Distributed synthesis: synchronous and asynchronous semantics

Paul Gastin
LSV
ENS de Cachan & CNRS
Paul.Gastin@lsv.ens-cachan.fr
EPIT, May 31st, 2006

Outline

Control for sequential systems

Control for distributed systems

Synchronous semantics

Asynchronous semantics

Open / Reactive system

Example: Elevator

Transition system

States:

- position of the cabin
- flag is_open for each door
- flag is_called for each level
- number of persons in the cabin

Events:

<table>
<thead>
<tr>
<th>Event</th>
<th>$\Sigma_{\text{uc}}$</th>
<th>$\Sigma_{\text{uo}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>call level $i$</td>
<td>enter/exit cabin</td>
<td></td>
</tr>
<tr>
<td>open/close door</td>
<td>move 1 level up/down</td>
<td></td>
</tr>
</tbody>
</table>

We get easily a finite and deterministic transition system.
**Specification**

Linear time: LTL, FO, MSO, regular, ...
- Safety: $G(\text{level} \neq i \rightarrow \text{is\_closed})$
- Liveness: $G(\text{is\_called}, \rightarrow F(\text{level} = i \land \text{is\_open}))$

Branching time: CTL, CTL*, $\mu$-calculus, ...
- $AG(\text{call})^T$ (call is uncontrollable)
- $AG EF(\text{level} = 0 \land \text{is\_open})$

**Controller**

Under full state-event observation
- Controller: $f : Q(SQ)^* \rightarrow 2^E$ with $\Sigma_{uc} \subseteq f(x)$ for all $x \in Q(SQ)^*$.
- Controlled behavior: $q_0, a_1, q_1, a_2, q_2, \ldots$ with $(q_{i-1}, a_i, q_i) \in \delta$ and $a_i \in f(q_0a_1q_1 \cdots q_{i-1})$ for all $i > 0$.
- Controlled execution tree: $t : D^* \rightarrow Q \times Q$ with
  - $t(e) = (a, q)$ ($a \in \Sigma$ fixed arbitrarily)
  - for all $x = d_1 \cdots d_n \in D^*$ with $t(d_1 \cdots d_i) = (a_i, q_i)$, we have:
  - $t(\text{seas}(x)) = \{(a, q) | a \in f(q_0a_1q_1 \cdots a_nq_n) \land (q_0, a, q) \in \delta\}$.

Under full event observation
- Controller: $f : \Sigma^* \rightarrow 2^E$ with $\Sigma_{uc} \subseteq f(x)$ for all $x \in \Sigma^*$.
- Remark: same as full state-event observation if the system is deterministic.

Under partial event observation
- Controller: $f : \Sigma_u^* \rightarrow 2^E$ with $\Sigma_{uc} \subseteq f(x)$ for all $x \in \Sigma^*$.
- Controlled behavior: $q_0, a_1, q_1, a_2, q_2, \ldots$ with $(q_{i-1}, a_i, q_i) \in \delta$ and $a_i \in f \circ \Pi_{u_0}(a_1 \cdots a_{i-1})$ for all $i > 0$.

**Control problem**

- Inputs from $E$ to $S$
- Outputs to $E$
- Controller $C$ enables/disables actions
- Observation
- Specification $\phi$

Two problems
- Control: Given a system $S$ and a specification $\phi$, decide whether there exists a controller $C$ such that $S \otimes C \models \phi$.
- Synthesis: Given a system $S$ and a specification $\phi$, build a controller $C$ (if one exists) such that $S \otimes C \models \phi$.

**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.

Control problem
Given a system $S$ and a specification $\phi$, does there exist a controller $C$ such that $L(C \otimes S) \subseteq L(\phi)$?

Theorem
If the system is finite state and the specification is regular then the control problem is decidable.
Moreover, when $(S, \phi)$ is controllable, we can synthesize a finite state controller.
Synthesis of reactive programs

Implementability problem

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 $\varphi = (y = 1) \leftrightarrow F(x = 1)$
  - $\forall \vec{x} \exists \vec{y} \varphi$ is valid.
  - $\varphi$ is not implementable by a reactive program.

