Some remarks on the control of distributed automata

Anca Muscholl (joint work with I. Walukiewicz, M. Zeitoun)

LaBRI, Bordeaux

Chennai, January 2009
Asynchronous (Z-)automata, traces and event structures informally

Representing executions

- As a word:
  \[ a_1 a_2 a_3 b_1 c a_2 d \] or \[ a_2 a_3 a_1 b_1 c a_2 d \]
- As a trace.
- The set of all executions can be represented as a tree,
  or as an event structure (richer: concurrency).
Asynchronous (Z-)automata, traces and event structures informally

Representing executions

- As a word:
  \[ a_1 a_2 a_3 b_1 c a_2 d \text{ or } a_2 a_3 a_1 b_1 c a_2 d \]
- As a trace.
- The set of all executions can be represented as a tree,

or as an event structure (richer: concurrency).
The synthesis problem

**Centralized synthesis**
- We are given a specification $K$.
- We want a finite automaton $C$ with $L(C) \subseteq K$ + additional requirements (e.g., inputs are unconstrained).

**Distributed synthesis**
- Comes along with a distributed architecture (e.g., distributed (trace) alphabet).
- In general undecidable (Peterson/Reif ’79, Pnueli/Rosner 90).
- Important: use adequate specifications (e.g. trace closed ones for asynchronous automata).
Asynchronous automaton: example

P₁: \{a₁, b₁, c\}
P₂: \{a₂, c, d\}
P₃: \{a₃, b₃, d\}

**Alphabet**

- ℙ: finite set of processes.
- Σ: finite set of letters.
- loc : Σ → (2^{ℙ} \setminus \emptyset): distribution of letters over processes.

loc(a₁) = \{P₁\}, loc(c) = \{P₁, P₂\}, ...
Asynchronous automaton: example

Alphabet

- $\mathbb{P}$: finite set of processes.
- $\Sigma$: finite set of letters.
- $loc: \Sigma \rightarrow (2^\mathbb{P} \setminus \emptyset)$: distribution of letters over processes.

$loc(a_1) = \{P_1\}, \quad loc(c) = \{P_1, P_2\}, \ldots$
Asynchronous automata formally

Alphabet

- $\mathbb{P}$: finite set of processes.
- $\Sigma$: finite set of letters.
- $\text{loc} : \Sigma \rightarrow (2^\mathbb{P} \setminus \emptyset)$: distribution of letters over processes.

A (deterministic) asynchronous automaton

$$\mathcal{A} = \langle \{S_p\}_{p \in \mathbb{P}}, s_{in}, \{\delta_a\}_{a \in \Sigma} \rangle$$

- $S_p$ states of process $p$
- $s_{in} \in \prod_{p \in \mathbb{P}} S_p$ is a (global) initial state,
- $\delta_a : \prod_{p \in \text{loc}(a)} S_p \rightarrow \prod_{p \in \text{loc}(a)} S_p$ is a transition relation.
Language of an asynchronous automaton

The language of the automaton

The (regular) language of the product automaton.

Independence/Dependence

- Function $loc : \Sigma \rightarrow (2^\mathbb{P} \setminus \emptyset)$ implies an independence relation on letters:

  \[ (a, b) \in I \iff \text{loc}(a) \cap \text{loc}(b) = \emptyset \]

- So the language is closed under commuting independent letters (trace closed):

  \[ vabw \in L(A) \implies vba w \in L(A) \]

- Dependence relation $D = (\Sigma \times \Sigma) \setminus I$. We will express it graphically:

  \[ a \rightarrow c \rightarrow b \]
Traces: an example

**Dependence relation**

```
<table>
<thead>
<tr>
<th></th>
<th>a_2</th>
<th>a_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b_3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**A trace**

```
<table>
<thead>
<tr>
<th></th>
<th>a_1</th>
<th>a_2</th>
<th>a_3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_1</td>
<td></td>
<td>a_3</td>
</tr>
<tr>
<td></td>
<td>b_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>a_3</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

7/30
Structure on traces

Prefix relation on traces

- The prefix relation on traces $\sqsubseteq$ is defined similarly as for words.
- Differently from words, a trace may have two prefixes that are themselves $\sqsubseteq$-incomparable.

$$t_1, t_2 \sqsubseteq t \text{ but } t_1 \not\sqsubseteq t_2 \text{ and } t_1 \not\sqsubseteq t_2$$

For example: $a$ and $b$ are both prefixes of $abc$ when $(a, b) \in I$.

