Runtime Verification for Alternation-Free HyperLTL

Borzoo Bonakdarpour
joint work with Noel Brett and Umair Siddique

McMaster University, ON., Canada

September 27, 2016

Real Motivation!

EDAS Conference and Journal Management System

Click on the menu items above to submit and review papers.

Please indicate whether you want to receive call-for-papers by updating your areas of interest.

Your conflicts-of-interest have not been updated in the last three months. (Persons with conflicts-of-interest are those who should not review p
the same institution.)

My pending, active and accepted papers

Only papers for upcoming conferences are shown.

Conference	Paper title (details)	Abstract or manuscript deadline	Edit	Add and delete authors	Upload paper	Files	Withdraw	Session
[illegible]	[redacted]	February 2, 2015 Anywhere on Earth			final deadline			(not yet assigned)
[illegible]	[redacted]	October 18, 2014 Anywhere on Earth			paper status			
[illegible]	[redacted]	October 18, 2014 Anywhere on Earth			withdrawn			
[illegible]	[redacted]	December 23, 2014 Anywhere on Earth			paper deadline			(not yet assigned)
[illegible]	[redacted]	December 23, 2014 Anywhere on Earth			paper deadline			

S. Agrawal and B. Bonakdarpour. **Runtime Verification of k-safety Hyperproperties in HyperLTL** (CSF 2016).

Why Alternation-free HyperLTL?

Consider formula

$$\forall\pi.\exists\pi'.\varphi$$

To reason about such a formula, we should have all traces.

This cannot be done by runtime techniques only.

Why Alternation-free HyperLTL?

Consider formula

$$\forall \pi . \exists \pi' . \varphi$$

To reason about such a formula, we should have all traces.

This cannot be done by runtime techniques only.

Consider formula

$$\forall \pi . \forall \pi' . \varphi$$

One can detect violation of such a formula by discovering two traces that do not satisfy φ.

Presentation outline

1. Finite Semantics for LTL

2. Challenges in RV for HyperLTL

3. Rewriting-based RV Algorithm for Alternation-free HyperLTL Formulas
   - Identifying the Propositions of Interest
   - Rewriting-based RV for FLTL

4. Conclusion

Framework

Definitions

Let AP be a set of atomic propositions and $\Sigma = 2^{AP}$ be the alphabet.

Framework

Definitions

Let AP be a set of atomic propositions and $\Sigma = 2^{AP}$ be the alphabet.

A word is a sequence $w = a_0 a_1 \cdots$, where each a_i $(i \geq 0)$ is a letter (or event) in Σ.

The set of all finite (respectively, infinite) words are Σ^* (respectively, Σ^ω).

For a word $w = a_0 a_1 \cdots$, w^i means the denote the suffix $a_i a_{i+1} \cdots$.

Framework

Definitions

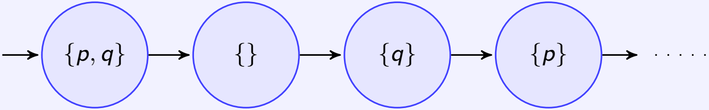
Let AP be a set of atomic propositions and $\Sigma = 2^{AP}$ be the alphabet.

A word is a sequence $w = a_0 a_1 \cdots$, where each a_i ($i \geq 0$) is a letter (or event) in Σ.

The set of all finite (respectively, infinite) words are Σ^* (respectively, Σ^ω).

For a word $w = a_0 a_1 \cdots$, w^i means the denote the suffix $a_i a_{i+1} \cdots$.

Example



Finite LTL (FLTL [Manna, Pnueli - 95)

The semantics of LTL is defined over infinite words.

Finite LTL (FLTL [Manna, Pnueli - 95)

The semantics of LTL is defined over infinite words.

Finite LTL

Finite LTL (FLTL) allows us to reason about finite words for verifying properties at run time.

Finite LTL (FLTL [Manna, Pnueli - 95)

The semantics of LTL is defined over infinite words.

Finite LTL

Finite LTL (FLTL) allows us to reason about finite words for verifying properties at run time.

FLTL Syntax

The syntax of FLTL is identical to that of LTL and the semantics is based on the truth values $\mathbb{B}_2 = \{\bot, \top\}$.

Finite LTL (FLTL [Manna, Pnueli - 95)

The semantics of LTL is defined over infinite words.

Finite LTL

Finite LTL (FLTL) allows us to reason about finite words for verifying properties at run time.

FLTL Syntax

The syntax of FLTL is identical to that of LTL and the semantics is based on the truth values $\mathbb{B}_2 = \{\perp, \top\}$.

FLTL Semantics

The semantics of FLTL for atomic propositions and Boolean operators are identical to those of LTL.

Finite LTL

FLTL Semantics

Let φ, φ_1, and φ_2 be LTL formulas, and $u = u_0 u_1 \cdots u_n$ be a finite word.

$$[u \models_F \mathbf{X}\, \varphi] = \begin{cases} [u^1 \models_F \varphi] & \text{if} \quad u^1 \neq \epsilon \\ \bot & \text{otherwise} \end{cases}$$

$$[u \models_F \bar{\mathbf{X}}\, \varphi] = \begin{cases} [u^1 \models_F \varphi] & \text{if} \quad u^1 \neq \epsilon \\ \top & \text{otherwise} \end{cases}$$

$$[u \models_F \varphi_1 \,\mathbf{U}\, \varphi_2] = \begin{cases} \top & \text{if} \quad \exists k \in [0, n] : [u^k \models_F \varphi_2] = \top \;\wedge \\ & \qquad \forall l \in [0, k) : [u^l \models_F \varphi_1] = \top \\ \bot & \text{otherwise} \end{cases}$$

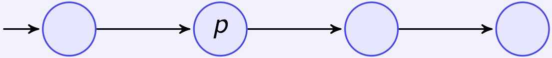
FLTL

Example

FLTL

Example

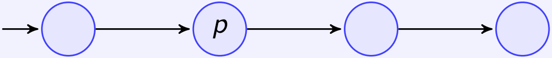
$[u \models_F \mathbf{X}p] = \top$



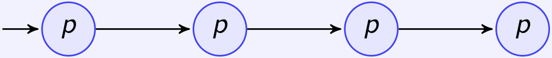
FLTL

Example

$[u \models_F \mathbf{X} p] = \top$



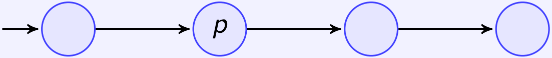
$[u \models_F p \, \mathbf{U} \, q] = \bot$



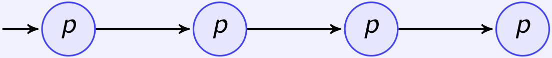
FLTL

Example

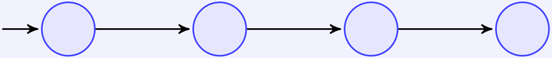
$[u \models_F \mathbf{X} p] = \top$



$[u \models_F p \, \mathbf{U} \, q] = \bot$



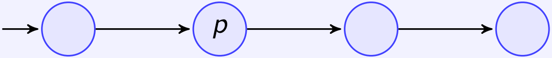
$[u \models_F \mathbf{F} p] = \bot$



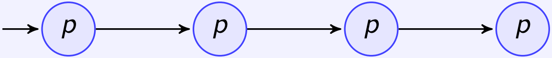
FLTL

Example

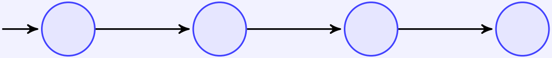
$[u \models_F \mathbf{X}p] = \top$



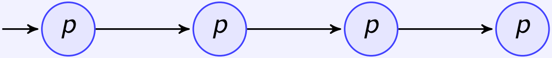
$[u \models_F p\,\mathbf{U}\,q] = \bot$



$[u \models_F \mathbf{F}p] = \bot$



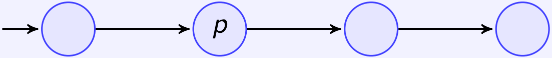
$[u \models_F \mathbf{G}p] = \top$



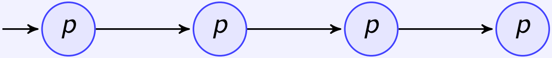
FLTL

Example

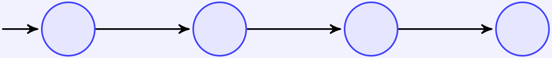
$[u \models_F \mathbf{X}p] = \top$



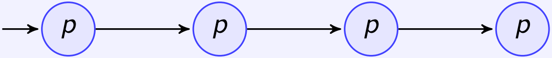
$[u \models_F p\,\mathbf{U}\,q] = \bot$



$[u \models_F \mathbf{F}p] = \bot$



$[u \models_F \mathbf{G}p] = \top$



FLTL Put into Perspective

FLTL evaluates a property for a finite word regardless of future executions.

Monitorability

An LTL formula φ is monitorable iff

$$\forall u \in \Sigma^*. \exists u' \in \Sigma^*. [uu' \models_F \varphi] \in \{\bot, \top\}$$

Example

Formula $\mathbf{GF}p$ is not monitorable.

Formual $a\, \mathbf{U}\, b$ is monitorable.

Presentation outline

Finite Semantics for HyperLTL

For a finite trace t, let $t[i, j]$ denote the subtrace of t from position i up to and including position j. And, $t[i, ..]$ denotes $t[i, |t| - 1]$

Trace assignment function is now $\Pi : \mathcal{V} \to \Sigma^*$.

