Fundamenta Informaticae 78 (2007) 131–159
IOS Press

Correct-by-Construction Asynchronous Implementation of Modular Synchronous Specifications

Dumitru Potop-Butucaru
INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt — B.P. 105
78153 Le Chesnay Cedex, France
Dumitru.Potop_Butucaru@inria.fr

Benoît Caillaud*
IRISA / INRIA Rennes, Campus universitaire de Beaulieu
35042 Rennes Cedex, France
Benoit.Caillaud@irisa.fr

Abstract. In this paper, we introduce a new model for the representation of distributed asynchronous implementations of synchronous specifications. The model covers classical implementations, where a notion of global synchronization is preserved by means of signaling, and globally asynchronous, locally synchronous (GALS) implementations where the global clock is removed.

The new model offers a unified framework for reasoning about two essential correctness properties of an implementation: the preservation of semantics and the absence of deadlocks. We use it to derive criteria ensuring the correct deployment of synchronous specifications over GALS architectures. As the model captures the internal concurrency of the synchronous specification, our criteria support implementations that are less constrained and more efficient than existing ones.

Our work also reveals strong ties between abstract semantics-preservation properties and more operational ones like the absence of deadlocks.

Keywords: Synchronous systems, Asynchronous systems, Globally asynchronous, locally synchronous (GALS) architectures, Kahn processes, Endochronous systems, Semantics preservation

*Address for correspondence: IRISA / INRIA Rennes, Campus universitaire de Beaulieu, 35042 Rennes Cedex, France

1. Introduction

Dealing with concurrency, time and causality in the design of electronic systems has become increasingly difficult as the complexity of the designs grew.

The *synchronous programming model* [1, 2, 3] has had major successes at the specification level because it provides a simpler way to employ the power of concurrency in functional specification. Provided that a few high-level constraints ensure compliance with the synchrony hypothesis, the designer can forget about timing and communication issues and concentrate on functionality. The synchronous model features deterministic concurrency and simple composition mechanisms facilitating the incremental development of large systems. Also, synchronous models are usually easier to analyze/verify/optimize compared to asynchronous counterparts, often because the state-transition representations are smaller.

Synchronous languages like ESTEREL, LUSTRE, and SIGNAL, the quasi-synchronous STATECHARTS modeling methodology, and design environments like SIMULINK / STATEFLOW all benefit from the simplicity of the *synchronous hypothesis*:

1. Cycle-based execution model. Behaviors are sequences of *reactions* indexed by a *global logical clock*.

2. Within each reaction, the behavior is non-divergent and causal, so that the status of every signal is defined prior to being used in computations.

Note that condition 2 empowers the conceptual abstraction that computations and communications are infinitely fast ("zero-time") and take place at discrete points in time, with no duration. It also allows universally-recognized mathematical models like the Mealy machines and the digital circuits to be used as semantic foundations.

Eventhough the synchronous assumption simplifies system specification and verification, the problem of deriving a correct physical implementation from it does remain [2]. In particular, difficulties arise when the target implementation architecture has a distributed nature that does not match the synchronous assumption because of large variance in computation and communication speeds and because of the difficulty of maintaining a global notion of time. This is increasingly the case in complex microprocessors and Systems-on-a-Chip (SoC), and for many important classes of embedded applications in avionics, industrial plants, and the automotive industry.

For instance, many industrial embedded applications consist of multiple processing elements, operating at different rates, distributed over an extended area, and connected via communication buses. To use a synchronous approach in the development of such applications, one solution is to replace the asynchronous buses with communication infrastructures that comply with a notion of global synchronization. This is examplified by the family of Timed-Triggered Architectures introduced and promoted by H. Kopetz [4]. However, such a fully synchronous implementation must be conservative, forcing the global clock to run as slow as the slowest computation/communication process. The overhead implied by time-triggered architectures and synchronous implementations is often large enough to convince designers to use asynchronous solutions.

Gathering advantages of both the synchronous and asynchronous approaches, Globally Asynchronous Locally Synchronous (GALS) architectures are emerging as an architecture of choice for implementing complex specifications in both hardware and software. In a GALS system, locally-clocked synchronous components are connected through asynchronous communication lines. Thus, unlike for

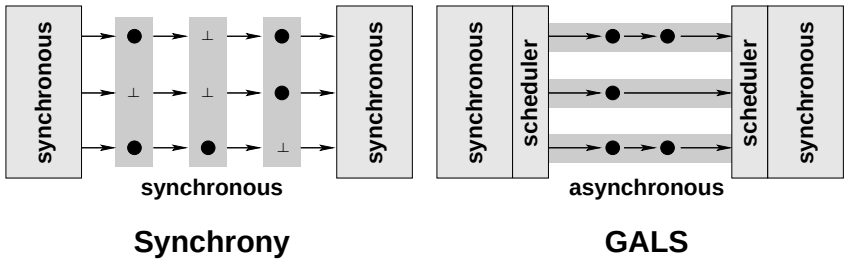


Figure 1. From synchrony to GALS. Bullets represent informative values (messages). Vertical gray boxes represent reactions. Horizontal ones represent asynchronous signals.

a purely asynchronous design, the existing synchronous tools can be used for most of the development process, while the implementation can exploit the more efficient/unconstrained/required asynchronous communication schemes.

