

On the reaction time of some synchronous systems

Ilias Garnier, Christophe Aussaguès, Vincent David

CEA, LIST, Embedded Real Time Systems Laboratory
Point Courrier 94, Gif-sur-Yvette, F-91191 France

Firstname.Lastname@cea.fr

Guy Vidal-Naquet

SUPELEC Systems Sciences (E3S)
Computer Science Department
91192 Gif-sur-Yvette Cedex, France

Guy.Vidal-Naquet@supelec.fr

This paper presents an investigation of the notion of reaction time in some synchronous systems. A state-based description of such systems is given, and the reaction time of such systems under some classic composition primitives is studied. Reaction time is shown to be non-compositional in general. Possible solutions are proposed, and applications to verification are discussed. This framework is illustrated by some examples issued from studies on real-time embedded systems.

1 Introduction

A primary concern when developing hard real-time embedded systems is to ensure the timeliness of computations. This kind of requirement is often expressed as a reaction time constraint, i.e. an upper bound on the time the system may take to process an input and produce the related output. When systems are composed of multiple communicating agents, this task may be difficult. In this paper, we propose a formalization of reaction time for a certain class of synchronous systems. We show that reaction time is a fine-grained notion of functional dependency, and we show that it is non-compositional. In order to solve this problem, we propose an approximate but compositional method to reason on functional dependency and reaction time.

Related work. The specification and verification of temporal properties traditionally relies on temporal logic [6] or related formalisms [2]. With these formalisms, the system designer gives a specification of some causality or quantitative property which is then verified by model-checking.

These methods are also applicable to the restricted class of synchronous systems [8]. The OASIS [5] system, which motivated this study, belongs to this class. A traditional compositional verification of synchronous systems using Moore machines [10] was given in [4]. Our formal framework to reason on reaction time was heavily inspired by the literature on information flow analysis [3] and on the category-theoretic view of process algebras [1]. It is also similar to *testing* methods [11].

2 Preliminaries

2.1 Case study: proving the reactivity of a simple system

Let S be a black box with two buttons A and B as inputs and the elements of any non-singleton set as outputs. In this example, we will assume that S is deterministic. Our goal is to decide whether pressing A has any observable effect on the system. A naive solution is to verify whether the new observable state is different from the previous one.

$$\begin{array}{l|l} \text{Input} & S \xrightarrow{A} S_A \\ \text{Observable state} & o \quad o' \end{array}$$

If $o \neq o'$, we may consider that the system seems to have answered to the pressing of A . There are two counter-arguments to this conclusion.

1. S may have decided in advance to output o' ;
2. there is no reason for an observable consequence to occur immediately after pressing the button.

In order to obtain a correct solution, the main point to take into account is that the observable state is not only function of the inputs but also of the internal state. From now on, we will assume that we have two identical copies of S , and we will proceed to the experiment simultaneously with the button A and the button B .

$$\begin{array}{c} \text{Input} \\ \text{Observable state} \end{array} \left| \begin{array}{c} S \xrightarrow{A} S_A \\ o \quad \quad o' \end{array} \right| \begin{array}{c} S \xrightarrow{B} S_B \\ o \quad \quad o'' \end{array}$$

If $o' \neq o''$, we deduce that the system has distinguished between pressing the button A and the button B . This experiment is thus strictly more informative than the previous one. In the other case, knowing that $o' = o''$ is not enough for us to extract any information on the internal behavior of the system. Indeed, the second counter-argument advances that the observable reaction can occur after an arbitrary number of transitions. A solution is to iterate the experiment on S_A and S_B until observing a difference, but the observable state then becomes possibly correlated to the other choices A or B performed during the experiment. The choices must thus be identical for S_A and S_B .

We can informally define reactivity by stating that if there exists a finite sequence of experiments (i.e. a word on $\{A, B\}$) allowing to distinguish the systems S_A and S_B , then S is reactive. In formal terms, this is equivalent to stating that S_A and S_B must be non-bisimilar. The reaction *time* is the maximum length of the minimal experiment allowing to prove non-bisimilarity.

2.2 Notations and definitions

Some definitions will be useful to our work. Let Σ be a set. The set of finite words on Σ is noted Σ^* , and the set of infinite words is noted $\Sigma^\omega \equiv \mathbb{N} \rightarrow \omega$. The set of finite and infinite output words is $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. The length of a finite word w will be noted $|w|$. For any word w we will note $\text{prefix}(w, len)$ the prefix of w of length len , and we also note $w[i]$ the i -th symbol of w , where $i \in [0; |w| - 1]$.

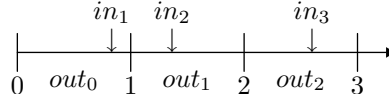
The singleton set is $\mathbb{1} = \{\star\}$ (up to isomorphism), and the disjoint union of two sets A and B is noted $A + B$. The set of natural integers strictly inferior to x is noted $\mathbb{N}^{<x}$.

3 A formal model of synchronous systems

We aim at giving a formal model of synchronous systems sufficiently expressive to encode languages such as Lustre [8] and PsyC [5]. Our formalism is an adaptation of Moore machines [10].

3.1 The synchronous abstraction

We will restrict ourselves to the set of systems which respect the synchronous mode of computation. In this model, the computation is divided in successive *rounds*. Each transition from a round to the next denotes the *tick* of a global logical clock. The observable state of a system is constant on each round, and changes only at the boundary between rounds. At each new round, the new internal state is a function of the current input and internal state (equivalently, the internal state is a function of the initial internal state and all previous inputs). The observable state is only function of the internal state. The following timeline shows an example of a deterministic synchronous computation involving three successive rounds.



In this example, the output out_0 is a function of the initial internal state only; the output out_1 is a function of the initial internal state and in_1 ; and the output out_2 is a function of the initial internal state, in_1 and in_2 . Thus, inputs have no *immediate* effect on the observable state.

3.2 State-based description of synchronous systems

We define our synchronous systems as a labeled transition system (LTS) inspired by Moore machines. We recall here some definitions which will be useful in the following developments.

Definition 1 (Synchronous system). *Let In be a set of inputs and Out be a set of outputs. A synchronous system $S = \langle In, Out, Q, E, out, q_i \rangle$ is the data of:*

- a set of states Q ,
- a transition relation $E \subseteq Q \times In \times Q$,
- a labeling function associating states to outputs $out : Q \rightarrow Out$,
- and an initial state $q_i \in Q$.

The sets In and Out are the signature of S . We will note $p \xrightarrow{a} q$ as a shorthand for $(p, a, q) \in E$. Moreover, we constrain our systems to be finitely branching and to be complete, i.e. $\forall p \in Q, \forall a \in In, \exists p \xrightarrow{a} q$ ¹.

The computational meaning of a LTS is expressed using the notion of *run*. It allows to define the *output language* associated to an input word.