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

The specification $\varphi \in$ LTL is implementable iff the formula
$$\mathcal{A}\varphi \land \mathcal{G}(\bigwedge_{a \in Q_x} \mathcal{E}X(x = a))$$
is satisfiable.

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

Synthesis of reactive programs

Pnueli-Rosner 89

Given a system $S$ (with accepting states) and a specification $K \subseteq \Sigma^*$, does there exist a controller $C$ such that $L(C \otimes S) = K$?

Other results
- control under partial observation
- maximal controllable sub-specification
- generalization to infinite behaviors (Thistle - Wonham)

Implementability problem

Given a system $S$, its controllability problem is decidable.
When $(S, K)$ is controllable, we can synthesize a finite state controller.

Theorem:

$(S, \text{Pref}(K))$ is controllable iff $\text{Pref}(K) \cdot \Sigma_{\text{inc}} \cap \text{Pref}(L(S)) \subseteq \text{Pref}(K)$.

$(S, K)$ is controllable without deadlock iff
- $\text{Pref}(K) \cdot \Sigma_{\text{inc}} \cap \text{Pref}(L(S)) \subseteq \text{Pref}(K)$
- $\text{Pref}(K) \cap L(S) = K$.

If $S$ is finite state and $K$ regular, then the control problem is decidable.
When $(S, K)$ is controllable, we can synthesize a finite state controller.

Synthesis of reactive programs

Implementability problem

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 $\varphi$?

- $Q_x$: domain for input variable $x$
- $Q_y$: domain for output variable $y$
- Program: $f : Q_x^+ \rightarrow Q_y$
- Input: $x_1, x_2, \ldots \in Q_x^*$
- Behavior: $(x_1, y_1)(x_2, y_2)(x_3, y_3)\ldots$ with $y_n = f_1(x_1 \cdots x_n)$ for all $n > 0$.
Program synthesis versus System control

Equivalence
The implementability problem for

\[ x \rightarrow y \]

is equivalent to the control problem for the system

\[ Q_x \rightarrow Q_y \]

Distributed control

Inputs from \( E \)  
Outputs to \( E \)

Controlled open distributed system \( S \)

\[ C_1 \rightarrow S_1 \rightarrow S_2 \rightarrow C_2 \]
\[ C_3 \rightarrow S_3 \rightarrow S_4 \rightarrow C_4 \]

 Specification \( \varphi \)

Two problems, again
- 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.

Peterson-Reif 1979, Pnueli-Rosner 1990

In general, the problems are undecidable.
Distributed systems with shared variables

Distributed system/plant/arena
- \( \mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W) \) architecture.
- \( Q_x \) (finite) domain for each variable \( x \in \mathcal{V} \).
- \( \delta_a \subseteq Q_R(a) \times Q_W(a) \) legal actions/moves for process/player \( a \in \mathcal{P} \).
- \( q^0 \in Q_V \) initial state

where \( Q_I = \prod_{x \in I} Q_x \) for \( I \subseteq \mathcal{V} \).

Outline

Control for sequential systems

Control for distributed systems

- Synchronous semantics

Asynchronous semantics

Distributed Synthesis

Problem
- Given a distributed system and a specification
- Problem existence/synthesis of programs/strategies for the processes/players such that the system satisfies the specification (whatever the environment/opponent does).

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\nfinite or infinite memory

Pnueli-Rosner (FOCS’90)

Pipeline

Restrictions
- Unique writer: \(|W^{-1}(x)| = 1\) for all \( x \in \mathcal{V} \)
- Unique reader: \(|R^{-1}(x)| = 1\) for all \( x \in \mathcal{V} \)
- Acyclic graph (0-delay)
- No restrictions on moves: \( \delta_a = Q_R(a) \times Q_W(a) \) for all \( a \in \mathcal{P} \).
- Synchronous behaviors: \( q_0^0 q_1^1 q_2^2 \cdots \) where \( q^i \in Q_V \) are global states.
- program with local memory: \( f_a : Q^*_R(a) \to Q_W(a) \) for all \( a \in \mathcal{P} \).
- Specification: LTL over input and output variables only.
  - Input variables: \( \text{In} = W(\text{environment}) \)
  - output variables: \( \text{Out} = R(\text{environment}) \)
0-delay synchronous semantics

Example

![Diagram](image)

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

- Input: \( \begin{pmatrix} u_1 & u_2 & u_3 & \cdots \\ v_1 & v_2 & v_3 & \cdots \end{pmatrix} \in (Q_u \times Q_v)^* \). 
- 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 \( n > 0 \).

