Monitoring Hyperproperties

Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup
Reactive Systems Group, Saarland University, Germany

The 17th International Conference on Runtime Verification
Seattle, USA, 2017

Hyperproperties

Definition

A Hyperproperty $H \subseteq 2^{TR}$ is a set of sets of execution traces [Clarkson, Schneider, '10].

Example

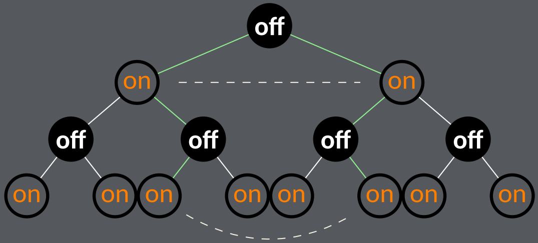
trace equality: "All traces agree on a proposition p."
observational determinism: "A program appears deterministic to low security users."
noninterference, generalized noninterference, noninference, declassification, ...

A Logical Approach to Information-Flow Control

HyperLTL [Clarkson, Finkbeiner, Koleini, Micinski, Rabe, Sánchez, '14]

HyperLTL

- LTL + explicit trace quantification:
  $\exists\pi.\exists\pi'.\ \Box on_\pi \wedge \Box\neg on_{\pi'}$
  satisfiable by $\{\{on\}^\omega, \{off\}^\omega\}$
- trace equality:
  $\forall\pi.\ \forall\pi'.\ \Box(on_\pi \leftrightarrow on_{\pi'})$
- observational determinism:
  $\forall\pi.\ \forall\pi'.\ (O_\pi = O_{\pi'})\ \mathcal{W}\ (I_\pi \neq I_{\pi'})$



Monitoring Hyperproperties

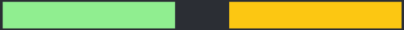
- we sequentially observe traces of a system
- when a new trace comes in, we check whether a given hyperproperty still holds

Monitoring Hyperproperties

- we sequentially observe traces of a system
- when a new trace comes in, we check whether a given hyperproperty still holds

Monitoring Hyperproperties

- we sequentially observe traces of a system
- when a new trace comes in, we check whether a given hyperproperty still holds



Monitoring Hyperproperties

- we sequentially observe traces of a system
- when a new trace comes in, we check whether a given hyperproperty still holds

Monitoring Hyperproperties

- we sequentially observe traces of a system
- when a new trace comes in, we check whether a given hyperproperty still holds

Monitoring Hyperproperties

- we sequentially observe traces of a system
- when a new trace comes in, we check whether a given hyperproperty still holds

Monitoring Hyperproperties

- we sequentially observe traces of a system
- when a new trace comes in, we check whether a given hyperproperty still holds

Monitoring Hyperproperties

- we sequentially observe traces of a system
- when a new trace comes in, we check whether a given hyperproperty still holds

Overview

1. monitor construction
2. two techniques to make monitoring of hyperproperties feasible in practice:
   - Trace Analysis: exploits a dominance relation between traces
   - Specification Analysis: exploits symmetry, transitivity, and reflexivity in the specification

Monitor Construction

- conference management system with author and pc traces
- no paper submission is lost:
  - every submission (s) is visible (v) to every pc member
  - when comparing two pc traces, they have to agree on v

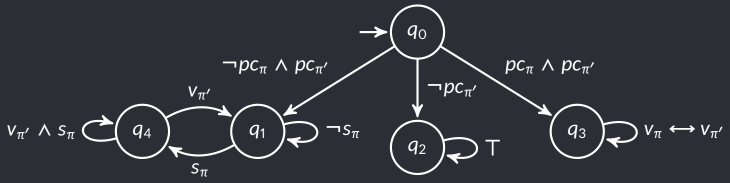
$$\forall \pi.\ \forall \pi'.\ (\neg pc_\pi \wedge pc_{\pi'}) \rightarrow \bigcirc\square(s_\pi \rightarrow \bigcirc v_{\pi'}) \wedge \tag{1}$$

$$(pc_\pi \wedge pc_{\pi'}) \rightarrow \bigcirc\square(v_\pi \leftrightarrow v_{\pi'}) \tag{2}$$

