Distributed synthesis for acyclic architectures

Anca Muscholl and Igor Walukiewicz

Bordeaux

TUM, July 2014
Motivation

\[ \varphi \subseteq (\text{In} \ ; \ \text{Out})^* \]

Synthesis (Church ’62)

- Given: specification \( \varphi \) relating inputs/outputs.
- Output: I/O automaton \( C \subseteq \varphi \) (controller) + additional requirements (e.g., unconstrained inputs).
Motivation

Synthesis (Church ’62)

- Given: specification $\varphi$ relating inputs/outputs.
- Output: I/O automaton $C \subseteq \varphi$ (controller) + additional requirements (e.g., unconstrained inputs).

Control (Ramadge & Wonham ’89)

Given
- plant = deterministic finite automaton $P$,
- some actions are uncontrollable, others are controllable
- specification $\varphi$. 
Motivation

Synthesis (Church ’62)
- Given: specification \( \varphi \) relating inputs/outputs.
- Output: I/O automaton \( C \subseteq \varphi \) (controller) + additional requirements (e.g., unconstrained inputs).

Control (Ramadge & Wonham ’89)
Given
- plant = deterministic finite automaton \( P \),
- some actions are uncontrollable, others are controllable
- specification \( \varphi \).

Search for a controller \( C \) such that \( P \times C \subseteq \varphi \). Controller must allow every uncontrollable action.
**Example**

- $\Sigma = \{a, b\}$ with $b$ uncontrollable
- Specification: never more than two consecutive $a$. 

Plant $P$
**Example**

- $\Sigma = \{a, b\}$ with $b$ uncontrollable
- Specification: never more than two consecutive $a$. 

![Diagram of Plant $P$ and Controller $C$]
Control

Example

- $\Sigma = \{a, b\}$ with $b$ uncontrollable
- Specification: never more than two consecutive $a$.

Plant $P$

Controller $C$

Controlled plant $= P \times C$
Distributed synthesis
Emerson and Clarke ’82, ‘Using branching time temporal logic to synthesize synchronization skeletons”

We present a method of constructing concurrent programs in which the synchronization skeleton of the program is automatically synthesized from a (branching time) temporal logic specification.
**Emerson and Clarke ’82, ‘Using branching time temporal logic to synthesize synchronization skeletons’**

*We present a method of constructing concurrent programs in which the synchronization skeleton of the program is automatically synthesized from a (branching time) temporal logic specification.*

BUT: *No environment. Synthesized programs are not guaranteed to be implementable in a distributed model.*
Emerson and Clarke ’82, “Using branching time temporal logic to synthesize synchronization skeletons”

We present a method of constructing concurrent programs in which the synchronization skeleton of the program is automatically synthesized from a (branching time) temporal logic specification.

BUT: No environment. Synthesized programs are not guaranteed to be implementable in a distributed model.

Pnueli/Rosner ’90, “Distributed reactive systems are hard to synthesize”

The limitation (of [CE82]) is that all the synthesis algorithms produce a program for a single module [...]. This is particularly embarrassing in cases that the problem we set out to solve is meaningful only in a distributed context, such as the mutual exclusion problem [...]. The somewhat ad-hoc solution [...] is to use first the general algorithm to produce a single module program, and then to decompose this program into a set of programs, one for each distributed component of the system.

BUT: the last task is undecidable.
Distributed synthesis

Distributed model?

- architecture: network of processes + local cooperation
- synchronous/asynchronous
- messages/shared variables/signals

Limits?

Synthesis is related to alternation. Alternation + multiple players is undecidable because of partial information (Peterson & Reif '79).
Distributed synthesis

Distributed model?
- architecture: network of processes + local cooperation
- synchronous/asynchronous
- messages/shared variables/signals

Limits?
Synthesis is related to alternation. Alternation + multiple players is undecidable because of partial information (Peterson & Reif ’79).
Distributed control

Ramadge and Wonham formulation

- Given a distributed automaton (plant) $P$ with two kinds of actions: controllable (system) and uncontrollable (environment), and a specification $\varphi$.

- Find another distributed automaton (controller) $C$ such that $P \times C \subseteq \varphi$. Controller cannot block uncontrollable actions.
**Distributed control**

**Ramadge and Wonham formulation**
- Given a **distributed** automaton (plant) $P$ with two kinds of actions: **controllable** (system) and **uncontrollable** (environment), and a specification $\varphi$.
- Find another distributed automaton (controller) $C$ such that $P \times C \subseteq \varphi$. Controller cannot block uncontrollable actions.