We define:

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

Finite Semantics for HyperLTL

Finite Semantics for HyperLTL

$$\left[\Pi \models^{F}_{T} \mathbf{X}\, \varphi\right] = \begin{cases} [\Pi[1,..] \models^{F}_{T} \varphi] & \text{if} \quad \Pi[1,..] \neq \epsilon \\ \bot & \text{otherwise} \end{cases}$$

$$\left[\Pi \models^{F}_{T} \bar{\mathbf{X}}\, \varphi\right] = \begin{cases} [\Pi[1,..] \models^{F}_{T} \varphi] & \text{if} \quad \Pi[1,..] \neq \epsilon \\ \top & \text{otherwise} \end{cases}$$

$$\left[\Pi \models^{F}_{T} \varphi_1 \,\mathbf{U}\, \varphi_2\right] = \begin{cases} \top & \text{if} \quad \exists i \geq 0 : \Pi[i,..] \neq \epsilon \wedge [\Pi[i,..] \models^{F}_{T} \varphi_2] = \top \;\wedge \\ & \qquad \forall j \in [0,i) : [\Pi[j,..] \models^{F}_{T} \varphi_1] = \top \\ \bot & \text{otherwise} \end{cases}$$

Monitorability in HyperLTL

Trace Set Prefix Relation

Let U be a finite set of finite traces and V is a finite or infinite set of traces, then the prefix relation $U \leq V$ is defined as

$$U \leq V \;\equiv\; \forall u \in U.\, (\exists v \in V.\, u \leq v)$$

Note that V may contain traces that have no prefix in U.

Monitorability in HyperLTL

Trace Set Prefix Relation

Let U be a finite set of finite traces and V is a finite or infinite set of traces, then the prefix relation $U \leq V$ is defined as

$$U \leq V \;\equiv\; \forall u \in U.\, (\exists v \in V.\, u \leq v)$$

Note that V may contain traces that have no prefix in U.

A HyperLTL formula φ is monitorable iff

$$\forall M \in \mathcal{P}^*(\Sigma^*). \exists M' \in \mathcal{P}^*(\Sigma^*).\, [MM' \models \varphi] \in \{\bot, \top\}$$

RV of HyperLTL: Challenges

- Hyperproperties are more complex than traditional trace properties; i.e., we need to reason over multiple execution traces.

RV of HyperLTL: Challenges

- Hyperproperties are more complex than traditional trace properties; i.e., we need to reason over multiple execution traces.
- Monitoring an execution may depend on the evaluation of past and future executions.

RV of HyperLTL: Challenges

- Hyperproperties are more complex than traditional trace properties; i.e., we need to reason over multiple execution traces.
- Monitoring an execution may depend on the evaluation of past and future executions.

RV of HyperLTL: Challenges

- Hyperproperties are more complex than traditional trace properties; i.e., we need to reason over multiple execution traces.
- Monitoring an execution may depend on the evaluation of past and future executions.

Goal is to develop RV algorithms for alternation-free HyperLTL formulas.

(a subset of k-safety hyperproperties)

Motivating Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 . \; a_{\pi_1} \; \mathbf{U} \; b_{\pi_2}$$

$t_1 =$ a a a b

$t_2 =$ a a

Motivating Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1}\ \mathbf{U}\ b_{\pi_2}$$

$$t_1 = \text{a a a b}$$

$$t_2 = \text{a a}$$

- Traces π_1 and π_2 violate the formula (even t_2 does not individually satisfy ϕ.

Motivating Example 2

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1}\ \mathbf{U}\ b_{\pi_2}$$

$t_1 =$ a a a a a b

$t_2 =$ a a a a b b

$t_3 =$ a a b b b b

$t_4 =$ a a a b a a

Motivating Example 2

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1}\ \mathbf{U}\ b_{\pi_2}$$

$t_1 =$ a a a a a b

$t_2 =$ a a a a b b

$t_3 =$ a a b b b b

$t_4 =$ a a a b a a

- Traces t_1, t_2, t_3, t_4, individually satisfy the formula ϕ.

Motivating Example 2

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \ \mathbf{U} \ b_{\pi_2}$$

$t_1 =$ a a a a a b

$t_2 =$ a a a a b b

$t_3 =$ a a b b b b

$t_4 =$ a a a b a a

- Traces t_1, t_2, t_3, t_4, individually satisfy the formula ϕ.
- However, any combination of these traces violates the formula as satisfaction must happen at the same location in all traces.

We require the information about the index of satisfaction

Motivating Example 3

$$\phi = \forall \pi_1 . \forall \pi_2 . \ a_{\pi_1} \longrightarrow \mathbf{F}\, b_{\pi_2}$$

$t_1 =$ d c f

$t_2 =$ a e b

Motivating Example 3

$$\phi = \forall\pi_1.\forall\pi_2.\ a_{\pi_1} \longrightarrow \mathbf{F}\, b_{\pi_2}$$

$t_1 =$ d c f

$t_2 =$ a e b

- Traces t_1 and t_2, individually satisfy the formula ϕ.

Motivating Example 3

$$\phi = \forall\pi_1.\forall\pi_2.\ a_{\pi_1} \longrightarrow \mathbf{F}\, b_{\pi_2}$$

$t_1 =$ d c f

$t_2 =$ a e b

- Traces t_1 and t_2, individually satisfy the formula ϕ.
- However, t_1 and t_2 collectively violate the formula as b never happened in π_1

Progressing through the traces requires information from previous traces

Motivating Example 4

$$\phi = \forall \pi_1 \forall \pi_2 .\ \mathbf{G}(a_{\pi_1} \longrightarrow a_{\pi_2})$$

$t_1 =$ a b b a b

$t_2 =$ a e c a c

$t_3 =$ a a c a a

Motivating Example 4

$$\phi = \forall \pi_1 \forall \pi_2.\ \mathbf{G}(a_{\pi_1} \longrightarrow a_{\pi_2})$$

$t_1 =$ a b b a b

$t_2 =$ a e c a c

$t_3 =$ a a c a a

- Traces t_1 and t_2 satisfy the formula ϕ.

Motivating Example 4

$$\phi = \forall \pi_1 \forall \pi_2 .\ \mathbf{G}(a_{\pi_1} \longrightarrow a_{\pi_2})$$

$t_1 =$ a b b a b

$t_2 =$ a e c a c

$t_3 =$ a a c a a

- Traces t_1 and t_2 satisfy the formula ϕ.
- However, trace t_3 violates their agreement; i.e., satisfaction must happen at the same location in all traces

Trace quantification brings more expressiveness as compared to LTL

Monitoring HyperLTL: Idea

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1}\ \mathbf{U}\ b_{\pi_2}$$

$t_1 =$ a a a a a b

$$constraint = \mathbf{X}^{[0,4]} \neg b \wedge \mathbf{X}^5 b$$

$\mathbf{X}^{[i,k]}\psi$ represents $\mathbf{X}^i \psi \wedge \cdots \wedge \mathbf{X}^k \psi$

Monitoring HyperLTL: Idea

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \ \mathbf{U} \ b_{\pi_2}$$

$t_1 =$ a a a a a b

$$constraint = \mathbf{X}^{[0,4]} \neg b \wedge \mathbf{X}^5 b$$

$\mathbf{X}^{[i,k]} \psi$ represents $\mathbf{X}^i \psi \wedge \cdots \wedge \mathbf{X}^k \psi$

Incoming traces have to satisfy ϕ and these constraints

$t_2 =$ a a a a b b ~~constraints~~

$t_3 =$ a a b b b b ~~constraints~~

$t_4 =$ a a a b a a ~~constraints~~

Monitoring HyperLTL: Idea (contd.)

$$\phi = \forall \pi_1 . \forall \pi_2 . \forall \pi_3 . \; (a_{\pi_1} \; \mathbf{U} \; b_{\pi_2}) \; \mathbf{U} \; c_{\pi_3}$$

$t_1 =$ ab a ac ac a b

Monitoring HyperLTL: Idea (contd.)

$$\phi = \forall\pi_1.\forall\pi_2.\forall\pi_3.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_3}$$

$t_1 =$ ab a ac ac a b

$t_2 =$ a b ab a ac b

Monitoring HyperLTL: Idea (contd.)

$$\phi = \forall \pi_1 . \forall \pi_2 . \forall \pi_3 .\ (a_{\pi_1} \ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_3}$$

$t_1 =$ ab a ac ac a b

$t_2 =$ a b ab a ac b

$t_3 =$ b a ac a a ab

Monitoring HyperLTL: Idea (contd.)

$$\phi = \forall\pi_1.\forall\pi_2.\forall\pi_3.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_3}$$

$t_1 =$ ab | a | ac | ac | a | b

$t_2 =$ a | b | ab | a | ac | b

$t_3 =$ b | a | ac | a | a | ab

How to generate constraints in this case?

Semantics of U

$$\Pi \models_{\mathtt{T}} \phi_1\ \mathbf{U}\ \phi_2 \quad \text{iff} \quad \exists \mathtt{i} \geq 0.(\Pi[\mathtt{i},\infty] \models_{\mathtt{T}} \phi_2 \land \forall \mathtt{j}.0 \leq \mathtt{j} < \mathtt{i}.\Pi[\mathtt{j},\infty] \models_{\mathtt{T}} \phi_1)$$

- In this formula, the propositions of interest are b and c.
- The first satisfaction of c should be at the same index in each trace.
- Each satisfaction of b should be agreed upon in all traces.

