Reactive Synthesis from Extended Bounded Response LTL Specifications

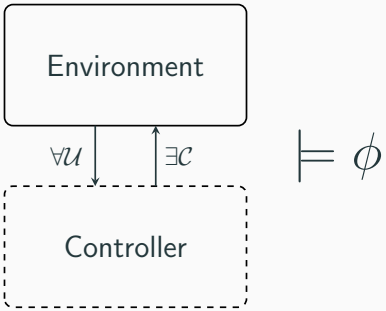
A. Cimatti[1] L. Geatti[1,2] N. Gigante[2] A. Montanari[2] S. Tonetta[1]

informal Formal Methods Meetings, 11 December 2020

[1]Fondazione Bruno Kessler
Trento, Italy

[2]University of Udine
Udine, Italy

Introduction



1. what are realizability and reactive synthesis?
   - model-based design: all the effort on the quality of the specification
   - culmination of declarative programming
2. complexity:
   - for S1S: non-elementary
   - for LTL: 2EXPTIME-complete.

Definition (Strategy)

Let $\Sigma = \mathcal{C} \cup \mathcal{U}$ be an alphabet partitioned into the set of controllable variables $\mathcal{C}$ and the set of uncontrollable ones $\mathcal{U}$, such that $\mathcal{C} \cap \mathcal{U} = \varnothing$. A strategy for Controller is a function

$$g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$$

that, given the sequence $\mathsf{U} = \langle \mathsf{U}_0, \ldots, \mathsf{U}_n \rangle$ of choices made by *Environment* so far, determines the current choices $\mathsf{C}_n = g(\mathsf{U})$ of *Controller*.

Definition (Strategy)

Let $\Sigma = \mathcal{C} \cup \mathcal{U}$ be an alphabet partitioned into the set of controllable variables $\mathcal{C}$ and the set of uncontrollable ones $\mathcal{U}$, such that $\mathcal{C} \cap \mathcal{U} = \varnothing$. A strategy for Controller is a function

$$g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$$

that, given the sequence $\mathsf{U} = \langle \mathsf{U}_0, \ldots, \mathsf{U}_n \rangle$ of choices made by *Environment* so far, determines the current choices $\mathsf{C}_n = g(\mathsf{U})$ of *Controller*.

Definition (Realizability and Synthesis)

Let ϕ be a temporal formula over the alphabet $\Sigma = \mathcal{C} \cup \mathcal{U}$. We say that ϕ is realizable if and only if

- $\exists g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$
- $\forall$ infinite sequence $\mathsf{U} = \langle \mathsf{U}_0, \mathsf{U}_1, \ldots \rangle \in (2^{\mathcal{U}})^\omega$
- $\langle \mathsf{U}_0 \cup g(\langle \mathsf{U}_0 \rangle), \mathsf{U}_1 \cup g(\langle \mathsf{U}_0, \mathsf{U}_1 \rangle), \ldots \rangle \models \phi$

If ϕ is realizable, the synthesis problem is the problem of computing such a strategy g.

Small Model Property

Definition (Finitely representable strategies)
Let $g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$ be a strategy. We say that g is finitely representable iff there exists a Mealy machine M_g over the input and ouput alphabet $\Sigma_{\mathcal{U}} = 2^{\mathcal{U}}$ and $\Sigma_{\mathcal{C}} = 2^{\mathcal{C}}$, respectively, such that $\mathcal{L}(g) = \mathcal{L}(M_g)$.

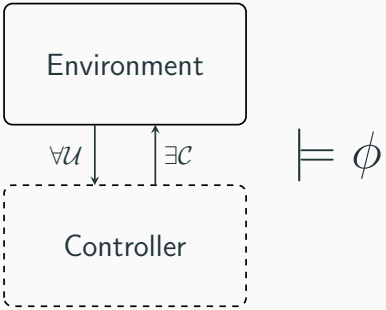
Small Model Property

Definition (Finitely representable strategies)
Let $g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$ be a strategy. We say that g is finitely representable iff there exists a Mealy machine M_g over the input and ouput alphabet $\Sigma_{\mathcal{U}} = 2^{\mathcal{U}}$ and $\Sigma_{\mathcal{C}} = 2^{\mathcal{C}}$, respectively, such that $\mathcal{L}(g) = \mathcal{L}(M_g)$.

Proposition (Small model property of LTL)
Let ϕ be an LTL *formula and $n = |\phi|$. If ϕ is realizable, then there exists a finitely representable strategy g such that*

1. *M_g has at most $2^{2^{c \cdot n}}$ states, for some constant $c \in \mathbb{N}$, and*
2. *$\mathcal{L}(M_g) \subseteq \mathcal{L}(\phi)$.*

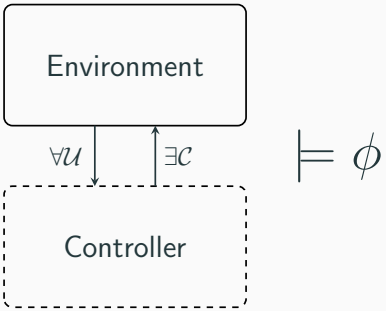
Introduction



- realizability is usually modeled as a two-players game over an arena/automaton $\mathcal{A}_\phi$ built from ϕ:

  $\Rightarrow$ objective: find M_g such that $\mathcal{L}(M_g) \subseteq \mathcal{L}(\mathcal{A}_\phi)$

Introduction



- realizability is usually modeled as a two-players game over an arena/automaton $\mathcal{A}_\phi$ built from ϕ:

  $\Rightarrow$ objective: find M_g such that
  $$\mathcal{L}(M_g) \subseteq \mathcal{L}(\mathcal{A}_\phi)$$
- Determinism is crucial because:
  - there are simple algorithms for synthesis over deterministic arenas

    $\Rightarrow$ backward fixpoint computations

Definition (Explicit Safety Automata)
A *safety automaton* (SSA) is a tuple $\mathcal{A} = (\Sigma, Q, I, T, S)$, such that

1. Σ is a set of *input variables*, and
2. Q is a set of states
3. $I \subseteq Q$, $T \subseteq Q \times Q$, and $S \subseteq Q$ the set of initial states, the transition relation, and the set of safe states, respectively.

$\mathcal{A}$ accepts all and only the infinite words for which there exists a run that visits only safe states.

Definition (Explicit Safety Automata)
A *safety automaton* (SSA) is a tuple $\mathcal{A} = (\Sigma, Q, I, T, S)$, such that

1. Σ is a set of *input variables*, and
2. Q is a set of states
3. $I \subseteq Q$, $T \subseteq Q \times Q$, and $S \subseteq Q$ the set of initial states, the transition relation, and the set of safe states, respectively.

$\mathcal{A}$ accepts all and only the infinite words for which there exists a run that visits only safe states.

Definition (Symbolic Safety Automata)

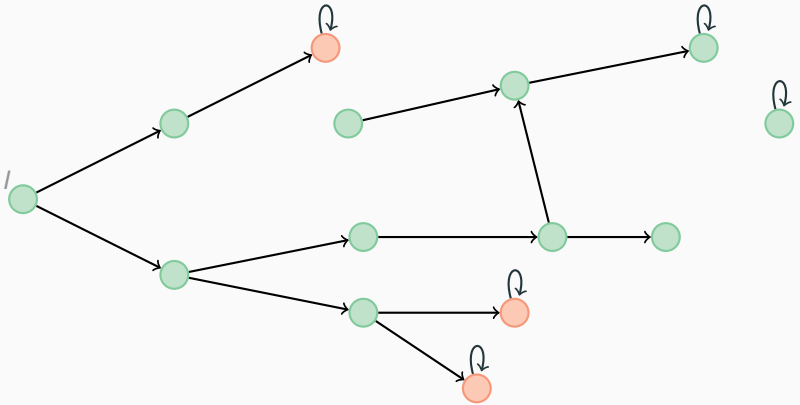
A *symbolic safety automaton* (SSA) is a tuple $\mathcal{A} = (V, I, T, S)$, such that

1. $V = X \cup \Sigma$, where X is a set of *state variables* and Σ is a set of *input variables*, and
2. $I(X)$, $T(X, \Sigma, X')$, and $S(X)$, with $X' = \{x' \mid x \in X\}$, are Boolean formulae which define the set of initial states, the transition relation, and the set of safe states, respectively.

