Deciding Hyperproperties

Lukas Nieuweboer

January 22, 2025

Radboud University

Papers

- Deciding Hyperproperties - Bernd Finkbeiner and Christopher Hahn
- A Temporal Logic Approach to Information-flow Control - Markus N. Rabe

Papers

- Deciding Hyperproperties - Bernd Finkbeiner and Christopher Hahn
- A Temporal Logic Approach to Information-flow Control - Markus N. Rabe

Overview

1. Linear Temperal Logic
2. HyperLTL and Hyperproperties
3. Complexity of HyperLTL Satisfiability
   - Alternation-free HyperLTL
   - $\exists^*\forall^*$ Fragment
   - $\forall\exists$ Fragment

Overview

1. Linear Temperal Logic
2. HyperLTL and Hyperproperties
3. Complexity of HyperLTL Satisfiability
   - Alternation-free HyperLTL
   - $\exists^*\forall^*$ Fragment
   - $\forall\exists$ Fragment

Overview

1. Linear Temperal Logic
2. HyperLTL and Hyperproperties
3. Complexity of HyperLTL Satisfiability
   - Alternation-free HyperLTL
   - $\exists^*\forall^*$ Fragment
   - $\forall\exists$ Fragment

Overview

1. Linear Temperal Logic
2. HyperLTL and Hyperproperties
3. Complexity of HyperLTL Satisfiability
   - Alternation-free HyperLTL
   - $\exists^*\forall^*$ Fragment
   - $\forall\exists$ Fragment

Overview

1. Linear Temperal Logic
2. HyperLTL and Hyperproperties
3. Complexity of HyperLTL Satisfiability
   - Alternation-free HyperLTL
   - $\exists^*\forall^*$ Fragment
   - $\forall\exists$ Fragment

Overview

1. Linear Temperal Logic
2. HyperLTL and Hyperproperties
3. Complexity of HyperLTL Satisfiability
    - Alternation-free HyperLTL
    - $\exists^*\forall^*$ Fragment
    - $\forall\exists$ Fragment

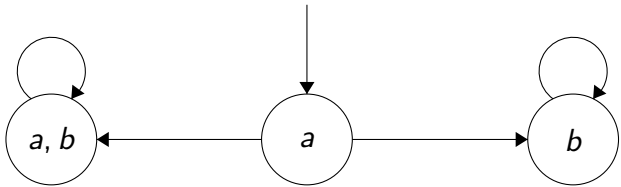
Remember LTL?

- $TR := (2^{AP})^{\omega}$

- For example $t = (\{a\}, \{a, b\}, \{a, b\} \ldots)$ and $t' = (\{a\}, \{b\}, \{b\} \ldots)$.
- Useful for analysing computation or execution traces of programs.

Remember LTL?

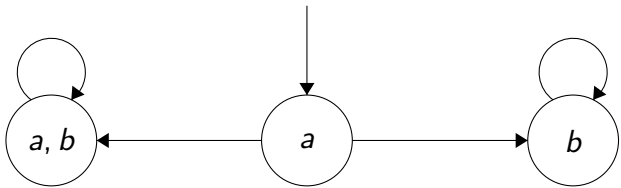
- $TR := (2^{AP})^{\omega}$



- For example $t = (\{a\}, \{a, b\}, \{a, b\} \ldots)$ and $t' = (\{a\}, \{b\}, \{b\} \ldots)$.
- Useful for analysing computation or execution traces of programs.

Remember LTL?

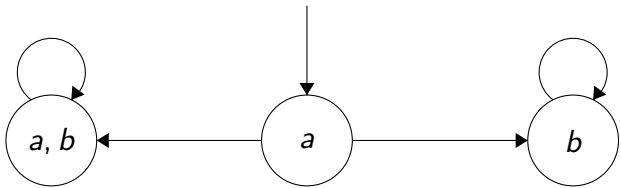
- $TR := (2^{AP})^{\omega}$



- For example $t = (\{a\}, \{a, b\}, \{a, b\} \ldots)$ and $t' = (\{a\}, \{b\}, \{b\} \ldots)$.
- Useful for analysing computation or execution traces of programs.

Remember LTL?

- $TR := (2^{AP})^{\omega}$



- For example $t = (\{a\}, \{a, b\}, \{a, b\} \ldots)$ and $t' = (\{a\}, \{b\}, \{b\} \ldots)$.
- Useful for analysing computation or execution traces of programs.

LTL Syntax

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi$$

- $\Diamond\varphi \equiv true \,\mathcal{U}\, \varphi$
- $\Box\varphi \equiv \neg \Diamond \neg\varphi$

LTL Syntax

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi$$

- $\Diamond\varphi \equiv true \,\mathcal{U}\, \varphi$
- $\Box\varphi \equiv \neg \Diamond \neg\varphi$

LTL Syntax

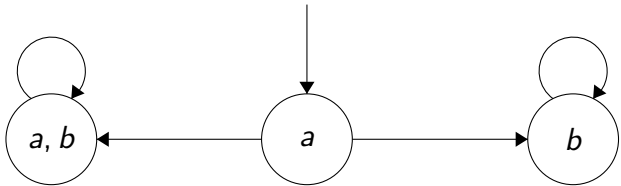
$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi$$

- $\Diamond\varphi \equiv true \,\mathcal{U}\, \varphi$
- $\Box\varphi \equiv \neg \Diamond \neg\varphi$

LTL Semantics

$$
\begin{aligned}
t \vDash\ & p && \iff p \in t[0] \\
t \vDash\ & \neg\varphi && \iff t \nvDash \varphi] \\
t \vDash\ & \varphi_1 \vee \varphi_2 && \iff t \vDash \varphi_1 \text{ or } t \vDash \varphi_2 \\
t \vDash\ & \bigcirc\varphi && \iff t[1, \infty] \vDash \varphi \\
t \vDash\ & \varphi_1 \,\mathcal{U}\, \varphi_2 && \iff \text{there exists } i \geq 0 : t[i, \infty] \vDash \varphi_2 \\
& && \quad \text{and for all } 0 \leq j < i \text{ we have } t[j, \infty] \vDash \varphi_1
\end{aligned}
$$

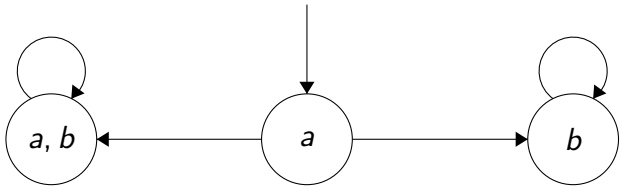
LTL Example



$t = (\{a\}, \{a, b\}, \{a, b\} \ldots)$ $t' = (\{a\}, \{b\}, \{b\} \ldots)$

- $\Diamond\Box a$ holds for t ($t \models \Diamond\Box a$), but not for t'.
- A *trace property* $T \subseteq TR$ is the set of traces for which a certain LTL formula holds.