Proposed Monitoring Approach

- Rewriting-based monitoring for HyperLTL
  - Using rewriting for the evaluation and progression of formula with respect to incoming events
  - Generating constraints to ensure the agreement among traces

- Hyper-Monitors for HyperLTL
  - Using LTL_4 (or RV-LTL) with counters to monitors the progress of each trace
  - Using hyper-monitors to find the violation at a given time instant

Proposed Monitoring Approach

- Rewriting-based monitoring for HyperLTL
  - Using rewriting for the evaluation and progression of formula with respect to incoming events
  - Generating constraints to ensure the agreement among traces
- Hyper-Monitors for HyperLTL
  - Using LTL_4 (or RV-LTL) with counters to monitors the progress of each trace
  - Using hyper-monitors to find the violation at a given time instant

Both approaches require to identify a set of variables requiring counting in a given HyperLTL formula (ϕ)

Proposed Monitoring Approach

- Rewriting-based monitoring for HyperLTL
  - Using rewriting for the evaluation and progression of formula with respect to incoming events
  - Generating constraints to ensure the agreement among traces
- Hyper-Monitors for HyperLTL
  - Using LTL_4 (or RV-LTL) with counters to monitors the progress of each trace
  - Using hyper-monitors to find the violation at a given time instant

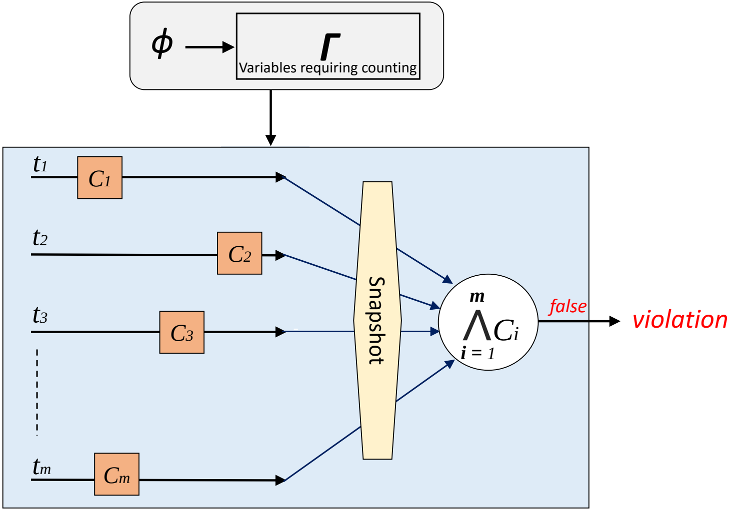
Both approaches require to identify a set of variables requiring counting in a given HyperLTL formula (ϕ)

$$\phi = \forall \pi_1 . \forall \pi_2 . \forall \pi_3 . \ (a_{\pi_1} \ \mathbf{U} \ b_{\pi_2}) \ \mathbf{U} \ c_{\pi_3}$$

Presentation outline

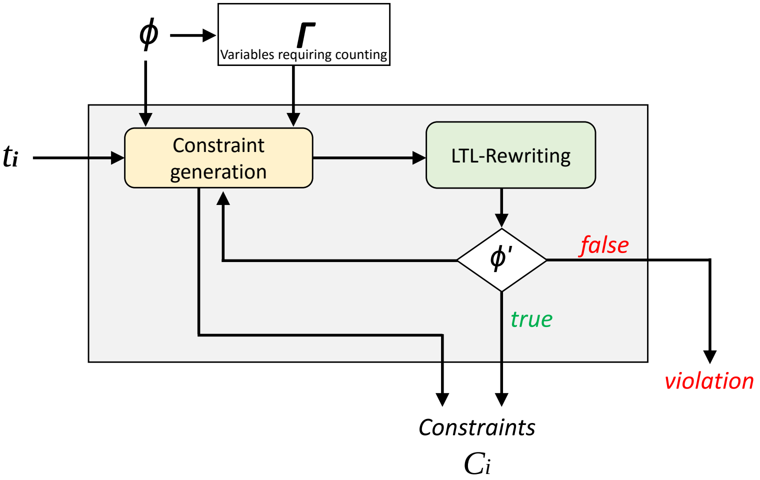
Proposed Monitoring Algorithms: An Overview

Rewriting-based Monitoring for HyperLTL



Proposed Monitoring Algorithms: An Overview

Rewriting-based Monitoring for HyperLTL



Outline

Algorithm to Find the set of Propositions of Interest (Γ)

Input: Syntax tree of the HyperLTL formula (ft)
Output: Set Γ
function $\Gamma(ft)$
$node := root(ft)$
$V := \{\}$ $\triangleright$*tracks record of trace quantifiers under the scope of "**U**"*
If $\neg(distinctQuantifiers(all.leaves))$ return $(\{\})$
match ($node$) **with**

Algorithm to Find the set of Propositions of Interest (Γ)

$|\ (\vee)$:

match $(\mathit{left}.\mathit{node})$ $(\mathit{right}.\mathit{node})$ **with**

$|\quad \mathit{leaf}_1 \qquad \mathit{leaf}_2\ :$

if $(\mathit{leaf}_1.\pi \neq \mathit{leaf}_2.\pi)$

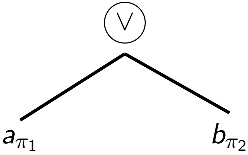
return $(\{\mathit{leaf}_1 \vee \mathit{leaf}_2\})$

else

return $(\{\})$

end if

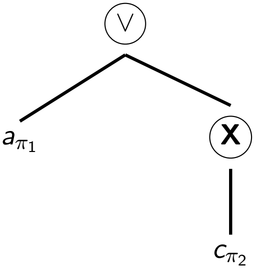
$\Gamma(a_{\pi_1} \vee b_{\pi_2}) = \{a \vee b\}$



Algorithm to Find the set of Propositions of Interest (Γ)

| ($\vee$):
match (*left.node*) (*right.node*) **with**
$\quad | \quad leaf_1 \quad - \quad : \text{return } (\{leaf_1\} \cup \Gamma(right.node))$

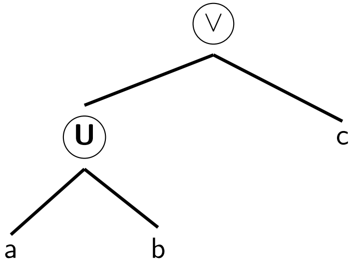
$\Gamma(a_{\pi_1} \vee \mathbf{X} c_{\pi_2}) = \{a\} \cup \Gamma(\mathbf{X} c_{\pi_2})$



Algorithm to Find the set of Propositions of Interest (Γ)

| ($\vee$):
  match (*left.node*) (*right.node*) **with**
    | $-$ *leaf*$_2$: return $(\Gamma(left.node) \cup \{leaf_2\})$

$\Gamma((a_{\pi_1}\ \mathbf{U}\ b_{\pi_2}) \vee c_{\pi_3}) =$
$\Gamma(a_{\pi_1}\ \mathbf{U}\ b_{\pi_2}) \cup \{c\}$



Algorithm to Find the set of Propositions of Interest (Γ)

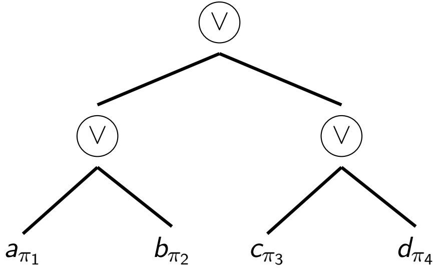
$\mid (\vee)$:

match $(left.node)$ $(right.node)$ **with**

$\mid \ \{\vee, \neg\} \quad \{\vee, \neg\}$: return $(\Gamma(left.node) \veebar \Gamma(right.node))$

$\veebar$ represents logical "OR" operation among the elements of two sets

$$\Gamma((a_{\pi_1} \vee b_{\pi_2}) \vee (c_{\pi_3} \vee d_{\pi_4})) =$$
$$\Gamma(a_{\pi_1} \vee b_{\pi_2}) \veebar \Gamma(c_{\pi_3} \vee d_{\pi_4}) =$$
$$\{a \vee b\} \veebar \{c \vee d\} =$$
$$\{a \vee b \vee c \vee d\}$$



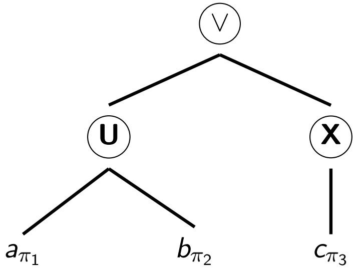
Algorithm to Find the set of Propositions of Interest (Γ)

| ($\vee$):

match (*left.node*) (*right.node*) **with**

| $-$ $-$: return ($\Gamma(\mathit{left.node}) \cup \Gamma(\mathit{right.node})$)

$\Gamma((a_{\pi_1}\ \mathbf{U}\ b_{\pi_2}) \vee \mathbf{X} c_{\pi_3}) =$
$\Gamma(a_{\pi_1}\ \mathbf{U}\ b_{\pi_2}) \cup \Gamma(\mathbf{X} c_{\pi_3})$



