Synthesis and Planning Under Environment Specifications

Antonio Di Stasio
distasio@diag.uniroma1.it

University of Rome "La Sapienza"

Reasoning Agents

ERC Advanced Grant
WhiteMech:
White-box Self Programming Mechanisms

SAPIENZA
UNIVERSITÀ DI ROMA

Reactive Synthesis

Given a specification φ over inputs I and outputs O, expressed in:

LTL (Pnueli 1977) or LTL$_f$ (De Giacomo, Vardi 2013)

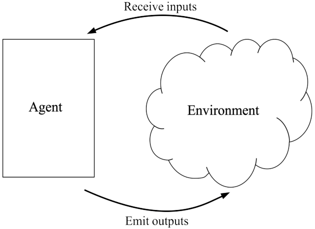
Syntax:

$$\varphi ::= a \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi \mid \Diamond\varphi \mid \Box\varphi$$

Semantic:

A trace *trace* is an infinite (LTL) or finite (LTL$_f$) sequence over I and O. We write *trace* $\models \varphi$ to mean that τ satisfies φ.

Reactive Synthesis



Agent and Environment Strategies, and Traces

For an agent strategy $\sigma_{ag} : I^{+} \rightarrow O$ and an environment strategy $\sigma_{env} : O^{*} \rightarrow I$, the trace

$$trace(\sigma_{ag}, \sigma_{env}) = (i_1 \cup o_1), (i_2 \cup o_2) \ldots \in 2^{I \cup O}$$

denotes the unique trace induced by both σ_{ag} and σ_{env}.

Problem

Given an LTL/ LTL$_f$ task *Goal* for the agent

$$\text{Find agent strategy } \sigma_{ag} \text{ such that } \forall \sigma_{env}.trace(\sigma_{ag}, \sigma_e nv) \models Goal$$

LTL and LTLf Synthesis

Algorithm for LTL synthesis

Given LTL formula φ

1: Compute corresponding NBA (Nondeterministic Buchi Aut.) (exponential)

2: Determinize NBA into DPA (Deterministic Parity Aut.) (exp in states, poly in priorities)

3: Synthesize winning strategy for Parity Game (poly in states, exp in priorities)

Algorithm for LTL_f synthesis

Given LTL_f formula φ

1: Compute corresponding NFA (Nondeterministic Finite Aut.) (exponential)

2: Determinize NFA to DFA (Deterministic Finite Aut.) (exponential)

3: Synthesize winning strategy for DFA game (linear)

Complexity

LTL and LTL_f synthesis are 2EXPTIME-complete

Planning (or Synthesis with a model of the world)

Domain

- Planning consider the agent acting in a **(nondeterministic) domain**
- The domain is a **model of how the world** (i.e. the environment) works
- That is, it is a **specification of the possible environment strategies**

$$[[Dom]] = \{\sigma_{env} | \sigma_{env} \text{ compliant with } Dom\}$$

Planning in nondeterministic domains

Given an LTL_f task *Goal* for the agent, and a domain *Dom* modeling the environment

$$\text{Find agent behavior } \sigma_{ag} \text{ such that } \forall \sigma_{env} \in [[Dom]].trace(\sigma_{ag}\sigma_{env}) \models Goal$$

Specifying possible environment specifications in LTL/LTLf

Environment specifications in LTL/LTL$_f$

Let *Env* be an LTL/LTL$_f$ formula over *I* and *O*.

$$[[Env]] = \{\sigma_{env} | \forall \sigma_{ag}.trace(\sigma_{ag}, \sigma_{env}) \models Env\}$$

i.e *Env* denotes all environment strategies that play according to the specification whatever is the agent strategy.