We further pursue in this paper our quest for correct-by-construction deployment of synchronous specifications over GALS architectures.

1.1. Informal discussion of the issues

In the *synchronous paradigm* [1, 2, 3], an execution of the program, also called *trace*, is a sequence of reactions, each reaction assigning a unique value (status) to each variable of the program. Not all variables need to be involved in each reaction. However, this is taken into account by extending the domain of values of all variables with an extra symbol $\perp$, which denotes absence. Thus, absence can be tested and used to exercise control.

No global clock exists in the *asynchronous paradigm*, meaning that no notion of reaction exists, and that absence ($\perp$) has no meaning and cannot be sensed. Only the sequences of values on individual channels can be observed, so that an *asynchronous observation* of the execution of a system is a function assigning to each communication channel the sequence of transmitted messages/values. Asynchronously observing a synchronous execution consists of removing the $\perp$ events and the synchronization boundaries of the reactions to obtain an asynchronous observation.

In many cases, applications designed in a synchronous framework will be implemented for use in an asynchronous environment. Two problems arise: First, the synchronous applications must be fitted with wrappers that read the asynchronous inputs and schedule them into reactions before giving them to the program and triggering the program clock (the scheduling operation inserts the missing $\perp$ values). As the synchronous paradigm is often used in the development of safety-critical systems, input reading and the system itself must be deterministic, or at least predictable. It is therefore essential to consider classes of synchronous specifications that facilitate the development of efficient wrappers which make input reading deterministic while not restricting the behavior of the system.

Second, the implementation must preserve the semantics of the synchronous specification, meaning that the set of asynchronous observations of the specification must be identical to the set of observations of the implementation. Preservation of semantics is important because the advantages of synchrony lie with specification and verification. We would therefore like each implementation trace to be covered by the verification of the synchronous model. This problem is of particular importance when the

synchronous specification must be implemented over a distributed architecture (an operation called *desynchronization*). In such cases, input reading and computation must be coordinated between distributed sites, and doing this without a careful analysis can be very inefficient (in terms of speed, consumption, communication, etc.) or simply incorrect.

This paper addresses the problem of desynchronizing a modular synchronous specification by replacing the communication lines between modules with asynchronous FIFOs. Instead of a single, global wrapper, we shall have one wrapper per system component, as pictured in fig. 1. The exact problem we address is that of *characterizing large classes of synchronous components for which small, simple wrappers*[1] *produce deterministic, efficient, and semantics-preserving GALS implementations*. These classes of systems can then be considered as the implementation space, and the remaining problem is that of making given synchronous systems belong to these classes (by adding supplementary signaling). Naturally, a larger implementation space covers better solutions that use less synchronization.

1.2. Previous work

Previous approaches to implementing modular synchronous specifications over GALS architectures are respectively based on *latency-insensitive systems*, on *Kahn process networks (KPN)*, and on *endochronous and isochronous systems*.

In the *latency-insensitive systems* of Carloni *et al.* [5], each synchronous component reads every input and writes every output at each reaction. The communication protocols effectively simulate a single-clock system, which is inefficient, but simplifies the implementation.

In a *Kahn process network* [6], requiring that each component has a deterministic input/output behavior implies the determinism of the global system (and thus any wrapper is a good one). Often used, due to its robustness, in the development of embedded systems, the KPN-based approach has been adapted by Caspi *et al.* for the desynchronization of functional dataflow synchronous specifications [7]. Giving the approach its strength, the determinism is also its main drawback, as non-determinism is often useful in the specification and analysis of concurrent systems. We also mention here the approach of Talpin *et al.* [8], which is based on a bounded version of the Kahn principle.

The approach based on *endo/isochronous systems* has been proposed by Benveniste *et al.* [9] in order to support the analysis of partial specifications (which can be non-deterministic, or incomplete), to exploit execution modes, and to cover truly concurrent and multi-clock implementations. Informally speaking, a synchronous component is endochronous when the presence and absence of each variable can be inferred incrementally during each reaction from the current state and from the values of already present values. An endochronous component knows how to read its inputs, meaning that no wrapper is needed. Unfortunately, endochronous components can exhibit no internal concurrency, which makes endochrony non-compositional (thus, incremental system development is impossible). Isochrony is a semantics-preservation criterion over pairs of synchronous systems. The work of Singh and Theobald on *generalized latency-insensitive systems* [10] can be seen as implementing endochrony in hardware.

Essential improvement is brought by the work by Potop *et al.* [11] on weak endochrony and weak isochrony. Weak endochrony extends endochrony by allowing operations within a component to run independently when no synchronization is necessary. The notion is compositional, allowing incremental

[1] For instance, wrappers that trigger a transition as soon as the needed input is available.

development of large systems. Being formulated in a non-causal[2] framework, this approach is also less constrained than the KPN-based one, allowing non-determinism in the less abstract causal model. The non-causal framework is also the main disadvantage, because it hides implementation properties, like the presence of synchronization or communication deadlocks (which are important in practice).

