Compositional Semantics of Non-deterministic Synchronous Languages*

(Extended Abstract)

F. Maraninchi and N. Halbwachs

VERIMAG**, Miniparc – ZIRST, F-38330 Montbonnot
email: {Florence.Maraninchi|Nicolas.Halbwachs}@imag.fr

Abstract. A key feature of synchronous languages is that they conciliate concurrency with determinism. This is achieved by detecting so-called causality errors [BG92, HM95, STB95], which occur when the composition of deterministic processes is not deterministic. Although the existing synchronous languages only deal with deterministic programs, it is not a restriction of the approach. We show how a synchronous language can be extended with non deterministic processes, while still ensuring that process composition will never *introduce* non determinism. We give an operational semantics of the language, and show that the equivalence relation induced by this semantics is a congruence for all the language constructs.

1 Introduction

Synchronous languages [IEE91, Hal93] have been proposed for programming reactive systems. Their main feature is that they conciliate *concurrency* with *determinism*: They offer a *logical* notion of concurrency to structure programs, together with a powerful synchronization mechanism, while ensuring that the concurrent composition of deterministic processes remains deterministic. This feature is very convenient to *program* reactive systems, since these systems are almost always deterministic.

But determinism is no longer a desirable property if the language is to be used for *specifying*, rather than for programming systems. Indeed, a non-deterministic language may be used to describe a *set* of behaviors in a concise way. This motivates our attempt to introduce non-deterministic processes in a synchronous language, while still ensuring that the composition of deterministic processes is deterministic. In other words, the composition of processes must not *introduce* non-determinism.

On the other hand, non-determinism may be simulated in existing synchronous languages by controlling it through auxiliary inputs sometimes called *oracles*. Following this idea, we can try to compose non-deterministic processes

* This work has been partly supported by EEC under ESPRIT-BRA action "REACT".

** Verimag is a joint laboratory of CNRS, Institut National Polytechnique de Grenoble, Universite J. Fourier and Verilog SA associated with IMAG.

by (1) introducing oracles, (2) composing the resulting deterministic processes in the usual way, and then (3) "removing" the oracles. This three-steps semantics induces an equivalence of non-deterministic systems: two systems are equivalent if they have the same image. It is interesting to see whether this equivalence coincides with the bisimulation-based equivalences of non-deterministic processes used in process algebra theory. One can also ask whether this equivalence is a congruence for the composition operators of a particular language.

We propose to introduce non-determinism in a synchronous language, and to study these questions.

First, we define Input/Output Boolean Automata, to serve as basic components. They are an extension to input/output-structured labels of the Boolean Transition Systems used in [CGS91] to represent S/R-processes [Kur89]. In spite of their name, they cannot easily be compared to IO-automata [LT89], where one can indeed distinguish between inputs and outputs, but cannot express that they should be simultaneous. The set of programs we consider is generated from the set of input/output Boolean automata provided with two operations: the synchronous product and the encapsulation. The latter is the most interesting one. It expresses the semantics of the *synchronous broadcast*, and differs from the simple constructs of the process algebra studied in [CGS91], where processes communicate through shared variables.

In a deterministic synchronous language, the basic components are deterministic, and the composed processes should be deterministic too. We recall that the product preserves this property, while the encapsulation does not. This leads to the notion of incorrect programs, which is simple way of expressing the well-known causality problems intrinsic to imperative synchronous languages like Esterel [BG92], Statecharts [Har84], or Argos [Mar92]: a program is incorrect if its global behavior is not deterministic or not reactive (i.e., not always receptive to all input events), while all its basic components are.

In a non-deterministic synchronous language, the basic components and the composed processes are allowed to be non-deterministic. But there is still a requirement: combining processes must not *introduce* non-determinism. It may, at most, preserve the non-determinism of the components. There is a notion of incorrect programs in this case too, but it cannot be defined in terms of the determinism of the composed processes. We have to check whether combining processes has *introduced* non-determinism. We propose to do so by determinizing the non-deterministic basic components by means of oracles, and by observing the composition of these deterministic objects. We will see that, surprisingly, this simple idea only works under strong constraints about the way oracles are introduced, and that its consistency is not that obvious to establish.

Finally, we show that the bisimulation extended to input/output labels is a congruence for the language constructs. This gives the formal basis for the semantics of a complete non-deterministic synchronous language, including causality error detection.

Only the main four proofs are given in the Appendix (Properties 4.2, 4.3, 4.3, and 4.4). All proofs will be included in the full paper.

2 Definitions and notations

If X is a finite set of variables, we will note $\mathcal{B}(X)$ the set of Boolean formulas (e.g., "$a \wedge (\neg b \vee c)$") with variables in X — which is isomorphic to the set of functions from 2^X to $\{0,1\}$ — and $\mathcal{M}(X)$ the set of complete Boolean monomials (e.g., "$a \wedge \neg b \wedge c$") with variables in X — which is isomorphic to 2^X. tt and ff respectively denote the identically true and false formulas.

2.1 Input/output Boolean automata

Definition 1 (input/output Boolean automata). An input/output Boolean automaton (IOB for short) is a tuple (S, s_0, I, O, T) where S is a set of *states*, $s_0 \in S$ is the *initial* state, I is the set of inputs, O is the set of outputs, and $T \subseteq S \times \mathcal{B}(I) - \{\mathsf{ff}\} \times 2^O \times T$ is the set of transitions. □

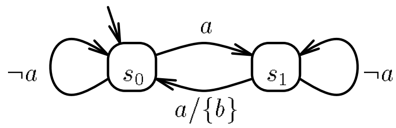
The first part of a transition label represents an *input condition* (guard). The variables in the Boolean formula represent the presence of input *signals*. The second part of a transition label is a set of *output* signals, emitted by the transition. Intuitively, a transition is validated by a set of present signals if its guard is satisfied; its firing provokes the *simultaneous* presence of its output signals.

Notations:

- If $o \subseteq O$ is a set of signals, we note $Bool_O(o)$ its characteristic Boolean formula. For instance, $Bool_{\{a,b,c\}}(\{a,c\}) = a \wedge \neg b \wedge c$.
- If $X = \{x_1, \ldots, x_n\}$ is a set of variable names, and if b is a Boolean formula, $\exists X.(b)$ denotes the formula $\exists x_1. \ldots \exists x_n.(b)$.
- A transition $(s_1, b, o, s_2) \in T$ will be denoted by: $s_1 \xrightarrow{b/o} s_2$.
- The set of input/output Boolean automata is denoted by IOB.

Example 1.

The IOB drawn besides emits a signal "b" whenever it has received "a" twice.



