Compositional control synthesis with temporal logic constraints

Jie Fu[1], Salar Moarref[2], Ufuk Topcu[3]

[1]Department of Computer Science
[2]Department Electrical and Systems Engineering
University of Pennsylvania

University of Texas at Austin

CDC Workshop 2015
A functional system is composed of *heterogeneous, reusable* modules (or functions) delivering distinct services while sharing *distributed computing and power resources* across networks.

The need of compositional/distributional synthesis methods.
Formal methods in control

- Formal specification: Temporal logic
- Formal synthesis method and verification

Diagram:

- LTL
- Discrete systems
- Environment
- System
- Abstraction
- Concrete implementation
- Synthesis
Stochastic system is abstracted into a Markov Decision Process (MDP).

An MDP: $M = (Q, A, P, u_0, R)$
- $Q$: Discrete state space
- $A$: input space
- $P$: transition function.
- $u_0$: Initial distribution
- $R$: Reward function

- Reward function is designed in a way such that Reward maximization $\Rightarrow$ Quantitative temporal logic constraints.
Reactive system is abstracted into a two-player game.

- Facilitates the design, focusing on “what”, instead of “how”
- No need for verification, the output system is correct by construction.
Formal methods in control: Challenges

- Computationally expensive: 2EXPTIME-complete for full LTL.
- State explosion problem: scalability is a big challenge for centralized approaches.
Scalable synthesis methods

Compositional synthesis for reactive systems with serial-interconnection [1]
- Break the problem into smaller pieces.
- Solve the sub-problems.
- Ensure the composition satisfies the specification.

Decomposition-based distributed optimal planning for large MDPs[2]
- Decompose large systems into smaller sub-systems.
- Distributed planning with alternating direction method of multipliers (ADMM).


**Definition: Serial-interconnected** reactive systems

- **I**: input variables, controlled by environment.
- **O₁**: output variables controlled by component 1.
- **O₂**: output variables controlled by component 2.
- **Serial interconnection**
  - **Acyclic dependency** between components’ outputs.
Global realizable GR(1) formula: Assume-Guarantee

\[ \phi := \phi_e \rightarrow \phi_s \]

Decomposition into local specifications

\[ \phi_1 := \phi_{e1} \rightarrow \phi_{s1} \]
\[ \phi_2 := \phi_{e2} \rightarrow \phi_{s2} \]

Decomposition satisfies

- **Weaker** assumption on the environment
  \[ \phi_e \rightarrow \phi_{e1} \land \phi_{e2} \]
- **Stronger** guarantee of the system
  \[ \phi_{s1} \land \phi_{s2} \rightarrow \phi_s \]

**Theorem:** \( \phi_e \rightarrow \phi_s \) is realizable if every \( \phi_{ei} \rightarrow \phi_{si} \) is realizable.

W.L.O.G, assume \( \phi_1 \) is realizable, \( \phi_2 \) is unrealizable.
An illustrative example

- Two controllable robots moving in a grid world
- Both must infinitely often visit the apple location
- And avoid collision

\[\phi_e \rightarrow \phi_1\]
Realizable

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>Apple</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

\[\phi_e \rightarrow \phi_2\]
Unrealizable

Strategy for \(R_1\)

```
start → 1 → 2 → 3 → 4 → start
```
Not only to decompose the global specification, but also to refine the local specifications for both to be realizable.

**Insight:** The guarantee of component 1 becomes the environment assumption of component 2.

- Synthesize assumptions/guarantees such that
  - Refined specifications are **both realizable**
    \[
    \phi_1^{\text{ref}} := \phi_{e1} \rightarrow (\phi_{s1} \land \varphi')
    \]
    \[
    \phi_2^{\text{ref}} := (\phi_{e2} \land \varphi) \rightarrow \phi_{s2}
    \]
  - Assumptions can be **ensured**
    \[
    \varphi' \rightarrow \varphi
    \]

\[
\phi_1^{\text{ref}} := \phi_{e1} \rightarrow (\phi_{s1} \land \varphi')
\]
\[
\phi_2^{\text{ref}} := (\phi_{e2} \land \varphi) \rightarrow \phi_{s2}
\]
Specification refinement

- How to compute the refinement formula $\varphi'$ and $\varphi$ such that both $\varphi_1^{ref}$ and $\varphi_2^{ref}$ are realizable?

$\varphi_1$ is realizable, $\varphi_2$ is unrealizable.

- Use counter-strategy to generate a refined assumption for the environment.
- **Iteratively refine** the assumption until the specification becomes realizable.
An illustrative example

- Unrealizable specification
  - Inferring LTL formulas from **counter-strategies**
  - Refining **environment** assumption for C2
  - Check if C1 can ensure the assumption
  - Refining **system guarantee** for C1

\[ \varphi = \varphi' = \square \lozenge Loc_{R_1} \neq Loc_{apple} \]

\[ \phi_e \rightarrow \phi_1 \land \varphi \]

**Guarantee Refinement**

| 2 | 3 | \n|---|---|---|
| R_1 | | |

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \varphi' \land \phi_e \rightarrow \phi_2 \]

**Assumption Refinement**
Generating refinement candidates

- How to compute the refinement formula $\varphi'$ and $\varphi$ such that both $\varphi_{ref}^1$ and $\varphi_{ref}^2$ are realizable?

  Possibly infinite many

- Restricted hypothesis space: **Patterns**

  LTL formulas of **special forms** which hold over all runs of the (counter-)strategy.