Backward fixpoint algorithm for safety automata

Start with the final states of $\mathcal{A}_\phi$ and go backward with the operator $Pre(S) := \{s \mid \forall u \in 2^{\mathcal{U}} \,.\, \exists c \in 2^{\mathcal{C}} \,.\, s \xrightarrow{u,c} s',\ \text{for some } s' \in S\}$.

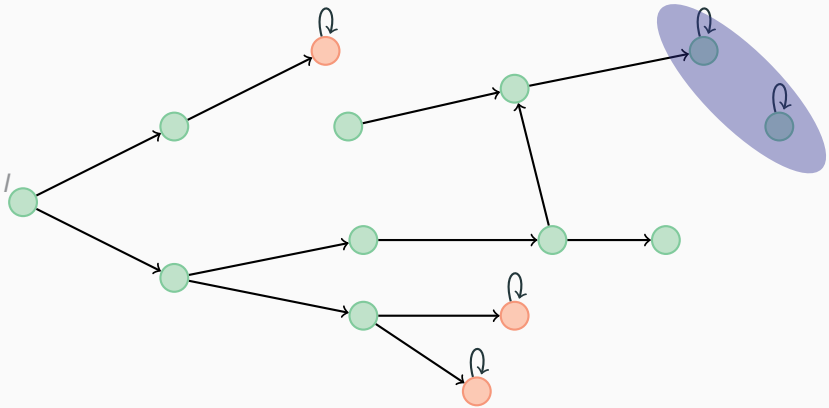
Stop when arrived to an initial state.



Backward fixpoint algorithm for safety automata

Start with the final states of $\mathcal{A}_\phi$ and go backward with the operator $Pre(S) := \{s \mid \forall u \in 2^{\mathcal{U}} \,.\, \exists c \in 2^{\mathcal{C}} \,.\, s \xrightarrow{u,c} s',\ \text{for some } s' \in S\}$.

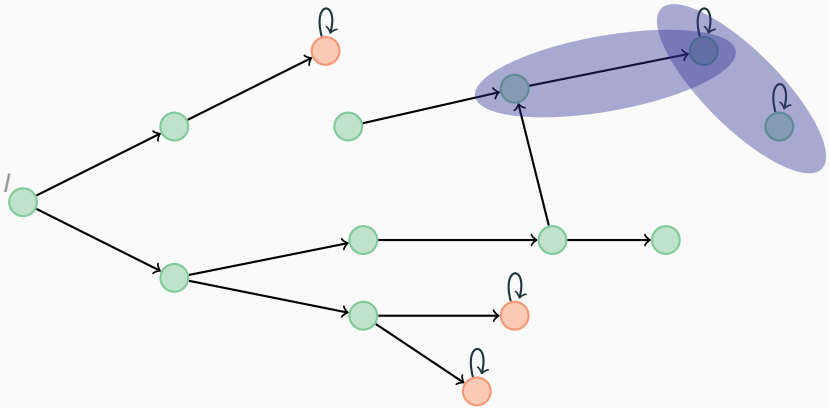
Stop when arrived to an initial state.



Backward fixpoint algorithm for safety automata

Start with the final states of $\mathcal{A}_\phi$ and go backward with the operator $Pre(S) := \{s \mid \forall u \in 2^{\mathcal{U}} \,.\, \exists c \in 2^{\mathcal{C}} \,.\, s \xrightarrow{u,c} s',\ \text{for some } s' \in S\}$.

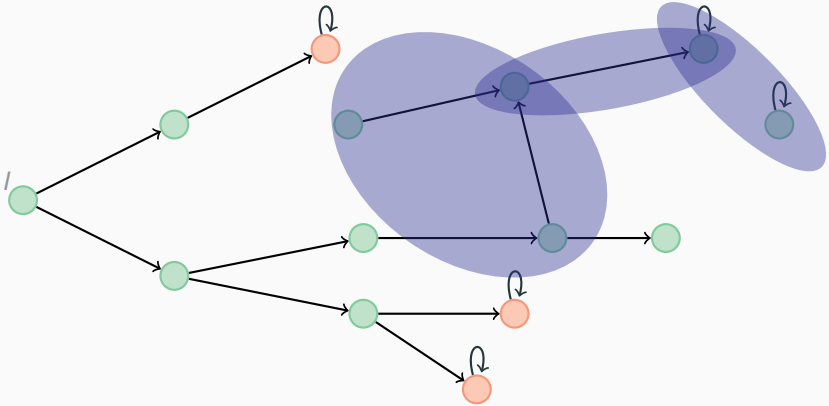
Stop when arrived to an initial state.



Backward fixpoint algorithm for safety automata

Start with the final states of $\mathcal{A}_\phi$ and go backward with the operator $Pre(S) := \{s \mid \forall u \in 2^{\mathcal{U}} \,.\, \exists c \in 2^{\mathcal{C}} \,.\, s \xrightarrow{u,c} s',\ \text{for some } s' \in S\}$.

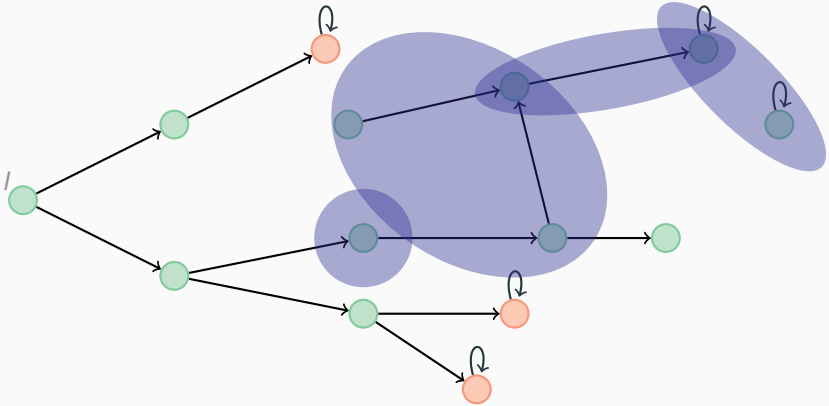
Stop when arrived to an initial state.



Backward fixpoint algorithm for safety automata

Start with the final states of $\mathcal{A}_\phi$ and go backward with the operator $Pre(S) := \{s \mid \forall u \in 2^{\mathcal{U}} \,.\, \exists c \in 2^{\mathcal{C}} \,.\, s \xrightarrow{u,c} s',\ \text{for some } s' \in S\}$.

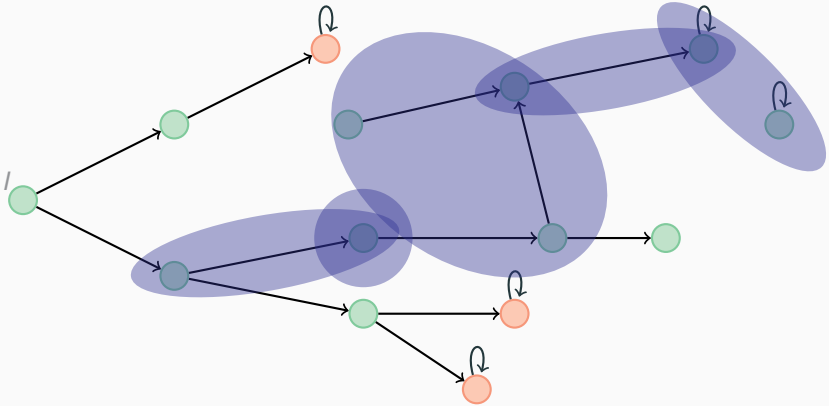
Stop when arrived to an initial state.



Backward fixpoint algorithm for safety automata

Start with the final states of $\mathcal{A}_\phi$ and go backward with the operator $Pre(S) := \{s \mid \forall u \in 2^{\mathcal{U}} \,.\, \exists c \in 2^{\mathcal{C}} \,.\, s \xrightarrow{u,c} s',\ \text{for some } s' \in S\}$.