The distribution of synchronous or strongly synchronized specifications has been studied in many other settings. We only mention here the Time-Triggered Architectures of Kopetz [4], the `ocrep` tool of Girault *et al.* [12], the AAA methodology of Sorel [13], and the desynchronization approach of Cortadella *et al.* [14].

From a more theoretical point of view, our work is closely related to results related to the confluence of asynchronous system models [15]. In this sense, our work is closely related to results concerning the design of delay-insensitive [16, 17, 18], speed-independent [19], and burst-mode [20] circuits (we will come back with a comparison in section 4.3).

1.3. Contribution

This paper brings an important improvement over previous work, by allowing us to *reason about concurrency and efficient synchronization in a causal, operational synchronous framework that takes into account the composition through read/write mechanisms*. The approach inherits the advantages of the weak endochrony-based approach: It allows the representation of non-deterministic specifications, takes into account execution and communication modes, and covers concurrent and multi-clock implementations. At the same time, it allows us to reason in a unified model about semantics-preservation and the absence of deadlocks due to synchronization and communication (which are both essential correctness properties of any implementation). As we shall see, *the level of detail is essential in this analysis, as it reveals the strong ties that exist between the two correctness properties, and simplifies the correctness analysis*.

Our main contribution is the definition of a new model for the representation of asynchronous implementations of synchronous specifications. The model covers classical implementations, where a notion of global synchronization is preserved by means of signaling, and globally asynchronous, locally synchronous (GALS) implementations where the global clock is removed. *We use this model to derive criteria ensuring the correct deployment of synchronous specifications over GALS architectures*.

1.4. Outline

The remainder of the paper is organized as follows: Section 2 defines the formal framework used throughout the paper, and section 3 gives intuitive examples and explains why the new structures are adapted to modeling and reasoning about the correctness of GALS implementations of synchronous specifications. Sections 4 defines criteria ensuring correct desynchronization. A short conclusion is given in section 5.

2. The model

This section defines our model of asynchronous implementation of a synchronous specification. We structured its presentation into several parts. The subsections 2.1, 2.2, and 2.3 introduce rather standard

[2]The term *causal/causality* covers here the execution order of the various operations that make up a synchronous reaction. The formalism presented in this paper has the means of representing this order. Other formalisms, including those of [9, 11], do not.

notations for transition systems (labels, traces, concurrent transition systems, and composition by synchronized product). Subsection 2.4 is the first to define communication channels, clocks, and the I/O transition systems which form our basic implementation model. Section 2.5 defines the synchronous transition systems – which are I/O transition systems satisfying the synchronous hypothesis. In section 2.6 we explain how transition systems are synchronously and asynchronously composed using FIFO models. Recall that intuitive examples are given later, in section 3.

2.1. Variables and labels

Our components and systems interact with each other and with their environment through *variables*. The *domain* of a variable v is denoted with $\mathcal{D}_v$. Given V a ***finite*** set of variables, a *label* over V is a partial valuation of its variables. Formally, the set of all labels over V is $\mathcal{L}_V = \prod_{v \in V} \mathcal{D}_v^\perp$, where $\mathcal{D}_v^\perp = \mathcal{D}_v \cup \{\perp\}$, and $\perp \notin \mathcal{D}_v$ is a special symbol denoting the *absence* of a value. The *support* of a label $l \in \mathcal{L}_V$ is $supp(l) = \{v \in V \mid l(v) \neq \perp\}$. We denote with $\perp_V$ the label of empty support over V. For simplicity, we shall usually write out a label as the set of its non-absent variable valuations. For instance, $< v = 0, u = 1 >_V$ denotes the label over V with support $\{u, v\}$ and which assigns 0 to v and 1 to u. When confusion is not possible, the set V of variables can be omitted from the notation. Also, when confusion is not possible and we need to save space (for instance in large system representations) we shall drop the "$< >$" delimiters.

If $l \in \mathcal{L}_V$ and V' is another set of variables, then the *image* of l through V' is the label $l\mid_{V'} \in \mathcal{L}_{V'}$ that equals l over $V \cap V'$ and equals $\perp$ on $V' \setminus V$.

The labels $l_i \in \mathcal{L}_{V_i}, i = 1, 2$, are *non-contradictory*, denoted $l_1 \bowtie l_2$, if for all $v \in V_1 \cap V_2$ such that $l_i(v) \neq \perp, i = 1, 2$ we have $l_1(v) = l_2(v)$. In this case, we define their:

- *union:* $l_1 \sqcup l_2 \in \mathcal{L}_{V_1 \cup V_2}$, of support $supp(l_1) \cup supp(l_2)$ and which equals l_i over $supp(l_i), i = 1, 2$.
- *intersection:* $l_1 \sqcap l_2 \in \mathcal{L}_{V_1 \cup V_2}$, of support $supp(l_1) \cap supp(l_2)$ and which equals l_i on its support.