Undecidability

Architecture \( A_0 \)

![Diagram](image)

Theorem (Pnueli-Rosner FOCS’90)

The synthesis problem for architecture \( A_0 \) and LTL (or CTL) specifications is undecidable.

Proof

Reduction from the halting problem on the empty tape.

Undecidability proof 1

SPEC1: processes \( a \) and \( b \) must output configurations

![Diagram](image)

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

where

\( y \in \Gamma^*Q\Gamma^+\#^\omega \ \overset{\text{def}}{=} \ y \in (Q \land X) (y \in \Gamma \land X \Gamma \land y = \#)) \)

Undecidability proof 2

SPEC2: processes \( a \) and \( b \) must start with the first configuration

![Diagram](image)

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

where

\( y \in \Gamma^*Q\Gamma^+\#^\omega \ \overset{\text{def}}{=} \ y \in (Q \land X) (y \in \Gamma \land X \Gamma \land y = \#)) \)
**Undecidability proof 3**

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

\[0^p1^p0\ldots \rightarrow a \quad 0^p1^p0\ldots \rightarrow b\]

\[\#^pC\#^\omega \quad \#^pC\#^\omega\]

$n(u) = n(v) \rightarrow G(x = y)$

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**

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

\[0^p1^{p+1}0\ldots \rightarrow a \quad 0^p1^{p+1}0\ldots \rightarrow b\]

\[\#^pC_x^{p+1}\#^\omega \quad \#^pC_x^{p+1}\#^\omega\]

$n(u) = n(v) + 1 \rightarrow x = y \cup (\text{Trans}(y, x) \land X^5 G \land x = y)$

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

\[\bigvee_{(p,a,q,b,\ldots) \in T} (y = cpq \land x = qeb) \lor \bigvee_{(p,a,q,b,\ldots) \in T} (y = pac \land x = bqc) \lor \bigvee_{(p,a,q,b,\ldots) \in T} (y = pax \land x = bq\sqrt{)}\]

**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**

\[0^p1^{p+1}0\ldots \rightarrow a \quad 0^p1^{p+1}0\ldots \rightarrow b\]

\[\text{Induction} \quad \text{SPEC}_3\]

\[\#^{p+1}C_p^{p+1}\#^\omega \quad \#^{p+1}C_p^{p+1}\#^\omega\]

**Corollary**

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

**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 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**.

More undecidable architectures

**Exercises**
1. Show that the architecture below is undecidable.

2. Show that the undecidability results also hold for CTL specifications.

Uncomparable information

**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 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 yields undecidability

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

Proof for LTL specifications

Decidability proof 1

From distributed to global
If \( f_y : Q^+_y \rightarrow Q_y \) and \( f_z : Q^+_z \rightarrow Q_z \) are local (distributed) strategies then we can define an equivalent global strategy \( h = f_y \otimes f_z : Q^+_y \rightarrow Q_y \times Q_z \) by

\[
h(x_1 \cdots x_n) = (y_n, f_z(y_1 \cdots y_n)) \quad \text{where} \quad y_i = f_y(x_1, \cdots, x_i).
\]

From global to distributed
\( z \) should only depend on \( y \).

We cannot transmit \( x \) to \( y \) if \( |Q_y| < |Q_z| \).

We have to check whether there exists a global strategy that can be distributed.

Decidability

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

Decidability proof 2

Proof
1. We first solve the global game: We obtain an ND tree-automaton \( \mathcal{A} \) accepting the global strategies \( h : Q^+_y \rightarrow Q_y \times Q_z \) that implement the specification.

   Easily obtained from a ND tree automaton for the specification.

