Distributed synthesis for synchronous systems[1]

Paul Gastin

LSV

ENS de Cachan & CNRS

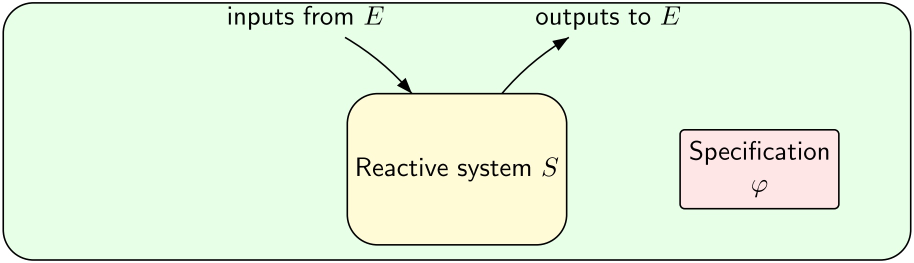
`Paul.Gastin@lsv.ens-cachan.fr`

Dec 6th, 2006

[1] Joint work with Nathalie Sznajder and Marc Zeitoun

Outline

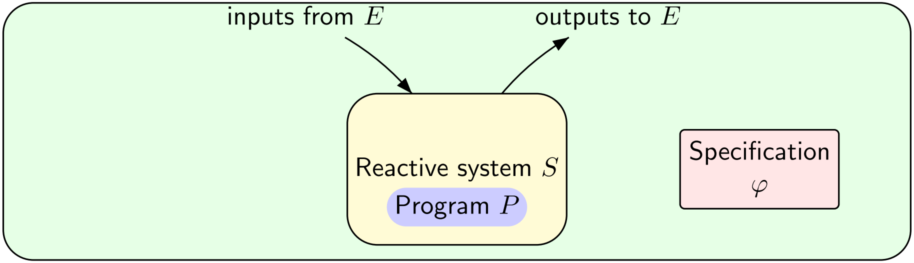
Open / Reactive system



Synthesis problem

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

Open / Reactive system



Synthesis problem

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

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: $\mathsf{G}(\texttt{level} \neq i \longrightarrow \texttt{is\_closed}_i)$
- Liveness: $\mathsf{G}(\texttt{is\_called}_i \longrightarrow \mathsf{F}(\texttt{level} = i \wedge \texttt{is\_open}_i))$

Branching time: CTL, CTL^*, μ-calculus, …

- $\mathsf{AG}\langle\texttt{call}_i\rangle\top$ ($\texttt{call}_i$ is uncontrollable)
- $\mathsf{AG}\,\mathsf{EF}(\texttt{level} = 0 \wedge \texttt{is\_open}_0)$

Specification

Example: Elevator

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

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

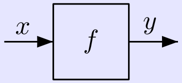
- Safety: $\mathsf{G}(\texttt{level} \neq i \longrightarrow \texttt{is\_closed}_i)$
- Liveness: $\mathsf{G}(\texttt{is\_called}_i \longrightarrow \mathsf{F}(\texttt{level} = i \wedge \texttt{is\_open}_i))$

Branching time: CTL, CTL^*, μ-calculus, ...

- $\mathsf{AG}\langle\texttt{call}_i\rangle\top$ ($\texttt{call}_i$ is uncontrollable)
- $\mathsf{AG}\,\mathsf{EF}(\texttt{level} = 0 \wedge \texttt{is\_open}_0)$

Synthesis of reactive programs

Reactive program



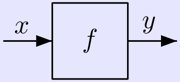
- Q_x: domain for input variable x
- Q_y: domain for output variable y
- Program: $f : Q_x^+ \to Q_y$
- Input: $x_1 x_2 \cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3) \cdots$ with $y_n = f(x_1 \cdots x_n)$ for all $n > 0$.

Chruch problem (implementability) 1962

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

Synthesis of reactive programs

Reactive program



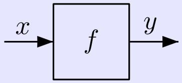
- Q_x: domain for input variable x
- Q_y: domain for output variable y
- Program: $f : Q_x^+ \to Q_y$
- Input: $x_1 x_2 \cdots \in Q_x^\omega$.
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3) \cdots$ with $y_n = f(x_1 \cdots x_n)$ for all $n > 0$.

Chruch problem (implementability) 1962

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

Synthesis of reactive programs

Reactive program



- Q_x: domain for input variable x
- Q_y: domain for output variable y
- Program: $f : Q_x^+ \to Q_y$
- 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 $n > 0$.

Chruch problem (implementability) 1962

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

Synthesis of reactive programs

Chruch problem (implementability) 1962

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

Implementability $\neq$ Satisfiability

- $Q_x = \{0, 1\}$ and $\varphi := \mathsf{F}(x = 1)$
- φ is satisfiable: $(1, 0)^\omega \models \varphi$
- φ 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 $\varphi := (y = 1) \longleftrightarrow \mathsf{F}(x = 1)$
- $\forall\vec{x}\ \exists\vec{y}\ \varphi$ is valid.
- φ is not implementable by a reactive program.

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

Synthesis of reactive programs

Chruch problem (implementability) 1962

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

Implementability $\neq$ Satisfiability

- $Q_x = \{0, 1\}$ and $\varphi := \mathsf{F}(x = 1)$
- φ is satisfiable: $(1, 0)^\omega \models \varphi$
- φ 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 $\varphi := (y = 1) \longleftrightarrow \mathsf{F}(x = 1)$
- $\forall\vec{x}\ \exists\vec{y}\ \varphi$ is valid.
- φ is not implementable by a reactive program.

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

Synthesis of reactive programs

Chruch problem (implementability) 1962

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

Implementability $\neq$ Satisfiability

- $Q_x = \{0,1\}$ and $\varphi := \mathsf{F}(x = 1)$
- φ is satisfiable: $(1,0)^\omega \models \varphi$
- φ 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 $\varphi := (y = 1) \longleftrightarrow \mathsf{F}(x = 1)$
- $\forall\vec{x}\ \exists\vec{y}\ \varphi$ is valid.
- φ is not implementable by a reactive program.

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

Synthesis of reactive programs

Chruch problem (implementability) 1962

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

Theorem (Pnueli-Rosner 89)

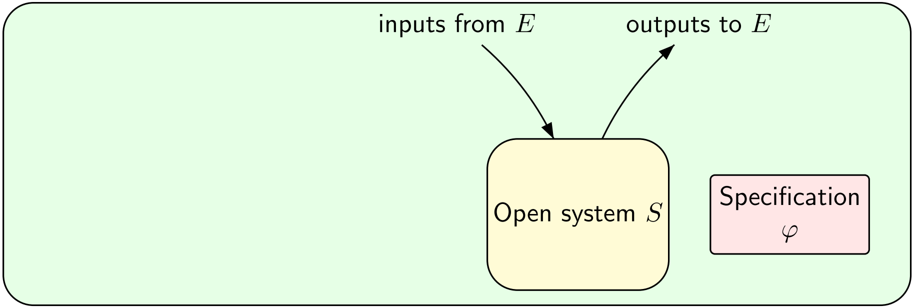
- The specification $\varphi \in \mathrm{LTL}$ is implementable iff the formula

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

  is satisfiable.

- When φ is implementable, we can construct a finite state implementation (program) in time doubly exponential in φ.