**Equivalent formulation**
Given a network $P$ of locally cooperating processes, find **local controllers** that communicate at the same time as the original processes.
Distributed control

Ramadge and Wonham formulation

- Given a distributed automaton (plant) $P$ with two kinds of actions: controllable (system) and uncontrollable (environment), and a specification $\varphi$.
- Find another distributed automaton (controller) $C$ such that $P \times C \subseteq \varphi$. Controller cannot block uncontrollable actions.

Equivalent formulation

Given a network $P$ of locally cooperating processes, find local controllers that communicate at the same time as the original processes.

How much communication?

- Restricting communication of controllers $\rightarrow$ undecidability.
- Extra communication of controllers $\rightarrow$ centralized control.
Distributed model: Zielonka automata
ZIELONKA AUTOMATA

Processes $p, q, r, \ldots$

Local states sets $S_p, S_q, S_r$.

Local transitions $\delta_b : S_q \times S_r \to S_q \times S_r, \ldots$
**Zielonka automata**

- **Processes** $p, q, r, \ldots$
- **Local states sets** $S_p, S_q, S_r$.
- **Local transitions** $\delta_b : S_q \times S_r \to S_q \times S_r, \ldots$

Process $p$ executes *local* action $c$. 
Processes $p, q, r, \ldots$

Local states sets $S_p, S_q, S_r$.

Local transitions $\delta_b : S_q \times S_r \rightarrow S_q \times S_r, \ldots$

Process $p$ executes local action $c$,
Processes $q, r$ synchronize on action $b$ (and update states) \ldots
ZIELONKA AUTOMATA

Processes $p, q, r, \ldots$

Local states sets $S_p, S_q, S_r$.

Local transitions $\delta_b : S_q \times S_r \rightarrow S_q \times S_r, \ldots$

Process $p$ executes local action $c$,
 Processes $q, r$ synchronize on action $b$ (and update states) . . .

Processes evolve asynchronously.
**Zielonka Automata**

- $\mathcal{P}$: finite set of processes, each $p \in \mathcal{P}$ has its set of states $S_p$
- **distributed alphabet** $\langle \Sigma, \text{dom} : \Sigma \to (2^{\mathcal{P}} \setminus \emptyset) \rangle$
  
  $\text{dom}(a) =$ set of processes involved in action $a$

- transition functions $\delta_a : \prod_{p \in \text{dom}(a)} S_p \to \prod_{p \in \text{dom}(a)} S_p$
ZIELONKA AUTOMATA

- $\mathbb{P}$: finite set of processes, each $p \in \mathbb{P}$ has its set of states $S_p$
- distributed alphabet $\langle \Sigma, \text{dom} : \Sigma \to (2^\mathbb{P} \setminus \emptyset) \rangle$
  $\text{dom}(a) =$ set of processes involved in action $a$
- transition functions $\delta_a : \prod_{p \in \text{dom}(a)} S_p \to \prod_{p \in \text{dom}(a)} S_p$

$\rightarrow$ exchange of information between processes executing $a$
ZIELONKA AUTOMATA

- $\mathbb{P}$: finite set of processes, each $p \in \mathbb{P}$ has its set of states $S_p$
- distributed alphabet $\langle \Sigma, \text{dom} : \Sigma \to (2^\mathbb{P} \setminus \emptyset) \rangle$
  $\text{dom}(a) =$ set of processes involved in action $a$
- transition functions $\delta_a : \prod_{p \in \text{dom}(a)} S_p \to \prod_{p \in \text{dom}(a)} S_p$
  $\rightarrow$ exchange of information between processes executing $a$

THE LANGUAGE OF THE AUTOMATON

The (regular) language of the product automaton.
Example

CAS

*Compare-and-swap (CAS):* \( \text{CAS}(x: \text{variable}; \ old, \ new: \ \text{int}) \).

Effect: return the value of \( x \) and set the value of \( x \) to \( new \), but only if the previous value of \( x \) was equal to \( old \).

Multi-threaded programs

- One process per thread \( t \) and per shared variable \( x \).
- CAS

\[
\begin{align*}
\text{CAS} \quad & x \quad i \quad \rightarrow \quad k \\
& t \quad s \quad \rightarrow \quad s' \\
& y = \text{CAS}_x(i, k)
\end{align*}
\[
\begin{align*}
\text{CAS} \quad & x \quad j \quad \rightarrow \quad j \\
& t \quad s \quad \rightarrow \quad s'' \\
& y = \text{CAS}_x(i, k)
\end{align*}
\]

In state \( s' \) we have \( y = i \), and in \( s'' \), we have \( y = j \).
Distributed control

Ramadge and Wonham formulation

- Given a Zielonka automaton $P$ (plant) with two kinds of actions: controllable (system) and uncontrollable (environment), and a specification $\varphi$.
- Find another Zielonka automaton (controller) $C$ such that $P \times C \subseteq \varphi$. Controller cannot block uncontrollable actions.
Distributed control

Ramadge and Wonham formulation

- Given a Zielonka automaton $P$ (plant) with two kinds of actions: controllable (system) and uncontrollable (environment), and a specification $\varphi$.
- Find another Zielonka automaton (controller) $C$ such that $P \times C \subseteq \varphi$. Controller cannot block uncontrollable actions.