□

Definition 2 (bisimulation of input/output Boolean automata). Two IOBs $A_1 = (S_1, s_{01}, I, O, T_1)$ and $A_2 = (S_2, s_{02}, I, O, T_2)$ are *bisimilar* (noted $A_1 \approx A_2$) if and only if there exists an equivalence relation $\mathcal{R} \subseteq S_1 \times S_2$ such that $s_{01} \mathcal{R} s_{02}$ and

$$s \mathcal{R} s' \implies \begin{cases} s \xrightarrow{b/o} r \implies \begin{cases} \exists b_1, \ldots, b_m, r'_1, \ldots, r'_m \text{ such that} \\ \quad \forall i \in [1,m],\ s' \xrightarrow{b_i/o} r'_i \text{ and } r \mathcal{R} r'_i \\ \quad \text{and } b \Rightarrow (\bigvee_i b_i) \end{cases} \\ \textit{and conversely.} \end{cases}$$

□

This notion of bisimulation is an extension to input/output-structured labels of the equivalence defined in [CGS91] for Boolean Processes. It allows a transition of one system to be mimicked by a *set* of transitions of the other. As usual, this bisimulation is extended to states:

Definition 3 (equivalence of states). Let s, s' be two states of an IOB (S, s_0, I, O, T). Then $s \approx s'$ iff $(S, s, I, O, T) \approx (S, s', I, O, T)$. □

Property 4 (monomial form). For any IOB A, there exists an IOB bisimilar to A, all the transitions of which are guarded by complete monomials ($\in \mathcal{M}(I)$). □

(Of course any guard b can be expressed as a disjunction $\bigvee_i m_i$ of complete monomials, so a transition $s \xrightarrow{b/o} s'$ can be replaced by a bundle of transitions $s \xrightarrow{m_i/o} s'$). We will often assume that IOBs are put in this monomial form, to make the proofs simpler.

Definition 5 (determinism). An IOB (S, s_0, I, O, T) is deterministic if and only if $\forall s \in S$,

$$\left(s \xrightarrow{b_1/o_1} s' \wedge s \xrightarrow{b_2/o_2} s''\right) \implies \begin{pmatrix} (o_1 = o_2 \wedge s' \approx s'') \\ \vee\ (b_1 \wedge b_2 = \mathsf{ff}) \end{pmatrix}$$ □

The notion of determinism is relative to the inputs: an automaton is deterministic if, in any state, for any input configuration, the output and the (equivalence class of the) next state are uniquely determined. The set of deterministic input/output Boolean automata is denoted by IOB_d.

Definition 6 (reactivity). An IOB (S, s_0, I, O, T) is reactive iff

$$\forall s \in S, \left(\bigvee_{s \xrightarrow{b/}} b \right) = \mathsf{tt}$$ □

The notion of reactivity is also relative to the inputs: an automaton is reactive if it cannot refuse inputs. It may have an *idle* reaction, which does not change states nor output signals, but the reaction has to be defined.

The set of reactive input/output Boolean automata is denoted by IOB_r. The set of deterministic and reactive input/output Boolean automata is denoted by IOB_{dr}.

Property 7 (bisimulation preserves reactivity and determinism). $\forall\ A_1, A_2 \in \mathsf{IOB}$,

$$\begin{array}{lll} A_1 \approx A_2 & \implies & (A_1 \in \mathsf{IOB}_r \iff A_2 \in \mathsf{IOB}_r) \\ A_1 \approx A_2 & \implies & (A_1 \in \mathsf{IOB}_d \iff A_2 \in \mathsf{IOB}_d) \end{array}$$ □

2.2 Operations on Input/Output Boolean Automata

Definition 8 (synchronous product of IOBs). Let $A_i = (S_i, s_{0i}, I_i, O_i, T_i)$, for $i \in \{1,2\}$ be two IOBs. Then their synchronous product is the IOB $\mathcal{F}_{||}(A_1, A_2) = (S_1 \times S_2, (s_{01}, s_{02}), I_1 \cup I_2, O_1 \cup O_2, T')$, where T' is defined by:

$$\left.\begin{array}{r}(s_1, b_1, o_1, s_1') \in T_1 \wedge \\ (s_2, b_2, o_2, s_2') \in T_2 \wedge \\ (b_1 \wedge b_2 \neq \texttt{ff})\end{array}\right\} \iff ((s_1, s_2), b_1 \wedge b_2, o_1 \cup o_2, (s_1', s_2')) \in T'$$

□

The synchronous product is intended to represent the free parallel execution of two systems, with no synchronization.

Definition 9 (encapsulation of IOBs). Let $A = (S, s_0, I, O, T)$ be an IOB and $Y \subseteq (I \cup O)$. Let us note $\mathcal{C}_Y(b, o)$ the formula[3] $\exists Y.(b \wedge \mathrm{Bool}_Y(o \cap Y))$. The encapsulation of A according to Y is the IOB $\mathcal{F}_Y(A) = (S, s_0, I - Y, O - Y, T')$ where T' is defined by:

$$\Big((s, b, o, s') \in T \ \wedge\ \mathcal{C}_Y(b, o) \neq \texttt{ff} \Big) \quad\iff\quad (s, \mathcal{C}_Y(b, o), o - Y, s') \in T'$$

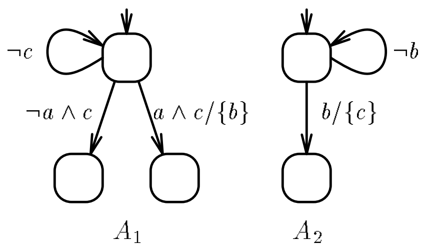
□

Intuitively, since $b = \bigvee m_i$ (Prop. 2.1), a transition with label b/o can be viewed as a set of transitions with labels m_i/o. For each of them, we require that: $\forall y \in Y$, either $y \in o$ and $m_i \implies y$ or, $y \notin o$ and $m_i \implies \neg y$. $\mathcal{C}_Y(b, o)$ is the formula obtained by hiding elements of Y in $\bigvee m_j$ for the m_j which satisfy the above condition.