Definition 2 (Run of a synchronous system, output language). *Let $S = \langle In, Out, Q, E, out, q_i \rangle$ be a synchronous system. We define the notions of finite run and the associated output language.*

Finite runs. *Let $w \in In^*$ be a finite input word. The set of finite, maximal runs of S on w starting from state $q_0 \in Q$ is $Runs_S^*(q_0, w) \subseteq Q \times (In \times Q)^*$ and is defined as:*

$$Runs_S^*(q_0, w) = \{q_0.w[0].q_1 \dots w[|w|-1].q_{|w|} \mid \forall i \in [0; |w|-1], q_i \xrightarrow{w[i]} q_{i+1}\}.$$

Output language. *The output language of S associated to w and q_0 is $\mathcal{L}_S^*(q_0, w) \subseteq Out^*$ and is defined as follows:*

$$\mathcal{L}_S^*(q_0, w) = \{out(q_0).out(q_1) \dots out(q_{n-1}) \mid q_0.a_0.q_1.a_1 \dots \in Runs_S^*(q_0, w)\}.$$

The classical equivalence relation on states of labeled transition systems is bisimilarity. Its definition is slightly adapted to our notion of synchronous system.

Definition 3 (Bisimilarity, non-bisimilarity). *Let $S = \langle In, Out, Q, E, out, q_i \rangle$ be a synchronous system. A relation $R \subseteq Q \times Q$ is said to be a (strong) **bisimulation** if and only if the following condition holds:*

$$\forall (p, q) \in R, out(p) = out(q) \wedge (\forall p \xrightarrow{a} p', \exists q \xrightarrow{a} q', (p', q') \in R) \wedge (\forall q \xrightarrow{a} q', \exists p \xrightarrow{a} p', (p', q') \in R).$$

If there exists such a relation R s.t. $(p, q) \in R$, then p and q are said to be bisimilar, which is noted $p \sim q$. Moreover, bisimilarity is an equivalence relation. Conversely, the negation of bisimilarity $\not\sim \subseteq Q \times Q$ is inductively defined by the rules below. In these rules, $p, q \in Q$ and $a \in In$ are universally quantified.

$$\frac{\text{BASE}}{out(p) \neq out(q) \rightarrow p \not\sim q} \quad \frac{\text{IND}}{\exists p \xrightarrow{a} p', \forall q \xrightarrow{a} q', p' \not\sim q' \vee \exists q \xrightarrow{a} q', \forall p \xrightarrow{a} p', p' \not\sim q'}{p \not\sim q}$$

¹In practice, these conditions constrain the input and output data sets to be finite.

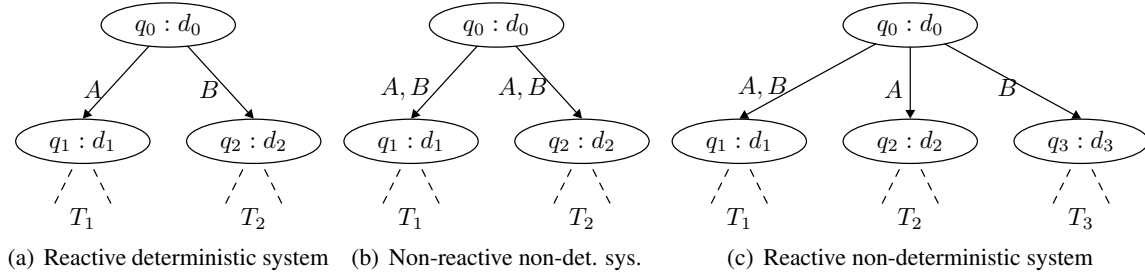


Figure 1: Some reactive and non-reactive systems

In this paper, except when stated otherwise, all state spaces shall be assumed to be quotiented by bisimulation equivalence.

4 Reaction time of a state in a synchronous system

This section formalizes the ideas exposed in the case study, in Sec.2.1, and extends them to non-deterministic systems. The case study proposes to model reaction as a functional dependency between a set of inputs and the future behavior of the system. In our formal model, these future behaviors are represented as the successor states of the considered state.

In this setting, we will first define a notion of reactivity inspired by functional dependency, and then define reaction time as the necessary time to prove that two successor states are not bisimilar.

4.1 Reactivity

Let $In = \{A, B\}$ and Out be a non-singleton set. Let $S = \langle In, Out, Q, out, E, q_i \rangle$ be a synchronous system. and let's assume that state $q \in Q$ is as depicted in Fig. 1(a). We observe that there are two inputs A and B leading to non-bisimilar states q_1 and q_2 . In this case, state q is thus reactive. In the case of non-deterministic systems, we must generalize this idea: if there exists an asymmetry in the possible transitions of a system, then it is reactive. Let's assume that state q is as depicted in Fig. 1(b). There, q is not reactive because there is a symmetry between the transitions possible with A and the transitions possible with B . This symmetry is broken in Fig. 1(c). Non-determinism highlights the fact that reactivity is a kind of non-bisimilarity.

Definition 4 (Reactivity of a state in a synchronous system, separating pair). *Let In, Out be two sets, $S = \langle In, Out, Q, out, E, q_i \rangle$ be a synchronous system and $q \in Q$ be a state. We will denote $reactive(q)$ the fact that q is reactive. The predicate $reactive(q)$ is defined as follows:*

$$reactive(q) \triangleq \exists a_1, a_2 \in In, a_1 \neq a_2 \wedge \left(\exists q \xrightarrow{a_1} q_1, \forall q \xrightarrow{a_2} q_2, q_1 \not\sim q_2 \right).$$

*The pair of inputs (a_1, a_2) is a **separating pair** of q . It is deterministic iff $\forall q \xrightarrow{a_1} q_1, \forall q \xrightarrow{a_2} q_2, q_1 \not\sim q_2$. The set of separating pairs of q is noted $SepPairs(q)$, and the deterministic subset is $DSepPairs(q)$.*

4.2 Observable effects

Observable effects stem from a fine-grained study of reactivity. In this section, we show that an observable effect characterizes a temporally localized difference between the behaviors of non-bisimilar states.

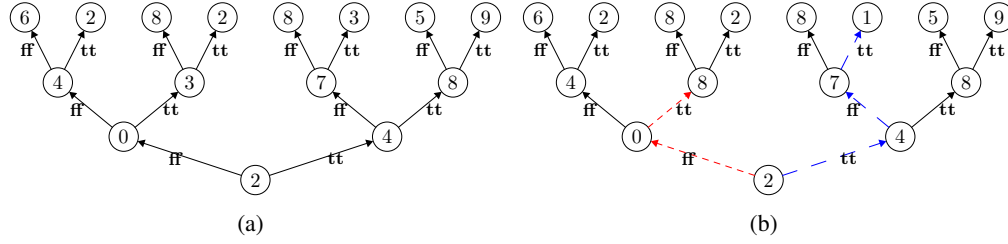


Figure 2: A pair of separable LTS

We show that an input data has observable effects on the system on a not necessarily finite interval.

Characterizing the difference between two states. Characterizing difference between states can be done by studying the negation of bisimulation.

