Verifying Hyperproperties of Hardware Systems

Bernd Finkbeiner
Saarland University

Markus N. Rabe
UC Berkeley

based on joint work with Michael R. Clarkson, Christopher Hahn, Masoud Koleini, Kristopher K. Micinski, and César Sánchez

FMCAD’16 Tutorial
Mountain View
October 3, 2016
Data leak at Yahoo: 500 million accounts stolen

For months there were rumors about a data breach at Yahoo. Thursday, the Internet company announced that data from 500 million users were stolen. It thus seems to be the biggest data breach ever.

The cyber-attack would have been done at the end of 2014. The stolen data includes email addresses, phone numbers, dates of birth, encrypted passwords, and in some cases security
Major Incidents in Information Security

- **Heartbleed**: 4.5m patient records leaked
  
  ```
  if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0;
  ```

- **Goto Fail**: encryption of >300M devices broken
  
  ```
  if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
  ```

- **Shellshock**: web servers attackable for 22 years
  
  ```
  parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
  ```
Embedded Systems / Hardware Security
Information-flow control

Public output should only depend on public input.

Typical information-flow property: Noninterference

$$\forall t, t' \in Traces(K) : t = I_{public} t' \Rightarrow t = O_{public} t'$$
Hyperproperties

Clarkson & Schneider ’10:
Hyperproperty $H$: a set of sets of traces

System $K$ satisfies $H$ iff $\text{Traces}(K) \in H$. 
Hyperproperties

Clarkson & Schneider’10:
Hyperproperty $H$: a set of sets of traces

System $K$ satisfies $H$ iff $\text{Traces}(K) \in H$.

Many information-flow properties can be formalized as hyperproperties.

Noninterference as hyperproperty:

$$\{ T \subseteq 2^{\text{Traces}} | \forall t, t' \in T : t =_{I_{\text{public}}} t' \Rightarrow t =_{O_{\text{public}}} t' \}$$
Case Study 1: Information flow in the I2C Bus

- Under which circumstances can information flow from the inputs through the Master to the bus (and vice versa)?
- Is there an expiration date for information?
Case study 2: Symmetry in Protocols

```plaintext
while (true) {
(1) choosing[i] = true;
(2) number[i] = max(number)+1;
(3) choosing[i] = false;
(4) for (int j=0; j < n; j++) {
   (5) while (choosing[j]) { ; }
   (6) while (j != i ∧ number[j] ≠ 0 ∧ (number[j],j) < (number[i],i)) { ; }
}
(7) critical
(8) number[i] = 0;
(9) non-critical
}
```

- Are the clients treated symmetrically?
Case study 3: Error-resistant codes

Different encoders from OpenCores.org.

- 8bit-10bit encoder, decoder
- Huffman encoder
- Hamming encoder

- Do codes for distinct inputs have at least Hamming distance $d$?
Automatic analysis techniques

- Security type systems
- Program analysis
- Dynamic approaches/taint tracking

Common problem: single-property techniques
Automatic analysis techniques

- Security type systems
- Program analysis
- Dynamic approaches/taint tracking

Common problem: single-property techniques

This tutorial: A unifying framework for the analysis of hyperproperties
Overview

I  HyperLTL
II  Examples
III  Model Checking
IV  Satisfiability
V  Beyond HyperLTL
Part I

HyperLTL
Temporal logics for information security?

**LTL:** Specifies computations
Example: $FG \ x = 0$  
“from some point on $x$ is 0”

**CTL/CTL***: Specifies computation trees
Example: $AG EF \ x = 0$  
“$x$ may always become 0 in the future”
A Simple Information-flow Policy

“All executions have the light on at the same time.”

“For all pairs of executions and all points in time, the light is on on the one execution iff it is on on the other execution.”

Information flow properties compare multiple executions!
LTL?

Syntax: \( \varphi ::= a_\pi \mid X\psi \mid G\psi \mid F\psi \mid \psi U \psi \mid \psi W \psi \)

Semantics: \( K \models \varphi \) iff \( Traces(K) \subseteq Traces(\varphi) \)

“All executions have the light on at the same time.”
CTL*?

Syntax: \( \varphi ::= a \mid A\varphi \mid E\varphi \mid X\varphi \mid G\varphi \mid \varphi U \varphi \mid \ldots \)

Semantics: \( K \models A\varphi \iff \text{for all } p \in \text{Paths}(K) : p \models \varphi \)

“All executions have the light on at the same time.” AA\(\varphi\)?
CTL*?

Syntax: \( \varphi ::= a \mid A\varphi \mid E\varphi \mid X\varphi \mid G\varphi \mid \varphi U \varphi \mid \ldots \)

Semantics: \( K \models A\varphi \) iff for all \( p \in \text{Paths}(K) : p \models \varphi \)

“All executions have the light on at the same time.” \( AA\varphi \)
CTL*?