- We write $t_1 \not\# t_2$ if the two traces do not have a common extension.
  For example: $ac \not\# aac$ when $(a, c) \notin I$. 
Event structures

From words to trees

A prefix-closed language $L \subseteq \Sigma^*$ defines a $\Sigma$-labeled tree:
- nodes are elements of $L$,
- the tree order is given by the prefix relation $\sqsubseteq$.
- the label of $w \in L$ is the last letter in $L$.

From traces to event structures

A prefix-closed language $L \subseteq \text{Tr}(\Sigma)$ defines a $\Sigma$-labeled event structure:
- nodes are prime traces from $L$.
- the partial order is given by the prefix relation $\sqsubseteq$.
- relation $\#$ is called conflict relation.
- the label of $t$ is the label of the maximal element of $t$.

$ES(\mathcal{A})$

We denote by $ES(\mathcal{A})$ the (trace) event structure of the language $L(\mathcal{A})$. 
Event structures: examples

From traces to event structures

A prefix-closed language $L \subseteq \text{Tr}(\Sigma)$ defines a $\Sigma$-labeled event structure:

- nodes are prime traces from $L$.
- the partial order is given by the prefix relation $\sqsubseteq$.
- relation $\#$ is called conflict relation.
- the label of $t$ is the label of the maximal element of $t$.

$\Sigma = \{a, b\}$, independent

\[\begin{array}{c}
a \\
\downarrow \\
a \\
\downarrow \\
\downarrow \\
a \\
\end{array} \quad \begin{array}{c}
b \\
\downarrow \\
b \\
\downarrow \\
\downarrow \\
b \\
\end{array}\]

$\Sigma = \{a, b, c\}$, $D : a \rightarrow c \rightarrow b$

\[\begin{array}{c}
c \\
\downarrow \\
a \\
\downarrow \\
\downarrow \\
a \\
\end{array} \quad \begin{array}{c}
b \\
\downarrow \\
b \\
\downarrow \\
b \\
\end{array} \quad \begin{array}{c}
c \quad c \\
\quad \downarrow \\
\end{array} \quad \begin{array}{c}
D : a \rightarrow c \rightarrow c \\
\downarrow \\
\end{array}\]

\[\begin{array}{c}
a \\
\downarrow \\
a \\
\downarrow \\
\downarrow \\
a \\
\end{array} \quad \begin{array}{c}
c \\
\downarrow \\
c \\
\downarrow \\
c \\
\end{array} \quad \begin{array}{c}
c \\
\downarrow \\
\end{array}\]
Specifying event structures

Logics for event structures

First-order logic (FOL) over the signature ≤, #, Pa for a ∈ Σ:

\[ x \leq x' \mid x \# x' \mid Pa(x) \mid \neg \varphi \mid \varphi \lor \psi \mid \exists x. \varphi(x). \]

Monadic second-order logic (MSOL):

\[ \ldots \exists x \in Z \mid \exists Z. \varphi(Z). \]

Monadic trace logic (MTL): quantification restricted to conflict free sets.

Theorem (Madhusudan)

The problem if a given formula holds in a given trace event structure is decidable for FOL and MTL.

Remark

There are trace event structures with undecidable MSOL theory (grid).
Part 1

Controlling asynchronous automata

- Process and action-based control.
- Reduction from process-based to action-based control.
- Encoding into MSOL theory of event structures.
Controlling an asynchronous automaton: an example

Example specifications

1. $a_i b_j c_k$ with $k = i$.
2. $a_i b_j c_k$ with $k = i \cdot j$.

Two methods of control

- **Process-based** [Madhusudan et al.]: Process decides what actions it can do.
- **Action-based** [Gastin et al.]: Actions decide whether they can execute.
Process-based control

Plant over $\mathbb{P}$, $loc : \Sigma \to (2^\mathbb{P} \setminus \emptyset)$ and $\Sigma = \Sigma^{sys} \cup \Sigma^{env}$

A deterministic asynchronous automaton.