Definition 5 (Separators, strongly separable states). *Let $S = \langle In, Out, Q, out, E, q_i \rangle$ be a synchronous system and $p_1, q_1 \in Q$ s.t. $p_1 \not\sim q_1$. A constructive proof of $p_1 \not\sim q_1$ is the data of (at least) two separating runs $r_1 \in Runs_S^*(p_1, w)$ and $r_2 \in Runs_S^*(q_1, w)$:*

$$r_1 = p_1 \cdot a_1 \cdot p_2 \cdot a_2 \dots a_n \cdot p_n \text{ and } r_2 = q_1 \cdot a_2 \cdot q_2 \cdot a_2 \dots a_n \cdot q_n.$$

*These runs are labelled on input by a finite word $w = a_1 \cdot a_2 \dots a_n$ called **separator**, and generate output words $o_1 \in Out^* = out(p_1) \cdot out(p_2) \dots out(p_n)$ and $o_2 \in Out^* = out(q_1) \cdot out(q_2) \dots out(q_n)$. More generally, any separator w induces a nonempty set $\mathcal{O}(p_1, q_1, w)$ of pairs of different output words (o_1, o_2) generated by separating runs s.t. $o_1 \in \mathcal{L}_S^*(p_1, w), o_2 \in \mathcal{L}_S^*(q_1, w)$.*

*A separator w is **deterministic** when all its runs are separating, i.e. all runs stems from a proof of $p_1 \not\sim q_1$. The set of separators of two states p, q is noted $S(p, q)$, and the set of deterministic separators is noted $DS(p, q)$. Note that $DS(p, q) \subseteq S(p, q)$.*

*Two states p, q are said to be **strongly separable**, noted $p \bowtie q$, iff all infinite inputs words are prefixed with a deterministic separator.*

Fig. 2 shows two separable LTS. The fact that they are non-bisimilar is proved by the existence of two separators (although one would suffice) of length two and three, as emphasized by the dotted paths.

Once separability of two states defined, we can define what is an observable effect and *when* it occurs. This is based on observing the differences in the output word pairs generated by a separator.

Definition 6 (Observable effect). *Let $p, q \in Q$ be two states. Let $w \in In^*$ be an input word. The observable effects are generated by all the prefixes of w which are separators.*

$$\begin{aligned} \text{diff}_{p,q} &: \prod_{w \in In^*} \mathbb{N}^{<|w|} \rightarrow \mathbb{1} + (Out \times Out) \\ \text{diff}_{p,q}(w, n) &= (x_1, x_2) \leftrightarrow \text{prefix}(w, n+1) \in S(p, q) \wedge (o_1, o_2) \in \mathcal{O}(p, q, \text{prefix}(w, n+1)) \wedge \\ & \quad x_1 = o_1[n] \wedge x_2 = o_2[n] \end{aligned}$$

$\text{diff}_{p,q}(w, n)$ returns $\star \in \mathbb{1}$ if there is no difference at index n or a pair $(o_1[n], o_2[n])$ s.t. $o_1[n] \neq o_2[n]$ at the same index. These differences are the **observable effects** induced by w .

The existence of a separator ensures that there *may* be an observable effect. Fig. 3 show two systems (whose state space is quotiented by bisimulation equivalence) with an unknown output data X in Fig.

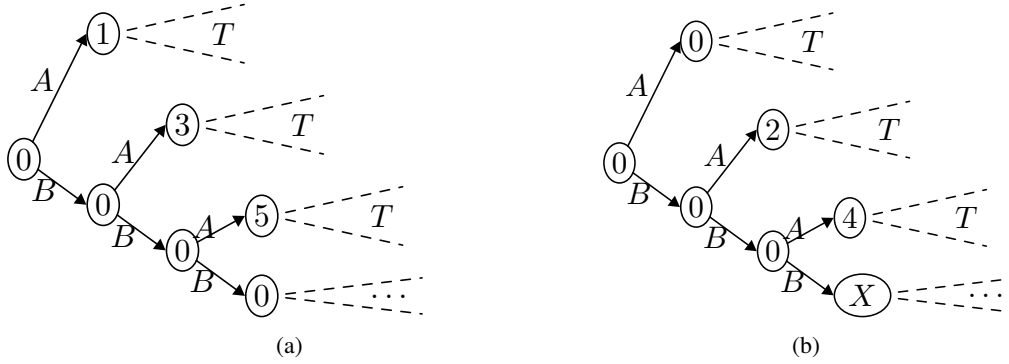


Figure 3: Partial separability

3(b). State names are omitted and the nodes only contain the output data. If $X = 0$, the initial states are separable with the words of the language B^*A . These are *deterministic* separators: reading them on input ensures an observable effect. We observe that the input words of the language B^∞ do not yield an observable effect. On the other hand, if $X \neq 0$, all infinite words are prefixed by a (deterministic) separator. An eventual observable reaction is guaranteed.

4.3 Reaction time

Since we work with logical time, the occurrence times of observable effects are their indices in the associated output traces. We may *define* the reaction time of a state as the maximum of the occurrence time of the first observable effect. This yields two possible views of reaction time: an optimistic one (an observable effect *may* arise ...) and a pessimistic one (an observable effect *must* arise). Moreover, the reaction time of a state can be valid for *all* contexts or just for *some*. Our application domain requires that we choose a pessimistic approach. Compositionality in turn requires that we quantify over all possible contexts when defining reaction time, as will be shown later.

Definition 7 (Deterministic reaction time). *The (deterministic) reaction time of a state w.r.t. an input is the maximum number of transitions that must be performed to see the first observable effect arise, for any input. Let $q \in Q$ be a state s.t. $\text{reactive}(q)$ holds. We note by $\text{detruntime}(q) = t$ the fact that q has a reaction time of t transitions, where:*

$$\text{detruntime}(q) = \max\{ n \mid (a_1, a_2) \in \text{DSepPairs}(q), q \xrightarrow{a_1} q_1, q \xrightarrow{a_2} q_2, q_1 \not\sim q_2, \\ w \in \text{In}^\omega, \text{diff}_{q_1, q_2}(w, n) \neq * \wedge \forall n' < n, \text{diff}(w, n') = * \}$$

5 Observable effects under composition

In the previous section, we have defined a notion of observable effects for synchronous systems. In this section, we will investigate the way observable effects evolve when synchronous systems are composed. To this end, we will define a small process algebra, inspired by the category-theoretical work of Abramsky on concurrency [1].

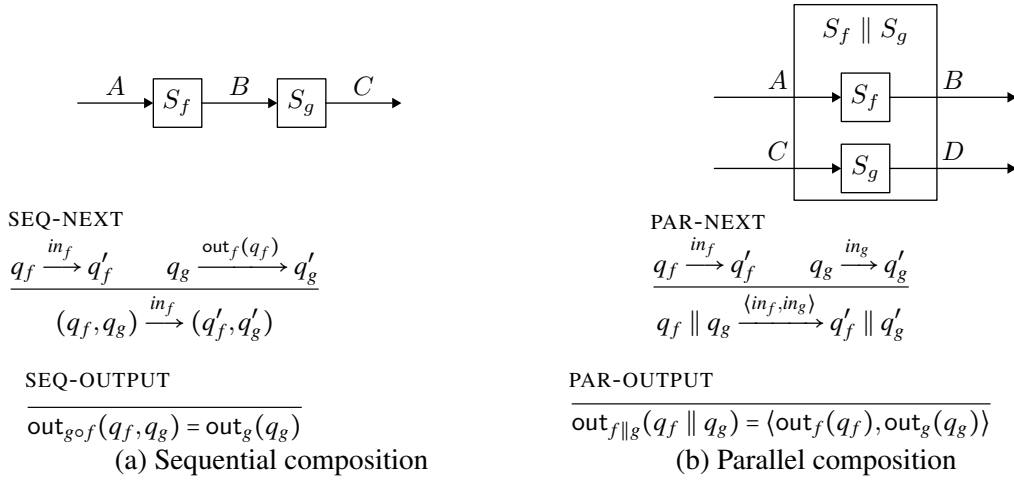


Figure 4: Definition of the composition operations

5.1 Data types

In order to model multiple input-output ports, we will force a monoidal structure on the data processed by our synchronous systems. Let $Basic = \{\mathbf{int}; \mathbf{bool}; \mathbb{1}; \dots\}$ be a set of basic datatypes. The set of datatypes is the monoid $\langle \mathcal{D}, \times \rangle$ generated by $Basic$ and closed by cartesian product.