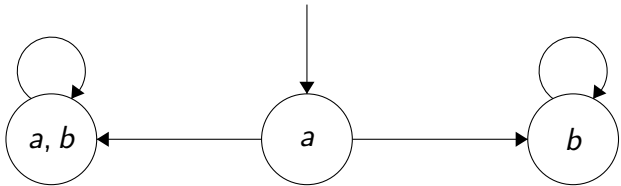
LTL Example



$t = (\{a\}, \{a, b\}, \{a, b\} \ldots)$ $t' = (\{a\}, \{b\}, \{b\} \ldots)$

- $\Diamond\Box a$ holds for t ($t \vDash \Diamond\Box a$), but not for t'.
- A *trace property* $T \subseteq TR$ is the set of traces for which a certain LTL formula holds.

LTL Example



$$t = (\{a\}, \{a, b\}, \{a, b\} \ldots) \quad t' = (\{a\}, \{b\}, \{b\} \ldots)$$

- $\Diamond\Box a$ holds for t ($t \vDash \Diamond\Box a$), but not for t'.
- A *trace property* $T \subseteq TR$ is the set of traces for which a certain LTL formula holds.

Complexity Intermezzo

$$PSPACE \subseteq EXPSPACE$$

Decision problems that can be solved using a Turing machine using a polynomial/exponential amount of space.

LTL Satisfiability

LTL-SAT is the problem of deciding whether for an arbitrary LTL formula φ there exists a trace $t \in TR$ such that $t \vDash \varphi$.

Theorem

LTL-SAT is PSPACE-complete.

LTL Satisfiability

LTL-SAT is the problem of deciding whether for an arbitrary LTL formula φ there exists a trace $t \in TR$ such that $t \vDash \varphi$.

Theorem

LTL-SAT is PSPACE-complete.

Overview

✓ Linear Temperal Logic

2. HyperLTL and Hyperproperties
3. Complexity of HyperLTL Satisfiability
   - Alternation-free HyperLTL
   - $\exists^*\forall^*$ Fragment
   - $\forall\exists$ Fragment

Limitations of LTL

- Using LTL we can characterize program behaviour for individual traces.
- Correctness proofs of properties like mutual exclusion or liveness.

$$\Box(c_1 \implies \neg c_2) \qquad \Box(t_1 \implies \Diamond c_1)$$

- **Limitations:**
  - Some security policies cannot be characterized as properties of individual traces.
  - Therefore we introduce **hyperproperties**, properties of sets of traces.

Limitations of LTL

- Using LTL we can characterize program behaviour for individual traces.
- Correctness proofs of properties like mutual exclusion or liveness.

$$\square(c_1 \implies \neg c_2) \qquad \square(t_1 \implies \Diamond c_1)$$

- **Limitations:**
  - Some security policies cannot be characterized as properties of individual traces.
  - Therefore we introduce **hyperproperties**, properties of sets of traces.

Limitations of LTL

- Using LTL we can characterize program behaviour for individual traces.
- Correctness proofs of properties like mutual exclusion or liveness.

$$\Box(c_1 \implies \neg c_2) \qquad \Box(t_1 \implies \Diamond c_1)$$

- **Limitations:**
  - Some security policies cannot be characterized as properties of individual traces.
  - Therefore we introduce **hyperproperties**, properties of sets of traces.

Limitations of LTL

- Using LTL we can characterize program behaviour for individual traces.
- Correctness proofs of properties like mutual exclusion or liveness.

$$\Box(c_1 \implies \neg c_2) \qquad \Box(t_1 \implies \Diamond c_1)$$

- **Limitations:**
  - Some security policies cannot be characterized as properties of individual traces.
  - Therefore we introduce **hyperproperties**, properties of sets of traces.

Limitations of LTL

- Using LTL we can characterize program behaviour for individual traces.
- Correctness proofs of properties like mutual exclusion or liveness.

$$\square(c_1 \implies \neg c_2) \qquad \square(t_1 \implies \Diamond c_1)$$

- **Limitations:**
  - Some security policies cannot be characterized as properties of individual traces.
  - Therefore we introduce **hyperproperties**, properties of sets of traces.

HyperLTL

- HyperLTL extends LTL with trace variables $\mathcal{V}$ and trace quantifiers $\forall$, $\exists$.

$$\psi ::= \exists \pi . \psi \mid \forall \pi . \psi \mid \varphi$$
$$\varphi ::= true \mid a_\pi \mid \neg \varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \, \mathcal{U} \, \varphi$$

- Derived operators $\Diamond$ and $\Box$ are the same.

HyperLTL

- HyperLTL extends LTL with trace variables $\mathcal{V}$ and trace quantifiers $\forall, \exists$.

$$\psi ::= \exists \pi. \psi \mid \forall \pi. \psi \mid \varphi$$
$$\varphi ::= true \mid a_\pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi$$

- Derived operators $\Diamond$ and $\Box$ are the same.

HyperLTL

- HyperLTL extends LTL with trace variables $\mathcal{V}$ and trace quantifiers $\forall, \exists$.

$$\psi ::= \exists \pi . \psi \mid \forall \pi . \psi \mid \varphi$$
$$\varphi ::= true \mid a_\pi \mid \neg \varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \, \mathcal{U} \, \varphi$$

- Derived operators $\Diamond$ and $\Box$ are the same.

Hyperproperty

- A HyperLTL formula defines a *hyperproperty* (a set of sets of traces).
- A set T of traces satisfies the hyperproperty if it is an element of this set of sets.
- Formally, we have a trace assignment $\Pi : \mathcal{V} \to T\,R$, a partial function mapping trace variables to actual traces.

Hyperproperty

- A HyperLTL formula defines a *hyperproperty* (a set of sets of traces).
- A set T of traces satisfies the hyperproperty if it is an element of this set of sets.
- Formally, we have a trace assignment $\Pi : \mathcal{V} \to T\,R$, a partial function mapping trace variables to actual traces.

Hyperproperty

- A HyperLTL formula defines a *hyperproperty* (a set of sets of traces).
- A set T of traces satisfies the hyperproperty if it is an element of this set of sets.
- Formally, we have a trace assignment $\Pi : \mathcal{V} \to TR$, a partial function mapping trace variables to actual traces.

