Synthesis of Asynchronous Automata
-a desired new approach-

Work: Alin Ştefănescu
Supervision: Prof. Dr. Javier Esparza
GKLI Kolloquium: First Day of Summer 2001

Overview

{ The problem }

{ History }

{ Definitions }

{ Examples }

{ Our approach }

{ Applications }

{ Conclusions }

Synthesis Problem (1/2)

- Given the [**specification**] of a system, (automatically) construct a [**system**] that satisfies the specification.
- Given [**all possible runs of a machine**], (automatically) construct a [**machine**] having this behaviour.
- Given [**all possible runs of an algorithm and a distribution structure**], (automatically) construct a [**distributed algorithm**] having the described distributed behaviour.

Synthesis Problem (2/2)

Let us fix the framework and see

- the algorithm as a *finite automaton*;
- all the runs of the algorithm as the accepted regular language when all the states are final and reachable;
- the distribution structure as a number of processors being able to execute a certain set of actions from the alphabet of the automaton.

The synthesis problem would be:

Given a [**regular language and a distribution structure**], (automatically) construct an [**asynchronous automaton**] that accepts it, in case there exist one.

Traces

Introduced by Mazurkiewicz ['77] for the theory of concurrent systems

- **independence alphabet**: a pair $(\Sigma, \|)$ where Σ is a finite alphabet (of actions) and $\| \subseteq \Sigma \times \Sigma$ is a symmetric and irreflexive relation called *independence relation*
- **monoid of traces** : $M(\Sigma, \|) := \Sigma^* / \sim$, where $\sim$ is the congruence generated by the independence relation $\{w\, \underline{ab}\, w' \sim w\, \underline{ba}\, w' \mid w, w' \in \Sigma^* \text{ and } a \| b\}$
- **trace**: an equivalence class in $M(\Sigma, \|)$

Comprehensive reference: Diekert, Rozenberg - *The Book of Traces* ['95]

Recognizable Trace Languages

The language $L \subseteq M(\Sigma, \|)$ is **recognizable** iff there exist a finite monoid S and a homomorphism $\eta : M(\Sigma, \|) \to S$ such that $L = (\eta^{-1} \circ \eta)(L)$ iff the syntactic equivalence $\sim_L$ is of finite index.

$$v \sim_L w \quad \Leftrightarrow \quad \forall x, y \in M(\Sigma, \|)\; xvy \in L \text{ iff } xwy \in L$$

Let $\mathrm{Rec}(\Sigma, \|)$ be the class of all recognizable trace languages over $(\Sigma, \|)$

- **Ochmański**'s theorem ['85]: $\mathrm{Rec}(\Sigma, \|) =$ the smallest class containing finite trace languages and closed under union, catenation and *concurrent* iteration
- **Zielonka**'s theorem ['87]: $\mathrm{Rec}(\Sigma, \|) =$ the class of languages accepted by deterministic asynchronous automata

Beyond Zielonka

Because of its complicated solution...

> *The question whether a substantially simpler construction for Zielonka's theorem exists, is still open.* (from 'The Book of Traces' ['95], p. 250)

More recent work:

- **Pighizzini** ['93] + **Klarlund et al.** ['94]: a new solution for Zielonka's theorem based on Ochmański's result
- **Morin** ['99]: an extension of Zielonka's result regarding the distribution structure based on a nice characterization of asynchronous automata

Question: Is it possible, even under certain restrictions, to use Morin's result in the (re)search for a 'simpler construction'?

Distribution Structure

A **distribution** Δ of the actions $(\Sigma, \|)$ over a set of processors *Proc* is a *domain* relation $dom \subseteq \Sigma \times Proc$ such that

$$\forall a, b \in \Sigma : \quad a \| b \Leftrightarrow dom(a) \cap dom(b) = \emptyset.$$

Refinement: a **communication structure** is a pair (R, W) of *read* and *write* relations in $\Sigma \times Proc$ with $W \subseteq R$ such that

$$\forall a, b \in \Sigma : [\, a \| b \Rightarrow W(a) \cap R(b) = \emptyset \,] \text{ and } [\, R(a) \cap R(b) = \emptyset \Rightarrow a \| b \,].$$

Choice: because the results are similar, for the sake of simplicity we restrict ourselves to the *classical* case $dom = R = W$.