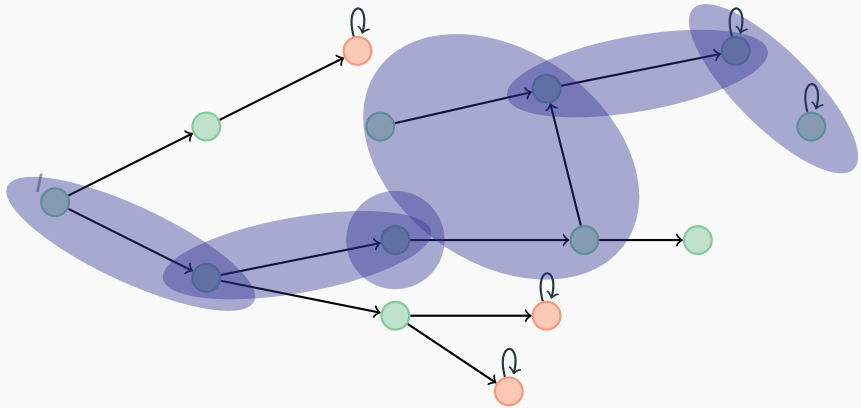
Stop when arrived to an initial state.



Backward fixpoint algorithm for safety automata

Start with the final states of $\mathcal{A}_\phi$ and go backward with the operator $Pre(S) := \{s \mid \forall u \in 2^{\mathcal{U}} \,.\, \exists c \in 2^{\mathcal{C}} \,.\, s \xrightarrow{u,c} s',\ \text{for some } s' \in S\}$.

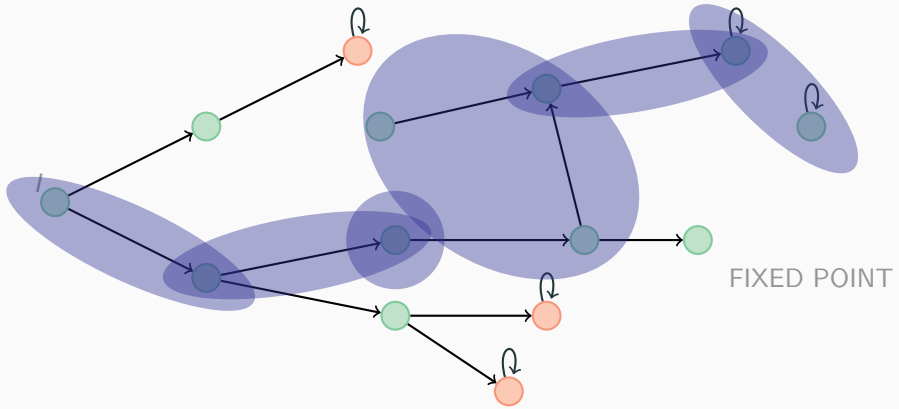
Stop when arrived to an initial state.



Backward fixpoint algorithm for safety automata

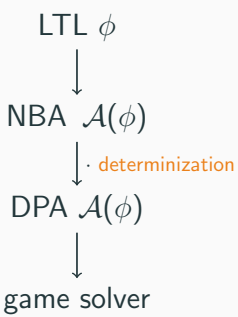
Start with the final states of $\mathcal{A}_\phi$ and go backward with the operator $Pre(S) := \{s \mid \forall u \in 2^{\mathcal{U}} \,.\, \exists c \in 2^{\mathcal{C}} \,.\, s \xrightarrow{u,c} s',\ \text{for some } s' \in S\}$.

Stop when arrived to an initial state.



Classical Approach for Reactive Synthesis of LTL

Standard approach:



Determinization of Büchi automata (Safra's algorithm)

- very complicated
- difficult to implement
- not amenable to optimizations

Related work

Standard approach:

Research mainly focused on two lines

- finding good algorithms for the average case
  - Safraless approaches
    - Bounded synthesis

explicit-state setting (possibly with symbolic labels)

Related work

Standard approach:

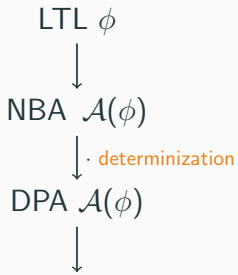
Research mainly focused on two lines

- finding good algorithms for the average case
  - Safraless approaches
    - Bounded synthesis
- restricting the expressiveness of the specification language
  - GR(1)
  - Safety LTL: only universal operators

explicit-state setting (possibly with symbolic labels)

Related work

Standard approach:



Research mainly focused on two lines

- finding good algorithms for the average case
  - Safraless approaches
    - Bounded synthesis
- restricting the expressiveness of the specification language
  - GR(1)
  - Safety LTL: only universal operators

explicit-state setting (possibly with symbolic labels)

Summary

In this work:

1. we introduce a new fragment of LTL, called $\mathsf{LTL}_{\mathsf{EBR}}$, which combines
   - bounded operators, like $\mathsf{G}^{[a,b]}$ and $\mathsf{F}^{[a,b]}$
   - universal unbounded operators, like G and $\mathcal{R}$.

Summary

In this work:

1. we introduce a new fragment of LTL, called $\mathrm{LTL}_{\mathrm{EBR}}$, which combines
   - bounded operators, like $\mathsf{G}^{[a,b]}$ and $\mathsf{F}^{[a,b]}$
   - universal unbounded operators, like G and $\mathcal{R}$.
2. we show that $\mathrm{LTL}_{\mathrm{EBR}}$ formulas can be directly translated into deterministic symbolic automata over infinite words.
   - the translation is fully symbolic: no explicit automaton is ever built.

Summary

In this work:

1. we introduce a new fragment of LTL, called $\mathrm{LTL}_{\mathrm{EBR}}$, which combines
   - bounded operators, like $\mathsf{G}^{[a,b]}$ and $\mathsf{F}^{[a,b]}$
   - universal unbounded operators, like G and $\mathcal{R}$.
2. we show that $\mathrm{LTL}_{\mathrm{EBR}}$ formulas can be directly translated into deterministic symbolic automata over infinite words.
   - the translation is fully symbolic: no explicit automaton is ever built.
3. we solve reactive synthesis from $\mathrm{LTL}_{\mathrm{EBR}}$ specs by using existing algorithms for safety synthesis to solve the game on the automaton.

Summary

In this work:

1. we introduce a new fragment of LTL, called $\mathrm{LTL}_{\mathrm{EBR}}$, which combines
   - bounded operators, like $\mathsf{G}^{[a,b]}$ and $\mathsf{F}^{[a,b]}$
   - universal unbounded operators, like G and $\mathcal{R}$.
2. we show that $\mathrm{LTL}_{\mathrm{EBR}}$ formulas can be directly translated into deterministic symbolic automata over infinite words.
   - the translation is fully symbolic: no explicit automaton is ever built.
3. we solve reactive synthesis from $\mathrm{LTL}_{\mathrm{EBR}}$ specs by using existing algorithms for safety synthesis to solve the game on the automaton.
4. implementation + experimental evaluation

Definition (The logic $\mathsf{LTL}_{\mathsf{EBR}}$)

Let $a, b \in \mathbb{N}$. An $\mathsf{LTL}_{\mathsf{EBR}}$ formula χ is inductively defined as follows:

Full Past Layer

$$\eta := p \mid \neg\eta \mid \eta_1 \vee \eta_2 \mid \mathsf{Y}\eta \mid \eta_1 \,\mathcal{S}\, \eta_2$$

Full Bounded Layer

$$\psi := \eta \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{X}\psi \mid \psi_1 \,\mathcal{U}^{[a,b]}\, \psi_2$$

Future Layer

$$\phi := \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid \psi \,\mathcal{R}\, \phi$$

Boolean Layer

$$\chi := \phi \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2$$

Examples:

- bounded response requirements:

$$\mathsf{G}(r \rightarrow \mathsf{F}^{[0,k]} g)$$

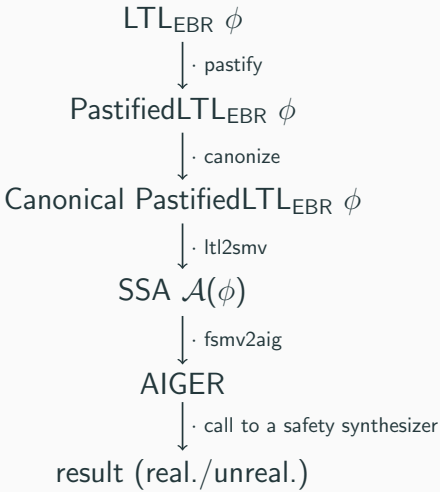
- mutual exclusion:

$$\mathsf{G}(\bigwedge_{1 \leq i < j \leq n} \neg(g_i \wedge g_j))$$

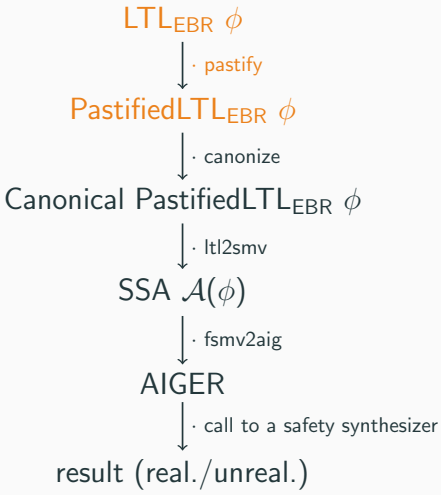
- prioritized bounded response requirements:

$$\bigwedge_{1 \leq i < j \leq n} \mathsf{G}((r_i \wedge r_j) \rightarrow (\neg g_j)\, \mathcal{U}^{[0,k]}\, g_i)$$