Algorithm to Find the set of Propositions of Interest (Γ)

| (**U**) :

match (*left.node*) (*right.node*) **with**

| $leaf_1$ $leaf_2$:

if $(V - \{leaf_2.\pi\} \neq \emptyset \vee leaf_1.\pi \neq leaf_2.\pi)$

return $(\{leaf_2\})$

else

return $(\{\})$

end if

Algorithm to Find the set of Propositions of Interest (Γ)

$|$ ($\mathbf{U}$) :

match ($left.node$) ($right.node$) **with**

$|$ $leaf_1$ $leaf_2$:

if $(V - \{leaf_2.\pi\} \neq \emptyset \vee leaf_1.\pi \neq leaf_2.\pi)$

return $(\{leaf_2\})$

else

return $(\{\})$

end if

$|$ $-$ $leaf_2$:

if $(\nexists x \in (left.node).x \in \{\mathbf{X}, \mathbf{U}\})$

return $(\# \odot \{leaf_2\})$

else

return $(\Gamma(left.node) \cup \# \odot \{leaf_2\})$

end if

$\#$ indicates that the corresponding variables need to be counted only once

$\odot$ represents application of unary operators (e.g., $\neg, \mathbf{X}$) to the elements of a set

Algorithm to Find the set of Propositions of Interest (Γ)

| (**U**) :
  match $(\mathit{left.node})\ (\mathit{right.node})$ **with**
    | leaf_1 $-$:
      $V \leftarrow V \cup \{\mathit{leaf}_1.\pi\}$;
      return $(\Gamma(\mathit{right.node}))$

Algorithm to Find the set of Propositions of Interest (Γ)

$|\ (\mathbf{U})$:

match $(left.node)\ (right.node)$ **with**

$|\quad leaf_1 \qquad - :$

$V \leftarrow V \cup \{leaf_1.\pi\}$;

return $(\Gamma(right.node))$

$|\quad - \qquad - :$

if $(\nexists x \in (left.node).x \in \{\mathbf{X}, \mathbf{U}\})$

return $(\# \odot \Gamma(right.node))$

else

return $(\Gamma(left.node) \cup \# \odot \Gamma(right.node))$

end if

Algorithm to Find the set of Propositions of Interest (Γ)

match (*node*) **with**

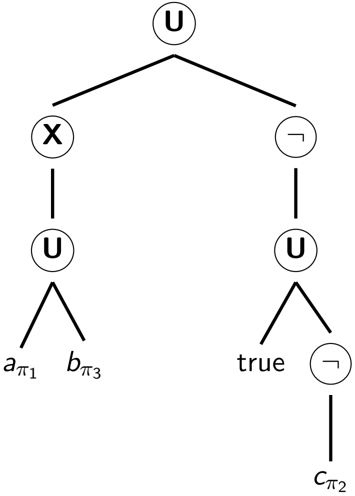
| (*leaf*) : return ($\{leaf\}$)

| ($\neg$) : return ($\neg \odot \Gamma(child.node)$)

| ($\mathbf{X}$) : return ($\mathbf{X} \odot \Gamma(child.node)$)

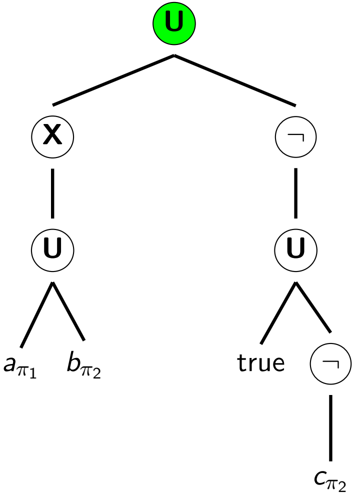
Algorithm to Find Γ: Running Example

$$\phi = \forall\pi_1.\forall\pi_2.\forall\pi_3 \mathbf{X}(a_{\pi_1} \,\mathbf{U}\, b_{\pi_3})\mathbf{U}\; \overbrace{\neg(\mathit{true}\,\mathbf{U}\,\neg c_{\pi_2})}^{\mathbf{G}c}$$



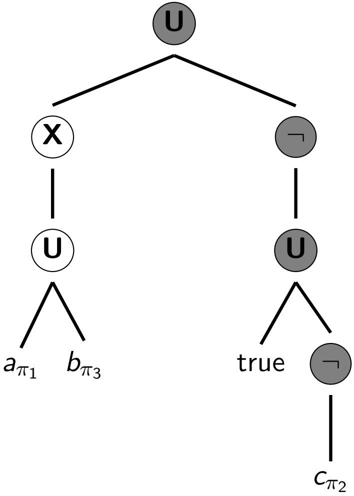
Algorithm to Find Γ: Running Example

$$\Gamma(\mathbf{X}(a_{\pi_1}\ \mathbf{U}\ b_{\pi_3})) \cup \Gamma(\neg(\mathit{true}\mathbf{U}\neg c_{\pi_2}))$$



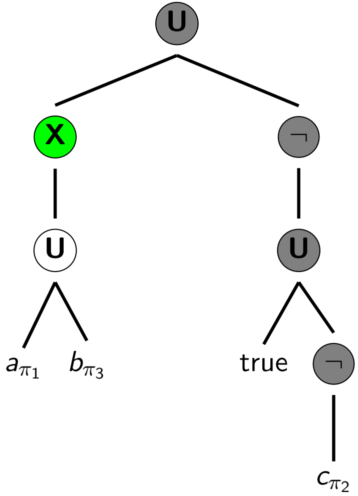
Algorithm to Find Γ: Running Example

$\Gamma(\mathbf{X}(a_{\pi_1} \ \mathbf{U} \ b_{\pi_3}))$



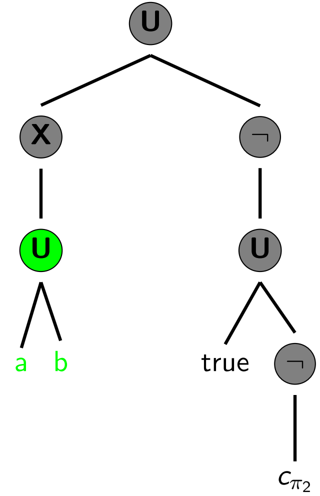
Algorithm to Find Γ: Running Example

$\mathbf{X} \odot \Gamma((a_{\pi_1} \ \mathbf{U} \ b_{\pi_3}))$



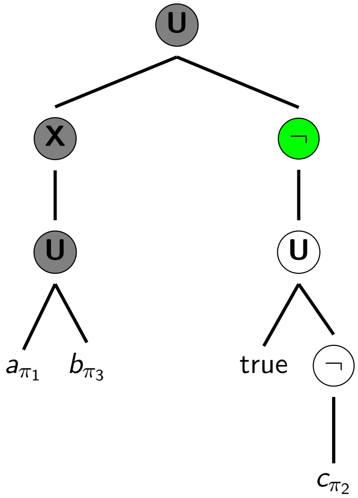
Algorithm to Find Γ: Running Example

$\mathbf{X} \odot \{b\} = \{\mathbf{X} b\}$



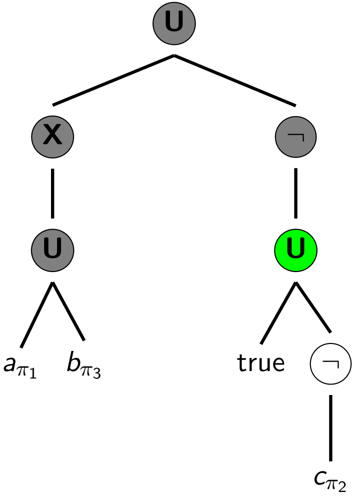
Algorithm to Find Γ: Running Example

$\neg \odot \Gamma((true\mathbf{U}\neg c_{\pi_2}))$



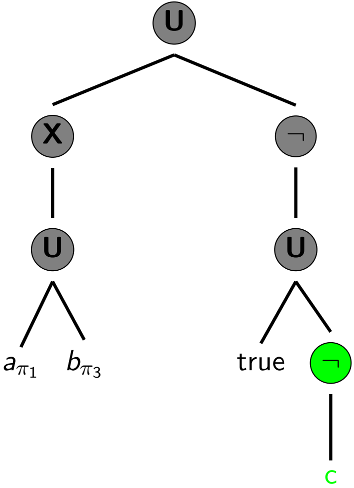
Algorithm to Find Γ: Running Example

$\neg \odot \Gamma(\neg c_{\pi_2})$



Algorithm to Find Γ: Running Example

$\neg \odot \{\neg c_{\pi_2}\} = \{c\}$

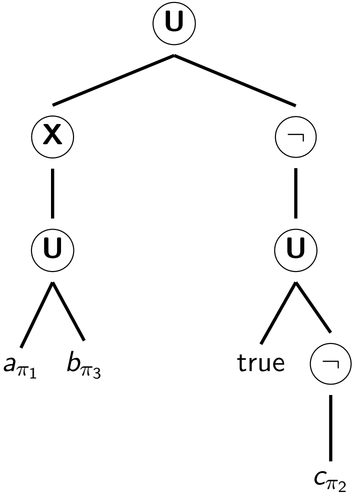


Algorithm to Find Γ: Running Example

$$\phi = \forall\pi_1.\forall\pi_2.\forall\pi_3\mathbf{X}(a_{\pi_1}\ \mathbf{U}\ b_{\pi_3})\mathbf{U}\ \overbrace{\neg(\mathit{true}\ \mathbf{U}\ \neg c_{\pi_2})}^{\mathbf{G}c}$$

$$\Gamma(\mathbf{X}(a_{\pi_1}\ \mathbf{U}\ b_{\pi_3})) \cup \Gamma(\neg(\mathit{true}\mathbf{U}\neg c_{\pi_2}))$$

$$\{\mathbf{X}b\} \cup \{c\} = \{\mathbf{X}b, c\}$$



Outline

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

Input: LTL formula ϕ, Event e
Output: Formula ψ

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

match (ϕ) **with**
$|\ (\phi_1 \vee \phi_2)$:
return $(REWRITE(\phi_1, e) \vee REWRITE(\phi_2, e))$