Control problem



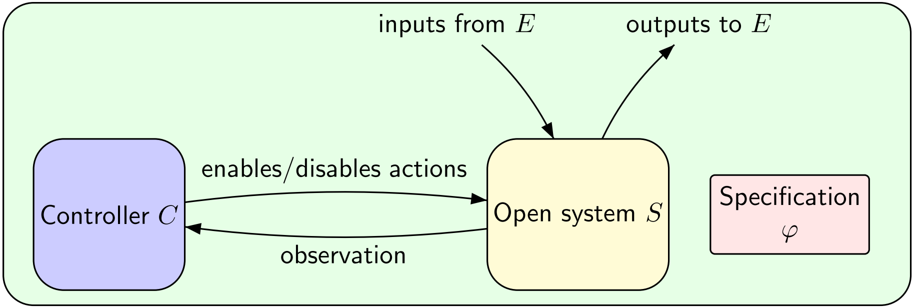
Open system: Transitions system $\mathcal{A} = (Q, \Sigma, q_0, \delta)$

- Q: finite or infinite set of states,
- δ: deterministic or non deterministic transition function.

Control problem

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

Control problem



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

- Q: finite or infinite set of states,
- δ: deterministic or non deterministic transition function.

Control problem

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

Control versus Game

Correspondance

Transition system	=	Game arena (graph).
Controllable events	=	Actions of player 1 (controller).
Uncontrollable events	=	Action of player 0 (opponent, environment).
Behavior	=	Play.
Controller	=	Strategy.
Specification	=	Winning condition.
Finding a controller	=	finding a winning strategy.

Theorem: Büchi - Landweber 1969

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

Control versus Game

Correspondance

Transition system	=	Game arena (graph).
Controllable events	=	Actions of player 1 (controller).
Uncontrollable events	=	Action of player 0 (opponent, environment).
Behavior	=	Play.
Controller	=	Strategy.
Specification	=	Winning condition.
Finding a controller	=	finding a winning strategy.

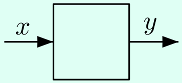
Theorem: Büchi - Landweber 1969

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

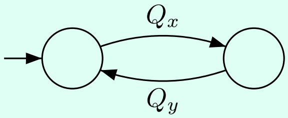
Program synthesis versus System control

Equivalence

The implementability problem for

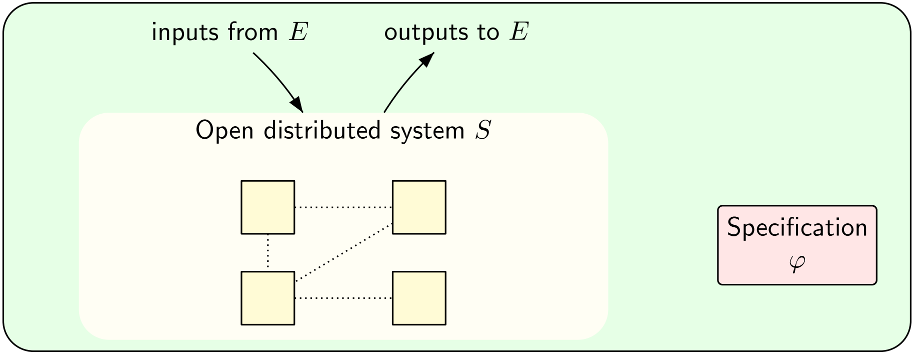


is equivalent to the control problem for the system



Outline

Distributed synthesis



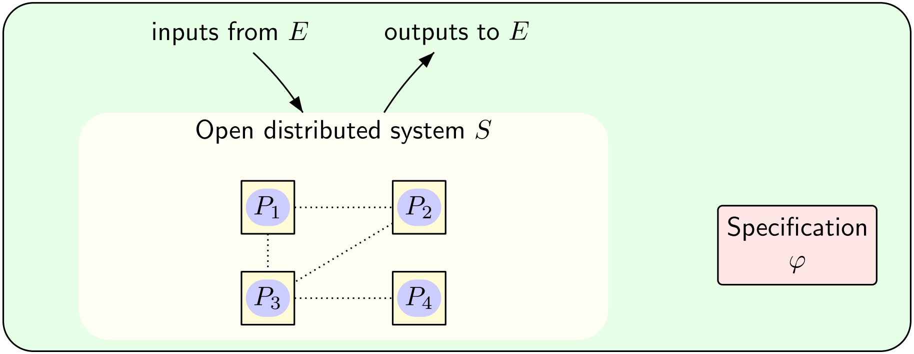
Distributed synthesis problem

- 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



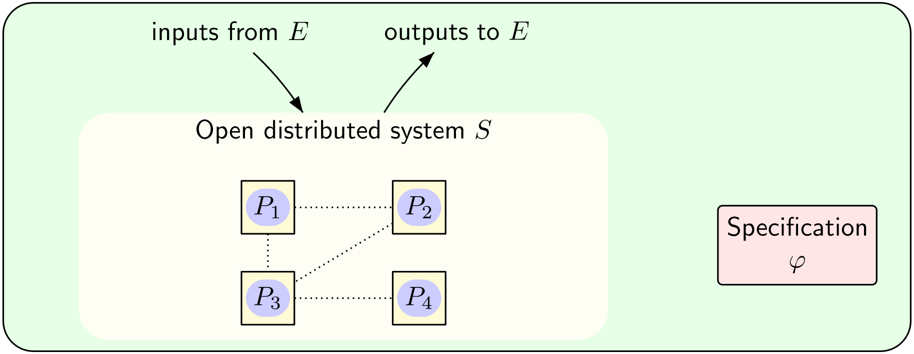
Distributed synthesis problem

- 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



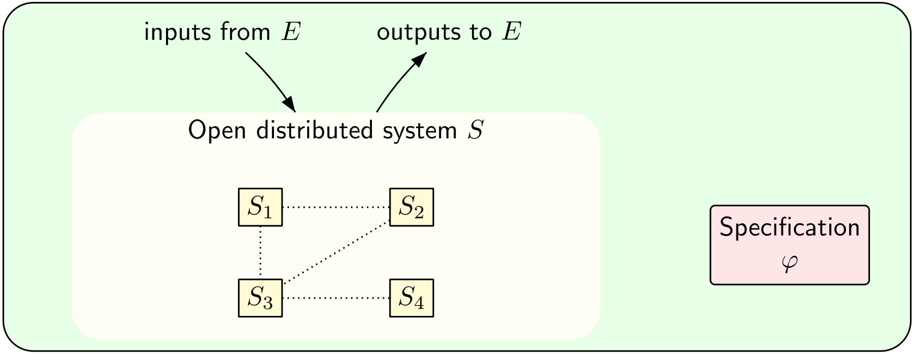
Distributed synthesis problem

- 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.

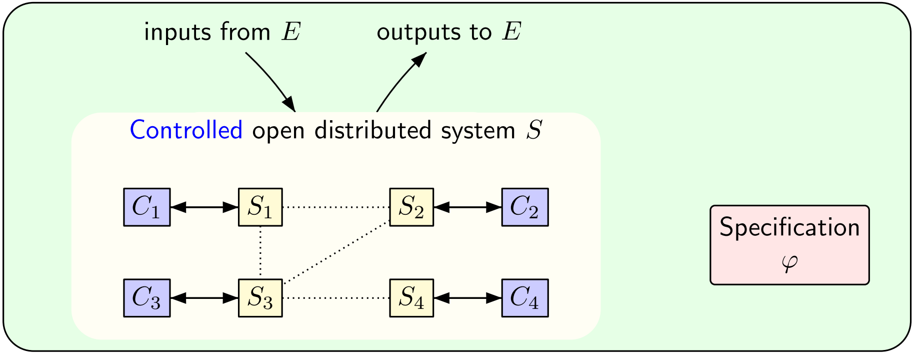
Distributed control



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.

