Distributed synthesis
for synchronous systems

Paul Gastin

LSV
ENS de Cachan & CNRS
Paul.Gastin@lsv.ens-cachan.fr

Dec 6th, 2006
Outline

1. Synthesis and control for sequential systems

Synthesis and control for distributed systems

Well-connected architectures
Open / Reactive system

Synthesis problem
- Given a specification $\varphi$, decide whether there exists a program $P$ such that $P\parallel E \models \varphi$ for all environment $E$.
- Build such a program $P$ (if one exists).
Open / Reactive system

inputs from $E$  outputs to $E$

Reactive system $S$

Program $P$

Specification $\varphi$

Synthesis problem

- Given a specification $\varphi$, decide whether there exists a program $P$ such that $P\parallel E \models \varphi$ for all environment $E$.
- Build such a program $P$ (if one exists).
Example: Elevator

- Inputs: call for level $i$.
- Outputs: open/close door $i$, move 1 level up/down.

Linear time: LTL, FO, MSO, regular, ...

- Safety: $G(\text{level} \neq i \rightarrow \text{is\_closed}_i)$
- Liveness: $G(\text{is\_called}_i \rightarrow F(\text{level} = i \land \text{is\_open}_i))$

Branching time: CTL, CTL*, $\mu$-calculus, ...

- $AG(\text{call}_i) \top$ (call$_i$ is uncontrollable)
- $AG EF(\text{level} = 0 \land \text{is\_open}_0)$
Specification

Example: Elevator

- Inputs: call for level $i$.
- Outputs: open/close door $i$, move 1 level up/down.

Linear time: LTL, FO, MSO, regular, ...

- Safety: $G(\text{level} \neq i \implies \text{is\_closed}_i)$
- Liveness: $G(\text{is\_called}_i \implies F(\text{level} = i \land \text{is\_open}_i))$

Branching time: CTL, CTL*, $\mu$-calculus, ...

- $AG\langle call_i \rangle \top$ (call$_i$ is uncontrollable)
- $AG EF(\text{level} = 0 \land \text{is\_open}_0)$
Synthesis of reactive programs

Reactive program

- $Q_x$: domain for input variable $x$
- $Q_y$: domain for output variable $y$
- Program: $f : Q_x^+ \rightarrow Q_y$
- Input: $x_1 x_2 \cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3) \cdots$ with $y_n = f(x_1 \cdots x_n)$ for all $n > 0$.

Churich problem (implementability) 1962

- Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?
- Given a branching time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that its run-tree satisfies $\varphi$?
Synthesis of reactive programs

Reactive program

- $Q_x$: domain for input variable $x$
- $Q_y$: domain for output variable $y$
- Program: $f : Q_x^+ \rightarrow Q_y$
- Input: $x_1 x_2 \cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3) \cdots$ with $y_n = f(x_1 \cdots x_n)$ for all $n > 0$.

Church problem (implementability) 1962

- Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?
- Given a branching time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that its run-tree satisfies $\varphi$?
Synthesis of reactive programs

Reactive program

- $Q_x$: domain for input variable $x$
- $Q_y$: domain for output variable $y$
- Program: $f : Q_x^+ \rightarrow Q_y$
- Input: $x_1x_2\cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3)\cdots$ with $y_n = f(x_1\cdots x_n)$ for all $n > 0$.

Chruch problem (implementability) 1962

- Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?
- Given a branching time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that its run-tree satisfies $\varphi$?
Synthesis of reactive programs

Chruch problem (implementability) 1962

Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?

Implementability $\neq$ Satisfiability

- $Q_x = \{0, 1\}$ and $\varphi := F(x = 1)$
- $\varphi$ is satisfiable: $(1, 0)^\omega \models \varphi$
- $\varphi$ is not implementable since the input is not controllable.

Implementability $\neq$ Validity of $\forall \vec{x} \exists \vec{y} \varphi$

- $Q_x = Q_y = \{0, 1\}$ and $\varphi := (y = 1) \leftarrow \rightarrow F(x = 1)$
- $\forall \vec{x} \exists \vec{y} \varphi$ is valid.
- $\varphi$ is not implementable by a reactive program.

For non-reactive terminating programs, Implementability $= \text{Validity of } \forall \vec{x} \exists \vec{y} \varphi$
Church problem (implementability) 1962

Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$,
Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?

Implementability $\neq$ Satisfiability

- $Q_x = \{0, 1\}$ and $\varphi := F(x = 1)$
- $\varphi$ is satisfiable: $(1, 0)^{\omega} \models \varphi$
- $\varphi$ is not implementable since the input is not controllable.

