Bridging the Gap between Reactive Synthesis and Supervisory Control

Stavros Tripakis
University of California, Berkeley

Joint work with Ruediger Ehlers (Berkeley, Cornell), Stéphane Lafortune (Michigan) and Moshe Vardi (Rice)

ExCAPE PI Meeting – June 2013
“Classic” Synthesis Frameworks

▶ **Reactive synthesis:**
  ▶ From *declarative specifications* (e.g., LTL formulas) to implementations (e.g., Mealy or Moore state machines).
  ▶ *On the Synthesis of a Reactive Module* [Pnueli-Rosner, POPL’89], but also earlier, e.g., [Church ’63].
  ▶ See Moshe’s summer school tutorial for details.

▶ **Supervisory control:**
  ▶ *Feedback control* for discrete-event systems (DES).
  ▶ *Supervisory control of a class of discrete event processes* and *On the supremal controllable sublanguage of a given language* [Ramadge-Wonham, SIAM J. Control Optim. ’87].
  ▶ See Stéphane’s textbook for more [Cassandras & Laforward ’08].
This Work

- Bridge the gap: how are the two frameworks related
  - in theory?
  - in practice?

- Bridge the communities.

- Pedagogical, although results are new to our knowledge.

- Work in progress.
Agenda

- Supervisory control.
- Reactive synthesis.
- Bridging the gap.
Agenda

- Supervisory control.
- Reactive synthesis.
- Bridging the gap.

More details in summer school talk.
SUPERVISORY CONTROL
Supervisory control problems (in general)

Given plant $G$, synthesize (if possible) supervisor $S$ such that the closed-loop system $S/G$ meets a certain specification.

Closed-loop system:

$S/G$: 

Diagram:

- Supervisor $S$
- Plant $G$
Plant generally modeled as \textit{discrete event system} (DES): regular language, deterministic finite automaton ($G$).
Supervisory Control: General Framework

closed-loop system $S/G$:

- Plant generally modeled as **discrete event system** (DES): regular language, deterministic finite automaton ($G$).
- Supervisor ($S$) can disable **controllable** events.

Specifications vary, but typically:
- **Safety**: all behaviors of the closed-loop system must be in some set of "good" behaviors.
- **Non-blockingness**: supervisor must always allow system to reach an accepting (aka marked) state.
- **Maximal permissiveness**: supervisor must not disable more events than strictly necessary.
Supervisory Control: General Framework

Plant generally modeled as **discrete event system** (DES): regular language, deterministic finite automaton ($G$).

Supervisor ($S$) can disable **controllable** events.

Specifications vary, but typically:

- **Safety**: all behaviors of the closed-loop system must be in some set of “good” behaviors.
- **Non-blockingness**: supervisor must always allow system to reach an accepting (aka *marked*) state.
- **Maximal permissiveness**: supervisor must not disable more events than strictly necessary.
Supervisor Synthesis: a basic problem

Simple Supervisory Control Problem (SSCP)

Given plant $G$, synthesize (if possible) supervisor $S$ such that:

- $S$ is non-blocking.
- $S$ is maximally-permissive, that is, for any other non-blocking supervisor $S'$:

$$L_m(S'/G) \subseteq L_m(S/G)$$
Supervisor Synthesis: a basic problem

Simple Supervisory Control Problem (SSCP)

Given plant $G$, synthesize (if possible) supervisor $S$ such that:

- $S$ is non-blocking.

- $S$ is **maximally-permissive**, that is, for any other non-blocking supervisor $S'$:

$$L_m(S'/G) \subseteq L_m(S/G)$$

- We proved: Can reduce the standard supervisory control problem (safety and non-blocking) to SSCP (non-blocking only).
Supervisor Synthesis: a basic problem

Simple Supervisory Control Problem (SSCP)

Given plant $G$, synthesize (if possible) supervisor $S$ such that:

- $S$ is non-blocking.
- $S$ is **maximally-permissive**, that is, for any other non-blocking supervisor $S'$:

\[ L_m(S'/G) \subseteq L_m(S/G) \]

- We proved: Can reduce the standard supervisory control problem (safety and non-blocking) to SSCP (non-blocking only).
- Can show that if a non-blocking supervisor exists, then the maximally-permissive non-blocking supervisor is **unique** and **state-based** (“memoryless”).
Supervisory Control of DES

- Much more to the story:
  - Partial observability
  - Decentralized, distributed, hierarchical control architectures
  - $\omega$-regular frameworks
  - Supervisory control of DES modeled as Petri nets
  - Not only control: Monitoring, fault diagnosis (partial observation)
  - ...