5.2 Composition operators

Our composition operators are sequential composition and parallel composition. The transition relations of the compound systems are defined in a classic way, using a small-step semantics given by SOS inference rules (c.f. Fig. 4).

Sequential composition. Let A, B, C be three sets. Let $S_f = \langle A, B, Q_f, E_f, out_f, q_{i,f} \rangle$ and $S_g = \langle B, C, Q_g, E_g, out_g, q_{i,g} \rangle$ be two systems. The sequential composition $S_g \circ S_f$ proceeds by redirecting the output of S_f to the input of S_g . The compound system is $S_g \circ S_f = \langle A, C, Q_f \times Q_g, E_{g \circ f}, out_{g \circ f}, (q_{i,f}, q_{i,g}) \rangle$, where $E_{g \circ f}$ and $out_{g \circ f}$ are defined in Fig. 4.

Parallel composition. Let A, B, C, D be four sets. Let $S_f = \langle A, B, Q_f, E_f, out_f, q_{i,f} \rangle$ and $S_g = \langle C, D, Q_g, E_g, out_g, q_{i,g} \rangle$ be two systems. The parallel composition proceeds by pairing the respective transitions of S_f and S_g in a synchronous way. The compound system is $S_f \parallel S_g = \langle A \times C, B \times D, Q_f \times Q_g, E_{f \parallel g}, out_{f \parallel g}, (q_{i,f}, q_{i,g}) \rangle$ where $E_{f \parallel g}$ and $out_{f \parallel g}$ are also defined in Fig. 4.

Other operations. An other important operation is the feedback. We omit it for space reasons, but it must be noted that it exhibits the same behavior as sequential composition. The other operations necessary to make our definitions into an usable process algebra are structural ones, like data duplication, erasure, etc. These important details are omitted from the following study.

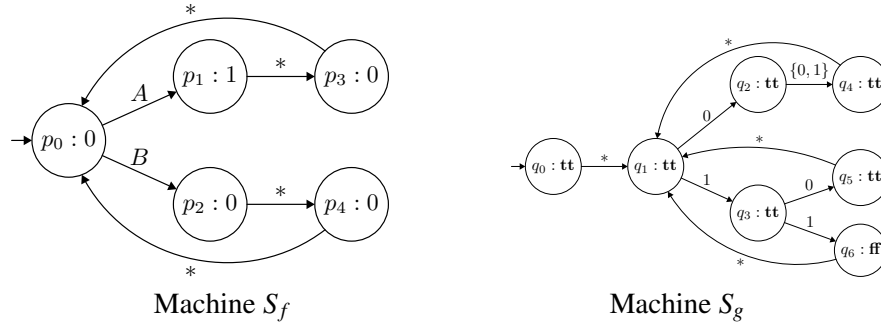
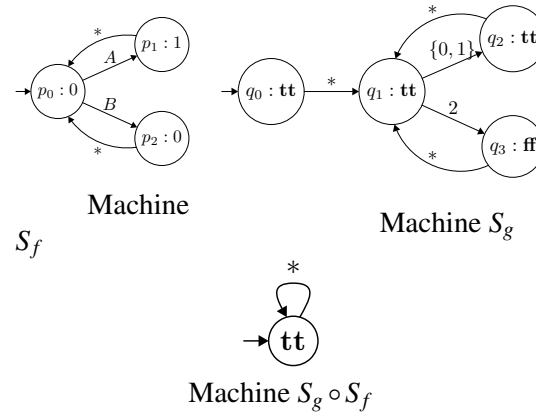


Figure 5: Example of disappearing separator

5.3 Observable effects w.r.t. sequential composition

In this section, we study the behavior of the observable effects of systems when they are composed. We restrict our attention to sequential composition since it is easy to show that parallel composition doesn't alter the behavior of the sub-components. We show that under sequential composition, whenever reactivity still holds, the observable effects can vary arbitrarily. Our examples will be given on Moore machines whose state space is not quotiented by bisimulation equivalence.

The proof that reactivity can be lost follows the same argument that shows that the composition of two non-constant total functions can be constant. The figure to the right shows the composition of two reactive Moore machines S_f and S_g whose composition is not reactive. In this example, this stems from the fact that the observable effect $(0, 1)$ of the input received in state p_0 of the machine S_f is not "taken into account" by the machine S_g , i.e. $(0, 1)$ is not a separating pair of state q_1 .



The fact that an observable effect of S_f is a separating pair of S_g is not enough to guarantee an observable effect on output. Sequential composition restricts the input language of the system in receiving position (here, S_g). This means that separators can appear and disappear arbitrarily. The two Moore machines in Fig. 5 are modifications of the earlier ones. The states p_0 and q_1 are still reactive, but when composed the output symbols on p_3 and p_4 restricts the set of inputs of the states q_2 and q_3 to the word 0 . Thus, q_2 and q_3 are no more separable and the result is the constant machine shown earlier.

The conclusion of this study confirms the intuition: there is no general way of guaranteeing functional dependencies. These results extend to reaction time, which is not conserved: the receiving machine may ignore the first observable effect and take into account ulterior ones.

In verification terms, this means that in order to verify that the composition of two systems is reactive, a full search of the state space for separators must be undertaken. In the next section, an approximate but compositional method to simplify this process is proposed.

6 Under-approximating observable effects

This section proposes a partial solution to some problems encountered earlier, namely:

1. the fact that non-deterministic separators do not guarantee an observable effect,
2. the non-compositionality of reactivity and observable effects.

We proceed by reducing our focus to the cases where reactivity, which is a branching-time property of states, can be reduced to a linear-time one. We show how to compute the separators and separating pairs which are preserved when “merging” all branches of the computation tree.

Let us assume the existence of two sets of data In and Out . Let q be a state such that $\text{reactive}(q)$ holds, and let $(a_1, a_2) \in \text{SepPairs}(s)$ be a separating pair of inputs. In Sec. 4.3, we observed that in order to ensure the occurrence of an observable effect and the existence of a reaction time, q must be such that all inputs are deterministic separators for all q_1 and q_2 s.t. $q \xrightarrow{a_1} q_1$ and $q \xrightarrow{a_2} q_2$. If this condition is met, we can compute *deterministic observable effects*, i.e. effects which exists for *all* separators. Similarly, we can define *deterministic separating pairs*.