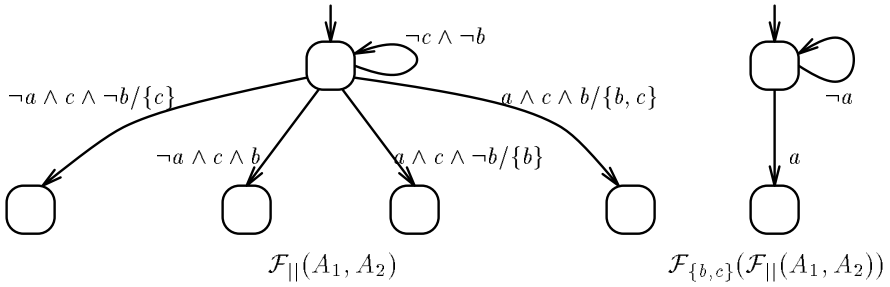
In some sense, before encapsulation, inputs and outputs can be considered as disjoint sets, even if they share signal names. Encapsulating a system with parameter $Y \subseteq I \cup O$ is a way of connecting the inputs whose names belong to Y to the outputs having the same names. It closes the system with respect to those names, having two effects: 1) for a transition, whose input guard m implies y, to be taken in the resulting system, the signal $y \in Y$ has to be output by the system itself, during the very reaction; conversely, a transition whose input guard implies $\neg y$ can be taken only if y is *not* output during the very reaction; 2) the signal y is no longer visible.

Example 2.

Let A_1 and A_2 be the two IOBs given besides. We show below the synchronous product $\mathcal{F}_{||}(A_1, A_2)$ and the encapsulation $\mathcal{F}_{\{b,c\}}(\mathcal{F}_{||}(A_1, A_2))$ of this product:



[3] Remember our notations: $\exists Y.(b)$ stands for $\exists y_1 \ldots \exists y_n.(b)$, and $\mathrm{Bool}_Y(o \cap Y)$ is the characteristic formula of $o \cap Y$ as a subset of Y.



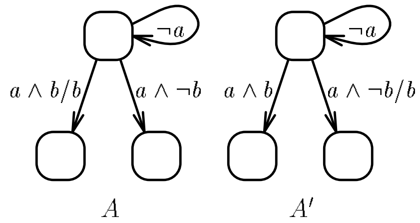
□

Property 10 (product preserves reactivity and determinism).

$$A_1 \in \mathsf{IOB}_{dr} \ \wedge \ A_2 \in \mathsf{IOB}_{dr} \quad \Longrightarrow \quad \mathcal{F}_{||}(A_1, A_2) \in \mathsf{IOB}_{dr}$$

□

This is not true for encapsulation. Two classical counter-examples are shown besides: Both A and A' are reactive and deterministic, but $F_{\{b\}}(A)$ is not deterministic and $F_{\{b\}}(A')$ is not reactive.



Property 11 ($\mathcal{F}_{||}$ and $\mathcal{F}_Y$ preserve the bisimulation). $\forall A_1, A_2, A \in \mathsf{IOB}$

$$A_1 \approx A_2 \quad \Longrightarrow \quad \mathcal{F}_{||}(A_1, A) \approx \mathcal{F}_{||}(A_2, A) \ \wedge \ \mathcal{F}_Y(A_1) \approx \mathcal{F}_Y(A_2)$$

□

3 A Deterministic Synchronous Language

We define a simple synchronous language as a set of constructs on input/output Boolean automata. The semantics and the compositionality property are borrowed from [Mar92].
The set $\mathcal{P}_d$ of programs is defined inductively by:

$$P \quad ::= \quad \mathrm{IOB}_{dr} \quad | \quad P||P \quad | \quad \overline{P^Y}.$$

In a deterministic language, the basic components are deterministic and reactive, and the composed processes should be deterministic and reactive too. Since the encapsulation does not preserve determinism nor reactivity, the semantics includes a notion of *incorrect* program.

The semantics is given by the function $\mathcal{S}_d : \mathcal{P}_d \mapsto \mathrm{IOB}_{dr} \cup \{\perp\}$ which maps *correct* programs to reactive and deterministic input/output Boolean automata and *incorrect* programs to the error value $\perp$.

$$\begin{aligned}
\mathcal{S}_d(A) &= A \text{ for } A \in \mathrm{IOB}_{dr} \\
\mathcal{S}_d(P_1 \| P_2) &= \begin{cases} \perp \text{ if } \mathcal{S}_d(P_1) = \perp \vee \mathcal{S}_d(P_2) = \perp \\ \mathcal{F}_{\|}(\mathcal{S}_d(P_1), \mathcal{S}_d(P_2)) \text{ otherwise.} \end{cases} \\
\mathcal{S}_d(\overline{P^Y}) &= \begin{cases} \perp \text{ if } \mathcal{S}_d(P) = \perp \text{ or } \mathcal{F}_Y(\mathcal{S}_d(P)) \notin \mathrm{IOB}_{dr} \\ \mathcal{F}_Y(\mathcal{S}_d(P)) \text{ otherwise.} \end{cases}
\end{aligned}$$

This definition does not give an efficient procedure for the detection of incorrect programs. In practice, depending on the language, the set of programs considered as incorrect may be larger than that [BG92, STB95, HM95]. However, it always contains the programs whose behavior would be non reactive or non-deterministic.

Property 12 (compositionality). We define an equivalence relation on programs by:

$$P_1 \equiv P_2 \iff \begin{cases} (\mathcal{S}_d(P_1) = \mathcal{S}_d(P_2) = \perp) \quad \vee \\ (\mathcal{S}_d(P_1) \neq \perp \wedge \mathcal{S}_d(P_2) \neq \perp \wedge \mathcal{S}_d(P_1) \approx \mathcal{S}_d(P_2)) \end{cases}$$

Then, $\equiv$ is a congruence for the language constructs. □

This means, in particular, that equivalent programs behave the same with respect to error detection.

4 A Non-Deterministic Synchronous Language

4.1 Syntax and Intuitive Semantics

In a non-deterministic synchronous language, the basic components may be non-deterministic, but they are still required to be reactive.
The set $\mathcal{P}_{nd}$ of programs is defined inductively by:

$$P \quad ::= \quad \mathrm{IOB}_r \quad | \quad P \| P \quad | \quad \overline{P^Y}.$$

The intuitive semantics is the following: A non-deterministic program represents a set of deterministic ones. We want to consider a non-deterministic program as correct if and only if all the deterministic programs it represents are correct.

To formalize this semantics, we will use automata controlled by additional inputs, called *oracles*. For simplicity and without lack of generality, in the remainder of the paper, we will always consider that automata are in monomial form (see property 2.1).

4.2 Controlled Input/Output Boolean Automata

We first define a special kind of IOBs, where all the transition guards are the disjunction of two conditions: A complete monomial on usual signals, and a condition on auxiliary *oracles* signals, which cannot be emitted. Intuitively, oracle conditions will be added to determinize a non-deterministic IOB.

Definition 13 (controlled input/output Boolean automata).
A controlled input/output Boolean automaton (CIOB for short) is a tuple $(S, s_0, I, O, \Omega, T)$ where

- Ω is a set of variables called *oracles* ($\Omega \cap (I \cup O) = \emptyset$).
- $T \subseteq S \times \mathcal{M}(I) \times 2^O \times \mathcal{B}(\Omega) \times S$. A transition (s, m, o, ω, s') of a CIOB will be denoted by: $s \xrightarrow[\omega]{m/o} s'$. □

For any state s, we note T^s the set of transitions starting from s, and T^s_m the set of these transitions that are guarded by the monomial m. For any transition t we note m_t, o_t, ω_t, and s_t its guard, output set, oracle condition and target state. We note CIOB the set of controlled IOBs.