Distributed control



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

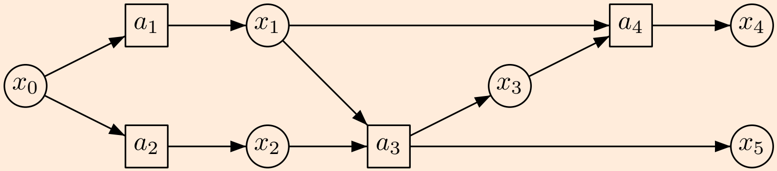
$$\begin{tikzcd}
 & a_1 \arrow[r] & x_1 \arrow[rrr] \arrow[ddr] & & & a_4 \arrow[r] & x_4 \\
x_0 \arrow[ur] \arrow[dr] & & & & x_3 \arrow[ur] & & \\
 & a_2 \arrow[r] & x_2 \arrow[r] & a_3 \arrow[ur] \arrow[rrr] & & & x_5
\end{tikzcd}$$

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


$$\begin{tikzcd}
 & \boxed{a_1} \arrow[r] & x_1 \arrow[rrr] \arrow[rd] & & & \boxed{a_4} \arrow[r] & x_4 \\
x_0 \arrow[ru] \arrow[rd] & & & & x_3 \arrow[ru] & & \\
 & \boxed{a_2} \arrow[r] & x_2 \arrow[r] & \boxed{a_3} \arrow[ru] \arrow[rrr] & & & x_5
\end{tikzcd}$$

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 $x \in \mathcal{V}$.

Distributed Synthesis or control

Main parameters

- Which subclass of architectures?
- Which semantics?

  synchronous (with our without delay), asynchronous
- 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

Main parameters

- Which subclass of architectures?
- Which semantics?

  synchronous (with our without delay), asynchronous
- 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

Main parameters

- Which subclass of architectures?
- Which semantics?

  synchronous (with our without delay), asynchronous
- 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

Main parameters

- Which subclass of architectures?
- Which semantics?

  synchronous (with our without delay), asynchronous
- 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

0-delay synchronous semantics

Example

$$\begin{tikzcd}
\circled{u} \arrow[r] & \boxed{a} \arrow[d] & \\
& \circled{x} \arrow[d] & \\
\circled{v} \arrow[r] & \boxed{b} \arrow[r] & \circled{z}
\end{tikzcd}$$

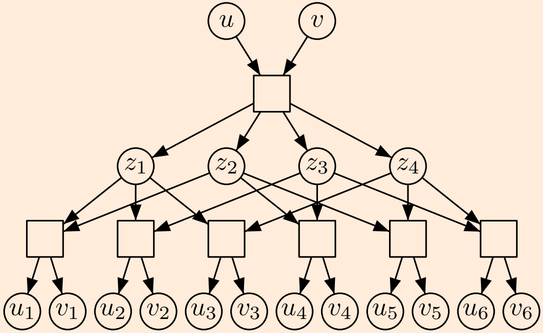
Programs with local memory: $f_x : Q_u^* \longrightarrow Q_x$ and $f_z : (Q_x \times Q_v)^* \longrightarrow Q_z$.

- 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

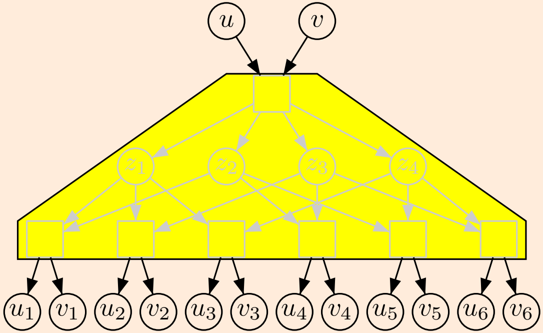


Lemma (Rasala Lehman–Lehman 2004)

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

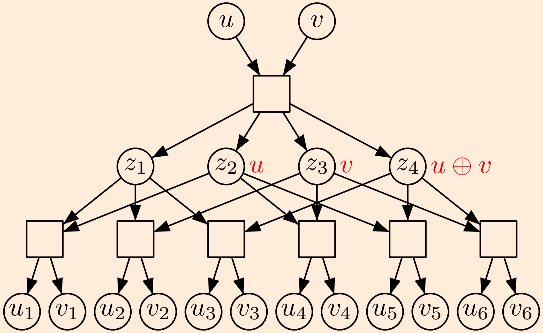

u
v
z_1
z_2
z_3
z_4
u_1 v_1 u_2 v_2 u_3 v_3 u_4 v_4 u_5 v_5 u_6 v_6

Lemma (Rasala Lehman–Lehman 2004)

If $f^1, \dots, f^n : S^2 \to S$ are pairwise independent functions, then $n \le |S| + 1$.
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

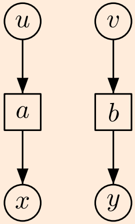


Lemma (Rasala Lehman–Lehman 2004)

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.

Undecidability

Architecture $\mathcal{A}_0$



$$\begin{tikzcd} u \arrow[d] & v \arrow[d] \\ \boxed{a} \arrow[d] & \boxed{b} \arrow[d] \\ x & y \end{tikzcd}$$

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

$$\begin{tikzcd}
u \arrow[d] & v \arrow[d] & 0^q 1^p 0 \cdots \quad : n(v) = p \\
\boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \\
x & y & \#^{q+p} C \#^\omega \quad : \text{where } C \in \Gamma^* Q \Gamma^+
\end{tikzcd}$$

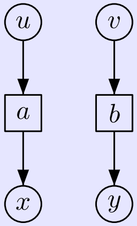
$$(v = 0 \wedge y = \#)\,\mathsf{W}\left(v = 1 \wedge (v = 1 \wedge y = \#)\,\mathsf{W}\,(v = 0 \wedge y \in \Gamma^* Q \Gamma^+ \#^\omega)\right)$$

where

$$y \in \Gamma^* Q \Gamma^+ \#^\omega \quad \overset{\text{def}}{=} \quad y \in \Gamma\,\mathsf{U}\left(y \in Q \wedge \mathsf{X}\big(y \in \Gamma\,\mathsf{U}\,(y \in \Gamma \wedge \mathsf{X}\,\mathsf{G}\,y = \#)\big)\right)$$

Undecidability proof 1

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



$$\begin{tikzcd} u \arrow[d] & v \arrow[d] \\ \boxed{a} \arrow[d] & \boxed{b} \arrow[d] \\ x & y \end{tikzcd}$$

$0^q1^p0\cdots$: $n(v) = p$

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

$$(v = 0 \wedge y = \#)\,\mathsf{W}\left(v = 1 \wedge (v = 1 \wedge y = \#)\,\mathsf{W}\,(v = 0 \wedge y \in \Gamma^*Q\Gamma^+\#^\omega)\right)$$

where

$$y \in \Gamma^*Q\Gamma^+\#^\omega \quad \overset{\text{def}}{=} \quad y \in \Gamma\,\mathsf{U}\left(y \in Q \wedge \mathsf{X}\big(y \in \Gamma\,\mathsf{U}\,(y \in \Gamma \wedge \mathsf{X}\,\mathsf{G}\,y = \#)\big)\right)$$

