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

2. The Synchronous Case

3. The Asynchronous Case

4. Conclusion
Syntax:

\[ \varphi ::= a \mid \neg a \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \mid \varphi R \varphi \mid F_p \varphi \]

where \(a\) ranges over a finite set \(AP\) of atomic propositions.
Syntax:

\[ \varphi ::= a \mid \neg a \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \mid \varphi R \varphi \mid F_P \varphi \]

where \( a \) ranges over a finite set \( AP \) of atomic propositions.

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

\[ (\rho, n, k) \models F_P \varphi : \rho \ldots \varphi \quad n \quad n + k \]
PROMPT-LTL

Syntax:

\[ \varphi ::= a \mid \neg a \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \mid \varphi R \varphi \mid F_P \varphi \]

where \( a \) ranges over a finite set \( AP \) of atomic propositions.

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

\[ (\rho, n, k) \models F_P \varphi : \quad \rho \quad \cdots \quad \varphi \quad \rho \quad \cdots \quad \varphi \quad n \quad \cdots \quad n + k \]

Example: \( G(q \rightarrow F_P p) \) 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 $O_p \subseteq AP$ of outputs (pairwise disjoint), and
- for all $p \in P \setminus \{p_{env}\}$ a set $I_p \subseteq AP$ of inputs.

Examples:

```
\begin{center}
\begin{tikzpicture}
  \node[rectangle, draw] (p1) at (0,0) {$p_1$};
  \node[rectangle, draw] (p2) at (0,-1) {$p_2$};
  \node[circle, draw] (p_env) at (-1,0) {$p_{env}$};

  \draw[->] (p_env) -- node[above] {$a$} (p1);
  \draw[->] (p_env) -- node[below] {$b$} (p2);
  \draw[->] (p1) -- node[above] {$c$} (p_env);
  \draw[->] (p2) -- node[below] {$d$} (p_env);
\end{tikzpicture}
\end{center}
```
An architecture consists of
- a finite set $P$ of processes with an environment process $p_{env}$,
- for all $p \in P$ a set $O_p \subseteq AP$ of outputs (pairwise disjoint), and
- for all $p \in P \setminus \{p_{env}\}$ a set $I_p \subseteq AP$ of inputs.

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

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

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

The PROMPT–LTL distributed realizability problem for a fixed architecture $A$ asks, given a PROMPT–LTL formula $\varphi$, 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 $\varphi$ 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 $O_p \subseteq \text{AP}$ of outputs (pairwise disjoint), and
- for all $p \in P \setminus \{p_{env}\}$ a set $I_p \subseteq \text{AP}$ of inputs.

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

The PROMPT–LTL distributed realizability problem for a fixed architecture $A$ asks, given a PROMPT–LTL formula $\varphi$, 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 $\varphi$ w.r.t. $k$.

Synthesis: compute such $f_p$, if they exist.
The Alternating Color Technique

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

$$
(r \rightarrow (r \cup (\neg r \cup \text{rel}(\psi)))) \land (\neg r \rightarrow (\neg r \cup (r \cup \text{rel}(\psi)))).
$$

Intuitively: $\psi$ has to be satisfied within one color change.
The Alternating Color Technique

1. Add fresh proposition $r \notin \text{AP}$: think of a coloring.

2. Obtain $\text{rel}(\varphi)$ by replacing each subformula $\mathbf{F}_P \psi$ of $\varphi$ by

$$(r \rightarrow (r \mathbf{U} (\neg r \mathbf{U} \text{rel}(\psi)))) \land (\neg r \rightarrow (\neg r \mathbf{U} (r \mathbf{U} \text{rel}(\psi)))).$$

Intuitively: $\psi$ has to be satisfied within one color change.
1. Add fresh proposition $r \notin \text{AP}$: think of a coloring.

2. Obtain $\text{rel}(\varphi)$ by replacing each subformula $F_P \psi$ of $\varphi$ by

$$
(r \rightarrow (r \mathbin{U} (\neg r \mathbin{U} \text{rel}(\psi)))) \land (\neg r \rightarrow (\neg r \mathbin{U} (r \mathbin{U} \text{rel}(\psi))))).
$$

Intuitively: $\psi$ has to be satisfied within one color change.
The Alternating Color Technique

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

\[
(r \rightarrow (r \bigvee (\neg r \bigvee \text{rel}(\psi)))) \land (\neg r \rightarrow (\neg r \bigvee (r \bigvee \text{rel}(\psi)))).
\]

Intuitively: \( \psi \) has to be satisfied within one color change.