First, we define some operations in order to merge sequences of observable effects. We define the operation $\oplus : (\mathbb{1} + Out \times Out) \times (\mathbb{1} + Out \times Out) \rightarrow (\mathbb{1} + Out \times Out)$ as:

$$\begin{aligned} x \oplus x &= x \\ x \oplus y &= * \text{ if } x \neq y. \end{aligned}$$

The extension of this operation to sequences of symbols on $(\mathbb{1} + Out \times Out)$ is defined straightforwardly. If $d_1, d_2 \in (\mathbb{1} + Out \times Out)^\omega$ are two infinite sequences, their merging is also noted $d_1 \oplus d_2$.

Definition 8 (Deterministic observable effects, observational order). *Let q be a state s.t. $\text{reactive}(q)$ holds. The sequence of deterministic observable effects of q is noted $\text{DOE}(q)$ and is defined as follows:*

$$\text{DOE}(q) = \bigoplus_{(a_1, a_2) \in \text{SepPairs}(q)} \{ \text{diff}_{q_1, q_2}(w) \mid q \xrightarrow{a_1} q_1, q \xrightarrow{a_2} q_2, w \in In^\omega \}.$$

It is possible to define a relation $< \subseteq (\mathbb{1} + Out \times Out)^\omega \times (\mathbb{1} + Out \times Out)^\omega$, where:

$$w_1 < w_2 \leftrightarrow \exists ! i, (\forall j \neq i, w_1[j] = w_2[j]) \wedge (w_1[i] = * \wedge w_2[i] \neq *).$$

The reflexive-transitive closure of $<$ is the **observational order** and is noted \leq . The set $\text{ObsOrder}(q)$ of infinite strings partially ordered by \leq which has $\text{DOE}(q)$ as greatest element and $*^\omega$ as least element is called by extension the *observational order on q* .

We must also define linear time-proof separating pairs, called *strongly separating pairs*. Let’s consider the systems in Fig. 6, in which only the output data is displayed and state names are omitted. The systems 1 and 2 are symmetrical and have both (A, B) as a separating pair for their initial state. However, (A, B) is *not* a separating pair for the *union* of the two systems. We must define a notion of separating pair for two systems which resists their union.

Definition 9 (Strongly separating pairs). *Let q_1, q_2 be two states s.t. $\text{reactive}(q_{1,2})$ holds. The pair $(a_1, a_2) \in \text{SepPairs}(q_{1,2})$ is a strongly separating pair of the union of q_1 and q_2 if and only if:*

$$\left(\exists q_1 \xrightarrow{a_1} q'_1, \forall q_2 \xrightarrow{a_2} q'_2, q'_1 \neq q'_2 \right) \vee \left(\exists q_1 \xrightarrow{a_2} q'_1, \forall q_2 \xrightarrow{a_1} q'_2, q'_1 \neq q'_2 \right).$$

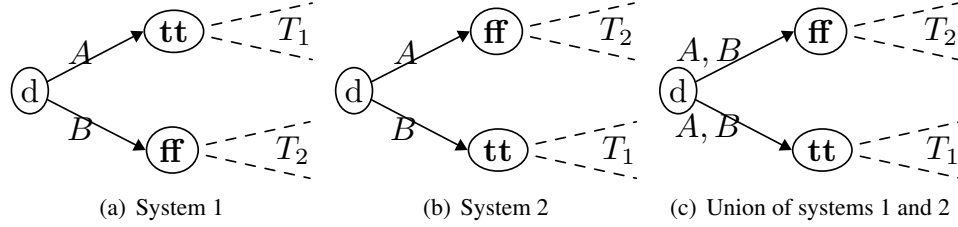


Figure 6: Union of transition systems

The set of strongly separating pairs of q_1 and q_2 is noted $\text{SSP}(q_1, q_2)$. Note that $\text{SepPairs}(q) = \text{SSP}(q, q)$. The sequence of strong separating pairs of q_1 and q_2 is noted $\text{SSPseq}(q_1, q_2)$ and is defined as follows:

$$\text{SSPseq}(q_1, q_2) = \text{SSP}(q_1, q_2) \cdot \left(\overline{\bigcap} \{ \text{SSPseq}(q'_1, q'_2) \mid (a_1, a_2) \in \text{SepPairs}(q), q_1 \xrightarrow{a_1} q'_1, q_2 \xrightarrow{a_2} q'_2 \} \right),$$

where $\overline{\bigcap}$ is the extension of set intersection to sequences of sets. Now, let q be s.t. $\text{reactive}(q)$ holds. The sequence of strongly separating pairs of q is:

$$\text{SSPseq}(q) = \text{SepPairs}(q) \cdot \overline{\bigcap} \{ \text{SSPseq}(q_1, q_2) \mid (a_1, a_2) \in \text{SepPairs}(q), q \xrightarrow{a_1} q_1, q \xrightarrow{a_2} q_2 \}.$$

Deterministic observable effects are in fact an *abstraction* of the original system. The concretization operation is the function associating to a sequence of deterministic observable effects the set of all systems which have at least these deterministic observable effects (w.r.t. \leq). Using this abstraction, checking the compositionality of sequential composition is straightforward.

Lemma 1 (Sequential composition of deterministic observable effects ensures reactivity). *Let $S_f = \langle A, B, Q_f, E_f, \text{out}_f, q_{i,f} \rangle$ and $S_g = \langle B, C, Q_g, E_g, \text{out}_g, q_{i,g} \rangle$ be two systems. Let $q_f \in Q_f$ and $q_g \in Q_g$ be two states such that (q_f, q_g) is in the state space of the sequential composition $S_g \circ S_f$. We have $\text{reactive}(q_f, q_g)$ if:*

$$\exists d \in \text{ObsOrder}(q_f), \exists i, d[i] \in \text{SSPseq}(q_g)[i+1].$$

Proof. Let $i \in \mathbb{N}$ be such that $d[i] \in \text{SSPseq}(q_g)[i+1]$. Having $d[i] = (x_1, x_2)$ implies that $\text{reactive}(q_f)$ holds. Hence, there exists $a_1 \neq a_2$ s.t. $\exists q_f \xrightarrow{a_1} q_f^1, \forall q_f \xrightarrow{a_2} q_f^2, q_f^1 \neq q_f^2$. By hypothesis, we know that all input words are separators of all such (q_f^1, q_f^2) . By definition, $d[i]$ is an observable effect of all these separators.

Hence, for all input words w there will exist two runs $q_f^1 \xrightarrow{\text{prefix}(w,i)^+} r_f^1$ and $q_f^2 \xrightarrow{\text{prefix}(w,i)^+} r_f^2$ s.t. $\text{out}(r_f^1) = x_1$ and $\text{out}(r_f^2) = x_2$. By definition of the sequential composition, this induces the runs $(q_f^1, q_g^1) \xrightarrow{\text{prefix}(w,i)^+} (r_f^1, r_g^1)$ and $(q_f^2, q_g^1) \xrightarrow{\text{prefix}(w,i)^+} (r_f^2, r_g^2)$ (with $q_g \xrightarrow{\text{out}(f)} q_g^1$). Since $(x_1, x_2) \in \text{SSP}(r_g^{1,2})$ (by definition of SSPseq), $(q_f^1, q_g^1) \neq (q_f^2, q_g^1)$. \square

The compositionality of this approach stems from the fact that for any systems S_f, S_g and respective states q_f and q_g , an element of $\text{ObsOrder}(q_f, q_g)$ can be computed using other elements from ObsOrder and SSP .