A CIOB can be considered as a special case of IOB, by considering any transition $s \xrightarrow[\omega]{m/o} s'$ as $s \xrightarrow{m \wedge \omega/o} s'$. As such CIOBs inherit the notion of determinism, and reactivity. As before, CIOB_d, CIOB_r, and CIOB_{dr} respectively denote the sets of deterministic, reactive, deterministic and reactive, controlled IOBs.

The operators $\mathcal{F}_{||}$ and $\mathcal{F}_Y$ are also available on CIOBs, with the following restrictions: Let $B_i = (S_i, s_{0i}, I_i, O_i, \Omega_i, T_i)$, $i = 1, 2$ be two CIOBs. Then

- $\mathcal{F}_{||}(B_1, B_2)$ is only defined if $\Omega_i \cap (I_j \cup O_j \cup \Omega_j) = \emptyset$, $i = 1, 2$, $j = 3 - i$
- $\mathcal{F}_Y(B_1)$ is only defined if $Y \cap \Omega_1 = \emptyset$

In other words, we forbid synchronization and encapsulation on oracle variables.

Definition 14 (removing oracles). We introduce the function η_Ω from CIOB to IOB: If $B = (S, s_0, I, O, \Omega, T')$ is a CIOB, $\eta_\Omega(B)$ is the IOB (S, s_0, I, O, T), where $T = \{(s, m, o, s') \mid \exists \omega, (s, m, o, \omega, s') \in T'\}$. □

Notations: Let $B_i = (S_i, s_{0i}, I, O, \Omega_i, T_i)$, $(i = 1, 2)$ be two CIOBs. Let $\mathcal{R} \subseteq (S_1 \cup S_2)^2$ be an equivalence relation on their states. Let $t_1 = s_1 \xrightarrow[\omega_1]{m/o} s'_1$ be a transition of B_1. s_2 be a state either of B_1 or of B_2. We note $\mathcal{T}_{s_2}(t_1)$ the set of transitions $\{s_2 \xrightarrow[\omega_2]{m/o} s'_2 \mid s'_1 \mathcal{R} s'_2\}$. Notice that $\mathcal{T}_{s_1}(t_1)$ denotes the bundle of transitions that are "equivalent" to t_1.

Definition 15 (CIOB control-equivalence). Two CIOBs $B_i = (S_i, s_{0i}, I, O, \Omega_i, T_i)$, $(i = 1, 2)$ are control-equivalent (noted $B_1 \sim B_2$), if and only if there exists an equivalence relation $\mathcal{R} \subseteq (S_1 \cup S_2)^2$ such that

1. $(s_{01}, s_{02}) \in \mathcal{R}$

2. $(s_1, s_2) \in \mathcal{R} \Rightarrow \begin{cases} \forall\ s_1 \xrightarrow[\omega_1]{m/o} s_1' \in T_1,\ \exists\ s_2 \xrightarrow[\omega_2]{m/o} s_2' \in T_2, \text{ with } (s_1', s_2') \in \mathcal{R} \\ \text{and conversely} \end{cases}$
3. $\forall s_1 \in S_1, s_2 \in S_2$ such that $s_1 \mathcal{R} s_2$, for any set $\{t_1, t_2, \ldots, t_n\}$ of transitions starting from s_1,

$$\left(\bigwedge_{i=1}^{n} \bigvee_{t \in \mathcal{T}_{s_1}(t_i)} \omega_t \right) = \mathsf{ff} \iff \left(\bigwedge_{i=1}^{n} \bigvee_{t \in \mathcal{T}_{s_2}(t_i)} \omega_t \right) = \mathsf{ff}$$

and conversely □

In this definition, points (1) and (2) express that $\eta_{\Omega_1}(B_1) \approx \eta_{\Omega_2}(B_2)$. Point (3) defines a notion of equivalence about oracle introduction.

Property 16 (preservation of control equivalence).
Let $B_i = (S_i, s_{0i}, I, O, \Omega_i, T_i), (i = 1, 2)$ be two CIOBs, with $B_1 \sim B_2$. Then,

- for any CIOB B, $\mathcal{F}_{||}(B_1, B) \sim \mathcal{F}_{||}(B_2, B)$
- for any $Y \subseteq (I \cup O)$, $\mathcal{F}_Y(B_1) \sim \mathcal{F}_Y(B_2)$ □

4.3 From non-deterministic programs to controlled programs

Definition 17 (controlled programs). The set $\mathcal{P}_c$ of controlled programs is defined by: $P \ ::= \ \mathrm{CIOB}_{dr} \ \mid \ P||P \ \mid \ \overline{P^Y}$. □

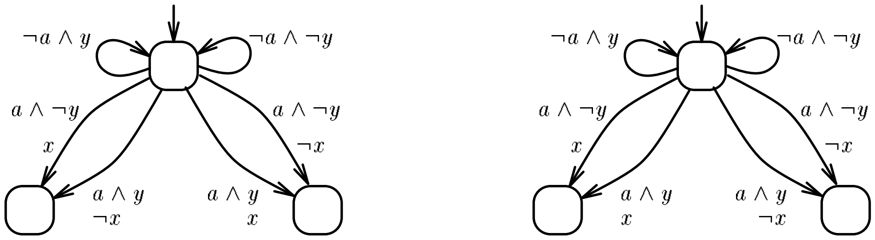
The semantics of controlled programs is:

$$\begin{aligned} \mathcal{S}_c(B) &= B \text{ for } B \in \mathrm{CIOB}_{dr} \\ \mathcal{S}_c(P_1||P_2) &= \begin{cases} \perp \text{ if } \mathcal{S}_c(P_1) = \perp \vee \mathcal{S}_c(P_2) = \perp \\ \mathcal{F}_{||}(\mathcal{S}_c(P_1), \mathcal{S}_c(P_2)) \text{ otherwise.} \end{cases} \\ \mathcal{S}_c(\overline{P^Y}) &= \begin{cases} \perp \text{ if } \mathcal{S}_c(P) = \perp \text{ or } \mathcal{F}_Y(\mathcal{S}_c(P)) \notin \mathrm{CIOB}_{dr} \\ \mathcal{F}_Y(\mathcal{S}_c(P)) \text{ otherwise.} \end{cases} \end{aligned}$$

In order to define the semantics of non-deterministic programs, we want (1) to transform its basic components into *deterministic* (and reactive) automata, by strengthening their transition guards with oracle conditions, (2) to check that determinism and reactivity are preserved when composing these components — thus detecting causality errors — and (3) to hide the oracles in the result, thus possibly introducing non-determinism again.