Syntax: $\varphi ::= a \mid A\varphi \mid E\varphi \mid X\varphi \mid G\varphi \mid \varphi U \varphi \mid \ldots$

Semantics: $K \models A\varphi$ iff for all $p \in \text{Paths}(K): p \models \varphi$

“All executions have the light on at the same time.” $\AA\varphi$
HyperLTL

Quantifiers with trace variables: \( \forall \pi. \varphi \quad \exists \pi. \varphi \)

Syntax: \[ \varphi ::= \forall \pi. \varphi \mid \exists \pi. \varphi \mid \psi \]
\[ \psi ::= a_\pi \mid X \psi \mid G \psi \mid F \psi \mid \psi U \psi \mid \psi W \psi \]

HyperLTL: Start with a quantifier prefix, then quantifier-free.
HyperLTL

Quantifiers with **trace variables**: \( \forall \pi . \varphi \quad \exists \pi . \varphi \)

**Syntax:**
\[
\varphi ::= \forall \pi . \varphi \mid \exists \pi . \varphi \mid \psi
\]
\[
\psi ::= a_\pi \mid X \psi \mid G \psi \mid F \psi \mid \psi U \psi \mid \psi W \psi
\]

HyperLTL: Start with a quantifier prefix, then quantifier-free

“All executions have the light on at the same time.”

\[
\forall \pi . \forall \pi'. G(\text{on}_\pi \leftrightarrow \text{on}_{\pi'})
\]
HyperLTL

Quantifiers with trace variables: \( \forall \pi. \varphi \quad \exists \pi. \varphi \)

Syntax:
\[\varphi ::= \forall \pi. \varphi \mid \exists \pi. \varphi \mid \psi \]
\[\psi ::= a_\pi \mid X\psi \mid G\psi \mid F\psi \mid \psi U \psi \mid \psi W \psi\]

HyperLTL: Start with a quantifier prefix, then quantifier-free

“All executions have the light on at the same time.”

\[\forall \pi. \forall \pi'. \, G(\text{on}_\pi \leftrightarrow \text{on}_\pi')\]
HyperLTL

Quantifiers with trace variables: \( \forall \pi. \varphi \quad \exists \pi. \varphi \)

Syntax:

\[ \varphi ::= \forall \pi. \varphi \mid \exists \pi. \varphi \mid \psi \]
\[ \psi ::= a_\pi \mid X\psi \mid G\psi \mid F\psi \mid \psi U \psi \mid \psi W \psi \]

HyperLTL: Start with a quantifier prefix, then quantifier-free

“All executions have the light on at the same time.”

\( \forall \pi. \forall \pi'. G(\text{on}_\pi \leftrightarrow \text{on}_{\pi'}) \)
Semantics

\[ \Pi \models_T a_{\pi} \iff a \in \Pi(\pi)(0) \]

\[ \Pi \models_T G \varphi \iff \forall i \geq 0 : \Pi[i, \infty] \models_T \varphi \]

\[ \Pi \models_T \forall \pi. \varphi \iff \forall t \in T : \Pi[\pi \mapsto t] \models_T \varphi \]

Semantics given with respect to a set of traces \( T \) and a trace environment \( \Pi : \text{Vars} \rightarrow T \).

A Kripke structure \( K \) satisfies a HyperLTL formula \( \varphi \) iff \( \emptyset \models_{\text{Traces}(K)} \varphi \)
Semantics

\[ \Pi \models_T a_\pi \iff a \in \Pi(\pi)(0) \]
\[ \Pi \models_T G \varphi \iff \forall i \geq 0 : \Pi[i, \infty] \models_T \varphi \]
\[ \Pi \models_T \forall \pi. \varphi \iff \forall t \in T : \Pi[\pi \mapsto t] \models_T \varphi \]

“All executions have the light on at the same time.”
Semantics

\[ \Pi \models_T a_\pi \iff a \in \Pi(\pi)(0) \]
\[ \Pi \models_T G\varphi \iff \forall i \geq 0 : \Pi[i, \infty] \models_T \varphi \]
\[ \Pi \models_T \forall \pi. \varphi \iff \forall t \in T : \Pi[\pi \mapsto t] \models_T \varphi \]

“All executions have the light on at the same time.”

1. \( \{ \pi \mapsto p \} \models_K \forall \pi'. G(\text{on}_\pi \leftrightarrow \text{on}_{\pi'}) \)
Semantics

\[ \Pi \models_T a_\pi \iff a \in \Pi(\pi)(0) \]
\[ \Pi \models_T G \varphi \iff \forall i \geq 0 : \Pi[i, \infty] \models_T \varphi \]
\[ \Pi \models_T \forall \pi. \varphi \iff \forall t \in T : \Pi[\pi \mapsto t] \models_T \varphi \]

“All executions have the light on at the same time.”

1. \( \{ \pi \mapsto p \} \models_K \forall \pi'. G(\text{on}_\pi \leftrightarrow \text{on}_{\pi'}) \)
2. \( \{ \pi \mapsto p, \pi' \mapsto p' \} \models_K G(\text{on}_\pi \leftrightarrow \text{on}_{\pi'}) \)
Semantics

\[
\begin{align*}
\Pi &\models_T a_\pi \iff a \in \Pi(\pi)(0) \\
\Pi &\models_T G \varphi \iff \forall i \geq 0 : \Pi[i, \infty] \models_T \varphi \\
\Pi &\models_T \forall \pi. \varphi \iff \forall t \in T : \Pi[\pi \mapsto t] \models_T \varphi
\end{align*}
\]

“All executions have the light on at the same time.”

