The Linear-Hyper-Branching Spectrum of Temporal Logics

Bernd Finkbeiner, Markus N. Rabe

Saarland University

Paris, 5 September 2014
A Simple Security Policy

Process 1: \( x := \text{input}(); \) // boolean secret
Process 2: \( \text{output}(0); \)
Process 3: \( \text{output}(1); \)

Information-flow Security:
“All pairs of executions have the same output.”
A Simple Security Policy

Process 1:  \( x := \text{input}(); \)  // boolean secret
Process 2:  \( \text{output}(0); \)
Process 3:  \( \text{output}(1); \)

Information-flow Security:
“All pairs of executions have the same output.”

Information flow properties compare **multiple executions**!
A Simple Security Policy

Process 1: \( x := \text{input}(); \) // boolean secret
Process 2: \( \text{output}(0); \)
Process 3: \( \text{output}(1); \)

Information-flow Security:
“All pairs of executions have the same output.”

Information flow properties compare multiple executions!

Clarkson&Schneider: Hyperproperties!
Temporal Logics

LTL, CTL, CTL*
Temporal Logics

LTL, CTL, CTL∗ 😞
Temporal Logics

LTL, CTL, CTL* 😞

HyperLTL, HyperCTL* can express many hyperproperties 😊
Temporal Logics

LTL, CTL, CTL* ☹

HyperLTL, HyperCTL* can express many hyperproperties 😊

“All pairs of executions have the same output.”

\[ \forall \pi. \forall \pi' : \square (\text{out}_\pi \leftrightarrow \text{out}_{\pi'}) \]
Temporal Logics

LTL, CTL, CTL* 😞

HyperLTL, HyperCTL* can express many hyperproperties 😊 and their model checking problems are decidable.

“All pairs of executions have the same output.”

\[ \forall \pi. \forall \pi' : \square (out_\pi \leftrightarrow out_{\pi'}) \]
Quantifiers with path variables:

Syntax: \( \varphi ::= a_\pi \mid \forall \pi.\varphi \mid \exists \pi.\varphi \mid \bigcirc \varphi \mid \varphi U \varphi \mid \ldots \)
Quantifiers with path variables:

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

HyperLTL: Formulas in prenex form.
Quantifiers with path variables:

Syntax: \[ \varphi ::= \alpha \pi \mid \forall \pi \varphi \mid \exists \pi \varphi \mid \bigcirc \varphi \mid \varphi U \varphi \mid \ldots \]

HyperLTL: Formulas in prenex form.

“All executions have the same output.” \( \forall \pi. \forall \pi'. \bigcirc (\text{out}_\pi \leftrightarrow \text{out}_{\pi'}) \)
Quantifiers with path variables:

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

HyperLTL: Formulas in prenex form.

“All executions have the same output.” \( \forall \pi. \forall \pi'. \Box (\text{out}_\pi \leftrightarrow \text{out}_{\pi'}) \)

1. \( \{\pi \mapsto p\} \models_K \forall \pi'. \Box (\text{out}_\pi \leftrightarrow \text{out}_{\pi'}) \)

2. \( \{\pi \mapsto p, \pi' \mapsto p'\} \models_K \Box (\text{out}_\pi \leftrightarrow \text{out}_{\pi'}) \)
HyperLTL/HyperCTL*

Quantifiers with path variables:

Syntax: \( \varphi ::= a_{\pi} \mid \forall \pi.\varphi \mid \exists \pi.\varphi \mid \bigcirc \varphi \mid \varphi U \varphi \mid \ldots \)

HyperLTL: Formulas in prenex form.

“All executions have the same output.” \( \forall \pi. \forall \pi'. \square(\text{out}_{\pi} \leftrightarrow \text{out}_{\pi'}) \)

1. \( \{\pi \mapsto p\} \models_K \forall \pi'. \square(\text{out}_{\pi} \leftrightarrow \text{out}_{\pi'}) \)
2. \( \{\pi \mapsto p, \pi' \mapsto p'\} \models_K \square(\text{out}_{\pi} \leftrightarrow \text{out}_{\pi'}) \)
3. \( \{\pi \mapsto p[i, \infty], \pi' \mapsto p'[i, \infty]\} \models_K \text{out}_{\pi} \leftrightarrow \text{out}_{\pi'} \)
A new kind of expressiveness?
A new kind of expressiveness?

The Spectrum of Temporal Logics:

Linear-time logics: LTL
  ▶ Trace equivalence

Branching-time logics: CTL, CTL*
  ▶ Bisimulation
A new kind of expressiveness?