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

match (ϕ) **with**
$|\ (\phi_1 \vee \phi_2)$:
return $(REWRITE(\phi_1, e) \vee REWRITE(\phi_2, e))$

$\phi = (a \vee b)$ In-coming event $= a$

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

match (ϕ) **with**
$|\ (\phi_1 \vee \phi_2)$:
return $(REWRITE(\phi_1, e) \vee REWRITE(\phi_2, e))$

$\phi = (a \vee b)$ In-coming event $= a$
$\implies$ return $(REWRITE(a, a) \vee REWRITE(b, a))$

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

match (ϕ) **with**
$| \ (\phi_1 \vee \phi_2)$:
return $(REWRITE(\phi_1, e) \vee REWRITE(\phi_2, e))$

$\phi = (a \vee b)$ In-coming event $= a$
$\implies$ return $(REWRITE(a, a) \vee REWRITE(b, a))$

Further steps will return the satisfaction by finding $(a \vDash a) \vee (b \vDash a)$

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$|\ (\phi_1 \ \mathbf{U}\ \phi_2)$:
if(last_event(e)) **then**
  return (REWRITE(ϕ_2, e))
else
  return (REWRITE(ϕ_2, e) $\vee$ *REWRITE*(ϕ_1, e) $\wedge$ ($\phi_1 \ \mathbf{U}\ \phi_2$)))

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$\mid (\phi_1 \ \mathbf{U} \ \phi_2)$:
if(last_event(e)) **then**
  return (REWRITE(ϕ_2, e))
else
  return (REWRITE($\phi_2, e) \vee REWRITE(\phi_1, e) \wedge (\phi_1 \ \mathbf{U} \ \phi_2)$))

$\phi = (a \ \mathbf{U} \ b)$ In-coming event $= a$

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$| \ (\phi_1 \ \mathbf{U} \ \phi_2)$:
if(last_event(e)) **then**
return (REWRITE(ϕ_2, e))
else
return (REWRITE(ϕ_2, e) $\vee$ $\mathit{REWRITE}(\phi_1, e) \wedge (\phi_1 \ \mathbf{U} \ \phi_2)$))

$\phi = (a \ \mathbf{U} \ b)$ In-coming event $= a$
$\implies$ return $(\mathit{REWRITE}(b, a) \vee (\mathit{REWRITE}(a, a) \wedge (a \ \mathbf{U} \ b)))$

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$\mid (\phi_1 \ \mathbf{U} \ \phi_2)$:
if(last_event(e)) **then**
  return (REWRITE(ϕ_2, e))
else
  return (REWRITE(ϕ_2, e) $\vee$ $REWRITE(\phi_1, e) \wedge (\phi_1 \ \mathbf{U} \ \phi_2)$))

$\phi = (a \ \mathbf{U} \ b)$ In-coming event $= a$
$\implies$ return $(\mathit{False} \vee (\mathit{True} \wedge (a \ \mathbf{U} \ b)))$

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$| \ (\phi_1 \ \mathbf{U} \ \phi_2)$:
if(last_event(e)) **then**
  return (REWRITE(ϕ_2, e))
else
  return (REWRITE($\phi_2, e) \vee \mathit{REWRITE}(\phi_1, e) \wedge (\phi_1 \ \mathbf{U} \ \phi_2)$))

$\phi = (a \ \mathbf{U} \ b)$ In-coming event $= a$
$\implies$ return $(\mathit{False} \vee (\mathit{True} \wedge (a \ \mathbf{U} \ b)))$

Formula has not yet been satisfied/violated

$a \ \mathbf{U} \ b$ will be used again to find the satisfaction/violation

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$|$ $(\mathbf{X}\phi)$:
if(last_event(e)) **then**
  return (*False*)
else
  return REWRITE(ϕ, e)

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$\mid (\mathbf{X}\phi)$:
if(last_event(e)) **then**
  return ($False$)
else
  return REWRITE(ϕ, e)

$\phi = (\mathbf{X}b)$ In-coming event $= a$

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$\mid (\mathbf{X}\phi)$:
if(last_event(e)) **then**
    return (*False*)
else
    return REWRITE(ϕ, e)

$\phi = (\mathbf{X} b)$ In-coming event $= a$
$\Longrightarrow$ return (REWRITE(b, e))

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$\mid$ $(\mathbf{X}\phi)$:
if(last_event(e)) **then**
  return ($False$)
else
  return REWRITE(ϕ, e)

$\phi = (\mathbf{X}b)$ In-coming event $= a$
$\implies$ return (REWRITE(b, e))

The next in-coming event will have to satisfy b

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$| \; (\mathbf{X}\phi)$:
if(last_event(e)) **then**
    return (*False*)
else
    return REWRITE(ϕ, e)

$\phi = (\mathbf{X}b)$ In-coming event $= a$
$\Longrightarrow$ return (REWRITE(b, e))

The next in-coming event will have to satisfy b

If a was last event, then violation found.

Implementation of "**strong next**"

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

$\mid (\phi[e])$:
if$(e \models \phi)$ **then**
    return $(True)$
elseif$(e \not\models \phi)$
    return$(False)$
$\mid (True)$:   return $(True)$
$\mid (False)$:   return $(False)$

Rewriting Algorithm [Havelund, Rosu, 2001]

REWRITE (ϕ, e)

match (ϕ) **with**
| $(\phi_1 \vee \phi_2)$:
  return $(REWRITE(\phi_1, e) \vee REWRITE(\phi_2, e))$
| $(\phi_1 \ \mathbf{U} \ \phi_2)$:
if(last_event(e)) **then**
  return (REWRITE(ϕ_2, e))
else
  return (REWRITE$(\phi_2, e) \vee REWRITE(\phi_1, e) \wedge (\phi_1 \ \mathbf{U} \ \phi_2)$))
| $(\mathbf{X}\phi)$:
if(last_event(e)) **then**
  return $(False)$
else
  return REWRITE(ϕ, e)
| $(\phi[e])$:
if$(e \models \phi)$ **then**
  return $(True)$
elseif$(e \not\models \phi)$
  return$(False)$
| $(True)$: return $(True)$
| $(False)$: return $(False)$

Rewriting Algorithm - Example

$$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = \epsilon}$$

Rewriting Algorithm - Example

$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = a}$ – Event comes in

Rewriting Algorithm - Example

$$\boxed{\phi = a \; \mathbf{U} \; b \quad \pi = a}$$

$$\phi \leftarrow \text{REWRITE}(\phi, a)$$

Rewriting Algorithm - Example

$$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = a}$$

$$\phi \leftarrow \text{REWRITE}(\phi, a)$$
$$\implies \text{ return } (REWRITE(b, a) \vee (REWRITE(a, a) \wedge (a \ \mathbf{U} \ b)))$$

Rewriting Algorithm - Example

$$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = a}$$

$$\phi \leftarrow \text{REWRITE}(\phi, a)$$

$$\implies \text{return } (\mathit{False} \vee (\mathit{True} \wedge (a \ \mathbf{U} \ b)))$$

Rewriting Algorithm - Example

$$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = a}$$

$$\phi \leftarrow \text{REWRITE}(\phi, a)$$
$$\phi \leftarrow (a \ \mathbf{U} \ b)$$

Rewriting Algorithm - Example

$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = aa}$ – Event comes in

Rewriting Algorithm - Example

$$\boxed{\phi = a\ \mathbf{U}\ b \quad \pi = aa}$$

$$\phi \leftarrow \text{REWRITE}(\phi, a)$$

Rewriting Algorithm - Example

$$\boxed{\phi = a \; \mathbf{U} \; b \quad \pi = aa}$$

$$\phi \leftarrow \text{REWRITE}(\phi, a)$$

$$\Longrightarrow \;\; \text{return } (REWRITE(b, a) \vee (REWRITE(a, a) \wedge (a \; \mathbf{U} \; b)))$$

Rewriting Algorithm - Example

$$\boxed{\phi = a\ \mathbf{U}\ b \quad \pi = a\color{red}{a}}$$

$$\phi \leftarrow \text{REWRITE}(\phi, a)$$

$$\Longrightarrow \ \text{return } (\color{red}{False} \vee (\color{red}{True} \wedge (a\ \mathbf{U}\ b)))$$

Rewriting Algorithm - Example

$$\boxed{\phi = a \; \mathbf{U} \; b \quad \pi = a{\color{red}a}}$$

$$\phi \leftarrow \text{REWRITE}(\phi, a)$$
$$\phi \leftarrow (a \; \mathbf{U} \; b)$$

Rewriting Algorithm - Example

$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = aab}$ – Event comes in

Rewriting Algorithm - Example

$$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = aab}$$

$$\phi \leftarrow \text{REWRITE}(\phi, b)$$

Rewriting Algorithm - Example

$$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = aab}$$

$$\phi \leftarrow \text{REWRITE}(\phi, b)$$

$$\implies \text{return } (REWRITE(b, b) \vee (REWRITE(a, b) \wedge (a \ \mathbf{U} \ b)))$$

Rewriting Algorithm - Example

$$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = aab}$$

$$\phi \leftarrow \text{REWRITE}(\phi, b)$$

$$\implies \text{return } (\mathit{True} \vee (\mathit{False} \wedge a \ \mathbf{U} \ b))$$

Rewriting Algorithm - Example

$$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = aab}$$

$$\phi \leftarrow \text{REWRITE}(\phi, b)$$
$$\phi \leftarrow True$$

Rewriting Algorithm - Example

$$\boxed{\phi = a \ \mathbf{U} \ b \quad \pi = aa\textcolor{red}{b}}$$

$$\phi \leftarrow \text{REWRITE}(\phi, b)$$

$$\phi \leftarrow \mathit{True}$$

Formula is satisfied!