1. \(\{\pi \mapsto p\} \models_K \forall \pi'. G (on_\pi \leftrightarrow on_{\pi'})\)
2. \(\{\pi \mapsto p, \pi' \mapsto p'\} \models_K G (on_\pi \leftrightarrow on_{\pi'})\)
3. \(\forall i \in \mathbb{N} : \{\pi \mapsto p[i, \infty], \pi' \mapsto p'[i, \infty]\} \models_K on_\pi \leftrightarrow on_{\pi'}\)
Full Semantics

\[ \Pi \models_T a_\pi \iff a \in \Pi(\pi)(0) \]
\[ \Pi \models_T G \varphi \iff \forall i \geq 0 : \Pi[i, \infty] \models_T \varphi \]
\[ \Pi \models_T \forall \pi. \varphi \iff \forall t \in T : \Pi[\pi \mapsto t] \models_T \varphi \]
\[ \Pi \models_T \neg \varphi \iff \Pi \not\models_T \varphi \]
\[ \Pi \models_T \varphi_1 \lor \varphi_2 \iff \Pi \models_T \varphi_1 \text{ or } \Pi \models_T \varphi_2 \]
\[ \Pi \models_T X \varphi \iff \Pi[1, \infty] \models_T \varphi \]
\[ \Pi \models_T F \varphi \iff \exists i \geq 0 : \Pi[i, \infty] \models_T \varphi \]
\[ \Pi \models_T \varphi_1 \lor \varphi_2 \iff \text{there exists } i \geq 0 : \Pi[i, \infty] \models_T \varphi_2 \]
\[ \text{and for all } 0 \leq j < i \text{ we have } \Pi[j, \infty] \models_T \varphi_1 \]
\[ \Pi \models_T \varphi_1 W \varphi_2 \iff \Pi \models_T \varphi_1 \lor \varphi_2 \text{ or } \Pi \models_T G \varphi_1 \]
Part II

Examples
Case Study 1: Information flow in the I2C Bus

▶ Under which circumstances can information flow from the inputs through the Master to the bus?

$$\forall \pi. \forall \pi'. G(\overline{\text{DAT}}_{\pi} = \overline{\text{DAT}}_{\pi'}) \Rightarrow G(\text{SDA}_{\pi} = \text{SDA}_{\pi'})$$

$p_{\pi} = p_{\pi'}$, is defined as $\land_{a \in P} a_{\pi} \leftrightarrow a_{\pi'}$.

$p_{\pi} = p_{\pi'}$, is defined as $(I \setminus P)_{\pi} = (I \setminus P)_{\pi'}$. 
Case Study 1: Information flow in the I2C Bus

Under which circumstances can information flow from the inputs through the Master to the bus?

\[ \forall \pi. \forall \pi'. \ G(\neg \text{WE}_\pi \land \overline{\text{DAT}}_\pi = \overline{\text{DAT}}_{\pi'}) \Rightarrow G(\text{SDA}_\pi = \text{SDA}_{\pi'}) \]

\[ P_\pi = P_{\pi'} \text{ is defined as } \bigwedge_{a \in P} a_\pi \leftrightarrow a_{\pi'} . \]

\[ \overline{P}_\pi = \overline{P}_{\pi'} \text{ is defined as } (I \setminus P)_\pi = (I \setminus P)_{\pi'} . \]
Case Study 1: Information flow in the I2C Bus

- Is there an expiration date for information?

\[ \forall \pi. \forall \pi'. (\overline{\text{DAT}_\pi} = \overline{\text{DAT}_{\pi'}}, \ U \ G (l_\pi = l_{\pi'}) \Rightarrow F G (\text{SDA}_\pi = \text{SDA}_{\pi'})) \]

\( P_\pi = P_{\pi'} \) is defined as \( \land_{a \in P} a_\pi \leftrightarrow a_{\pi'} \).

\( \overline{P}_\pi = \overline{P}_{\pi'} \) is defined as \( (l \setminus P)_\pi = (l \setminus P)_{\pi'} \).
Variants of noninterference in HyperLTL

- Observational determinism [Zdancewich & Myers’03]
  \[ \forall \pi. \forall \pi'. \text{lowIn}_\pi = \text{lowIn}_{\pi'} \Rightarrow G(\text{lowOut}_\pi = \text{lowOut}_{\pi'}) \]

- Generalized noninterference [McCullough’88]
  \[ \forall \pi. \forall \pi'. \exists \pi''. G(\text{highIn}_\pi = \text{highIn}_{\pi''}) \land G(\text{lowIn}_{\pi'} = \text{lowIn}_{\pi''} \land \text{lowOut}_{\pi'} = \text{lowOut}_{\pi''}) \]

- Noninference [McLean’94]
  \[ \forall \pi. \exists \pi'. G(\text{highIn}_{\pi'}) \land G(\text{lowIn}_\pi = \text{lowIn}_{\pi'} \land \text{lowOut}_\pi = \text{lowOut}_{\pi'}) \]
Case study 2: Symmetry in Protocols

```latex
\textbf{while} (true) \{
(1) \hspace{1em} \text{choosing}[i] = true;
(2) \hspace{1em} \text{number}[i] = \max(\text{number})+1;
(3) \hspace{1em} \text{choosing}[i] = false;
(4) \hspace{1em} \textbf{for} (\text{int } j=0; j < n; j++) \{
(5) \hspace{2em} \textbf{while} (\text{choosing}[j]) \{ \, ; \, \}
(6) \hspace{2em} \textbf{while} (j \neq i \land \text{number}[j] \neq 0 \land (\text{number}[j],j) < (\text{number}[i],i)) \{ \, ; \, \}
\}
(7) \hspace{1em} \textbf{critical}
(8) \hspace{1em} \text{number}[i] = 0;
(9) \hspace{1em} \textbf{non-critical}
\}
```

- Are the clients treated symmetrically?

