Distributed PROMPT-LTL Synthesis

Joint work with Swen Jacobs and Leander Tentrup (Saarland University)

Martin Zimmermann

Saarland University

September 16th, 2016

GandALF 2016, Catania, Italy

Motivation

- LTL is the standard language for the specification of reactive systems...
- but it cannot express timing constraints, e.g., every request is answered within a bounded amount of time.

Motivation

- LTL is the standard language for the specification of reactive systems...
- but it cannot express timing constraints, e.g., every request is answered within a bounded amount of time.
- PROMPT–LTL is able to express such properties.

Theorem (Kupferman et al. '07)

PROMPT–LTL *model checking (synthesis) is as hard as* LTL *model checking (synthesis).*

Motivation

- LTL is the standard language for the specification of reactive systems...
- but it cannot express timing constraints, e.g., every request is answered within a bounded amount of time.
- PROMPT–LTL is able to express such properties.

Theorem (Kupferman et al. '07)

PROMPT–LTL *model checking (synthesis) is as hard as* LTL *model checking (synthesis).*

Note: The synthesis result requires a perfect information setting!

Motivation

- LTL is the standard language for the specification of reactive systems...
- but it cannot express timing constraints, e.g., every request is answered within a bounded amount of time.
- PROMPT–LTL is able to express such properties.

Theorem (Kupferman et al. '07)

PROMPT–LTL *model checking (synthesis) is as hard as* LTL *model checking (synthesis).*

Note: The synthesis result requires a perfect information setting!

Here: synthesis of distributed systems, i.e., multiple components with imperfect information.

Outline

1. Definitions
PROMPT-LTL
Distributed Synthesis
The Alternating Color Technique

2. The Synchronous Case

3. The Asynchronous Case

4. Conclusion

PROMPT-LTL

Syntax:

$$\varphi ::= a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \varphi\,\mathbf{R}\,\varphi \mid \mathbf{F_P}\,\varphi$$

where a ranges over a finite set AP of atomic propositions.

PROMPT-LTL

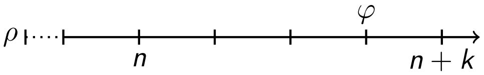
Syntax:

$$\varphi ::= a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \varphi\,\mathbf{R}\,\varphi \mid \mathbf{F_P}\,\varphi$$

where a ranges over a finite set AP of atomic propositions.

Semantics: defined with respect to a fixed bound $k \in \mathbb{N}$

$(\rho, n, k) \models \mathbf{F_P}\,\varphi$:



PROMPT-LTL

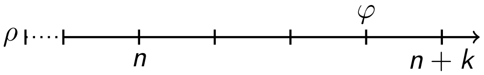
Syntax:

$$\varphi ::= a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \varphi\,\mathbf{R}\,\varphi \mid \mathbf{F_P}\,\varphi$$

where a ranges over a finite set AP of atomic propositions.

Semantics: defined with respect to a fixed bound $k \in \mathbb{N}$

$(\rho, n, k) \models \mathbf{F_P}\,\varphi$:



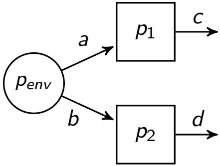
Example: $\mathbf{G}(q \rightarrow \mathbf{F_P}\,p)$ w.r.t. bound k: every request q is answered by response p within k steps.

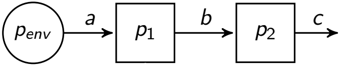
Distributed Synthesis

An architecture consists of

- a finite set P of processes with an environment process p_{env},
- for all $p \in P$ a set $O_p \subseteq \mathrm{AP}$ of outputs (pairwise disjoint), and
- for all $p \in P \setminus \{p_{env}\}$ a set $I_p \subseteq \mathrm{AP}$ of inputs.

Examples:





Distributed Synthesis

An architecture consists of

- a finite set P of processes with an environment process p_{env},
- for all $p \in P$ a set $O_p \subseteq \mathrm{AP}$ of outputs (pairwise disjoint), and
- for all $p \in P \setminus \{p_{env}\}$ a set $I_p \subseteq \mathrm{AP}$ of inputs.