Overall procedure

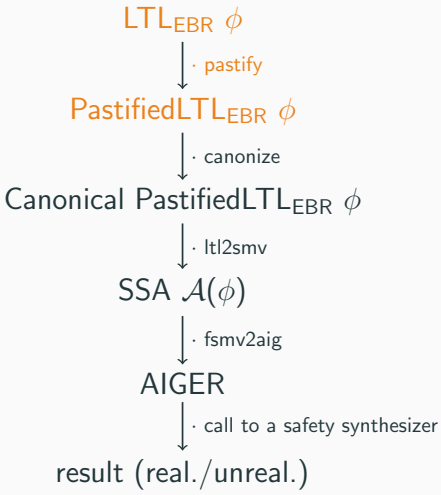


From $\mathsf{LTL}_{\mathsf{EBR}}$ to symbolic automata



Pastification: (full-bounded $\psi \rightsquigarrow$ full-past ψ')

From $\mathsf{LTL}_{\mathsf{EBR}}$ to symbolic automata

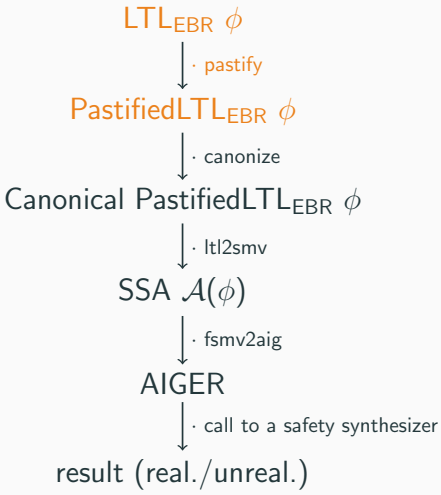


Pastification: (full-bounded $\psi \rightsquigarrow$ full-past ψ')

- Motivation: temporal monitors (aka temporal testers) for full-past formulas are deterministic.

  $\Rightarrow$ "the past already happened"

From $\mathsf{LTL}_{\mathsf{EBR}}$ to symbolic automata



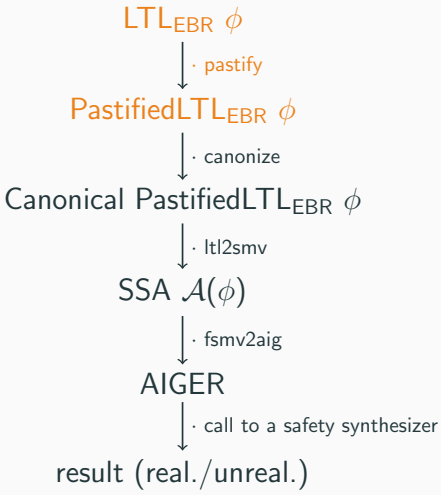
Pastification: (full-bounded $\psi \rightsquigarrow$ full-past ψ')

- Motivation: temporal monitors (aka temporal testers) for full-past formulas are deterministic.

  $\Rightarrow$ "the past already happened"

- Thus, we want to pastify as much subformulas of ϕ as we can.

From $\mathrm{LTL}_{\mathrm{EBR}}$ to symbolic automata



Pastification: (full-bounded $\psi \rightsquigarrow$ full-past ψ')

- Motivation: temporal monitors (aka temporal testers) for full-past formulas are deterministic.

  $\Rightarrow$ "the past already happened"

- Thus, we want to pastify as much subformulas of ϕ as we can.
- All the *full-bounded* subformulas of ϕ can be pastified.

Pastification

- originally introduced for the synthesis of explicit-state timed automata[1].

[1] Maler, Nickovic, and Pnueli, "On synthesizing controllers from bounded-response properties".

Pastification

- originally introduced for the synthesis of explicit-state timed automata[1].
- Let $\phi \in \mathsf{LTL}_{\mathsf{FB}}$, that is, all future temporal operators are bounded. There exists a time point $d \in \mathbb{N}$, the temporal depth of ϕ, such that the subsequent states cannot be constrained by ϕ.

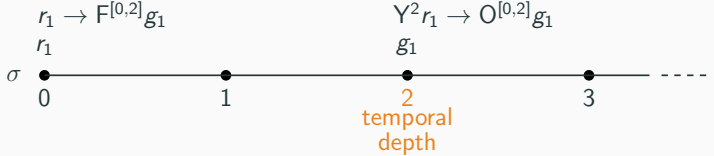
- Example: $\phi \coloneqq r_1 \rightarrow \mathsf{F}^{[0,2]} g_1$

[1] Maler, Nickovic, and Pnueli, "On synthesizing controllers from bounded-response properties".

Pastification

- originally introduced for the synthesis of explicit-state timed automata[1].
- Let $\phi \in \mathrm{LTL}_{\mathrm{FB}}$, that is, all future temporal operators are bounded. There exists a time point $d \in \mathbb{N}$, the temporal depth of ϕ, such that the subsequent states cannot be constrained by ϕ.

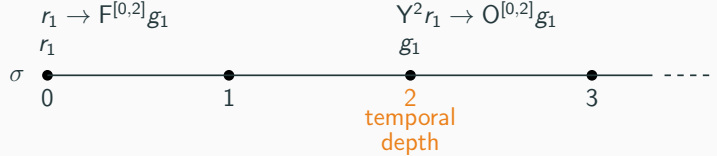
- Example: $\phi \coloneqq r_1 \rightarrow \mathsf{F}^{[0,2]} g_1$



[1] Maler, Nickovic, and Pnueli, "On synthesizing controllers from bounded-response properties".

Pastification

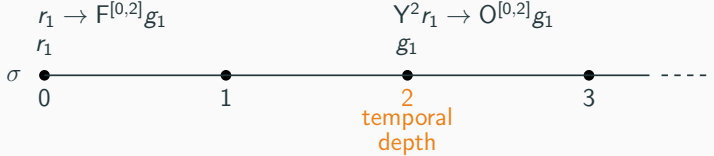
- originally introduced for the synthesis of explicit-state timed automata[1].
- Let $\phi \in \mathsf{LTL}_{\mathsf{FB}}$, that is, all future temporal operators are bounded. There exists a time point $d \in \mathbb{N}$, the temporal depth of ϕ, such that the subsequent states cannot be constrained by ϕ.
- Thus, we can write a formula, the pastification of ϕ, that uses only past operators and that is equivalent to ϕ when interpreted at d.
- Example: $\phi := r_1 \rightarrow \mathsf{F}^{[0,2]} g_1$



[1] Maler, Nickovic, and Pnueli, "On synthesizing controllers from bounded-response properties".

Pastification

- originally introduced for the synthesis of explicit-state timed automata[1].
- Let $\phi \in \mathsf{LTL}_{\mathsf{FB}}$, that is, all future temporal operators are bounded. There exists a time point $d \in \mathbb{N}$, the temporal depth of ϕ, such that the subsequent states cannot be constrained by ϕ.
- Thus, we can write a formula, the pastification of ϕ, that uses only past operators and that is equivalent to ϕ when interpreted at d.
- Example: $\phi := r_1 \rightarrow \mathsf{F}^{[0,2]} g_1$



It holds that: $r_1 \rightarrow \mathsf{F}^{[0,2]} g_1 \equiv \mathsf{X}^2(\mathsf{Y}^2 r_1 \rightarrow \mathsf{O}^{[0,2]} g_1)$.

[1] Maler, Nickovic, and Pnueli, "On synthesizing controllers from bounded-response properties".