- Application areas:
  - Automated systems in control engineering: manufacturing, transportation, process control, etc.
  - Recently: Controlling execution of software for avoiding deadlocks in multithreaded programs [Wang et al. POPL’09]. (Cf. summer school lecture of Stéphane.)
REACTIVE SYNTHESIS
Reactive Synthesis Problem (RSP)

Given LTL formula $\phi$ with input/output atomic propositions, synthesize (if possible) a controller $M$ (Moore or Mealy machine) such that all behaviors of $M$ (inputs are uncontrollable) satisfy $\phi$.

This is the *implementability problem* [Pnueli-Rosner POPL 1989].
Specification: (G: always, X: next)

\[ \phi := G\left(c \rightarrow (Xg \land XXg \land XXX(b \land \neg g))\right) \]
Specification: (G: always, X: next)

\[ \phi := G\left( c \rightarrow (Xg \land XXg \land XXX(b \land \neg g)) \right) \]

Controller interface:
RSP

Specification: (G: always, X: next)

\[ \phi := G\left(c \rightarrow (Xg \land XXg \land XXX(b \land \neg g))\right) \]

Controller interface:

Controller generates a computation tree: all its paths must satisfy \( \phi \)
BRIDGING THE GAP
Summary: Main Differences

- Supervisory control has explicit plants – reactive synthesis does not.
- Supervisors are parents – controllers are ... controllers.
- Supervisory control asks for maximally-permissive controllers – these generally don’t exist in reactive synthesis.
- (Most of) supervisory control theory done in a finite-string setting – reactive synthesis is about infinite strings.
Controllers, Plants and Closed-Loop Systems in the Reactive Synthesis Framework

ExCAPE PI Meeting – June 2013
Stavros Tripakis (UC Berkeley)
Bridging the Gap
Controllers, Plants and Closed-Loop Systems in the Reactive Synthesis Framework

ExCAPE PI Meeting – June 2013
Stavros Tripakis (UC Berkeley)
Controllers, Plants and Closed-Loop Systems in the Reactive Synthesis Framework

inputs → Controller → outputs

Plant (Environment)
Controllers, Plants and Closed-Loop Systems in the Reactive Synthesis Framework

How to capture plants in the reactive synthesis framework?
Instead of asking for a controller implementing

$$\phi$$

we can ask for a controller implementing

$$\phi_{\text{plant}} \Rightarrow \phi$$

where $\phi_{\text{plant}}$ models the plant.
Capturing Plants in RSP

Example: plant never issues two $p$'s in a row

$$\phi_{plant} := G(p \Rightarrow X\neg p)$$
Capturing Plants in RSP

Example: plant never issues two \( p \)'s in a row

\[ \phi_{\text{plant}} \ := \ G(p \Rightarrow X \neg p) \]

However:

- Not always natural to capture the plant’s behavior: plant typically modeled as an automaton.
- Inefficient to do so: most synthesis algorithms start by transforming the formula into some automaton form.
  - This typically incurs an exponential blow-up.
Capturing Plants in RSP

Example: plant never issues two $p$’s in a row

$$\phi_{plant} := G(p \Rightarrow X\neg p)$$

However:

- Not always natural to capture the plant’s behavior: plant typically modeled as an automaton.
- Inefficient to do so: most synthesis algorithms start by transforming the formula into some automaton form.
  - This typically incurs an exponential blow-up.

$\Rightarrow$ motivation to define a reactive synthesis problem with plants
Reactive Synthesis with Plants

Inspired from [Kupferman et al CONCUR 2000]:

Reactive Synthesis Control Problem (RSCP)

Given plant $P$ and temporal logic formula $\phi$ synthesize (if possible) a strategy $f$ such that the closed-loop system satisfies $\phi$. 
Reactive Synthesis with Plants

Inspired from [Kupferman et al CONCUR 2000]:

**Reactive Synthesis Control Problem (RSCP)**

Given plant $P$ and temporal logic formula $\phi$ synthesize (if possible) a strategy $f$ such that the closed-loop system satisfies $\phi$.

- Plant modeled as a **transition system** with **system** states and **environment** states.
Reactive Synthesis with Plants

Inspired from [Kupferman et al CONCUR 2000]:

Reactive Synthesis Control Problem (RSCP)

Given plant $P$ and temporal logic formula $\phi$ synthesize (if possible) a strategy $f$ such that the closed-loop system satisfies $\phi$.

- Plant modeled as a transition system with system states and environment states.
- Strategy disables some successors of system states.
Reactive Synthesis with Plants

Inspired from [Kupferman et al CONCUR 2000]:

Reactive Synthesis Control Problem (RSCP)

Given plant $P$ and temporal logic formula $\phi$ synthesize (if possible) a strategy $f$ such that the closed-loop system satisfies $\phi$.

- Plant modeled as a transition system with system states and environment states.
- Strategy disables some successors of system states.
- Different versions of the problem depending on the temporal logic used: RSCP-LTL, RSCP-CTL, RSCP-CTL*, ...
Maximal Permissiveness in RSCP

Generally no unique maximally-permissive strategy.