An implementation of a process $p \neq p_{env}$ is a finite transducer computing a function $f_p \colon (2^{I_p})^\omega \to (2^{O_p})^\omega$.

Distributed Synthesis

An architecture consists of

- a finite set P of processes with an environment process p_{env},
- for all $p \in P$ a set $O_p \subseteq \mathrm{AP}$ of outputs (pairwise disjoint), and
- for all $p \in P \setminus \{p_{env}\}$ a set $I_p \subseteq \mathrm{AP}$ of inputs.

An implementation of a process $p \neq p_{env}$ is a finite transducer computing a function $f_p \colon (2^{I_p})^\omega \to (2^{O_p})^\omega$.

The PROMPT–LTL distributed realizability problem for a fixed architecture $\mathcal{A}$ asks, given a PROMPT–LTL formula φ, to decide whether implementations f_p for every $p \neq p_{env}$ and a bound k exist s.t. every outcome $w \in \bigoplus_p f_p$ satisfies φ w.r.t. k.

Distributed Synthesis

An architecture consists of

- a finite set P of processes with an environment process p_{env},
- for all $p \in P$ a set $O_p \subseteq \mathrm{AP}$ of outputs (pairwise disjoint), and
- for all $p \in P \setminus \{p_{env}\}$ a set $I_p \subseteq \mathrm{AP}$ of inputs.

An implementation of a process $p \neq p_{env}$ is a finite transducer computing a function $f_p \colon (2^{I_p})^\omega \to (2^{O_p})^\omega$.

The PROMPT–LTL distributed realizability problem for a fixed architecture $\mathcal{A}$ asks, given a PROMPT–LTL formula φ, to decide whether implementations f_p for every $p \neq p_{env}$ and a bound k exist s.t. every outcome $w \in \bigoplus_p f_p$ satisfies φ w.r.t. k.

Synthesis: compute such f_p, if they exist.

The Alternating Color Technique

1. Add fresh proposition $r \notin \mathrm{AP}$: think of a coloring.
2. Obtain $\mathit{rel}(\varphi)$ by replacing each subformula $\mathbf{F_P}\,\psi$ of φ by

$$(r \rightarrow (r \,\mathbf{U}\, (\neg r \,\mathbf{U}\, \mathit{rel}(\psi)))) \wedge (\neg r \rightarrow (\neg r \,\mathbf{U}\, (r \,\mathbf{U}\, \mathit{rel}(\psi)))).$$

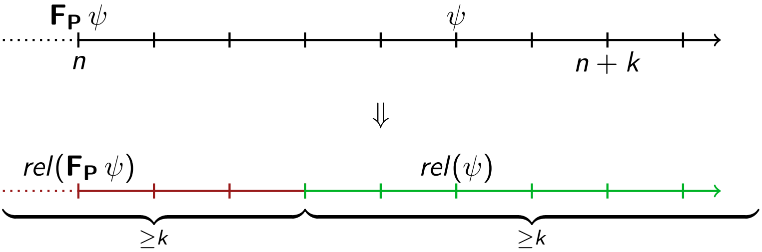
Intuitively: ψ has to be satisfied within one color change.

The Alternating Color Technique

1. Add fresh proposition $r \notin \mathrm{AP}$: think of a coloring.
2. Obtain $\mathit{rel}(\varphi)$ by replacing each subformula $\mathbf{F_P}\,\psi$ of φ by

$$(r \rightarrow (r\ \mathbf{U}\ (\neg r\ \mathbf{U}\ \mathit{rel}(\psi)))) \wedge (\neg r \rightarrow (\neg r\ \mathbf{U}\ (r\ \mathbf{U}\ \mathit{rel}(\psi)))).$$

Intuitively: ψ has to be satisfied within one color change.