Implementability $\neq$ Validity of $\forall \vec{x} \exists \vec{y} \varphi$

- $Q_x = Q_y = \{0, 1\}$ and $\varphi := (y = 1) \iff F(x = 1)$
- $\forall \vec{x} \exists \vec{y} \varphi$ is valid.
- $\varphi$ is not implementable by a reactive program.

For non-reactive terminating programs, Implementability $= \text{Validity of } \forall \vec{x} \exists \vec{y} \varphi$
Synthesis of reactive programs

Chruch problem (implementability) 1962

Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?

Implementability $\neq$ Satisfiability

- $Q_x = \{0, 1\}$ and $\varphi := F(x = 1)$
- $\varphi$ is satisfiable: $(1, 0)^\omega \models \varphi$
- $\varphi$ is not implementable since the input is not controllable.

Implementability $\neq$ Validity of $\forall \vec{x} \exists \vec{y} \varphi$

- $Q_x = Q_y = \{0, 1\}$ and $\varphi := (y = 1) \iff F(x = 1)$
- $\forall \vec{x} \exists \vec{y} \varphi$ is valid.
- $\varphi$ is not implementable by a reactive program.

For non-reactive terminating programs, Implementability $= \text{ Validity of } \forall \vec{x} \exists \vec{y} \varphi$
Synthesis of reactive programs

Chruch problem (implementability) 1962

Given a linear time specification $\varphi$ over the alphabet $\Sigma = Q_x \times Q_y$, Does there exist a program $f$ such that all $f$-behaviors satisfy $\varphi$?

Theorem (Pnueli-Rosner 89)

- The specification $\varphi \in LTL$ is implementable iff the formula

$$A \varphi \land AG(\bigwedge_{a \in Q_x} EX(x = a))$$

is satisfiable.

- When $\varphi$ is implementable, we can construct a finite state implementation (program) in time doubly exponential in $\varphi$. 
**Control problem**

Given a system $S$ and a specification $\varphi$, decide whether there exists a controller $C$ such that $(S \otimes C) || E \models \varphi$.

Build such a controller $C$ (if one exists).

**Open system: Transitions system** $A = (Q, \Sigma, q_0, \delta)$

- $Q$: finite or infinite set of states,
- $\delta$: deterministic or non-deterministic transition function.
Control problem

Given a system $S$ and a specification $\varphi$, decide whether there exists a controller $C$ such that $(S \otimes C) \parallel E \models \varphi$.

Build such a controller $C$ (if one exists).

Open system: Transitions system $\mathcal{A} = (Q, \Sigma, q_0, \delta)$

- $Q$: finite or infinite set of states,
- $\delta$: deterministic or non deterministic transition function.
Control versus Game

<table>
<thead>
<tr>
<th>Correspondance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition system</td>
<td>= Game arena (graph).</td>
</tr>
<tr>
<td>Controllable events</td>
<td>= Actions of player 1 (controller).</td>
</tr>
<tr>
<td>Uncontrollable events</td>
<td>= Action of player 0 (opponent, environment).</td>
</tr>
<tr>
<td>Behavior</td>
<td>= Play.</td>
</tr>
<tr>
<td>Controller</td>
<td>= Strategy.</td>
</tr>
<tr>
<td>Specification</td>
<td>= Winning condition.</td>
</tr>
<tr>
<td>Finding a controller</td>
<td>= finding a winning strategy.</td>
</tr>
</tbody>
</table>

Theorem: Büchi - Landweber 1969

If the system is finite state and the specification is regular then the control problem is decidable.
Moreover, when \((S, \varphi)\) is controllable, we can synthesize a finite state controller.
Control versus Game

Correspondance

<table>
<thead>
<tr>
<th>Transition system</th>
<th>= Game arena (graph).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllable events</td>
<td>= Actions of player 1 (controller).</td>
</tr>
<tr>
<td>Uncontrollable events</td>
<td>= Action of player 0 (opponent, environment).</td>
</tr>
<tr>
<td>Behavior</td>
<td>= Play.</td>
</tr>
<tr>
<td>Controller</td>
<td>= Strategy.</td>
</tr>
<tr>
<td>Specification</td>
<td>= Winning condition.</td>
</tr>
<tr>
<td>Finding a controller</td>
<td>= finding a winning strategy.</td>
</tr>
</tbody>
</table>

Theorem: Büchi - Landweber 1969

If the system is finite state and the specification is regular then the control problem is decidable.

Moreover, when \((S, \varphi)\) is controllable, we can synthesize a finite state controller.
Program synthesis versus System control

Equivalence

The implementability problem for

\[ x \rightarrow \square \rightarrow y \]

is equivalent to the control problem for the system

\[ Q_x \rightarrow \bigcirc \rightarrow Q_y \]
Outline

Synthesis and control for sequential systems

2 Synthesis and control for distributed systems

Well-connected architectures
Distributed synthesis

Open distributed system $S$

inputs from $E$

outputs to $E$

Peterson-Reif 1979, Pnueli-Rosner 1990

In general, the problem is undecidable.
Distributed synthesis

Open distributed system $S$

inputs from $E$  outputs to $E$

$P_1$  $P_2$

$P_3$  $P_4$

Specification $\varphi$

Distributed synthesis problem

- Decide whether there exists a distributed program st.
  $P_1 \parallel \cdots \parallel P_n \parallel E \models \varphi$.
- Synthesis: If so, compute such a distributed program.

Peterson-Reif 1979, Pnueli-Rosner 1990

In general, the problem is undecidable.
Distributed synthesis

inputs from $E$  \quad outputs to $E$

Open distributed system $S$

$P_1$ $P_2$  $P_3$ $P_4$

Specification $\varphi$

Distributed synthesis problem

- Decide whether there exists a distributed program st. $P_1 \parallel \cdots \parallel P_n \parallel E \models \varphi$.
- Synthesis: If so, compute such a distributed program.

Peterson-Reif 1979, Pnueli-Rosner 1990

In general, the problem is undecidable.
Distributed control problem

- Decide whether there exists a distributed controller st.
  \[(S_1 \otimes C_1) \parallel \cdots \parallel (S_n \otimes C_n) \parallel E = \varphi.\]
- Synthesis: If so, compute such a distributed controller.
Distributed control

inputs from $E$  outputs to $E$

Controlled open distributed system $S$

$C_1$ $S_1$ $S_2$ $C_2$

$C_3$ $S_3$ $S_4$ $C_4$

Specification $\varphi$

Distributed control problem

- Decide whether there exists a distributed controller st.
  $$(S_1 \otimes C_1) \parallel \cdots \parallel (S_n \otimes C_n) \parallel E \models \varphi.$$  
- Synthesis: If so, compute such a distributed controller.
Architectures with shared variables

Example

Architecture $\mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W)$

- $\mathcal{P}$ finite set of processes/agents.
- $\mathcal{V}$ finite set of Variables.
- $R \subseteq \mathcal{P} \times \mathcal{V}$: $(a, x) \in R$ iff $a$ reads $x$.
  - $R(a)$ variables read by process $a \in \mathcal{P}$,
  - $R^{-1}(x)$ processes reading variable $x \in \mathcal{V}$.
- $W \subseteq \mathcal{P} \times \mathcal{V}$: $(a, x) \in W$ iff $a$ writes to $x$.
  - $W(a)$ variables written by process $a \in \mathcal{P}$,
  - $W^{-1}(x)$ processes writing to variable $x \in \mathcal{V}$. 
Architectures with shared variables

Example

Architecture $A = (\mathcal{P}, \mathcal{V}, R, W)$

- $\mathcal{P}$ finite set of processes/agents.
- $\mathcal{V}$ finite set of Variables.
- $R \subseteq \mathcal{P} \times \mathcal{V}$: $(a, x) \in R$ iff $a$ reads $x$.
  - $R(a)$ variables read by process $a \in \mathcal{P}$,
  - $R^{-1}(x)$ processes reading variable $x \in \mathcal{V}$.
- $W \subseteq \mathcal{P} \times \mathcal{V}$: $(a, x) \in W$ iff $a$ writes to $x$.
  - $W(a)$ variables written by process $a \in \mathcal{P}$,
  - $W^{-1}(x)$ processes writing to variable $x \in \mathcal{V}$.
Main parameters

- Which subclass of architectures?
- Which semantics?
  - synchronous (with or without delay), asynchronous
- What kind of specification?
  - LTL, CLT*, \( \mu \)-calculus
  - Rational, Recognizable
  - word/tree
- What kind of memory for the programs?
  - memoryless, local memory, causal memory
  - finite or infinite memory
## Main parameters

- **Which subclass of architectures?**
  - synchronous (with or without delay), asynchronous

- **Which semantics?**

- **What kind of specification?**
  - LTL, CLT*, μ-calculus
  - Rational, Recognizable
  - word/tree

- **What kind of memory for the programs?**
  - memoryless, local memory, causal memory
  - finite or infinite memory
## Distributed Synthesis or control

<table>
<thead>
<tr>
<th>Main parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Which subclass of architectures?</td>
</tr>
<tr>
<td>- Which semantics?</td>
</tr>
<tr>
<td>- synchronous (with or without delay), asynchronous</td>
</tr>
<tr>
<td>- What kind of specification?</td>
</tr>
<tr>
<td>- LTL, CLT*, $\mu$-calculus</td>
</tr>
<tr>
<td>- Rational, Recognizable</td>
</tr>
<tr>
<td>- word/tree</td>
</tr>
<tr>
<td>- What kind of memory for the programs?</td>
</tr>
<tr>
<td>- memoryless, local memory, causal memory</td>
</tr>
<tr>
<td>- finite or infinite memory</td>
</tr>
</tbody>
</table>
## Main parameters

- **Which subclass of architectures?**
- **Which semantics?**
  - synchronous (with our without delay), asynchronous
- **What kind of specification?**
  - LTL, CLT*, $\mu$-calculus
  - Rational, Recognizable
  - word/tree
- **What kind of memory for the programs?**
  - memoryless, local memory, causal memory
  - finite or infinite memory
Programs with local memory: $f_x : Q_u^* \rightarrow Q_x$ and $f_z : (Q_x \times Q_v)^* \rightarrow Q_z$.

- **Input:** $\begin{pmatrix} u_1 & u_2 & u_3 & \cdots \\ v_1 & v_2 & v_3 & \cdots \end{pmatrix} \in (Q_u \times Q_v)^\omega$.

- **Behavior:** $\begin{pmatrix} u_1 & u_2 & u_3 & \cdots \\ v_1 & v_2 & v_3 & \cdots \\ x_1 & x_2 & x_3 & \cdots \\ z_1 & z_2 & z_3 & \cdots \end{pmatrix}$

with $\begin{cases} x_n = f_x(u_1 \cdots u_n) \\ z_n = f_z((x_1, v_1) \cdots (x_n, v_n)) \end{cases}$ for all $n > 0$. 
Global versus distributed synthesis

Network information flow

Lemma (Rasala Lehman–Lehman 2004)

If \( f^1, \ldots, f^n : S^2 \to S \) are pairwise independent functions, then \( n \leq |S| + 1 \).

\( f^i, f^j \) are independent if \( (f^i, f^j) : S^2 \to S^2 \) is one to one.
Global versus distributed synthesis

Network information flow

Lemma (Rasala Lehman–Lehman 2004)

If $f^1, \ldots, f^n : S^2 \to S$ are pairwise independent functions, then $n \leq |S| + 1$. $f^i, f^j$ are independent if $(f^i, f^j) : S^2 \to S^2$ is one to one.
Global versus distributed synthesis

Network information flow

Lemma (Rasala Lehman–Lehman 2004)

If $f_1, \ldots, f_n : S^2 \to S$ are pairwise independent functions, then $n \leq |S| + 1$. $f_i, f_j$ are independent if $(f_i, f_j) : S^2 \to S^2$ is one to one.
Theorem (Pnueli-Rosner FOCS’90)

The synthesis problem for architecture $\mathcal{A}_0$ and LTL (or CTL) specifications is undecidable.

Proof

Reduction from the halting problem on the empty tape.
Undecidability proof 1

**SPEC$_1$:** processes $a$ and $b$ must output configurations

$$0^q 1^p 0 \cdots : n(v) = p$$

$$\#^{q+p}C\#^\omega : \text{where } C \in \Gamma^*Q\Gamma^+$$

$$(v = 0 \land y = \#) W (v = 1 \land (v = 1 \land y = \#) W (v = 0 \land y \in \Gamma^*Q\Gamma^+\#^\omega))$$

where

$$y \in \Gamma^*Q\Gamma^+\#^\omega \overset{\text{def}}{=} y \in \Gamma U (y \in Q \land X (y \in \Gamma U (y \in \Gamma \land X G y = \#)))$$
**Undecidability proof 1**

**SPEC$_1$:** processes $a$ and $b$ must output configurations

- $u \xrightarrow{a} a \xrightarrow{x} v$ and $v \xrightarrow{b} b \xrightarrow{y} v$
- $0^q 1^p 0 \cdots : n(v) = p$
- $\#^{q+p} C \#^\omega : \text{where } C \in \Gamma^* \Gamma^+$

$$(v = 0 \land y = \#) \mathcal{W} \left( v = 1 \land (v = 1 \land y = \#) \mathcal{W} (v = 0 \land y \in \Gamma^* \Gamma^+ \#^\omega) \right)$$

where

$$y \in \Gamma^* \Gamma^+ \#^\omega \overset{\text{def}}{=} y \in \Gamma \cup \left( y \in Q \land X(y \in \Gamma \cup (y \in \Gamma \land X G y = \#)) \right)$$
Undecidability proof 2

\textbf{SPEC}_2: processes \( a \) and \( b \) must start with the first configuration:

- \( u \) and \( v \)
- \( 0^q 10 \cdots : n(v) = 1 \)
- \( \#^{q+1} C_1 \#^\omega \)

\[ v = 0 \mathcal{W} (v = 1 \land \mathcal{X}(v = 0 \rightarrow y \in C_1 \#^\omega)) \]
**Undecidability proof 2**

**SPEC**$_2$: processes $a$ and $b$ must start with the first configuration $u x v 0^q 10 \cdots : n(v) = 1$

\[
v = 0 \mathcal{W} \left( v = 1 \land \mathcal{X} (v = 0 \rightarrow y \in C_1 \#^\omega) \right)
\]
Undecidability proof 3

**SPEC\(_3\):** if \( n(u) = n(v) \) are synchronized then \( x = y \)

\[
0^q1^p0\ldots \\
u \quad b
\]

\[
\#^{q+p}C\#^\omega \\
x \quad y
\]

\[
n(u) = n(v) \quad \rightarrow \quad G(x = y)
\]

where

\[
n(u) = n(v) \overset{\text{def}}{=} (u = v = 0) \cup (u = v = 1 \land (u = v = 1 \cup u = v = 0))
\]
**SPEC₃:** if \( n(u) = n(v) \) are synchronized then \( x = y \)

\[
0^q1^p0\cdots \xrightarrow{a} x \xleftarrow{\#^q+p\#^\omega} 0^q1^p0\cdots \xrightarrow{b} y
\]

\[
n(u) = n(v) \rightarrow G(x = y)
\]

where

\[
n(u) = n(v) \overset{\text{def}}{=} (u = v = 0) \cup (u = v = 1 \land (u = v = 1 \cup u = v = 0))
\]
Undecidability proof 4

**SPEC₄:** if \( n(u) = n(v) + 1 \) are synchronized then \( C_y \vdash C_x \)

\[
\begin{align*}
0^q1^p+10\ldots & \quad u \quad 0^q1^p+1p0\ldots \\
\downarrow & \quad \downarrow \\
\vdots & \quad \vdots \\
\uparrow & \quad \uparrow \\
0^q1^p+1C_x \# & \omega \\
\downarrow & \\
x & \\
\downarrow & \\
\# & \omega \\
\uparrow & \\
y & \\
\downarrow & \\
0^q1^p+1C_y \# & \omega
\end{align*}
\]

\[
n(u) = n(v) + 1 \quad \rightarrow \quad x = y \cup (\text{Trans}(y, x) \land X^3 G x = y)
\]

where \( \text{Trans}(y, x) \) is defined by

\[
\bigvee_{(p,a,q,b,\leftarrow) \in T, c \in \Gamma} (y = cpa \land x = qcb) \quad \bigvee_{(p,a,q,b,\rightarrow) \in T, c \in \Gamma} (y = pac \land x = bqc) \\
\bigvee_{(p,a,q,b,\rightarrow) \in T} (y = pa\# \land x = bq\Box)
\]
**Undecidability proof 4**

**SPEC₄**: if \( n(u) = n(v) + 1 \) are synchronized then \( C_y \models C_x \)

\[
\begin{align*}
0^q1^p+10\ldots & \quad \begin{array}{c}
\text{u} \\
\downarrow \\
\text{a} \\
\downarrow \\
\text{x}
\end{array} \\
\begin{array}{c}
\text{v} \\
\downarrow \\
\text{b} \\
\downarrow \\
\text{y}
\end{array} & \quad 0^{q+1}1^p0\ldots
\end{align*}
\]

\[
\begin{align*}
\#^{q+p+1}C_x \#^\omega & \quad \#^{q+p+1}C_y \#^\omega
\end{align*}
\]

\[
n(u) = n(v) + 1 \quad \longrightarrow \quad x = y \cup \left( \text{Trans}(y, x) \land X^3 \text{G} x = y \right)
\]

where \( \text{Trans}(y, x) \) is defined by

\[
\bigvee_{(p,a,q,b,\leftarrow) \in T, c \in \Gamma} (y = cpa \land x = qcb) \quad \bigvee_{(p,a,q,b,\rightarrow) \in T, c \in \Gamma} (y = pac \land x = bqc) \quad \bigvee_{(p,a,q,b,\rightarrow) \in T} (y = pa\# \land x = bq\Box)
\]
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

Corollary

Specifications 1-4 and 5: $G x \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

Corollary

Specifications 1-4 and 5: $G_x \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

Induction

Corollary

Specifications 1-4 and 5: $G_x \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

\[
\begin{array}{c}
0^{q+1}1^p0\ldots \\
\downarrow \\
\text{Induction} \\
\#^{q+p+1}C_p\#^\omega \\
\downarrow \\
x \\
\end{array}
\quad
\begin{array}{c}
u \\
\downarrow \\
\text{SPEC}_3 \\
\#^{q+p+1}C_p\#^\omega \\
\downarrow \\
y \\
\end{array}
\quad
\begin{array}{c}
0^{q+1}1^p0\ldots \\
\downarrow \\
a \\
\downarrow \\
b \\
\end{array}
\]

Corollary

Specifications 1-4 and 5: $Gx \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Undecidability proof 5

Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

Specifications 1-4 and 5: $G \ x \neq \ \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Lemma: winning strategies must simulate the Turing machine

For each $p \geq 1$, if $n(u) = p$ then $C_x = C_p$ is the $p$-th configuration of the Turing machine starting from the empty tape.

Proof

$0^q 1^{p+1} 0 \ldots$  $u$  $v$  $0^{q+1} 1^p 0 \ldots$

SPEC$_4$  SPEC$_3$

$0^q 1^{p+1} C_{p+1} \#^\omega$  $x$  $y$  $0^{q+1} 1^p \#^\omega$

Corollary

Specifications 1-4 and 5: $G_x \neq \text{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.
Decidability of distributed synthesis

Some examples

Undecidable

Decidable

Undecidable
Decidability

Pipeline

Pnueli-Rosner (FOCS’90)
The synthesis problem for pipeline architectures and LTL specifications is non elementary decidable.

Peterson-Reif (FOCS’79)
multi-person games with incomplete information. → non-elementary lower bound for the synthesis problem.
The synthesis problem is non elementary decidable for

- one-way chain, one-way ring, two-way chain and two-way ring,
- \( \text{CTL}^* \) specifications (or tree-automata specifications) on all variables,
- synchronous, 1-delay semantics,
- local strategies.
Decidability

Kupferman-Vardi (LICS’01)

The synthesis problem is non elementary decidable for

- one-way chain, one-way ring, two-way chain and two-way ring,
- CTL\(^*\) specifications (or tree-automata specifications) on all variables,
- synchronous, 1-delay semantics,
- local strategies.

one-way ring

```
\[\begin{array}{c}
\text{x} \\
\downarrow \\
\text{a}_1 \\
\downarrow \\
\text{y}_1 \\
\downarrow \\
\text{z}_1 \\
\end{array} \quad \begin{array}{c}
\text{a}_2 \\
\downarrow \\
\text{y}_2 \\
\downarrow \\
\text{z}_2 \\
\end{array} \quad \begin{array}{c}
\text{a}_3 \\
\downarrow \\
\text{y}_3 \\
\end{array} \]
```

\[\begin{array}{c}
\text{a}_1 \\
\downarrow \\
\text{a}_2 \\
\downarrow \\
\text{a}_3 \\
\end{array} \quad \begin{array}{c}
\text{y}_1 \\
\downarrow \\
\text{y}_2 \\
\downarrow \\
\text{y}_3 \\
\end{array} \quad \begin{array}{c}
\text{z}_1 \\
\downarrow \\
\text{z}_2 \\
\downarrow \\
\text{z}_3 \\
\end{array} \]
The synthesis problem is non elementary decidable for

- one-way chain, one-way ring, two-way chain and two-way ring,
- $\text{CTL}^*$ specifications (or tree-automata specifications) on all variables,
- synchronous, 1-delay semantics,
- local strategies.

**two-way chain**
Example

Programs: \( f_x : Q_u^* \rightarrow Q_x \) and \( f_z : (Q_x \times Q_v)^* \rightarrow Q_z \).

- Input: \( \begin{pmatrix} u_1 & u_2 & u_3 & \cdots \\ v_1 & v_2 & v_3 & \cdots \end{pmatrix} \in (Q_u \times Q_v)^\omega \).

- Behavior:
  \[
  \begin{pmatrix}
    u_1 & u_2 & u_3 & \cdots \\
    v_1 & v_2 & v_3 & \cdots \\
    x_1 & x_2 & x_3 & \cdots \\
    z_1 & z_2 & z_3 & \cdots
  \end{pmatrix}
  \]

with
\[
\begin{align*}
x_{n+1} &= f_x(u_1 \cdots u_n) \\
z_{n+1} &= f_z((x_1, v_1) \cdots (x_n, v_n))
\end{align*}
\] for all \( n > 0 \).
Decidability

An adequately connected architecture is equivalent to a singleton architecture.
The synthesis problem is decidable for LTL specifications and pipelines of adequately connected architectures.

\[ Q_x = Q \text{ for all } x \in \mathcal{V} \]
Adequately connected sub-architecture

\[ Q_x = Q \text{ for all } x \in \mathcal{V} \]

- An adequately connected architecture is equivalent to a singleton architecture.
- The synthesis problem is decidable for LTL specifications and pipelines of adequately connected architectures.

Pnueli-Rosner (FOCS’90)
Adequately connected sub-architecture

\[ Q_x = Q \quad \text{for all} \quad x \in V \]

\[ x = u \otimes v \]

Pnueli-Rosner (FOCS’90)

- An adequately connected architecture is equivalent to a singleton architecture.
- The synthesis problem is decidable for LTL specifications and pipelines of adequately connected architectures.
Decidability

An adequately connected architecture is equivalent to a singleton architecture.
The synthesis problem is decidable for LTL specifications and pipelines of adequately connected architectures.
Information fork criterion
(Finkbeiner–Schewe LICS ’05)
Information fork criterion
(Finkbeiner–Schewe LICS ’05)
Information fork criterion
(Finkbeiner–Schewe LICS ’05)
Outline

Synthesis and control for sequential systems

Synthesis and control for distributed systems

Well-connected architectures
**Definition**

For an output variable $y$, $\text{View}(y)$ is the set of input variables $x$ such that there is a path from $x$ to $y$.

**Definition**

An architecture is uniformly well connected if there is a uniform way to route variables in $\text{View}(y)$ to $y$ for each output variable $y$.

**Example**

![Graph](image)
Uniformly well connected architectures

**Definition**

For an output variable $y$, $\text{View}(y)$ is the set of input variables $x$ such that there is a path from $x$ to $y$.

**Definition**

An architecture is uniformly well connected if there is a uniform way to route variables in $\text{View}(y)$ to $y$ for each output variable $y$.

**Example**

![Diagram of uniformly well connected architecture](image-url)
Uniformly well connected architectures

**Definition**

An architecture is uniformly well connected if there is a uniform way to route variables in View(\(v\)) to \(v\) for each output variable \(v\).

- If the capacity of internal variables is big enough then the architecture is uniformly well-connected.
- If the architecture is uniformly well-connected then we can use causal strategies instead of local ones.

**Proposition**

Checking whether a given architecture is uniformly well connected is NP-complete.

**Proof**

Reduction to the multicast problem in Network Information Flow. The multicast problem is NP-complete (Rasala Lehman-Lehman 2004).
Uniformly well connected architectures

Definition
An architecture is uniformly well connected if there is a uniform way to route variables in \( \text{View}(v) \) to \( v \) for each output variable \( v \).

- If the capacity of internal variables is big enough then the architecture is uniformly well-connected.
- If the architecture is uniformly well-connected then we can use causal strategies instead of local ones.

Proposition
Checking whether a given architecture is uniformly well connected is NP-complete.

Proof
Reduction to the multicast problem in Network Information Flow. The multicast problem is NP-complete (Rasala Lehman-Lehman 2004).
An architecture has **uncomparable information** if there exist $y_1, y_2$ output variables such that $\text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset$ and $\text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset$.

Otherwise it is said to have **preordered information**.
An architecture has **uncomparable information** if there exist \( y_1, y_2 \) output variables such that \( \text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset \) and \( \text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset \).

Otherwise it is said to have **preordered information**.
An architecture has **uncomparable information** if there exist $y_1, y_2$ output variables such that $\text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset$ and $\text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset$.

Otherwise it is said to have **preordered information**.
Definition

An architecture has **uncomparable information** if there exist $y_1,y_2$ output variables such that $\text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset$ and $\text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset$.

Otherwise it is said to have **preordered information**.
Definition

An architecture has **uncomparable information** if there exist $y_1, y_2$ output variables such that $\text{View}(y_2) \setminus \text{View}(y_1) \neq \emptyset$ and $\text{View}(y_1) \setminus \text{View}(y_2) \neq \emptyset$.

Otherwise it is said to have **preordered information**.
Theorem

Architectures with uncomparable information are undecidable for LTL or CTL input-output specifications.

Proof.
Theorem

Architectures with uncomparable information are undecidable for LTL or CTL input-output specifications.

Proof.

Diagram showing the relationship between $x_0$, $x_1$, $y_0$, and $y_1$. The diagram illustrates the undecidability through a series of connected nodes and arrows, indicating the flow of information and the impossibility of a decision under uncomparable information.
Uncomparable information yields undecidability

Theorem

Architectures with uncomparable information are undecidable for LTL or CTL input-output specifications.

