISA Helmholtz Center for Information Security, Germany

TOBIAS NIESSEN*, TU Wien, Austria

BERND FINKBEINER, CISA Helmholtz Center for Information Security, Germany

GEORG WEISSENBACHER, TU Wien, Austria

Many important hyperproperties, such as refinement and generalized non-interference, fall into the class of $\forall\exists$ hyperproperties and require, for each execution trace of a system, the existence of another trace relating to the first one in a certain way. The alternation of quantifiers renders $\forall\exists$ hyperproperties extremely difficult to verify, or even just to test. Indeed, contrary to trace properties, where it suffices to find a single counterexample trace, refuting a $\forall\exists$ hyperproperty requires not only to find a trace, but also a proof that no second trace satisfies the specified relation with the first trace. As a consequence, automated testing of $\forall\exists$ hyperproperties falls out of the scope of existing automated testing tools. In this paper, we present a fully automated approach

Reasoning about Quality in Hyperproperties

Samuel Graepler 

Fakultät für Mathematik und Informatik, Universität Leipzig, Germany

Benjamin Monmege  

Aix Marseille Univ, CNRS, LIS, Marseille, France

Jean-Marc Talbot  

Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France

Abstract

Hyperproperties allow one to specify properties of systems that inherently involve not single executions of the system, but several of them at once: observational determinism and non-inference are two examples of such properties used to study the security of systems. Logics like HyperLTL have been studied in the past to model check hyperproperties of systems. However, most of the time, requiring strict security properties is actually ineffective as systems do not meet such requirements. To overcome this issue, we introduce qualitative reasoning in HyperLTL, inspired by a similar work on LTL by Almagor, Podelski and Kupferman [2] where a formula has a value in the interval $[0, 1]$

Hyperproperties of Real-Valued Signals

Luan Viet Nguyen

University of Texas at Arlington

luanvnguyen@mavs.uta.edu

James Kapinski

Toyota Motor North America R&D

jim.kapinski@toyota.com

Xiaoqing Jin

Toyota Motor North America R&D

xiaoqing.jin@toyota.com

Jyotirmoy V. Deshmukh

Toyota Motor North America R&D

jyotirmoy.deshmukh@toyota.com

Taylor T. Johnson

Vanderbilt University

taylor.johnson@vanderbilt.edu

ABSTRACT

A *hyperproperty* is a property that requires two or more execution traces to check. This is in contrast to properties expressed using temporal logics such as LTL, MTL and STL, which can be checked over individual traces. Hyperproperties are important as they are used to specify critical system performance objectives, such as those related to security, stochastic (or average) performance, and relationships between behaviors. We present the first study of hyperproperties of cyber-physical systems (CPSs). We introduce a new formalism for specifying a class of hyperproperties defined over real-valued signals, called HyperSTL. The proposed logic extends signal temporal logic (STL) by adding existential and universal trace

cyber-physical systems (CPSs), and we present a taxonomy of hyperproperties used to address security and control design concerns for CPSs. Also, we provide practical techniques for automating the process of testing hyperproperties for CPSs.

In contrast to trace properties expressed over individual execution traces, hyperproperties are defined over multiple execution traces. For example, one execution of a system cannot be checked against a service level agreement property such as “*the average time elapsed between a user’s request and response over all executions should be less than 1 second*”; the property can only be evaluated over all system execution traces. Moreover, we can consider an information-flow policy of *noninterference* specified as “*for all pairs*

A formalization of Heisenbugs and their causes in terms of hyperproperties

Sarah Sallinger¹  · Georg Weissenbacher¹  · Florian Zuleger¹ 

Received: 25 April 2024 / Revised: 2 September 2025 / Accepted: 5 November 2025 / Published online: 24 February 2026
© The Author(s) 2026

Abstract

The already challenging task of identifying the cause of a bug becomes even more cumbersome if those bugs disappear or change their behavior under observation. Such bugs occur in a range of contexts including elusive concurrency bugs as well as unintended system alterations during debugging and—as a pun on the name of Werner Heisenberg—are often referred to as Heisenbugs. Heisenbugs can be caused by various sources of nondeterminism on different system levels, many of which developers and testers might not even be aware of. This paper provides formal foundations for rigorously reasoning about causes of Heisenbugs. It provides a formal definition of Heisenbugs in terms of a hyperproperty and introduces a framework for determining the causality of Heisenbugs in presence of multiple candidate causes based on said hyperproperty. We analyze

Hypertesting of Programs: Theoretical Foundation and Automated Test Generation

Michele Pasqua
michele.pasqua@univr.it
Dept. of Computer Science
University of Verona
Verona, Italy

Mariano Ceccato
mariano.ceccato@univr.it
Dept. of Computer Science
University of Verona
Verona, Italy

Paolo Tonella
paolo.tonella@usi.ch
Software Institute
Università della Svizzera italiana
Lugano, Switzerland

ABSTRACT

Hyperproperties are used to define correctness requirements that involve relations between multiple program executions. This allows, for instance, to model *security and concurrency requirements*, which cannot be expressed by means of trace properties.

In this paper, we propose a novel systematic approach for automated testing of hyperproperties. Our contribution is both foundational and practical. On the foundational side, we define a *hypertesting framework*, which includes a novel hypercoverage adequacy criterion designed to guide the synthesis of test cases for hyperproperties. On the practical side, we instantiate such framework by implementing HyperFuzz and HyperEvo, two test generators targeting the *Non-Interference* security requirement, that rely respectively

program should not threaten human life) and *security* (e.g., confidential information should not be leaked publicly). Most existing *software Verification & Validation (V&V)* approaches focus on functional correctness properties that belong to the family of so called *trace properties*. These correspond to program requirements that can be checked by observing single program executions.

Consider, as an example, a smartphone app and suppose to check whether the app crashes or not. If the app can indeed crash, then by observing its execution when inputs are smartly selected or generated, we can eventually notice the crash. There are, however, important program requirements that go beyond the family of trace properties. For instance, *security requirements* may need to compare more than one execution at a time, in order to spot a defect.

Important dates

19 October 2026

Email me your preferences by 12:30

Papers and groups announced same day

12-13 November 2026

Group presentations, 14:00-16:00

5609.EG.038 (00.09.038)

4-5 February 2027

Individual presentations, 14:00-18:00

5609.EG.038 (00.09.038)

19 February 2027

Summary submissions (by email)

Next steps

- Skim the abstracts and introductions of the papers.
- Sort the papers by decreasing order of your interest.
- Email me your sorted list **by 12:30 on 19 October (Monday)**.
- Keep the presentation dates free.

Next steps

- Skim the abstracts and introductions of the papers.
- Sort the papers by decreasing order of your interest.
- Email me your sorted list **by 12:30 on 19 October (Monday)**.
- Keep the presentation dates free.

Questions?