\[
\forall \pi. \forall \pi'. \ G(\text{select}_0_\pi \leftrightarrow \text{select}_1_\pi', \land \text{select}_1_\pi \leftrightarrow \text{select}_0_\pi') \Rightarrow \ G(\text{critical}_0_\pi \leftrightarrow \text{critical}_1_\pi', \land \text{critical}_1_\pi \leftrightarrow \text{critical}_0_\pi')
\]
Case study 2: Symmetry in Protocols

\[\textbf{while (true) \{}\]
\begin{enumerate}
    \item choosing\([i]\) = \textit{true};
    \item number\([i]\) = max(number) + 1;
    \item choosing\([i]\) = \textit{false};
    \item \textbf{for (int } j=0; j < n; j++) \{} \\
        \item while (choosing\([j]\)) \{} ; \} \\
        \item while (j \neq i \land \text{number}[j] \neq 0 \land \\
            f \land (\text{number}[j], j) < (\text{number}[i], i) \\
            \lor \\
            \neg f \land (\text{number}[j], i) < (\text{number}[i], j) \} ; \}
    \item \textbf{critical}
    \item number\([i]\) = 0;
    \item \textbf{non-critical}
\end{enumerate}

\[\forall \pi \forall \pi'. G (select_0_\pi \leftrightarrow select_1_\pi, \land select_1_\pi \leftrightarrow select_0_\pi, \land \pi \leftrightarrow \neg \pi') \Rightarrow \\
G (critical_0_\pi \leftrightarrow critical_1_\pi, \land critical_1_\pi \leftrightarrow critical_0_\pi)\]
Case study 3: Error-resistant codes

Different encoders from OpenCores.org.

- 8bit-10bit encoder, decoder
- Huffman encoder
- Hamming encoder

- Do codes for distinct inputs have at least Hamming distance $d$?

\[ \forall \pi, \forall \pi'. F(DAT_\pi \neq DAT_{\pi'}) \Rightarrow \neg \text{Ham}(d, \pi, \pi') \]

where we define:

\[
\text{Ham}(0, \pi, \pi') = \text{false}
\]
\[
\text{Ham}(d, \pi, \pi') = o_\pi = o_{\pi'} \land (o_\pi \neq o_{\pi'} \land X \text{Ham}(d - 1, \pi, \pi'))
\]
Part III

Model Checking
Model Checking Alternation-free HyperLTL

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of sets of propositions.
- Projection handles quantifiers.
Model Checking Alternation-free HyperLTL

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of sets of propositions.
- Projection handles quantifiers.

Example: $\forall \pi. \forall \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \rightarrow G(o_\pi \leftrightarrow o_{\pi'})$
Model Checking Alternation-free HyperLTL

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of sets of propositions.
- Projection handles quantifiers.

Example: $\forall \pi. \forall \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \rightarrow G(o_\pi \leftrightarrow o_{\pi'})$

Negated: $\exists \pi. \exists \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \land F(o_\pi \not\leftrightarrow o_{\pi'})$
Model Checking Alternation-free HyperLTL

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of sets of propositions.
- Projection handles quantifiers.

Example: $\forall \pi. \forall \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \rightarrow G(o_\pi \leftrightarrow o_{\pi'})$

Negated: $\exists \pi. \exists \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \land F(o_\pi \not\leftrightarrow o_{\pi'})$

$\mathcal{A}$ with alphabet $2^{AP} \times 2^{AP}$
Model Checking Alternation-free HyperLTL

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of sets of propositions.
- Projection handles quantifiers.

Example: $\forall \pi. \forall \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \rightarrow G(o_\pi \leftrightarrow o_{\pi'})$

Negated: $\exists \pi. \exists \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \land F(o_\pi \not\leftrightarrow o_{\pi'})$

$\mathcal{A}$ with alphabet $2^{AP} \times 2^{AP}$

$\mathcal{A}'$ with alphabet $2^{AP}$
Model Checking Alternation-free HyperLTL

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of sets of propositions.
- Projection handles quantifiers.

Example: $\forall \pi. \forall \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \rightarrow G(o_\pi \leftrightarrow o_{\pi'})$

Negated: $\exists \pi. \exists \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \land \neg F(o_\pi \leftrightarrow o_{\pi'})$

$\mathcal{A}$ with alphabet $2^{\text{AP}} \times 2^{\text{AP}}$

$\mathcal{A}'$ with alphabet $2^{\text{AP}}$

$\mathcal{A}'''$ with 1-letter alphabet
Model Checking General HyperLTL

Complexity depends on the quantifier alternation depth.

1. $\forall \pi. \forall \pi'. \psi$  
   - PSPACE in $|\psi|$,  
   - NLOGSPACE in $|K|$

   ▶ Observational determinism
Model Checking General HyperLTL

Complexity depends on the quantifier alternation depth.

0. $\forall \pi. \forall \pi'. \psi$ \quad PSPACE in $|\psi|$, \quad NLOGSPACE in $|K|$
   - Observational determinism

1. $\forall \pi. \exists \pi'. \psi$ \quad EXPSPACE in $|\psi|$, \quad PSPACE in $|K|$
   - Noninference
   - Generalized noninterference

2. ...