Undecidability proof 2

SPEC$_2$: processes a and b must start with the first configuration

$$\begin{tikzcd}
u \arrow[d] & v \arrow[d] & 0^q10\cdots \quad : n(v)=1 \\
\boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \\
x & y & \#^{q+1}C_1\#^\omega
\end{tikzcd}$$

$$v = 0 \,\mathsf{W}\, \Big(v = 1 \wedge \mathsf{X}\big(v = 0 \longrightarrow y \in C_1\#^\omega\big)\Big)$$

Undecidability proof 2

SPEC$_2$: processes a and b must start with the first configuration

$$\begin{tikzcd}
u \arrow[d] & v \arrow[d] & 0^q10\cdots \quad : n(v) = 1 \\
\boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \\
x & y & \#^{q+1}C_1\#^\omega
\end{tikzcd}$$

$$v = 0 \,\mathsf{W} \left(v = 1 \wedge \mathsf{X}\big(v = 0 \longrightarrow y \in C_1\#^\omega\big)\right)$$

Undecidability proof 3

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

$$\begin{tikzcd}
0^q1^p0\cdots & u \arrow[d] & v \arrow[d] & 0^q1^p0\cdots \\
& \boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \\
\#^{q+p}C\#^{\omega} & x & y & \#^{q+p}C\#^{\omega}
\end{tikzcd}$$

$$n(u) = n(v) \longrightarrow \mathsf{G}(x = y)$$

where

$$n(u) = n(v) \quad \overset{\text{def}}{=} \quad (u = v = 0)\ \mathsf{U}\ (u = v = 1 \wedge (u = v = 1\ \mathsf{U}\ u = v = 0))$$

Undecidability proof 3

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

$$\begin{tikzcd}
0^q1^p0\cdots & u \arrow[d] & v \arrow[d] & 0^q1^p0\cdots \\
 & \boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \\
\#^{q+p}C\#^{\omega} & x & y & \#^{q+p}C\#^{\omega}
\end{tikzcd}$$

$$n(u) = n(v) \longrightarrow \mathsf{G}(x = y)$$

where

$$n(u) = n(v) \quad \overset{\text{def}}{=} \quad (u = v = 0)\ \mathsf{U}\ (u = v = 1 \land (u = v = 1\ \mathsf{U}\ u = v = 0))$$

Undecidability proof 4

SPEC$_4$: if $n(u) = n(v) + 1$ are synchronized then $C_y \vdash C_x$

$$\begin{tikzcd}
0^q1^{p+1}0\cdots & u \arrow[d] & v \arrow[d] & 0^{q+1}1^p0\cdots \\
 & \boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \\
\#^{q+p+1}C_x\#^\omega & x & y & \#^{q+p+1}C_y\#^\omega
\end{tikzcd}$$

$$n(u) = n(v) + 1 \quad \longrightarrow \quad x = y \, \mathsf{U} \left(\mathrm{Trans}(y, x) \wedge \mathsf{X}^3 \, \mathsf{G} \, x = y \right)$$

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

$$\bigvee_{(p,a,q,b,\leftarrow) \in T, c \in \Gamma} (y = cpa \wedge x = qcb) \quad \vee \quad \bigvee_{(p,a,q,b,\rightarrow) \in T, c \in \Gamma} (y = pac \wedge x = bqc)$$

$$\vee \quad \bigvee_{(p,a,q,b,\rightarrow) \in T} (y = pa\# \wedge x = bq\square)$$

Undecidability proof 4

SPEC$_4$: if $n(u) = n(v) + 1$ are synchronized then $C_y \vdash C_x$

$$\begin{tikzcd}
0^q 1^{p+1} 0 \cdots & u \arrow[d] & v \arrow[d] & 0^{q+1} 1^p 0 \cdots \\
& \boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \\
\#^{q+p+1} C_x \#^\omega & x & y & \#^{q+p+1} C_y \#^\omega
\end{tikzcd}$$

$$n(u) = n(v) + 1 \quad \longrightarrow \quad x = y \,\mathsf{U}\, \Big(\mathrm{Trans}(y, x) \wedge \mathsf{X}^3\, \mathsf{G}\, x = y\Big)$$

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

$$\bigvee_{(p,a,q,b,\leftarrow)\in T, c\in\Gamma} (y = cpa \wedge x = qcb) \quad \vee \bigvee_{(p,a,q,b,\rightarrow)\in T, c\in\Gamma} (y = pac \wedge x = bqc)$$

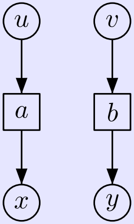
$$\vee \bigvee_{(p,a,q,b,\rightarrow)\in T} (y = pa\# \wedge x = bq\Box)$$

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


$$\begin{tikzcd} u \arrow[d] & v \arrow[d] \\ \boxed{a} \arrow[d] & \boxed{b} \arrow[d] \\ x & y \end{tikzcd}$$

Corollary

Specifications 1-4 and 5: G $x \neq \mathrm{stop}$ 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

$$\begin{tikzcd}
0^q 1 0 \cdots & u \arrow[d] & v \arrow[d] \\
\mathsf{SPEC}_2 & \boxed{a} \arrow[d] & \boxed{b} \arrow[d] \\
\#^{q+1} C_1 \#^{\omega} & x & y
\end{tikzcd}$$

Corollary

Specifications 1-4 and 5: $\mathsf{G}\, x \neq \mathrm{stop}$ 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

$$\begin{tikzcd}
0^{q+1}1^{p}0\cdots & \textcircled{u} \arrow[d] & \textcircled{v} \arrow[d] \\
\text{Induction} & \boxed{a} \arrow[d] & \boxed{b} \arrow[d] \\
\#^{q+p+1}C_p\#^{\omega} & \textcircled{x} & \textcircled{y}
\end{tikzcd}$$

Corollary

Specifications 1-4 and 5: $\mathsf{G}\, x \neq \mathrm{stop}$ 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

$$\begin{tikzcd}
0^{q+1}1^{p}0\cdots & u \arrow[d] & v \arrow[d] & 0^{q+1}1^{p}0\cdots \\
\text{Induction} & \boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \text{SPEC}_3 \\
\#^{q+p+1}C_p\#^{\omega} & x & y & \#^{q+p+1}C_p\#^{\omega}
\end{tikzcd}$$

Corollary

Specifications 1-4 and 5: $\mathsf{G}\, x \neq \mathrm{stop}$ 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

$$\begin{tikzcd}
0^q 1^{p+1} 0 \cdots & u \arrow[d] & v \arrow[d] & 0^{q+1} 1^p 0 \cdots \\
\mathsf{SPEC}_4 & \boxed{a} \arrow[d] & \boxed{b} \arrow[d] & \mathsf{SPEC}_3 \\
\#^{q+p+1} C_{p+1} \#^{\omega} & x & y & \#^{q+p+1} C_p \#^{\omega}
\end{tikzcd}$$

Corollary

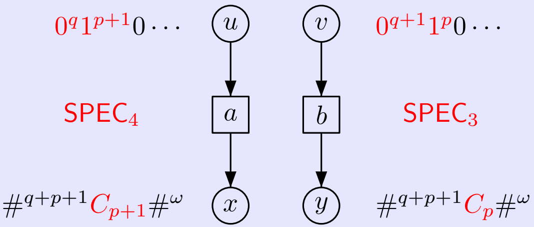
Specifications 1-4 and 5: $\mathsf{G}\, x \neq \mathrm{stop}$ are implementable iff the Turing machine does not halt starting from the empty tape.