Rewriting-based Monitoring for HyperLTL: Challenges

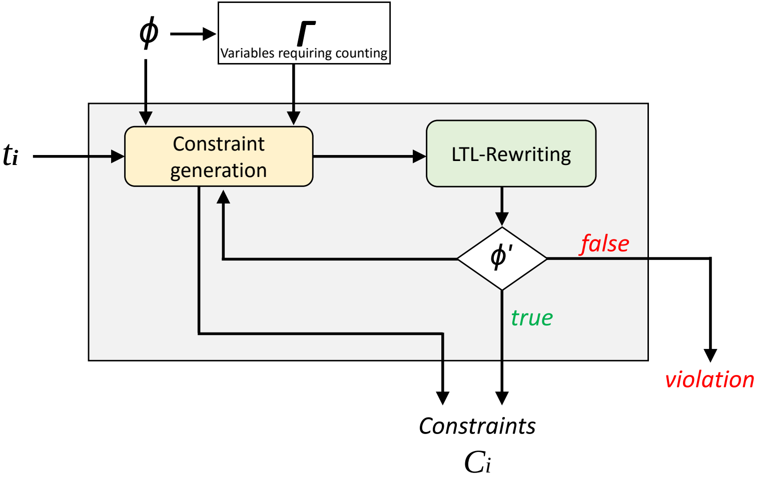
- Progressing through the traces requires information from previous traces
  $\phi = \forall\pi_1.\forall\pi_2.\ a_{\pi_1} \longrightarrow \mathbf{F}\, b_{\pi_2}$
- Tracking the satisfaction of sub-formulas is required
- Constraints generation is dynamic depending upon the agreement amongst traces

Rewriting-based Monitoring: Proposed Approach

- Find the set of variables which require counting (Γ)
- Use rewriting to check the satisfaction and progress of the formula
- Generate the constraints (required for trace agreement) for each incoming event

These steps are performed for all incoming traces

Rewriting-based Monitoring for HyperLTL



Rewriting-based Monitoring

Input: HyperLTL formula ϕ, Γ, set of incoming traces M
Output: $\lambda = \{\bot, ?\}$

Rewriting-based Monitoring

Input: HyperLTL formula ϕ, Γ, set of incoming traces M
Output: $\lambda = \{\bot, ?\}$

```
while (1) do
    for each m ∈ M do
        C_m ← ConstraintsHandler (φ, Γ)        ▷ Generate constraints
```

Rewriting-based Monitoring

Input: HyperLTL formula ϕ, Γ, set of incoming traces M
Output: $\lambda = \{\bot, ?\}$

while (1) **do**
  for each $m \in M$ **do**
    $C_m \leftarrow$ `ConstraintsHandler` (ϕ, Γ) ▷ *Generate constraints*
Take a snapshot for counters $\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$ at time instant
$\beta = \mathtt{SAT}(\bigwedge_{m=1}^{M}(C_m))$ ▷ *Check the satisfiability of constraints*
if $(\beta = \mathit{false})$ **then**
  $\lambda := \bot$
return (λ)

Rewriting-based Monitoring

```
ConstraintsHandler (\phi, \Gamma)
if (\phi = \phi_1 \vee \phi_2) then
    \psi_1 \leftarrow ConstraintsHandler (\phi_1, \Gamma(\phi_1))
    \psi_2 \leftarrow ConstraintsHandler (\phi_2, \Gamma(\phi_2))
    if (\psi_1 = False \wedge \psi_2 = False) then
        return (False)
    else if (\psi_1 = False) then
        return (\psi_2)
    else if (\psi_2 = False) then
        return (\psi_1)
        else
            return (\psi_1 \wedge \psi_2)
else
    if (\phi := \phi_1 U \phi_2 \wedge ((\phi_1 := \phi_L \vee \phi_R) \wedge
    distinctQuantifiers(\phi_L, \phi_R) )) then
        \psi_1 \leftarrow ConstraintsHandler (\phi_L U \phi_2, \Gamma)
        \psi_2 \leftarrow ConstraintsHandler (\phi_R U \phi_2, \Gamma)
        if \psi_1 = False \wedge \psi_2 = False then
            return (False)
        else if \psi_1 = False then
            return ((\phi_R U \phi_2) \wedge \psi_2)
        else if \psi_2 = False then
            return ((\phi_L U \phi_2) \wedge \psi_1)
        else
            return (((\phi_R U \phi_2) \vee (\phi_R U \phi_2)) \wedge \psi_1)
    else
        r \leftarrow ConstraintsTraces(\Gamma,\phi)
        if (r = False) then
            return False
        else
            return r
```

Rewriting-based Monitoring

ConstraintsHandler (ϕ, Γ)
if $(\phi = \phi_1 \vee \phi_2)$ **then**
  $\psi_1 \leftarrow$ ConstraintsHandler $(\phi_1, \Gamma(\phi_1))$
  $\psi_2 \leftarrow$ ConstraintsHandler $(\phi_2, \Gamma(\phi_2))$
  if $(\psi_1 = \mathit{False} \wedge \psi_2 = \mathit{False})$ **then**
    return (False) ▷ *Return False if both formulas are False*
  else if $(\psi_1 = \mathit{False})$ **then**
    return (ψ_2)
  else if $(\psi_2 = \mathit{False})$ **then**
    return (ψ_1)
  else
    return $(\psi_1 \wedge \psi_2)$

- Divide ϕ into subformulas
- Seperate variables from Γ

- Constraints refinement

Rewriting-based Monitoring

ConstraintsHandler (ϕ, Γ)
else
 if $(\phi := \phi_1 \ \mathbf{U}\ \phi_2 \wedge ((\phi_1 := \phi_L \vee \phi_R) \wedge$ distinctQuantifiers(ϕ_L, ϕ_R) $))$ **then**
  $\psi_1 \leftarrow$ ConstraintsHandler $(\phi_L\ \mathbf{U}\ \phi_2, \Gamma)$
  $\psi_2 \leftarrow$ ConstraintsHandler $(\phi_R\ \mathbf{U}\ \phi_2, \Gamma)$
  if $(\psi_1 = False \wedge \psi_2 = False)$ **then**
   return $(False)$
  else if $(\psi_1 = False)$ **then**
   return $((\phi_R\ \mathbf{U}\ \phi_2) \wedge \psi_2)$
  else if $(\psi_2 = False)$ **then**
   return $((\phi_L\ \mathbf{U}\ \phi_2) \wedge \psi_1)$
  else
   return $(((\phi_L\ \mathbf{U}\ \phi_2) \vee (\phi_R\ \mathbf{U}\ \phi_2)) \wedge \psi_1)$
 else
  $r \leftarrow$ ConstraintsTraces(Γ, ϕ)
  if $(r = False)$ **then**
   return $False$
  else
   return r

- Left side of Until is disjunction with different quantifiers
- Divide to find which side of disjunction finds satisfaction

- Constraints are refined according to the satisfaction of disjunction

- Generate Constraint for each trace

Rewriting-based Monitoring - Algorithm for Constraints

ConstraintsTraces(Γ: *Set of variables require counting,* ϕ)
$\Gamma' \leftarrow \Gamma$
for *each* $a \in \Gamma$ **do**
    $i_a \leftarrow 0$
$i \leftarrow 0$
$r \leftarrow True$
$\phi' \leftarrow$ quantifier-elimination(ϕ)   ▷*Eliminate trace quantifiers*
while getEvent *(e)* **do**
    $\phi' \leftarrow$ REWRITE (e, ϕ')
    if $(\phi' = False)$ **then**
        return ϕ'
    for *(each* $a \in \Gamma$ *s.t.* $e \models a$*)* **do**
        if $(a = a'_{\#})$ **then**
            $\Gamma \leftarrow \Gamma \setminus a$
        $r \leftarrow r \wedge \mathbf{X}^i a$
        if $a \in \Gamma'$ **then**
            $\Gamma' \leftarrow \Gamma' \setminus a$
        if $(i > 0) \wedge (i_a \neq i)$ **then**
            $r \leftarrow r \wedge \mathbf{X}^{[i_a, i-1]} \neg a$
        $i_a \leftarrow i + 1$
    if $(\phi' = True)$ **then**
        Break
    for *(each* $a \in \Gamma$ *s.t.* $a \ = \mathbf{X}a'$*)* **do**
        $\Gamma \leftarrow (\Gamma \setminus \{a\}) \cup \{a'\}$
        $\Gamma' \leftarrow (\Gamma' \setminus \{a\}) \cup \{a'\}$
        $i_a{+}{+}$
    $i{+}{+}$
for *each* $b \in \Gamma'$ **do**
    $r \leftarrow r \wedge \mathbf{G} \neg b$
return r