Synthesis under environment specifications in LTL/LTL$_f$

Given an LTL/LTL$_f$ task *Task* for the agent, and an LTL/LTL$_f$ environment specification *Env*:

$$\text{Find agent strategy } \sigma_{ag} \text{ such that } \forall \sigma_{env} \in [[Env]].trace(\sigma_{ag}, \sigma_{env}) \models Goal$$

Environment specifications in LTL/LTLf

LTL/LTL$_f$ for modeling the environment

1. Nondeterministic planning domains
2. Forms fairness ($\Box\Diamond\phi$) and stability ($\Diamond\Box\phi$) [ZhuDeGiacomoPuVardiAAAI20]
3. Safety properties. [DeGiacomoDiStasioPerelliZhuSubmitted]
4. General LTL environment specifications.[DeGiacomoDiStasioVardiZhuKR2020]
5. ...

Environment specifications in LTL/LTLf

LTL/LTL_f for modeling the environment

1. Nondeterministic planning domains
2. Forms fairness ($\Box\Diamond\phi$) and stability ($\Diamond\Box\phi$) [ZhuDeGiacomoPuVardiAAAI20]
3. Safety properties. [DeGiacomoDiStasioPerelliZhuSubmitted]
4. General LTL environment specifications.[DeGiacomoDiStasioVardiZhuKR2020]
5. ...

Consistent environment specifications

Is any LTL/LTL_f formula a valid environment specification? No, *Env* needs to be "consistent"!:

$$[[Env]] \neq \emptyset \qquad \text{i.e. } \exists\sigma_e.\forall\sigma_{ag}.trace(\sigma_{ag},\sigma_e) \models Env$$

Synthesis Under Environment Specifications

Environment Specifications

Let *Env* be an LTL/LTL$_f$ formula over $I \cup O$.

$$[[Env]] = \{\sigma_{env} | \sigma_{env} \text{ satisfies } Env \text{ whatever is the agent strategy}\}$$

Synthesis with environment specifications in LTL/LTL$_f$

Given an LTL/ LTL$_f$ task *Goal* for the agent, and an LTL/LTL$_f$ environment specification *Env*:

$$\text{Find agent strategy } \sigma_{ag} \text{ such that } \forall \sigma_{env} \in [[Env]].trace(\sigma_{ag}, \sigma_{env}) \models Goal$$

Theorem [AminofDeGiacomoMuranoRubinICAPS2019]

To find agent strategy realizing *Goal* under the environment specification *Env*, we can use standard LTL/LTL$_f$ synthesis for

$$Env \rightarrow Goal$$

LTL_f Synthesis Under Safety Environment Specifications

Safety Properties

Definition

A safety property is a property which specifies that some (bad) behavior will never occur.

Examples:

- "always at most one process is in its critical section"
- "money can only be withdrawn once a correct PIN has been provided"

Important property

Any infinite trace violating the property has a finite prefix that is "bad";

- ... two processes are in the critical section ...

Usually: $\Box \neg \dots$

Safety Properties

Consider a language $\mathcal{L} \subseteq \Sigma^\omega$.

Bad Prefix

A finite word $x \in \Sigma^*$ is a **bad prefix** for $\mathcal{L}$ if and only if for all infinite words $y \in \Sigma^\omega$, we have $x \cdot y \notin \mathcal{L}$.

Safety Languages and Formulas

A language $\mathcal{L}$ is a **safety language** iff every $w \notin \mathcal{L}$ has a bad prefix. A formula φ is a **safety formula** iff $\mathcal{L}(\varphi) = \{w \in \Sigma^\omega \mid w \models \varphi\}$ is a safety language.

Safety Properties

Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be a NBA, we define its looping automaton $\mathcal{A}^{loop} = (\Sigma, Q \cup \{q_{sink}\}, q_0, \delta, Q)$, i.e., the automaton defined as $\mathcal{A}$ in which every state has been made into an accepting state.

Theorem [KuperfmanVardiCAV99]

$\mathcal{A}$ specifies a safety formula if and only if $\mathcal{L}(\mathcal{A}) = L(\mathcal{A}^{loop})$.

Properties of looping automata:

1. Looping automata recognize safety languages.
2. Looping automata can be determinized by using **the standard subset construction**.

LTLf under Safety Environment Specifications

Problem

Solve the synthesis problem for

$$\mathit{Env} \rightarrow \mathit{Goal}$$

where *Env* is a safety environment specification expressed in LTL and *Goal* is an LTL_f formula.