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



Corollary

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

Decidability of distributed synthesis

Some examples

$$\begin{tikzcd}
u \arrow[d] & v \arrow[d] \\
\boxed{a} \arrow[d] & \boxed{b} \arrow[d] \\
x & y
\end{tikzcd}$$

Undecidable

$$\begin{tikzcd}
u \arrow[d] & & v \arrow[d] \\
\boxed{a} \arrow[d] \arrow[r] & z \arrow[r] & \boxed{b} \arrow[d] \\
x & & y
\end{tikzcd}$$

Decidable

$$\begin{tikzcd}
u \arrow[d] & w \arrow[dl] & & v \arrow[d] \\
\boxed{a} \arrow[d] \arrow[rr] & & z \arrow[r] & \boxed{b} \arrow[d] \\
x & & & y
\end{tikzcd}$$

Undecidable

Decidability

Pipeline

$$\begin{tikzcd}
x \arrow[r] & \boxed{a_1} \arrow[r] \arrow[d] & y_1 \arrow[r] & \boxed{a_2} \arrow[r] \arrow[d] & y_2 \arrow[r] & \boxed{a_3} \arrow[r] \arrow[d] & y_3 \arrow[r] & \boxed{a_4} \arrow[d] \\
& z_1 & & z_2 & & z_3 & & z_4
\end{tikzcd}$$

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.

$\Longrightarrow$ non-elementary lower bound for the synthesis problem.

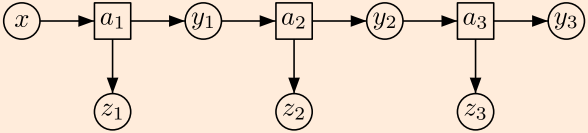
Decidability

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 chain


$$\begin{tikzcd}
x \arrow[r] & a_1 \arrow[r] \arrow[d] & y_1 \arrow[r] & a_2 \arrow[r] \arrow[d] & y_2 \arrow[r] & a_3 \arrow[r] \arrow[d] & y_3 \\
 & z_1 & & z_2 & & z_3 &
\end{tikzcd}$$

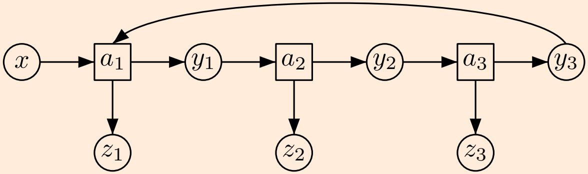
Decidability

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


$$\begin{tikzcd}
x \arrow[r] & \boxed{a_1} \arrow[r] \arrow[d] & y_1 \arrow[r] & \boxed{a_2} \arrow[r] \arrow[d] & y_2 \arrow[r] & \boxed{a_3} \arrow[r] \arrow[d] & y_3 \arrow[lllll, bend right] \\
 & z_1 & & z_2 & & z_3 &
\end{tikzcd}$$

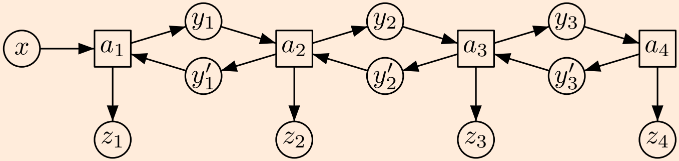
Decidability

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.

two-way chain



1-delay synchronous semantics

Example

$$\begin{tikzcd} u \arrow[r] & \boxed{a} \arrow[d] & \\ & x \arrow[d] & \\ v \arrow[r] & \boxed{b} \arrow[r] & z \end{tikzcd}$$

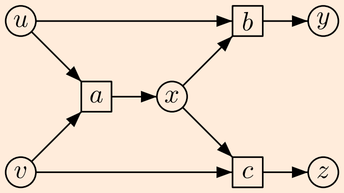
Programs: $f_x : Q_u^* \to Q_x$ and $f_z : (Q_x \times Q_v)^* \to Q_z$.

- 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 $\begin{cases} x_{n+1} = f_x(u_1 \cdots u_n) \\ z_{n+1} = f_z((x_1, v_1) \cdots (x_n, v_n)) \end{cases}$ for all $n > 0$.

Decidability

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

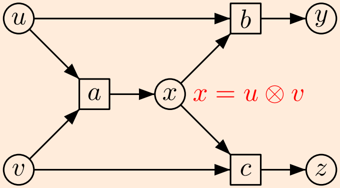


Pnueli-Rosner (FOCS'90)

- 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

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

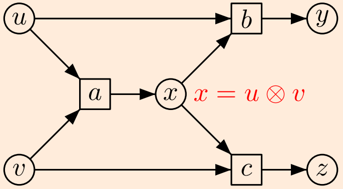


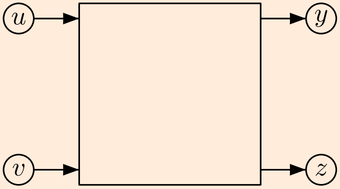
Pnueli-Rosner (FOCS'90)

- 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

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



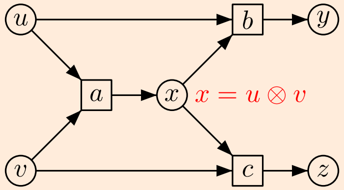


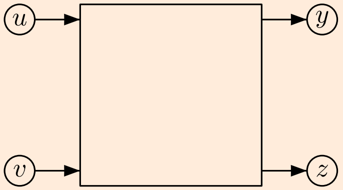
Pnueli-Rosner (FOCS'90)

- 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

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





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.

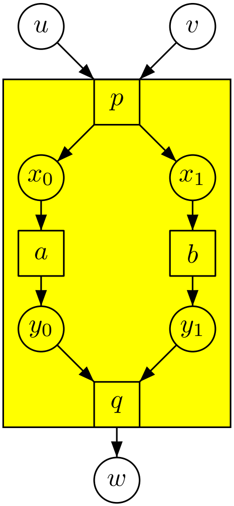
Information fork criterion (Finkbeiner–Schewe LICS ’05)

$$\begin{tikzcd}
u \arrow[dr] & & v \arrow[dl] \\
& \boxed{p} \arrow[dl] \arrow[dr] & \\
x_0 \arrow[d] & & x_1 \arrow[d] \\
\boxed{a} \arrow[d] & & \boxed{b} \arrow[d] \\
y_0 \arrow[dr] & & y_1 \arrow[dl] \\
& \boxed{q} \arrow[d] & \\
& w &
\end{tikzcd}$$

Information fork criterion (Finkbeiner–Schewe LICS '05)

$$\begin{tikzcd}
u \arrow[dr] & & v \arrow[dl] \\
& \boxed{p} \arrow[dl] \arrow[dr] & \\
x_0 \arrow[d] & & x_1 \arrow[d] \\
\boxed{a} \arrow[d] & & \boxed{b} \arrow[d] \\
y_0 \arrow[dr] & & y_1 \arrow[dl] \\
& \boxed{q} \arrow[d] & \\
& w &
\end{tikzcd}$$

Information fork criterion (Finkbeiner–Schewe LICS '05)



Outline

Uniformly well connected architectures

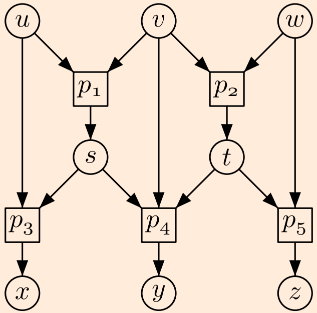
Definition

For an output variable y, 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 View(y) to y for each output variable y.