HyperLTL Semantics

- $\Pi[\pi \mapsto t]$ denotes that π is mapped to t with everything else mapped according to Π.
- $\Pi[i, \infty]$ denotes the trace assignment that is equal to $\Pi(\pi)[i, \infty]$ for all π.

$$
\begin{aligned}
\Pi \vDash_T \ & \exists \pi . \psi && \iff \text{there exists } t \in T : \Pi[\pi \mapsto t] \vDash_T \psi \\
\Pi \vDash_T \ & \forall \pi . \psi && \iff \text{for all } t \in T : \Pi[\pi \mapsto t] \vDash_T \psi \\
\Pi \vDash_T \ & a_\pi && \iff a \in \Pi(\pi)[0] \\
\Pi \vDash_T \ & \neg \psi && \iff \Pi \nvDash_T \psi \\
\Pi \vDash_T \ & \psi_1 \vee \psi_2 && \iff \Pi \vDash_T \psi_1 \text{ or } \Pi \vDash_T \psi_2 \\
\Pi \vDash_T \ & \bigcirc \psi && \iff \Pi[1, \infty] \vDash_T \psi \\
\Pi \vDash_T \ & \psi_1 \, \mathcal{U} \, \psi_2 && \iff \text{there exists } i \geq 0 : \Pi[i, \infty] \vDash_T \psi_2 \\
& && \quad \text{and for all } 0 \leq j < i \text{ we have } \Pi[j, \infty] \vDash_T \psi_1
\end{aligned}
$$

HyperLTL Semantics

- $\Pi[\pi \mapsto t]$ denotes that π is mapped to t with everything else mapped according to Π.
- $\Pi[i, \infty]$ denotes the trace assignment that is equal to $\Pi(\pi)[i, \infty]$ for all π.

$$
\begin{aligned}
\Pi \vDash_T \ & \exists \pi . \psi && \iff \text{there exists } t \in T : \Pi[\pi \mapsto t] \vDash_T \psi \\
\Pi \vDash_T \ & \forall \pi . \psi && \iff \text{for all } t \in T : \Pi[\pi \mapsto t] \vDash_T \psi \\
\Pi \vDash_T \ & a_\pi && \iff a \in \Pi(\pi)[0] \\
\Pi \vDash_T \ & \neg \psi && \iff \Pi \nvDash_T \psi \\
\Pi \vDash_T \ & \psi_1 \vee \psi_2 && \iff \Pi \vDash_T \psi_1 \text{ or } \Pi \vDash_T \psi_2 \\
\Pi \vDash_T \ & \bigcirc \psi && \iff \Pi[1, \infty] \vDash_T \psi \\
\Pi \vDash_T \ & \psi_1 \, \mathcal{U} \, \psi_2 && \iff \text{there exists } i \geq 0 : \Pi[i, \infty] \vDash_T \psi_2 \\
& && \quad \text{and for all } 0 \leq j < i \text{ we have } \Pi[j, \infty] \vDash_T \psi_1
\end{aligned}
$$

HyperLTL Semantics

- $\Pi[\pi \mapsto t]$ denotes that π is mapped to t with everything else mapped according to Π.
- $\Pi[i, \infty]$ denotes the trace assignment that is equal to $\Pi(\pi)[i, \infty]$ for all π.

$$
\begin{array}{lll}
\Pi \vDash_T & \exists \pi . \psi & \iff \text{there exists } t \in T : \Pi[\pi \mapsto t] \vDash_T \psi \\
\Pi \vDash_T & \forall \pi . \psi & \iff \text{for all } t \in T : \Pi[\pi \mapsto t] \vDash_T \psi \\
\Pi \vDash_T & a_\pi & \iff a \in \Pi(\pi)[0] \\
\Pi \vDash_T & \neg \psi & \iff \Pi \nvDash_T \psi \\
\Pi \vDash_T & \psi_1 \vee \psi_2 & \iff \Pi \vDash_T \psi_1 \text{ or } \Pi \vDash_T \psi_2 \\
\Pi \vDash_T & \bigcirc \psi & \iff \Pi[1, \infty] \vDash_T \psi \\
\Pi \vDash_T & \psi_1 \, \mathcal{U} \, \psi_2 & \iff \text{there exists } i \geq 0 : \Pi[i, \infty] \vDash_T \psi_2 \\
 & & \quad \text{and for all } 0 \leq j < i \text{ we have } \Pi[j, \infty] \vDash_T \psi_1
\end{array}
$$

What Now?

- We can describe properties over multiple traces.
- Like observational determinism:

$$\forall \pi . \forall \pi' . \square(I_\pi = I_{\pi'}) \implies \square(O_\pi = O_{\pi'})$$

  where I is the set of observable inputs and O is the set of observable outputs.
- Is this HyperLTL formula satisfiable?

What Now?

- We can describe properties over multiple traces.
- Like observational determinism:

$$\forall \pi . \forall \pi' . \square(I_\pi = I_{\pi'}) \implies \square(O_\pi = O_{\pi'})$$

where I is the set of observable inputs and O is the set of observable outputs.

- Is this HyperLTL formula satisfiable?

What Now?

- We can describe properties over multiple traces.
- Like observational determinism:

$$\forall \pi . \forall \pi' . \square(I_\pi = I_{\pi'}) \implies \square(O_\pi = O_{\pi'})$$

  where I is the set of observable inputs and O is the set of observable outputs.
- Is this HyperLTL formula satisfiable?

HyperLTL Satisfiability

HyperLTL-SAT is the problem of deciding whether for an arbitrary HyperLTL formula ψ there exists a non-empty set of traces T such that $\Pi \vDash_T \psi$ where Π is the empty trace assignment and $\vDash_T$ is the smallest relation satisfying the conditions of the semantics.

Overview

- ✓ Linear Temperal Logic
- ✓ HyperLTL and Hyperproperties
- 3. Complexity of HyperLTL Satisfiability
  - Alternation-free HyperLTL
  - $\exists^*\forall^*$ Fragment
  - $\forall\exists$ Fragment

Overview

- ✓ Linear Temperal Logic
- ✓ HyperLTL and Hyperproperties
- 3. Complexity of HyperLTL Satisfiability
  - Alternation-free HyperLTL: **PSPACE-complete**
  - $\exists^*\forall^*$ Fragment
  - $\forall\exists$ Fragment

Overview

- ✓ Linear Temperal Logic
- ✓ HyperLTL and Hyperproperties
- 3. Complexity of HyperLTL Satisfiability
  - Alternation-free HyperLTL: **PSPACE-complete**
  - $\exists^*\forall^*$ Fragment: **EXPSPACE-complete**
  - $\forall\exists$ Fragment

Overview

- ✓ Linear Temperal Logic
- ✓ HyperLTL and Hyperproperties
- 3. Complexity of HyperLTL Satisfiability
  - Alternation-free HyperLTL: **PSPACE-complete**
  - $\exists^*\forall^*$ Fragment: **EXPSPACE-complete**
  - $\forall\exists$ Fragment: **Undecidable**

Alternation-free HyperLTL

Definition

A HyperLTL formula ψ is *alternation-free* if the quantifier prefix consists only of either universal or only existential quantifiers, denoted by $\forall^*$ and $\exists^*$.

- We want to show HyperLTL-SAT is PSPACE-complete for alternation-free HyperLTL formulas.