Rarely need more than one quantifier alternation!
Symbolic Model Checking for Circuits

- Alternation-free HyperLTL
- Clean extension of the circuit construction for LTL
- Leverages existing symbolic model checkers (e.g. ABC)
Reduction to Safety Property on Circuits

\[ \forall \pi. \forall \pi'. \ G(i_\pi \leftrightarrow i_{\pi'}) \Rightarrow G(o_\pi \leftrightarrow o_{\pi'}) \]
Reduction to Safety Property on Circuits

\[ \forall \pi. \forall \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \Rightarrow G(o_\pi \leftrightarrow o_{\pi'}) \]

Negated: \[ \exists \pi. \exists \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \land F(o_\pi \not\leftrightarrow o_{\pi'}) \]

safety/liveness violation
Reduction to Safety Property on Circuits

\[ \forall \pi. \forall \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \Rightarrow G(o_\pi \leftrightarrow o_{\pi'}) \]

Negated: \[ \exists \pi. \exists \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \land F(o_\pi \not\leftrightarrow o_{\pi'}) \]

safety/liveness violation

\[ G(i_\pi \leftrightarrow i_{\pi'}) \land F(o_\pi \not\leftrightarrow o_{\pi'}) \]

system
Reduction to Safety Property on Circuits

\[ \forall \pi. \forall \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \Rightarrow G(o_\pi \leftrightarrow o_{\pi'}) \]

Negated: \( \exists \pi. \exists \pi'. G(i_\pi \leftrightarrow i_{\pi'}) \land F(o_\pi \not\leftrightarrow o_{\pi'}) \)

\textit{safety/liveness violation}

\[
G(i_\pi \leftrightarrow i_{\pi'}) \land \\
F(o_\pi \not\leftrightarrow o_{\pi'})
\]

system

i, o

\[
i, i', o'
\]

\[i, i'
\]
Implementation - MCHyper

A transformation on Aiger circuits

Workflow

1. Convert VHDL/Verilog to Aiger
2. Run MCHyper with a formula and the circuit
3. Call a hardware model checker on the resulting circuit

Tool website:
https://www.react.uni-saarland.de/tools/mchyper/
An Example Circuit

∀π.∀π'. G(reset_π ↔ reset_π') ⇒ G(Q_π ↔ Q_π')
An Example Circuit

∀π.∀π’. G(reset_π ↔ reset_π’) ⇒ G(Q_π ↔ Q_π’)

[Diagram of an example circuit with labeled nodes and edges representing the circuit's logic and interactions.]
## Information flow - Experimental Data

<table>
<thead>
<tr>
<th>Model</th>
<th>#Latches</th>
<th>#Gates</th>
<th>IC3</th>
<th>INT</th>
<th>BMC</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF1</td>
<td>(NI1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times)</td>
</tr>
<tr>
<td>IF2</td>
<td>(NI2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times)</td>
</tr>
<tr>
<td>IF3</td>
<td>(NI3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times)</td>
</tr>
<tr>
<td>IF4</td>
<td>(NI4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\checkmark)</td>
</tr>
<tr>
<td>IF5</td>
<td>(NI5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times)</td>
</tr>
<tr>
<td>IF6</td>
<td>(NI6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times)</td>
</tr>
<tr>
<td>IF7</td>
<td>(NI7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times)</td>
</tr>
<tr>
<td>IF8</td>
<td>(NI8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times)</td>
</tr>
<tr>
<td>IF9</td>
<td>(NI2')</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times)</td>
</tr>
</tbody>
</table>

Model: I2C Master

Verification time in s:

- IC3:
  - IF1: 95.17
  - IF2: 53.08
  - IF3: 168.96
  - IF4: 438.41
  - IF5: 717.74
  - IF6: 186.20
  - IF7: TO
  - IF8: 1557.14
  - IF9: TO

- INT:
  - IF1: 1.13
  - IF2: 1.16
  - IF3: 1.38
  - IF4: 1.01
  - IF5: 8.31
  - IF6: 1.10
  - IF7: 6.82
  - IF8: 2.92
  - IF9: 155.77

- BMC:
  - IF1: 0.07
  - IF2: 0.08
  - IF3: -
  - IF4: 0.99
  - IF5: 0.77
  - IF6: 0.07
  - IF7: 0.55
  - IF8: 0.16
  - IF9: 6.27

Result:
- \(\times\) for failed verification
- \(\checkmark\) for successful verification
### Symmetry in Protocols - Experimental Data

<table>
<thead>
<tr>
<th>Model</th>
<th>#Latches</th>
<th>#Gates</th>
<th>IC3</th>
<th>INT</th>
<th>BMC</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym1</td>
<td>(S1)</td>
<td>Bakery</td>
<td>46</td>
<td>1829</td>
<td>6.34</td>
<td>0.88</td>
</tr>
<tr>
<td>Sym2</td>
<td>(S2)</td>
<td>Bakery</td>
<td>47</td>
<td>1588</td>
<td>69.12</td>
<td>TO</td>
</tr>
<tr>
<td>Sym3</td>
<td>(S3)</td>
<td>Bakery.a</td>
<td>47</td>
<td>1618</td>
<td>26.31</td>
<td>4.75</td>
</tr>
<tr>
<td>Sym4</td>
<td>(S3)</td>
<td>Bakery.a.n</td>
<td>47</td>
<td>1532</td>
<td>66.41</td>
<td>TO</td>
</tr>
<tr>
<td>Sym5</td>
<td>(S4)</td>
<td>Bakery.a.n.s</td>
<td>47</td>
<td>1532</td>
<td>16.83</td>
<td>TO</td>
</tr>
<tr>
<td>Sym6</td>
<td>(S5)</td>
<td>Bakery.a.n.s.5proc</td>
<td>90</td>
<td>3762</td>
<td>97.45</td>
<td>TO</td>
</tr>
<tr>
<td>Sym7</td>
<td>(S7)</td>
<td>Bakery.a.n.s.7proc</td>
<td>136</td>
<td>6775</td>
<td>312.53*</td>
<td>TO</td>
</tr>
</tbody>
</table>
## Error Correcting Codes - Experimental Data