Example



Uniformly well connected architectures

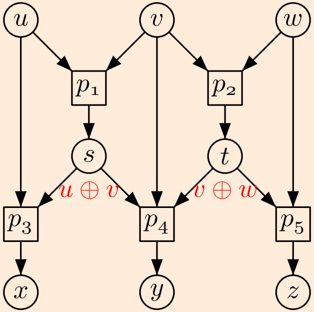
Definition

For an output variable y, 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 View(y) to y for each output variable y.

Example



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 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

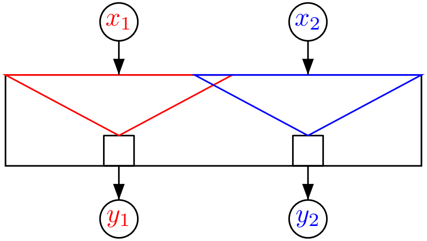
Reduction to the multicast problem in Network Information Flow.
The multicast problem is NP-complete (Rasala Lehman-Lehman 2004).

Uncomparable 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.

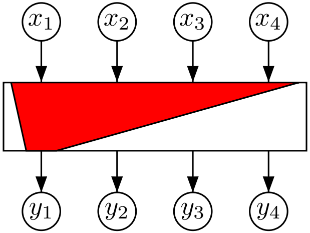


Uncomparable information

Definition

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

Otherwise it is said to have preordered information.

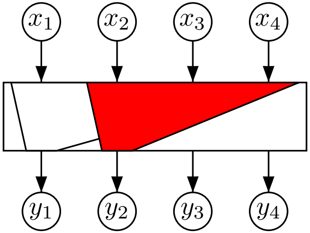


Uncomparable information

Definition

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

Otherwise it is said to have preordered information.

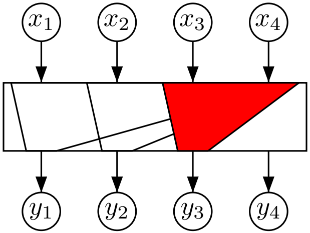


Uncomparable information

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.

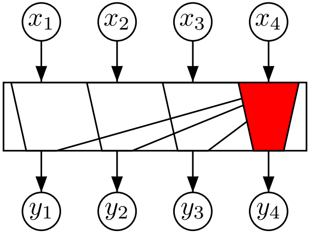


Uncomparable information

Definition

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

Otherwise it is said to have preordered information.

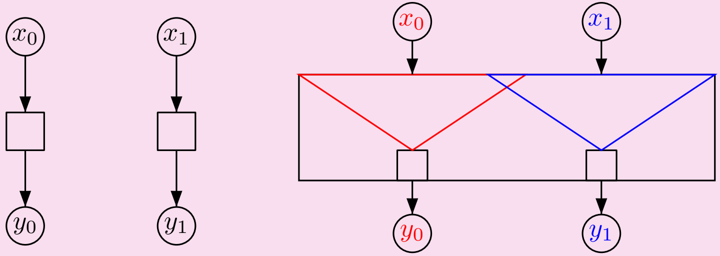


Uncomparable information yields undecidability

Theorem

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

Proof.

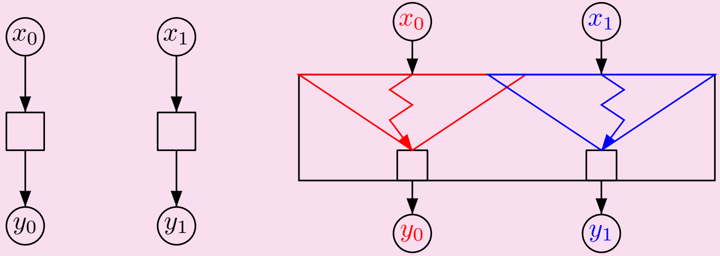


Uncomparable information yields undecidability

Theorem

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

Proof.

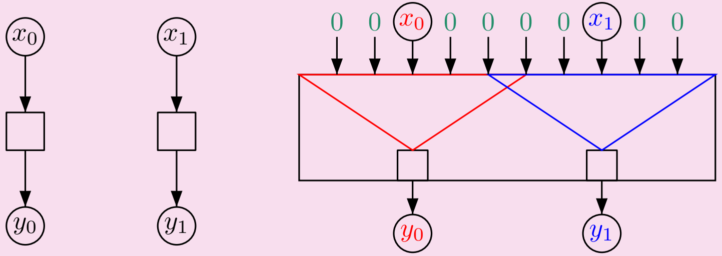


Uncomparable information yields undecidability

Theorem

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

Proof.

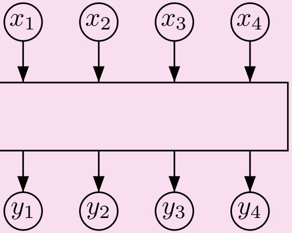


Uniformly well connected architectures

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)

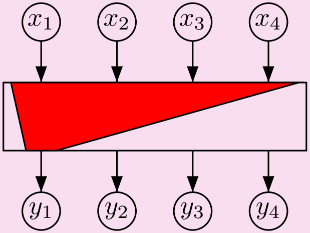
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

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)

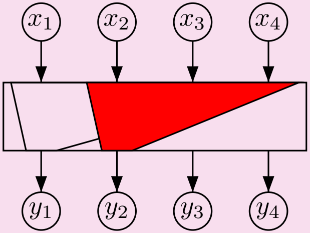
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

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)

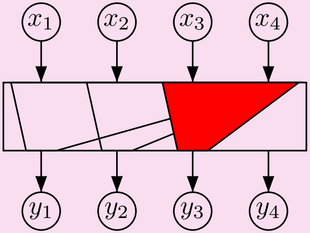
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

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)

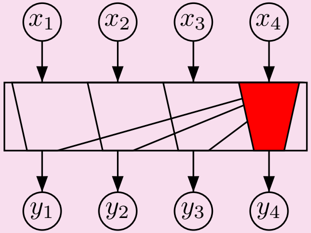
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

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)

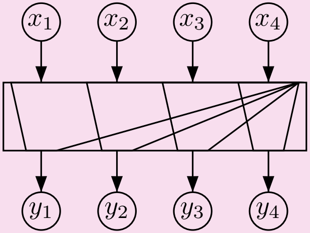
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

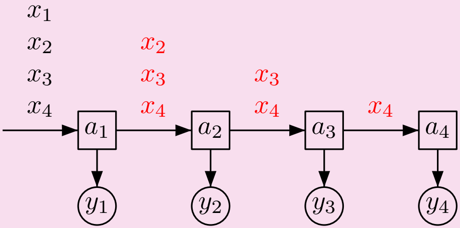
Uniformly well connected architectures

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)

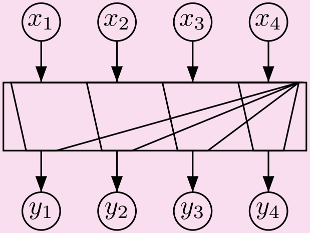
The synthesis problem is decidable for pipeline architectures and CTL* specifications on all variables.

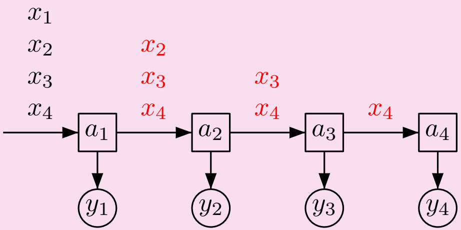
Uniformly well connected architectures

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.