Naive Solution

Translate to LTL and then do standard LTL synthesis for $\mathit{Env} \rightarrow \mathit{Goal}$.

... we can do better, since:

- *Env*: Safety

LTLf under Safety Environment Specifications

Problem

Solve the synthesis problem for

$$Env \rightarrow Goal$$

where *Env* is a safety environment specification expressed in LTL and *Goal* is an LTL_f formula.

Naive Solution

Translate to LTL and then do standard LTL synthesis for $Env \rightarrow Goal$.

... we can do better, since:

- *Env*: Safety
- *Goal*: LTL_f

1 ° Stage

1. Build the corresponding deterministic looping automaton $\mathcal{D}_{Env} = (\Sigma, S \cup \{s_{sink}\}, s_0, \delta, S)$ of Env.

1 ° Stage

1. Build the corresponding deterministic looping automaton $\mathcal{D}_{Env} = (\Sigma, S \cup \{s_{sink}\}, s_0, \delta, S)$ of Env.
2. Solve the safety game for the environment over $\mathcal{D}_{Env}$:

$$Safe_{env} = \nu Z.(S \cap Pre_{env}(Z))$$
$$Pre_{env}(Z) = \{s | \exists X \forall Y. \delta(s, X \cup Y) \in Z\}$$

LTLf under Safety Environment Specifications

1 ° Stage

1. Build the corresponding deterministic looping automaton $\mathcal{D}_{Env} = (\Sigma, S \cup \{s_{sink}\}, s_0, \delta, S)$ of Env.
2. Solve the safety game for the environment over $\mathcal{D}_{Env}$:

$$Safe_{env} = \nu Z.(S \cap Pre_{env}(Z))$$
$$Pre_{env}(Z) = \{s | \exists X \forall Y. \delta(s, X \cup Y) \in Z\}$$

3. Remove the winning region for the agent from $\mathcal{D}_{Env}$, say $\mathcal{D}'_{Env}$.

LTLf under Safety Environment Specifications

2 ° Stage

1. Build the corresponding DFA $\mathcal{A}_{Goal}$

2 ° Stage

1. Build the corresponding DFA $\mathcal{A}_{Goal}$
2. Do the cartesian product $\mathcal{G} = \mathcal{D}'_{Env} \times \mathcal{A}_{Goal}$.

LTLf under Safety Environment Specifications

2 ° Stage

1. Build the corresponding DFA $\mathcal{A}_{Goal}$
2. Do the cartesian product $\mathcal{G} = \mathcal{D}'_{Env} \times \mathcal{A}_{Goal}$.
3. Solve the reachability game for the agent over $\mathcal{G}$.

2 ° Stage

1. Build the corresponding DFA $\mathcal{A}_{Goal}$
2. Do the cartesian product $\mathcal{G} = \mathcal{D}'_{Env} \times \mathcal{A}_{Goal}$.
3. Solve the reachability game for the agent over $\mathcal{G}$.
4. Return the winning strategy for the agent if one exists.

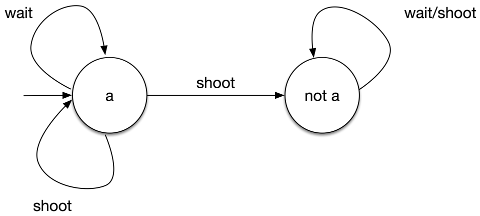
LTL_f Synthesis Under General LTL Environment Specifications

LTLf Synthesis Under LTL Environment Specifications

For example let the environment specifications be formed by $Env_1 \wedge Env_2$ where:

Env_1 is the LTL formula expressing the dynamics of the environment (as a planning domain):

