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 Summer School – 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 & Lafortune ’08].
This Talk

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

- Bridge the communities.

- Recent (unpublished) work, in progress.
Agenda

- Supervisory control.
- Reactive synthesis.
- Bridging the gap.
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:
Supervisory Control: General Framework

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

```
Supervisor S
```

```
Plant G
```

Closed-loop system $S/G$:

- 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

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.
Plant: DES = a deterministic finite automaton (DFA)

\[ G = (X, x_0, X_m, E, \delta) \]
 Supervisory Control: Plants

- **Plant**: DES = a deterministic finite automaton (DFA)

\[ G = (X, x_0, X_m, E, \delta) \]

- \( X \): set of states, \( x_0 \in X \) initial state, \( X_m \subseteq X \) marked states.
Supervisory Control: Plants

- **Plant**: DES = a deterministic finite automaton (DFA)

\[ G = (X, x_0, X_m, E, \delta) \]

- **X**: set of states, \( x_0 \in X \) initial state, \( X_m \subseteq X \) marked states.

- **E**: set of events partitioned into **controllable** and **uncontrollable** events

\[ E = E_c \cup E_{uc} \]
Supervisory Control: Plants

- **Plant**: DES = a deterministic finite automaton (DFA)

  \[ G = (X, x_0, X_m, E, \delta) \]

  - \( X \): set of states, \( x_0 \in X \) initial state, \( X_m \subseteq X \) marked states.
  - \( E \): set of events partitioned into **controllable** and **uncontrollable** events

  \[ E = E_c \cup E_{uc} \]

  - \( \delta : X \times E \rightarrow X \): transition function (possibly partial).
Example: DES

- $c_1, c_2$: controllable events.
- $u$: uncontrollable event.
Supervisory Control: Supervisors

- **Supervisor**: a total function

\[ S : E^* \rightarrow 2^{E_c} \]
Supervisory Control: Supervisors

- **Supervisor**: a total function
  \[ S : E^* \rightarrow 2^{E_c} \]

- **State-based** ("memoryless") supervisor: decision only depends on current state:
  \[ S : X \rightarrow 2^{E_c} \]
Supervisory Control: Supervisors

- **Supervisor**: a total function

\[ S : E^* \rightarrow 2^{Ec} \]

- **State-based** ("memoryless") supervisor: decision only depends on current state:

\[ S : X \rightarrow 2^{Ec} \]

- **Closed-loop system**:

\[ S/G : \]

\begin{center}
\begin{tikzpicture}[node distance=2cm, auto]
    \node (input) {events};
    \node (supervisor) [right of=input] {Supervisor \( S \)};
    \node (plant) [right of=supervisor] {Plant \( G \)};
    \node (output) [right of=plant] {disabling actions};

    \draw[->] (input) -- (supervisor);
    \draw[<-] (supervisor) -- (plant);
    \draw[->] (plant) -- (output);
\end{tikzpicture}
\end{center}
Supervisory Control: Supervisors

- **Supervisor**: a total function

\[ S : E^* \rightarrow 2^{E_c} \]

- **State-based** ("memoryless") supervisor: decision only depends on current state:

\[ S : X \rightarrow 2^{E_c} \]

- Closed-loop system:

  \[ S/G : \]

- **Note**: full observability (and plant is deterministic).
Example: Two Closed-Loop Systems

original DES

under supervisor 1

under supervisor 2
Example: Two Closed-Loop Systems

original DES

under supervisor 1

under supervisor 2

supervisor = “parent”
 Supervisory Control: Specifications

- Plant language $L_m(G)$ generally contains unsafe behaviors.
- **Safety specification**: regular language $L_{good} \subseteq L_m(G)$.
- Plant language $L_m(G)$ generally contains \textit{unsafe} behaviors.
- **Safety specification**: regular language $L_{\text{good}} \subseteq L_m(G)$.
- Supervisor’s goal: restrict $L_m(G)$ so that all behaviors are in $L_{\text{good}}$.

$$L_m(S/G) \subseteq L_{\text{good}}$$
Plant language $L_m(G)$ generally contains \textit{unsafe} behaviors.

**Safety specification**: regular language $L_{good} \subseteq L_m(G)$.

Supervisor’s goal: restrict $L_m(G)$ so that all behaviors are in $L_{good}$.

\[
L_m(S/G) \subseteq L_{good}
\]

supervisor does not have own acceptance conditions.
Supervisory Control: Specifications

- Plant language $L_m(G)$ generally contains *unsafe* behaviors.

- **Safety specification**: regular language $L_{good} \subseteq L_m(G)$.

- Supervisor’s goal: restrict $L_m(G)$ so that all behaviors are in $L_{good}$.

  $$L_m(S/G) \subseteq L_{good}$$

  supervisor does not have own acceptance conditions.

- **Trivial synthesis problem**: simply check whether the most-restrictive supervisor (disables everything it can) works.
Plant language $L_m(G)$ generally contains unsafe behaviors.

**Safety specification:** regular language $L_{good} \subseteq L_m(G)$.

Supervisor’s goal: restrict $L_m(G)$ so that all behaviors are in $L_{good}$.

\[
L_m(S/G) \subseteq L_{good}
\]

supervisor does not have own acceptance conditions.