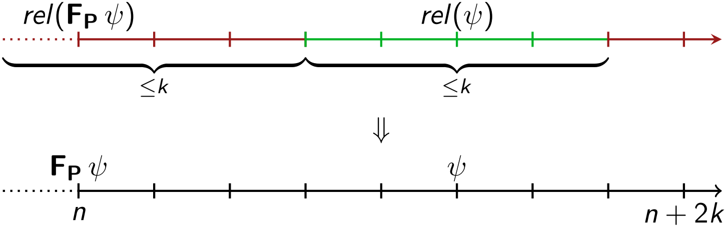


The Alternating Color Technique

1. Add fresh proposition $r \notin \mathrm{AP}$: think of a coloring.
2. Obtain $\mathit{rel}(\varphi)$ by replacing each subformula $\mathbf{F_P}\,\psi$ of φ by

$$(r \rightarrow (r\,\mathbf{U}\,(\neg r\,\mathbf{U}\,\mathit{rel}(\psi)))) \wedge (\neg r \rightarrow (\neg r\,\mathbf{U}\,(r\,\mathbf{U}\,\mathit{rel}(\psi)))).$$

Intuitively: ψ has to be satisfied within one color change.



The Alternating Color Technique

1. Add fresh proposition $r \notin \mathrm{AP}$: think of a coloring.
2. Obtain $rel(\varphi)$ by replacing each subformula $\mathbf{F_P}\,\psi$ of φ by

$$(r \rightarrow (r\,\mathbf{U}\,(\neg r\,\mathbf{U}\,rel(\psi)))) \wedge (\neg r \rightarrow (\neg r\,\mathbf{U}\,(r\,\mathbf{U}\,rel(\psi)))).$$

   Intuitively: ψ has to be satisfied within one color change.

Lemma (Kupferman et al. '07)

Let φ be a PROMPT–LTL formula, $w \in (2^{\mathrm{AP}})^\omega$, and $w' \in (2^{\mathrm{AP}\cup\{r\}})^\omega$ s.t. w and w' coincide on P at every position.

1. *If $(w,k) \models \varphi$ and distance between color changes is at least k in w', then $w' \models rel(\varphi)$.*
2. *Let $k \in \mathbb{N}$. If $w' \models rel(\varphi)$ and distance between color-changes is at most k in w', then $(w,2k) \models \varphi$.*

Outline

1. **Definitions**
   PROMPT-LTL
   Distributed Synthesis
   The Alternating Color Technique

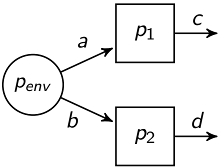
2. **The Synchronous Case**

3. **The Asynchronous Case**

4. **Conclusion**

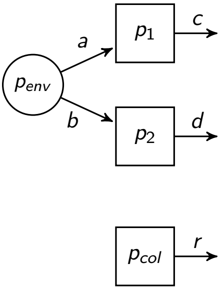
The Synchronous Case

Given architecture $\mathcal{A}$, let $\mathcal{A}^r$ be $\mathcal{A}$ with a new input-free (coloring) process p_{col} that outputs r.



The Synchronous Case

Given architecture $\mathcal{A}$, let $\mathcal{A}^r$ be $\mathcal{A}$ with a new input-free (coloring) process p_{col} that outputs r.



The Synchronous Case

Given architecture $\mathcal{A}$, let $\mathcal{A}^r$ be $\mathcal{A}$ with a new input-free (coloring) process p_{col} that outputs r.

Theorem

A PROMPT–LTL *formula* φ *is realizable in* $\mathcal{A}$ *if, and only if,* $rel(\varphi) \wedge \mathbf{G}\,\mathbf{F}\, r \wedge \mathbf{G}\,\mathbf{F}\, \neg r$ *is realizable in* $\mathcal{A}^r$.

The Synchronous Case

Given architecture $\mathcal{A}$, let $\mathcal{A}^r$ be $\mathcal{A}$ with a new input-free (coloring) process p_{col} that outputs r.