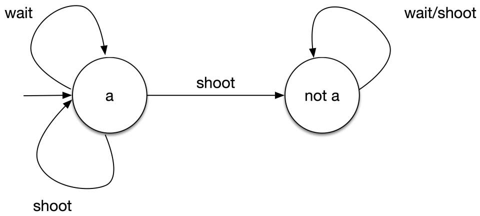
- $\Box(alive \rightarrow \bigcirc(wait \rightarrow alive))$
- $\Box(alive \rightarrow \bigcirc(shoot \rightarrow (alive \vee \neg alive)))$
- $\Box(\neg alive \rightarrow \bigcirc(wait \rightarrow \neg alive))$
- $\Box(\neg alive \rightarrow \bigcirc(shoot \rightarrow \neg alive))$
- $\Box((wait \wedge shoot) \wedge (wait \rightarrow \neg shoot))$



For example let the environment specifications be formed by $Env_1 \wedge Env_2$ where:

Env_1 is the LTL formula expressing the dynamics of the environment (as a planning domain):

- $\Box(alive \to \bigcirc(wait \to alive))$
- $\Box(alive \to \bigcirc(shoot \to (alive \vee \neg alive))$
- $\Box(\neg alive \to \bigcirc(wait \to \neg alive))$
- $\Box(\neg alive \to \bigcirc(shoot \to \neg alive))$
- $\Box((wait \wedge shoot) \wedge (wait \to \neg shoot))$



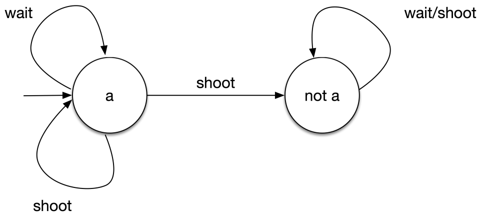
Env_2 is the LTL formula expressing some fairness over nondeterministic effects, e.g.,

$$\Box\Diamond shoot \to \Diamond\neg a$$

For example let the environment specifications be formed by $Env_1 \wedge Env_2$ where:

Env_1 is the LTL formula expressing the dynamics of the environment (as a planning domain):

- $\Box(alive \rightarrow \bigcirc(wait \rightarrow alive))$
- $\Box(alive \rightarrow \bigcirc(shoot \rightarrow (alive \vee \neg alive))$
- $\Box(\neg alive \rightarrow \bigcirc(wait \rightarrow \neg alive))$
- $\Box(\neg alive \rightarrow \bigcirc(shoot \rightarrow \neg alive))$
- $\Box((wait \wedge shoot) \wedge (wait \rightarrow \neg shoot))$



Env_2 is the LTL formula expressing some fairness over nondeterministic effects, e.g.,

$$\Box\Diamond shoot \rightarrow \Diamond\neg a$$

Let $Goal$ be an LTL_f formula which expresses an agent task, e.g.,

$$\Diamond\neg a$$

Problem

Solve the synthesis problem for

$$Env_1 \wedge Env_2 \rightarrow Goal$$

LTLf Synthesis Under LTL Environment Specifications

Problem

Solve the synthesis problem for

$$Env_1 \wedge Env_2 \rightarrow Goal$$

Naive Solution

Translate to LTL and then do standard LTL synthesis for $Env_1 \wedge Env_2 \rightarrow Goal$.

LTLf Synthesis Under LTL Environment Specifications

Problem

Solve the synthesis problem for

$$Env_1 \wedge Env_2 \rightarrow Goal$$

Naive Solution

Translate to LTL and then do standard LTL synthesis for $Env_1 \wedge Env_2 \rightarrow Goal$.

... but we can exploit the simplicity of dealing with LTL_f given:

LTLf Synthesis Under LTL Environment Specifications

Problem

Solve the synthesis problem for

$$Env_1 \wedge Env_2 \rightarrow Goal$$

Naive Solution

Translate to LTL and then do standard LTL synthesis for $Env_1 \wedge Env_2 \rightarrow Goal$.

... but we can exploit the simplicity of dealing with LTL_f given:

- Env_1: LTL

LTLf Synthesis Under LTL Environment Specifications

Problem

Solve the synthesis problem for

$$Env_1 \wedge Env_2 \rightarrow Goal$$

Naive Solution

Translate to LTL and then do standard LTL synthesis for $Env_1 \wedge Env_2 \rightarrow Goal$.