Lemma (Kupferman et al. ’07)

Let \( \varphi \) be a PROMPT–LTL formula, \( w \in (2^{\text{AP}})^{\omega} \), and \( w' \in (2^{\text{AP} \cup \{r\}})^{\omega} \) 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 \( w' \models \text{rel}(\varphi) \).
2. Let \( k \in \mathbb{N} \). If \( w' \models \text{rel}(\varphi) \) 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
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 $\phi$ is realizable in $\mathcal{A}$ if, and only if, $\text{rel}(\phi) \land G F r \land G F \neg r$ is realizable in $\mathcal{A}^r$.

**Proof Idea:**

Martin Zimmermann Saarland University Distributed PROMPT-LTL Synthesis 8/15
Given architecture \( \mathcal{A} \), let \( \mathcal{A}' \) be \( \mathcal{A} \) with a new input-free (coloring) process \( p_{col} \) that outputs \( r \).

\[ p_{env} \rightarrow p_{col} \]

\[ p_{env} \rightarrow p_1 \rightarrow c \]

\[ p_{env} \rightarrow p_2 \rightarrow d \]

\[ p_{col} \rightarrow r \]
The Synchronous Case

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

**Theorem**

A PROMPT–LTL formula $\varphi$ is realizable in $\mathcal{A}$ if, and only if, $rel(\varphi) \land GF r \land GF\neg r$ is realizable in $\mathcal{A}'$. 

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 $\varphi$ is realizable in $\mathcal{A}$ if, and only if, $rel(\varphi) \land G F r \land G F \neg r$ is realizable in $\mathcal{A}^r$.

**Proof Idea:**

- Let $\varphi$ 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 $rel(\varphi) \land G F r \land G F \neg r$ 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 $\varphi$ is realizable in $\mathcal{A}$ if, and only if, $rel(\varphi) \land GF r \land GF \neg r$ is realizable in $\mathcal{A}^r$.

**Proof Idea:**

- Let $rel(\varphi) \land GF r \land GF \neg r$ 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 $\varphi$ in $\mathcal{A}$ with bound $2k$. 
Theorem (Finkbeiner & Schewe '05)

The LTL distributed realizability problem for $A$ is decidable if, and only if, $A$ has no information fork.

Adding the coloring process does not introduce information forks.

Corollary

The PROMPT–LTL distributed realizability problem for $A$ is decidable if, and only if, $A$ has no information fork.
Information Forks

Theorem (Finkbeiner & Schewe ’05)

The LTL distributed realizability problem for $A$ is decidable if, and only if, $A$ has no information fork.
Theorem (Finkbeiner & Schewe ’05)

The LTL distributed realizability problem for $A$ is decidable if, and only if, $A$ has no information fork.

Adding the coloring process does not introduce information forks.

Corollary

The PROMPT–LTL distributed realizability problem for $A$ is decidable if, and only if, $A$ has no information fork.
Outline

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

2. The Synchronous Case

3. The Asynchronous Case

4. Conclusion
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.
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 \( l_p \).

Implementation may change its state only if enabled.

\[ \Rightarrow \] Need assumptions on scheduler: bounded fairness

\[ \bigwedge_p GF P \, sched_p \]

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

$$\bigwedge_p GF_P sched_p$$

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

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

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

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

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

Proof

\[ \varphi_A = GFp \circ \lor FG \neg \circ \]

\[ \varphi_G = false \]
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 implementation.

Proof

\[ p_{env} \quad \square p_1 \xrightarrow{o} \]

\[ \varphi_A = GFp \ o \lor FG \neg \ o \]
[1]

\[ \varphi_G = false \]

- Implementation of \( p_1 \) has to falsify assumption \( \varphi_A \), i.e., satisfy \( F \neg Fp \ o \lor GF \ o \) 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.
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.
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 PROMPT–LTL realizability for $\mathcal{A}$ is decidable if, and only if, synchronous LTL realizability for $\mathcal{A}$ is decidable.
- Asynchronous 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 PROMPT–LTL?
- Distributed PROMPT–LTL synthesis as an optimization problem (see next talk for the single process case!)
Conclusion

Results

- For a fixed architecture $\mathcal{A}$: synchronous PROMPT–LTL realizability for $\mathcal{A}$ is decidable if, and only if, synchronous LTL realizability for $\mathcal{A}$ is decidable.
- Asynchronous 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 PROMPT–LTL?
- Distributed PROMPT–LTL synthesis as an optimization problem (see next talk for the single process case!)