Definition 10 (Compositionality of deterministic observable effects). *Let $q_f \in Q_f, q_g \in Q_g$ be two states. Let $\text{doe}_f \in \text{ObsOrder}(q_f)$ and $\text{dsp}_g \in \text{SSP}(q_g)$. If there exists a t s.t. $\text{doe}_f[t] \in \text{dsp}_g[t+1]$ then there exists an element $\text{doe}_{g \circ f}^t \in \text{ObsOrder}(q_f, q_g)$ s.t.:*

$$\text{doe}_{g \circ f}^t = \star \cdot \star^t \cdot \bigoplus \{ \text{doe}_{g'} \mid (q_f, q_g) \rightarrow^{t+1} (q'_f, q'_g), \text{doe}_{g'} \in \text{ObsOrder}(q'_g) \}.$$

$$\begin{array}{c}
\frac{}{\text{skip} : \text{comm}} \quad \frac{}{\Gamma, x : \sigma \vdash x : \sigma} \quad \frac{b \in \{\mathbf{tt}, \mathbf{ff}\}}{\Gamma \vdash b : \text{exp}(\text{bool})} \quad \frac{n \in \mathbb{N}}{\Gamma \vdash n : \text{exp}(\text{int})} \quad \frac{\Gamma \vdash V : \text{var}(\tau)}{\Gamma \vdash !V : \text{exp}(\tau)} \quad \frac{\Gamma \vdash V : \text{var}(\tau) \quad \Gamma \vdash E : \text{exp}(\tau)}{\Gamma \vdash V := E : \text{comm}} \\
\frac{\Gamma \vdash A_0 : \text{exp}(\text{bool}) \quad \Gamma \vdash A_0 : \sigma \quad \Gamma \vdash A_1 : \sigma}{\Gamma \vdash \text{if } A_0 \text{ then } A_1 \text{ else } A_2 : \text{comm}} \quad \frac{\Gamma \vdash A_0 : \text{comm} \quad \Gamma \vdash A_1 : \sigma}{\Gamma \vdash A_0; A_1 : \text{comm}} \quad \frac{\Gamma \vdash A_0 : \text{exp}(\text{bool}) \quad \Gamma \vdash A_1 : \text{comm}}{\Gamma \vdash \text{while } A_0 \text{ do } A_1 \text{ done} : \text{comm}} \\
\frac{\Gamma \vdash A_0 : \pi_0 \text{ Out} \quad \dots \quad \Gamma \vdash A_{|Out|-1} : \pi_{|Out|-1} \text{ Out}}{\Gamma \vdash \text{tick}(A_0 \dots A_{|Out|-1}) : \text{comm}} \quad \frac{}{\Gamma \vdash \text{get}_i : \pi_i \text{ In} \quad i \in \mathbb{N}^{<|In|}}
\end{array}$$

Figure 7: Definition of well-typed programs.

This element is computed by merging the deterministic observable effects of states of q_g reachable in $t + 1$ transitions. The initial \star stems from the delay induced by communication in the synchronous model.

A similar property holds for separating pairs. We only describe informally how to proceed, since the general idea is similar to the case of deterministic observable effects. A strongly separating pair of q_f exists in $S_g \circ S_f$ if the states q_f^1, q_f^2 reachable by this pair have a common observable effect (computable using \oplus) which corresponds to a strongly separating pair of q_g .

7 Example

As explained in the introduction, our work focuses on a real-time system called OASIS, which provides a real-time kernel and a multi-agent synchronous-like language called PsyC (an extension of C with synchronous primitives). We have given a formal semantics of a simplification of PsyC called Psy-ALGOL. We will use this semantics to highlight a common use case of our framework.

7.1 Syntax and semantics of a simple synchronous language.

In order to give the semantics of a program, we have to define how its LTS is generated. We briefly survey a subset of the syntax of the language. The connection between the semantics and the resulting LTS should be straightforward, we will thus omit the derivations.

Syntax.

The syntax definition is given in an inductive way using inference rules on judgments of the shape $\Gamma \vdash M : \sigma$, meaning “in the context Γ , the program M has type σ ”. A context is a list of the shape $x_1 : \sigma_1, \dots, x_n : \sigma_k$. It associates variables x_i to their types σ_i . For the sake of simplicity, we assume that all variables are declared beforehand and initialized to their default value. We ignore procedures and we keep the other syntactical forms as simple as possible. The types σ and default values are defined as follows:

$$\begin{array}{ll}
\tau ::= \mathbf{int} \mid \mathbf{bool} & \text{default}_{\mathbf{int}} = 0 \\
\sigma ::= \mathbf{comm} \mid \mathbf{var}(\tau) \mid \mathbf{exp}(\tau) & \text{default}_{\mathbf{bool}} = \mathbf{ff}
\end{array}$$

Assuming that the programs have input and output types $\langle In, Out \rangle$, the set $Prog$ of correctly typed programs is defined in Fig. 7. We omit the arithmetical operators.

Semantics.

We will define the operational semantics for our language as a small-step relation. An operational semantics is usually a kind of relation associating a program in its initial configuration to its final outcome (be it a final value or divergence).

Our aim is slightly different: we want to view the evaluation of a program as a synchronous system. This means that instead of producing a final value or diverging, we want to quantify over all possible inputs at each logical step, and produce a LTS. In order to simplify matters, our LTS will be given in unfolded form, as an infinitely deep tree. Each step of the evaluation will grow this tree downward, and the limit of this process will be the semantics of the program. Let's proceed to some definitions.

Definition 11. *The set of configurations is $Config \triangleq Store(V) \times In \times Prog$, where V is the set of variables of the program and $Store(V)$ is a mapping from variables to constants.*

Definition 12. *The sets *Trees* of finite (resp. infinite) partial evaluation trees are generated by the inductive (resp. co-inductive) interpretation of the following rules.*

$$\frac{\text{LEAF} \quad conf \in Config}{conf \in Trees} \quad \frac{\text{NODE} \quad out \in Out \quad \forall in, tr_{in} \in Trees}{(out, \{(in, tr_{in}) \mid in \in In\})}$$

Let T be a partial evaluation tree with leaves $(st_i, in_i, prog_i)$. Given a syntactical mapping $map : Prog \rightarrow Prog$, we note $map \downarrow T$ the extension of map to the leaves of T such that $(map \downarrow T)$ has leaves $(st_i, in_i, map(prog_i))$.

The one-step reduction relation $\rightarrow \subseteq FiniteTrees \times FiniteTrees$ is defined in terms of a relation $\rightsquigarrow \subseteq Config \times FiniteTrees$ defined by the rules in Fig. 8.

We define \rightarrow as the application of \rightsquigarrow to the leaves of a tree. From there, we can define the standard reflexive-transitive closure of \rightarrow and its co-inductive counterpart as in [9].

7.2 Example of synchronous programs.

We will study the behavior of two programs, whose texts and associated (deterministic) LTS are displayed in Fig. 9. The first program captures an input data at the beginning of the outer loop and releases it on output when the inner loop finishes executing itself. The second program proceeds similarly, except that the inner loop termination is ensured by the usage of a decreasing counter y initialized to a constant N . In the LTS of program 2, this corresponds to the dashed transition between q_1 and q_2 , which should be understood as $N - 2$ omitted states with decreasing values of y .

We want to check whether the data inputted at the **tick**(!x); lines highlighted in both programs yield a finite reaction time. These instructions corresponds to states p_0, p_2 in LTS 1, and q_0, q_3 in LTS 2. Thus, in order for these instructions to be “reactive”, the corresponding states must have a finite reaction time. In order to study this, we will compute their deterministic observable effects.

Program 1. The set of separating pairs of p_0 and p_2 is $SepPairs(p_{0,2}) = \{(\mathbf{tt}, \mathbf{ff})\}$. This fact is proved by the transitions $p_0 \xrightarrow{\mathbf{tt}} p_1$, $p_0 \xrightarrow{\mathbf{ff}} p_0$, and $p_2 \xrightarrow{\mathbf{tt}} p_1$, $p_2 \xrightarrow{\mathbf{ff}} p_0$ where $p_0 \not\sim p_1$. The only separator of (p_0, p_1) is the one-symbol word $w = \mathbf{ff}$. This fact is proved by the transitions $p_0 \xrightarrow{\mathbf{ff}} p_0$ and $p_1 \xrightarrow{\mathbf{ff}} p_2$, where $out(p_0) \neq out(p_2)$. The separator w is *deterministic*, since the underlying automaton is itself deterministic. This separator induces a pair of output words $(o_1, o_2) = (\mathbf{ff}.\mathbf{ff}, \mathbf{ff}.\mathbf{tt})$ and thus an observable

$\frac{\text{CONST}}{(st, in, c) \succ (st, in, c) \quad c \in \mathcal{V}}$	$\frac{\text{VAR}}{(st, in, x) \succ (st, in, x) \quad x \in \text{dom}(st)}$	$\frac{\text{SEQ-CONTEXT}}{(st, in, A_0) \succ T}$ $\frac{}{(st, in, A_0; A_1) \succ (\lambda A'_0 \cdot A'_0; A_1) \downarrow T}$
$\frac{\text{SEQ-SKIP}}{(st, in, \mathbf{skip}; A_1) \succ (st, in, A_1)}$	$\frac{\text{DEREF-CONTEXT}}{(st, in, A) \succ T}$ $\frac{}{(st, in, !A) \succ (\lambda A' \cdot !A') \downarrow T}$	$\frac{\text{DEREF-VAR}}{}{(st, in, !x) \succ (st, in, st(x))}$
$\frac{\text{ASSIGN-CONTEXT}}{(st, in, A) \succ T}$ $\frac{}{(st, in, x := A) \succ (\lambda A' \cdot x := A') \downarrow T}$	$\frac{\text{ASSIGN-CONST}}{}{(st, in, x := v) \succ (st x \mapsto v, in, \mathbf{skip})}$	$\frac{\text{GET}}{}{(st, in, \mathbf{get}_i) \succ (st, in, \pi_i \text{ in})}$
$\frac{\text{IF-COND-CONTEXT}}{(st, in, A_0) \succ T}$ $\frac{}{(st, in, \mathbf{if } A_0 \mathbf{ then } A_1 \mathbf{ else } A_2) \succ (\lambda A'_0 \cdot \mathbf{if } A'_0 \mathbf{ then } A_1 \mathbf{ else } A_2) \downarrow T}$		
$\frac{\text{IF-TRUE}}{}{(st, in, \mathbf{if } \mathbf{tt} \mathbf{ then } A_1 \mathbf{ else } A_2) \succ (st, in, A_1)}$	$\frac{\text{IF-FALSE}}{}{(st, in, \mathbf{if } \mathbf{ff} \mathbf{ then } A_1 \mathbf{ else } A_2) \succ (st, in, A_2)}$	
$\frac{\text{WHILE-UNFOLD}}{}{(st, in, \mathbf{while } A_0 \mathbf{ do } A_1 \mathbf{ done}) \succ (st, in, \mathbf{if } A_0 \mathbf{ then } A_1; \mathbf{while } A_0 \mathbf{ do } A_1 \mathbf{ done else skip})}$		
$\frac{\text{TICK-CONTEXT-I}}{(st, in, A_i) \succ T}$ $\frac{}{(st, in, \mathbf{tick}(c_0 \dots c_{i-1}, A_i \dots A_{ Out -1})) \succ (\lambda A'_i \cdot \mathbf{tick}(c_0 \dots c_{i-1}, A'_i \dots A_{ Out -1})) \downarrow T}$		
$\frac{\text{TICK-CONSTANT}}{}{(st, in, \mathbf{tick}(c_0 \dots c_{ Out -1})) \succ ((c_0 \dots c_{ Out -1}), \{(input, (st, input, \mathbf{skip})) \mid input \in In\})}$		

Figure 8: One-step reduction relation

effect $\text{diff}_{p_0, p_1}(w, 1) = (\mathbf{ff}, \mathbf{tt})$. However, this observable effect is **not** deterministic, since there exist an infinite input word \mathbf{tt}^ω which generates no observable effect. Thus, the sequence of observable effects of p_0 and p_2 is \star^ω and the reaction time for the highlighted line instruction does not exist (or, equivalently, is infinite).

Program 2. The set of separating pairs of q_0 and q_3 is still $\text{SepPairs}(q_{0,3}) = \{(\mathbf{tt}, \mathbf{ff})\}$. The corresponding transitions are $q_0 \xrightarrow{\mathbf{tt}} q_1$, $q_0 \xrightarrow{\mathbf{ff}} q_0$ and $q_3 \xrightarrow{\mathbf{tt}} q_1$, $q_3 \xrightarrow{\mathbf{ff}} q_0$ with $q_0 \not\sim q_1$. The (deterministic) separators of (q_0, q_1) are $\text{S}(q_0, q_1) = \text{DS}(q_0, q_1) = \{\mathbf{ff}; \mathbf{tt.ff}; \mathbf{tt.tt.ff}; \dots; \mathbf{tt}^{N-2}.\mathbf{ff}; \mathbf{tt}^{N-1}.\mathbf{bool}\}$. The table below lists the observable differences associated to each separator.