2. We build from \( \mathcal{A} \) an alternating tree automaton \( \mathcal{A}’ \) accepting a local strategy

\[
f_z : Q^+_y \rightarrow Q_z \quad \text{if there exists a local strategy} \quad f_y : Q^+_y \rightarrow Q_y \quad \text{such that}
\]

\[
h = f_y \otimes f_z : Q^+_y \rightarrow Q_y \times Q_z \quad \text{is accepted by} \quad \mathcal{A}
\]
**Tree automata**

**Decidability proof 3**

Proof

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q^+_y \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q^+_z \rightarrow Q_z$ iff there exists a local strategy $f_y : Q^+_y \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q^+_z \rightarrow Q_y \times Q_z$ is accepted by $A$.

3. Transform the alternating TA $A'$ to an equivalent non-deterministic TA $A_1$ (Muller and Schupp 1985). Exponential blow-up.

4. Iterate and check the last automaton for emptiness.

**Decidability proof 4**

Proof

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q^+_y \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.

2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q^+_z \rightarrow Q_z$ iff there exists a local strategy $f_y : Q^+_y \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q^+_z \rightarrow Q_y \times Q_z$ is accepted by $A$.

3. Transform the alternating TA $A'$ to an equivalent non-deterministic TA $A_1$ (Muller and Schupp 1985). Exponential blow-up.

4. Iterate and check the last automaton for emptiness.
Decidability proof 4

1. We first solve the global game: We obtain an ND tree-automaton $A$ accepting the global strategies $h : Q_x^* \rightarrow Q_y \times Q_z$ that implement the specification. Easily obtained from a ND tree automaton for the specification.
2. We build from $A$ an alternating tree automaton $A'$ accepting a local strategy $f_z : Q_y^* \rightarrow Q_z$ iff there exists a local strategy $f_y : Q_y^* \rightarrow Q_y$ such that $h = f_y \otimes f_z : Q_y^* \rightarrow Q_y \times Q_z$ is accepted by $A$.
3. Transform the alternating TA $A'$ to an equivalent non deterministic TA $A_1$ (Muller and Schupp 1985). Exponential blow-up.
4. Iterate and check the last automaton for emptiness.

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.

1-delay synchronous semantics

Example

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

- Input: $u_1 \ u_2 \ u_3 \ldots \ v_1 \ v_2 \ v_3 \ldots$

- Behavior: $x_1 \ x_2 \ x_3 \ldots$

with $x_{n+1} = f_x(u_1 \ldots \ u_n)$ and $z_{n+1} = f_z((x_1, v_1) \ldots (x_n, v_n))$ for all $n > 0$. 

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.
Decidability

Adequately connected sub-architecture

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

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.

Uniformly well connected architectures

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

\[ P_1 \rightarrow P_2 \rightarrow P_3 \]

\[ P_4 \rightarrow P_5 \rightarrow P_6 \]

\[ P_7 \rightarrow P_8 \rightarrow P_9 \]

\[ P_{10} \rightarrow P_{11} \rightarrow P_{12} \]

Information fork criterion (Finkbeiner–Schewe LICS ’05)

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

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 \( \varphi = \bigvee_{x \in \text{Out}} \varphi_x \) where \( \varphi_x \) depends only on \( \text{View}(x) \cup \{ z \} \).

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

Proof.

Outline

Control for sequential systems

Control for distributed systems

Synchronous semantics

Asyncronous semantics
An example: Romeo and Juliet

Romeo and Juliet against the environment

Want to communicate through the same communication line.
At any time, one line is broken.
Environment looks where R&J are connected, and then, atomically, changes (possibly) the broken line.
Romeo/Juliet looks status of lines and, atomically, chooses where to connect.

Romeo and Juliet (continued)

Legal moves: \( \delta_a \subseteq Q_{R(a)} \times Q_{W(a)} \)

A distributed play of the asynchronous system, R & J against E

Distributed Behaviors

A play is a Mazurkiewicz (real) trace

- A finite play:

- Move: extension of the current Mazurkiewicz trace following the rules.
- The game is not “position based”, nor “turn based”.
- Winning condition: set of finite or infinite Mazurkiewicz traces \( W \subseteq R(\Sigma, D) \).
  Team 0 wins plays of \( W \) and loses plays of \( R(\Sigma, D) \setminus W \).

Romeo and Juliet

\( W \) imposes fairness conditions to the environment.
Memory for strategies

Memory
- Each player only has a partial view of the global history.
- Memoryless: move can depend only on the current state.
- Local memory: a player can remember its read history.