Example: no unique maximally-permissive strategy to ensure $F_p$:

- **original plant**
- **strategy 1**
- **strategy 2**

- $\bigcirc$: system state
- $\square$: environment state
Maximal Permissiveness in RSCP

Generally no unique maximally-permissive strategy.

Example: no unique maximally-permissive strategy to ensure $Fp$:

- **original plant**
- **strategy 1**
- **strategy 2**

○: system state
□: environment state

Many other (non-state-based) strategies.
Yet for some formulas maximally-permissive strategies always exist:

**Theorem**

For any CTL formula $\phi := \text{AG EF } p$, where $p$ is a state formula, RSCP admits a unique maximally-permissive state-based strategy enforcing $\phi$ (if such a strategy exists).
Yet for some formulas maximally-permissive strategies always exist:

**Theorem**

For any CTL formula $\phi := \operatorname{AG \ EF} p$, where $p$ is a state formula, RSCP admits a unique maximally-permissive state-based strategy enforcing $\phi$ (if such a strategy exists).

We therefore define a variant of RSCP-CTL:

**RSCP-CTL}_{max}$

Given plant $P$ and CTL $\phi := \operatorname{AG \ EF} p$ compute (if it exists) the unique maximally-permissive state-based strategy enforcing $\phi$. 
Relations between different synthesis problems:

- **BSCP-NB** → **SSCP** (Corollary 1, special case)
- **SSCP** → **RSCP-CTL_{max}** (Theorem 5)
- **RSCP-CTL_{max}** → **RSCP-LTL** → **RSP**

**supervisory control problems**

**reactive synthesis problems**

Cf. technical report under preparation.

---

ExCAPE PI Meeting – June 2013
Stavros Tripakis (UC Berkeley)
Bridging the Gap

---

: work in progress
Results

Relations between different synthesis problems:

Cf. technical report under preparation.

---

- BSCP-NB
- SSCP
- RSCP-NB
- RSCP-CTL
- RSCP-LTL
- RSP

Corollary 1

special case

Theorem 5

Section 3.4

Section 3.5

supervisory control problems

reactive synthesis problems

----- : work in progress
Reducing SSCP to RSCP-CTL_{max}

Main idea:
▶ DES can be transformed to a transition system.
   ▶ Marked states labeled with atomic proposition $acc$.

▶ Non-blockingness can be expressed in CTL:

$$\phi_{nb} := AG EF acc$$

i.e., from any reachable state, there exists a path to an accepting state.
Reducing SSCP to RSCP-CTL$_{max}$: Example

DES $G$

Transition system $P_G$

○: system state
□: environment state
Theorem

Let $G$ be a DES plant and $P_G$ its transformation.

1. A non-blocking supervisor exists for $G$ iff a strategy enforcing $\phi_{nb} := \text{AG EF acc}$ exists for $P_G$.

2. Assuming supervisor/strategy exist, there is a 1-1 computable mapping between the unique non-blocking maximally-permissive state-based supervisor for $G$, and the unique maximally-permissive state-based strategy enforcing $\phi_{nb}$ on $P_G$. 
Conclusions and Perspectives

First (to our knowledge) bridge between the reactive synthesis and DES/supervisory control problems and communities.
Conclusions and Perspectives

First (to our knowledge) bridge between the reactive synthesis and DES/supervisory control problems and communities.

This work would not have happened without ExCAPE!

- Partial observability.
- Modular, decentralized, hierarchical control architectures.
- Algorithmic procedures.
- $\omega$-regular supervisory control theory (cf. [Thistle '96]).
- Supervisory control of Petri nets.
Conclusions and Perspectives

First (to our knowledge) bridge between the reactive synthesis and DES/supervisory control problems and communities.

This work would not have happened without ExCAPE!

Merely scratched the surface; expand bridge to:

- Partial observability.
- Modular, decentralized, hierarchical control architectures.
- Algorithmic procedures.
- $\omega$-regular supervisory control theory (cf. [Thistle ’96]).
- Supervisory control of Petri nets.
Bibliography

C. Cassandras and S. Lafortune.
*Introduction to Discrete Event Systems.*

A. Church.
Logic, arithmetic and automata.

Open systems in reactive environments: Control and synthesis.

A. Pnueli and R. Rosner.
On the synthesis of a reactive module.

P. Ramadge and W. Wonham.
Supervisory control of a class of discrete event processes.

J.G. Thistle.
Supervisory control of discrete event systems.

Y. Wang, S. Lafortune, T. Kelly, M. Kudlur, and S. Mahlke.
The theory of deadlock avoidance via discrete control.

W. Wonham and P. Ramadge.
On the supremal controllable sublanguage of a given language.