... but we can exploit the simplicity of dealing with LTL_f given:

- Env_1: $\cancel{\text{LTL}} \rightarrow \text{LTL}_f$

LTLf Synthesis Under LTL Environment Specifications

Problem

Solve the synthesis problem for

$$Env_1 \wedge Env_2 \rightarrow Goal$$

Naive Solution

Translate to LTL and then do standard LTL synthesis for $Env_1 \wedge Env_2 \rightarrow Goal$.

... but we can exploit the simplicity of dealing with LTL_f given:

- Env_1: ~~LTL~~ $\rightarrow \text{LTL}_f$
- Env_2: LTL

LTLf Synthesis Under LTL Environment Specifications

Problem

Solve the synthesis problem for

$$Env_1 \wedge Env_2 \rightarrow Goal$$

Naive Solution

Translate to LTL and then do standard LTL synthesis for $Env_1 \wedge Env_2 \rightarrow Goal$.

... but we can exploit the simplicity of dealing with LTL_f given:

- Env_1: ~~LTL~~ $\rightarrow$ LTL_f
- Env_2: LTL
- $Goal$: LTL_f

LTLf Synthesis Under LTL Environment Specifications

Separating LTL_f environment specifications

$$(Env_1 \wedge Env_2 \rightarrow Goal) \iff (Env_2 \rightarrow Env_1 \rightarrow Goal) \iff (Env_2 \rightarrow \neg Env_1 \vee Goal)$$

where $Goal' = \neg Env_1 \vee Goal$ is expressed in LTL_f and Env_2 in LTL.

LTLf Synthesis Under LTL Environment Specifications

Separating LTL_f environment specifications

$$(Env_1 \wedge Env_2 \rightarrow Goal) \iff (Env_2 \rightarrow Env_1 \rightarrow Goal) \iff (Env_2 \rightarrow \neg Env_1 \vee Goal)$$

where $Goal' = \neg Env_1 \vee Goal$ is expressed in LTL_f and Env_2 in LTL.

Problem

Solve the synthesis problem for

$$Env_2 \rightarrow Goal'$$

LTLf Synthesis Under LTL Environment Specifications

Separating LTL_f environment specifications

$$(Env_1 \wedge Env_2 \rightarrow Goal) \iff (Env_2 \rightarrow Env_1 \rightarrow Goal) \iff (Env_2 \rightarrow \neg Env_1 \vee Goal)$$

where $Goal' = \neg Env_1 \vee Goal$ is expressed in LTL_f and Env_2 in LTL.

Problem

Solve the synthesis problem for

$$Env_2 \rightarrow Goal'$$

How can we exploit that $Goal'$ is LTL_f?

LTLf Synthesis Under LTL Environment Specifications

Separating LTL_f environment specifications

$$(Env_1 \wedge Env_2 \rightarrow Goal) \iff (Env_2 \rightarrow Env_1 \rightarrow Goal) \iff (Env_2 \rightarrow \neg Env_1 \vee Goal)$$

where $Goal' = \neg Env_1 \vee Goal$ is expressed in LTL_f and Env_2 in LTL.

Problem

Solve the synthesis problem for

$$Env_2 \rightarrow Goal'$$

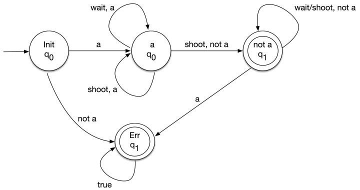
How can we exploit that $Goal'$ is LTL_f?

Two-stage technique!

Two-Stage Technique for Synthesis

1 ° Stage

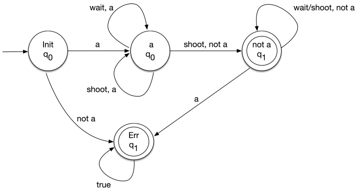
1. Compute the corresponding DFA $\mathcal{A}$ of
   $\neg Env_1 \vee Goal$.



