Traces and Distributed Synthesis (Tracing the footsteps of Thiagu)

Igor Walukiewicz

CNRS, LaBRI, Bordeaux

Chennai, December 16th, 2008

Part I

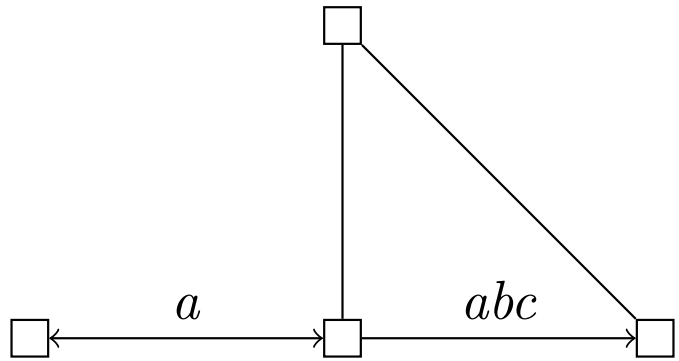
Introduction

Why distributed setting is interesting for specification, verification and synthesis.

Models of distributed systems

Distributed systems

Boxes with links. In every box there is a finite automaton.



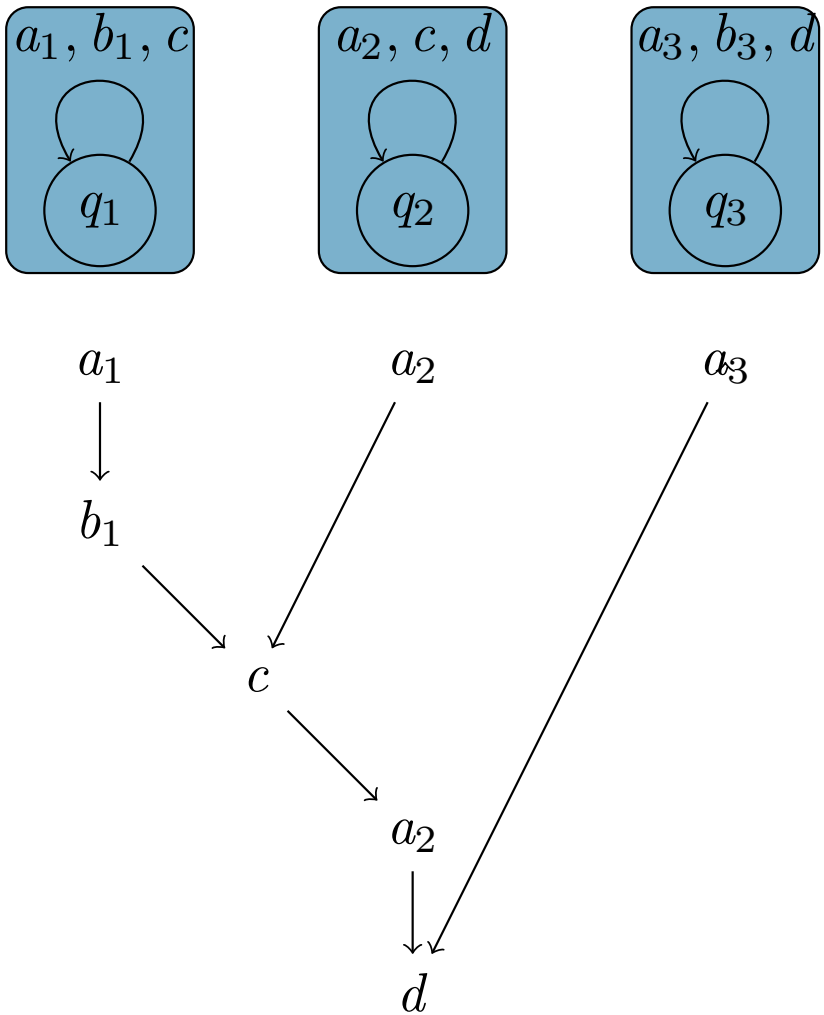
Links as channels

Every link is channel and boxes have send and receive operations: communicating finite state machines, MSC's

Links as synchronization

Communication over the link requires two processes to synchronize: asynchronous automata, traces, event structures.

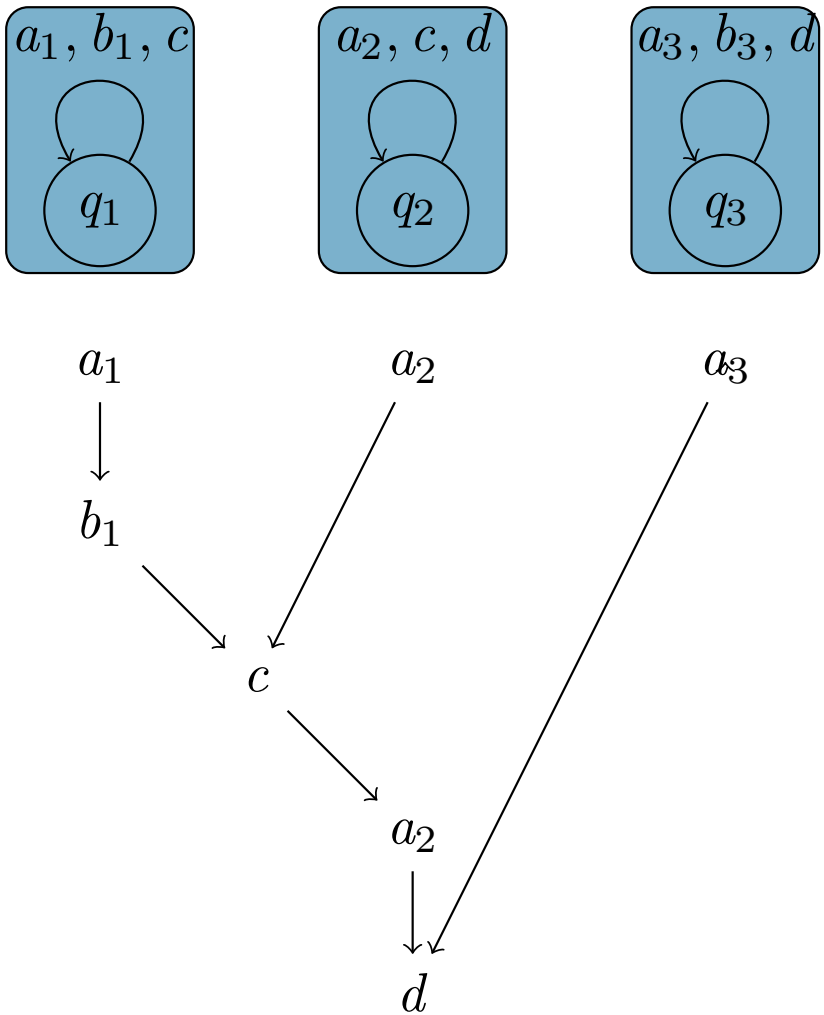
Asynchronous automata, traces and event structures informally