Of course, this semantics must be independent of the way the program is determinized during step (1) above. Surprisingly, this involves very strong constraints on the way one is allowed to introduce oracles:

- Of course, oracles must determinize the automaton, but their introduction must also preserve its reactivity: This implies that for any state s and any monomial m, we must have $\bigvee_{t \in T_m^s} \omega_t = \mathsf{tt}$.
- But *oracles must not do anything else*. Their introduction must not separate transitions that do not correspond to non-deterministic choices. For instance, consider the following two different (careless) determinizations of the same automaton, using a single oracle x:



When encapsulated by $\mathcal{F}_{\{y\}}$ the first one detects an error (non-determinism on both $a \wedge x$ and $a \wedge \neg x$) while the second does not.

This motivates the *separation condition* in the following definition, which states how one can safely use oracles to determinize an automaton:

Definition 18 (determinizing functions). Let $A = (S, s_0, I, O, T) \in \mathrm{IOB}_r$, and Ω be a set of oracles. The set $\mathrm{Det}_\Omega(A)$ of determinization functions on Ω is the set of functions δ_Ω such that

- $\delta_\Omega(A) = (S, s_0, I, O, \Omega, T') \in \mathrm{CIOB}_{dr}$
- $\eta_\Omega(\delta_\Omega(A)) = A$
- (Separation condition) For any state s and any set of transitions $\{t_1, \ldots, t_n\} \subseteq T^s$ such that $(i \neq j) \Rightarrow (m_{t_i} \neq m_{t_j})$ we have $\bigwedge_{i=1}^{n} \omega_{t_i} \neq \mathsf{ff}$. □

The separation condition expresses that oracle conditions never separate sets of transitions which do not correspond to non-deterministic choices.

It is easy to see that it is always possible to find a set Ω large enough such that $\mathrm{Det}_\Omega(A)$ is not empty. We give here a particular procedure to build such a function: Let us associate with each $m \in \mathcal{M}(I)$ a set of oracles Ω_m, all these sets being pairwise disjoint. Let s be a state of A, $m \in \mathcal{M}(I)$. Then we can associate with each $t \in T^s_m$ a condition $\omega_t \in \mathcal{B}(\Omega_m)$ in such a way that

$$\bigvee_{t \in T^s_m} \omega_t = \mathsf{tt} \text{ and } (t \neq t') \Rightarrow (\omega_t \wedge \omega_{t'} = \mathsf{ff})$$

The fact that disjoint sets of oracles are involved in conditions associated with different monomials, ensures the separation condition.

The following property and its corollary show that the choice of a particular determinizing function has no importance:

Property 19 (from bisimilar IOBs to control-equivalent CIOBs). Let A_1, A_2 be two bisimilar IOBs, δ_i be a determinizing functions of A_i, $(i = 1, 2)$. Then $\delta_1(A_1) \sim \delta_2(A_2)$. □

Corollary 20. *Let A be an IOB, δ_1, δ_2 be two determinizing functions of A. Then $\delta_1(A) \sim \delta_2(A)$.* □

Now, we use the determinizing functions to associate controlled programs to programs, by means of the function $\mathcal{D}_\Omega$:

Definition 21 (oracle introduction). $\mathcal{D}_\Omega : \mathcal{P}_{nd} \mapsto \mathcal{P}_c$

$$\begin{aligned}
\mathcal{D}_\Omega(A) &= \delta_\Omega(A) \text{ with } \delta_\Omega \in \mathrm{Det}_\Omega(A), \text{ for } A \in \mathrm{IOB}_r \\
\mathcal{D}_\Omega(P_1 \| P_2) &= \mathcal{D}_{\Omega_1}(P_1) \| \mathcal{D}_{\Omega_2}(P_2) \quad (\text{with } \Omega_1 \cap \Omega_2 = \emptyset \text{ and } \Omega = \Omega_1 + \Omega_2) \\
\mathcal{D}_\Omega(\overline{P^Y}) &= \overline{\mathcal{D}_\Omega(P)}^Y
\end{aligned}$$ □

The definition of $\mathcal{D}_\Omega$ depends on the particular choice of a determinizing function δ_Ω for basic automata. From the corollary 13, these different choices provide results that are control-equivalent.

The following property is the basis of our semantics of non-deterministic programs:

Property 22 (control-equivalence preserves error detection). Let $B_i = (S_i, s_{0i}, I, O, \Omega_i, T_i), (i = 1, 2)$ be two deterministic and reactive CIOBs, with $B_1 \sim B_2$. Then,

- for any CIOB B, $\mathcal{F}_{||}(B_1, B) \in \mathsf{CIOB}_{dr} \iff \mathcal{F}_{||}(B_2, B) \in \mathsf{CIOB}_{dr}$
- for any $Y \subseteq (I \cup O)$, $\mathcal{F}_Y(B_1) \in \mathsf{CIOB}_{dr} \iff \mathcal{F}_Y(B_2) \in \mathsf{CIOB}_{dr}$ □

4.4 Compositional semantics

Definition 23 (semantics of non-deterministic programs).
The semantics of non-deterministic programs is given by the function $\mathcal{S}_{nd} : \mathcal{P}_{nd} \mapsto \mathsf{IOB}_r \cup \{\perp\}$ which maps correct programs to reactive IOBs and incorrect ones to $\perp$:

$$\mathcal{S}_{nd}(P) = \textbf{let } Z = \mathcal{S}_c(\mathcal{D}_\Omega(P)) \textbf{ in } (\text{if } Z \in \mathrm{CIOB}_{dr} \text{ then } \eta_\Omega(Z) \text{ else } \perp).$$ □

We can also define a direct mapping $\mathcal{F}$ from $\mathcal{P}_{nd}$ to IOB, which is not aware of causality errors:

$$\begin{aligned}
\mathcal{F}(A) &= A \;\text{ for } A \in \mathrm{IOB} \\
\mathcal{F}(P_1 \| P_2) &= \mathcal{F}_{||}(\mathcal{F}(P_1), \mathcal{F}(P_2)) \\
\mathcal{F}(\overline{P^Y}) &= \mathcal{F}_Y(\mathcal{F}(P))
\end{aligned}$$

Now, we have defined the semantics of non-deterministic programs via their determinization in controlled programs, the detection of causality errors during the composition of *CIOB*s, and, in absence of such causality error, the removal of oracles in the result. We have also the direct translation $\mathcal{F}$ (see diagram besides). Below, we show that, when no error is detected, this diagram commutes (Prop. 4.4) and that the semantics is compositional with respect to bisimulation (Prop. 4.4).