Alternation-free HyperLTL

Definition

A HyperLTL formula ψ is *alternation-free* if the quantifier prefix consists only of either universal or only existential quantifiers, denoted by $\forall^*$ and $\exists^*$.

- We want to show HyperLTL-SAT is PSPACE-complete for alternation-free HyperLTL formulas.

Equisatisfiable

- Reduction to an *equisatisfiable* LTL formula of the same size. And using the fact that LTL-SAT is PSPACE-complete.

$$\forall \psi \in \text{Alternation-free}. \left(\exists T. \Pi \vDash_T \psi \iff \exists t. t \vDash \hat{\psi} \right)$$

The $\forall^*$ Fragment

$$\forall \pi_1 \ldots \forall \pi_n . \varphi$$

- Restrict models to singleton sets of traces.
- Ignore trace variables and quantifiers and interpret the HyperLTL formula as plain LTL.

The $\forall^*$ Fragment

$$\forall \pi_1 \ldots \forall \pi_n . \varphi$$

- Restrict models to singleton sets of traces.
- Ignore trace variables and quantifiers and interpret the HyperLTL formula as plain LTL.

The $\forall^*$ Fragment

$$\forall \pi_1 \ldots \forall \pi_n . \varphi$$

- Restrict models to singleton sets of traces.
- Ignore trace variables and quantifiers and interpret the HyperLTL formula as plain LTL.

The $\forall^*$ Fragment

Example

$$\forall \pi_1 \forall \pi_2 . \Box b_{\pi_1} \wedge \Box \neg b_{\pi_2}$$

becomes

$$\Box b \wedge \Box \neg b$$

The $\forall^*$ Fragment

Example

$$\forall \pi_1 \forall \pi_2 . \Box b_{\pi_1} \wedge \Box \neg b_{\pi_2}$$

becomes

$$\Box b \wedge \Box \neg b$$

The $\forall^*$ Fragment

Lemma

HyperLTL-SAT is PSPACE-complete for the $\forall^*$ fragment.

PSPACE: Equisatifiable LTL formula in polynomial space.

hardness: Reduce LTL-SAT to HyperLTL-SAT:

$$\varphi \text{ is satisfiable by } t \iff \forall \pi . \varphi(\pi) \text{ is satisfiable by } \{t\}$$

The $\forall^*$ Fragment

Lemma

HyperLTL-SAT is PSPACE-complete for the $\forall^*$ fragment.

PSPACE: Equisatifiable LTL formula in polynomial space.

hardness: Reduce LTL-SAT to HyperLTL-SAT:

$$\varphi \text{ is satisfiable by } t \iff \forall \pi . \varphi(\pi) \text{ is satisfiable by } \{t\}$$

The $\forall^*$ Fragment

Lemma

HyperLTL-SAT is PSPACE-complete for the $\forall^*$ fragment.

PSPACE: Equisatifiable LTL formula in polynomial space.

hardness: Reduce LTL-SAT to HyperLTL-SAT:

$$\varphi \text{ is satisfiable by } t \iff \forall \pi . \varphi(\pi) \text{ is satisfiable by } \{t\}$$

The $\exists^*$ Fragment

$$\exists \pi_1 \ldots \exists \pi_n . \varphi$$

- Models may have more than one trace: $\exists \pi_1 \exists \pi_2 . a_{\pi_1} \wedge \neg a_{\pi_2}$.
- Reduction by *zipping* traces together.

The $\exists^*$ Fragment

$$\exists \pi_1 \ldots \exists \pi_n . \varphi$$

- Models may have more than one trace: $\exists \pi_1 \exists \pi_2 . a_{\pi_1} \wedge \neg a_{\pi_2}$.
- Reduction by *zipping* traces together.

The $\exists^*$ Fragment

$$\exists \pi_1 \ldots \exists \pi_n . \varphi$$

- Models may have more than one trace: $\exists \pi_1 \exists \pi_2 . a_{\pi_1} \wedge \neg a_{\pi_2}$.
- Reduction by *zipping* traces together.

The $\exists^*$ Fragment

Example

Consider $AP := \{a, b\}$

$$\exists \pi_1 \exists \pi_2 . a_{\pi_1} \wedge \square \neg b_{\pi_1} \wedge \square b_{\pi_2} \tag{1}$$

converts to equisatisfiable LTL over $AP := \{a_1, b_1, b_2\}$

$$a_1 \wedge \square \neg b_1 \wedge \square b_2$$

which is satisfied by $t = (\{a_1, b_2\})^\omega$. $T = \{(\{a\})^\omega, (\{b\})^\omega\}$ satisfies (1).

The $\exists^*$ Fragment

Example

Consider $AP := \{a, b\}$

$$\exists \pi_1 \exists \pi_2 . a_{\pi_1} \wedge \square \neg b_{\pi_1} \wedge \square b_{\pi_2} \tag{1}$$

converts to equisatisfiable LTL over $AP := \{a_1, b_1, b_2\}$

$$a_1 \wedge \square \neg b_1 \wedge \square b_2$$

which is satisfied by $t = (\{a_1, b_2\})^\omega$. $T = \{(\{a\})^\omega, (\{b\})^\omega\}$ satisfies (1).

The $\exists^*$ Fragment

Example

Consider $AP := \{a, b\}$

$$\exists \pi_1 \exists \pi_2 . a_{\pi_1} \wedge \square \neg b_{\pi_1} \wedge \square b_{\pi_2} \tag{1}$$

converts to equisatisfiable LTL over $AP := \{a_1, b_1, b_2\}$

$$a_1 \wedge \square \neg b_1 \wedge \square b_2$$

which is satisfied by $t = (\{a_1, b_2\})^\omega$. $T = \{(\{a\})^\omega, (\{b\})^\omega\}$ satisfies (1).

The $\exists^*$ Fragment

Example

Consider $AP := \{a, b\}$

$$\exists \pi_1 \exists \pi_2 . a_{\pi_1} \wedge \square \neg b_{\pi_1} \wedge \square b_{\pi_2} \tag{1}$$

converts to equisatisfiable LTL over $AP := \{a_1, b_1, b_2\}$

$$a_1 \wedge \square \neg b_1 \wedge \square b_2$$

which is satisfied by $t = (\{a_1, b_2\})^\omega$. $T = \{(\{a\})^\omega, (\{b\})^\omega\}$ satisfies (1).

The $\exists^*$ Fragment

Lemma

HyperLTL-SAT is PSPACE-complete for the $\exists^*$ fragment.

PSPACE: Equisatifiable LTL formula in polynomial space.

hardness: Reduce LTL-SAT to HyperLTL-SAT:

$$\varphi \text{ is satisfiable by } t \iff \exists \pi . \varphi(\pi) \text{ is satisfiable by } \{t\}$$

The $\exists^*$ Fragment

Lemma

HyperLTL-SAT is PSPACE-complete for the $\exists^*$ fragment.