Two-Stage Technique for Synthesis

1 ° Stage

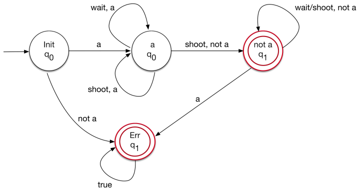
1. Compute the corresponding DFA $\mathcal{A}$ of $\neg Env_1 \vee Goal$.
2. Solve the reachability game for the agent over $\mathcal{A}$.



Two-Stage Technique for Synthesis

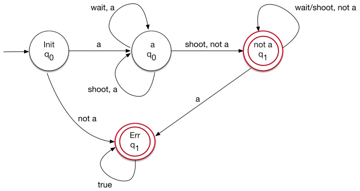
1 ° Stage

1. Compute the corresponding DFA $\mathcal{A}$ of $\neg Env_1 \vee Goal$.
2. Solve the reachability game for the agent over $\mathcal{A}$.
3. Check whether the initial state is winning for the agent.



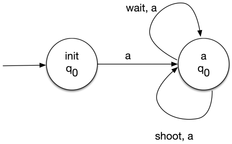
1 ° Stage

1. Compute the corresponding DFA $\mathcal{A}$ of $\neg Env_1 \vee Goal$.
2. Solve the reachability game for the agent over $\mathcal{A}$.
3. Check whether the initial state is winning for the agent.
4. If the initial state is not winning go to Stage 2, otherwise return the agent winning strategy.



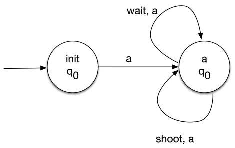
2 ° Stage

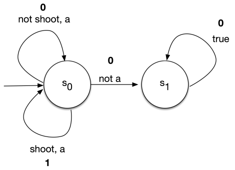
1. Remove from $\mathcal{A}$ the agent winning set of Stage 1, say $\mathcal{A}'$.



2 ° Stage

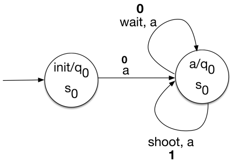
1. Remove from $\mathcal{A}$ the agent winning set of Stage 1, say $\mathcal{A}'$.
2. Compute the corresponding DPA $\mathcal{B}$ of Env_2.





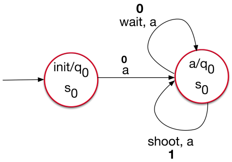
2° Stage

1. Remove from $\mathcal{A}$ the agent winning set of Stage 1, say $\mathcal{A}'$.
2. Compute the corresponding DPA $\mathcal{B}$ of Env_2.
3. Do the cartesian product between $\mathcal{A}'$ and $\mathcal{B}$.



2 ° Stage

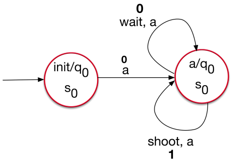
1. Remove from $\mathcal{A}$ the agent winning set of Stage 1, say $\mathcal{A}'$.
2. Compute the corresponding DPA $\mathcal{B}$ of Env_2.
3. Do the cartesian product between $\mathcal{A}'$ and $\mathcal{B}$.
4. Solve the parity game for the environment over $\mathcal{A}' \times \mathcal{B}$.



Two-Stage Technique for Synthesis

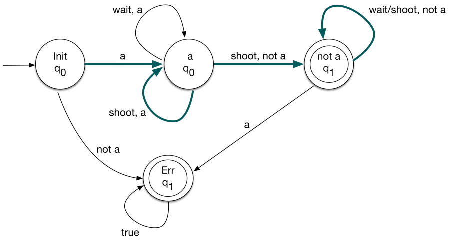
2 ° Stage

1. Remove from $\mathcal{A}$ the agent winning set of Stage 1, say $\mathcal{A}'$.
2. Compute the corresponding DPA $\mathcal{B}$ of *Env*$_2$.
3. Do the cartesian product between $\mathcal{A}'$ and $\mathcal{B}$.
4. Solve the parity game for the environment over $\mathcal{A}' \times \mathcal{B}$.
5. Check if the initial state is winning for the agent; if not return "Unrealizable".