Views for a process $p \in \mathbb{P}$

- Let $view_p(t)$ be the smallest prefix of $t$ containing all $p$-actions.
- Let $Plays_p(A) = \{view_p(t) : t \in L(A)\}$.

Strategy

- A strategy is a tuple of functions $f_p : Plays_p(A) \to 2^{\Sigma^{sys}}$ for $p \in \mathbb{P}$.
- Plays respecting $\sigma = \{f_p\}_{p \in \mathbb{P}}$. Assume $u \in Plays(A, \sigma)$.
  - if $a \in \Sigma^{env}$ and $ua \in Plays(A)$ then $ua$ is in $Plays(A, \sigma)$.
  - if $a \in \Sigma^{sys}$ and $ua \in Plays(A)$ then $ua \in Plays(A, \sigma)$ provided that
    $a \in f_p(view_p(u))$ for all $p \in loc(a)$.
Process-based control

Requirements

- We are given asynchronous automaton $\mathcal{A}$ and a regular trace language $K$.
- A strategy $\sigma = \{f_p\}_{p \in \mathbb{P}}$ gives us a set of traces $\text{Plays}^\omega(\mathcal{A}, \sigma)$.
- A strategy is non-blocking if every trace in $\text{Plays}(\mathcal{A}, \sigma)$ that has an extension in $\text{Plays}(\mathcal{A})$, also has an extension in $\text{Plays}(\mathcal{A}, \sigma)$.

The control problem

Given $\mathcal{A}$ and $K$, decide if there is a non-blocking strategy $\sigma$ such that $\text{Plays}^\omega(\mathcal{A}, \sigma) \subseteq K$. 
## Action-based control

<table>
<thead>
<tr>
<th>Process based</th>
<th>Action based</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{view}_p(t)$</td>
<td>$\text{view}_a(t) = \bigcup { \text{view}_p(t) : p \in \text{loc}(a) }$</td>
</tr>
<tr>
<td>$\text{Plays}_p(A)$</td>
<td>$\text{Plays}_a(A) = { \text{view}_a(t) : t \in L(A) }$</td>
</tr>
<tr>
<td>$f_p : \text{Plays}_p(A) \to 2^{\Sigma^{sys}}$</td>
<td>$g_a : \text{Plays}_a(A) \to {tt, ff}$</td>
</tr>
<tr>
<td>$\sigma = { f_p }_{p \in P}$</td>
<td>$\rho = { g_a }_{a \in \Sigma^{sys}}$</td>
</tr>
</tbody>
</table>

### $\text{Plays}^\omega(A, \rho)$

- if $a \in \Sigma^{env}$ and $ua \in \text{Plays}(A)$ then $ua$ is in $\text{Plays}(A, \rho)$.
- if $a \in \Sigma^{sys}$ and $ua \in \text{Plays}(A)$ then $ua \in \text{Plays}(A, \rho)$ provided that $g_a(\text{view}_a(u)) = tt$. 
Reduction “process-based” to “action-based”

Observation 1
If there is a process-based controller then there is an action-based controller.

Observation 2
This does not in principle imply that process-based control is easier than action-based control (nor vice-versa).

Fact
For every asynchronous automaton $\mathcal{A}$ and MSOL specification $\alpha$, one can construct $\overline{\mathcal{A}}$ and $\overline{\alpha}$ such that:

\[
\text{process-based controller for } (\mathcal{A}, \alpha) \text{ exists iff action-based controller for } (\overline{\mathcal{A}}, \overline{\alpha}) \text{ exists.}
\]
Reduction: example

\[ P_1 : \{a_1\}\quad P_2 : \{c, d\}\quad P_3 : \{a_3\} \]

\[ \Sigma_{sys} = \{a_1, a_3, c, d\} \]

- \(P_1: a_1\) always possible, \(c\) only after \(a_1\)
- \(P_2: c\) always possible, \(d\) after \(c\) or if no \(A_2\) before
- \(P_3: a_3\) always possible, \(d\) only after \(a_3\)
Reduction: example (cont.)