The Spectrum of Temporal Logics:

Linear-time logics: LTL
- Trace equivalence

Branching-time logics: CTL, CTL*, HyperCTL*
- Bisimulation
A new kind of expressiveness?

The Spectrum of Temporal Logics:

Linear-time logics: LTL, HyperLTL
  ▶ Trace equivalence

Branching-time logics: CTL, CTL*, HyperCTL*
  ▶ Bisimulation
A new kind of expressiveness?

The Spectrum of Temporal Logics:

Linear-time logics: LTL, HyperLTL
  ▶ Trace equivalence

Branching-time logics: CTL, CTL*, HyperCTL*
  ▶ Bisimulation

\[
\text{LTL} \subset \text{HyperLTL} \subset \text{CTL}^* \subset \text{HyperCTL}^*
\]
Example

Process 1:
\[ x := \text{input}(); \]

Process 2:
\[ \text{output}(0); \]

Process 3:
\[ \text{output}(1); \]
Example

Process 1: \[ x := \text{input}(); \]
Process 2: \[ \text{output}(0); \]
Process 3: \[ \text{output}(1); \]

\[ \forall \pi. \forall \pi' : \Box (\text{out}_\pi \leftrightarrow \text{out}_{\pi'}) \]
Example

Process 1:  Process 2:  Process 3:
\[ x := \text{input}(); \quad \text{output}(0); \quad \text{output}(1); \]

Restrict scheduler to commit initially on a sequence.

\[ \forall \pi. \forall \pi': \square (\text{out}_\pi \leftrightarrow \text{out}_{\pi'}) \]
Example

Process 1: \( x := \text{input}(); \) \hspace{1cm} Process 2: \( \text{output}(0); \) \hspace{1cm} Process 3: \( \text{output}(1); \)

Restrict scheduler to commit initially on a sequence.

\[ \forall \pi. \forall \pi': \square(\text{out}_\pi \leftrightarrow \text{out}_{\pi'}) \]
Example

Process 1: \( x := \text{input}(); \)  
Process 2: \( \text{output}(0); \)  
Process 3: \( \text{output}(1); \)

Restrict scheduler to commit initially on a sequence.

\[ \forall \pi. \square (p_1 \implies \forall \pi'. \forall \pi''. \square (\text{out}_{\pi'} = \text{out}_{\pi''})) \]
Example

Process 1: \(x := \text{input}();\)  
Process 2: \(\text{output}(0);\)  
Process 3: \(\text{output}(1);\)

Restrict scheduler to commit initially on a sequence.

\[
\forall \pi. \square(p_1 \implies \forall \pi'. \forall \pi''. \square(\text{out}_{\pi'} = \text{out}_{\pi''}))
\]
Conclusions

Logics for Hyperproperties: How information leaves the system.

Branching-time logics: How information enters the system.
Conclusions

Logics for Hyperproperties: How information **leaves** the system.

Branching-time logics: How information **enters** the system.

*Robin Milner attributed this insight to Carl-Adam Petri: “information enters a non-deterministic process in finite quantities throughout time” and that the branching-time view allows us to observe “in what states, and what ways” this happens.*

Robin Milner, *What is a process?*, September 2009.
Declassification: “When”

Weak until \( \varphi \mathcal{W} \psi \)

“\( \varphi \) holds forever, or from now until \( \psi \) occurs.”

Reactive variation of observational determinism:

\[ \forall \pi. \forall \pi'. \text{lowOut}_{\pi} = \text{lowOut}_{\pi'}, \ \mathcal{W} \ \text{lowIn}_{\pi} = \text{lowIn}_{\pi'} \]

Let’s add declassification:

\[ \forall \pi. \forall \pi'. \text{lowOut}_{\pi} = \text{lowOut}_{\pi'}, \ \mathcal{W} \ (\text{lowIn}_{\pi} = \text{lowIn}_{\pi'} \ \lor \ \psi) \]

where \( \psi \) is a temporal condition like “User presses send button”.