The union and intersection operators are associative and commutative. When the labels $l_1, l_2 \in \mathcal{L}_V$ are non-contradictory, we also define their *difference* $l_1 \setminus l_2 \in \mathcal{L}_V$ by $l_1 \setminus l_2(u) = \begin{cases} l_1(u), \text{ if } u \notin supp(l_2) \\ \perp, \text{ otherwise} \end{cases}$.

When the non-contradictory labels $l_i \in \mathcal{L}_{V_i}, i = 1, 2$ are equal over $V_1 \cap V_2$, they are called *synchronizable*. Their union is also called in this case *product* and denoted with $l_1 \otimes l_2$. Note that $l_1 \otimes l_2$ equals l_i on V_i, $i = 1, 2$. The labels $l_i \in \mathcal{L}_{V_i}, i = 1, 2$ are *disjoint* if $supp(l_1) \cap supp(l_2) = \emptyset$.

Assume that v and v' are variables with the same domain (*i.e.* $\mathcal{D}_v = \mathcal{D}_{v'}$). Given $l \in \mathcal{L}_V$, with $v \in V$ and $v' \notin V \setminus \{v\}$, the name change operator associates $l[v/v'] \in \mathcal{L}_{V \setminus \{v\} \cup \{v'\}}$ with:

$$l[v/v'](u) = \begin{cases} l(u), \text{ if } u \neq v' \\ l(v), \text{ if } u = v' \end{cases}$$

We define the "sub-label" partial order relation $\leq$ over $\mathcal{L}_V$: $l_1 \leq l_2$ if $\forall v : (l_1(v) \neq \perp \Rightarrow l_1(v) = l_2(v))$.

2.2. Traces

A *trace* over the set of variables V is a *finite* sequence of labels over V. The set of all traces over V is denoted with $Traces(V) = \mathcal{L}_V^* = \{(l_i)_{0\leq i<n} \mid n \in I\!N \wedge \forall i : l_i \in \mathcal{L}_V\}$. Given a trace $\varphi = (l_i)_{0\leq i<n}$ we denote $length(\varphi) = n$ and $\varphi[i] = l_i$. Note that any label is a trace of length 1. We denote with ϵ any sequence of length 0. In particular ϵ denotes the empty trace, regardless of the variable set.

Any two traces $\varphi_1, \varphi_2 \in Traces(V)$ can be concatenated (by juxtaposition $\varphi_1\varphi_2$). The trace φ_1 is a prefix of φ_2 (written $\varphi_1 \preceq \varphi_2$) if, by definition, $\varphi_2 = \varphi_1\varphi_3$ for some φ_3. The prefix relation is a partial order over traces. The image operator is extended component-wise on traces: $(l_i)_{0\leq i<n} \mid_{V'} = (l_i \mid_{V'})_{0\leq i<n}$. The traces $\varphi_i \in Traces(V_i), i = 1, 2$ are called *synchronizable* if $length(\varphi_1) = length(\varphi_2)$ and if for all j the labels $\varphi_1[j]$ and $\varphi_2[j]$ are synchronizable. In this case, we can define the *product trace* $\varphi_1 \otimes \varphi_2 = (\varphi_1[i] \otimes \varphi_2[i])_{0\leq i<length(\varphi_1)}$. The product operator is associative and commutative. The *support* of a trace φ, denoted $supp(\varphi)$, is the union of the supports of its labels.

2.3. Generalized concurrent transition systems

The *generalized concurrent transition systems* (GCTS) form our (asynchronous) implementation model. GCTSs are step transition systems where steps are syntactic represention of the concurrency between atomic operations (which assign or test a single variable). They generalize the concurrent transition systems of Stark [21], and can be seen as a sub-set of the step transition systems of Mukund [22].

Definition 2.1. (generalized concurrent transition system)
A generalized concurrent transition system (GCTS) is a tuple $\Sigma = (S, \hat{s}, V, \circ\!\!\rightarrow_\Sigma)$, where S is the set of states (not necessarily finite) , $\hat{s} \in S$ is the initial state, V is the finite set of communication variables, and $\circ\!\!\rightarrow_\Sigma \subseteq S \times \mathcal{L}_V \times S$ is a transition relation satisfying:

GCTS1 (void transition): $\forall s \in S : \; s \xrightarrow[\Sigma]{\perp_V} s$.

GCTS2 (prefix closure): If $s \xrightarrow[\Sigma]{l} s'$ and $l' \leq l$, then there exists $s'' \in S$ such that $s \xrightarrow[\Sigma]{l'} s''$ and $s'' \xrightarrow[\Sigma]{l\backslash l'} s'$.

When there is no ambiguity, Σ can be dropped from the transition relation notation.