Asynchronous Automata

An **asynchronous automaton** with distribution $\Delta = (Proc, \Sigma, \|)$ is an independent automaton $\mathcal{A} = (Q, \Sigma, \longrightarrow, S, F, \|)$ for which there are

- a family of sets of local states $(Q_k)_{k \in Proc}$ such that $Q \subseteq \prod_{k \in Proc} Q_k$
- for each action $a \in \Sigma$, $\delta_a \subseteq \prod_{k \in dom(a)} Q_k \times \prod_{k \in dom(a)} Q_k$
- S and F subsets of Q as starting and final states

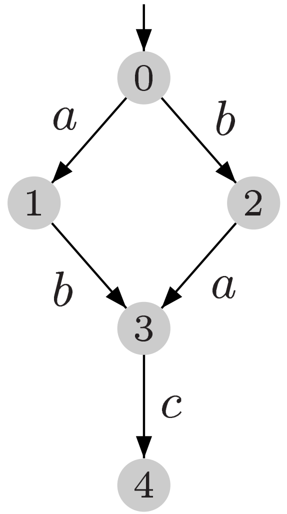
such that the transitions of $\mathcal{A}$ respect the following synchronization rule

$$(q_k)_{k \in Proc} \xrightarrow{a} (q'_k)_{k \in Proc} \Leftrightarrow \begin{cases} \forall k \notin dom(a), q'_k = q_k \\ \big((q_k)_{k \in dom(a)}, (q'_k)_{k \in dom(a)}\big) \in \delta_a \end{cases}$$

If all δ_a's are functions and $card(S) = 1$, then $\mathcal{A}$ is called *deterministic*.

Example

Independent Automaton



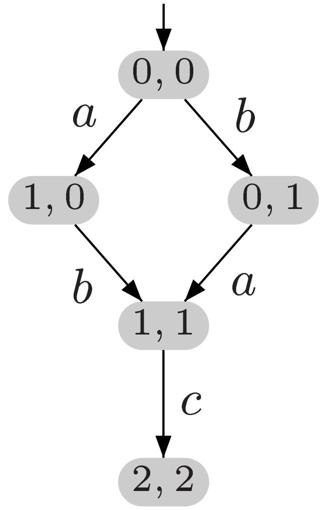
$$\Sigma = \{a, b, c\}$$

$$a \| b$$

$$Proc = \{p_1, p_2\}$$

	dom
a	p_1
b	p_2
c	p_1, p_2

Asynchronous Automaton



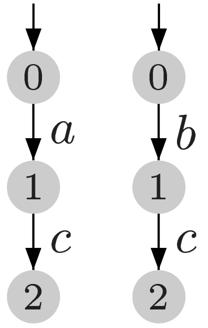
$$Q_{p1} = \{0, 1, 2\}$$

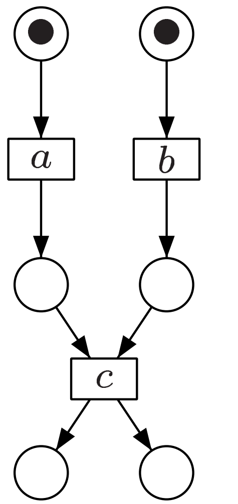
$$Q_{p2} = \{0, 1, 2\}$$

$$\delta_a(0) = 1, \quad \delta_b(0) = 1,$$

$$\delta_c(1,1) = (2,2)$$

Visual Alternatives





Morin's Result

Theorem: Let $\mathcal{A}$ be a deterministic independent automaton and $\Delta = (Proc, \Sigma, \|)$ a distribution structure. There is a the least family of equivalences $(\equiv_k)_{k \in Proc}$ over the states of $\mathcal{A}$ such that

DE$_1$: $q \xrightarrow{a} q' \wedge k \notin dom(a) \Rightarrow q \equiv_k q'$

DE$_2$: $$\left.\begin{array}{c} q_1 \xrightarrow{a} q_1' \wedge q_2 \xrightarrow{a} q_2' \\ \forall k \in dom(a), q_1 \equiv_k q_2 \end{array}\right\} \Rightarrow \forall k \in dom(a), q_1' \equiv_k q_2'$$