Examples

“Paths \( \pi \) and \( \pi' \) agree on the low output in the current step”:

\[
\text{lowOut}_\pi = \text{lowOut}_{\pi'}
\]

- **Noninference**
  
  \[
  \forall \pi. \exists \pi'. (G\lambda_{\pi'}) \land G(\text{lowIn}_\pi = \text{lowIn}_{\pi'} \land \text{lowOut}_\pi = \text{lowOut}_{\pi'})
  \]
  
  McLean’94

- **Generalized Noninterference**

  \[
  \forall \pi. \forall \pi'. \exists \pi''. (G\text{highIn}_\pi = \text{highIn}_{\pi''}) \land
  \hspace{1cm} \land G(\text{lowIn}_{\pi'} = \text{lowIn}_{\pi''} \land \text{lowOut}_{\pi'} = \text{lowOut}_{\pi''})
  \]

  McCullough’88

- **Observational Determinism**

  \[
  \forall \pi. \forall \pi'. \text{lowIn}_\pi = \text{lowIn}_{\pi'} \Rightarrow G(\text{lowOut}_\pi = \text{lowOut}_{\pi'})
  \]

  Zdancewich&Myers’03
Declassification: “What”

Reactive variation of observational determinism:

\[ \forall \pi. \forall \pi'. lowOut_{\pi} = lowOut_{\pi'}, \forall lowIn_{\pi} = lowIn_{\pi'} \]

Let’s add declassification:

\[ \forall \pi. \forall \pi'. (lowOut_{\pi} = lowOut_{\pi'}, \forall lowIn_{\pi} = lowIn_{\pi'} \lor \psi_\pi \land \neg \psi_{\pi'} \]

where \( \psi \) is a condition on the secret describing a fact that may be learned like “the high user logs in eventually.”
Composing larger policies

Let $\varphi(A, B)$ describe your favorite notion of information flow for users/agents/variables $A$, $B$, and $C$.

The expression: $\varphi(A, B) \land \varphi(B, C) \land \varphi(A, C)$

This creates a “security lattice”.
Related specification frameworks

Security specific frameworks:

- Selective interleaving functions
  - McLean’94
- MAKS
  - Mantel’00

Epistemic temporal knowledge

- Model checking knowledge with perfect recall
  - vdMeyden, Shilov’99
- Expressing security properties in temporal epistemic logics
  - vdMeyden, Wilke’07
  - Halpern, O’Neill’08
  - Balliu, Dam, Guernic’11

Fixed point logics

- Polyadic modal $\mu$-calculus
  - Anderssen’94
- Holistic (incremental) hyperproperties
  - Milushev, Clarke’12
Model checking techniques for security policies

- Self-composition to verify observational determinism
  Barthe, D’Argenio, Rezk’04

- Expressing observational determinism in temporal logics on the self-composition of a system
  Huisman, Worah, Sunesen’06

- Checking a form of declassification on the self-composition of a system
  Terauchi, Aiken; Huisman, Blondeel’11

Our algorithm:
For expressions with quantifier alternation depth 0, the projection operation replaces self-composition. E.g.:

$$\forall \pi \forall \pi'. G(out_{\pi} \leftrightarrow out_{\pi'})$$

- 2 copies of AP
- 1 copy of AP
System model

Kripke structure $K$

- States $s \in S$, initial state $s_{\text{init}}$
- Transition relation $\delta : S \rightarrow 2^S$
- Labeling function $L : S \rightarrow 2^{AP}$
System model

Kripke structure $K$

- **States** $s \in S$, initial state $s_{\text{init}}$
- **Transition relation** $\delta : S \rightarrow 2^S$
- **Labeling function** $L : S \rightarrow 2^{AP}$

Paths $p \in \text{Paths}$ are sequences of states and sets of labels

$$
\begin{align*}
    p &= (s_0, A_0)(s_1, A_1)(s_2, A_2) \ldots \\
    p[i, \infty] &= (s_i, A_i)(s_{i+1}, A_{i+1})(s_{i+2}, A_{i+2}) \ldots \\
    p(i) &= s_i \\
    A_i(p) &= A_i
\end{align*}
$$
System model

Kripke structure $K$

- States $s \in S$, initial state $s_{\text{init}}$
- Transition relation $\delta : S \to 2^S$
- Labeling function $L : S \to 2^{\text{AP}}$

Paths $p \in \text{Paths}$ are sequences of states and sets of labels

$$p = (s_0, A_0)(s_1, A_1)(s_2, A_2) \ldots$$

$$p[i, \infty] = (s_i, A_i)(s_{i+1}, A_{i+1})(s_{i+2}, A_{i+2}) \ldots$$

$$p(i) = s_i$$

$$A_i(p) = A_i$$

Path assignments $\Pi$ map path variables $\pi$ to paths $p$.