$$\begin{tikzcd}
\mathcal{P}_{nd}\ (\approx) \arrow[r, "\mathcal{D}_\Omega"] \arrow[d, "\mathcal{F}"'] & \mathcal{P}_c\ (\sim) \arrow[d, "\mathcal{S}_c"] \\
\mathsf{IOB} & \mathsf{CIOB} \cup \{\perp\} \arrow[l, "\eta_\Omega"]
\end{tikzcd}$$

Lemma 24 ($\mathcal{F}_{||}$ and $\mathcal{F}_Y$ commute with η_Ω). $\forall A_1, A_2 \in \mathsf{IOB}_r, \forall Y,$

$$\eta_\Omega(\mathcal{F}_{||}(A_1, A_2)) = \mathcal{F}_{||}(\eta_\Omega(A_1), \eta_\Omega(A_2)) \ \textit{and}\ \eta_\Omega(\mathcal{F}_Y(A_1)) = \mathcal{F}_Y(\eta_\Omega(A_1)) \quad \square$$

Property 25 (direct and indirect definitions of the semantics). For programs $P \in \mathcal{P}_{nd}$, if $\mathcal{S}_{nd}(P) \neq \perp$ then $\mathcal{S}_{nd}(P) = \mathcal{F}(P)$. $\square$

Property 26 (compositionality). We define an equivalence relation on programs by:

$$\begin{aligned} P_1 \equiv P_2 \iff & (\mathcal{S}_{nd}(P_1) = \mathcal{S}_{nd}(P_2) = \perp) \vee \\ & (\mathcal{S}_{nd}(P_1) \neq \perp \wedge \mathcal{S}_{nd}(P_2) \neq \perp \wedge \mathcal{S}_{nd}(P_1) \approx \mathcal{S}_{nd}(P_2)) \end{aligned}$$

Then, $\equiv$ is a congruence for the language constructs. $\square$

5 Conclusion

We showed how to introduce non-determinism in a simple imperative synchronous language, where causality problems arise. They lead to a notion of incorrect programs, which have to be detected.

For deterministic languages, a class of incorrect programs can be characterized by the fact that the composition of deterministic components is non-deterministic. This gives a detection procedure. For a non-deterministic language, the causality problems do not disappear, and we should, at least, define precisely what an incorrect program is. When a composed program happens to be non-deterministic, there are two possible reasons: either this comes from the *intrinsic* non-determinism of the basic components, or it has been introduced by the composition. Defining and detecting incorrect programs requires that we be able to distinguish between these two cases. The method we propose consists in "marking" the non-deterministic situations of the basic components with formulas built on additional variables called oracles.

It appears that the oracle introduction phase has to be defined very carefully. Requiring that the bisimulation of non-deterministic behaviours be a congruence for the operators we consider (parallel composition and the encapsulation which captures exactly the semantics of the synchronous broadcast) gave us sufficient conditions on the way oracles should be introduced. They appear to be stronger than what one could expect from practise and informal considerations.

We showed how to introduce oracles by defining the structure of *controlled* automata. This makes the definitions and the proofs easier, because the oracle condition of the transition labels does not interfere at all with the combination of inputs and outputs. However, it is possible to consider that the oracle condition is integrated into the input condition of the transitions, provided that the oracles and the ordinary inputs and outputs constitute disjoint sets of variables. This gives a practical procedure for dealing with non-determinism in the compiler of an existing deterministic language: one just has to implement the determinizing functions for the basic components, introducing fresh variables. Then the usual composition algorithm can be used on these deterministic objects. Finally the oracles have to be removed.

Finally, notice that determinization with oracles is not only useful for causality analysis. Determinism with respect to inputs is also a key property for symbolic verification: This explains the remarkable success of BDD-based techniques in the domain of circuits — which are input-deterministic machines — and their relative failure in dealing with non-deterministic systems. So, determinization with oracles is surely a useful first step in the verification of non-deterministic systems.

References

[BG92] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[CGS91] C. Courcoubetis, S. Graf, and J. Sifakis. An algebra for boolean processes. In *Workshop on Computer-Aided Verification 91, Aalborg*. LNCS Vol. 575, June 1991.

[Hal93] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer Academic Pub., 1993.

[Har84] D. Harel. Statecharts: A visual approach to complex systems. In *Advanced NATO Institute on Logics and Models for Verification and Specification of Concurrent Systems*, La Colle-sur-Loup, 1984.

[HM95] N. Halbwachs and F. Maraninchi. On the symbolic analysis of combinational loops in circuits and synchronous programs. In *Euromicro'95*, Como (Italy), September 1995.

[IEE91] Another look at real-time programming. *Special Section of the Proceedings of the IEEE*, 79(9), September 1991.

[Kur89] R. P. Kurshan. Analysis of discrete event coordination. In J.W. de Bakker, W.-P. de Roever, and G. Rozemberg, editors, *REX Workshop on Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness*. LNCS 430, Springer Verlag, May 1989.

[LT89] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. CWI-Quaterly 3, CWI, Amsterdam, 1989.

[Mar92] F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. In *CONCUR'92*, Stony Brook, August 1992. LNCS 630, Springer Verlag.

[STB95] T. R. Shiple, H. Touati, and G. Berry. Causality analysis of circuits. In preparation 1995.

Appendix

Proof of Property 4.2

$[\mathcal{F}_{||}]$: Since $B_1 \sim B_2$, there exists and equivalence $\mathcal{R}$ on $S_1 \cup S_2$ satisfying the conditions of the definition 10. Let us extend this relation to $(S_1 \times S) \cup (S_2 \times S)$ by stating $(s_1, s)\mathcal{R}(s_2, s')$ iff $s_1 \mathcal{R} s_2$ and $s = s'$. This new relation obviously satisfies the conditions (1) and (2) of the definition 10.

Let $\mathcal{F}_{||}(B_i, B) = (S_i \times S, (s_{0i}, s_0), I \cup I', O \cup O', \Omega_i \cup \Omega, T^{(i)})$, $(i = 1, 2)$. Now, consider $s_1 \in S_1$, $s_2 \in S_2$, $s \in S$ with $(s_1, s)\mathcal{R}(s_2, s)$. Let $(T_1, \ldots, T_n)$ be a set of transitions starting from (s_1, s). Each T_k results from the combination of a transition $t_k^{(1)}$, starting from s_1 in B_1, and a transition t, starting from s in B. From the definition of $\mathcal{R}$, $\mathcal{T}_{(s_1,s)}(T_k) = \mathcal{T}_{s_1}(t_k^{(1)}) \times \{t_k\}$.