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.

Asynchronous 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.

Question: why not stay with words and trees?

Specification

Expressing concurrency. How to say that we want (or allow) two actions to be concurrent?

Verification

State explosion problem. It may be possible to gain in efficiency if the structure of the system is known.

Synthesis

Undecidability. With specifications that do not respect concurrency of the system the synthesis problem becomes undecidable.

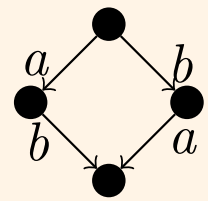
Ad specification

How to express concurrency between a and b?

1. We can write

   $vabw \in L(\alpha)$ iff $vbaw \in L(\alpha)$; for all $v, w \in \Sigma^*$

2. We can require the diamond property of the automaton.

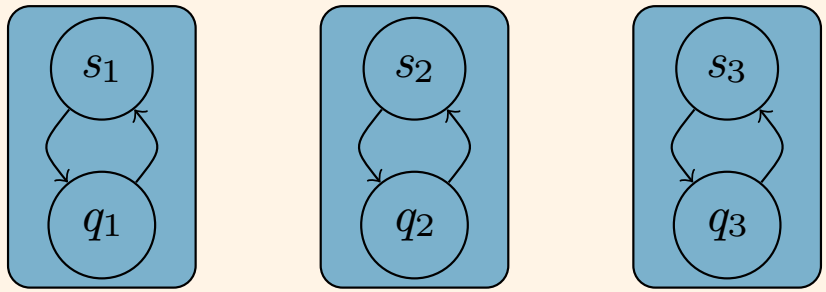


Problems

1. The first property is not MSOL definable.
2. The diamond property of the automaton is not bisimulation invariant.

Ad verification

The state explosion problem

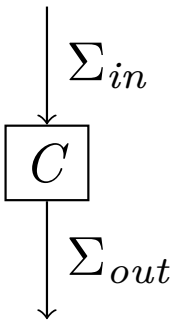


One can always pass from distributed to sequential setting.

One can hope that keeping additional information about the structure of the system will make verification easier.

- Unfolding techniques [McMilan, Esparza,. . .]
- Partial order methods [Peled,Valmari,. . .]

The synthesis problem



$$K \subseteq (\Sigma_{in}\Sigma_{out})^*$$

Centralized synthesis

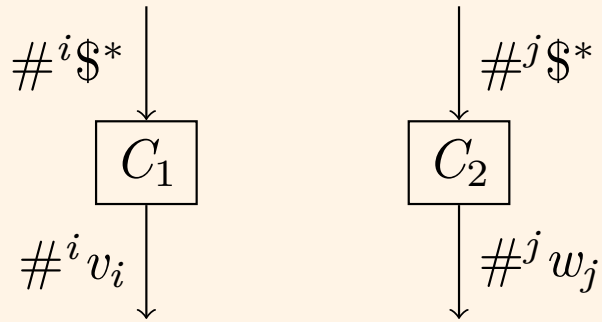
- We are given a specification K.
- We want a finite automaton C with $L(C) \subseteq K$.

Additional requirements

- $\varepsilon \in L(C)$.
- If $w \in L(C)$ and w ends in Σ_{out} then for every $a \in \Sigma_{in}$, $wa \in L(C)$.
- If $w \in L(C)$ and w ends in Σ_{in} then there is $a \in \Sigma_{out}$ with $wa \in L(C)$.

Reasons for undecidability

A simple distributed architecture



$$v_i = w_i$$

$$v_i \vdash_M w_{i+1}$$

Specification talks about inputs and outputs of the two processes while each processes knows only its input.

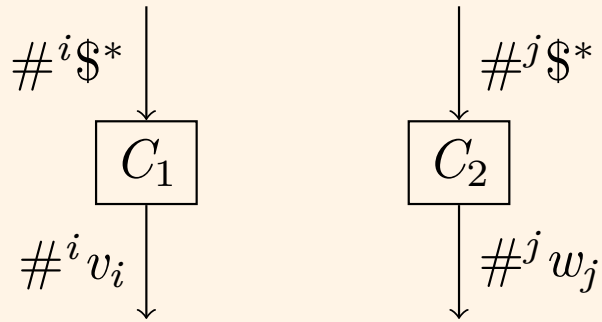
Behaviour

$$v_1 = w_1, \quad v_1 \vdash w_2, \quad v_2 = w_2, \quad v_2 \vdash w_3, \ldots$$

$$v_1 \vdash w_2 = v_2 \vdash w_3 = v_3 \ldots$$

Reasons for undecidability

A simple distributed architecture



$$v_i = w_i$$

$$v_i \vdash_M w_{i+1}$$

Specification talks about inputs and outputs of the two processes while each processes knows only its input.