New arena for action-based strategy

- New (local) system actions: $\top, \{a_1\}, \{a_1, c\}, \{c, d\}, \{c\}, \ldots$
- New (local) environment actions: $\bot, (a, P_1), (c, P_1), (d, P_2), \ldots$
- $\top$ winning and $\bot$ losing (for system)
Reduction: example (cont.)

New arena for action-based strategy

- System proposes its set of local actions in form of new actions (process-wise), e.g. \( \{a_1, c\}, \{c\} \). If proposed sets have empty \( \cap \) (although actions are possible) then \( \bot \) is possible.

- Environment chooses one of the proposed actions (process-wise). If it chooses maliciously (e.g. \( (a_1, P_1), (c, P_2) \)) then \( \top \) is possible.
Encoding process-based control

MSOL encoding (Madhusudan et al.)

For a MSOL specification $\alpha$ there is a MSOL formula $\varphi_\alpha$ such that $ES(A) \models \varphi_\alpha$ iff the process-based control problem for $(A, \alpha)$ has a solution.

Remark

The same can be done for action-based control.
Writing the formula $\varphi_\alpha$

### Encoding strategies

- Take $\sigma = \{ f_p \}_{p \in \mathbb{P}}$ where each $f_p : \text{Plays}_p(A) \rightarrow 2^{\Sigma_{sys}}$.
- Encode $\sigma$ with the help of variables $Z^a_p$ for $a \in \Sigma_{sys}$ and $p \in \mathbb{P}$.

for every $e \in ES(A)$ \quad $e \in Z^a_p$ \quad iff \quad $a \in f_p(e)$

### Encoding action-based control

- Write a formula $\pi(X, Z^a_p, \ldots)$ defining $\text{Plays}(A, \sigma)$.
- Write a formula $\pi^\omega(X, Z^a_p, \ldots)$ defining $\text{Plays}^\omega(A, \sigma)$.
- Say that all paths in $\text{Plays}^\omega(A, \rho)$ satisfy the specification: \quad $\forall X. \pi^\omega(X, Z^a_p, \ldots) \Rightarrow \alpha(X)$.
- The required formula is: $\exists Z^a_p \ldots \forall X. \pi^\omega(X, Z^a_p, \ldots) \Rightarrow \alpha(X)$. 
Decidability of MSOL is not necessary

**Definition**

A trace alphabet is a co-graph if it does not contain the induced subgraph $a - b - c - d$.

**Theorem (Gastin, Lerman, Zeitoun)**

*The action-based control problem is decidable for automata over co-graph trace alphabets.*

**Remark**

Alphabet $\Sigma = \{a, b, c\}$ with $a - c - b$ is a co-graph. There is $A$ over this alphabet whose $ES(A)$ has undecidable MSOL theory.
Part 2

MSOL and Thiagarajan’s conjecture

- Thiagarajan’s conjecture
- Co-graph dependence alphabets
**Synchronizing automata**

An automaton \( A \) is not synchronizing if there are traces \( x, u, v, y \) such that

- \( u, v \) are nonempty and independent from each other.
- \( xu vy \) is a prime trace.
- \( xu^* v^* y \subseteq L(A) \).

**Remark**

If \( A \) is not synchronizing then \( ES(A) \) has undecidable MSOL theory.

**Conjecture**

If \( A \) is synchronizing then the MSOL theory of \( ES(A) \) is decidable.
Strongly synchronizing automata

An asynchronous automaton $\mathcal{A}$ is strongly synchronizing if in every prime trace of $L(\mathcal{A})$, each of its events has at most $|\mathcal{A}|$ many concurrent events.

Theorem (Madhusudan, Thiagarajan, Yang)

*If $\mathcal{A}$ is strongly synchronizing then the MSOL theory of $ES(\mathcal{A})$ is decidable.*

Corollary

Both process- and action-based control are decidable for strongly synchronizing automata.
**Remark**

There are automata $\mathcal{A}$ that are not strongly synchronizing but still MSOL theory of $ES(\mathcal{A})$ is decidable.

**Example:** $\Sigma = \{a, b, c\}$, $D : a - c - b$, $L(\mathcal{A}) = a^*ba^*c + c$

- This event structure is not strongly synchronizing.
- It has decidable MSOL theory.