Moreover $\mathcal{A}$ is **equivalent** with an asynchronous automaton with distribution Δ iff the two following conditions are satisfied:

DS$_1$: $\forall q_1, q_2 \in Q : [\forall k \in Proc, q_1 \equiv_k q_2] \Rightarrow q_1 = q_2$

DS$_2$: $\forall a \in \Sigma, \forall q_1, q_2 \in Q : \left[q_1 \xrightarrow{a} \wedge \forall k \in dom(a), q_1 \equiv_k q_2\right] \Rightarrow q_2 \xrightarrow{a}$

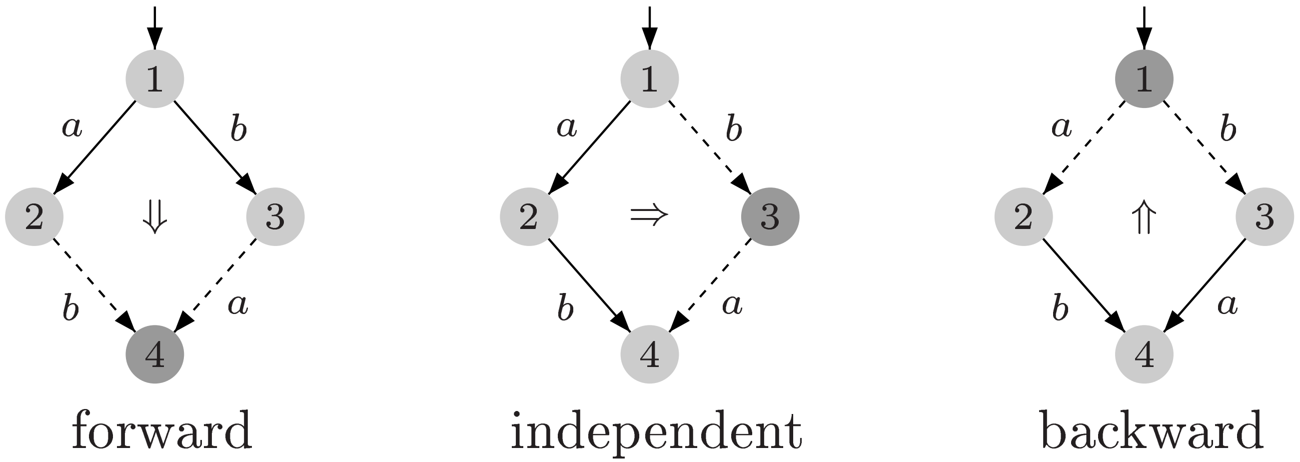
Diamonds Rules

Let the *forward*, *independent* and *backward* diamond rules be:

FD: $q_1 \xrightarrow{a} q_2 \wedge q_1 \xrightarrow{b} q_3 \wedge a \| b \Rightarrow \exists q_4 \in Q, q_2 \xrightarrow{b} q_4 \wedge q_3 \xrightarrow{a} q_4$

ID: $q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_4 \wedge a \| b \Rightarrow \exists q_3 \in Q, q_1 \xrightarrow{b} q_3 \xrightarrow{a} q_4$

BD: $q_1 \xrightarrow{a} q \wedge q_2 \xrightarrow{b} q \Rightarrow a \| b \wedge q_1 \neq q_2 \wedge \exists q_0 \in Q, q_0 \xrightarrow{b} q_1 \wedge q_0 \xrightarrow{a} q_2$



Fact: Any asynchronous automaton satisfies the **FD** and **ID** rules

Unfolding Technique (1/2)

Conjecture: For any *distribution* that is a covering by cliques for the dependence relation and for any deterministic finite *automaton* that satisfy FD and ID rules, there exists an asychronous automaton **bisimilar** with the given automaton.