We have:

$$\begin{aligned}
\bigwedge_{k=1..n} \bigvee_{T \in \mathcal{T}_{(s_1,s)}(T_k)} \omega_T &= \bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_1}(t_k^{(1)})} \left(\omega_{t_k^{(1)}} \wedge \omega_{t_k}\right) \\
&= \bigwedge_{k=1..n} \left(\omega_{t_k} \wedge \bigvee_{t \in \mathcal{T}_{s_1}(t_k^{(1)})} \omega_{t_k^{(1)}}\right) \\
&= \bigwedge_{k=1..n} \omega_{t_k} \wedge \left(\bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_1}(t_k^{(1)})} \omega_{t_k^{(1)}}\right)
\end{aligned}$$

Now, since $\Omega \cap \Omega_1 = \emptyset$, this last expression is ff if and only if its terms are both ff. Now, from $B_1 \sim B_2$, we have

$$\bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_1}(t_k^{(1)})} \omega_{t_k^{(1)}} = \mathsf{ff} \text{ iff } \bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_2}(t_k^{(1)})} \omega_{t_k^{(1)}} = \mathsf{ff}$$

So, finally

$$\left(\bigwedge_{k=1..n} \bigvee_{T \in \mathcal{T}_{(s_1,s)}(T_k)} \omega_T\right) = \mathsf{ff} \text{ iff } \left(\bigwedge_{k=1..n} \bigvee_{T \in \mathcal{T}_{(s_2,s)}(T_k)} \omega_T\right) = \mathsf{ff}$$

$[\mathcal{F}_Y]$: Let $\mathcal{R}$ be the equivalence on $S_1 \cup S_2$ satisfying the conditions of the definition 10. From property 2.2, it still satisfies the conditions (1) and (2) on $\mathcal{F}_Y(B_1)$ and $\mathcal{F}_Y(B_2)$. Concerning condition (3), let us observe that the fact that a transition t, starting from s_1 in B_1, is dropped out by the operator $\mathcal{F}_Y$ only depends on its guard m_t and its output o_t. As a consequence, it t is dropped out, so are all the transitions in $\mathcal{T}_{s_1}(t)$ and in $\mathcal{T}_{s_2}(t)$, for any state s_2 of B_2 with $s_1 \mathcal{R} s_2$.

Let s_1 be a state of $\mathcal{F}_Y(B_1)$, s_2 be a state of B_2 with $s_1 \mathcal{R} s_2$, and $(t_1, \ldots, t_n)$ be a set of transitions starting from s_1. Each t_k comes from a transition t'_k in B_1, with the same oracle condition. So,

$$\bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_1}(t_k)} \omega_t = \bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_1}(t'_k)} \omega_t \quad \text{in } B_1$$

Now, since $B_1 \sim B_2$

$$\bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_1}(t'_k)} \omega_t = \mathsf{ff} \iff \bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_2}(t'_k)} \omega_t = \mathsf{ff} \quad \text{in } B_2$$

and $\displaystyle \bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_2}(t'_k)} \omega_t = \bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_2}(t_k)} \omega_t \quad \text{in } \mathcal{F}_Y(B_2)$

Proof of Property 4.3

Let $B_i = \delta_i(A_i)$ $(i = 1, 2)$. Each B_i has the same set of states, say S_i, as A_i. Since $A_1 \approx A_2$, there exists an equivalence relation $\mathcal{R} \subseteq S_1 \times S_2$ as in definition 2. Obviously $\mathcal{R}$ satisfies points (1) and (2) of definition 10. Now, consider $(s_1, s_2) \in \mathcal{R}$, and let $\{t_1, \ldots, t_n\}$ be a set of transitions $s_1 \xrightarrow{m_k/o_k} s'_k$ in A_1. Then,

- either $\forall k, \ell = 1 \ldots n$ with $k \neq \ell$ we have $m_k = m_\ell \Rightarrow t_k \in \mathcal{T}_{s_1}(t_\ell)$. In that case, we can select a maximal subset $\{t_{k_1}, \ldots, t_{k_p}\}$ of $\{t_1, \ldots, t_n\}$ such that all the m_{k_j} are different, and we have

$$\left(\bigwedge_{k=1}^{n} \bigvee_{t \in \mathcal{T}_{s_1}(t_k)} \omega_t^i \right) = \left(\bigwedge_{j=1}^{p} \bigvee_{t \in \mathcal{T}_{s_1}(t_{k_j})} \omega_t^i \right) \quad (i = 1, 2)$$

  Now, from the separation condition, for each set $\{(t'_{k_1}, \ldots, t'_{k_p}) | t'_{k_j} \in \mathcal{T}_{s_i}(t_{k_j})\}$,

$$\bigwedge_{j=1..p} \omega^i_{t'_{k_j}} \neq \mathsf{ff} \quad (i = 1, 2)$$

  so $\displaystyle \bigwedge_{j=1..p} \bigvee_{t \in \mathcal{T}_{s_i}(t_{k_j})} \omega_t^i \neq \mathsf{ff} \quad (i = 1, 2)$

- or $\exists k, \ell$ with $m_k = m_\ell$ and $t_k \notin \mathcal{T}_{s_1}(t_\ell)$. In that case, since both B_i are deterministic,

$$\forall t'_k \in \mathcal{T}_{s_i}(t_k),\ \forall t'_\ell \in \mathcal{T}_{s_i}(t_\ell),\ \omega^i_{t'_k} \wedge \omega^i_{t'_\ell} = \mathsf{ff} \quad (i = 1, 2)$$

  so $\displaystyle \left(\bigvee_{t' \in \mathcal{T}_{s_i}(t_k)} \omega^i_{t'} \right) \wedge \left(\bigvee_{t' \in \mathcal{T}_{s_i}(t_\ell)} \omega^i_{t'} \right) = \mathsf{ff} \quad (i = 1, 2)$

  and $\displaystyle \left(\bigwedge_{k=1..n} \bigvee_{t \in \mathcal{T}_{s_i}(t_k)} \omega_t^i \right) = \mathsf{ff} \quad (i = 1, 2)$

Proof of Property 4.3

There is nothing to prove about parallel composition, which preserves determinism and reactivity. So, we have only to prove that, for any pair of control equivalent, reactive and deterministic CIOBs $B_1 = (S_1, s_{01}, I, O, \Omega_1, T_1)$ and $B_2 = (S_2, s_{02}, I, O, \Omega_2, T_2)$, for any $Y \subseteq I \cup O$, $\mathcal{F}_Y(B_1) \in CIOB_{dr} \iff \mathcal{F}_Y(B_2) \in CIOB_{dr}$.