<table>
<thead>
<tr>
<th>Model</th>
<th>#Latches</th>
<th>#Gates</th>
<th>Verification time in s</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huff1 (HD1)</td>
<td>Huffman_enc</td>
<td>19</td>
<td>3.08 37.19 -</td>
<td>✓</td>
</tr>
<tr>
<td>Huff2 (HD2)</td>
<td>571</td>
<td></td>
<td>0.62 0.09 0.02</td>
<td>×</td>
</tr>
<tr>
<td>8b10b_1 (HD1)</td>
<td>8b10b_enc</td>
<td>39</td>
<td>0.32 0.09 0.02</td>
<td>×</td>
</tr>
<tr>
<td>8b10b_2 (HD1')</td>
<td>271</td>
<td></td>
<td>1.19 9.06 -</td>
<td>✓</td>
</tr>
<tr>
<td>8b10b_3 (HD2')</td>
<td>0.03 0.04 0.02</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8b10b_4 (HD1'')</td>
<td>8b10b_dec</td>
<td>19</td>
<td>0.05 0.09 -</td>
<td>✓</td>
</tr>
<tr>
<td>Hamm1 (HD1_1)</td>
<td>Hamming_enc</td>
<td>27</td>
<td>0.02 0.04 0.02</td>
<td>×</td>
</tr>
<tr>
<td>Hamm2 (HD1_2)</td>
<td></td>
<td></td>
<td>0.02 0.03 0.02</td>
<td>×</td>
</tr>
<tr>
<td>Hamm3 (HD1_3)</td>
<td></td>
<td></td>
<td>0.03 0.04 0.02</td>
<td>×</td>
</tr>
<tr>
<td>Hamm3' (HD1_3')</td>
<td></td>
<td></td>
<td>7.34 18.0 0.18</td>
<td>-</td>
</tr>
<tr>
<td>Hamm4 (HD1_4)</td>
<td></td>
<td></td>
<td>66.93 0.10 -</td>
<td>✓</td>
</tr>
<tr>
<td>Hamm5 (HD2_1)</td>
<td></td>
<td></td>
<td>11.83 1.31 -</td>
<td>✓</td>
</tr>
<tr>
<td>Hamm6 (HD2_2)</td>
<td></td>
<td></td>
<td>14.44 0.78 -</td>
<td>✓</td>
</tr>
<tr>
<td>Hamm7 (HD3)</td>
<td></td>
<td></td>
<td>12.23 1.25 -</td>
<td>✓</td>
</tr>
</tbody>
</table>
Part IV

Satisfiability
Satisfiability of HyperLTL

HyperLTL-SAT is the problem to decide whether there exists a non-empty trace set $T$ satisfying a HyperLTL formula $\varphi$.

Application: Two versions of Observational Determinism:

- $\forall \pi. \forall \pi'. G(l_{\pi} = l_{\pi'}) \rightarrow G(O_{\pi} = O_{\pi'})$
- $\forall \pi. \forall \pi'. (O_{\pi} = O_{\pi'}) W (l_{\pi} \neq l_{\pi'})$

Which version is stronger?
Challenge

LTL Satisfiability Solving

- Translate LTL formula into Büchi automaton
- Check the automaton for emptiness
- PSPACE-complete

HyperLTL Satisfiability Solving

- A Hyperproperty is not necessarily $\omega$-regular
- Standard automata approach cannot be applied
Solving HyperLTL-SAT

1. Alternation-free fragments ($\forall^* \& \exists^*$)
2. Alternation starting with existential quantifier ($\exists^* \forall^*$)
3. Alternation starting with universal quantifier ($\forall^* \exists^*$)
Existential Fragment

Theorem
\( \exists^* \text{HyperLTL-SAT is PSPACE-complete.} \)

Example
\( \exists \pi_0 \exists \pi_1. \ Ga_{\pi_0} \land Gb_{\pi_0} \land Gc_{\pi_0} \land Ga_{\pi_1} \land G\neg c_{\pi_1} \)

Replace indexed atomic propositions with fresh atomic propositions.

\( Ga_0 \land Gb_0 \land Gc_0 \land Ga_1 \land G\neg c_1 \)
Existential Fragment

Theorem
\( \exists^* \text{HyperLTL-SAT is PSPACE-complete.} \)

Example
\( \exists \pi_0 \exists \pi_1. \ Ga_{\pi_0} \land Gb_{\pi_0} \land Gc_{\pi_0} \land Ga_{\pi_1} \land G\neg c_{\pi_1} \)

Replace indexed atomic propositions with fresh atomic propositions.

\[
Ga_0 \land Gb_0 \land Gc_0 \land Ga_1 \land G\neg c_1
\]
\[
t : \{a_0, b_0, c_0, a_1\}^\omega
\]
Existential Fragment

Theorem
\[ \exists^* \text{ HyperLTL-SAT is PSPACE-complete}. \]

Example
\[ \exists \pi_0 \exists \pi_1. Ga_{\pi_0} \land Gb_{\pi_0} \land Gc_{\pi_0} \land Ga_{\pi_1} \land G\neg c_{\pi_1} \]
Replace indexed atomic propositions with fresh atomic propositions.