$$\Pi[i, \infty](\pi) = \Pi(\pi)[i, \infty]$$ denotes the $i$-th successor.
Semantics (HyperLTL)

HyperLTL formulas are in prenex form.

\[ \Pi \models_K a_{\pi} \quad \text{iff} \quad a \in A_0(\Pi(\pi)) \]

\[ \Pi \models_K G \varphi \quad \text{iff} \quad \forall i \geq 0 : \Pi[i, \infty] \models_K \varphi \]

\[ \Pi \models_K \forall \pi. \varphi \quad \text{iff} \quad \forall p \in \text{Paths}_K(\text{init}) : \Pi[\pi \mapsto p] \models_K \varphi \]

\[ \Pi \models_K \neg \varphi \quad \text{iff} \quad \Pi \not\models_K \varphi \]

\[ \Pi \models_K \varphi_1 \lor \varphi_2 \quad \text{iff} \quad \Pi \models_K \varphi_1 \text{ or } \Pi \models \varphi_2 \]

\[ \Pi \models_K \bigcirc \varphi \quad \text{iff} \quad \Pi[1, \infty] \models_K \varphi \]

\[ \Pi \models_K \varphi_1 \mathcal{U} \varphi_2 \quad \text{iff} \quad \text{there exists } i \geq 0 : \Pi[i, \infty] \models_K \varphi_2 \]

and for all \( 0 \leq j < i \) we have \( \Pi[j, \infty] \models_K \varphi_1 \)
Semantics (HyperCTL*)

In HyperCTL* there is no restriction on the use of quantifiers!

- \( \Pi \models_K a_\pi \) iff \( a \in A_0(\Pi(\pi)) \)
- \( \Pi \models_K G \varphi \) iff \( \forall i \geq 0 : \Pi[i, \infty] \models_K \varphi \)
- \( \Pi \models_K \forall \pi. \varphi \) iff \( \forall p \in \text{Paths}_K(\Pi(\pi^*)(0)) : \Pi[\pi \mapsto p, \pi^* \mapsto p] \models_K \varphi \)
- \( \Pi \models_K \neg \varphi \) iff \( \Pi \not\models_K \varphi \)
- \( \Pi \models_K \varphi_1 \lor \varphi_2 \) iff \( \Pi \models_K \varphi_1 \) or \( \Pi \models \varphi_2 \)
- \( \Pi \models_K \bigcirc \varphi \) iff \( \Pi[1, \infty] \models_K \varphi \)
- \( \Pi \models_K \varphi_1 U \varphi_2 \) iff there exists \( i \geq 0 : \Pi[i, \infty] \models_K \varphi_2 \) and for all \( 0 \leq j < i \) we have \( \Pi[j, \infty] \models_K \varphi_1 \)
Model Checking (2)

Decidable!
Model Checking (2)

Decidable!

Decidability by a reduction to QPTL satisfiability

- Algorithm based on Büchi automata;
  uses projection and complementation

Complexity in $|\varphi|$: $2^{2^{\cdots 2^{\varphi}}}$ \{ one level per quantifier alternation \}
Model Checking

Complexity depends on the quantifier alternation depth.

0. $\forall \pi. \forall \pi'. \psi$  PSPACE in $|\psi|$,  NLOGSPACE in $|K|$  
   - Observational determinism
   - Goguen&Meseguer noninterference
Model Checking

Complexity depends on the quantifier alternation depth.

0. $\forall \pi. \forall \pi'. \psi$  \quad \text{PSPACE in } |\psi|, \quad \text{NLOGSPACE in } |K| 
   - Observational determinism
   - Goguen&Meseguer noninterference

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

2. ...
Model Checking

Complexity depends on the quantifier alternation depth.

0. $\forall \pi. \forall \pi'. \psi$ \hspace{1em} PSPACE in $|\psi|$, \hspace{1em} NLOGSPACE in $|K|$
   - Observational determinism
   - Goguen&Meseguer noninterference

1. $\forall \pi. \exists \pi'. \psi$ \hspace{1em} EXPSPACE in $|\psi|$, \hspace{1em} PSPACE in $|K|$
   - Noninference
   - Generalized noninterference

2. ... 

Security properties often need at most one quantifier alternation!
Trace properties

Trace property $P$: **Set of traces**
- $M \models P$ iff $\text{Tr}(M) \subseteq P$

Hyperproperty $H$: **Set of sets of traces**
- $M \models H$ iff $\text{Tr}(M) \in H$
- Contain many information-flow security policies