Robust specifications

Definition

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

Theorem

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

Proof.

Robust specifications

Definition

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

Theorem

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

Proof.

Robust specifications

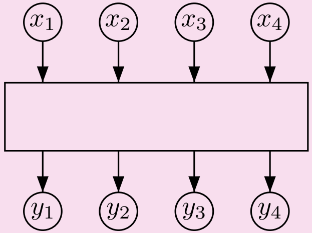
Definition

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

Theorem

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

Proof.



Robust specifications

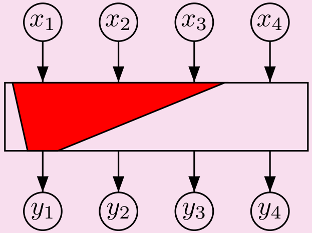
Definition

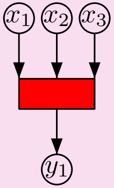
A specification φ is robust if it can be written $\varphi = \bigvee \bigwedge_{z \in \mathrm{Out}} \varphi_z$ where φ_z depends only on View$(z) \cup \{z\}$.

Theorem

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

Proof.





Robust specifications

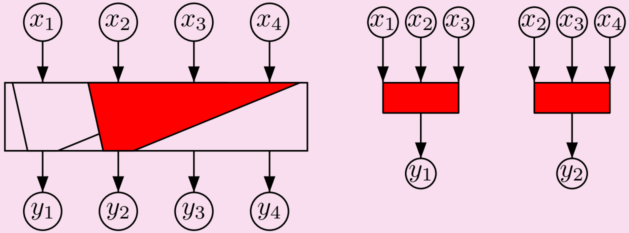
Definition

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

Theorem

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

Proof.



Robust specifications

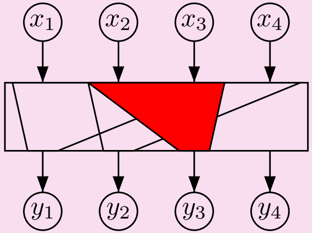
Definition

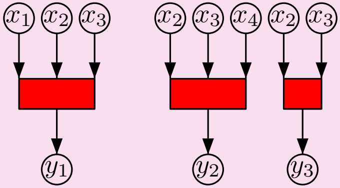
A specification φ is robust if it can be written $\varphi = \bigvee \bigwedge_{z \in \mathrm{Out}} \varphi_z$ where φ_z depends only on View$(z) \cup \{z\}$.

Theorem

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

Proof.





Robust specifications

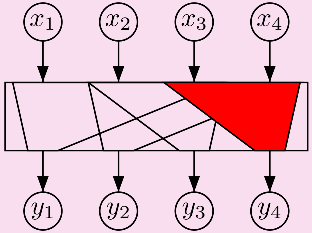
Definition

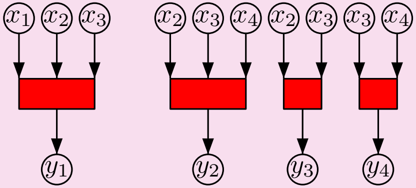
A specification φ is robust if it can be written $\varphi = \bigvee \bigwedge_{z \in \mathrm{Out}} \varphi_z$ where φ_z depends only on View$(z) \cup \{z\}$.

Theorem

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

Proof.



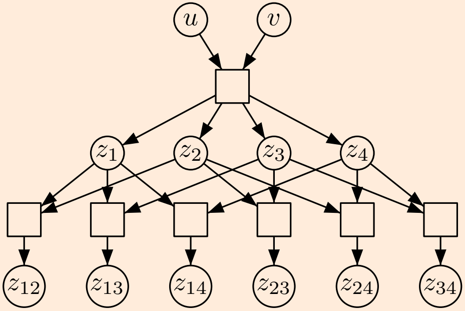


Well-connected architectures

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.

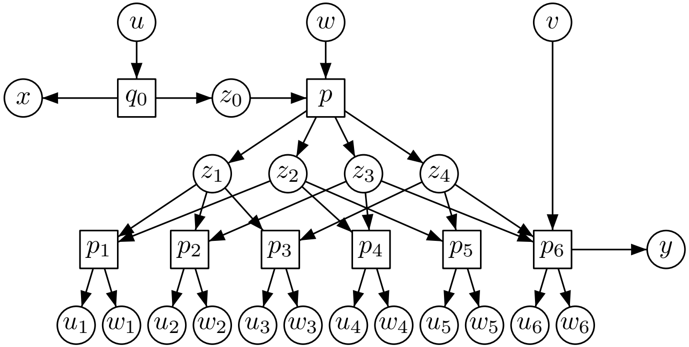
Corollary

One can decide whether an architecture is well-connected in polynomial time.

Well connected preordered architectures

Theorem

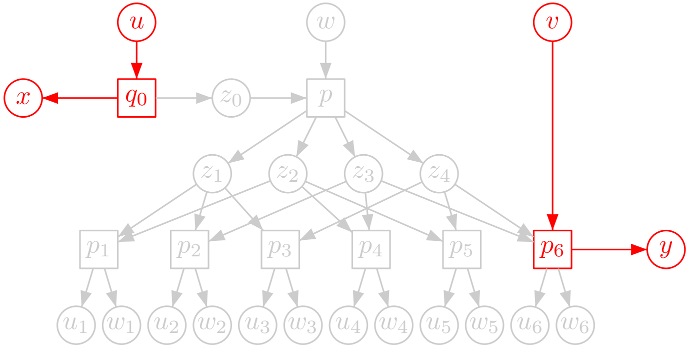
The synthesis problem for LTL specifications and well connected architectures with preordered information is undecidable.



Well connected preordered architectures

Theorem

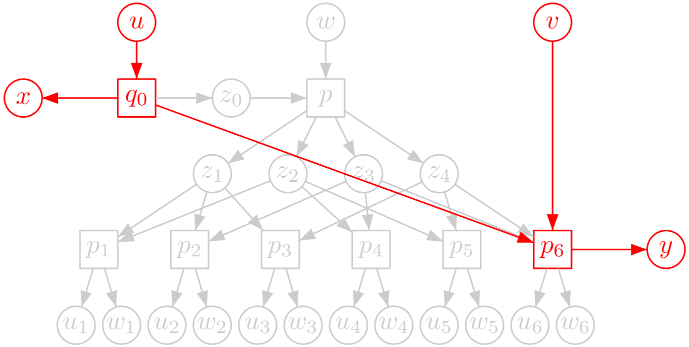
The synthesis problem for LTL specifications and well connected architectures with preordered information is undecidable.