2 ° Stage

1. Remove from $\mathcal{A}$ the agent winning set of Stage 1, say $\mathcal{A}'$.
2. Compute the corresponding DPA $\mathcal{B}$ of Env_2.
3. Do the cartesian product between $\mathcal{A}'$ and $\mathcal{B}$.
4. Solve the parity game for the environment over $\mathcal{A}' \times \mathcal{B}$.
5. Check if the initial state is winning for the agent; if not return "Unrealizable".
6. Return the agent winning strategy by combing the agent winning strategies in Stage 1 and 2.



Experimental Analysis

We have

- implemented the two-stage technique in a new tool called **2SLS**, written in C++, that exploits CUDD package as library for the manipulation of Binary Decisions Diagrams (BDDs);
- compared **2SLS** to a direct reduction to LTL synthesis by employing the LTLf -to-LTL translator **SPOT** and **Strix** (Meyer, Sickert, and Luttenberger 2018) as the LTL synthesis solver;
- compared **2SLS** with FSyft and StSyft (Zhu et al. 2020) in special cases where environment specifications are LTL formulas of the form $\Box\Diamond a$ (fairness) and $\Diamond\Box a$ (stability), with a propositional.

Experiments on Fairness and Stability

- Given a counter game where the environment chooses whether to increment the counter or not and the agent can choose to grant the request or ignore it;
- The fairness specification is $\Box\Diamond$*increment*; the stability specification is $\Diamond\Box$*increment*;
- The goal is to get the counter having all bits set to 1.

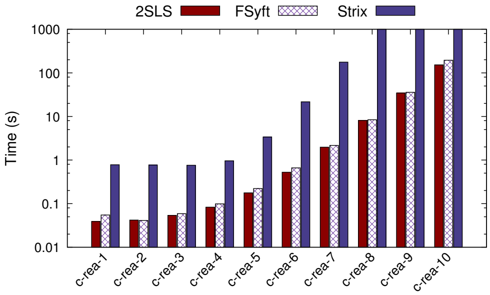


Figure 1: LTL_f synthesis under fairness specifications.

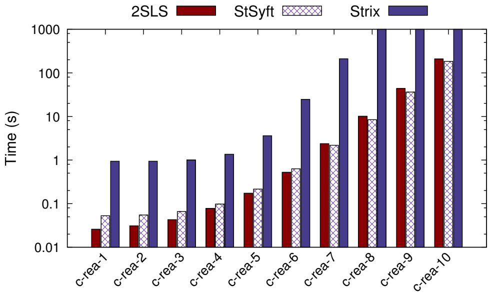


Figure 2: LTL_f synthesis under stability specifications.

Experiments of General LTL Environment Specifications

- Given *Goal* as a conjunction of increasing size of random LTL_f formulas of the form $\Box(p_j \rightarrow \Diamond q_j)$ with p_j and q_j propositions under the control of the environment and the agent, respectively;
- *Env* is a conjunction of formulas of the form $(\Box\Diamond p_i \vee \Diamond\Box q_i)$, where we start with one conjunct and introduce a new conjunct every 10 conjuncts in *Goal*.

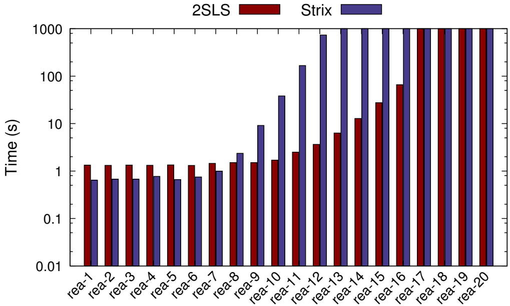


Figure 3: LTL_f synthesis under general LTL environment specifications.

Project

Develop $\mathtt{LTL}_f$ synthesis under safety environment specifications where you can use:

- Spot (spot.lrde.epita.fr) for building the looping determinstic automaton
- LTLf2DFA (ltlf2dfa.diag.uniroma1.it) for constructing the DFA.