We shall say that $\varphi = l_1 \ldots l_n \in Traces(V)$ is a trace of the GCTS $\Sigma = (S, \hat{s}, V, \circ\!\!\rightarrow_\Sigma)$ starting in the state $s \in S$ if there exist $s_1, \ldots, s_n \in S$ such that $s \xrightarrow{l_1} s_1 \xrightarrow{l_2} \cdots \xrightarrow{l_n} s_n$. In this case, we also write $s \xRightarrow{\varphi} s_n$. The set of all traces of Σ starting in s is denoted by $Traces_\Sigma(s)$, and the set of all destination states of such traces is: $RSS_s(\Sigma) = \{s' \in S \mid \exists \varphi : \; s \xRightarrow{\varphi} s'\}$. The *reachable state space of* Σ is $RSS(\Sigma) =_{def} RSS_{\hat{s}}(\Sigma)$.

Generalized concurrent transition systems are composed by means of synchronized product. Consider two GCTSs $\Sigma_i = (S_i, \hat{s_i}, V_i, \circ\!\!\rightarrow_{\Sigma_i}), i = 1, 2$, then their product is defined as follows:

$$\Sigma_1 \otimes \Sigma_2 = (S_1 \times S_2, (\hat{s_1}, \hat{s_2}), V_1 \cup V_2, \circ\!\!\rightarrow_{\Sigma_1 \otimes \Sigma_2})$$

where $(s_1, s_2) \overset{l}{\underset{\Sigma_1 \otimes \Sigma_2}{\circ\!\!\longrightarrow}} (s_1', s_2') \Leftrightarrow s_i \overset{l|_{V_i}}{\underset{\Sigma_i}{\circ\!\!\longrightarrow}} s_i'$, $i = 1, 2$. The $\otimes$ operator is well-defined — It preserves the properties GCTS1 and GCTS2. It is also associative and commutative, and:

$$Traces_{\bigotimes_{i=1}^{n} \Sigma_i}((s_i)_{1 \le i \le n}) = \{\bigotimes_{i=1}^{n} \varphi_i \mid \varphi_i \in Traces_{\Sigma_i}(s_i) \text{ pairwise synchronizable}\}$$

The variable name change operator is extended to GCTSs. If $\Sigma = (S, \hat{s}, V, \circ\!\!\rightarrow_{\Sigma})$, $v \in V$, $v' \notin V \setminus \{v\}$, and $\mathcal{D}_v = \mathcal{D}_{v'}$, then:

$$\Sigma[v/v'] = (S, \hat{s}, V \setminus \{v\} \cup \{v'\}, \{\ s \overset{l[v/v']}{\underset{\Sigma[v/v']}{\circ\!\!\longrightarrow}} s' \ \mid \ s \overset{l}{\underset{\Sigma}{\circ\!\!\longrightarrow}} s'\ \})$$

2.4. I/O causality. Channels and clocks

In practice, communications between the different components of a system are directed. One component emits a value on a channel, and another reads it. To take this into account, we use *directed communication channels* that are pairs of *directed variables*. We emit a value on a channel c by assigning the variable $!c$, and we receive a value by reading the variable $?c$. The variables $!c$ and $?c$ have the same domain, denoted with $\mathcal{D}_c$. We denote with $\mathcal{C}(V) = \{c \mid !c \in V \text{ or } ?c \in V\}$ the set of channels associated with a set of variables V. To simplify the model, we assume that every channel connects at most one emitter with at most one receiver, meaning that $!c$ is variable of at most one component in the system, the same holding for $?c$ (a simple renaming technique allows the use of multicast, but we shall not cover the subject here). We further assume that the only variables that are not directed are the *clocks* of the synchronous components. A clock is a variable whose domain is $\mathcal{D}_{clk} = \{\top\}$ ($\top$ stands for the "clock tick"). Given a set V of variables we shall denote with $Clocks(V)$ the subset of clock variables, and with $Directed(V) = V \setminus Clocks(V)$ the subset of directed variables. To simplify the notations, we abbreviate the clock tick valuation $\tau = \top$ with τ (for any clock variable τ).

Definition 2.2. (I/O transition system)
We say that a GCTS is an I/O transition system when all its variables are either directed or clocks. ***From now on, this paper only considers I/O transition systems.***

To reason about desynchronization properties, we shall need the following function, which removes the clock synchronization barriers, so that only messages (and not absence) are visible, along with message ordering on each channel: $\delta : (\mathcal{D}_v^{\perp})^* \to \mathcal{D}_v^*$, defined by $\delta(\epsilon) = \epsilon$ and:

$$\delta(v\varphi) = \begin{cases} v\delta(\varphi), \text{ if } v \neq \perp \\ \delta(\varphi), \text{ otherwise} \end{cases}$$