Pastification - Soundness and Complexity

Proposition (Soundness of pastification)

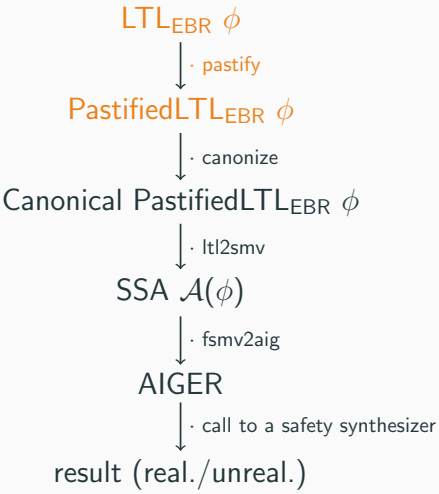
Let φ be a $\mathsf{LTL}_{\mathsf{FB}}$ formula. For all state sequences $\sigma \in (2^{\Sigma})^{\omega}$, all $i \in \mathbb{N}$, and all $d \geq D(\phi)$, it holds that:

$$\sigma, i \models \varphi \;\Leftrightarrow\; \sigma, i \models \mathsf{X}^{d}\Pi(\varphi, d)$$

Proposition

Let ϕ be a $\mathsf{LTL}_{\mathsf{FB}}$ formula. Then, pastify(ϕ) is a formula of size $\mathcal{O}(n^2 \cdot M^{\log_2 n+1})$, where $n = |\phi|$ and M is the greatest constant in ϕ.

Pastification of all $\mathrm{LTL}_{\mathrm{FB}}$ subformulas



We pastify all the $\mathrm{LTL}_{\mathrm{FB}}$ subformulas of the starting $\mathrm{LTL}_{\mathrm{EBR}}$ formula ϕ. Example:

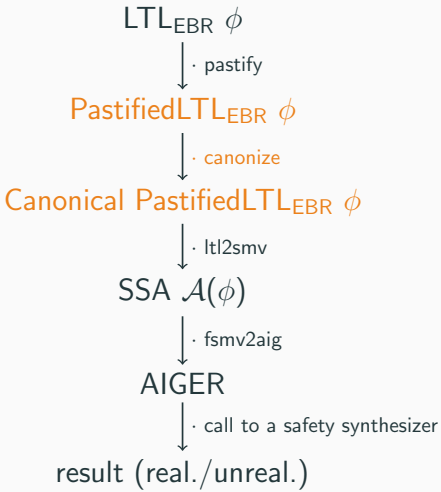
$$\mathsf{G}(r_1 \rightarrow \mathsf{F}^{[0,2]} g_1) \quad \vee \quad \mathsf{G}(r_1 \rightarrow \mathsf{F}^{[0,3]}(g1 \wedge g2))$$

$$\mathsf{G}(\mathsf{X}^2(\mathsf{Y}^2 r_1 \rightarrow \mathsf{O}^{[0,2]} g_1)) \vee \mathsf{G}(\mathsf{X}^3(\mathsf{Y}^3 r_1 \rightarrow \mathsf{O}^{[0,3]}(g_1 \wedge g_2)))$$

Proposition

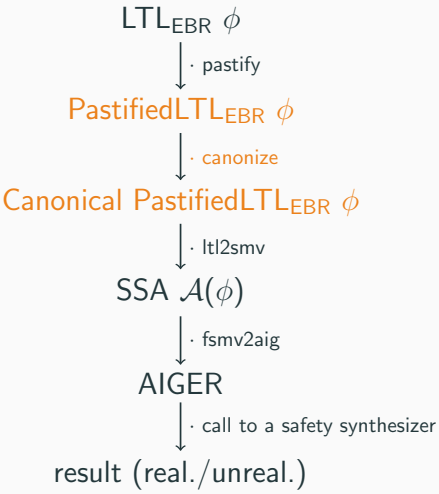
For each $\mathrm{LTL}_{\mathrm{EBR}}$ *formula* ϕ*, there is an equivalent* PastifiedLTL$_{\mathrm{EBR}}$ *formula* ϕ' *of size* $\mathcal{O}(n^3 \cdot M^{\log_2 n+1})$*, where* $n = |\phi|$ *and* M *is the greatest constant in* ϕ*.*

Canonization



- Now, the objective is to turn ϕ into a form amenable to direct translation into symbolic automata.

Canonization

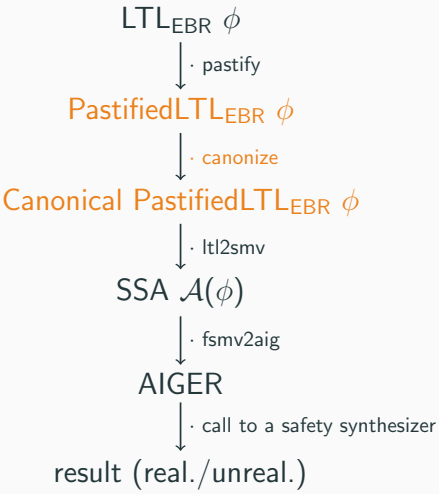


- Now, the objective is to turn ϕ into a form amenable to direct translation into symbolic automata.
- Canonical Form:

$$\mathsf{X}^{i_1}\psi \otimes \cdots \otimes \mathsf{X}^{i_2}\psi \otimes$$
$$\mathsf{X}^{i_3}\mathsf{G}\psi \otimes \cdots \otimes \mathsf{X}^{i_4}\mathsf{G}\psi \otimes$$
$$\mathsf{X}^{i_5}(\psi_1 \mathcal{R} \psi_2) \otimes \cdots \otimes \mathsf{X}^{i_6}(\psi_1 \mathcal{R} \psi_2)$$

where each ψ, ψ_1, ψ_2 is a full-past formula, and $\otimes \in \{\wedge, \vee\}$.

Canonization - Rewriting Rules



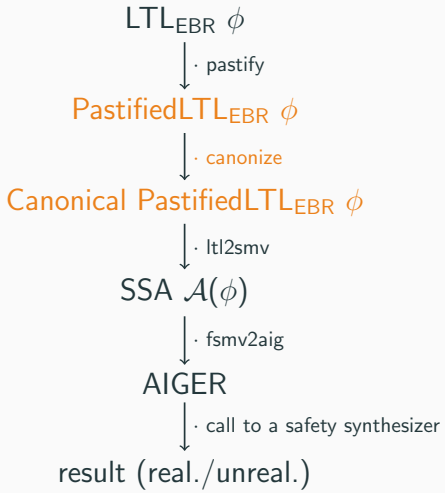
canonize(ϕ) is the formula obtained by recursively apply the $R_1 - R_7$ rules in a bottom-up fashion, followed by the application of the R_{flat} rule:

$$R_1 : \mathsf{X}(\psi_1 \wedge \psi_2) \rightsquigarrow \mathsf{X}\psi_1 \wedge \mathsf{X}\psi_2$$

$$R_2 : \psi \,\mathcal{R}\, (\psi_1 \wedge \psi_2) \rightsquigarrow \psi \,\mathcal{R}\, \psi_1 \wedge \psi \,\mathcal{R}\, \psi_2$$

$$R_3 : (\mathsf{X}^i\psi_1) \,\mathcal{R}\, (\mathsf{X}^j\psi_2) \rightsquigarrow \begin{cases} \mathsf{X}^i(\psi_1 \,\mathcal{R}\, (\mathsf{Y}^{i-j}\psi_2)) & \text{if } i > j \\ \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1) \,\mathcal{R}\, \psi_2) & \text{otherwise} \end{cases}$$

Canonization - Example



$R_4 : (\mathsf{X}^i\psi_1)\,\mathcal{R}\,(\mathsf{X}^j(\psi_2\,\mathcal{R}\,\psi_3)) \rightsquigarrow$

$$\begin{cases} \mathsf{X}^i(\psi_1\,\mathcal{R}\,((\mathsf{Y}^{i-j}\psi_2)\,\mathcal{R}\,(\mathsf{Y}^{i-j}\psi_3))) & \text{if } i > j \\ \mathsf{X}^j((\mathsf{Y}^{j-i}\psi_1)\,\mathcal{R}\,(\psi_2\,\mathcal{R}\,\psi_3)) & \text{otherwise} \end{cases}$$

$R_5 : G\mathsf{X}^i\mathsf{G}\psi \rightsquigarrow \mathsf{X}^i\mathsf{G}\psi$

$R_6 : G\mathsf{X}^i(\psi_1\,\mathcal{R}\,\psi_2) \rightsquigarrow \mathsf{X}^i\mathsf{G}\psi_2$