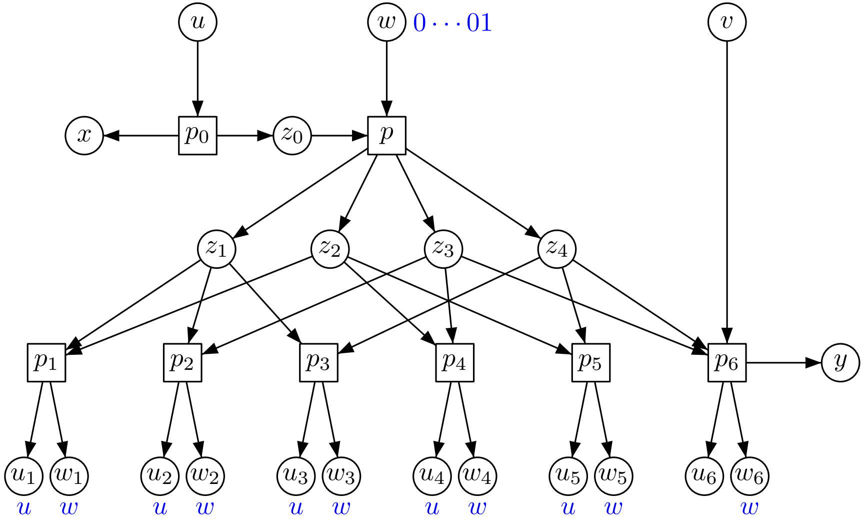
Well connected preordered architectures

Theorem

The synthesis problem for LTL specifications and well connected architectures with preordered information is undecidable.

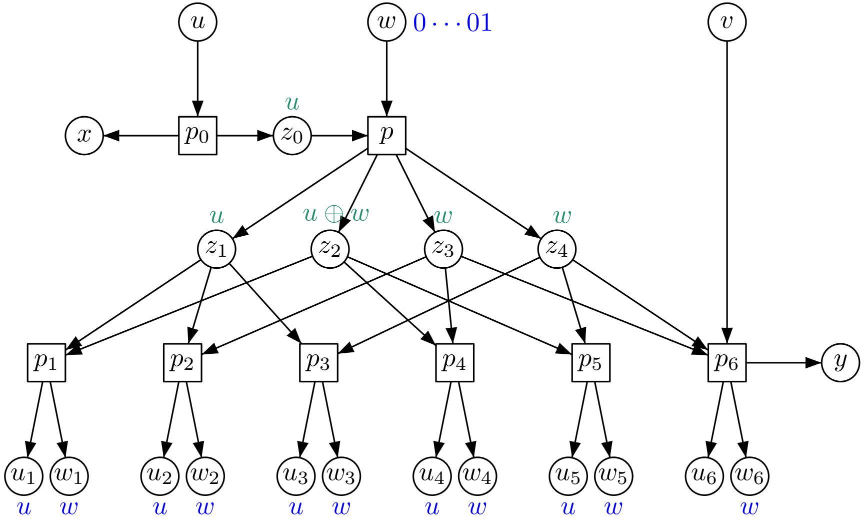


Specification and routing



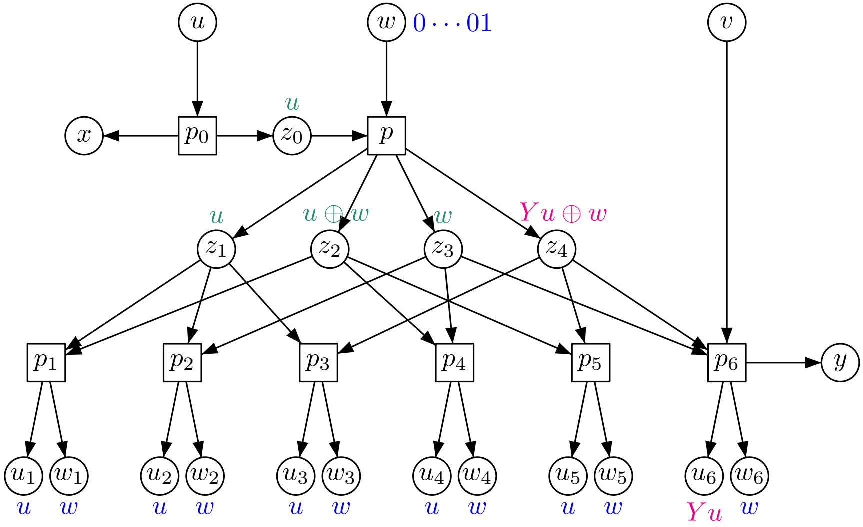
u
w
$0 \cdots 01$
v
x
p_0
z_0
p
z_1
z_2
z_3
z_4
p_1
p_2
p_3
p_4
p_5
p_6
y
u_1 w_1 u_2 w_2 u_3 w_3 u_4 w_4 u_5 w_5 u_6 w_6
u w u w u w u w u w w

Specification and routing

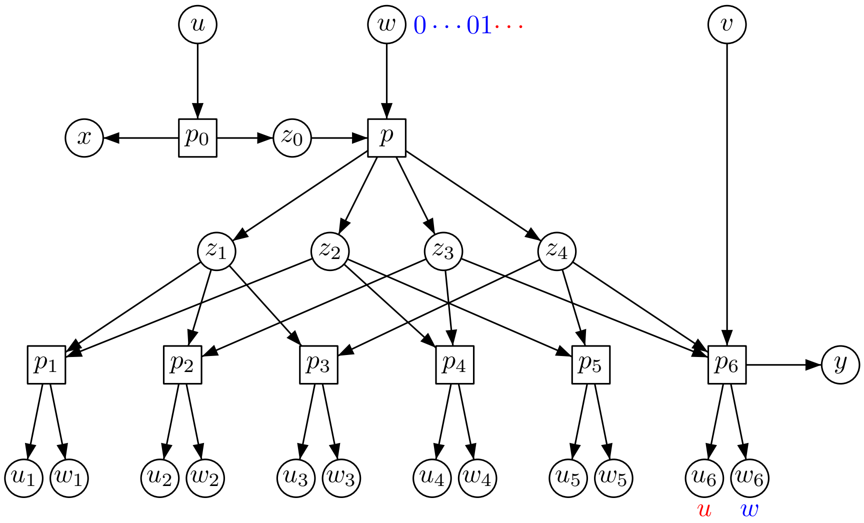


u
w
$0\cdots01$
v
x
p_0
u
z_0
p
u
$u \oplus w$
w
w
z_1
z_2
z_3
z_4
p_1
p_2
p_3
p_4
p_5
p_6
y
u_1 w_1 u_2 w_2 u_3 w_3 u_4 w_4 u_5 w_5 u_6 w_6
u w u w u w u w u w w

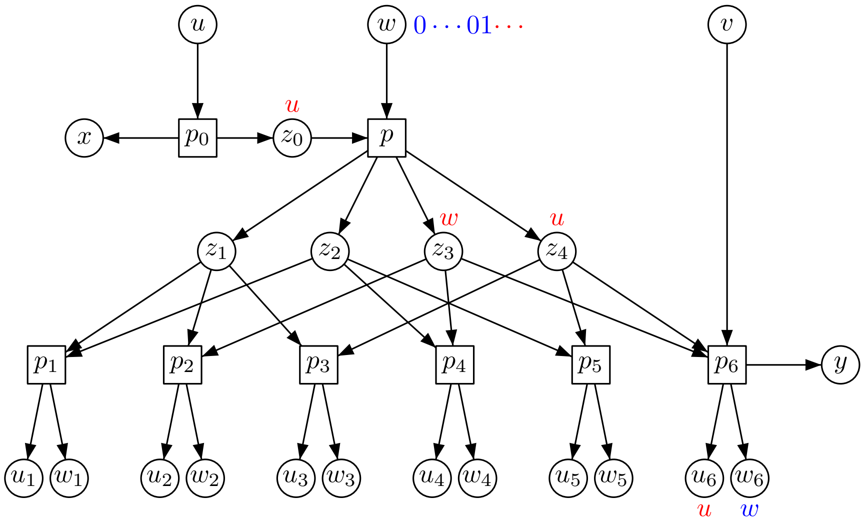
Specification and routing



Specification and routing



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.