Using this notation, we extend the relation $\le$ (first defined on labels) to a preorder over $Traces(V)$: Given $\varphi_1, \varphi_2 \in Traces(V)$, we write $\varphi_1 \le \varphi_2$ whenever we have $\delta(\varphi_1 \mid_{\{v\}}) \preceq \delta(\varphi_2 \mid_{\{v\}})$ for all $v \in Directed(V)$. If $\varphi_1 \le \varphi_2$ and $\varphi_2 \le \varphi_1$, then we say that φ_1 and φ_2 are *asynchronously equivalent*, denoted $\varphi_1 \sim \varphi_2$. When for all $v \in Directed(V)$ we have $\delta(\varphi_1 \mid_{\{v\}}) \preceq \delta(\varphi_2 \mid_{\{v\}})$ or $\delta(\varphi_2 \mid_{\{v\}}) \preceq \delta(\varphi_1 \mid_{\{v\}})$, then we say that φ_1 and φ_2 are *asynchronously non-contradictory*, and write $\varphi_1 \bowtie \varphi_2$. Note that $\bowtie$ extends to traces the non-contradiction relation over labels. Moreover, we can extend the label

difference operator to non-contradictory traces by defining the asynchronous difference of traces $\varphi_1 \setminus \varphi_2$ by induction:

$$\begin{cases} \varphi_1 \setminus (l\varphi_2) = (\varphi_1 \setminus l) \setminus \varphi_2 \\ (l_1\varphi_1) \setminus l_2 = (l_1 \setminus l_2)(\varphi_1 \setminus (l_2 \setminus l_1)) \end{cases}$$

If φ is a trace of an I/O transition system, then we denote with $\mid \varphi \mid$ the number of assignments of non-clock variables contained in φ.

2.5. Synchronous transition systems

Our synchronous transition systems represent causal synchronous specifications or, equivalently, implementations of synchronous specifications where the global clock is preserved by some communication infrastructure by means of added signalization[3]. A synchronous transition system is an I/O transition system with a single clock variable, and satisfying the synchronous hypothesis and a stuttering-invariance property (which is necessary if we want to derive GALS implementations).

Definition 2.3. (synchronous transition system)
A microstep synchronous transition system (μSTS) is a tuple $\Sigma = (S, \hat{s}, V, \tau, \circ\!\!\rightarrow)$ where all the variables of V are directed, where τ is a clock variable (the clock of the component), and where $(S, \hat{s}, V \cup \{\tau\}, \circ\!\!\rightarrow)$ is a GCTS satisfying:

μSTS1 (clock transitions): if $s \overset{l}{\circ\!\!\rightarrow} s'$ and $l(\tau) \neq \perp$ then $l \mid_V = \perp_V$.

μSTS2 (stuttering-invariance): $\hat{s} \overset{<\tau>}{\circ\!\!\rightarrow} \hat{s}$ and ($s \overset{<\tau>}{\circ\!\!\rightarrow} s' \Rightarrow s' \overset{<\tau>}{\circ\!\!\rightarrow} s'$)

μSTS3 (single assignment): two assignments of a same variable must be separated by a clock transition. More exactly, if $s_0 \overset{l_1}{\circ\!\!\rightarrow} s_1 \overset{l_2}{\circ\!\!\rightarrow} \cdots \overset{l_n}{\circ\!\!\rightarrow} s_n$ and $\forall i : l_i \neq \tau$, then $l_1, \ldots, l_n$ are pairwise disjoint.

Note that axiom μSTS1 identifies the *clock transitions* – with label $< \tau >$ – which are the only transitions where the clock variable is present. Such transitions separate synchronous reactions during which a variable cannot be assigned more than once (cf. axiom μSTS3). A state which is destination of a clock transition is called *synchronizing state*. Given a trace φ of a synchronous system, we can decompose it into reactions $\varphi = Step_0(\varphi) < \tau > Step_1(\varphi) < \tau > \ldots$ where each reaction $Step_i(\varphi)$ contains no clock transition. As the transitions of each $Step_i(\varphi)$ are disjoint, we can denote with $< Step_i(\varphi) >$ the union of all its labels. We shall say that a trace φ is *complete* if it ends with a $< \tau >$ transition. We say that a μSTS is *non-blocking* if from any reachable state there is a path towards a stuttering state. Note that in a non-blocking μSTS any trace can be completed. Blocking systems are considered incorrect.

An isomorphism λ between two GCTSs $\Sigma_i = (S_i, \hat{s_i}, V_i, \circ\!\!\rightarrow_{\Sigma_i}), i = 1, 2$ consists of two bijections $\lambda^S : S_1 \rightarrow S_2$ and $\lambda^V : V_1 \rightarrow V_2$ having the properties: (i) $\forall v : \mathcal{D}_v = \mathcal{D}_{\lambda^V(v)}$, (ii) $\lambda^S(\hat{s_1}) = \hat{s_2}$, and (iii) $s \overset{l}{\underset{\Sigma_1}{\circ\!\!\rightarrow}} s' \Leftrightarrow \lambda^S(s) \overset{\lambda(l)}{\underset{\Sigma_2}{\circ\!\!\rightarrow}} \lambda^S(s')$, where $\lambda(l)$ denotes the label obtained from l by renaming v with $\lambda^V(v)$ for all $v \in V_1$. If Σ_1 and Σ_2 are I/O transition systems, we say that λ is an isomorphism of I/O

[3] Such as in the *Time-Triggered Architectures* of Kopetz[4].

transition systems if λ^V maps read variables onto read variables, write variables onto write variables, and clocks onto clocks. If Σ_1 and Σ_2 are μSTSs, then λ is an isomorphism of μSTSs if it is an isomorphism of I/O transition systems.