\[ Ga_0 \land Gb_0 \land Gc_0 \land Ga_1 \land G\neg c_1 \]
\[ t : \{a_0, b_0, c_0, a_1\}^\omega \]
\[ T = \{\{a, b, c\}^\omega, \{a\}^\omega\} \]
Universal Fragment

Theorem
\( \forall^* \text{ HyperLTL-SAT is PSPACE-complete.} \)

Example
\( \forall \pi \forall \pi'. G b_{\pi} \land G \neg b_{\pi'} \equiv G b \land G \neg b \)
\[ \downarrow \downarrow \]
\( t \quad t \)
\( \downarrow \)
unsatisfiable

Discard indexes from indexed propositions
Solving HyperLTL-SAT

1. Alternation-free fragments ($\forall^* \& \exists^*$)
2. Alternation starting with existential quantifier ($\exists^* \forall^*$)
3. Alternation starting with universal quantifier ($\forall^* \exists^*$)
Lemma

For every $\exists \pi_1 \ldots \exists \pi_n \forall \pi'_1 \ldots \forall \pi'_m \cdot \varphi$ HyperLTL formula, there exists an equisatisfiable $\exists^* \forall^* \text{HyperLTL}$ formula.

Example

$\exists \pi_0 \exists \pi_1 \forall \pi'_0 \forall \pi'_1 \cdot (Ga_{\pi'_0} \landGb_{\pi'_1}) \land (Gc_{\pi_0} \land Gd_{\pi_1})$

Unroll universal quantifiers

$\exists \pi_0 \exists \pi_1 \cdot (Ga_{\pi_0} \landGb_{\pi_0}) \land (Gc_{\pi_0} \land Gd_{\pi_1})$

$\land (Ga_{\pi_1} \landGb_{\pi_0}) \land (Gc_{\pi_0} \land Gd_{\pi_1})$

$\land (Ga_{\pi_0} \landGb_{\pi_1}) \land (Gc_{\pi_0} \land Gd_{\pi_1})$

$\land (Ga_{\pi_1} \landGb_{\pi_1}) \land (Gc_{\pi_0} \land Gd_{\pi_1})$
Complexity of $\exists^* \forall^*$ HyperLTL-SAT

Theorem

Let $n$ be the number of existential quantifiers and $m$ be the number of universal quantifiers. $\exists^* \forall^*$ HyperLTL-SAT is EXPSPACE-complete in $m$.

- Unrolling results in formula of size $O(n^m)$.
- Hardness follows from an encoding of an EXPSPACE-bounded Turing machine in this fragment.
Application: Implication Checking of Quantifier-alternation-free Hyperproperties

$\psi$ implies $\varphi$?
Check the negation $\psi \land \neg \varphi$ for unsatisfiability.

- If one formula is in the $\forall^*$ fragment and the other in the $\exists^*$ fragment, the resulting formula is alternation-free.
- If both $\psi$ and $\varphi$ are in the same fragment, then the resulting formula is in the $\exists^*\forall^*$ fragment.

Theorem

Implication Checking of quantifier-alternation-free HyperLTL formulas is EXPSPACE-complete.
Theorem

Bounded $\exists^* \forall^b$ HyperLTL-SAT is PSPACE-complete.

Observation: In practice, many properties of interest quantify universally over pairs of traces

$\forall \pi. \forall \pi'. \text{G} (l_\pi = l_{\pi'}) \rightarrow \text{G} (O_\pi = O_{\pi'})$
Solving HyperLTL-SAT

1. Alternation-free fragments ($\forall^* \& \exists^*$)
2. Alternation starting with existential quantifier ($\exists^*\forall^*$)
3. Alternation starting with universal quantifier ($\forall^*\exists^*$)
The Power of $\forall \exists$

$$\forall \pi \exists \pi'. \text{ } a_{\pi'}$$  \hspace{1cm} (1)

$$\land G (a_{\pi} \rightarrow X G \neg a_{\pi})$$  \hspace{1cm} (2)

$$\land G (a_{\pi} \rightarrow X a_{\pi'})$$  \hspace{1cm} (3)

$t_1 : \{a\}(\{\})^\omega$

$t_2 : \{\}{a}\{\}\{\}^\omega$

$t_3 : \{\}{\}{a}\{\}\{\}^\omega$

$$\ldots$$

$\rightarrow$ Model has infinitely many traces.
Undecidability of $\forall \exists$ HyperLTL-SAT

Theorem
The satisfiability problem for any fragment of HyperLTL that contains the $\forall \exists$ formulas is undecidable.

- Undecidability follows from a reduction from Post's Correspondence Problem.
## Summary HyperLTL-SAT

<table>
<thead>
<tr>
<th>( \exists^* )</th>
<th>( \forall^* )</th>
<th>( \exists^* \forall^* )</th>
<th>Bounded ( \exists^* \forall^* )</th>
<th>( \forall \exists )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PSPACE-complete</strong></td>
<td><strong>PSPACE-complete</strong></td>
<td><strong>EXPSPACE-complete</strong></td>
<td><strong>PSPACE-complete</strong></td>
<td><strong>undecidable</strong></td>
</tr>
</tbody>
</table>
Part V

Beyond HyperLTL
Hyperproperties and branching-time logics

**Observation:**
Hyperproperties induce trace equivalence.

\[ \forall K, K'. \text{ Traces}(K) = \text{Traces}(K') \implies K \models H \iff K' \models H \]
Hyperproperties and branching-time logics