\mathbf{ff}	$\mapsto \star.(\mathbf{ff}, \mathbf{tt}).\star^\omega$
$\mathbf{tt.ff}$	$\mapsto \star.\star.(\mathbf{ff}, \mathbf{tt}).\star^\omega$
	...
$\mathbf{tt}^{N-2}.\mathbf{ff}$	$\mapsto \star^{N-1}.(\mathbf{ff}, \mathbf{tt}).\star^\omega$
$\mathbf{tt}^{N-1}.\mathbf{bool}$	$\mapsto \star^N.(\mathbf{ff}, \mathbf{tt}).\star^\omega$

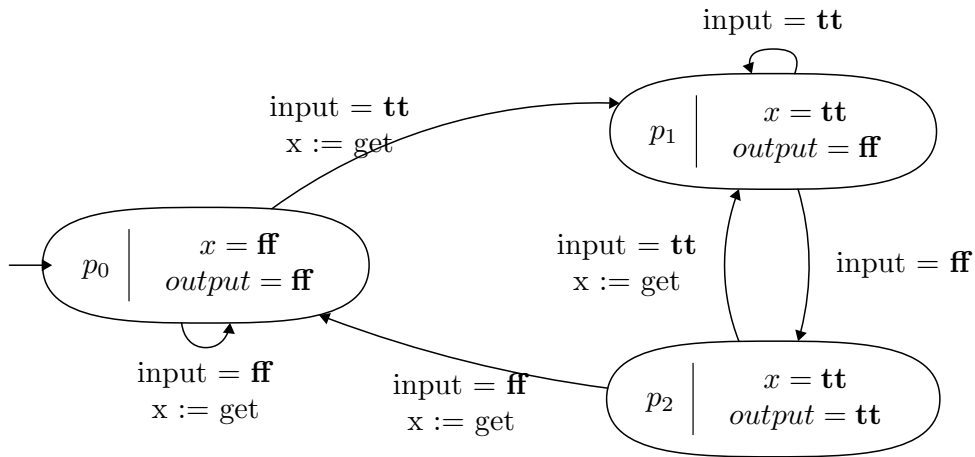
When merging these observable differences, we obtain $\text{DOE}(q_{0,3}) = \star^\omega$. This means that even though the program 2 is reactive with a finite reaction time, it is still non-compositional within our simple framework. This is due to the fact that the observable effects occurrence time are non-uniform w.r.t. inputs, i.e. non-constant.

```

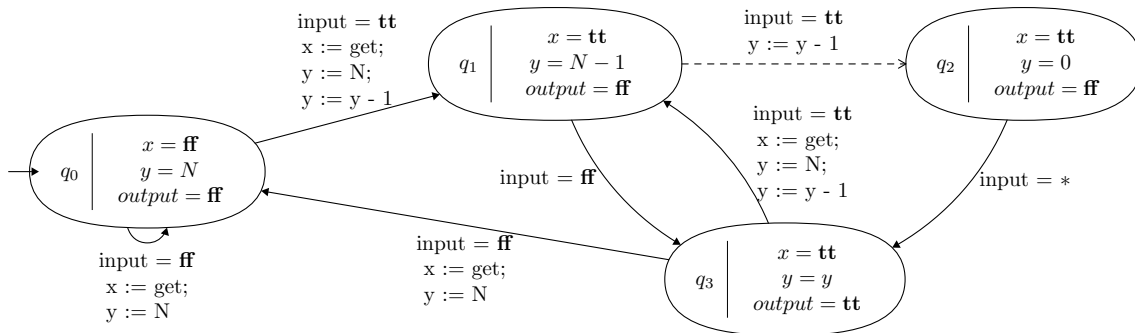
PROGRAM 1
x := ff ;
while tt do
  tick(!x); ←
  x := get;
  while get do
    tick(ff)
  done;
done

PROGRAM 2
x := ff ;
y := N;
while tt do
  tick(!x); ←
  x := get;
  y := N;
  while get ^ !y ≠ 0 do
    y := !y - 1;
    tick(ff)
  done;
done
    
```

(a) Programs 1 and 2



(b) LTS of program 1



(c) LTS of program 2

Figure 9: LTS of programs 1 and 2

8 Conclusions and future works

We have formalized in this paper the notions of functional dependency and reaction time for some synchronous systems. These notions are adapted to the formal investigation of reaction time constraints for the aforementioned synchronous systems. Functional dependencies were shown to be brittle and not suited to composition and verification. To answer this problem, we proposed an approximated method which gains compositionality by restricting its scope to deterministic separators.

Our work opens some other research directions: a broader investigation of the notion of reaction time in a more general setting [7] could prove fruitful and lead to simpler, more abstract and general definitions. Our composition operators are quite restricted, as shown in the example, but making it more flexible should be possible by making deterministic effects a function of some arbitrary decidable specification. It seems also possible to apply our ideas to the refinement-based development of systems.

We provide a formal framework allowing to reason on functional dependencies and reaction time which is amenable to automated verification. This effort should help the software designer and programmer to deliver reliable, predictable and efficient systems.

References

- [1] S. Abramsky, S. Gay & R. Nagarajan (1996): *Interaction Categories and the Foundations of Typed Concurrent Programming*. In M. Broy, editor: *Proceedings of the 1994 Marktoberdorf Summer School on Deductive Program Design*, Springer-Verlag, pp. 35–113.
- [2] Rajeev Alur & David L. Dill (1994): *A Theory of Timed Automata*. *Theoretical Computer Science* 126, pp. 183–235, doi:10.1016/0304-3975(94)90010-8.
- [3] R. Barbuti, C. Bernardeschi & N. De Francesco (2002): *Abstract interpretation of operational semantics for secure information flow*. *Inf. Process. Lett.* 83(2).
- [4] E. Clarke, D. Long & K. McMillan (1989): *Compositional model checking*. In: *Proceedings of the Fourth Annual Symposium on Logic in computer science*, IEEE Press, Piscataway, NJ, USA, pp. 353–362, doi:10.1109/LICS.1989.39190.
- [5] Vincent David, Jean Delcoigne, Evelyne Leret, Alain Ourghanlian, Philippe Hilsenkopf & Philippe Paris (1998): *Safety Properties Ensured by the OASIS Model for Safety Critical Real-Time Systems*. In: *SAFE-COMP*.
- [6] E. Allen Emerson & Joseph Y. Halpern (1982): *Decision procedures and expressiveness in the temporal logic of branching time*. In: *Proceedings of the fourteenth annual ACM symposium on Theory of computing, STOC '82*, ACM, New York, NY, USA, pp. 169–180, doi:10.1145/800070.802190.
- [7] Esfandiar Haghverdi, Paulo Tabuada & George J. Pappas (2005): *Bisimulation relations for dynamical, control, and hybrid systems*. *Theor. Comput. Sci.* 342, pp. 229–261, doi:10.1016/j.tcs.2005.03.045.
- [8] N. Halbwachs, P. Caspi, P. Raymond & D. Pilaud (1991): *The synchronous dataflow programming language Lustre*. *Proceedings of the IEEE* 79(9), pp. 1305–1320, doi:10.1109/5.97300.
- [9] Xavier Leroy & Hervé Grall (2008): *Coinductive big-step operational semantics* Available at <http://arxiv.org/abs/0808.0586>.
- [10] Edward F. Moore (1956): *Gedanken Experiments on Sequential Machines*. In: *Automata Studies*, Princeton U., pp. 129–153.
- [11] Mike Stannett (2006): *Simulation testing of automata*. *Formal Aspects of Computing* 18, pp. 31–41. doi:10.1007/s00165-005-0080-y.