$R_7 : (\mathsf{X}^i\psi_1)\,\mathcal{R}\,(\mathsf{X}^j\mathsf{G}\psi_2) \rightsquigarrow$

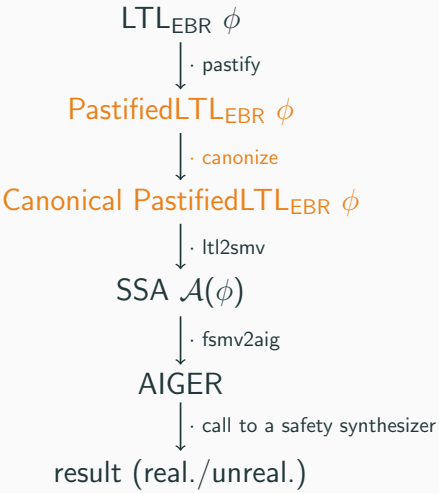
$$\begin{cases} \mathsf{X}^i\mathsf{G}\mathsf{Y}^{i-j}\psi_2 & \text{if } i > j \\ \mathsf{X}^j\mathsf{G}\psi_2 & \text{otherwise} \end{cases}$$

$R_{flat} : \mathsf{X}^i(\psi_1\,\mathcal{R}\,(\psi_2\,\mathcal{R}\,(\ldots(\psi_{n-1}\,\mathcal{R}\,\psi_n)\ldots))) \rightsquigarrow$

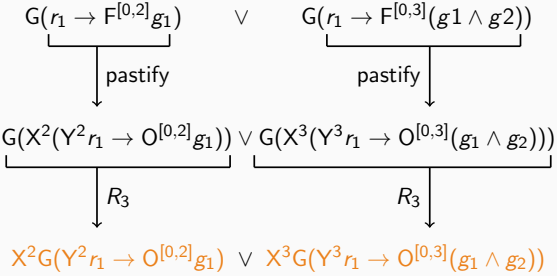
$\mathsf{X}^i((\psi_{n-1} \wedge \mathsf{O}(\psi_{n-2} \wedge \ldots \mathsf{O}(\psi_1 \wedge \mathsf{Y}^i\top)\ldots))\,\mathcal{R}\,\psi_n)$

for any $n \geq 3$

Canonization - Example



Previous example:



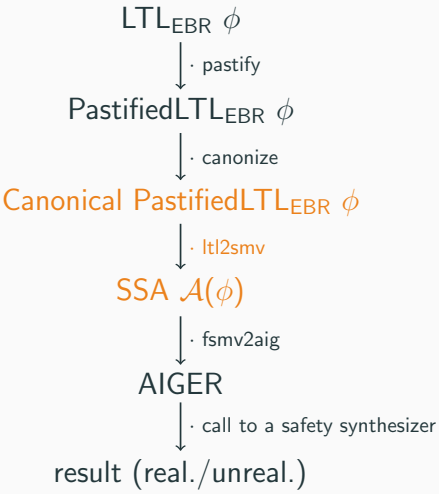
Canonize - Soundness and Complexity

Lemma (Soundness of canonize$(\cdot)$)
For any PastifiedLTL$_{\text{EBR}}$ *formula* ϕ*, it holds that* ϕ *and canonize*(ϕ) *are equivalent and canonize*(ϕ) *is a Canonical* PastifiedLTL$_{\text{EBR}}$ *formula.*

Proposition (Complexity of canonize$(\cdot)$)

For any PastifiedLTL$_{\text{EBR}}$ *formula* ϕ*, canonize*(ϕ) *can be built in* $\mathcal{O}(n)$ *time, and the size of canonize*(ϕ) *is* $\mathcal{O}(n)$*, where* $n = |\phi|$*.*

From Canonical Form to Deterministic Automata

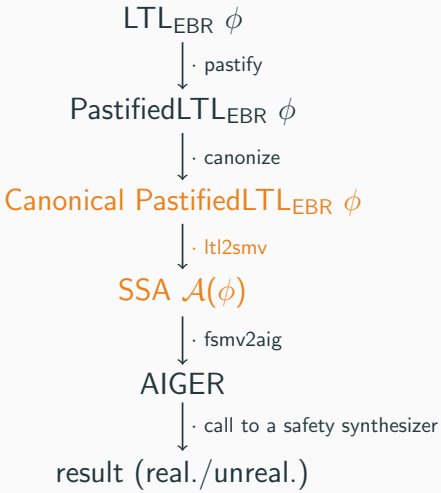


Definition (Symbolic Safety Automata)

A *symbolic safety automaton* (SSA) is a tuple $\mathcal{A} = (V, I, T, S)$, such that

- $V = X \cup \Sigma$, where X is a set of *state variables* and Σ is a set of *input variables*, and
- $I(X)$, $T(X, \Sigma, X')$, and $S(X)$, with $X' = \{x' \mid x \in X\}$, are Boolean formulae which define the set of initial states, the transition relation, and the set of safe states, respectively.

From Canonical Form to Deterministic Automata



Definition (Deterministic SSA)

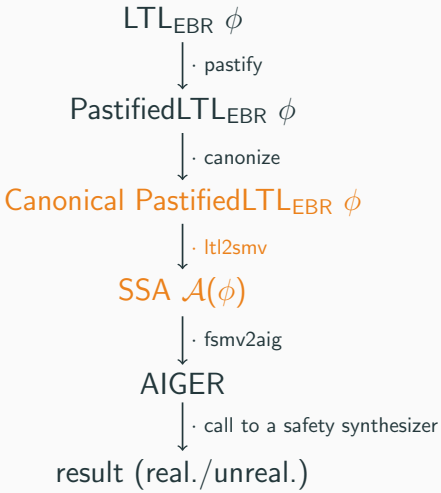
An SSA $\mathcal{A} = (V, I, T, S)$ is *deterministic* if:

- the formula I has exactly one satisfying assignment;
- the transition relation is of the form:

$$T(X, \Sigma, X') := \bigwedge_{x \in X} (x' \leftrightarrow \beta_x(X \cup \Sigma))$$

where each $\beta_x(X \cup \Sigma)$ is a Boolean formula over X and Σ.

From Canonical Form to Deterministic Automata



We use the SMV language to express symbolic automata.

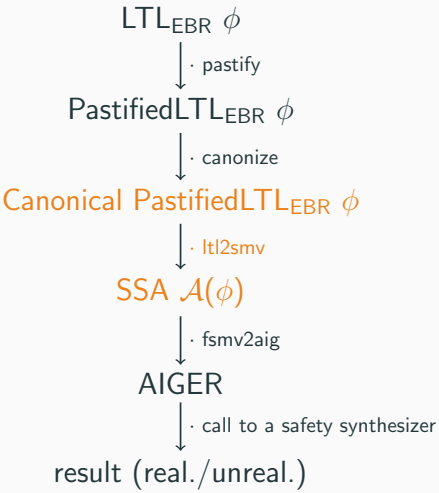
- Past operators: "the past already happened"

$$\begin{aligned}&\texttt{next}(v_{Y\alpha}) \;:=\; v_\alpha \wedge \texttt{counter} > 0\\ &\texttt{DEFINE}\\ &\quad v_{\alpha\mathcal{S}\beta} \;:=\; v_\beta \vee (v_\alpha \wedge v_{Y(\alpha\mathcal{S}\beta)})\end{aligned}$$

- Boolean operators:

$$\begin{aligned}&\texttt{DEFINE}\\ &\quad v_p \;:=\; p\\ &\quad v_{\neg\alpha} \;:=\; \neg v_\alpha\\ &\quad v_{\alpha\vee\beta} \;:=\; v_\alpha \vee v_\beta\end{aligned}$$

From Canonical Form to Deterministic Automata



- Counter:

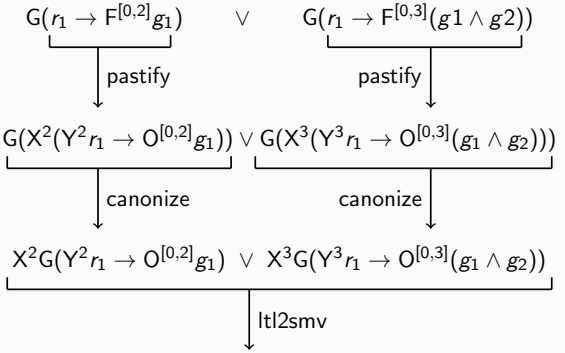
$$\begin{aligned}&\texttt{next}(\texttt{counter}_0) := \neg\ \texttt{counter}_0\\&\texttt{next}(\texttt{counter}_i) := (\texttt{counter}_{i-1} \vee \\&\qquad \texttt{counter}_i) \wedge \neg\texttt{counter}_i\end{aligned}$$

- Globally operator $X^iG\psi$:

$$\begin{aligned}&\texttt{next}(\mathit{error}_{X^iG\psi}) := \texttt{case}\\&\qquad \mathit{counter} < i\ :\ \texttt{FALSE};\\&\qquad \neg\mathit{error}_{X^iG\psi} \wedge v_\psi\ :\ \texttt{FALSE};\\&\qquad \texttt{TRUE}\ :\ \texttt{TRUE};\\&\quad \texttt{esac}\end{aligned}$$

- Release operator: ...