2.6. Synchronous and asynchronous composition

As earlier mentioned, we simplify the model by only allowing point-to-point communication, and we enforce this rule by syntactic means. However, broadcast can be simulated by replicating and renaming variables.

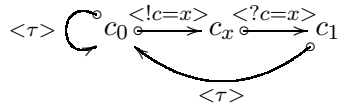
Definition 2.4. (composable transition systems)
We say that the I/O transition systems $\Sigma_i, i = \overline{1,n}$ are *composable* if their variable sets are mutually disjoint.

Note that the definition requires not only point-to-point communications (no directed variable is shared by two or more systems), but also the non-overlapping of clock sets (which is natural). Also note that a system can have both $!c$ and $?c$ as variables, thus allowing the representation of systems obtained by composition.

The composition of synchronous and asynchronous systems is defined by means of synchronized product, using FIFO models to represent communication through synchronous and asynchronous channels. To represent synchronous communication, we use 1-place synchronous FIFO models (which are μSTSs themselves). The FIFO model associated with a channel c is:

$$SFIFO(c,\tau) = \\ (\{c_0, c_1\} \cup \bigcup_{x \in \mathcal{D}_c} \{c_x\},\ c_0,\ \bigcup_{x \in \mathcal{D}_c} \{!c = x, ?c = x\},\ \tau,\ \circ\!\!\rightarrow_S)$$

where the transition relation is defined by:



Note that modeling multicast communication (a feature that will not be addressed in this paper), can simply be done by renaming channel read variables in a component-wise fashion, and then modifying the FIFO model to allow the concurrent read of the value from different sites.

Asynchronous communication involves infinite asynchronous FIFO models (which are not μSTSs):

$$AFIFO(c) = (\mathcal{D}_c^*,\ \epsilon,\ \bigcup_{x \in \mathcal{D}_c} \{!c = x, ?c = x\},\ \circ\!\!\rightarrow_A)$$

where the transition relation contains all the transitions of the form:

$$x_1 \ldots x_n \xrightarrow{<!c=x_{n+1}>} x_1 \ldots x_n x_{n+1} \xrightarrow{<?c=x_1>} x_2 \ldots x_{n+1}$$

Definition 2.5. (synchronous composition of μSTSs)
Let $\Sigma_i = (S_i, \hat{s_i}, V_i, \tau_i, \circ\!\!\rightarrow_{\Sigma_i}), i = 1, 2$ be composable μSTSs and let τ be a clock variable. Then, the synchronous composition of Σ_1 and Σ_2 over the base clock τ is:

$$\Sigma_1 \ |^{\tau} \ \Sigma_2 = \Sigma_1[\tau_1/\tau] \otimes \Sigma_2[\tau_2/\tau] \otimes \bigotimes_{c \in \mathcal{C}(V_1) \cap \mathcal{C}(V_2)} SFIFO(c, \tau)$$

Lemma 2.1. (Properties of the synchronous composition)
The synchronous composition of the μSTSs Σ_1 and Σ_2 over the base clock τ is a μSTS of clock τ. The result of the synchronous composition is unique upto renaming of the base clock, so that we can omit the base clock τ from the notation. Moreover, the operator $|$ is associative and commutative, modulo isomorphism.

In addition, note that synchronizing states of $|_{i=1}^{n}\Sigma_i$ have void communication lines (all synchronous FIFO models are in their unique synchronizing state).

Definition 2.6. (asynchronous composition of I/O systems)
Let $\Sigma_i = (S_i, \hat{s_i}, V_i, \circ\!\!\rightarrow_{\Sigma_i}), i = 1, 2$ be composable I/O transition systems. Then, the asynchronous composition of Σ_1 and Σ_2 is:

$$\Sigma_1 \ \| \ \Sigma_2 = \Sigma_1 \otimes \Sigma_2 \otimes \bigotimes_{c \in \mathcal{C}(V_1) \cap \mathcal{C}(V_2)} AFIFO(c)$$

Lemma 2.2. (asynchronous composition properties)
The asynchronous composition of I/O transition systems results in another I/O transition system. The $\|$ operator is associative and commutative. The asynchronous composition of two μSTSs is not a μSTS.

Proof:(lemmas 2.1 and 2.2) The operator $\otimes$ is associative and commutative, which implies the associativity and commutativity of the synchronous and asynchronous composition operators. The isomorphism of $\Sigma_1 \ |^{\tau_1} \ \Sigma_2$ and $\Sigma_1 \ |^{\tau_2} \ \Sigma_2$ is given by the renaming of the clock variable. □

2.7. Product states and product traces

Note that the state of a synchronous or asynchronous product of I/O systems is not only given by the state of the components, but also by the state of its communication channels. Indeed, given the composable μSTSs $\Sigma_i, i = \overline{1,n}$, connected through the channels $c_i, i = \overline{1,m}$, the state of $|_{i=1}^{n}\Sigma_i$ is $((s_i)_{i=\overline{1,n}}, (c_i^s)_{i=\overline{1,m}})$, and the state of $\|_{i=1}^{n}\Sigma_i$ is $((s_i)_{i=\overline{1,n}}, (c_i^a)_{i=\overline{1,m}})$, where c_i^s denotes states of $SFIFO(c_i, \tau)$ and c_i^a denotes states of $AFIFO(c_i)$.