Trivial synthesis problem: simply check whether the most-restrictive supervisor (disables everything it can) works.

Need maximal permissiveness and non-blockingness.
Non-blocking supervisor: from every reachable state in closed-loop system, there exists a path to an accepting state.

- **Original DES**
  - Blocking

- **Under supervisor 1**
  - Blocking

- **Under supervisor 2**
  - Non-blocking
Supervisory Control: Non-blockingness

Formally, DES $G$ is non-blocking iff:

$$L(G) \subseteq \text{pref\_closure}(L_m(G))$$

where $L(G)$ is the *unmarked* language of $G$: pretend that all states are accepting.

ExCAPE Summer School – June 2013
Stavros Tripakis (UC Berkeley)
Bridging the Gap 14 / 41
Supervisory Control: Non-blockingness

- Non-blockingness can capture safety specifications too.
- Ask for a supervisor for the product $G \times A$, where $A$ is a DFA with total transition function that accepts $L_{good}$. 
Non-blockingness can capture safety specifications too.

Ask for a supervisor for the product $G \times A$, where $A$ is a DFA with total transition function that accepts $L_{good}$.

⇒ All we need is non-blockingness and maximal-permissiveness.
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 $S'$:

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

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

- Can show that if a non-blocking supervisor exists, then the maximally-permissive non-blocking supervisor is **unique** and **state-based** ("memoryless").
SSCP: efficiently solvable with simple iterative procedure:

- Identify **Bad** (blocking) states.
- Repeat
  - “cutting” controllable transitions to **Bad** states
  - marking as **Bad** any uncontrollable predecessors of **Bad** states
  - marking as **Bad** any new blocking states (e.g., deadlocks).
Supervisor Synthesis

SSCP: efficiently solvable with simple iterative procedure:

- Identify **Bad** (blocking) states.
- Repeat
  - “cutting” controllable transitions to **Bad** states
  - marking as **Bad** any uncontrollable predecessors of **Bad** states
  - marking as **Bad** any new blocking states (e.g., deadlocks).

![Diagram of states and transitions](image)
Supervisor Synthesis

SSCP: efficiently solvable with simple iterative procedure:

- Identify **Bad** (blocking) states.
- Repeat
  - “cutting” controllable transitions to **Bad** states
  - marking as **Bad** any uncontrollable predecessors of **Bad** states
  - marking as **Bad** any new blocking states (e.g., deadlocks).
SSCP: efficiently solvable with simple iterative procedure:

- Identify **Bad** (blocking) states.
- Repeat
  - “cutting” controllable transitions to **Bad** states
  - marking as **Bad** any uncontrollable predecessors of **Bad** states
  - marking as **Bad** any new blocking states (e.g., deadlocks).
Supervisor Synthesis

SSCP: efficiently solvable with simple iterative procedure:

- Identify **Bad** (blocking) states.
- Repeat
  - “cutting” controllable transitions to **Bad** states
  - marking as **Bad** any uncontrollable predecessors of **Bad** states
  - marking as **Bad** any new blocking states (e.g., deadlocks).

![Diagram of a supervisor synthesis model with states $x_0, x_1, x_2, x_3$ and transitions $c_1, c_2, u$.]
Supervisor Synthesis

SSCP: efficiently solvable with simple iterative procedure:

- Identify **Bad** (blocking) states.
- Repeat
  - “cutting” controllable transitions to **Bad** states
  - marking as **Bad** any uncontrollable predecessors of **Bad** states
  - marking as **Bad** any new blocking states (e.g., deadlocks).

Finite # states $\Rightarrow$ termination
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: \((G: \text{always}, X: \text{next})\)

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

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

Controller interface:
Specification: \((G: \text{ always}, X: \text{ next})\)

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

Controller interface:

Controller generates a computation tree: all its paths must satisfy \(\phi\)
\( \phi := G(p \Rightarrow Fq) \quad p: \text{input}, \ q: \text{output}. \)

always ( \( p \Rightarrow \text{eventually } q \))
RSP: Example

\[ \phi := G(p \Rightarrow Fq) \quad p : \text{input, } q : \text{output.} \]

always ( \( p \Rightarrow \) eventually \( q \))

Possible solutions:

\[ \begin{align*}
\checkmark p/q \\
\checkmark \neg p/\neg q
\end{align*} \]
RSP: Example

\[ \phi := G(p \Rightarrow Fq) \quad p : \text{input, } q : \text{output.} \]

always \( (p \Rightarrow \text{eventually } q) \)

Possible solutions:

- \( p/q \)
- \( \neg p/\neg q \)
- \( q \)
RSP: Example

\[ \phi := G(p \Rightarrow Fq) \]

*always* ( \( p \Rightarrow \text{eventually } q \) )

Possible solutions:

- \( p/q \)
- \( \neg p/\neg q \)
- \( q \)
- \( \neg q \)
- \( \neg p \)
- \( p \)
- \( \neg p \)
- \( \neg q \)
- \( q \)
Another example

What about this?

\[ \phi := y \iff Fx \]

\( y : \text{output}, \ x : \text{input}. \)
Another example

What about this?

\[ \phi := y \iff Fx \quad y : \text{output}, \ x : \text{input}. \]

Spec is not implementable (no controller exists).