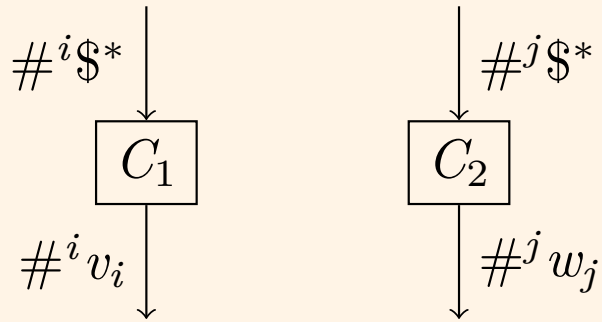
Behaviour

$v_1 = w_1, \quad v_1 \vdash w_2, \quad v_2 = w_2, \quad v_2 \vdash w_3, \ldots$

$$v_1 \vdash w_2 = v_2 \vdash w_3 = v_3 \ldots$$

Reasons for undecidability

A simple distributed architecture



$$v_i = w_i$$

$$v_i \vdash_M w_{i+1}$$

Specification talks about inputs and outputs of the two processes while each processes knows only its input.

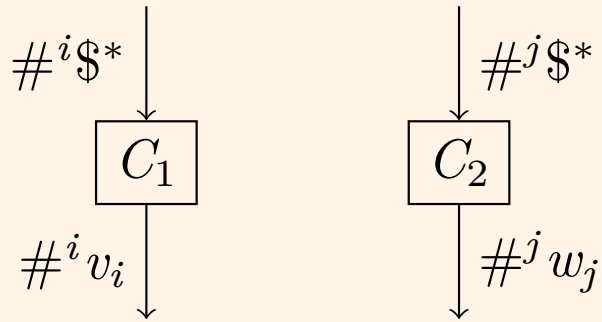
Behaviour

$v_1 = w_1, \quad v_1 \vdash w_2, \quad v_2 = w_2, \quad v_2 \vdash w_3, \ldots$

$$v_1 \vdash w_2 = v_2 \vdash w_3 = v_3 \ldots$$

Reasons for undecidability

A simple distributed architecture



$$v_i = w_i$$

$$v_i \vdash_M w_{i+1}$$

Specification talks about inputs and outputs of the two processes while each processes knows only its input.

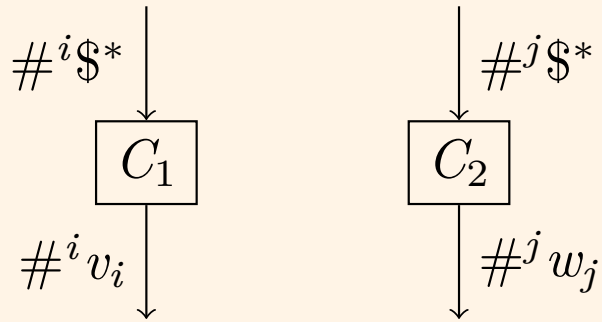
Behaviour

$v_1 = w_1, \quad v_1 \vdash w_2, \quad v_2 = w_2, \quad v_2 \vdash w_3, \ldots$

$$v_1 \vdash w_2 = v_2 \vdash w_3 = v_3 \ldots$$

Reasons for undecidability

A simple distributed architecture



$$v_i = w_i$$

$$v_i \vdash_M w_{i+1}$$

Specification talks about inputs and outputs of the two processes while each processes knows only its input.

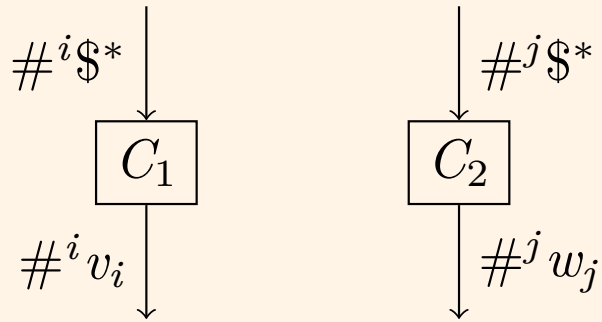
Behaviour

$v_1 = w_1, \quad v_1 \vdash w_2, \quad v_2 = w_2, \quad v_2 \vdash w_3, \ldots$

$v_1 \vdash w_2 = v_2 \vdash w_3 = v_3 \ldots$

Reasons for undecidability

A simple distributed architecture



$$v_i = w_i$$

$$v_i \vdash_M w_{i+1}$$

Specification talks about inputs and outputs of the two processes while each processes knows only its input.

Behaviour

$$v_1 = w_1, \quad v_1 \vdash w_2, \quad v_2 = w_2, \quad v_2 \vdash w_3, \ldots$$

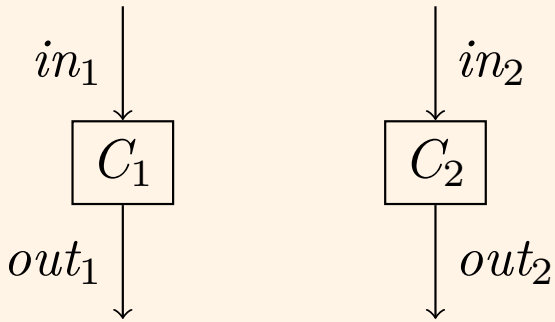
$$v_1 \vdash w_2 = v_2 \vdash w_3 = v_3 \ldots$$

Decidable specifications

Local specification

A specification is local if it is a conjunction of requirements on each controller.

Decidability for local specifications



$$L_1 \subseteq (in_1 \cup out_1)^*$$

$$L_2 \subseteq (in_2 \cup out_2)^*$$

Theorem (Thiagarajan & Madhusudan)

The synthesis problem above is decidable for local specifications.