Proof.
Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS’01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS'01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS'01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS’01)
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS'01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Uniformly well connected architectures

Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for $\text{CTL}^*$ external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS'01)

The synthesis problem is decidable for pipeline architectures and $\text{CTL}^*$ specifications on all variables.
Theorem (PG, Nathalie Sznajder, Marc Zeitoun)

Uniformly well connected architectures with preordered information are decidable for CTL* external specifications.

Proof.

Theorem: Kupferman-Vardi (LICS’01)

The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.
Robust specifications

Definition
A specification $\varphi$ is robust if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

Theorem
The synthesis problem for uniformly well-connected architectures and external and robust $\text{CTL}^*$ specifications is decidable.

Proof.
Robust specifications

Definition
A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

Theorem
The synthesis problem for **uniformly well-connected** architectures and external and robust $\text{CTL}^*$ specifications is decidable.

Proof.
Robust specifications

**Definition**

A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**

The synthesis problem for uniformly well-connected architectures and external and robust CTL* specifications is decidable.

**Proof.**

![Diagram showing a network of nodes and arrows representing the robust specifications and proof.

$\begin{align*}
x_1 &\rightarrow y_1 \\
x_2 &\rightarrow y_2 \\
x_3 &\rightarrow y_3 \\
x_4 &\rightarrow y_4
\end{align*}$


Robust specifications

**Definition**

A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**

The synthesis problem for uniformly well-connected architectures and external and robust CTL* specifications is decidable.

**Proof.**

[Diagram of a system with nodes $x_1, x_2, x_3, x_4$ and $y_1, y_2, y_3, y_4$, with arrows indicating connections and a shaded region.]
Robust specifications

**Definition**
A specification $\varphi$ is *robust* if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**
The synthesis problem for uniformly well-connected architectures and external and robust $\text{CTL}^*$ specifications is decidable.

**Proof.**

Diagram showing the connections between variables $x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4$. The diagram is divided into three parts, with $x_1$ and $x_2$ connected to $x_3$ and $x_4$, and $y_1$ and $y_2$ connected to $y_3$ and $y_4$. The robust specifications ensure that the system remains connected and operational under various external conditions.
Robust specifications

**Definition**

A specification $\varphi$ is **robust** if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**

The synthesis problem for uniformly well-connected architectures and external and robust CTL* specifications is decidable.

**Proof.**

![Diagram](image-url)
**Definition**

A specification $\varphi$ is robust if it can be written $\varphi = \bigvee \bigwedge_{z \in \text{Out}} \varphi_z$ where $\varphi_z$ depends only on $\text{View}(z) \cup \{z\}$.

**Theorem**

The synthesis problem for uniformly well-connected architectures and external and robust CTL* specifications is decidable.

**Proof.**
Well-connected architectures

**Definition**

An architecture is well connected if, for each output variable $y$, the subarchitecture formed by $(E^*)^{-1}(y)$ is uniformly well connected.

**Example: well-connected but not UWC**
Well-connected architectures

Definition
An architecture is well connected if, for each output variable $y$, the subarchitecture formed by $(E^*)^{-1}(y)$ is uniformly well connected.

Rasala Lehman–Lehman 2004
One can solve the network information flow in the special case where there is a unique sink in polynomial time.

Corollary
One can decide whether an architecture is well-connected in polynomial time.
The synthesis problem for LTL specifications and well connected architectures with preordered information is undecidable.
Theorem
The synthesis problem for LTL specifications and well connected architectures with preordered information is undecidable.
The synthesis problem for LTL specifications and well connected architectures with preordered information is undecidable.
Specification and routing
Specification and routing

$x \rightarrow p_0 \rightarrow z_0 \rightarrow p \rightarrow \cdots \rightarrow p_6 \rightarrow y$

$p_0 \rightarrow p_1 \rightarrow u_1 \rightarrow w_1$

$p_0 \rightarrow p_2 \rightarrow u_2 \rightarrow w_2$

$p_0 \rightarrow p_3 \rightarrow u_3 \rightarrow w_3$

$p_0 \rightarrow p_4 \rightarrow u_4 \rightarrow w_4$

$p_0 \rightarrow p_5 \rightarrow u_5 \rightarrow w_5$

$p_0 \rightarrow p_6 \rightarrow u_6 \rightarrow w_6$

$w \rightarrow 0 \cdots 01$
Specification and routing
Specification and routing

0 \ldots 01\ldots
Specification and routing

One bit of $u$ is hidden to $p_6$
Open problem

- Find a decidability criterium for external specifications and well-connected architectures.
- Find a decidability criterium for external specifications and arbitrary architectures.
- Decidability of the distributed control/synthesis problem for robust and external specifications.