From Canonical Form to Deterministic Automata - Example



To Deterministic Automata - Soundness and Complexity

Proposition

Let ϕ be a canonical PastifiedLTL$_{\text{EBR}}$ *formula, with $|\phi| = n$. Then, there exists a deterministic SSA of size $\mathcal{O}(n)$ that accepts the same language.*

Theorem

Let ϕ be an LTL$_{\text{EBR}}$ *formula, with $|\phi| = n$, and let M be the greatest constant in ϕ. Then, there exists a deterministic SSA of size $\mathcal{O}(n^3 \cdot M^{\log_2 n+1})$ that accepts the same language.*

Solving the game

- Since the organization of the SYNTCOMP, many optimized tools have been proposed in the literature to solve safety games.
  - backward fixpoint algorithms

[a] Bloem, Könighofer, and Seidl, "SAT-based synthesis methods for safe
[b] Biere, Heljanko, and Wieringa, "AIGER 1.9 and beyond".

Solving the game

- Since the organization of the SYNTCOMP, many optimized tools have been proposed in the literature to solve safety games.
  - backward fixpoint algorithms
- We use a safety synthesizer as a black box. We have chosen the SAT-based tool demiurge[a].

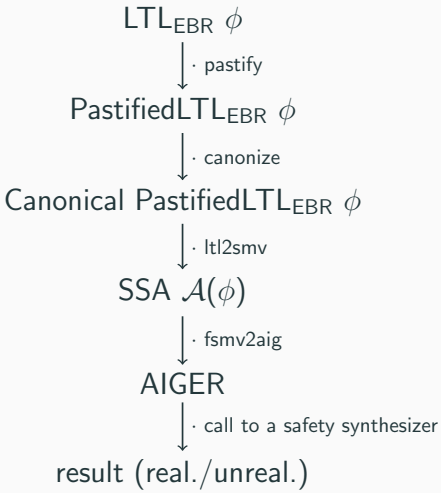
[a] Bloem, Könighofer, and Seidl, "SAT-based synthesis methods for safe
[b] Biere, Heljanko, and Wieringa, "AIGER 1.9 and beyond".

Solving the game

- Since the organization of the SYNTCOMP, many optimized tools have been proposed in the literature to solve safety games.
  - backward fixpoint algorithms
- We use a safety synthesizer as a black box. We have chosen the SAT-based tool demiurge[a].
- The majority of these tools accept as input a symbolic arena described in terms of and-inverter graphs (or AIGER format[b])

[a] Bloem, Könighofer, and Seidl, "SAT-based synthesis methods for safe
[b] Biere, Heljanko, and Wieringa, "AIGER 1.9 and beyond".

Solving the game



Theorem
The realizability problem for $\mathsf{LTL}_{\mathsf{EBR}}$ *belongs to* 2EXPTIME. *If no constant is admitted, it belongs to* EXPTIME.

Experimental Evaluation

- Implementation in a tool called ebr-ltl-synth.
- The transformation from LTL_{EBR} to deterministic SSA together with the translation to AIGER has been implemented inside the nuXmv model checker
- Backend: demiurge.

Experimental Evaluation

- Implementation in a tool called ebr-ltl-synth.
- The transformation from LTL_{EBR} to deterministic SSA together with the translation to AIGER has been implemented inside the nuXmv model checker
- Backend: demiurge.

- Comparison with:
  - ltlsynt: translation to parity games. Full LTL. Uses SPOT.
  - Strix: translation to parity games. Full LTL. Winner of the last two SYNTCOMP.
  - Ssyft: translation to explicit-state DFA. Safety LTL. Uses MONA.
- 800 scalable benchmarks, half realizable and half unrealizable.
- SYNTCOMP benchmarks (syntactically) in LTL_{EBR}:
  - 29 formulas out of 346 ($\sim 8.4\%$).
  - 97% of the SafetyLTL ones.

Experimental Evaluation

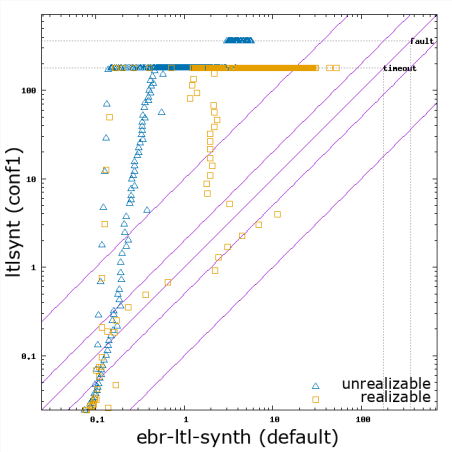


Figure 1: ebr-ltl-synth vs ltlsynt (best conf.) on all scalable benchmarks.

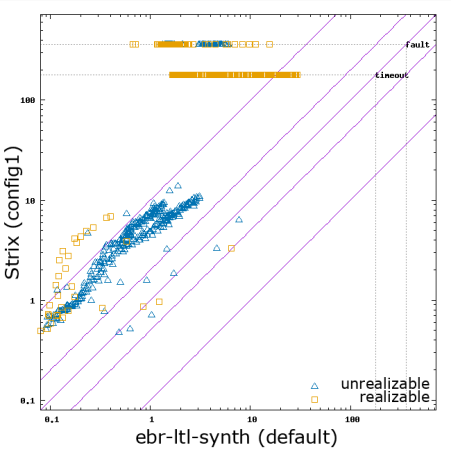


Figure 2: ebr-ltl-synth vs Strix (best conf.) on all scalable benchmarks.

Experimental Evaluation

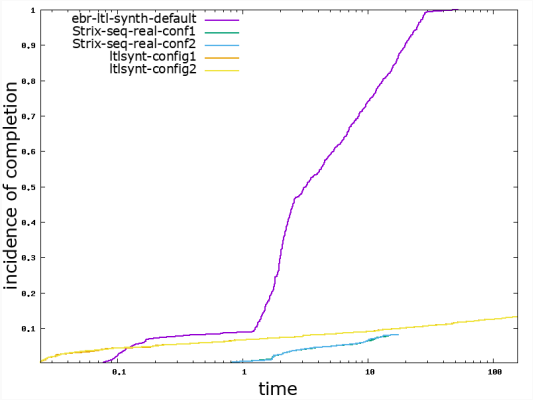


Figure 3: Survival plot for realizable scalable benchmarks.

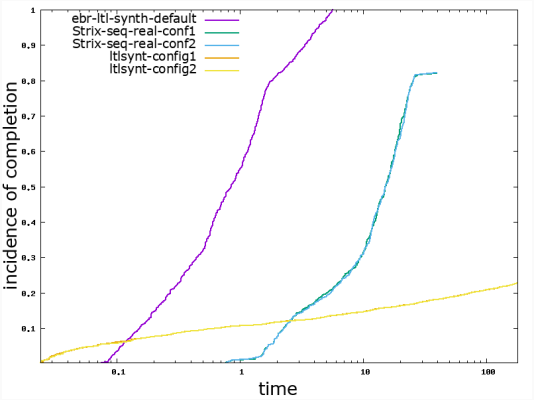


Figure 4: Survival plot for unrealizable scalable benchmarks.