Remark

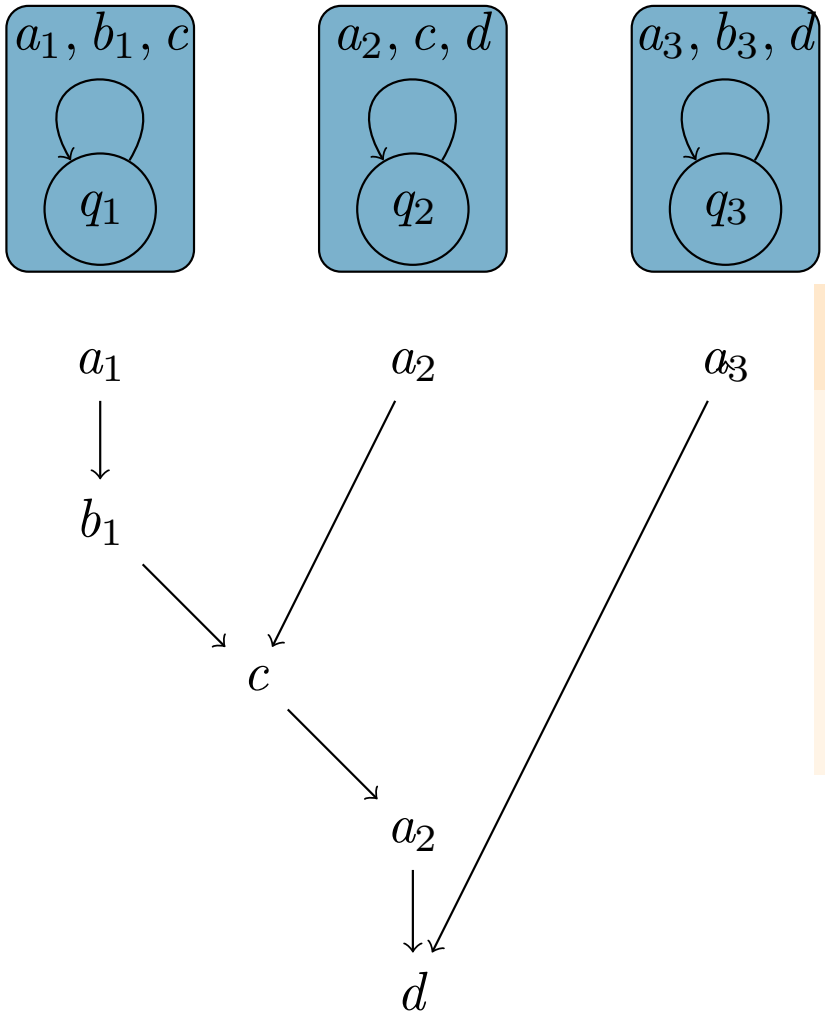
Local specifications and trace closed specifications are the same in this case.

Part II

Distributed systems and ways of specifying their behaviour

- Asynchronous automata
- Traces
- Event structures.

Asynchronous automaton: example



Alphabet

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

Asynchronous automata formally

Alphabet

- $\mathbb{P}$: finite set of processes.
- Σ: finite set of letters.
- $loc : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)$: 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,
- $\delta_a : \prod_{p \in loc(a)} S_p \stackrel{\cdot}{\to} \prod_{p \in loc(a)} S_p$ is a transition relation.

Language of an asynchronous automaton

The language of the automaton

The (regular) language of the product automaton.

Independence/Dependence

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

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

- So the language is a closed under permutations of independent letters:

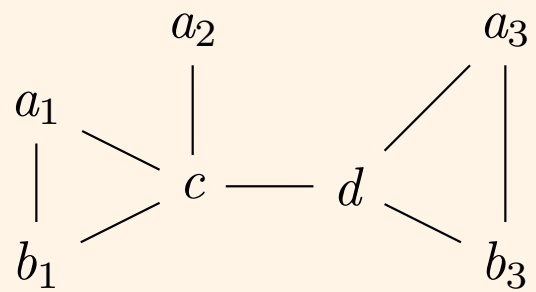
$$wabv \in L(\mathcal{A}) \quad \text{implies} \quad wbav \in L(\mathcal{A})$$

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

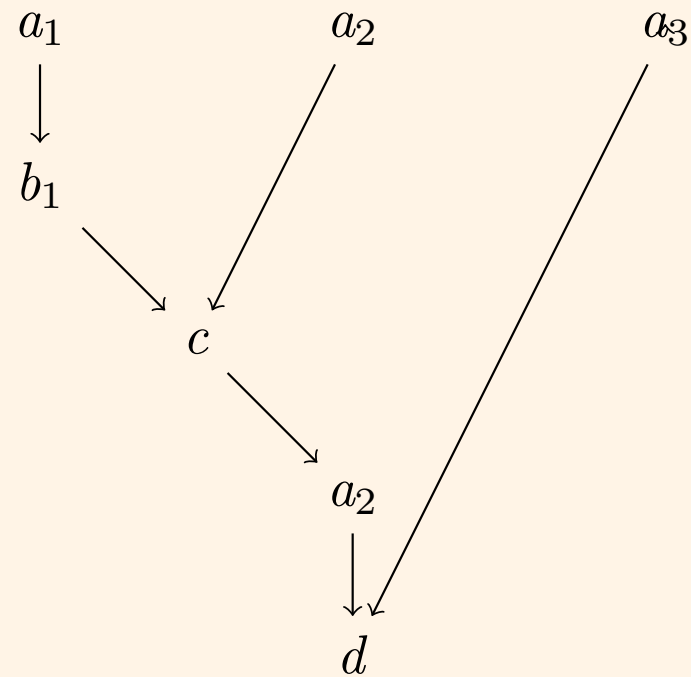
$$a - c - b$$

Traces: an example

Dependence relation



A trace



Traces: definition

Alphabet

Σ alphabet, $D \subseteq \Sigma \times \Sigma$ dependency relation (reflexive and symmetric).

Trace

Trace or dependency graph is a partial order

$$T = \langle E, \leq, \lambda \rangle$$

(T1) $\forall e \in E.$ $\quad e{\downarrow}$ is a finite set
(T2) $\forall e, e' \in E.$ $\quad D(\lambda(e), \lambda(e'))$ iff $e \leq e'$ or $e' \leq e$.