Determinism: Assume $\mathcal{F}_Y(B_1)$ is not deterministic. Then it has two transitions $s_1 \xrightarrow[\omega_1]{m/o} r_1$ and $s_1 \xrightarrow[\omega'_1]{m/o'} r'_1$ with $\omega_1 \wedge \omega'_1 \neq \mathsf{ff}$ and either $o \neq o'$ or $r \not\approx r'$. These transitions result from the presence in B_1 of two transitions $s_1 \xrightarrow[\omega_1]{m \wedge m'/}$ and $s_1 \xrightarrow[\omega'_1]{m \wedge m''/}$

where $m', m'' \in \mathcal{M}(Y)$, and $m' \neq m''$ since B_1 is deterministic and $\omega_1 \wedge \omega_1' \neq \mathsf{ff}$. From the control equivalence of B_1 and B_2, there is a state s_2 in B_2 with two corresponding transitions $s_2 \xrightarrow[\omega_2]{m \wedge m'/}$ and $s_2 \xrightarrow[\omega_2']{m \wedge m''/}$, with $\omega_2 \wedge \omega_2' \neq \mathsf{ff}$. These transitions result in a non-deterministic choice in $\mathcal{F}_Y(B_2)$.

Reactivity: Assume $\mathcal{F}_Y(B_1)$ is not reactive; so, there exists a state s_1 and a monomial m such that $\left(\bigvee_{t \in T_m^{s_1}} \omega_t\right) \neq \mathsf{tt}$. Then we show that, in B_1, for each $m' \in \mathcal{M}(Y)$, we can choose a transition $t_{m'} \in T^{s_1}_{m \wedge m'}$ in such a way that

1. $\mathcal{C}_Y(m \wedge m', o_{t_{m'}}) = \mathsf{ff}$ (so, $t_{m'}$ is dropped out by $\mathcal{F}_Y$, with its whole class $\mathcal{T}_{s_1}(t_{m'})$)
2. $\left(\bigwedge_{m'} \bigvee_{t \in \mathcal{T}_{s_1}(t_{m'})} \omega_t\right) \neq \mathsf{ff}$
3. $\left(\bigwedge_{m'} \bigvee_{t \in \mathcal{T}_{s_1}(t_{m'})} \omega_t\right) \wedge \left(\bigvee_{t \notin \bigcup_{m'} \mathcal{T}_{s_1}(t_{m'})} \omega_t\right) = \mathsf{ff}$

Point (1) comes from the fact that, since B_1 is reactive, $\bigvee_{t \in T^s_{m \wedge m'}} \omega_t = \mathsf{tt}$, so, if $\mathcal{F}_Y$ is not reactive, some transitions of $T^s_{m \wedge m'}$ must have been dropped out. Moreover, these "holes" in the projection of the $T^s_{m \wedge m'}$ do not mask each other (point(2)). And point (3) comes from the determinism of B_1: Each $\omega_{t_{m'}}$ is disjoint with any other ω_t, for $t \notin \mathcal{T}_{s_1}(t_{m'})$, so the conjunction of all the $\omega_{t_{m'}}$ is disjoint with the disjunction of other ω_t.

Now, since B_2 is control equivalent with B_1, there is a state s_2 in B_2 whose outgoing transitions satisfy the same properties. From point (3) in B_2, it follows that $\mathcal{F}_Y(B_2)$ is not reactive.

Proof of Property 4.4

By induction on $\mathcal{P}_{nd}$, using lemma 16 and the following observation: $\mathcal{S}_{nd}(P) \neq \perp \implies \mathcal{S}_{nd}(Q) \neq \perp$ for all Q subcomponent of P.

(1) For $A \in \mathrm{IOB}_r$, $\mathcal{S}_{nd}(A) = \eta_\Omega(\mathcal{S}_c(\mathcal{D}_\Omega(A))) = \eta_\Omega(\mathcal{D}_\Omega(A)) = A = \mathcal{F}(A)$.

(2)

$\mathcal{S}_{nd}(P_1 \| P_2) =$	
$\eta_\Omega(\mathcal{S}_c(\mathcal{D}_\Omega(P_1 \| P_2))) =$	by definition of $\mathcal{S}_{nd}$, for a given $\mathcal{D}_\Omega$
$\eta_\Omega(\mathcal{S}_c(\mathcal{D}_{\Omega_1}(P_1) \| \mathcal{D}_{\Omega_2}(P_2))) =$	by definition of $\mathcal{D}_\Omega$ ($\Omega_1 \cap \Omega_2 = \emptyset$)
$\eta_\Omega(\mathcal{F}_{\|}(\mathcal{S}_c(\mathcal{D}_{\Omega_1}(P_1)), \mathcal{S}_c(\mathcal{D}_{\Omega_2}(P_2)))) =$	by definition of $\mathcal{S}_c$, since $\mathcal{S}_c(\mathcal{D}_{\Omega_i}(P_i)) \neq \perp$
$\mathcal{F}_{\|}(\eta_\Omega(\mathcal{S}_c(\mathcal{D}_{\Omega_1}(P_1))), \eta_\Omega(\mathcal{S}_c(\mathcal{D}_{\Omega_2}(P_2)))) =$	since η_Ω and $\mathcal{F}_{\|}$ commute (lemma 16)
$\mathcal{F}_{\|}(\eta_{\Omega_1}(\mathcal{S}_c(\mathcal{D}_{\Omega_1}(P_1))), \eta_{\Omega_2}(\mathcal{S}_c(\mathcal{D}_{\Omega_2}(P_2)))) =$	since $\Omega_i \subseteq \Omega$
$\mathcal{F}_{\|}(\mathcal{S}_{nd}(P_1), \mathcal{S}_{nd}(P_2)) =$	by definition of $\mathcal{S}_{nd}$
$\mathcal{F}_{\|}(\mathcal{F}(P_1), \mathcal{F}(P_2)) =$	by induction on P_1, P_2
$\mathcal{F}(P_1 \| P_2)$	by definition of $\mathcal{F}$

(3) With the same steps: $\mathcal{S}_{nd}(\overline{P^Y}) = \eta_\Omega(\mathcal{S}_c(\mathcal{D}_\Omega(\overline{P^Y}))) = \eta_\Omega(\mathcal{S}_c(\overline{\mathcal{D}_\Omega(P)}^Y)) = \eta_\Omega(\mathcal{F}_Y(\mathcal{S}_c(\mathcal{D}_\Omega(P)))) = \mathcal{F}_Y(\eta_\Omega(\mathcal{S}_c(\mathcal{D}_\Omega(P)))) = \mathcal{F}_Y(\mathcal{S}_{nd}(P)) = \mathcal{F}_Y(\mathcal{F}(P)) = \mathcal{F}(\overline{P^Y})$.