Controller cannot predict the future.
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

How to capture plants in the reactive synthesis framework?
Controllers, Plants and Closed-Loop Systems in the Reactive Synthesis Framework

Diagram:

```
inputs ? ----------------- Controller ----------------- outputs ?
```

ExCAPE Summer School – June 2013

Stavros Tripakis (UC Berkeley)

Bridging the Gap
Controllers, Plants and Closed-Loop Systems in the Reactive Synthesis Framework

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

How to capture plants in the reactive synthesis framework?

Diagram:

```
inputs          outputs
|                |
| Controller     |
|                |
| Plant (Environment) |
```

Instead of asking for a controller implementing

\[ \phi \]

we can ask for a controller implementing

\[ \phi_{plant} \Rightarrow \phi \]

where \( \phi_{plant} \) models the plant.
Example: plant never issues two $p$’s in a row

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

⇒ motivation to define a reactive synthesis problem with plants
Reactive Synthesis **with Plants**

Following [Kupferman et al CONCUR 2000]:

- Plant modeled as a *transition system*:

\[ P = (W, w_0, R, AP, L) \]
Reactive Synthesis with Plants

Following [Kupferman et al CONCUR 2000]:

- Plant modeled as a transition system:

\[ P = (W, w_0, R, AP, L) \]

- \( W \): set of states, \( w_0 \in W \) initial state.
  \( W \) partitioned into system ("controllable") and environment ("uncontrollable") states:

\[ W = W_s \cup W_e \]
Reactive Synthesis with Plants

Following [Kupferman et al CONCUR 2000]:

- Plant modeled as a transition system:
  
  \[ P = (W, w_0, R, AP, L) \]

- \( W \): set of states, \( w_0 \in W \) initial state.
  \( W \) partitioned into system (“controllable”) and environment (“uncontrollable”) states:
  
  \[ W = W_s \cup W_e \]

- \( R \subseteq W \times W \): transition relation (total, i.e., no deadlocks).
Reactive Synthesis with Plants

Following [Kupferman et al CONCUR 2000]:

- Plant modeled as a transition system:
  \[ P = (W, w_0, R, AP, L) \]

- \( W \): set of states, \( w_0 \in W \) initial state.
- \( W \) partitioned into system ("controllable") and environment ("uncontrollable") states:
  \[ W = W_s \cup W_e \]

- \( R \subseteq W \times W \): transition relation (total, i.e., no deadlocks).
- \( AP \): set of atomic propositions and \( L : W \rightarrow 2^{AP} \) labeling function.
**Strategy**: a function

\[ f : W^* \times W_s \rightarrow 2^W \]

Given history \( \rho \in W^* \) and current system state \( w \in W_s \), \( f(\rho, w) \) is a non-empty subset of the successors of \( w \).

That is, \( f \) **disables** some (but not all) successors of \( w \).
Strategy: a function

\[ f : W^* \times W_s \rightarrow 2^W \]

Given history \( \rho \in W^* \) and current system state \( w \in W_s \), \( f(\rho, w) \) is a non-empty subset of the successors of \( w \).

That is, \( f \) disables some (but not all) successors of \( w \).

Strategy \( f \) restricts plant \( P \) and produces a new plant \( P^f \) (the closed-loop system).
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 $P^f$ satisfies $\phi$. 
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 $P^f$ satisfies $\phi$.

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

○: system state
□: environment state
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

〇: system state
□: environment state

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

**Theorem**

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

We therefore define a variant of RSCP-CTL:

RSCP-CTL$_{\text{max}}$
Given plant $P$ and CTL $\phi := \text{AG EF } p$ compute (if it exists) the unique maximally-permissive state-based strategy enforcing $\phi$. 
Results

Relations between different synthesis problems:

BSCP-NB \rightarrow \text{special case} \rightarrow SSCP \rightarrow \text{Theorem 5} \rightarrow RSCP-LTL_{max} \rightarrow RSP

Corollary 1

supervisory control problems

reactive synthesis problems

Cf. technical report under preparation.

--- : work in progress
Results

Relations between different synthesis problems:

BSCP-NB \rightarrow \text{SSCP} \quad \text{special case}

\text{Corollary 1}

\text{Theorem 5}

\text{max}

RSCP-LTL \rightarrow \text{RSCP-CTL} \rightarrow \text{RSP}

Section 3.4

Section 3.5

\text{supervisory control problems}

\text{reactive synthesis problems}

Cf. technical report under preparation.

\rightarrow : \text{work in progress}
Reminder: SSCP and RSCP-CTL\textsubscript{max}

**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 $S'$:

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

**RSCP-CTL\textsubscript{max}**

Given plant $P$ and CTL $\phi := \text{AG EF } p$ compute (if it exists) the unique maximally-permissive state-based strategy enforcing $\phi$. 
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$^{\text{max}}$: 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 $\phi_{nb} := AG\ EF_{acc}$ 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!
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.
Thank you

Questions?
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.  