$\mathrm{tr}(\Sigma, D)$ – the set of all traces over (Σ, D).

Specifying trace properties

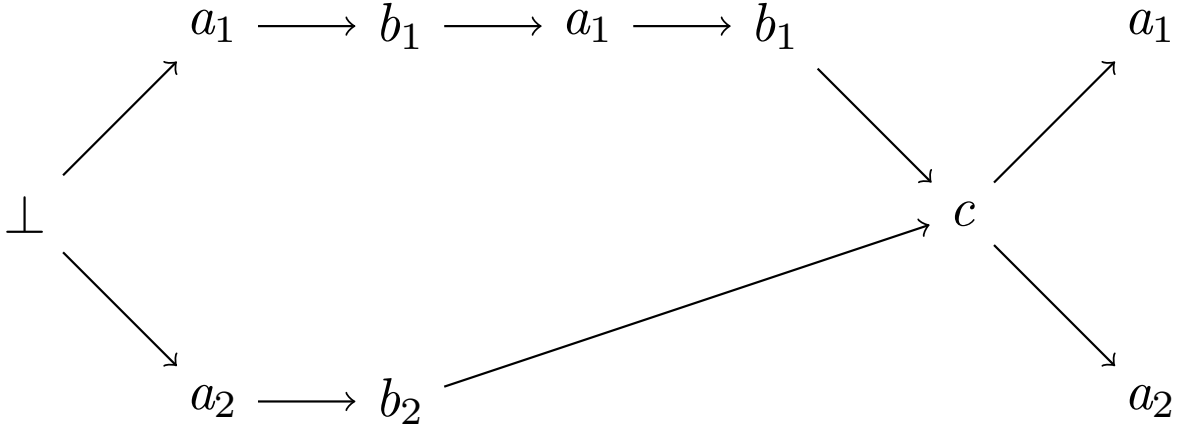
Theorem (Zielonka, Thomas, Ebinger, Muscholl. . .)

- *Trace languages definable by asynchronous automata are exactly those definable in MSOL over traces.*
- *These are the languages whose set of linearizations is regular.*
- *MSOL over traces* $\equiv$ *MSOL over linearizations of traces.*

Characterisations of recognizable trace languages

- MSOL logic
- Asynchronous Büchi automata
- Recognizing morphisms
- c-regular expressions: a, $\cup$, $\cdot$, $c-*$, $c-\omega$
- A kind of μ-calculus

Global vs. local logics



Some notions

- Configuration: downwards closed set of events.
- Prime trace: one maximal element.
- $C \Rightarrow_a C'$

Global and local logics

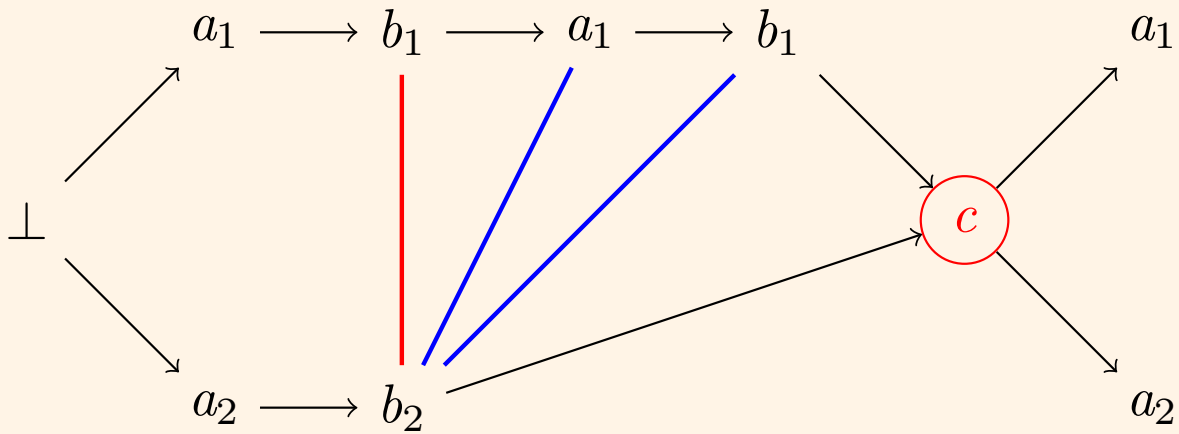
- Global logic is evaluated in configurations: $T, C \vDash \alpha$.
- Local logic is evaluated in events: $T, e \vDash \alpha$.

Global temporal logic $LTrL$

Syntax

$$LTrL(\Sigma, D) ::= tt \mid \neg\alpha \mid \alpha \vee \beta \mid \langle a \rangle \alpha \mid \alpha \mathbb{U} \beta \mid \langle a^{-1} \rangle tt$$

Semantics



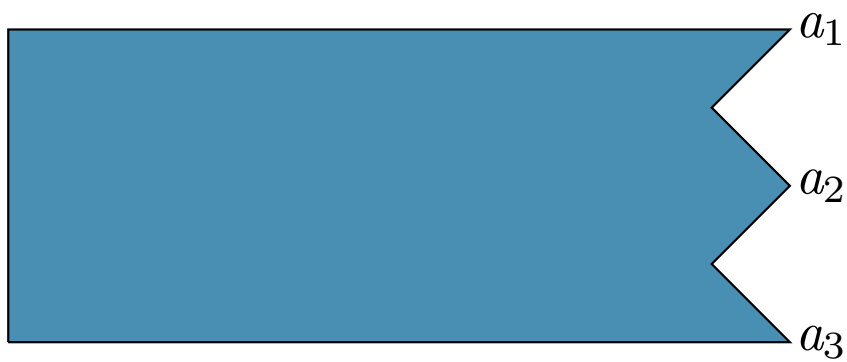
$\alpha \mathbb{U} \beta$

$$\mathbb{E}\alpha \equiv tt\ \mathbb{U}\alpha \qquad \mathbb{A}\alpha \equiv \neg\mathbb{E}\neg\alpha$$

Example

There are three independent events labelled by a_1, a_2 and a_3.

$$\mathbb{E}\Big((\langle a_1\rangle tt) \wedge (\langle a_2\rangle tt) \wedge (\langle a_3\rangle tt)\Big)$$



Expressiveness and Complexity

Theorem (Thiagarajan & W.)

$LTrL(\Sigma, D)$ is expressively equivalent to $\mathrm{FOL}(\Sigma, D)$.

Theorem

$LTrL(\Sigma, D)$ is non-elementary.

Theorem

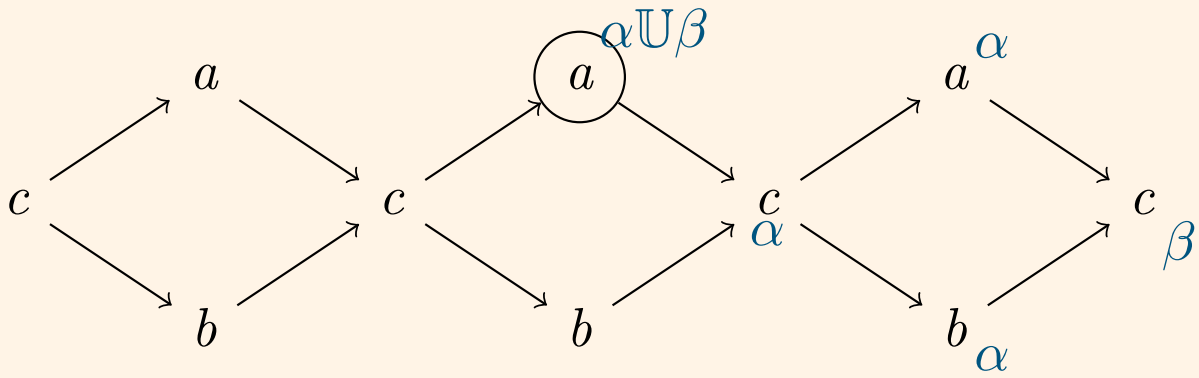
$LTrL^{-}(\Sigma, D)$ is EXPSPACE-complete.

Where $LTrL^{-}(\Sigma, D)$ is the logic obtained by removing $\mathbb{U}$ and adding $\mathbb{E}$ as a primitive.

Local Temporal Logics

LTL ::= $tt \mid \neg\alpha \mid \alpha \wedge \beta \mid \langle a \rangle \alpha \mid \alpha \mathbb{U} \beta$

Semantics of Unitl



Theorem (Diekert & Gastin)

LTL over traces $\equiv$ *FOL over traces.*

Theorem (Gastin & Kuske)

Every local trace logic with a finite number of operators definable in MSOL is decidable in PSPACE.

Structure on traces

Prefix relation on traces

- The prefix relation on traces, $t_1 \sqsubset t_2$ is defined very similarly as for words.
- Differently from words, a trace may have two prefixes that are themselves incomparable by a prefix relation.

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

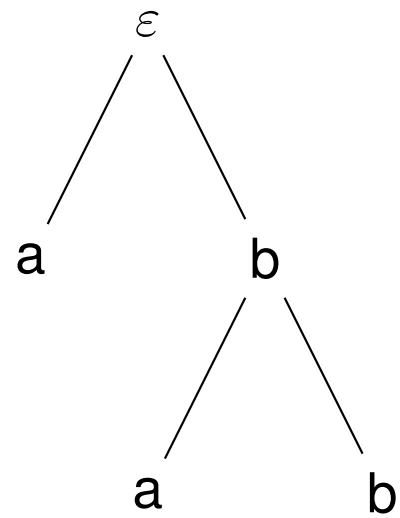
  For example: a and b are both prefixes of abc.
- We write $t_1 \overline{\wedge} t_2$ if the two traces do not have a common extension.
  For example: $ac \overline{\wedge} aac$

Event structures

From words to trees

A prefix-closed language $L \subseteq \Sigma^*$ defines a Σ labeled tree:

- nodes are elements of L,
- the tree order is given by the prefix relation $\sqsubset$.
- the label of $w \in L$ is the last letter in L.



From traces to event structures

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

- nodes are prime traces from L.
- the partial order is given by the prefix relation $\sqsubset$ (on traces).
- the conflict relation $\#$ is $\overline{\wedge}$ on traces.
- 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 $L(\mathcal{A})$.

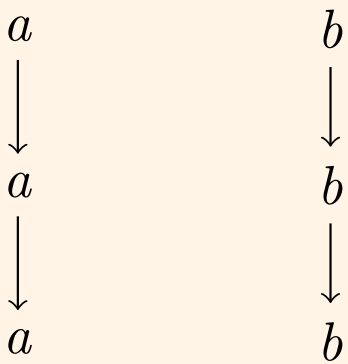
Event structures: examples

From traces to event structures

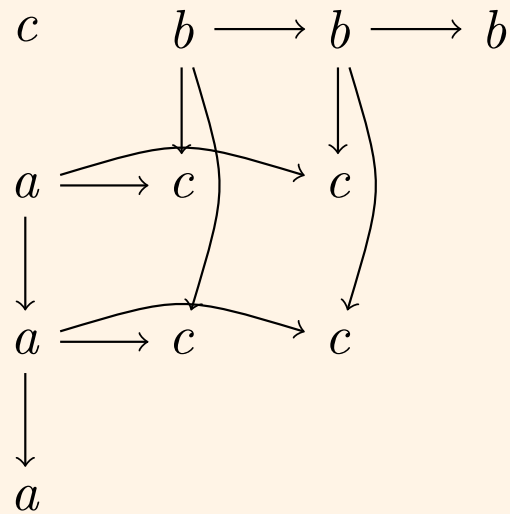
A prefix-closed language $L \subseteq \mathrm{tr}(\Sigma)$ defines a Σ labeled event structure:

- nodes are prime traces from L.
- the partial order is given by the prefix relation $\sqsubset$ (on traces).
- the conflict relation $\#$ is $\overline{\wedge}$ on traces.
- the label of t is the label of the maximal element of t.

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



$\Sigma = \{a, b, c\}$, c common



Event structures

Event structure $(E, \leq, \#)$

- E: a set of events
- $\leq$: a partial order on E (causality relation)
- $\#$: symmetric and irreflexive relation (conflict relation)

Two conditions:

1. $e{\downarrow}$ is a finite set for every $a \in E$.
2. For $e_1, e_2, e_3 \in E$, if $e_1 \# e_2$ and $e_2 \leq e_3$ then $e_1 \# e_3$.

Question

When such an object comes from a finite device?

Regular event structures

Suffix of an event structure

E/e is the part of the event structure $E - (\{e\} \cup \{e' : e' \# e\})$

Regular event structure

E is regular if there are finitely many E/e and there is a uniform bound on the out-degree of each event.

Conjecture [Thiagarajan]

Does every regular event structure comes from a 1-safe Petri Net?

Regular event structures

Suffix of an event structure

E/e is the part of the event structure $E - (\{e\} \cup \{e' : e' \# e\})$

Regular event structure

E is regular if there are finitely many E/e and there is a uniform bound on the out-degree of each event.

Theorem (Lodaya, Paul)

Every regular event structure comes from a 1-safe Petri Net.

Specifying event structures

Logics for event structures

First-order logic (FOL) over the signature $\leq$, $\#$, P_a for $a \in \Sigma$:

$$x \leq x' \mid x \# x' \mid P_a(x) \mid \neg\varphi \mid \varphi \vee \psi \mid \exists x.\varphi(x).$$

Monadic second-order logic (MSOL)

$$\ldots x \in Z \mid \exists Z.\varphi(Z).$$

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

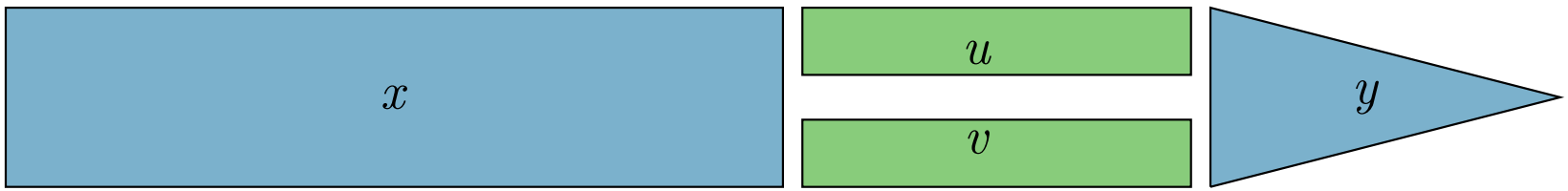
There are trace event structures with undecidable MSOL theory.

Thiagarajan's conjecture

Synchronizing automata

An automaton $\mathcal{A}$ is not synchronizing if there are traces x, u, v, y such that

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



Remark

If $\mathcal{A}$ is not synchronizing then $ES(\mathcal{A})$ has undecidable MSOL theory.

Conjecture

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

Strongly strongly-synchronizing automata

Strongly synchronizing automaton

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.

Remark

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

Fact

Thiagarajan's conjecture holds when alphabet is a cograph.

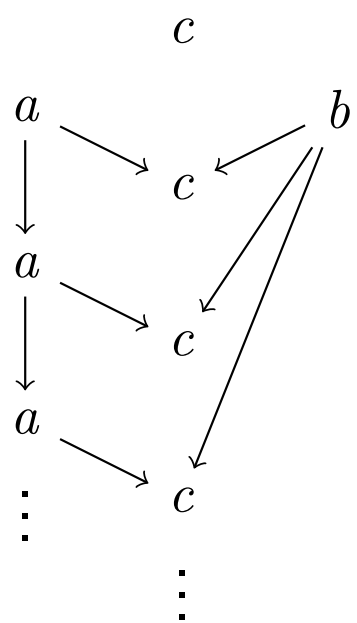
Strongly synchronizing are too strong

Remark

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

Example: $\Sigma = \{a, b, c\}$, $L(\mathcal{A}) = a^*bc$

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

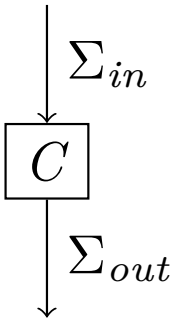


Part III

Controlling asynchronous automata

- Process and action based-control.
- Encoding into MSOL theory of event structures. Decidability results.
- Reduction from process to action-based control.

Back to synthesis



$K \subseteq (\Sigma_{in}\Sigma_{out})^*$

Centralized synthesis

- We are given a specification K.
- We want a finite automaton C with $L(C) \subseteq K$.

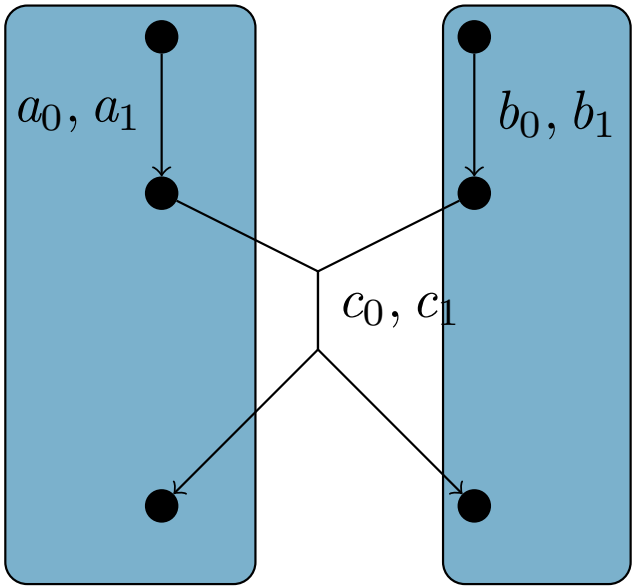
Additional requirements

- If $w \in L(C)$ and w ends in Σ_{out} then for every $a \in \Sigma_{in}$, $wa \in L(C)$.
- If $w' \in L(C)$ and w ends in Σ_{in} then there is $a \in \Sigma_{out}$ with $wa \in L(C)$.

Remark

Synthesis is about branching time properties.

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: Process decides what actions it can do.
- Action-based: Actions decide whether they can execute.

Process based control

Plant over $\mathbb{P}$, $loc : \Sigma \to \mathbb{P}$ 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 $Plays_p(\mathcal{A}) = \{view_p(t) : t \in L(\mathcal{A})\}$.

Strategy

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

Process based control

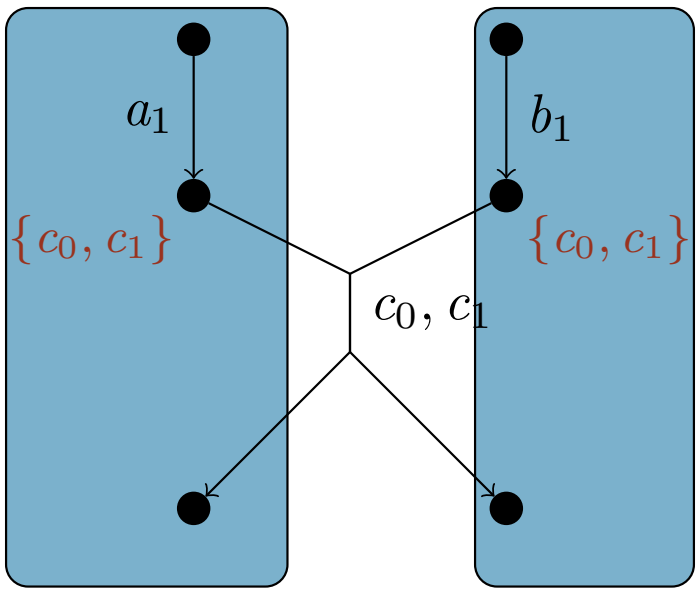
Requirements

- 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 $Plays^{\omega}(\mathcal{A}, \sigma)$.
- A strategy is non-blocking if every trace in $Plays(\mathcal{A}, \sigma)$ that has a prolongation in $Plays(\mathcal{A})$ has a prolongation in $Plays(\mathcal{A}, \sigma)$.

The synthesis problem

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

Process based control



Example specifications

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

Action based control

Process based	Action based
$view_p(t)$	$view_a(t) = \bigcup\{view_p(t) : p \in loc(a)\}$
$Plays_p(\mathcal{A})$	$Plays_a(\mathcal{A}) = \{view_a(t) : t \in L(\mathcal{A})\}$
$f_p : Plays_p(\mathcal{A}) \to \Sigma^{sys}$	$g_a : Plays_a(\mathcal{A}) \to \{tt, ff\}$
$\sigma = \{f_p\}_{p \in \mathbb{P}}$	$\rho = \{g_a\}_{a \in \Sigma^{sys}}$

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

- if $a \in \Sigma^{env}$ and $ua \in Plays(\mathcal{A})$ then ua is in $Plays(\mathcal{A}, \rho)$.
- if $a \in \Sigma^{sys}$ and $ua \in Plays(\mathcal{A})$ then $ua \in Plays(\mathcal{A}, \rho)$ provided that $g_a(view_a(u)) = tt$.

Encoding the process-based synthesis problem

Encoding the process based synthesis

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

Corollary [Madhusudan & Thiagarajan]

The process-based control problem is decidable for strongly synchronizing automata.

Remark

The same can be done for action-based control.

Writing the formula φ_α

Encoding strategies

- Take $\sigma = \{f_a\}_{p \in \mathbb{P}}$ where each $f_p : Plays_p(\mathcal{A}) \to \Sigma$.
- Encode σ with the help of variables Z_p^a for $a \in \Sigma^{sys}$ and $p \in \mathbb{P}$.

for every $e \in ES(\mathcal{A})$ $\qquad e \in Z_p^a$ iff $a \in f_p(e)$

Encoding action-based control

- Write a formula defining $Plays(\mathcal{A}, \sigma)$: $\pi(X, Z_p^a, \dots)$.
- Write a formula defining $Plays^\omega(\mathcal{A}, \sigma)$: $\pi(X, Z_p^a, \dots)$.
- Say that all paths in $Plays^\omega(\mathcal{A}, \rho)$ satisfy the specification: $\forall X.\pi^\omega(X, Z_p^a, \dots) \Rightarrow \alpha(X)$.
- The required formula is: $\exists Z_p^a \dots \forall X.\pi^\omega(X, Z_p^a, \dots) \Rightarrow \alpha(X)$.

Reduction "process-based" to "action-based"

Observation 1

If there is a process-based controller than 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 α, one can construct $\overline{\mathcal{A}}$ and $\overline{\alpha}$ such that:

action-based controller for $(\overline{\mathcal{A}}, \overline{\alpha})$ exists
iff
process-based controller for $(\mathcal{A}, \alpha)$ exists.

Decidability of MSOL is not necessary

Observation

Thiagarajan's conjecture implies decidability of process-based and action-based control for synchronizing automata.

Definition

A trace alphabet is a co-graph if it does not contain the pattern $x_1 - x_2 - x_3 - x_4$ in its induced graph.

Theorem (Gastin & Lehrman & Zeitoun)

The action-based control problem is decidable for automata over trace alphabets that are co-graphs.

Remark

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

Conclusions

- Non-interleaving semantics have big potential, especially in the context of synthesis.
- While traces are relatively well understood, event-structures are much less studied.
- 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.
- It may well be the case that action based control is decidable for all asynchronous automata.