Rewriting-based Monitoring - Algorithm for Constraints

ConstraintsTraces(Γ: *Set of variables require counting*, ϕ)
$\Gamma' \leftarrow \Gamma$
for *each* $a \in \Gamma$ **do**
  $i_a \leftarrow 0$
$i \leftarrow 0$
$r \leftarrow True$

- Set counter and state variables here
- i is the main counter for incoming events
- r represents constraints

Rewriting-based Monitoring - Algorithm for Constraints

ConstraintsTraces(Γ: *Set of variables require counting*, ϕ)
$\Gamma' \leftarrow \Gamma$
for *each* $a \in \Gamma$ **do**
  $i_a \leftarrow 0$
$i \leftarrow 0$
$r \leftarrow True$
$\phi' \leftarrow$ quantifier-elimination(ϕ)   ▷*Eliminate trace quantifiers*
while getEvent *(e)* **do**
  $\phi' \leftarrow$ REWRITE (e, ϕ')
  if $(\phi' = False)$
  then
    return ϕ'

- Set counter and state variables here
- i is the main counter for incoming events
- r represents constraints

- *Events of a trace come in*
- *Use rewriting for formula evaluation and progression*
- *In case of violation, False is returned*

Rewriting-based Monitoring - Algorithm for Constraints

ConstraintsTraces(Γ: *Set of variables require counting*, ϕ)

$\Gamma' \leftarrow \Gamma$
for *each* $a \in \Gamma$ **do**
  $i_a \leftarrow 0$
$i \leftarrow 0$
$r \leftarrow$ *True*

- Set counter and state variables here
- i is the main counter for incoming events
- r represents constraints

$\phi' \leftarrow$ quantifier-elimination(ϕ) ▷*Eliminate trace quantifiers*

while getEvent *(e)* **do**
  $\phi' \leftarrow$ REWRITE (e, ϕ')
  if $(\phi' = False)$
  then
    return ϕ'

- *Events of a trace come in*
- *Use rewriting for formula evaluation and progression*
- *In case of violation, False is returned*

  for *(each* $a \in \Gamma$ *s.t.* $e \vDash a$*)* **do**
    if $(a = a'_{\#})$ **then**
      $\Gamma \leftarrow \Gamma \setminus a$
    $r \leftarrow r \wedge \mathbf{X}^i a$
    if $a \in \Gamma'$ **then**
      $\Gamma' \leftarrow \Gamma' \setminus a$
    if $(i > 0) \wedge (i_a \neq i)$ **then**
      $r \leftarrow r \wedge \mathbf{X}^{[i_a, i-1]} \neg a$
    $i_a \leftarrow i + 1$

-Check if incoming event needs to be counted once
- Delete such variable from Γ
- Generate the constraint ensuring its occurance at location i
- Generate the constraints ensuring its absence from other lolcations
- Update the event counter

Rewriting-based Monitoring - Algorithm for Constraints

`ConstraintsTraces`(Γ: *Set of variables require counting*, ϕ)
$\Gamma' \leftarrow \Gamma$
for *each* $a \in \Gamma$ **do**
    $i_a \leftarrow 0$
$i \leftarrow 0$
$r \leftarrow$ *True*

- Set counter and state variables here
- i is the main counter for incoming events
- r represents constraints

$\phi' \leftarrow$ `quantifier-elimination`(ϕ)   ▷*Eliminate trace quantifiers*

while `getEvent` *(e)* **do**
    $\phi' \leftarrow$ `REWRITE` (e, ϕ')
    if $(\phi' = \mathit{False})$
    then
        return ϕ'

- *Events of a trace come in*
- *Use rewriting for formula evaluation and progression*
- *In case of violation, False is returned*

    for *(each* $a \in \Gamma$ *s.t.* $e \models a$*)* **do**
        if $(a = a'_{\#})$ **then**
            $\Gamma \leftarrow \Gamma \setminus a$
        $r \leftarrow r \wedge \mathbf{X}^{i} a$
        if $a \in \Gamma'$ **then**
            $\Gamma' \leftarrow \Gamma' \setminus a$
        if $(i > 0) \wedge (i_a \neq i)$ **then**
            $r \leftarrow r \wedge \mathbf{X}^{[i_a, i-1]} \neg a$
        $i_a \leftarrow i + 1$

-Check if incoming event needs to be counted once
- Delete such variable from Γ
- Generate the constraint ensuring its occurance at location i
- Generate the constraints ensuring its absence from other lolcations
- Update the event counter

    if $(\phi' = \mathit{True})$ **then**
        `Break`

If formula is true then stop generating constraints

Rewriting-based Monitoring - Algorithm for Constraints

ConstraintsTraces(Γ: *Set of variables require counting*, ϕ)
$\Gamma' \leftarrow \Gamma$
for *each* $a \in \Gamma$ **do**
  $i_a \leftarrow 0$
$i \leftarrow 0$
$r \leftarrow True$

- Set counter and state variables here
- i is the main counter for incoming events
- r represents constraints

$\phi' \leftarrow$ quantifier-elimination(ϕ) ▷*Eliminate trace quantifiers*
while getEvent (e) **do**
  $\phi' \leftarrow$ REWRITE (e, ϕ')
  if $(\phi' = False)$
  then
    return ϕ'

- *Events of a trace come in*
- *Use rewriting for formula evaluation and progression*
- *In case of violation, False is returned*

  for (*each* $a \in \Gamma$ *s.t.* $e \models a$) **do**
    if $(a = a'_{\#})$ **then**
      $\Gamma \leftarrow \Gamma \setminus a$
    $r \leftarrow r \wedge \mathbf{X}^i a$
    if $a \in \Gamma'$ **then**
      $\Gamma' \leftarrow \Gamma' \setminus a$
    if $(i > 0) \wedge (i_a \neq i)$ **then**
      $r \leftarrow r \wedge \mathbf{X}^{[i_a, i-1]} \neg a$
    $i_a \leftarrow i + 1$

-Check if incoming event needs to be counted once
- Delete such variable from Γ
- Generate the constraint ensuring its occurance at location i
- Generate the constraints ensuring its absence from other locations
- Update the event counter

  if $(\phi' = True)$ **then**
    Break

If formula is true then stop generating constraints

  for (*each* $a \in \Gamma$ *s.t.* $a = \mathbf{X}a'$) **do**
    $\Gamma \leftarrow (\Gamma \setminus \{a\}) \cup \{a'\}$
    $\Gamma' \leftarrow (\Gamma' \setminus \{a\}) \cup \{a'\}$
    $i_a{+}{+}$
  $i{+}{+}$

-Counters that include "**X**" will have "**X**" removed
- To be counted next round

for *each* $b \in \Gamma'$ **do**
  $r \leftarrow r \wedge \mathbf{G}\neg b$
return r

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \longrightarrow \mathbf{F} b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee \mathbf{F} b_{\pi_2}$

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee (true \mathbf{U} b_{\pi_2})$

$\Gamma = \{\neg a\}$

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \longrightarrow \mathbf{F} b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2}.\ \neg a_{\pi_1} \vee \mathbf{F} b_{\pi_2}$

$\phi = \forall_{\pi_1 \pi_2}.\ \neg a_{\pi_1} \vee (true \mathbf{U} b_{\pi_2})$

$\Gamma = \{\neg a\}$

Step 2: Separating formula and finding constraints for each part

$\pi_1 =$ d c b

$constraints_{\mathbf{F} b} =$

$constraints_{\neg a} =$

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \longrightarrow \mathbf{F} b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee \mathbf{F} b_{\pi_2}$

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee (true \mathbf{U} b_{\pi_2})$

$\Gamma = \{\neg a\}$

Step 2: Separating formula and finding constraints for each part

$\pi_1 =$ d c b

$constraints_{\mathbf{F} b} = ?$

$constraints_{\neg a} = \neg a$

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \longrightarrow \mathbf{F} b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee \mathbf{F} b_{\pi_2}$

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee (\mathit{true} \mathbf{U} b_{\pi_2})$

$\Gamma = \{\neg a\}$

Step 2: Separating formula and finding constraints for each part

$\pi_1 =$ d c b

$\mathit{constraints}_{\mathbf{F} b} = ?$

$\mathit{constraints}_{\neg a} = \neg a$

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \longrightarrow \mathbf{F} b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee \mathbf{F} b_{\pi_2}$

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee (true \mathbf{U} b_{\pi_2})$

$\Gamma = \{\neg a\}$

Step 2: Separating formula and finding constraints for each part

$\pi_1 =$ d c b

$constraints_{\mathbf{F} b} = \mathbf{F} b$

$constraints_{\neg a} = \neg a$

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall\pi_1.\forall\pi_2.\ a_{\pi_1} \longrightarrow \mathbf{F}\, b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1\pi_2}.\ \neg a_{\pi_1} \vee \mathbf{F}\, b_{\pi_2}$

$\phi = \forall_{\pi_1\pi_2}.\ \neg a_{\pi_1} \vee (\mathit{true}\,\mathbf{U}\, b_{\pi_2})$