Causal memory (intuitively, maximal history a player can observe)
Players gather and forward as much information as possible.
but no global view, the choice for an action cannot depend on a concurrent event.

Finite abstraction of the causal memory

Distributed memory
A distributed memory is a mapping \( \mu : M(\Sigma, D) \rightarrow M \) satisfying the following equivalent properties:
1. \( \mu^{-1}(m) \) is recognizable for each \( m \in M \),
2. \( \mu \) is an abstraction of an asynchronous mapping (cf. Zielonka),
3. \( \mu \) can be computed in a distributed way
   (allowing additional contents inside existing communications (piggy-backing),
   but no extra communications).

Strategy with memory \( \mu \)
\( f_a : M \times Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} \)
the associated strategy is defined by
\[ f_a^\mu(t, q) = f_a(\mu(t), q) \]
If \( M \) is finite then \( f_a^\mu \) is a distributed strategy with finite memory.
If \( |M| = 1 \) then \( f_a^\mu \) is memoryless.

Winning strategies
Tuple \((f_a)_{a \in P_0} \) where \( f_a \) tells player \( a \in P_0 \) how to play.

<table>
<thead>
<tr>
<th>Memoryless</th>
<th>( f_a : Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local memory</td>
<td>( f_a : (Q_{R(a)})^* \rightarrow Q_{W(a)} \cup \text{Stop} )</td>
</tr>
<tr>
<td>Causal memory</td>
<td>( f_a : M(\Sigma, D) \times Q_{R(a)} \rightarrow Q_{W(a)} \cup \text{Stop} )</td>
</tr>
</tbody>
</table>

\( f \)-maximal \( f \)-plays
Given a strategy \( f = (f_a)_{a \in P_0} \), one looks at plays \( t \) which are
- consistent with \( f \): all \( a \)-moves played according to \( f_a \) (\( f \)-play).
- maximal: \( f \) predicts to \text{Stop} for all \( a \)-moves enabled at \( t \) with \( a \in P_0 \).

Winning strategies
A strategy \( f \) is winning in \( G \) if all \( f \)-maximal \( f \)-plays in \( G \) are in \( W \).

Embedding causal memory inside games

Proposition: PG-Lerman-Zeiloum (LATIN’04)
For a distributed game \( G \) and a distributed memory \( \mu \), one can build a game \( G^\mu \) such that
- team 0 has a WDS in \( G \) with memory \( \mu \)
- team 0 has a memoryless WDS in \( G^\mu \).

Proof.
\[ G^\mu = G \times \mu \]
From distributed to sequential games

Theorem: PG-Lerman-Zeitoun (LATIN’04)

Given a distributed game $(G, W)$, we can effectively build a finite sequential 2-players game $(\hat{G}, \hat{W})$ st. the following are equivalent:
- There exists a memoryless distributed WS for team 0 in $(G, W)$.
- There exists a memoryless WS for player 0 in $(\hat{G}, \hat{W})$.
- There exists a WS for player 0 in $(\hat{G}, \hat{W})$.

Moreover, if $W$ is recognizable then so is $\hat{W}$.

Naive idea Consider the game on the global transition system.
Main problem The controller has more information than its causal memory.
Solution
- The opponent controls the linearization to be played.
- Using reset moves, he can replay different linearizations for the same play.
- The winning condition $\hat{W}$ makes sure that the strategy followed by the controller is indeed distributed.

(Un)deciding games

Proposition: (Folklore)

Deciding whether team 0 has a distributed WS with causal memory is undecidable for rational winning conditions.

Proof. Simple reduction of the universality problem for rational trace languages.

Peterson-Reif Madhusudan–Thiagarajan Bernet–Janin–Walukiewicz

Deciding whether team 0 has a distributed WS with local memory is undecidable even:
- for reachability or safety winning conditions.
- with 3 players against the environment.

Series-parallel architectures

Theorem: PG-Lerman-Zeitoun (FSTTCS’04)

Distributed games with recognizable winning conditions are decidable for series-parallel systems and causal memory strategies.