Decidability status: OPEN

Local controllers exchange information when communicating. This entails a potentially unbounded information flow. Unclear whether a finite-state $C$ exists.
Distributed control

Ramadge and Wonham formulation

- Given a Zielenka automaton $P$ (plant) with two kinds of actions: controllable (system) and uncontrollable (environment), and a specification $\varphi$.
- Find another Zielenka automaton (controller) $C$ such that $P \times C \subseteq \varphi$. Controller cannot block uncontrollable actions.

Decidability status: open

Local controllers exchange information when communicating. This entails a potentially unbounded information flow. Unclear whether a finite-state $C$ exists.
Distributed control

Example

- Plant:

  \[
  \begin{align*}
  p & \quad c \\
  q & \quad a \quad \alpha \\
  r & \quad b \\
  \end{align*}
  \]

- Process \( q: (ab + ba)(\alpha + \beta) \)

- Controllable actions: \( c \) and \( d \)

- Specification:

  \[
  \begin{align*}
  p & \quad c \\
  q & \quad a \quad \alpha \\
  r & \quad b \\
  \end{align*}
  \]

  \[
  \begin{align*}
  p & \quad a \\
  q & \quad b \quad \beta \\
  r & \quad d \\
  \end{align*}
  \]

Plant is controllable: by communication with \( q \), processes \( p \) and \( r \) can know about the order between \( a \) and \( b \).
Decidability: partial results

[Madhusudan & Thiagarajan 2002]
Decidability for restricted local strategies:
- clocked: depending only on time, not history
- synchronization-rigid: each local strategy proposes either local actions or communication with the same process.

[Gastin & Lerman & Zeitoun 2004]
Decidability for restricted alphabets of actions: co-graphs.

[Madhusudan & Thiagarajan & Yang 2005]
Decidability for restricted Zielonka automata: every process misses only bounded knowledge. MSO specifications.

[Genest & Gimbert & M & Walukiewicz 2013]
Decidability for acyclic process communication and local reachability conditions (blocking). Shared actions controllable.
Complexity: non-elementary (complete). EXPTIME-complete for depth one.
Decidability for acyclic process communication

**Setting**
- Shared actions are binary. Communication graph is **acyclic**.
- Shared actions are **uncontrollable**. Not a restriction.
- Each process has its own ω-regular specification.

**Result**
For a given plant (Zielonka automaton) \( \mathcal{A} \) and local ω-regular specification \( \phi \) it is decidable whether a controller (Zielonka automaton) \( \mathcal{C} \) exists s.t. \( \mathcal{A} \times \mathcal{C} \models \phi \). Complexity is **non-elementary**, EXPTIME-complete for depth one.
**Proof**

**Main idea**
Reduction by simulating a leaf process by its parent.

![Diagram of process simulation](image)

**Further ideas**
- We can assume that there is a **bound** on the number of local actions of process \( r \) between consecutive synchronizations with \( q \).
- In \( A^\triangledown \), process \( q \) simulates process \( r \) by **choosing an \( r \)-local strategy** (until the simulation of a synchronization between \( q \) and \( r \)).
Proof

Simulation of \( r \) by \( q \)

In \( \mathcal{A}_q \) and \( \mathcal{A}_r \).

In \( \mathcal{A}_q^\nabla \).
CONCLUSION

- We have solved the control problem for acyclic communication. Complexity of server-client architectures is EXPTIME-c (this is undecidable in Pnueli & Rosner model).

- Open for arbitrary architectures.

- Application: control of hierarchical topologies with shared variables. We can translate reads/writes or more complex instructions into Zielonka automata. The reverse translation appears to require instructions like CAS.

- Open if control is decidable for hierarchical topologies with R/W.
CONCLUSION

- We have solved the control problem for acyclic communication. Complexity of server-client architectures is EXPTIME-c (this is undecidable in Pnueli & Rosner model).
- Open for arbitrary architectures.
- Application: control of hierarchical topologies with shared variables. We can translate reads/writes or more complex instructions into Zielonka automata. The reverse translation appears to require instructions like CAS.
- Open if control is decidable for hierarchical topologies with R/W.

Thank you!