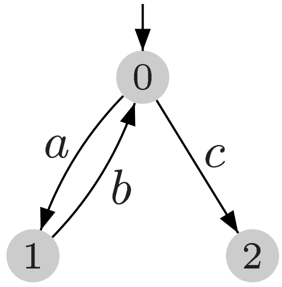
Idea:

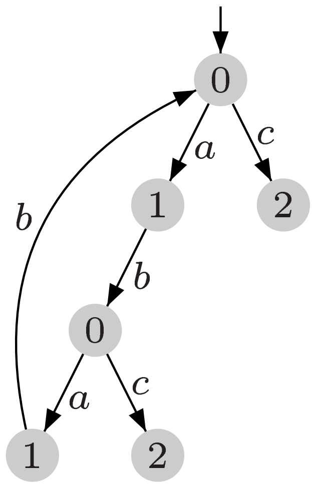
- Unfold the automaton into the tree of all accepted 'traces', identifying states with the same label only when this is required by the ID and FD rules
- Identify the states with the same label such that after each step the conditions DS_1 and DS_2 are fulfilled

Duty:

The above procedure yields a **finite** automaton

Unfolding Technique (2/2)





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

$$Indep = \emptyset$$

$$Proc = \{p_1, p_2, p_3\}$$

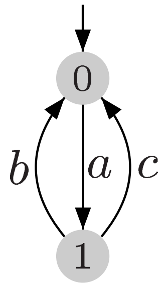
	dom
a	p_1, p_2
b	p_2, p_3
c	p_1, p_3

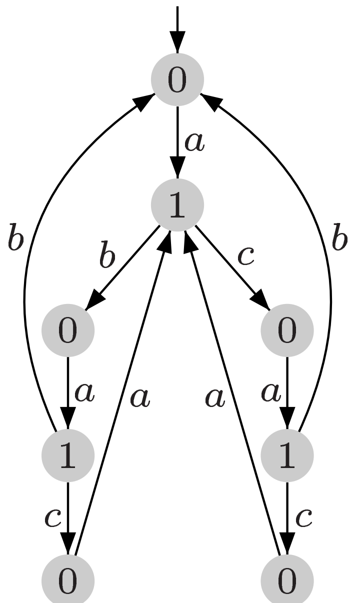
Automata above are not asynchronous:

$0 \equiv_{p_1,p_3} 1$, so $0 \equiv_{dom(c)} 1$

but only one of 0 and 1

has an out-transition $\xrightarrow{c}$

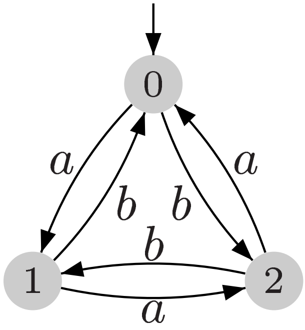




Acyclic Case

Theorem: For any *acyclic*, independent automaton that fulfills FD, ID and BD rules, there exists an asynchronous automaton **equivalent** with it.

Cyclic counterexample: $a \| b$



Corollary: For any *acyclic*, independent automaton that fulfills ID rule, there exists an asynchronous automaton that **accepts** the same language.

> *Proof*: We unfold the automaton and we identify states only for ID and BD reasons. We complete this automaton with non-final states to have the FD rule satisfied. Then we use the previous theorem.

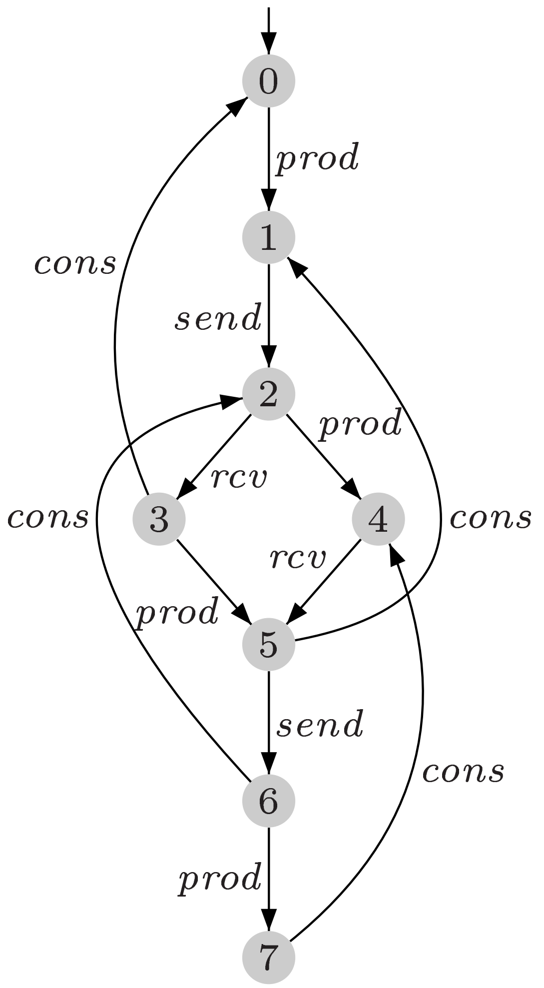
Producer-Consumer (1/2)