PSPACE: Equisatifiable LTL formula in polynomial space.

hardness: Reduce LTL-SAT to HyperLTL-SAT:

$$\varphi \text{ is satisfiable by } t \iff \exists \pi . \varphi(\pi) \text{ is satisfiable by } \{t\}$$

The $\exists^*$ Fragment

Lemma

HyperLTL-SAT is PSPACE-complete for the $\exists^*$ fragment.

PSPACE: Equisatifiable LTL formula in polynomial space.

hardness: Reduce LTL-SAT to HyperLTL-SAT:

$$\varphi \text{ is satisfiable by } t \iff \exists \pi . \varphi(\pi) \text{ is satisfiable by } \{t\}$$

Overview

- ✓ Linear Temperal Logic
- ✓ HyperLTL and Hyperproperties
- 3. Complexity of HyperLTL Satisfiability
  - ✓ Alternation-free HyperLTL: **PSPACE-complete**
  - $\exists^*\forall^*$ Fragment: **EXPSPACE-complete**
  - $\forall\exists$ Fragment: **Undecidable**

The $\exists^*\forall^*$ Fragment

Definition

A HyperLTL formula is a $\exists^*\forall^*$ fragment if its of the form $\exists\pi_1 \ldots \exists\pi_n \forall\pi'_1 \ldots \forall\pi'_m.\varphi$

Lemma

HyperLTL-SAT is EXPSPACE-complete for the $\exists^*\forall^*$ fragment.

- Bounded $\exists^*\forall\pi_1 \ldots \forall\pi_b$ is PSPACE-complete.
- Checking implication between alternation-free HyperLTL formulas is EXPSPACE-complete.

The $\exists^*\forall^*$ Fragment

Definition

A HyperLTL formula is a $\exists^*\forall^*$ fragment if its of the form $\exists\pi_1 \ldots \exists\pi_n \forall\pi'_1 \ldots \forall\pi'_m.\varphi$

Lemma

HyperLTL-SAT is EXPSPACE-complete for the $\exists^*\forall^*$ fragment.

- Bounded $\exists^*\forall\pi_1 \ldots \forall\pi_b$ is PSPACE-complete.
- Checking implication between alternation-free HyperLTL formulas is EXPSPACE-complete.

The $\exists^*\forall^*$ Fragment

Definition

A HyperLTL formula is a $\exists^*\forall^*$ fragment if its of the form $\exists\pi_1 \ldots \exists\pi_n \forall\pi'_1 \ldots \forall\pi'_m.\varphi$

Lemma

HyperLTL-SAT is EXPSPACE-complete for the $\exists^*\forall^*$ fragment.

- Bounded $\exists^*\forall\pi_1 \ldots \forall\pi_b$ is PSPACE-complete.
- Checking implication between alternation-free HyperLTL formulas is EXPSPACE-complete.

The $\exists^*\forall^*$ Fragment

Definition

A HyperLTL formula is a $\exists^*\forall^*$ fragment if its of the form $\exists\pi_1 \ldots \exists\pi_n \forall\pi'_1 \ldots \forall\pi'_m.\varphi$

Lemma

HyperLTL-SAT is EXPSPACE-complete for the $\exists^*\forall^*$ fragment.

- Bounded $\exists^*\forall\pi_1 \ldots \forall\pi_b$ is PSPACE-complete.
- Checking implication between alternation-free HyperLTL formulas is EXPSPACE-complete.

Overview

- ✓ Linear Temperal Logic
- ✓ HyperLTL and Hyperproperties
- 3. Complexity of HyperLTL Satisfiability
  - ✓ Alternation-free HyperLTL: **PSPACE-complete**
  - ✓ $\exists^*\forall^*$ Fragment: **EXPSPACE-complete**
  - • $\forall\exists$ Fragment: **Undecidable**

The ∀∃ fragment

- Any extension makes satisfiability problem undecidable.
- Reduction from Post's Correspondence Problem

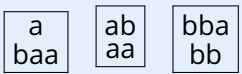
The ∀∃ fragment

- Any extension makes satisfiability problem undecidable.
- Reduction from Post's Correspondence Problem

Post's Correspondence Problem

Intuition

- Finite amount of domino stones over some alphabet:

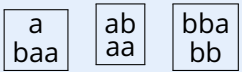


- Find sequence of stones such that the top and bottom have the same word.
- For example $(3, 2, 3, 1)$ is a solution since:

Post's Correspondence Problem

Intuition

- Finite amount of domino stones over some alphabet:

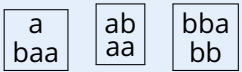


- Find sequence of stones such that the top and bottom have the same word.
- For example $(3, 2, 3, 1)$ is a solution since:

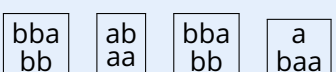
Post's Correspondence Problem

Intuition

- Finite amount of domino stones over some alphabet:



- Find sequence of stones such that the top and bottom have the same word.
- For example $(3, 2, 3, 1)$ is a solution since:



Reduction

- Pairs of the alphabet Σ as AP, e.g. (a, b).
- Encode stones as sequence of such pairs, creating a trace.

Example

$(a, b), (\#, b), (\#, a)$

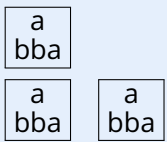
$(\dot{a}, \dot{b}), (\dot{a}, b), (\#, a), (\#, \dot{b}), (\#, b), (\#, a)$

- $\tilde{a}$ if we do not care if a or $\dot{a}$ and $*$ for arbitrary letter from the alphabet.

Reduction

- Pairs of the alphabet Σ as AP, e.g. (a, b).
- Encode stones as sequence of such pairs, creating a trace.

Example



$(a, b), (\#, b), (\#, a)$

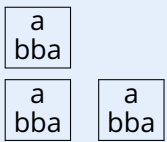
$(\dot{a}, \dot{b}), (\dot{a}, b), (\#, a), (\#, \dot{b}), (\#, b), (\#, a)$

- $\tilde{a}$ if we do not care if a or $\dot{a}$ and $*$ for arbitrary letter from the alphabet.

Reduction

- Pairs of the alphabet Σ as AP, e.g. (a, b).
- Encode stones as sequence of such pairs, creating a trace.

Example



$(a, b), (\#, b), (\#, a)$

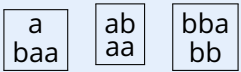
$(\dot{a}, \dot{b}), (\dot{a}, b), (\#, a), (\#, \dot{b}), (\#, b), (\#, a)$

- $\tilde{a}$ if we do not care if a or $\dot{a}$ and $*$ for arbitrary letter from the alphabet.