Experimental Evaluation

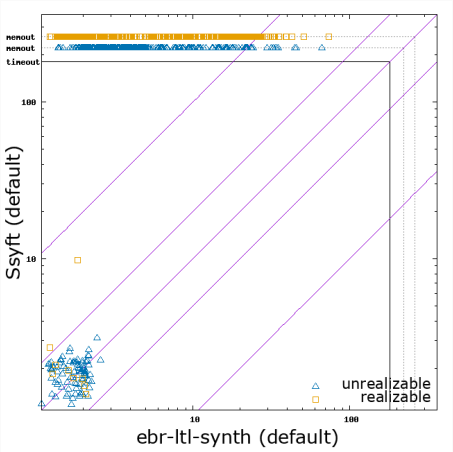


Figure 5: ebr-ltl-synth vs Ssyft on scalable benchmarks.

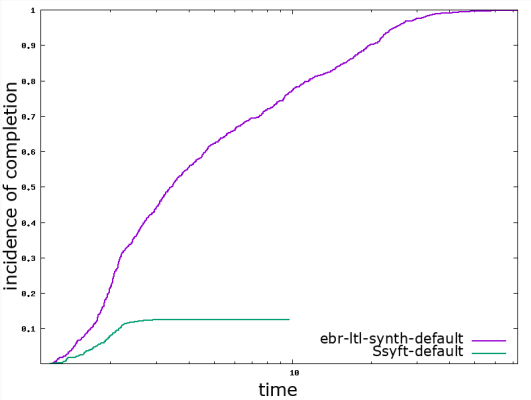


Figure 6: Survival plot for ebr-ltl-synth and Ssyft on scalable benchmarks.

Conclusions

Recap:

- we introduced the logic LTL_{EBR}
- we focused on realizability and reactive synthesis for this logic
- main contribution: fully symbolic translation from any LTL_{EBR} formula to symbolic safety automata.
- complexity: 2EXPTIME in general, but EXPTIME if no constant is used.
- very good performance against the other tools

Future work:

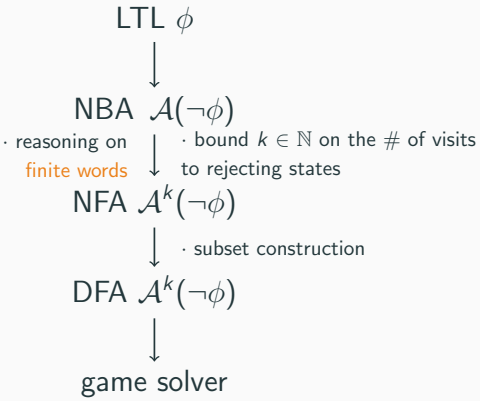
- use LTL_{EBR} for model-checking
- accomodate assumptions (like GR(1))
- use LTL_{EBR} for the safety problems originated from bounded synthesis techniques

Thank You!

Questions?

Appendix: Bounded Synthesis

Bounded Synthesis:



Research mainly focused on two lines

- finding good algorithms for the average case
  - Safraless approaches
    - Bounded synthesis

Appendix: Safety LTL Synthesis

Safety LTL Synthesis:

Research mainly focused on two lines

- finding good algorithms for the average case
  - Safraless approaches
    - Bounded synthesis
- restricting the expressiveness of the specification language
  - GR(1)
  - Safety LTL: only universal operators

Appendix: Release Monitor

Release Monitor Monitor for $\phi := \mathsf{X}^i(\psi_1 \mathcal{R} \psi_2)$:

$\mathtt{next}(error_{\mathsf{X}^i(\psi_1 \mathcal{R} \psi_2)}) := \mathtt{case}$
$\quad counter < i : \mathrm{FALSE};$
$\quad \neg error_{\mathsf{X}^i(\psi_1 \mathcal{R} \psi_2)} \wedge v^i_{\psi_1^p} : \mathrm{FALSE};$
$\quad \neg error_{\mathsf{X}^i(\psi_1 \mathcal{R} \psi_2)} \wedge v_{\psi_1} \wedge v_{\psi_2} : \mathrm{FALSE};$
$\quad \neg error_{\mathsf{X}^i(\psi_1 \mathcal{R} \psi_2)} \wedge v_{\psi_2} : \mathrm{FALSE};$
$\quad \mathrm{TRUE} : \mathrm{TRUE};$
$\mathtt{esac}$

$\mathtt{next}(v^i_{\psi_1^p}) := \mathtt{case}$
$\quad counter < i : \mathrm{FALSE};$
$\quad v_{\psi_1^p} : \mathrm{TRUE};$
$\quad v^i_{\psi_1^p} : \mathrm{TRUE};$
$\quad \mathrm{TRUE} : \mathrm{FALSE};$
$\mathtt{esac}$

Appendix: Experimental Evaluation

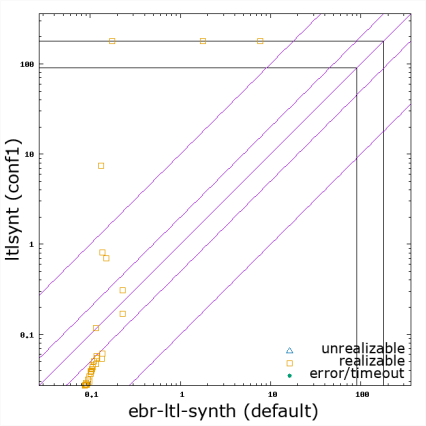


Figure 7: Scatter plot for ebr-ltl-synth and ltlsynt on SYNTCOMP benchmarks.

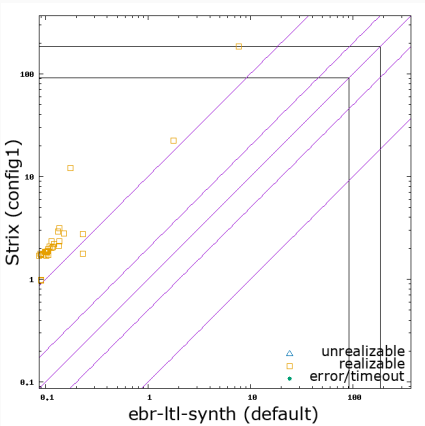


Figure 8: Scatter plot for ebr-ltl-synth and Strix on SYNTCOMP benchmarks.

Appendix: Experimental Evaluation

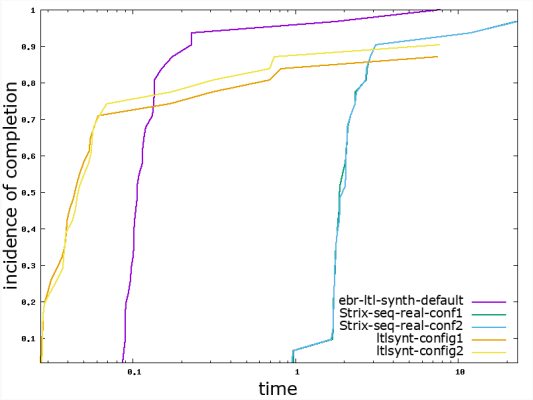


Figure 9: Survival plot for SYNTCOMP benchmarks.

Appendix: Experimental Evaluation

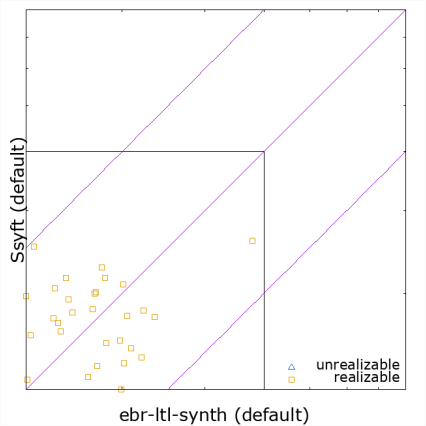


Figure 10: Scatter plot for ebr-ltl-synth and Ssyft on SYNTCOMP benchmarks.

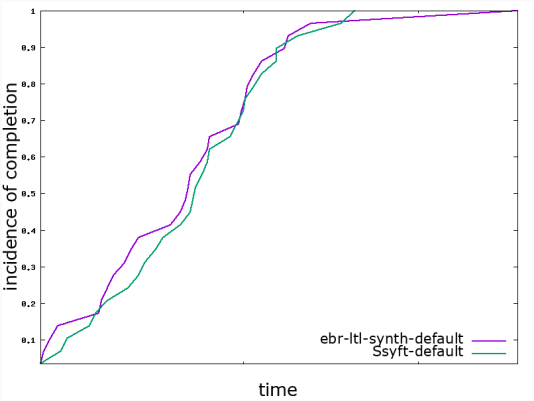


Figure 11: Survival plot for ebr-ltl-synth and Ssyft on SYNTCOMP benchmarks.