Theorem

A PROMPT–LTL *formula* φ *is realizable in* $\mathcal{A}$ *if, and only if,* $rel(\varphi) \wedge \mathbf{G}\,\mathbf{F}\, r \wedge \mathbf{G}\,\mathbf{F}\, \neg r$ *is realizable in* $\mathcal{A}^r$*.*

Proof Idea:

- Let φ be realizable in $\mathcal{A}$ with bound k by implementations f_p.
- Add the implementation producing $(\emptyset^k\{r\}^k)$ for p_{col} in $\mathcal{A}^r$.
- Every outcome in $\mathcal{A}^r$ coincides on P with an outcome in $\mathcal{A}$.
- So, the implementations realize $rel(\varphi) \wedge \mathbf{G}\,\mathbf{F}\, r \wedge \mathbf{G}\,\mathbf{F}\, \neg r$ in $\mathcal{A}^r$.

The Synchronous Case

Given architecture $\mathcal{A}$, let $\mathcal{A}^r$ be $\mathcal{A}$ with a new input-free (coloring) process p_{col} that outputs r.

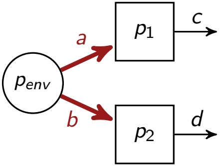
Theorem

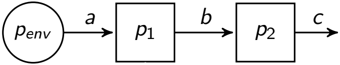
A PROMPT–LTL *formula* φ *is realizable in* $\mathcal{A}$ *if, and only if,* $rel(\varphi) \wedge \mathbf{G}\,\mathbf{F}\, r \wedge \mathbf{G}\,\mathbf{F}\, \neg r$ *is realizable in* $\mathcal{A}^r$*.*

Proof Idea:

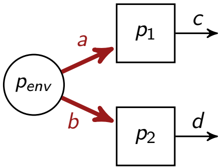
- Let $rel(\varphi) \wedge \mathbf{G}\,\mathbf{F}\, r \wedge \mathbf{G}\,\mathbf{F}\, \neg r$ be realizable in $\mathcal{A}^r$ by implementations f_p.
- As the implementation for p_{col} is finite-state, there is a bound k on the distance between color changes.
- Thus, the implementations also realize φ in $\mathcal{A}$ with bound $2k$.

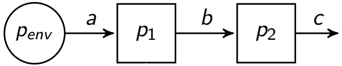
Information Forks





Information Forks

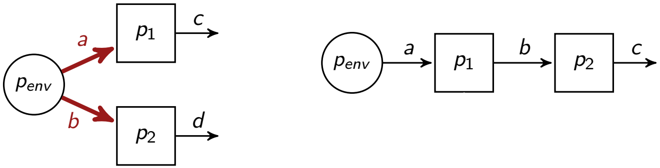




Theorem (Finkbeiner & Schewe '05)

The LTL *distributed realizability problem for* $\mathcal{A}$ *is decidable if, and only if,* $\mathcal{A}$ *has no information fork.*

Information Forks



Theorem (Finkbeiner & Schewe '05)

The LTL *distributed realizability problem for $\mathcal{A}$ is decidable if, and only if, $\mathcal{A}$ has no information fork.*

Adding the coloring process does not introduce information forks.

Corollary

The PROMPT–LTL *distributed realizability problem for $\mathcal{A}$ is decidable if, and only if, $\mathcal{A}$ has no information fork.*

Outline

The Asynchronous Case

- Add a scheduler, which is part of the (antagonistic) environment: For every $p \in P$ add scheduling proposition $sched_p$ to $O_{p_{env}}$ and to I_p.
- Implementation may change its state only if enabled.

The Asynchronous Case

- Add a scheduler, which is part of the (antagonistic) environment: For every $p \in P$ add scheduling proposition $sched_p$ to $O_{p_{env}}$ and to I_p.
- Implementation may change its state only if enabled.
  $\Rightarrow$ Need assumptions on scheduler: bounded fairness

$$\bigwedge_p \mathbf{GF_P}\, sched_p$$

- Solution: assume-guarantee realizability for PROMPT–LTL.

The Asynchronous Case

- Add a scheduler, which is part of the (antagonistic) environment: For every $p \in P$ add scheduling proposition $sched_p$ to $O_{p_{env}}$ and to I_p.
- Implementation may change its state only if enabled.
  $\Rightarrow$ Need assumptions on scheduler: bounded fairness

$$\bigwedge_p \mathbf{GF_P}\, sched_p$$

- Solution: assume-guarantee realizability for PROMPT–LTL.