Monitor Construction

$$\forall \pi.\ \forall \pi'.\ (\neg pc_\pi \wedge pc_{\pi'}) \rightarrow \bigcirc \Box (s_\pi \rightarrow \bigcirc v_{\pi'}) \wedge$$
$$(pc_\pi \wedge pc_{\pi'}) \rightarrow \bigcirc \Box (v_\pi \leftrightarrow v_{\pi'})$$

$$\Downarrow$$



Monitor Construction

Deterministic monitor template $\mathscr{M} = (\Sigma, Q, \delta, q_0)$:

- finite alphabet $\Sigma = 2^{AP \times \mathscr{V}}$

The automaton runs in parallel over n-ary tuple $N \in ((2^{AP})^*)^n$ of finite traces:

$$\delta\left(q_i, \bigcup_{j=1}^{n} \bigcup_{a \in N(j)(i)} \{(a, \pi_j)\}\right) = q_{i+1} \quad .$$

Monitor Construction

Deterministic monitor template $\mathcal{M} = (\Sigma, Q, \delta, q_0)$:

- finite alphabet $\Sigma = 2^{AP \times \mathcal{V}}$

The automaton runs in parallel over n-ary tuple $N \in ((2^{AP})^*)^n$ of finite traces:

$$\delta\left(q_i, \bigcup_{j=1}^{n} \bigcup_{a \in N(j)(i)} \{(a, \pi_j)\}\right) = q_{i+1} \quad .$$

Monitor Construction

Deterministic monitor template $\mathcal{M} = (\Sigma, Q, \delta, q_0)$:

- finite alphabet $\Sigma = 2^{AP \times \mathcal{V}}$

The automaton runs in parallel over n-ary tuple $N \in ((2^{AP})^*)^n$ of finite traces:

$$\delta\left(q_i, \bigcup_{j=1}^{n} \bigcup_{a \in N(j)(i)} \{(a, \pi_j)\}\right) = q_{i+1} \quad .$$

Monitor Construction

Deterministic monitor template $\mathscr{M} = (\Sigma, Q, \delta, q_0)$:

- finite alphabet $\Sigma = 2^{AP \times \mathscr{V}}$

The automaton runs in parallel over n-ary tuple $N \in ((2^{AP})^*)^n$ of finite traces:

$$\delta\left(q_i, \bigcup_{j=1}^{n} \bigcup_{a \in N(j)(i)} \{(a, \pi_j)\}\right) = q_{i+1} \quad .$$

Monitor Construction

Deterministic monitor template $\mathscr{M} = (\Sigma, Q, \delta, q_0)$:

- finite alphabet $\Sigma = 2^{AP \times \mathcal{V}}$

The automaton runs in parallel over n-ary tuple $N \in ((2^{AP})^*)^n$ of finite traces:

$$\delta\left(q_i, \bigcup_{j=1}^{n} \bigcup_{a \in N(j)(i)} \{(a, \pi_j)\}\right) = q_{i+1} \quad .$$

Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

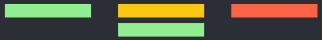
The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!



Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

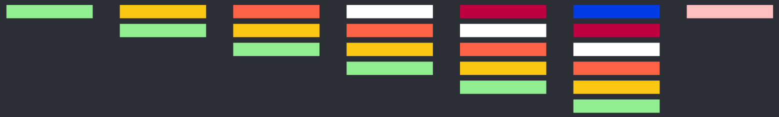
The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!

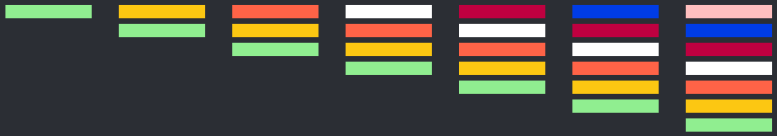
Memory Explosion

The naive approach always stores every trace seen so far!



Memory Explosion

The naive approach always stores every trace seen so far!



Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!

Memory Explosion

The naive approach always stores every trace seen so far!

Trace Analysis: discard traces that are dominated by other traces

Trace Analysis - Example

{}	{s}	{}	{}	{}
{}	{}	{s}	{}	{}

an author submits a paper

another author submits a paper

Trace Analysis - Example

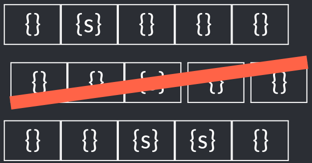
{}	{s}	{}	{}	{}
{}	{}	{s}	{}	{}
{}	{}	{s}	{s}	{}

an author submits a paper

another author submits a paper

an author submits two papers

Trace Analysis - Example

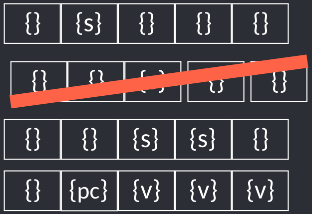


an author submits a paper

another author submits a paper

an author submits two papers

Trace Analysis - Example



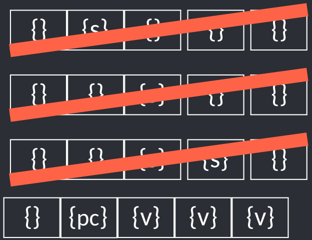
an author submits a paper

another author submits a paper

an author submits two papers

a pc observes 3 submissions

Trace Analysis - Example



an author submits a paper

another author submits a paper

an author submits two papers

a pc observes 3 submissions

Trace Analysis - Example

{}	{pc}	{v}	{v}	{v}

a pc member observes three submissions

Trace Analysis - Example

{}	{pc}	{v}	{v}	{v}
{}	{pc}	{v}	{v}	{}

a pc member observes three submissions

↯ a pc member observes two submissions ↯

Trace Analysis

Definition (Trace Redundancy)

- HyperLTL formula φ
- trace set T

a trace t is (T, φ)-redundant if

$$T \text{ is a model of } \varphi \quad \text{if and only if} \quad T \cup \{t\} \text{ is a model of } \varphi$$

Dominance Checking

- HyperLTL formula φ
- traces t and t'
- monitor template $\mathscr{M}_\varphi$

t' dominates t if and only if $\bigwedge_{\pi \in \mathcal{V}} \mathscr{L}(\mathscr{M}_\varphi[t'/\pi]) \subseteq \mathscr{L}(\mathscr{M}_\varphi[t/\pi])$

Storage Minimization Algorithm

input : HyperLTL formula φ, redundancy free trace set T, trace t
output : redundancy free set of traces $T_{min} \subseteq T \cup \{t\}$

$\mathscr{M}_\varphi = \texttt{build\_template}(\varphi)$

```
foreach t′ ∈ T do
    if t′ dominates t then
        return T
    end
end
foreach t′ ∈ T do
    if t dominates t′ then
        T := T \ {t′}
    end
end
return T ∪ {t}
```

Specification Analysis

Basic Idea: We use the HyperLTL-Sat solver EAHyper [Finkbeiner, H., Stenger, ’17] to check whether HyperLTL formulas are symmetric, transitive or reflexive.

- Symmetry: we omit at least half of the monitor instantiations
- Transitivity: we reduce the instantiations to two
- Reflexivity: we omit the reflexive monitor instantiation

Symmetry - Example

For observational determinism

$$\forall \pi. \forall \pi'. (O_\pi = O_{\pi'}) \, W \, (I_\pi \neq I_{\pi'})$$

we check whether the following formula is valid:

$$\forall \pi. \forall \pi'. (O_\pi = O_{\pi'}) \, W \, (I_\pi \neq I_{\pi'}) \\ \longleftrightarrow (O_{\pi'} = O_\pi) \, W \, (I_{\pi'} \neq I_\pi)$$

$\Rightarrow$ we can omit the symmetric monitor instantiations

Transitivity - Example

For output-equality

$$\forall \pi. \forall \pi'. O_\pi = O_{\pi'}$$

we check whether the following formula is valid:

$$\forall \pi. \forall \pi'. \forall \pi''. (O_\pi = O_{\pi'}) \wedge (O_{\pi'} = O_{\pi''})$$
$$\rightarrow (O_{\pi'} = O_{\pi'''})$$

$\Rightarrow$ it is sufficient to store one reference trace

Reflexivity - Example

For observational determinism

$$\forall \pi. \forall \pi'. (O_\pi = O_{\pi'}) \, W \, (I_\pi \neq I_{\pi'})$$

we check whether the following formula is valid:

$$\forall \pi. (O_\pi = O_\pi) \, W \, (I_\pi \neq I_\pi)$$

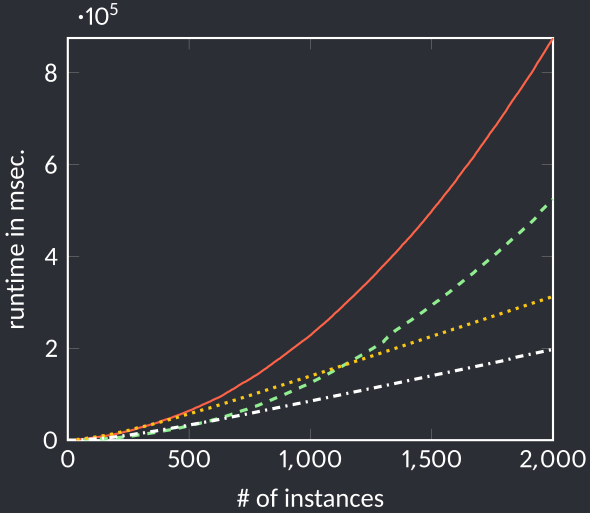
$\Rightarrow$ we can omit the reflexive monitor

Experiments

$$\forall \pi. \forall \pi'. (O_\pi = O_{\pi'}) \, W \, (I_\pi \neq I_{\pi'})$$

- naive monitoring approach
- trace analysis
- specification analysis
- combination of both

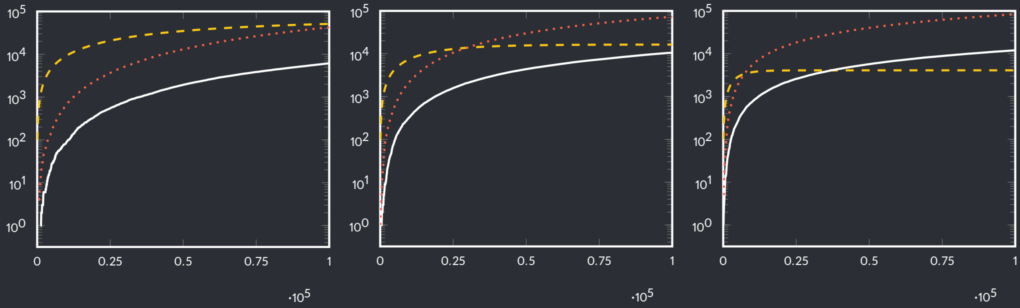
runtime on randomly generated traces



Experiments: Trace Analysis

$$\forall \pi. \forall \pi'. \square_{<n}(I_\pi = I_{\pi'}) \rightarrow \square_{<n+c}(O_\pi = O_{\pi'})$$

$n = 16$ $n = 14$ $n = 12$



- absolute numbers of violations
- number of instances stored
- number of instances pruned

10^5 randomly generated traces of length 100000

Experiments: Specification Analysis

		symm	trans	refl
ObsDet1	$\forall \pi. \forall \pi'.\ \Box(I_\pi = I_{\pi'}) \rightarrow \Box(O_\pi = O_{\pi'})$	✓	✗	✓
ObsDet2	$\forall \pi. \forall \pi'.\ (I_\pi = I_{\pi'}) \rightarrow \Box(O_\pi = O_{\pi'})$	✓	✗	✓
ObsDet3	$\forall \pi. \forall \pi'. (O_\pi = O'_\pi) \mathcal{W} (I_\pi \neq I'_\pi)$	✓	✗	✓
QuantNoninf	$\forall \pi_0 \ldots \forall \pi_c.\ \neg((\bigwedge_i I_{\pi_i} = I_{\pi_0}) \wedge \bigwedge_{i \neq j} O_{\pi_i} \neq O_{\pi_j})$	✓	✗	✓
EQ	$\forall \pi. \forall \pi'. \Box(a_\pi \leftrightarrow a_{\pi'})$	✓	✓	✓
ConfMan	$\forall \pi \forall \pi'. \big((\neg pc_\pi \wedge pc_{\pi'}) \rightarrow \bigcirc\Box(s_\pi \rightarrow \bigcirc v_{\pi'})\big)$ $\wedge \big((pc_\pi \wedge pc_{\pi'}) \rightarrow \bigcirc\Box(v_\pi \leftrightarrow v_{\pi'})\big)$	✗	✗	✗

- preprocessing can be done in a couple of seconds with EAHyper
- saves tremendous amount of time during the monitoring process

Summary

- monitoring hyperproperties in theory:

Monitor Template

Memory Explosion

- monitoring hyperproperties in practice:
  - Trace Analysis: exploits a dominance relation between traces
  - Specification Analysis: exploits symmetry, transitivity, and reflexivity in the specification

Bibliography

[Clarkson, Schneider, ’10] Clarkson, M. R., and F. B. Schneider. "Hyperproperties." Journal of Computer Security 18.6 (2010): 1157-1210.

[Clarkson, Finkbeiner, Koleini, Micinski, Rabe, Sánchez, ’14] Clarkson, M. R., Finkbeiner, B., Koleini, M., Micinski, K. K., Rabe, M. N., & Sánchez, C. (2014, April). Temporal logics for hyperproperties. In International Conference on Principles of Security and Trust (pp. 265-284).

[Finkbeiner, H., ’16] Finkbeiner, Bernd, Hahn, Christopher. Deciding hyperproperties. 27th International Conference on Concurrency Theory, CONCUR 2016

[Finkbeiner, H., Stenger, ’17] Bernd Finkbeiner, Christopher Hahn, and Marvin Stenger. EAHyper: Satisfiability, Implication, and Equivalence Checking of Hyperproperties. International Conference on Computer Aided Verification (2017).

Pictures: http://russia-insider.com/sites/insider/files/20110226_bbd001_0.jpg

Monitorability

Theorem

Given a HyperLTL formula $\varphi = \forall \pi_1 \ldots \forall \pi_k.\psi$, where $\psi \not\equiv true$ is an LTL formula. φ is monitorable if, and only if, $\forall u \in \Sigma_{\mathcal{V}}^*.\exists v \in \Sigma_{\mathcal{V}}^*.uv \in bad(\mathcal{L}(\psi))$.

Theorem

Given an alternation-free HyperLTL formula φ. Deciding whether φ is monitorable is PSpace*-complete.*

Finite Trace Semantics

$$t[i,j] = \begin{cases} \epsilon & \text{if } i \geq |t| \\ t[i, min(j, |t|-1)], & \text{otherwise} \end{cases}$$

$\Pi_{fin} \models_T a_\pi$	if $a \in \Pi_{fin}(\pi)[0]$
$\Pi_{fin} \models_T \neg\varphi$	if $\Pi_{fin} \not\models_T \varphi$
$\Pi_{fin} \models_T \varphi \vee \psi$	if $\Pi_{fin} \models_T \varphi$ or $\Pi_{fin} \models_T \psi$
$\Pi_{fin} \models_T \bigcirc\varphi$	if $\Pi_{fin}[1,\ldots] \models_T \varphi$
$\Pi_{fin} \models_T \varphi\, U\, \psi$	if $\exists i \geq 0. \Pi_{fin}[i,\ldots] \models_T \psi \wedge \forall 0 \leq j < i. \Pi_{fin}[j,\ldots] \models_T \varphi$
$\Pi_{fin} \models_T \exists\pi.\varphi$	if there is some $t \in T$ such that $\Pi_{fin}[\pi \mapsto t] \models_T \varphi$

Alternation

An offline monitor for a $\forall^n\exists^m$HyperLTL and $\exists^m\forall^n$HyperLTL formula has to perform the checks

$$\bigwedge_{N\in T^n} \bigvee_{M\in T^m} \text{check if } \mathscr{M}_\varphi \text{ accepts } N \times M \text{ , and}$$

$$\bigvee_{M\in T^m} \bigwedge_{N\in T^n} \text{check if } \mathscr{M}_\varphi \text{ accepts } M \times N \text{ , respectively.}$$