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:
  \[ \exists \pi. \exists \pi'. \Box on_\pi \land \Box \neg on_{\pi'} \]
  satisfiable by \( \{ \{ on \}^\omega, \{ off \}^\omega \} \)

- trace equality:
  \[ \forall \pi. \forall \pi'. \Box (on_\pi \leftrightarrow on_{\pi'}) \]

- observational determinism:
  \[ \forall \pi. \forall \pi'. (O_\pi = O_{\pi'}) \land (l_\pi \neq l_{\pi'}) \]
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 $v$

\[
\forall \pi. \forall \pi'. (\neg pc_\pi \land pc_{\pi'}) \rightarrow \Box (s_\pi \rightarrow \Box v_{\pi'}) \land \\
(pc_\pi \land pc_{\pi'}) \rightarrow \Box (v_\pi \leftrightarrow v_{\pi'})
\]
\( \forall \pi. \forall \pi'. (\neg pc_\pi \land pc_{\pi'}) \rightarrow \square \square (s_\pi \rightarrow \square v_{\pi'}) \land \\
(pc_\pi \land pc_{\pi'}) \rightarrow \square \square (v_\pi \leftrightarrow v_{\pi'}) \)

\[\downarrow\]

\[
\begin{array}{c}
q_0 \\
\rightarrow \neg pc_\pi \land pc_{\pi'} \\
\rightarrow pc_\pi \land pc_{\pi'} \\
\rightarrow \neg pc_{\pi'} \\
\rightarrow p_\pi \leftrightarrow v_{\pi'} \\
q_1 \\
\rightarrow v_{\pi'} \land s_\pi \\
\rightarrow s_\pi \land \neg s_\pi \\
q_2 \\
\rightarrow \neg pc_{\pi'} \\
\rightarrow p_\pi \land pc_{\pi'} \\
\rightarrow T \\
q_3 \\
\rightarrow v_\pi \leftrightarrow v_{\pi'} \\
q_4 \\
\rightarrow v_{\pi'} \land s_\pi \\
\rightarrow s_\pi \\
\end{array}
\]
Monitor Construction

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

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

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

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

Deterministic monitor template $\mathcal{M} = (\Sigma, Q, \delta, q_0)$:
- finite alphabet $\Sigma = 2^{AP \times \nu}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

![Diagram of monitor construction](image)
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!
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!
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!
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

an author submits a paper

another author submits a paper
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
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

\begin{array}{|c|c|c|c|c|c|}
\hline
{} & \{pc\} & \{v\} & \{v\} & \{v\} & \{v\} \\
\hline
\end{array}

*a pc member observes three submissions*
Trace Analysis - Example

A PC member observes three submissions

A PC member observes two submissions
Definition (Trace Redundancy)

- HyperLTL formula $\varphi$
- trace set $T$

A trace $t$ is $(T, \varphi)$-redundant if $T$ is a model of $\varphi$ if and only if $T \cup \{t\}$ is a model of $\varphi$. 
Dominance Checking

• HyperLTL formula $\varphi$
• traces $t$ and $t'$
• monitor template $M_\varphi$

$t'$ dominates $t$ if and only if $\bigwedge_{\pi \in \mathcal{V}} L(M_\varphi[t'/\pi]) \subseteq L(M_\varphi[t/\pi])$
Storage Minimization Algorithm

**input**: HyperLTL formula $\varphi$, redundancy free trace set $T$, trace $t$

**output**: redundancy free set of traces $T_{min} \subseteq T \cup \{t\}$

$M_\varphi = \text{build_template}(\varphi)$

**foreach** $t' \in T$ **do**

**if** $t'$ dominates $t$ **then**

**return** $T$

**end**

**end**

**foreach** $t' \in T$ **do**

**if** $t$ dominates $t'$ **then**

$T := T \setminus \{t'\}$

**end**

**end**

**return** $T \cup \{t\}$
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

∀ \pi. \forall \pi'. (O_\pi = O_{\pi'}) \land (I_\pi \neq I_{\pi'})

we check whether the following formula is valid:

∀ \pi. \forall \pi'. (O_\pi = O_{\pi'}) \land (I_\pi \neq I_{\pi'})

⇔ (O_{\pi'} = O_\pi) \land (I_{\pi'} \neq I_\pi)

⇒ 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:

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

⇒ it is sufficient to store one reference trace
Re/uni FB02 exivity - 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:

\[ \forall \pi. (O_\pi = O_{\pi}) W (I_\pi \neq I_{\pi}) \]

⇒ we can omit the reflexive monitor
Experiments

∀π. ∀π'. (O_π = O_{π'}) \land (I_π \neq I_{π'})

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

runtime on randomly generated traces
Experiments: Trace Analysis

\[ \forall \pi. \forall \pi'. \square_{<n}(l_\pi = l_{\pi'}) \rightarrow \square_{<n+c}(O_\pi = O_{\pi'}) \]

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

10^5 randomly generated traces of length 100000
Experiments: Specification Analysis

<table>
<thead>
<tr>
<th></th>
<th>ObsDet1</th>
<th>ObsDet2</th>
<th>ObsDet3</th>
<th>QuantNoninf</th>
<th>EQ</th>
<th>ConfMan</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObsDet1</td>
<td>$\forall \pi. \forall \pi' . \Box (I_\pi = I_{\pi'}) \rightarrow \Box (O_\pi = O_{\pi'})$</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>ObsDet2</td>
<td>$\forall \pi. \forall \pi' . (I_\pi = I_{\pi'}) \rightarrow \Box (O_\pi = O_{\pi'})$</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
| ObsDet3              | $\forall \pi. \forall \pi' . (O_\pi = O_{\pi'}) \not\!
\not\!(I_\pi \neq I_{\pi'})$ | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ |
| QuantNoninf          | $\forall \pi_0 \ldots \forall \pi_c . \neg ((\bigwedge_i I_{\pi_i} = I_{\pi_0}) \land \bigwedge_{i \neq j} O_{\pi_i} \neq O_{\pi_j})$ | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ |
| EQ                   | $\forall \pi. \forall \pi' . \Box (a_\pi \leftrightarrow a_{\pi'})$ | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ |
| ConfMan              | $\forall \pi. \forall \pi' . ((\neg pc_\pi \land pc_{\pi'}) \rightarrow \Box (s_\pi \rightarrow O_{\pi'}))$ | ❌ | ❌ | ❌ | ✔️ | ✔️ |

- 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

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


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

Theorem

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

Theorem

Given an alternation-free HyperLTL formula $\phi$. Deciding whether $\phi$ is monitorable is PSpace-complete.
Finite Trace Semantics

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

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

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

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

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