Observation:
Hyperproperties induce trace equivalence.

$$\forall K, K'. \text{ Traces}(K) = \text{Traces}(K') \implies K \models H \iff K' \models H$$

Hyperproperties are not models for branching-time logics.
HyperCTL*

Syntax: \( \varphi ::= a_\pi \mid \forall \pi. \varphi \mid \exists \pi. \varphi \mid X \varphi \mid G \varphi \mid \varphi U \varphi \mid \ldots \)

- HyperLTL: no quantifiers under temporal operators
- HyperCTL*: no restriction
- HyperCTL* with 1 path variable \( \approx \) CTL*
What do we get beyond HyperLTL and CTL*?

```cpp
bool y;
bool x = read(); // secret
output(y);
```
What do we get beyond HyperLTL and CTL*?

```c
bool y;
bool x = read(); // secret
output(y);
```

![Diagram](attachment:hyperltl_diagram.png)
What do we get beyond HyperLTL and CTL*?

```cpp
bool y;
bool x = read(); // secret
output(y);
```

\[ \forall \pi. G(\text{read}_\pi \rightarrow \forall \pi'. G(o_\pi \leftrightarrow o'_{\pi'})) \]
The Linear-Hyper-Branching Spectrum

The induced process equivalence of HyperLTL is trace equivalence. Two systems with the same set of traces satisfy the same HyperLTL formulas.

The induced process equivalence of HyperCTL* is bisimulation. Two bisimular systems satisfy the same HyperCTL* formulas.
HyperCTL* Model Checking

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of states.
- Projection handles quantifiers.
- Complementation handles quantifier alternations.
HyperCTL* Model Checking

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of states.
- Projection handles quantifiers.
- Complementation handles quantifier alternations.

Example: \( \forall \pi. G(a_\pi \Rightarrow \forall \pi'. G(o_\pi \leftrightarrow o_{\pi'}) \)  

Negated: \( \exists \pi. F(a_\pi \land \exists \pi'. F(o_\pi \leftrightarrow o_{\pi'}) \)  

\[ 57 / 60 \]
HyperCTL* Model Checking

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of states.
- Projection handles quantifiers.
- Complementation handles quantifier alternations.

Example: $\forall \pi. G(a_\pi \Rightarrow \forall \pi'. G(o_\pi \leftrightarrow o_{\pi'}))$

Negated: $\exists \pi. F(a_\pi \land \exists \pi'. \underbrace{F(o_\pi \nleftrightarrow o_{\pi'})}_{\mathcal{A} \text{ with alphabet } S \times S}$
HyperCTL* Model Checking

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of states.
- Projection handles quantifiers.
- Complementation handles quantifier alternations.

Example: \( \forall \pi. G(a_\pi \Rightarrow \forall \pi'. G(o_\pi \leftrightarrow o_{\pi'}) ) \)

Negated: \( \exists \pi. F(a_\pi \land \exists \pi'. \underbrace{F(o_\pi \not\leftrightarrow o_{\pi'})}_{\mathcal{A} \text{ with alphabet } S \times S}) \)

\( \mathcal{A} \) with alphabet \( S \times S \)

\( \mathcal{A}' \) with alphabet \( S \)
HyperCTL* Model Checking

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of states.
- Projection handles quantifiers.
- Complementation handles quantifier alternations.

Example: $\forall \pi. G(a_\pi \Rightarrow \forall \pi'. G(o_\pi \leftrightarrow o_{\pi'}))$

Negated: $\exists \pi. F(a_\pi \land \exists \pi'. F(o_\pi \not\leftrightarrow o_{\pi'}))$

$\mathcal{A}$ with alphabet $S \times S$

$\mathcal{A}'$ with alphabet $S$

$\mathcal{A}''$ with alphabet $S$
HyperCTL* Model Checking

- Reduction to emptiness of Büchi word automata
- Alphabet consists of tuples of \textit{states}.
- \textbf{Projection} handles quantifiers.
- \textbf{Complementation} handles quantifier alternations.

Example: $\forall \pi. G(a_\pi \Rightarrow \forall \pi'. G(o_\pi \leftrightarrow o_{\pi'}) )$

Negated: $\exists \pi. F(a_\pi \land \exists \pi'. \underbrace{F(o_\pi \not\leftrightarrow o_{\pi'})}_{\mathcal{A} \text{ with alphabet } S \times S})$

$\mathcal{A}'$ with alphabet $S$

$\mathcal{A}''$ with alphabet $S$

$\mathcal{A}'''$ with 1-letter alphabet
Part VI

Conclusions
Conclusions

- HyperLTL is a powerful tool for information security and beyond
  - Information-flow control
  - Symmetries in distributed systems
  - Error resistant codes
- Efficient model checking for alternation-free HyperLTL (non-elementary in general)
- Efficient satisfiability/implication/equivalence checking for alternation-free HyperLTL (undecidable in general)
Conclusions

- HyperLTL is a powerful tool for information security and beyond
  - Information-flow control
  - Symmetries in distributed systems
  - Error resistant codes

- Efficient model checking for alternation-free HyperLTL (non-elementary in general)

- Efficient satisfiability/implication/equivalence checking for alternation-free HyperLTL (undecidable in general)

Open problems

- HyperLTL on software
- Quantitative hyperproperties
- Specialized model checking algorithms
References

- Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. “Algorithms for Model Checking HyperLTL and HyperCTL*.” CAV 2015

Not covered in this tutorial: