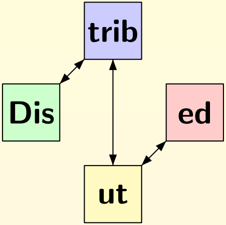
Automatic Synthesis of



Transition Systems

Alin Ştefănescu
(University of Konstanz)

Rennes – March 28th, A.D. 2006

Design of Complex Systems

Specification vs. **Implementation**

Wanted: a correct implementation w.r.t. the specification.

Two approaches:

- Given a specification and an implementation, check if the implementation satisfies the specification

  [Model Checking]

- From a given specification, automatically construct an implementation

  $\longrightarrow$ [Synthesis]

I. Synthesis... In which setting?

Synthesis: The Sequential Case

Specification

Synthesis: The Sequential Case

Specification

$+$

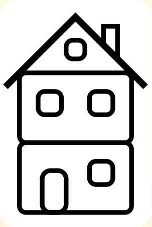
One Agent

$\overset{?}{\Longrightarrow}$

Synthesis: The Sequential Case

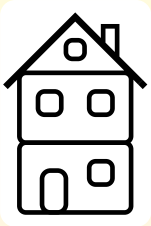
Specification + One Agent $\stackrel{?}{\Rightarrow}$ Implementation

Synthesis: The Distributed Case



Specification

Synthesis: The Distributed Case



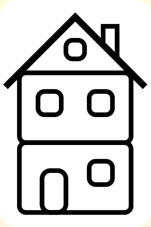
Specification

$+$

Team of
Communicating Agents

$\stackrel{?}{\Rightarrow}$

Synthesis: The Distributed Case

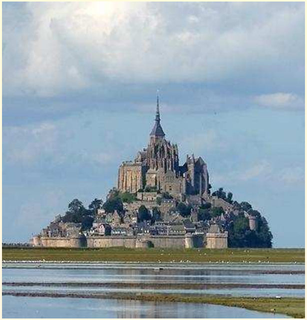


Specification

$+$

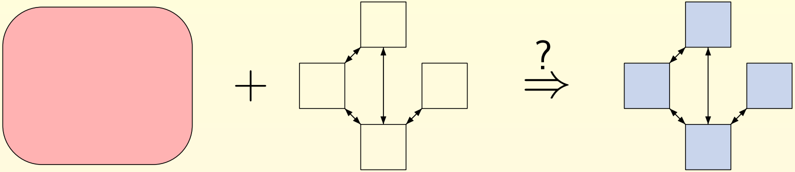
Team of
Communicating Agents

$\stackrel{?}{\Rightarrow}$



Distributed Implementation

The Problem



Labeled Transition System

Distribution

Distributed Transition System

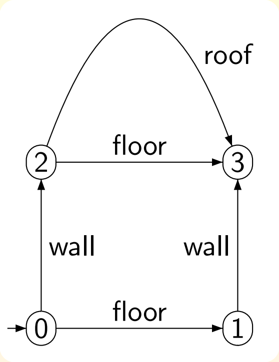
Synthesis of Distributed Transition Systems

Input: Given a labeled transition system *TS* and a **distribution** Δ of actions over a set of agents,

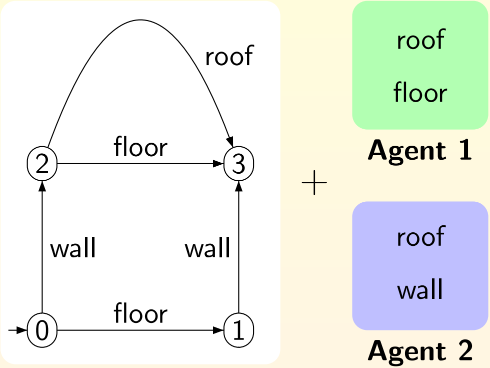
Output: Build, if possible, a distributed transition system over Δ whose global state space is equivalent to *TS*

equivalent : graph-isomorphic / trace-equivalent / bisimilar

Building a House...



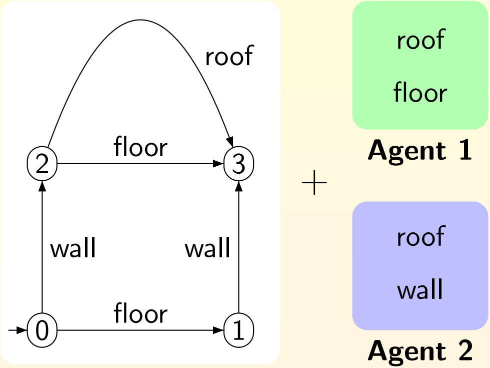
Building a House...



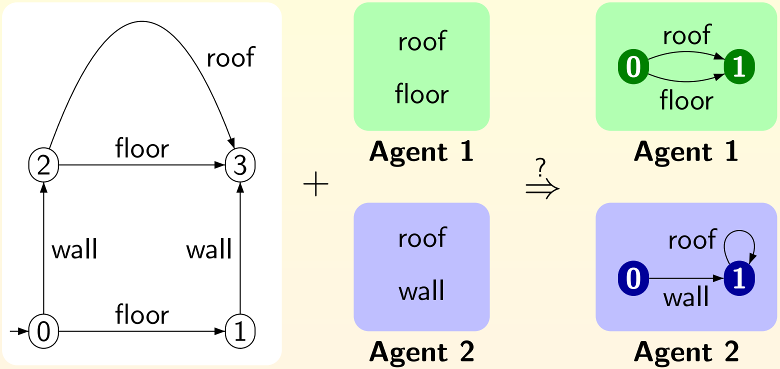
Distribution of {floor,wall,roof} over {1,2}:

- $\Sigma_{local}(1)=\{\text{roof,floor}\}$, $\Sigma_{local}(2)=\{\text{roof,wall}\}$
- dom(roof)={1,2}, dom(floor)={1}, dom(wall)={2}

Building a House...



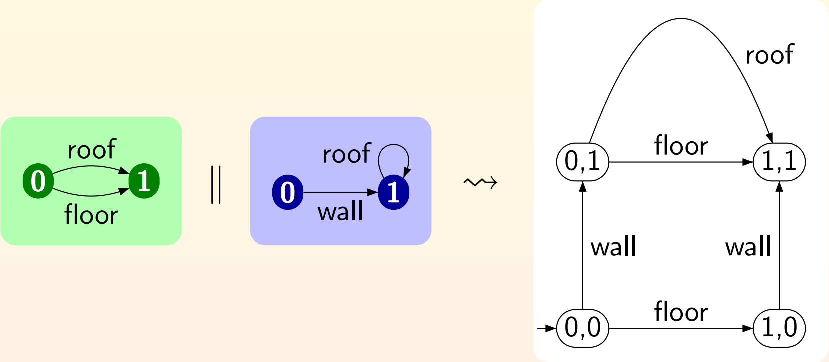
Building a House...



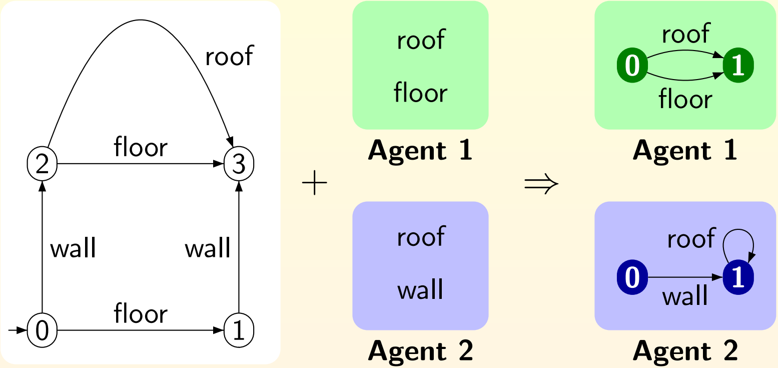
Synchronous Products of Transition Systems

A synchronous product of transition systems consists of a set of local transition systems synchronizing on common actions.

An action is executed if only if all local transition systems from its domain are able to execute that action.

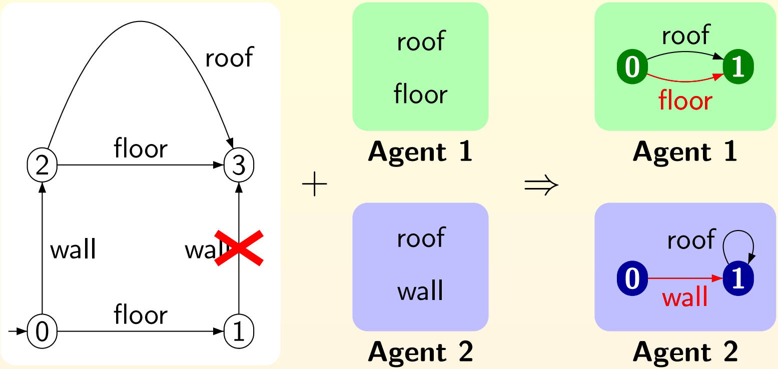


Building a House...



The specification is implementable!

Building a House... Not Always Possible!



When the edge (1,wall,3) is deleted,

the specification is no longer implementable!

Asynchronous Automata

Asynchronous automata [Zielonka87] generalize the synchronous products allowing more communication during synchronization.

An action is executed only for chosen tuples of local states of its domain.

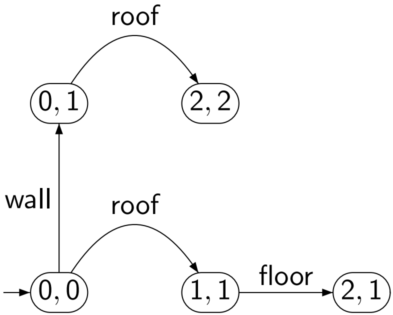
$$1 \longrightarrow_{\mathit{floor}} 2$$

$$0 \longrightarrow_{\mathit{wall}} 1$$

$$(0,0) \longrightarrow_{\mathit{roof}} (1,1)$$

$$(0,1) \longrightarrow_{\mathit{roof}} (2,2)$$

$\rightsquigarrow$



Asynchronous Automata

Asynchronous automata [Zielonka87] generalize the synchronous products allowing more communication during synchronization.

An action is executed only for chosen tuples of local states of its domain.

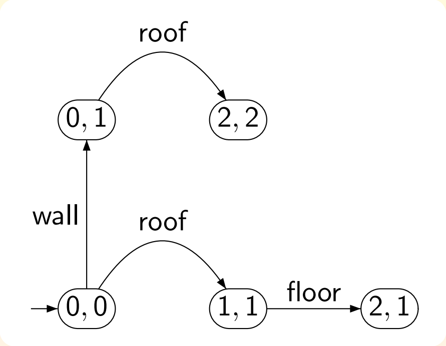
$$1 \longrightarrow_{floor} 2$$

$$0 \longrightarrow_{wall} 1$$

$$(0,0) \longrightarrow_{roof} (1,1)$$

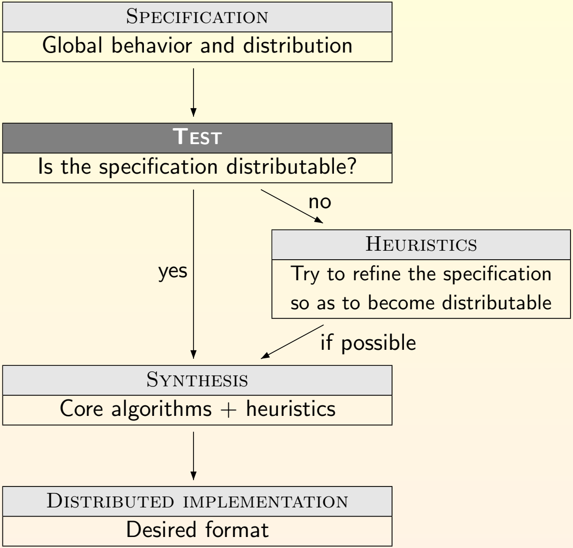
$$(0,1) \longrightarrow_{roof} (2,2)$$

$\rightsquigarrow$



Not implementable as a synchronous product! (cf. wall roof floor)

Synthesis Flow – the whole truth



II. Distributed systems... Characterizations?

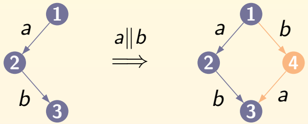
The Diamonds of Independence

A distribution generates an independence relation $\| \subseteq \Sigma \times \Sigma$

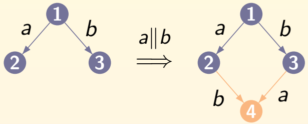
$$a \| b \quad \Leftrightarrow \quad dom(a) \cap dom(b) = \emptyset$$

The independent and forward diamond rules are:

ID



FD



The global state space of a distributed system satisfies ID and FD.

Characterizations

Characterizations of ‘distributable’ global transitions systems given in the literature:

[Zielonka87], [Morin98,99], [CastellaniMukundThiagarajan99]

- modulo isomorphism: theory of regions
  (ID and FD necessary, but not sufficient)
- modulo trace-equivalence:
  - $\rightarrow$ $\mathcal{SP}$: product languages
  - $\rightarrow$ $\mathcal{AA}$: ID and FD necessary and sufficient
- modulo bisimulation: by some modifications of the above

Traces of Distributed Transition Systems

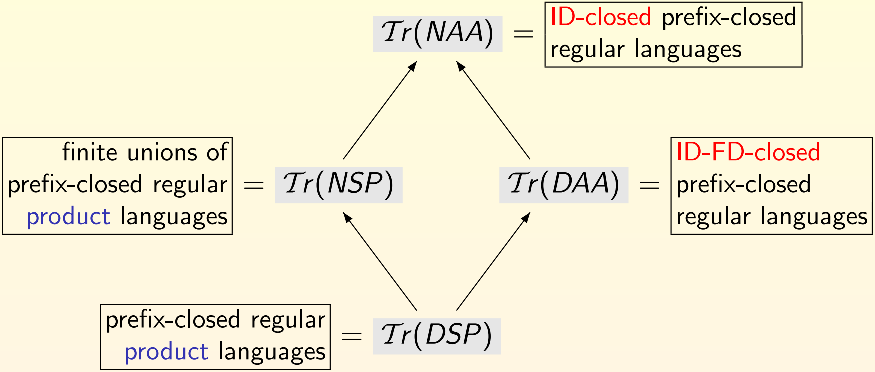
The execution trace language $\mathcal{T}r(TS)$ = the set of all possible finite executions of TS starting in an initial state.

- any execution trace language $\mathcal{T}r(TS)$ is prefix-closed
- For any asynchronous automaton $\mathcal{AA}$, $\mathcal{T}r(\mathcal{AA})$ is ID-closed, i.e., $uabv \in \mathcal{T}r(\mathcal{AA}) \wedge a \| b \Rightarrow ubav \in \mathcal{T}r(\mathcal{AA})$
- For any deterministic asynch. aut. $\mathcal{AA}$, $\mathcal{T}r(\mathcal{AA})$ is FD-closed, i.e., $ua \in \mathcal{T}r(\mathcal{AA}) \wedge ub \in \mathcal{T}r(\mathcal{AA}) \wedge a \| b \Rightarrow uab \in \mathcal{T}r(\mathcal{AA})$

Zielonka's Theorem (variant)

For any prefix-closed ID-FD-closed regular language L, there exists a finite deterministic asynch. automaton $\mathcal{AA}$ with $\mathcal{T}r(\mathcal{AA}) = L$.

Languages of Distributed Transition Systems



Several other variants classified:

$\longrightarrow$ global final states / local final states / acyclic specifications

III. Implementability Test... How difficult?

The Implementability Test

Distributed Implementability

Instance: a transition system TS and
a **distribution** Δ of actions over a set of agents

Question: Is there a distributed transition system over Δ
equivalent with TS?

distributed transition system : $\mathcal{SP}$ / $\mathcal{AA}$
equivalent : isomorphic / trace-equivalent / bisimilar

Previous characterizations provide decision procedures, leading easily to upper bounds. We filled most of the missing lower bounds.

Complexity Bounds Overview

Synchronous products (with one global initial state)

Specification (*TS*)	Isomorphism	Trace Equivalence	Bisim. (determ. impl.)
Nondeterministic Deterministic	NP-complete P [Mor98]	PSPACE-complete	PSPACE-complete

Asynchronous automata (with multiple global initial states)

Specification (*TS*)	Isomorphism	Trace Equivalence	Bisim. (determ. impl.)
Nondeterministic Deterministic	NP-complete P [Mor98]	PSPACE-complete P	P

Complexity Bounds Overview

Synchronous products (with one global initial state)

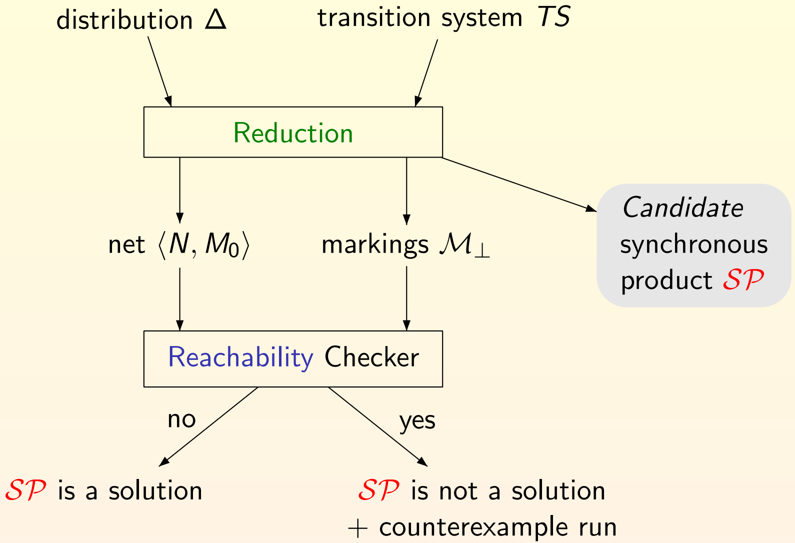
Specification (*TS*)	Isomorphism	Trace Equivalence	Bisim. (determ. impl.)
Nondeterministic Deterministic	NP-complete P [Mor98]	PSPACE-complete	PSPACE-complete
Acyclic & Nondet. Acyclic & Determ.	NP-complete P [Mor98]	coNP-complete	coNP-complete

Asynchronous automata (with multiple global initial states)

Specification (*TS*)	Isomorphism	Trace Equivalence	Bisim. (determ. impl.)
Nondeterministic Deterministic	NP-complete P [Mor98]	PSPACE-complete P	P
Acyclic & Nondet. Acyclic & Determ.	NP-complete P [Mor98]	coNP-complete P	P

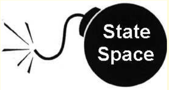
IV. Synthesis of deterministic distributed transition systems... More efficient?

Synthesis of Deterministic Synchronous Products

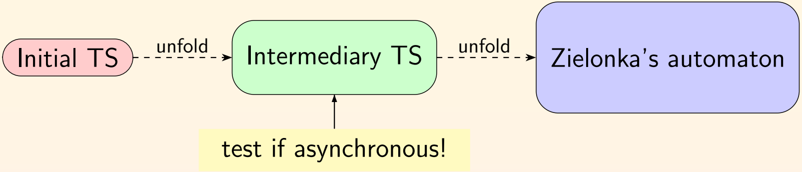


A Heuristic for Smaller Asynchronous Automata

- Zielonka's procedure outputs very large asynchronous automata



- Usually smaller asynchronous automata accepting the same language exist
- **Heuristic idea**
  Unfold the initial transition system guided by Zielonka's construction and test if any of the intermediary transition systems is already asynchronous (modulo isomorphism):



Some Special Cases

Using the characterization for implementability modulo isomorphism, we gave alternatives to Zielonka's construction in the particular cases of:

- transitive distributions
- conflict-free specifications
- acyclic specifications

Relaxed Synthesis

If the initial specification is not 'distributable'...

Relaxed synthesis problem

Given a distribution Δ and a transition system TS, find an asynchronous automaton $\mathcal{AA}$ over Δ such that $\mathcal{T}r(\mathcal{AA}) \subseteq \mathcal{T}r(TS)$ and $\Sigma(\mathcal{AA}) = \Sigma(TS)$.

We proved the above problem to be undecidable.

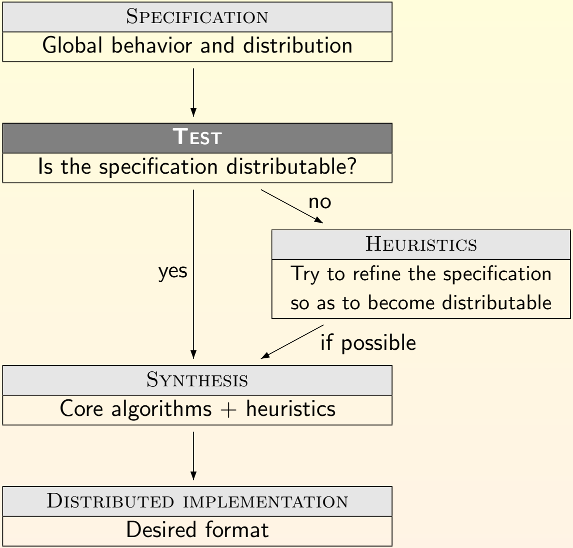
Proposed NP-complete heuristic:

IDFD subautomaton synthesis problem

Given a transition system TS, find a reachable subautomaton $\mathcal{A}$ with $\Sigma(\mathcal{A}) = \Sigma(TS)$ satisfying ID&FD.

V. A Case Study – Mutual exclusion

Synthesis Flow – reloaded



Mutual Exclusion (n=2)

- actions: $\Sigma := \{req_1, enter_1, exit_1, req_2, enter_2, exit_2\}$
- processes: $Proc := \{A_1, A_2, V_1, V_2\}$

	req_1	$enter_1$	$exit_1$	req_2	$enter_2$	$exit_2$
dom	$\{A_1, V_1\}$	$\{A_1, V_2\}$	$\{A_1, V_1\}$	$\{A_2, V_2\}$	$\{A_2, V_1\}$	$\{A_2, V_2\}$

$\longrightarrow$ req_1 and req_2 are independent

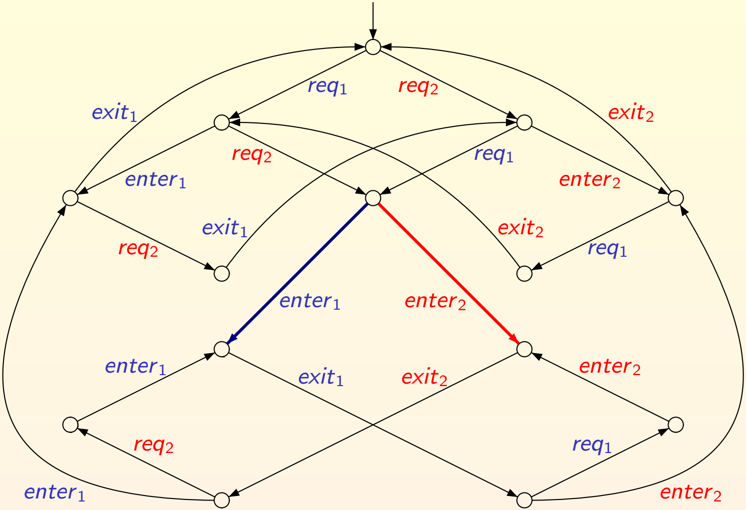
$\longrightarrow$ each action involves only one process and one variable

Regular Specification for *Mutex*(2)

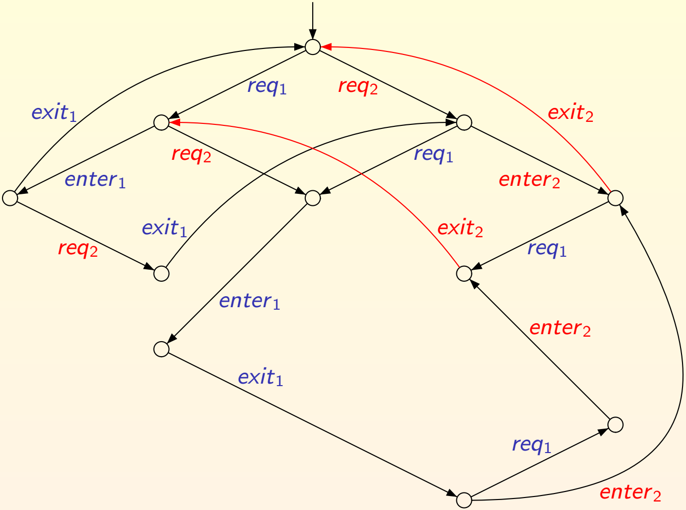
Behavior *Mutex* of a mutual exclusion algorithm:

- the runs are interleavings of the local behaviours $(req_i \; enter_i \; exit_i)^*$
- **forbid** sequences where $enter_1$ is followed by $enter_2$ without $exit_1$ in between (mutual exclusion)
- **forbid** sequences where req_1 is followed by two $enter_2$ without $enter_1$ in between (strong absence of starvation)
- any execution of *Mutex* is the prefix of another execution of *Mutex* (deadlock freedom)

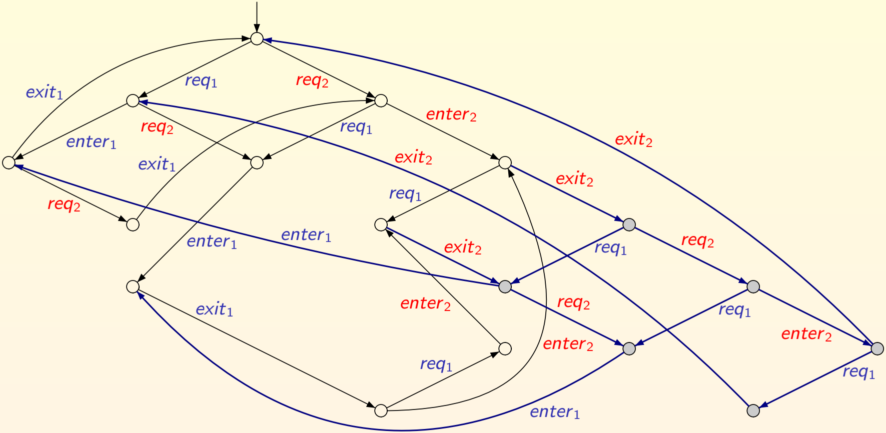
Global Automaton (1) – FD not satisfied



Global Automaton (2) – Distributable



Global Automaton of the Solution



Synthesized *Mutex*(2) Algorithm

Initialization: $v_1 := 0;\ v_2 := 0$

Agent 1 || **Agent 2**

Agent 1

ncs_1: [NCS1];
$\langle$ **case** $(v_1 = 0)$: $v_1 := 1$; **goto** e_1
case $(v_1 = 2)$: $v_1 := 1$; **goto** e'_1
case $(v_1 = 3)$: $v_1 := 4$; **goto** e'_1 $\rangle$
e_1: $\langle$ **await** $v_2 \in \{0, 1\}$ **then**
case $(v_2 = 0)$: **goto** cs_1
case $(v_2 = 1)$: **goto** cs'_1 $\rangle$
e'_1: $\langle$ **await** $v_2 \in \{2, 3\}$ **then**
case $(v_2 = 2)$: $v_2 := 0$; **goto** cs_1
case $(v_2 = 3)$: $v_2 := 1$; **goto** cs'_1 $\rangle$
cs_1: [CS1]; $v_1 := 0$; **goto** ncs_1
cs'_1: [CS1]; $v_1 := 3$; **goto** ncs_1

Agent 2

ncs_2: [NCS2];
$\langle$ **case** $(v_2 = 0)$: $v_2 := 1$; **goto** e_2
case $(v_2 = 2)$: $v_2 := 3$; **goto** e_2
e_2: $\langle$ **await** $v_1 \in \{0, 2, 3, 4\}$ **then**
case $(v_1 = 0)$: $v_1 := 2$; **goto** cs_2
case $(v_1 = 2)$: $v_1 := 0$; **goto** cs_2
case $(v_1 = 3)$: $v_1 := 2$; **goto** cs_2
case $(v_1 = 4)$: $v_1 := 1$; **goto** cs_2 $\rangle$
cs_2: [CS2];
case $(v_2 = 1)$: $v_2 := 2$; **goto** ncs_2
case $(v_2 = 3)$: $v_2 := 0$; **goto** ncs_2

Particularity: Priority is given to the first process in case both processes request access

Prototype Implementations

Prototypes to support the full synthesis cycle:

- Synchronous products:
  - $\longrightarrow$ Via projections on local alphabets [translation to the input of the reachability checkers of the Model-Checking Kit]
- Asynchronous automata:
  - $\longrightarrow$ heuristics for finding an ID-FD subautomata [implementation in the constraint-based logic programming framework Smodels]
  - $\longrightarrow$ unfolding-based heuristics for Zielonka [implementation in C]
- Benchmarks: mutual exclusion, dining philosophers.
  E.g., for mutual exclusion with N processes:
  - $\longrightarrow$ original Zielonka's construction can handle only N=2 processes (specification size: $|TS| = 14$, $|Proc| = 4$, $|\Sigma| = 6$)
  - $\longrightarrow$ our heuristics can handle up to N=5 processes (specification size: $|TS| = 25,537$, $|Proc| = 10$, $|\Sigma| = 15$)

VI. Coming to an end...

Online References

The results presented today can be found online at:

http://www.inf.uni-konstanz.de/∼stefanes/phd-thesis.pdf

or on DBLP:

2005		
3	EE	Keijo Heljanko, Alin Stefanescu: Complexity Results for Checking Distributed Implementability. ACSD 2005: 78-87
2003		
2	EE	Alin Stefanescu, Javier Esparza, Anca Muscholl: Synthesis of Distributed Algorithms Using Asynchronous Automata. CONCUR 2003: 27-41
2002		
1	EE	Alin Stefanescu: Automatic Synthesis of Distributed Systems. ASE 2002: 315

Today's menu

Synthesis of synchronous products and asynchronous automata:

- A careful study and survey of characterizations of the global structure (graph isomorphism) and behaviors (traces of executions) of the two theoretical models with several variants
- Matching computational complexity bounds for the implementability tests for several combinations
- Alternatives to Zielonka's construction in special cases
- Several heuristics for finding smaller synthesized solutions
- Prototype implementations for most of the algorithms

Merci de votre patience !

Appendix

Synchronous Products (Formally)

A synchronous product of transition systems $\mathcal{SP}$ over a distribution $(\Sigma, Proc, \Delta)$ consists of

- a set of *local* state spaces $(Q_p)_{p \in Proc}$ and
- a set of *local* transitions relations $(\rightarrow_p)_{p \in Proc}$ with $\rightarrow_p \subseteq Q_p \times \Sigma_{local}(p) \times Q_p$.

The global state space of $\mathcal{SP}$ consists of the global states $Q \subseteq \prod_{p \in Proc} Q_p$ *reachable* from a set of initial global states I by

$$(q_p)_{p \in Proc} \xrightarrow{a} (q'_p)_{p \in Proc} \Leftrightarrow \begin{cases} q_p \xrightarrow{a}_p q'_p & \text{for all } p \in dom(a) \\ q_p = q'_p & \text{for all } p \notin dom(a) \end{cases}$$

Asynchronous Automata (Formally)

An asynchronous automaton $\mathcal{AA}$ over a distribution $(\Sigma, Proc, \Delta)$ consists of

- a set of *local* state spaces $(Q_p)_{p \in Proc}$ and
- a set of *local* transition relations $(\rightarrow_a)_{a \in \Sigma}$ with $\rightarrow_a \subseteq \prod_{p \in dom(a)} Q_p \times \prod_{p \in dom(a)} Q_p$

The global state space of $\mathcal{AA}$ consists of the global states $Q \subseteq \prod_{p \in Proc} Q_p$ *reachable* from a set of initial global states I by

$$(q_p)_{p \in Proc} \xrightarrow{a} (q'_p)_{p \in Proc} \Leftrightarrow \begin{cases} (q_p)_{p \in dom(a)} \rightarrow_a (q'_p)_{p \in dom(a)} \text{ and} \\ q_p = q'_p \quad \text{for all } p \notin dom(a). \end{cases}$$

Characterization of Async. Automata modulo Isomorphism

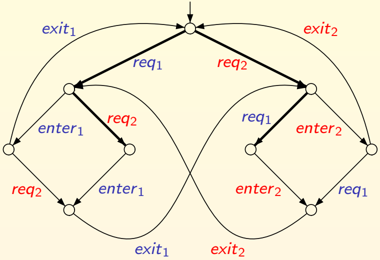
Theorem (Morin99)

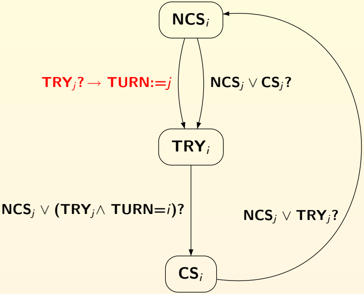
Let $(\Sigma, Proc, \Delta)$ be a distribution and $TS = (Q, \Sigma, \to, I)$ be a transition system. Then, TS is isomorphic to an asynchronous automaton over Δ if and only if for each $p \in Proc$ there exists an equivalence relation $\equiv_p \subseteq Q \times Q$ with:

- AA_1: *If $q_1 \xrightarrow{a} q_2$, then $q_1 \equiv_{Proc \setminus dom(a)} q_2$.*
- AA_2: *If $q_1 \equiv_{Proc} q_2$, then $q_1 = q_2$.*
- AA_3: *If $q_1 \xrightarrow{a} q_1'$ and $q_1 \equiv_{dom(a)} q_2$, then there exists q_2' such that $q_2 \xrightarrow{a} q_2'$ and $q_1' \equiv_{dom(a)} q_2'$.*

Mutual Exclusion – A Classical Solution

[EmersonClarke82] automatically synthesise a *Mutex* alg. from a CTL spec.





- actions req_1 and req_2 are **not** independent
- involved implementation (req_i tests TRY_j, update TURN, moves to TRY_i)

Regular Specification for *Mutex*(2)

$Spec \subseteq \Sigma^*$ for $Mutex(2)$ can be chosen such that:

- $Spec \subseteq$ shuffle(prefix($(req_1 enter_1 exit_1)^*$), prefix($(req_2 enter_2 exit_2)^*$))
- $Spec \subseteq \Sigma^* \setminus [\Sigma^* enter_1 (\Sigma \setminus exit_1)^* enter_2 \Sigma^*]$ and its dual (mutual exclusion)
- $Spec \subseteq \Sigma^* \setminus [\Sigma^* req_1 (\Sigma \setminus enter_1)^* enter_2 (\Sigma \setminus enter_1)^* enter_2 \Sigma^*]$ and its dual (absence of starvation)