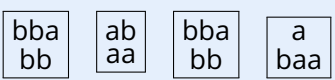
Reduction

Example

$AP := \{a, b, \dot{a}, \dot{b}, \#\}^2$



With solution (3,2,3,1)



This reduces to the following HyperLTL formula

HyperLTL Formula

$$\forall \pi \exists \pi_s \exists \pi' . \Big(((\dot{a}, \dot{a})_{\pi_s} \vee (\dot{b}, \dot{b})_{\pi_s}) \\ \wedge ((\tilde{a}, \tilde{a})_{\pi_s} \vee (\tilde{b}, \tilde{b})_{\pi_s}) \, \mathcal{U} \, \square(\#, \#)_{\pi_s} \Big) \\ \wedge \Diamond \square (\#, \#)_{\pi} \\ \wedge \Big(\bigvee_{i \in \{1,2,3\}} StoneEncoding_i(\pi, \pi') \\ \vee \square(\#, \#)_{\pi} \Big)$$

HyperLTL Formula

$$\forall \pi \exists \pi_s \exists \pi' . \Big(((\dot{a}, \dot{a})_{\pi_s} \vee (\dot{b}, \dot{b})_{\pi_s}) \wedge ((\tilde{a}, \tilde{a})_{\pi_s} \vee (\tilde{b}, \tilde{b})_{\pi_s}) \mathcal{U} \square(\#, \#)_{\pi_s} \Big) \wedge \Diamond\square(\#, \#)_{\pi} \wedge \Big(\bigvee_{i \in \{1,2,3\}} StoneEncoding_i(\pi, \pi') \vee \square(\#, \#)_{\pi} \Big)$$

$\dot{a}$: new stone starting a

$\tilde{a}$: new stone or not

$*$: arbitrary letter

HyperLTL Formula

$$\forall \pi \exists \pi_s \exists \pi' . \Big(((\dot{a}, \dot{a})_{\pi_s} \vee (\dot{b}, \dot{b})_{\pi_s}) \wedge ((\tilde{a}, \tilde{a})_{\pi_s} \vee (\tilde{b}, \tilde{b})_{\pi_s}) \, \mathcal{U} \, \square(\#, \#)_{\pi_s} \Big) \wedge \diamond\square(\#, \#)_{\pi} \wedge \Big(\bigvee_{i \in \{1,2,3\}} StoneEncoding_i(\pi, \pi') \vee \square(\#, \#)_{\pi} \Big)$$

$\dot{a}$: new stone starting a

$\tilde{a}$: new stone or not

$*$: arbitrary letter

HyperLTL Formula

$$\forall \pi \exists \pi_s \exists \pi' . \Big(((\dot{a}, \dot{a})_{\pi_s} \vee (\dot{b}, \dot{b})_{\pi_s}) \wedge ((\tilde{a}, \tilde{a})_{\pi_s} \vee (\tilde{b}, \tilde{b})_{\pi_s}) \, \mathcal{U} \, \square(\#, \#)_{\pi_s} \Big) \wedge \Diamond\square(\#, \#)_{\pi} \wedge \Big(\bigvee_{i \in \{1,2,3\}} StoneEncoding_i(\pi, \pi') \vee \square(\#, \#)_{\pi} \Big)$$

$\dot{a}$: new stone starting a

$\tilde{a}$: new stone or not

$*$: arbitrary letter

HyperLTL Formula

$$\forall \pi \exists \pi_s \exists \pi' . \Big(((\dot{a}, \dot{a})_{\pi_s} \vee (\dot{b}, \dot{b})_{\pi_s}) \wedge ((\tilde{a}, \tilde{a})_{\pi_s} \vee (\tilde{b}, \tilde{b})_{\pi_s}) \, \mathcal{U} \, \square(\#, \#)_{\pi_s} \Big) \wedge \Diamond\square(\#, \#)_{\pi} \wedge \Big(\bigvee_{i \in \{1,2,3\}} StoneEncoding_i(\pi, \pi') \vee \square(\#, \#)_{\pi} \Big)$$

$\dot{a}$: new stone starting a

$\tilde{a}$: new stone or not

$*$: arbitrary letter

HyperLTL Formula

$$\forall \pi \exists \pi_s \exists \pi' . \Big(((\dot{a}, \dot{a})_{\pi_s} \vee (\dot{b}, \dot{b})_{\pi_s}) \wedge ((\tilde{a}, \tilde{a})_{\pi_s} \vee (\tilde{b}, \tilde{b})_{\pi_s}) \, \mathcal{U} \, \square(\#, \#)_{\pi_s} \Big) \wedge \Diamond\square(\#, \#)_{\pi} \wedge \Big(\bigvee_{i \in \{1,2,3\}} StoneEncoding_i(\pi, \pi') \vee \square(\#, \#)_{\pi} \Big)$$

$\dot{a}$: new stone starting a

$\tilde{a}$: new stone or not

$*$: arbitrary letter

Trace Set

Start

$\pi_s : (\dot{b}, \dot{b})(b, b)(a,\dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \ldots$

Delete stone 3

$\pi'_s : (\dot{a},\dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 2

$\pi''_s : (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

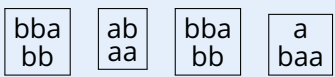
Delete stone 3

$\pi'''_s : (\dot{a}, \dot{b})(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 1

$\pi''''_s : (\#, \#)(\#, \#) \ldots$

End



Trace Set

Start

$\pi_s : (\dot{b}, \dot{b})(b, b)(a,\dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \ldots$

Delete stone 3

$\pi'_s : (\dot{a},\dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 2

$\pi''_s : (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

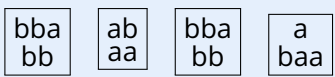
Delete stone 3

$\pi'''_s : (\dot{a}, \dot{b})(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 1

$\pi''''_s : (\#, \#)(\#, \#) \ldots$

End



Trace Set

Start

$\pi_s : (\dot{b}, \dot{b})(b, b)(a,\dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \ldots$

Delete stone 3

$\pi'_s : (\dot{a},\dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 2

$\pi''_s : (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

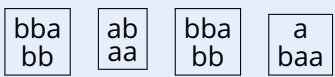
Delete stone 3

$\pi'''_s : (\dot{a}, \dot{b})(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 1

$\pi''''_s : (\#, \#)(\#, \#) \ldots$

End



Trace Set

Start

$\pi_s : (\dot{b}, \dot{b})(b, b)(a, \dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \ldots$

Delete stone 3

$\pi'_s : (\dot{a}, \dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 2

$\pi''_s : (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

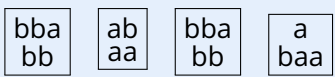
Delete stone 3

$\pi'''_s : (\dot{a}, \dot{b})(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 1

$\pi''''_s : (\#, \#)(\#, \#) \ldots$

End



Trace Set

Start

$\pi_s : (\dot{b}, \dot{b})(b, b)(a, \dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \ldots$