  - Discovering implicit GR(1)-form guarantees of a strategy
    
    \[ \square \lozenge \varphi, \square \varphi \text{ and } \square (\varphi \rightarrow \lozenge \varphi') \]

    - **Complement** of the forms in GR(1)
      
      \[ \lozenge \square \neg \varphi, \lozenge \neg \varphi \text{ and } \lozenge (\neg \varphi \land \lozenge \neg \varphi') \]

    - **Restrict** the system or the environment by adding their complement

\[ \begin{align*}
\phi_{ref}^1 & := \phi_{e1} \rightarrow (\phi_{s1} \land \varphi') \\
\phi_{ref}^2 & := (\phi_{e2} \land \varphi) \rightarrow \phi_{s2}
\end{align*} \]
○ R₁ eventually reaches and stays in cell 4
  • ◊□Loc_{R₁} ∈ \{4\}
  • □◊Loc_{R₁} \notin \{4\} → \varphi
○ Restrict R₁ by adding the complement to guarantees
○ A new strategy computed

\begin{align*}
\phi_e \rightarrow \phi_1 \land \varphi \\
\text{Guarantee Refinement}
\end{align*}

\begin{align*}
\varphi' \land \phi_e \rightarrow \phi_2 \\
\text{Assumption Refinement}
\end{align*}
Overview

**Global specification**

\[ \phi_e \rightarrow \phi_s \]

**Local specifications**

\[ \phi_1 := \phi_e \rightarrow \phi_{s1} \]
\[ \phi_2 := \phi_e \rightarrow \phi_{s2} \]

**Both realizable?**

- \( \phi_1 \) realizable
- \( \phi_2 \) not

**Return local strategies**

**Specification refinement**

\[ \phi_{e2} := \phi_{e2} \land \varphi' \]
\[ \phi_{s1} := \phi_{s1} \land \varphi \]
\[ \varphi \Rightarrow \varphi' \]

**Rule out counter-strategy**

\[ \varphi' = \neg \psi \]

**Discover patterns in counter-strategy**

\[ \psi \]

**Counter-Strategy**

**Environment**

- \( O_1 \)

**Component 1**

- \( O_1 \)

**Component 2**

- \( O_2 \)
Scalable synthesis methods

Compositional synthesis for reactive systems with serial-interconnection [1]

Decomposition-based distributed optimal planning for large MDPs [2]

MDPs with LTL constraints

Maximizing the probability for satisfying LTL formula

- Augmenting MDP with **memory states**
- Reduction to finite-horizon optimal planning problem

**MDP**

Visit 1 before 2

\[ \Diamond (1 \land \Diamond 2) \]

one memory

\( Y \): visited 1
\( N \): not visited 1 yet

**Augmented MDP**

Average reward under Büchi acceptance condition

- Augmenting the state space with finite memory
- Reduction to average-reward optimal planning problem
Decomposition of MDP from state space partitioning [Dean&Lin1995]:

Partition the state space:

S1: {4,5,6}
K1: {5,6}

S2: {0,1,2,3,7}
K2: {0,1,2,3}

K0: {4,2,7}

Interacting states

By deleting all states in K0 and their outgoing & incoming transitions, we obtain disconnected, sub-MDPs.
Recap: Solving MDP with linear programming

\[\min_{X \geq 0} c^T X \quad \text{s.t.} \quad AX = b.\]

\[\sum_a x(s', a) = \sum_{s \in S} \sum_{a \in A} x(s, a) P(s' \mid s, a) + \mu_0(s')\]

**Lemma:** From \(s \in K_i\), one can only reach a state in \(K_i\) or \(K_0\)

\[\min_{X_0, X_1, \ldots, X_N \geq 0} \sum_{i=0}^{N} c_i^T X_i \quad \text{s.t.} \quad AX = b\]

\[A = \begin{bmatrix}
A_{00} & A_{01} & A_{02} & \ldots & A_{0N} \\
A_{10} & A_{11} & & & \\
A_{20} & & A_{22} & & \\
\vdots & & & \ddots & \\
A_{N0} & & & & A_{NN}
\end{bmatrix}\]

**Can be formulated in ADMM form**
Experiments

Discounted reward case for 1000 x 10 gridworld example

<table>
<thead>
<tr>
<th>K1</th>
<th>K0</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td>K0</td>
<td>K0</td>
<td>K0</td>
</tr>
<tr>
<td>K3</td>
<td>K0</td>
<td>K4</td>
</tr>
</tbody>
</table>

Convergence

Each iteration 2.6 sec.

After 9000 iterations, the relative primal infeasibility is

\[
\frac{\|AX - b\|_2}{1 - \|b\|_1} = 0.29 \times 10^{-4}
\]

Average reward case for 50 x 50 gridworld with a Büchi objective

\[ \Box \Diamond (R_1 \land R_2) \land \Box \Diamond (R_3 \land R_4) \]

Converge in 14000 iteration to an optimal solution with infeasibility 0.015
Conclusion: Scalable synthesis methods

<table>
<thead>
<tr>
<th>Abstract system</th>
<th>Synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamical</td>
<td>2-Player game</td>
</tr>
<tr>
<td>Stochastic</td>
<td>MDP</td>
</tr>
</tbody>
</table>

○ Compositional reactive synthesis:
  ◆ Extension to handle more complex system structures.
  ◆ Extending to the full class of LTL.

○ Distributed planning:
  ◆ Efficient for systems with sparse interacting components.
  ◆ Able to handle the full class of LTL but the computation of almost-sure winning states is computational heavy.

Thank you!