- Encode prime trace $[a^m bc]$ by the word $a^m bc$, etc.
- Translate MSOL over event structure into MSOL over $\{a, b, c\}$-tree.
- Ex: partial order $[a^n] < [a^n bc]$ translates to $a^n < a^n bc$ (word prefix).
- Rem: encoding $[a^n bc]$ by $ba^n c$ does not work, since $a^n$ and $ba^n$ far apart in the tree.
**Strongly synchronizing are too strong**

**Remark**

There are automata $A$ that are not strongly synchronizing but still MSOL theory of $ES(A)$ is decidable.

**Example:** $\Sigma = \{a, b, c\}$, $D : a \rightarrow c \rightarrow b$, $L(A) = a^*ba^*c + c$

- This event structure is not strongly synchronizing
- It has decidable MSOL theory.

- Encode prime trace $[a^m bc]$ by the word $a^m bc$, etc.
- Translate MSOL over event structure into MSOL over $\{a, b, c\}$-tree.
- Ex: partial order $[a^n] < [a^n bc]$ translates to $a^n < a^n bc$ (word prefix).
- Rem: encoding $[a^n bc]$ by $ba^n c$ does not work, since $a^n$ and $ba^n$ far apart in the tree.
Towards a solution: co-graphs

Trace normal form

We look for trace normal forms $\text{nf}(t)$ that behave well w.r.t. prefix relation:

for all traces $t < t'$ there are words $p, s, s'$ s.t.

$$\text{nf}(t') = ps', \text{nf}(t) = ps \text{ and } s \text{ is small}$$

Co-graphs

- In trace $t$: $a \parallel b$ and $a \uparrow \cap b \uparrow \neq \emptyset$.
- For co-graphs (and $A$ synchronizing) there is no extension $t'$ of $t$ such that

$$\Delta_{t'}(a, b) > \Delta_t(a, b) \text{ or } \Delta_{t'}(b, a) > \Delta_t(b, a).$$
Normal form

Dynamical lexicographic form

- Enforce more order to the trace partial order:
  \[ \text{For } a \parallel b \text{ let } a \prec b \text{ if } |\Delta_t(a, b)| > N, \quad N = |\mathcal{A}|. \]

- Trace partial order \( t \) plus \( \prec \) is acyclic: \( t \prec \).
- The priority normal form is the lexicographic normal form of \( t \prec \).
- If \( \mathcal{A} \) is strongly synchronizing then it coincides with the lexicographic normal form.

Priority normal form and reduction

- Priority normal form has the desired property:
  for all traces \( t \prec t' \) there are words \( p, s, s' \) s.t.
  \[ nf(t') = ps', \quad nf(t) = ps \text{ and } s \text{ is small} \]

- Reduction of MSOL over \( ES(\mathcal{A}) \) to MSOL over \( \Sigma \)-tree works by identifying \( t \) with word \( p \) in the tree and checking that small \( s \) fits correctly into \( s' \).
Normal form

Dynamical lexicographic form

- Enforce more order to the trace partial order:
  
  $$\text{For } a \parallel b \text{ let } a \prec b \text{ if } |\Delta_t(a, b)| > N, N = |\mathcal{A}|.$$ 

- Trace partial order $$t$$ plus $$\prec$$ is acyclic: $$t \prec$$.

- The priority normal form is the lexicographic normal form of $$t \prec$$.

- If $$\mathcal{A}$$ is strongly synchronizing then it coincides with the lexicographic normal form.

Priority normal form and reduction

- Priority normal form has the desired property:
  
  for all traces $$t < t'$$ there are words $$p, s, s'$$ s.t.
  
  $$nf(t') = ps', \ nf(t) = ps \text{ and } s \text{ is small}$$

- Reduction of MSOL over $$ES(\mathcal{A})$$ to MSOL over $$\Sigma$$-tree works by identifying $$t$$ with word $$p$$ in the tree and checking that small $$s$$ fits correctly into $$s'$$. 
Conclusions

• While traces are relatively well understood, event structures are much less.
• From the synthesis point of view, event structures are more fundamental than traces.
• Thiagarajan’s conjecture is an important milestone in understanding the decidability frontier.
• Thiagarajan’s conjecture is true for co-graphs. The general case remains open.
• It may well be the case that action based control is decidable for all asynchronous automata.