Delete stone 3

$\pi'_s : (\dot{a}, \dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 2

$\pi''_s : (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

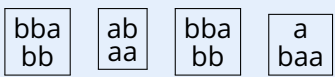
Delete stone 3

$\pi'''_s : (\dot{a}, \dot{b})(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 1

$\pi''''_s : (\#, \#)(\#, \#) \ldots$

End



Trace Set

Start

$\pi_s : (\dot{b}, \dot{b})(b, b)(a, \dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \ldots$

Delete stone 3

$\pi'_s : (\dot{a}, \dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 2

$\pi''_s : (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

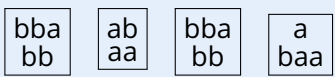
Delete stone 3

$\pi'''_s : (\dot{a}, \dot{b})(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 1

$\pi''''_s : (\#, \#)(\#, \#) \ldots$

End



Trace Set

Start

$\pi_s : (\dot{b}, \dot{b})(b, b)(a, \dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \ldots$

Delete stone 3

$\pi'_s : (\dot{a}, \dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 2

$\pi''_s : (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

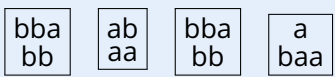
Delete stone 3

$\pi'''_s : (\dot{a}, \dot{b})(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 1

$\pi''''_s : (\#, \#)(\#, \#) \ldots$

End



Trace Set

Start

$\pi_s : (\dot{b}, \dot{b})(b, b)(a, \dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \ldots$

Delete stone 3

$\pi'_s : (\dot{a}, \dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 2

$\pi''_s : (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

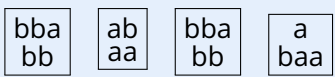
Delete stone 3

$\pi'''_s : (\dot{a}, \dot{b})(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 1

$\pi''''_s : (\#, \#)(\#, \#) \ldots$

End



Trace Set

Start

$\pi_s : (\dot{b}, \dot{b})(b, b)(a, \dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \ldots$

Delete stone 3

$\pi'_s : (\dot{a}, \dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 2

$\pi''_s : (\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \ldots$

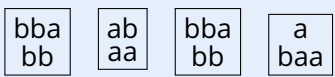
Delete stone 3

$\pi'''_s : (\dot{a}, \dot{b})(\#, a)(\#, \#)(\#, \#) \ldots$

Delete stone 1

$\pi''''_s : (\#, \#)(\#, \#) \ldots$

End



$StoneEncoding_3(\pi, \pi') =$

$$\Big(\big((\dot{b}, \dot{b})_\pi \wedge \bigcirc(b, b)_\pi \wedge \bigcirc\bigcirc(a, \dot{*})_\pi \wedge \bigcirc\bigcirc\bigcirc(\dot{*}, \tilde{*})_\pi\big)$$

$$\vee\ ((\dot{b}, \dot{b})_\pi \wedge \bigcirc(b, b)_\pi \wedge \bigcirc\bigcirc(a, \#)_\pi \wedge \bigcirc\bigcirc\bigcirc(\#, \#)_\pi)\Big)$$

$$\wedge\ \Box(\bigcirc\bigcirc\bigcirc(\tilde{a}, \tilde{*})_\pi \implies (\tilde{a}, \tilde{*})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc\bigcirc(\tilde{b}, \tilde{*})_\pi \implies (\tilde{b}, \tilde{*})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc\bigcirc(\#, \tilde{*})_\pi \implies (\#, \tilde{*})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc(\tilde{*}, \tilde{a})_\pi \implies (\tilde{*}, \tilde{a})_{\pi'})$$

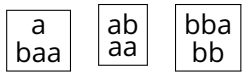
$$\wedge\ \Box(\bigcirc\bigcirc(\tilde{*}, \tilde{b})_\pi \implies (\tilde{*}, \tilde{b})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc(\tilde{*}, \#)_\pi \implies (\tilde{*}, \#)_{\pi'})\Big)$$

$\dot{a}$: new stone starting a

$\tilde{a}$: new stone or not

$*$: arbitrary letter



$StoneEncoding_3(\pi, \pi') =$

$$\Big(\big((\dot{b}, \dot{b})_\pi \wedge \bigcirc(b, b)_\pi \wedge \bigcirc\bigcirc(a, \dot{*})_\pi \wedge \bigcirc\bigcirc\bigcirc(\dot{*}, \tilde{*})_\pi\big)$$

$$\vee\ ((\dot{b}, \dot{b})_\pi \wedge \bigcirc(b, b)_\pi \wedge \bigcirc\bigcirc(a, \#)_\pi \wedge \bigcirc\bigcirc\bigcirc(\#, \#)_\pi)\Big)$$

$$\wedge\ \Box(\bigcirc\bigcirc\bigcirc(\tilde{a}, \tilde{*})_\pi \implies (\tilde{a}, \tilde{*})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc\bigcirc(\tilde{b}, \tilde{*})_\pi \implies (\tilde{b}, \tilde{*})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc\bigcirc(\#, \tilde{*})_\pi \implies (\#, \tilde{*})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc(\tilde{*}, \tilde{a})_\pi \implies (\tilde{*}, \tilde{a})_{\pi'})$$

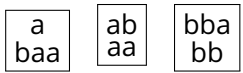
$$\wedge\ \Box(\bigcirc\bigcirc(\tilde{*}, \tilde{b})_\pi \implies (\tilde{*}, \tilde{b})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc(\tilde{*}, \#)_\pi \implies (\tilde{*}, \#)_{\pi'})\Big)$$

$\dot{a}$: new stone starting a

$\tilde{a}$: new stone or not

$*$: arbitrary letter



$StoneEncoding_3(\pi, \pi') =$

$$\Big(\big((\dot{b}, \dot{b})_\pi \wedge \bigcirc(b, b)_\pi \wedge \bigcirc\bigcirc(a, \dot{*})_\pi \wedge \bigcirc\bigcirc\bigcirc(\dot{*}, \tilde{*})_\pi\big)$$

$$\vee\ ((\dot{b}, \dot{b})_\pi \wedge \bigcirc(b, b)_\pi \wedge \bigcirc\bigcirc(a, \#)_\pi \wedge \bigcirc\bigcirc\bigcirc(\#, \#)_\pi)\Big)$$

$$\wedge\ \Box(\bigcirc\bigcirc\bigcirc(\tilde{a}, \tilde{*})_\pi \implies (\tilde{a}, \tilde{*})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc\bigcirc(\tilde{b}, \tilde{*})_\pi \implies (\tilde{b}, \tilde{*})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc\bigcirc(\#, \tilde{*})_\pi \implies (\#, \tilde{*})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc(\tilde{*}, \tilde{a})_\pi \implies (\tilde{*}, \tilde{a})_{\pi'})$$