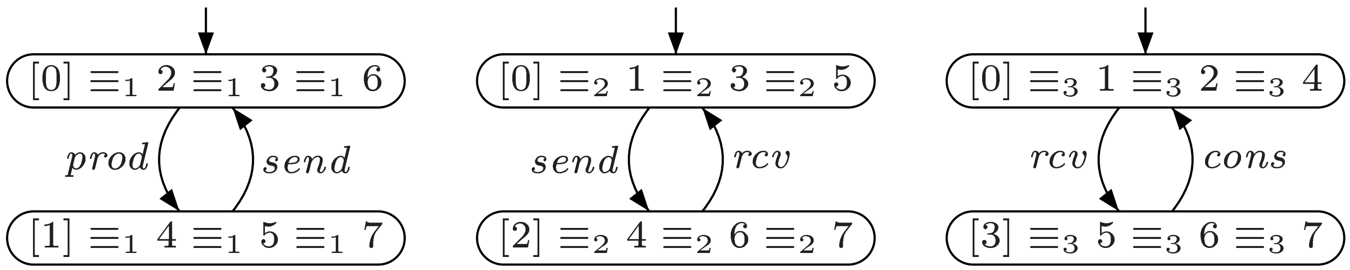
We model the trade between a [**producer**] and a [**consumer**] by:

- actions $\Sigma := \{prod, send, rcv, cons\}$
- regular specification S:
  - $S \subseteq \{u_1 v_1 u_2 v_2 \ldots u_k v_k \mid k \geq 1, u_1 u_2 \ldots u_k \in (prod\ send)^*, v_1 v_2 \ldots v_k \in (rcv\ cons)^*\}$, so S is in the shuffle of 'producer' and 'consumer'
  - $S \subseteq (T^* send\, T^* rcv\, T^*)^*$, with $T = (\Sigma \backslash \{send, rcv\})$
- independences: $prod \| cons$, $prod \| rcv$, $send \| cons$
- distribution on three processors:

	$prod$	$send$	rcv	$cons$
dom	p_1	p_1, p_2	p_2, p_3	p_3

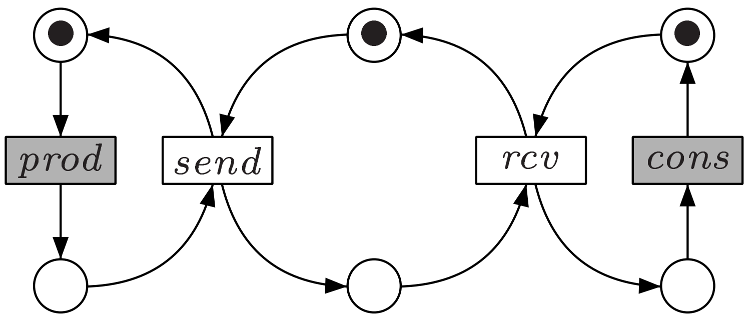
Producer-Consumer (2/2)





$$\delta_{prod}([0]_1) = [1]_1 \quad \delta_{send}([1]_1, [0]_2) = ([0]_1, [2]_2)$$

$$\delta_{cons}([3]_3) = [0]_3 \quad \delta_{rcv}([2]_2, [0]_3) = ([0]_2, [3]_3)$$



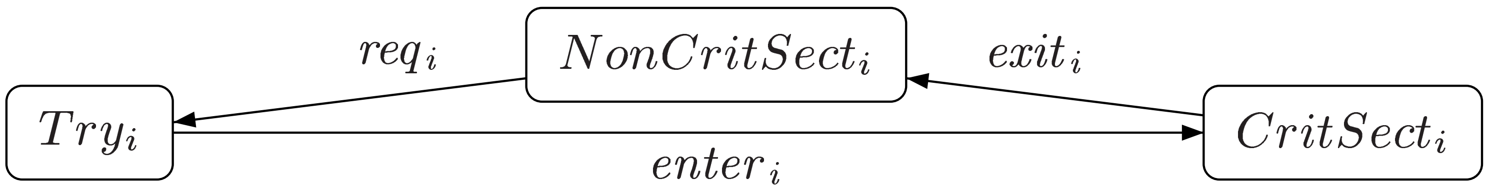
Mutual Exclusion (1/3)

A **mutual exclusion** situation appears when at least two processes are trying to access for 'private' use a common resource (e.g. two candidates for presidency trying to talk to the nation in a pre-election TV show).

For two processes we can choose the following actions

$$\Sigma := \{req_1, enter_1, exit_1, req_2, enter_2, exit_2\},$$

every concurrent process should behave like this



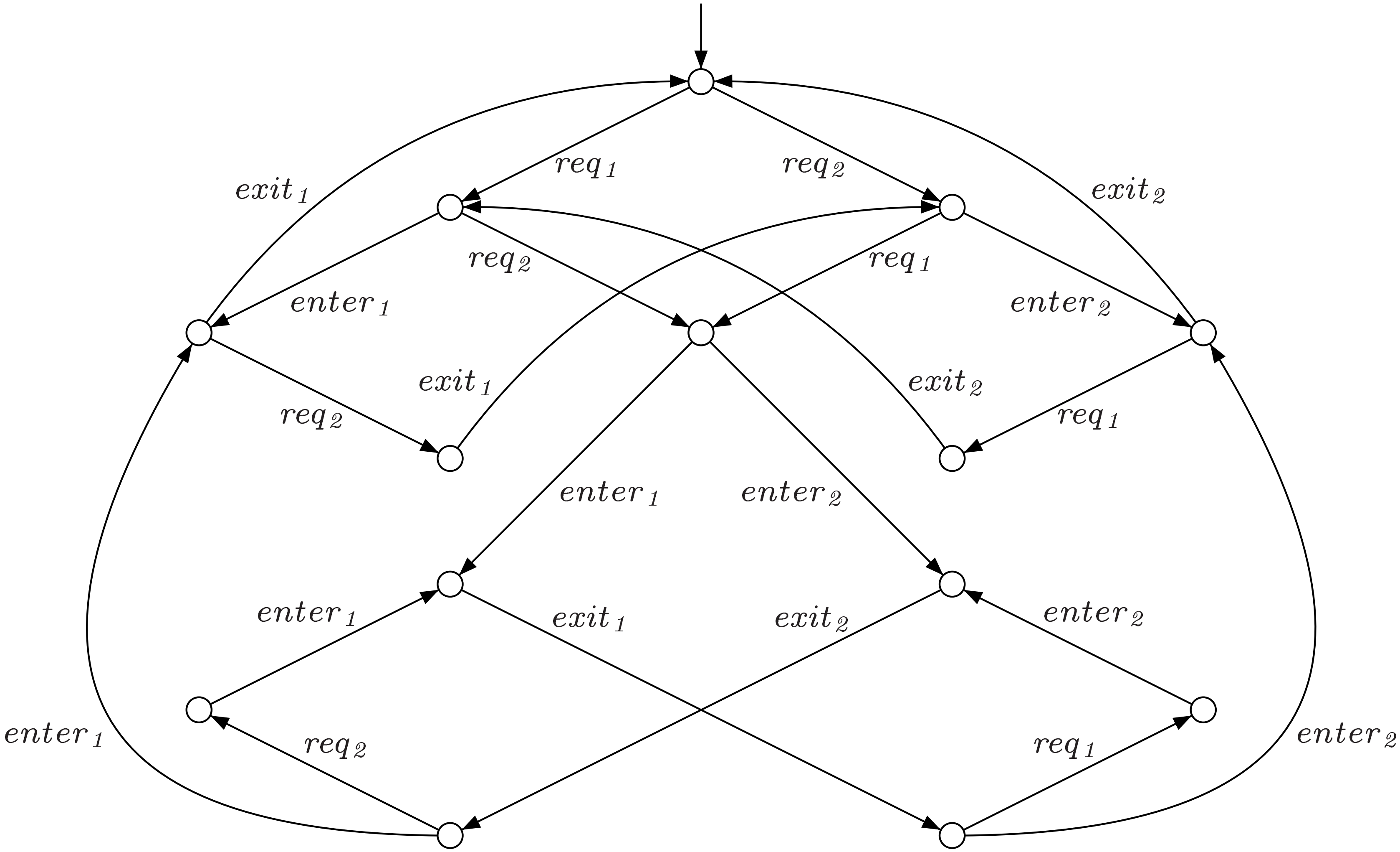
and to have the independences: $req_1 \| req_2$, $exit_1 \| exit_2$, $req_1 \| exit_2$, $exit_1 \| req_2$

Mutual Exclusion (2/3)

The regular specification should be the minimal to fulfill:

- $S \subseteq \Sigma^*$ is a regular prefix-closed language
- $S \subseteq \{u_1 v_1 u_2 v_2 \ldots u_k v_k \mid k \geq 1, u_1 u_2 \ldots u_k \in (\mathit{req}_1\, \mathit{enter}_1\, \mathit{exit}_1)^*, v_1 v_2 \ldots v_k \in (\mathit{req}_2\, \mathit{enter}_2\, \mathit{exit}_2)^*\}$, so S is in the shuffle of $(\mathit{req}_i\, \mathit{enter}_i\, \mathit{exit}_i)^*$, $i = 1, 2$
- **[liveness]** $S \subseteq \Sigma^* \setminus [\Sigma^* \mathit{req}_1 (\Sigma \setminus \mathit{enter}_1)^* \mathit{enter}_2 (\Sigma \setminus \mathit{enter}_1)^* \mathit{enter}_2 \Sigma^*]$ and its symmetric version
- **[mutual exclusion]** $S \subseteq \Sigma^* \setminus [\Sigma^* \mathit{enter}_1 (\Sigma \setminus \mathit{exit}_1)^* \mathit{enter}_2 \Sigma^*]$ and its symmetric version
- **[deadlock freedom]** for any word $w \in S$, there exists an action $a \in \Sigma$ such that $wa \in S$

The Automaton



Mutual Exclusion (3/3)

Using Morin's result our automaton can be distributed over

$$P_1 = \{req_1, enter_1, exit_1, enter_2\},\ P_2 = \{req_2, enter_2, exit_2, enter_1\}$$

Making simplifications we get the implementation using two variables (v_1, v_2):

while $(true)$ do	while $(true)$ do
[Noncritical Section 1]	[Noncritical Section 2]
$v_1 := \delta_{req_1}(v_1)$	$v_2 := \delta_{req_2}(v_2)$
repeat	repeat
if $(v_1, v_2) \in Dom\, \delta_{enter_1}$ then	if $(v_1, v_2) \in Dom\, \delta_{enter_2}$ then
$(v_1, v_2) := \delta_{enter_1}(v_1, v_2)$	$(v_1, v_2) := \delta_{enter_2}(v_1, v_2)$
until $(v_1, v_2) \in Codom\, \delta_{enter_1}$	until $(v_1, v_2) \in Codom\, \delta_{enter_2}$
[Critical Section 1]	[Critical Section 2]
$v_1 := \delta_{exit_1}(v_1)$	$v_2 := \delta_{exit_2}(v_2)$
od	od

Final Remarks

Done:

{ 'Visits' in the area of the problem }

{ Intuitive unfoldings for many examples }

{ Implementation of Morin's algorithm }

To Do:

{ Fix up the unfolding procedure }

{ Implementation of the procedure }

{ More case studies }

... Thank You! ...