$\Gamma = \{\neg a\}$

$\mathit{constraints}_{\mathbf{F}b} = \mathbf{F}\, b$

$\mathit{constraints}_{\neg a} = \neg a$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ e e e

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall\pi_1.\forall\pi_2.\ a_{\pi_1} \longrightarrow \mathbf{F}\, b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1\pi_2}.\ \neg a_{\pi_1} \vee \mathbf{F}\, b_{\pi_2}$

$\phi = \forall_{\pi_1\pi_2}.\ \neg a_{\pi_1} \vee (true\mathbf{U}\, b_{\pi_2})$

$\Gamma = \{\neg a\}$

$constraints_{\mathbf{F}b} = \mathbf{F}\, b$

$constraints_{\neg a} = \neg a$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ e e e

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \longrightarrow \mathbf{F}\, b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee \mathbf{F}\, b_{\pi_2}$

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee (\mathit{true}\, \mathbf{U}\, b_{\pi_2})$

$\Gamma = \{\neg a\}$

$\mathit{constraints}_{\mathbf{F} b} = \mathbf{F}\, b$

$\mathit{constraints}_{\neg a} = \neg a$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ e e e

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \longrightarrow \mathbf{F} b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee \mathbf{F} b_{\pi_2}$

$\phi = \forall_{\pi_1 \pi_2} .\ \neg a_{\pi_1} \vee (\mathit{true} \mathbf{U} b_{\pi_2})$

$\Gamma = \{\neg a\}$

$\cancel{\mathit{constraints}_{\mathbf{F} b} = \mathbf{F} b}$

$\mathit{constraints}_{\neg a} = \neg a$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ e e e

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1.\forall \pi_2.\ a_{\pi_1} \longrightarrow \mathbf{F}\, b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2}.\ \neg a_{\pi_1} \vee \mathbf{F}\, b_{\pi_2}$
$\phi = \forall_{\pi_1 \pi_2}.\ \neg a_{\pi_1} \vee (\mathit{true}\, \mathbf{U}\, b_{\pi_2})$
$\Gamma = \{\neg a\}$

$\cancel{\mathit{constraints}_{\mathbf{F}b} = \mathbf{F}b}$
$\mathit{constraints}_{\neg a} = \neg a$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ e e e

π_2 only agrees with $\mathit{constraints}_{\neg a}$, no b was found

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 . \; a_{\pi_1} \longrightarrow \mathbf{F} b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2} . \; \neg a_{\pi_1} \vee \mathbf{F} b_{\pi_2}$

$\phi = \forall_{\pi_1 \pi_2} . \; \neg a_{\pi_1} \vee (true \mathbf{U} b_{\pi_2})$

$\Gamma = \{\neg a\}$

$\cancel{constraints_{\mathbf{F} b} = \mathbf{F} b}$

$constraints_{\neg a} = \neg a$

Step 3: Check incoming traces agree with the constraints

$\pi_3 =$ a c b

Rewriting-Based Monitoring: Running Example 1

$$\phi = \forall \pi_1 . \forall \pi_2 .\ a_{\pi_1} \longrightarrow \mathbf{F}\, b_{\pi_2}$$

Step 1: Finding Γ

$\phi = \forall_{\pi_1 \pi_2}.\ \neg a_{\pi_1} \vee \mathbf{F}\, b_{\pi_2}$

$\phi = \forall_{\pi_1 \pi_2}.\ \neg a_{\pi_1} \vee (\mathit{true}\, \mathbf{U}\, b_{\pi_2})$

$\Gamma = \{\neg a\}$

$\cancel{\mathit{constraints}_{\mathbf{F}b} = \mathbf{F} b}$

$\cancel{\mathit{constraints}_{\neg a} = \neg a}$

Step 3: Check incoming traces agree with the constraints

$\pi_3 =$ a c b

π_3 disagrees with the last constraint, a was observed in the first location

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1 \pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

Step 1: Finding Γ
$\Gamma = \{b, \#c\}$

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1 \pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$\Gamma = \{b, \#c\}$

Step 2: Finding constraints

$\pi_1 =$ a a ab ac ac ab

constraints $=$

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1 \pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$\Gamma = \{b, \#c\}$

Step 2: Finding constraints

$\pi_1 =$ a a ab ac ac ab

constraints $= \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c$

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1 \pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$\Gamma = \{b\}$

Step 2: Finding constraints

$\pi_1 =$ a a ab ac ac ab

$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 \neg b \wedge \mathbf{X}^3 c$

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$\Gamma = \{b\}$

Step 2: Finding constraints

$\pi_1 =$ a a ab ac ac ab

$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 \neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4 \neg b$

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1 \pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$\Gamma = \{b\}$

Step 2: Finding constraints

$\pi_1 =$ a a ab ac ac ab

$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 \neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4 \neg b \wedge \mathbf{X}^5 b$

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1 \pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$\Gamma = \{b\}$

Step 2: Finding constraints

$\pi_1 =$ a a ab ac ac ab

constraints $= \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 \neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4 \neg b \wedge \mathbf{X}^5 b$

we count only the first occurrence of c

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2\neg c \wedge \mathbf{X}^3\neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4\neg b \wedge \mathbf{X}^5 b$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ a a ab ac ac ab

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2\neg c \wedge \mathbf{X}^3\neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4\neg b \wedge \mathbf{X}^5 b$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ a a ab ac ac ab

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2\neg c \wedge \mathbf{X}^3\neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4\neg b \wedge \mathbf{X}^5 b$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ a a ab ac ac ab

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1 \pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 \neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4 \neg b \wedge \mathbf{X}^5 b$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ a a ab ac ac ab

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2\neg c \wedge \mathbf{X}^3\neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4\neg b \wedge \mathbf{X}^5 b$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ a a ab ac ac ab

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1 \pi_2}. \; (a_{\pi_1} \; \mathbf{U} \; b_{\pi_2}) \; \mathbf{U} \; c_{\pi_1}$$

$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 \neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4 \neg b \wedge \mathbf{X}^5 b$

Step 3: Check incoming traces agree with the constraints

$\pi_2 =$ a a ab ac ac ab

π_2 does not create a violation, even though c is observed more than once

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$$\mathit{constraints} = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2\neg c \wedge \mathbf{X}^3\neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4\neg b \wedge \mathbf{X}^5 b$$

Step 3: Check incoming traces agree with the constraints

$\pi_3 =$ abc ab ab ac ac ab

Rewriting-Based Monitoring: Running Example 2

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

~~$constraints = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2\neg c \wedge \mathbf{X}^3\neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4\neg b \wedge \mathbf{X}^5 b$~~

Step 3: Check incoming traces agree with the constraints

$\pi_3 =$ abc ab ab ac ac ab

π_3 violates our constraint as the first location has b and c

Rewriting-Based Monitoring: Running Example 3

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$$\Gamma = \{b, \#c\}$$

Rewriting-Based Monitoring: Running Example 3

$$\phi = \forall_{\pi_1 \pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$$\Gamma = \{b, \#c\}$$

$\pi_1 =$ a a ab

$\pi_2 =$ a a

$constraints_{\pi_1} = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c$

$constraints_{\pi_2} = \neg(bc) \wedge \mathbf{X}\neg(bc)$

Rewriting-Based Monitoring: Running Example 3

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$$\Gamma = \{b, \#c\}$$

$\pi_1 =$ a a ab

$\pi_2 =$ a a

$constraints_{\pi_1} = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2\neg c$

$constraints_{\pi_2} = \neg(bc) \wedge \mathbf{X}\neg(bc)$

At this point, there is no constraint disagreement

Rewriting-Based Monitoring: Running Example 3

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$$\Gamma = \{b, \#c\}$$

$\pi_1 =$ a a ab ac a ab

$\pi_2 =$ a a ab ab

$constraints_{\pi_1} = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 \neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4 \neg b \wedge \mathbf{X}^5 b$

$constraints_{\pi_2} = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 b \wedge \mathbf{X}^3 \neg c$

Rewriting-Based Monitoring: Running Example 3

$$\phi = \forall_{\pi_1\pi_2}.\ (a_{\pi_1}\ \mathbf{U}\ b_{\pi_2})\ \mathbf{U}\ c_{\pi_1}$$

$$\Gamma = \{b, \#c\}$$

$\pi_1 =$ a a ab ac a ab

$\pi_2 =$ a a ab ab

$$constraints_{\pi_1} = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 \neg b \wedge \mathbf{X}^3 c \wedge \mathbf{X}^4 \neg b \wedge \mathbf{X}^5 b$$

$$constraints_{\pi_2} = \neg(bc) \wedge \mathbf{X}\neg(bc) \wedge \mathbf{X}^2 b \wedge \mathbf{X}^2 \neg c \wedge \mathbf{X}^3 b \wedge \mathbf{X}^3 \neg c$$

There is a violation among constraints for π_1 and π_2

Presentation outline

1. Finite Semantics for LTL
2. Challenges in RV for HyperLTL
3. Rewriting-based RV Algorithm for Alternation-free HyperLTL Formulas
   - Identifying the Propositions of Interest
   - Rewriting-based RV for FLTL
4. Conclusion

Conclusion

Summary

- Monitorability for HyperLTL
- Finite semantics for HyperLTL
- Rewriting-based RV algorithm for alternation-free HyperLTL

Future Work

- Automata-based monitoring (hyper monitors!)
- RV for Alternating HyperLTL

Questions

Thank You!