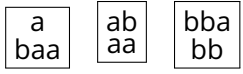
$$\wedge\ \Box(\bigcirc\bigcirc(\tilde{*}, \tilde{b})_\pi \implies (\tilde{*}, \tilde{b})_{\pi'})$$

$$\wedge\ \Box(\bigcirc\bigcirc(\tilde{*}, \#)_\pi \implies (\tilde{*}, \#)_{\pi'})\Big)$$

$\dot{a}$: new stone starting a

$\tilde{a}$: new stone or not

$*$: arbitrary letter



Undecidability of $\forall\exists$

Lemma

HyperLTL-SAT is undecidable for the $\forall\exists$ fragment.

$$\varphi_{reduc} := \forall\pi.\exists\pi_s.\exists\pi'.\varphi_{sol}(\pi_s) \wedge \varphi_{validStone}(\pi) \wedge \varphi_{delete}(\pi,\pi') \wedge \Diamond\Box(\#,\#)_\pi$$

PCP instance has solution $i \iff \varphi_{reduc}$ is satisfied by a trace set T

$\Rightarrow$ Given a solutions consisting of a sequence of domino stones construct the solution traces and all truncated traces. This gives a trace set T.

$\Leftarrow$ By construction, there exists $T_{min} \subseteq T$. Order traces by the amount of dots*, i.e. domino stones and reconstruct the solution.

* Use reduction from $\exists^*$ fragment.

Undecidability of $\forall\exists$

Lemma

HyperLTL-SAT is undecidable for the $\forall\exists$ fragment.

$$\varphi_{reduc} := \forall\pi.\exists\pi_s.\exists\pi'.\varphi_{sol}(\pi_s) \wedge \varphi_{validStone}(\pi) \wedge \varphi_{delete}(\pi, \pi') \wedge \Diamond\Box(\#, \#)_\pi$$

PCP instance has solution i $\iff$ φ_{reduc} is satisfied by a trace set T

$\Rightarrow$ Given a solutions consisting of a sequence of domino stones construct the solution traces and all truncated traces. This gives a trace set T.

$\Leftarrow$ By construction, there exists $T_{min} \subseteq T$. Order traces by the amount of dots*, i.e. domino stones and reconstruct the solution.

* Use reduction from $\exists^*$ fragment.

Undecidability of $\forall\exists$

Lemma

HyperLTL-SAT is undecidable for the $\forall\exists$ fragment.

$$\varphi_{reduc} := \forall\pi.\exists\pi_s.\exists\pi'.\varphi_{sol}(\pi_s) \wedge \varphi_{validStone}(\pi) \wedge \varphi_{delete}(\pi, \pi') \wedge \Diamond\Box(\#, \#)_\pi$$

PCP instance has solution i $\iff$ φ_{reduc} is satisfied by a trace set T

$\Rightarrow$ Given a solutions consisting of a sequence of domino stones construct the solution traces and all truncated traces. This gives a trace set T.

$\Leftarrow$ By construction, there exists $T_{\text{min}} \subseteq T$. Order traces by the amount of dots $\dot{}$, i.e. domino stones and reconstruct the solution.

- Use reduction from $\exists^*$ fragment.

Undecidability of $\forall\exists$

Lemma

HyperLTL-SAT is undecidable for the $\forall\exists$ fragment.

$$\varphi_{reduc} := \forall\pi.\exists\pi_s.\exists\pi'.\varphi_{sol}(\pi_s) \wedge \varphi_{validStone}(\pi) \wedge \varphi_{delete}(\pi, \pi') \wedge \Diamond\Box(\#, \#)_\pi$$

PCP instance has solution i $\iff$ φ_{reduc} is satisfied by a trace set T

$\Rightarrow$ Given a solutions consisting of a sequence of domino stones construct the solution traces and all truncated traces. This gives a trace set T.

$\Leftarrow$ By construction, there exists $T_{\text{min}} \subseteq T$. Order traces by the amount of dots $\dot{}$, i.e. domino stones and reconstruct the solution.

- Use reduction from $\exists^*$ fragment.

Undecidability of $\forall\exists$

Lemma

HyperLTL-SAT is undecidable for the $\forall\exists$ fragment.

$$\varphi_{reduc} := \forall\pi.\exists\pi_s.\exists\pi'.\varphi_{sol}(\pi_s) \wedge \varphi_{validStone}(\pi) \wedge \varphi_{delete}(\pi, \pi') \wedge \Diamond\Box(\#, \#)_\pi$$

PCP instance has solution i $\iff$ φ_{reduc} is satisfied by a trace set T

- $\Rightarrow$ Given a solutions consisting of a sequence of domino stones construct the solution traces and all truncated traces. This gives a trace set T.
- $\Leftarrow$ By construction, there exists $T_{\text{min}} \subseteq T$. Order traces by the amount of dots $\dot{}$, i.e. domino stones and reconstruct the solution.
- Use reduction from $\exists^*$ fragment.

Undecidability of $\forall\exists$

Lemma

HyperLTL-SAT is undecidable for the $\forall\exists$ fragment.

$$\varphi_{reduc} := \forall\pi.\exists\pi_s.\exists\pi'.\varphi_{sol}(\pi_s) \wedge \varphi_{validStone}(\pi) \wedge \varphi_{delete}(\pi, \pi') \wedge \Diamond\Box(\#, \#)_\pi$$

PCP instance has solution $i \iff \varphi_{reduc}$ is satisfied by a trace set T

- $\Rightarrow$ Given a solutions consisting of a sequence of domino stones construct the solution traces and all truncated traces. This gives a trace set T.
- $\Leftarrow$ By construction, there exists $T_{\text{min}} \subseteq T$. Order traces by the amount of dots $\dot{}$, i.e. domino stones and reconstruct the solution.
- Use reduction from $\exists^*$ fragment.

Conclusions

- Hyperproperties:
  - Properties of sets of traces, captured using **HyperLTL**.
  - Extends LTL with **trace variables** and **quantifiers** ($\forall$, $\exists$).
  - Analyse multi-trace behaviours like *observational determinism*.
- Complexity Results:
  - LTL-SAT: **PSPACE-complete**.
  - HyperLTL-SAT:
    - Alternation-free ($\forall^*$ or $\exists^*$): **PSPACE-complete**.
    - $\exists^*\forall^*$: **EXPSPACE-complete**.
    - $\forall\exists$: **Undecidable**.

Conclusions

- Hyperproperties:
  - Properties of sets of traces, captured using **HyperLTL**.
  - Extends LTL with **trace variables** and **quantifiers** ($\forall$, $\exists$).
  - Analyse multi-trace behaviours like *observational determinism*.
- Complexity Results:
  - LTL-SAT: **PSPACE-complete**.
  - HyperLTL-SAT:
    - Alternation-free ($\forall^*$ or $\exists^*$): **PSPACE-complete**.
    - $\exists^*\forall^*$: **EXPSPACE-complete**.
    - $\forall\exists$: **Undecidable**.