The asynchronous assume-guarantee realizability problem for a fixed architecture $\mathcal{A}$ asks, given PROMPT–LTL formulas φ_A, φ_G, to decide whether implementations f_p for every $p \neq p_{env}$ exist s.t.

$$\forall k_A \;\exists k_G \;\forall w \in \bigoplus_p f_p : (w, k_A) \models \varphi_A \text{ implies } (w, k_G) \models \varphi_G.$$

The Asynchronous Case

Lemma

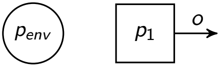
There exists an assume-guarantee PROMPT–LTL *specification that can be realized with an infinite-state implementation, but not with a finite-state implemenation.*

The Asynchronous Case

Lemma

There exists an assume-guarantee PROMPT–LTL *specification that can be realized with an infinite-state implementation, but not with a finite-state implemenation.*

Proof



$$\varphi_A = \mathbf{GF_P}\, o \vee \mathbf{FG}\, \neg o$$

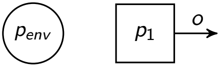
$$\varphi_G = \texttt{false}$$

The Asynchronous Case

Lemma

There exists an assume-guarantee PROMPT–LTL *specification that can be realized with an infinite-state implementation, but not with a finite-state implemenation.*

Proof



$$\varphi_A = \mathbf{GF_P}\, o \vee \mathbf{FG}\, \neg o$$
$$\varphi_G = \texttt{false}$$

- Implementation of p_1 has to falsify assumption φ_A, i.e., satisfy $\mathbf{F}\, \neg\, \mathbf{F_P}\, o \wedge \mathbf{G}\, \mathbf{F}\, o$ for every bound k
- This requires to produce infix $\emptyset^k$ for every k, but not suffix $\emptyset^\omega$.
- This is impossible for finite-state transducers.

The Asynchronous Case

Asynchronous LTL realizability is undecidable for architectures with at least two processes **[Schewe & Finkbeiner '06]**.

Theorem

The PROMPT–LTL *distributed assume-guarantee realizability problem is semi-decidable.*

The Asynchronous Case

Asynchronous LTL realizability is undecidable for architectures with at least two processes **[Schewe & Finkbeiner '06]**.

Theorem

The PROMPT–LTL distributed assume-guarantee realizability problem is semi-decidable.

Proof Sketch

- PROMPT–LTL assume-guarantee model checking is decidable **[Kupferman et al. '07]**.
- Apply bounded synthesis **[Finkbeiner & Schewe '07]**: Search through the space of transducers and model check whether they satisfy the assume-guarantee specification.

Outline

1. **Definitions**
   PROMPT-LTL
   Distributed Synthesis
   The Alternating Color Technique
2. **The Synchronous Case**
3. **The Asynchronous Case**
4. **Conclusion**

Conclusion

Results

- For a fixed architecture $\mathcal{A}$: synchronous $\mathrm{PROMPT–LTL}$ realizability for $\mathcal{A}$ is decidable if, and only if, synchronous LTL realizability for $\mathcal{A}$ is decidable.
- Asynchronous $\mathrm{PROMPT–LTL}$ assume-guarantee realizability is semi-decidable, just as for LTL.
- Both results can be extended to synthesis and to stronger logics.

Conclusion

Results

- For a fixed architecture $\mathcal{A}$: synchronous $\mathrm{PROMPT–LTL}$ realizability for $\mathcal{A}$ is decidable if, and only if, synchronous LTL realizability for $\mathcal{A}$ is decidable.
- Asynchronous $\mathrm{PROMPT–LTL}$ assume-guarantee realizability is semi-decidable, just as for LTL.
- Both results can be extended to synthesis and to stronger logics.

Open problems

- Single process asynchronous LTL realizability is decidable. What about $\mathrm{PROMPT–LTL}$?
- Distributed $\mathrm{PROMPT–LTL}$ synthesis as an optimization problem (see next talk for the single process case!)