Definition : let $\mathcal{A} = (\mathcal{P}, \mathcal{V}, R, W)$ be some architecture.
- $\mathcal{A}$ is a parallel product if $P = A \sqcup B$ with $R(a) \cap W(b) = \emptyset$ for all $(a, b) \in A \times B$.
- $\mathcal{A}$ is a serial product if $P = A \sqcup B$ with $R(a) \cap W(b) \neq \emptyset$ for all $(a, b) \in A \times B$.
- $\mathcal{A}$ is series-parallel if it can be obtained from singletons ($|\mathcal{P}| = 1$) using serial and parallel compositions.
- $\mathcal{A}$ is series-parallel iff the associated dependence relation does not contain a $P_4$: $a \rightarrow b \rightarrow c \rightarrow d$ as induced subgraph.
- Behaviors of series parallel architectures are series-parallel posets.

Proof outline

Team 0 has a WDS $\Rightarrow$ it has a WDS with a “small” distributed memory.

Induction on $\Sigma$.

Difficult case: serial product.

1. A WS on $A \sqcup B$ induces WS on the restrictions of the game to $A$ and $B$.
2. Replace the WS on $A, B$ by WS with small memory (induction).
3. Finally, glue together these WS on $A$ and $B$ to obtain a WS on $A \sqcup B$ using small memory.

Main problem

Team 0 must know on which small game it is playing.
Team 0 has to compute this information in a distributed way.
Madhusudan and Thiagarajan (Concur’02)

Setting
- Architecture: \( A = (P, V, R, W) \) with \( R(a) = W(a) \) for all \( a \in P \).
- Moves: \( \delta_a \) are built from local moves for variables \( \delta_{a,x} \subseteq Q_x \times Q_x \):
  \[
  \delta_a = \prod_{x \in R(a)} \delta_{a,x}
  \]
- Strategies with local memory: associated with variables and not with agents, and only predict the next actions and not the next state:
  \[
  f_x : Q^*_x \rightarrow 2^{R^{-1}(x)}
  \]
  action \( a \) is enabled by \( (f_x)_{x \in V} \) at some finite play \( t \) if
  \[
  \forall x \in R(a), \quad a \in f_x(\pi_Q, t))
  \]
- The environment decides which \( a \)-transition should be taken among the actions \( a \) enabled by the strategies.

Madhusudan and Thiagarajan (Concur’02)

Restricted control synthesis problem
- Given a distributed system and a recognizable specification,
- Question existence of a clocked and com-rigid non-blocking winning distributed strategy with local memory.
- clocked: \( f_x(w) \) only depends on \( |w| \).
- com-rigid: \( a, b \in f_x(w) \) implies \( R(a) = R(b) \).

Theorem
1. The restricted control synthesis problem is decidable.
2. It becomes undecidable if one of the red condition is dropped.

Mohalik and Walukiewicz (FSTTCS’03)

Restrictions
- Controllable actions: \( R(a) = W(a) \) is a singleton for all \( a \in P_0 \).
- Environment actions: \( R(e) = W(e) = V \) and \( P_1 = \{e\} \).
- Moves: \( \delta_e \subseteq Q_V \times Q_V \).
- Strategies: local memory with stuttering reduction so that a player \( a \in P_0 \) cannot see how long it has been idle.

Theorem
- Previous settings with local memory can be encoded.
- Two constructions to solve the distributed control problem subsuming previously known decidable cases with local memory.

Open problems
- Generalization to arbitrary symmetric architectures.
- Generalization to non-symmetric architectures.
- Reasonable upper bounds for synthesis?
Symmetric architecture

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

- Restrictions: \[
\begin{align*}
\forall a \in \mathcal{P} & \quad \emptyset \neq W(a) \subseteq R(a) \\
\forall a, b \in \mathcal{P} & \quad R(a) \cap W(b) \neq \emptyset \iff R(b) \cap W(a) \neq \emptyset
\end{align*}
\]
- Dependence: $a \overset{\text{D}}{\leftrightarrow} b \iff R(a) \cap W(b) \neq \emptyset \iff R(b) \cap W(a) \neq \emptyset$

Legal and forbidden architectures

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$\overset{\text{R}}{\leftrightarrow}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>OK</td>
<td>Forbidden (not symmetric)</td>
</tr>
</tbody>
</table>

65 / 65