Nevertheless, for space reasons, we shall consider in this article only examples where the tuple $(s_i)_{i=\overline{1,n}}$ unambiguously identifies the state of the product. Thus, we can use the component state tuple alone to label states.

As should be expected, the synchronous composition binds tighter than the asynchronous one. Indeed, given the composable μSTSs $\Sigma_i = (S_i, \hat{s_i}, V_i, \tau_i, \circ\!\!\rightarrow_{\Sigma_i}), i = \overline{1,n}$, we can map the state space of $|_{i=1}^{n}\Sigma_i$ onto the state space of $\|_{i=1}^{n}\Sigma_i$:

$$\iota : RSS(|_{i=1}^{n}\Sigma_i) \hookrightarrow RSS(\|_{i=1}^{n}\Sigma_i)$$

by mapping for each communication channel c the state of $SFIFO(c,\tau)$ onto the state of $AFIFO(c)$ using: $c_0 \mapsto \epsilon$, $c_1 \mapsto \epsilon$, and $\forall x \in \mathcal{D}_c : c_x \mapsto x$. Similarly, we can define for any $s_i \in S_i, i = \overline{1,n}$ the injective "inclusion morphism" that maps traces of the synchronous product into traces of the asynchronous product:

$$\iota : Traces_{|_{i=1}^{n}\Sigma_i}(s) \hookrightarrow Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$$

defined inductively by $\iota(\epsilon) = \epsilon$, by $\iota(\varphi_1\varphi_2) = \iota(\varphi_1)\iota(\varphi_2)$, and (for labels) by:

$$\iota(l) = \begin{cases} l\,|_{\bigcup_{i=1}^{n} V_i \cup \{\tau_i | i=\overline{1,n}\}}, \text{ if } l \neq <\tau> \\ <\tau_1, \ldots, \tau_n>, \text{ if } l = <\tau> \end{cases}$$

where τ is the base clock of the synchronous composition. With these notations we have:

$$s \xrightarrow[|_{i=1}^{n}\Sigma_i]{\varphi} s' \;\Rightarrow\; \iota(s) \xrightarrow[\|_{i=1}^{n}\Sigma_i]{\iota(\varphi)} \iota(s')$$

2.8. Projection operators. Traces of a GALS system

The operator $\pi_i^{\sigma}()$ projects a state or transition label of the synchronous product $|_{i=1}^{n}\Sigma_i$ onto the corresponding state or transition label of Σ_i. Similarly, $\pi_i^{\alpha}()$ projects states and transitions of the asynchronous product $\|_{i=1}^{n}\Sigma_i$ onto states and transitions of Σ_i. The definition of $\pi_i^{\sigma}()$ and $\pi_i^{\alpha}()$ is trivial, with the exception of $\pi_i^{\sigma}()$ over transition labels, which involves the renaming of the common clock τ to the local clock τ_i.

Note that, while not constrained by global clock synchronization, the traces of $\|_{i=1}^{n}\Sigma_i$ still satisfy a FIFO consistency property that requires that a value is read from a channel only after being emitted. The following definition formalizes this for traces starting with void channels (from the initial state). Intuitively, we require that in any trace of the composed system the sequence of values read from a channel is a prefix of the sequence of values that are written. Moreover, we require that a write operation occurs before the corresponding read operation.

Definition 2.7. (FIFO consistency)
Let $\Sigma_i = (S_i, \hat{s}_i, V_i, \tau_i, \circ\!\!\rightarrow_i)$, $1 \leq i \leq n$ be composable μSTSs, and let φ be some trace in $Traces(\bigcup_{i=1}^{n}(V_i \cup \{\tau_i\}))$. We say that φ is FIFO consistent if for each channel c shared between two components we have $\delta(\varphi\,|_{\{?c\}}) \preceq \delta(\varphi\,|_{\{!c\}})$ and the rank of $\delta(\varphi\,|_{\{?c\}})[j]$ in φ is greater than the rank of $\delta(\varphi\,|_{\{!c\}})[j]$ in φ for all $j \leq length(\delta(\varphi\,|_{\{?c\}}))$.

We can now characterize the traces of $\|_{i=1}^{n}\Sigma_i$:

Lemma 2.3. (GALS traces)
Let $\Sigma_1, \ldots, \Sigma_n$ be composable μSTSs and let $s \in RSS(|_{i=1}^{n}\Sigma_i)$ be a synchronizing state. Then, $\varphi \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ if and only if $\forall i : \pi_i^{\alpha}(\varphi) \in Traces_{\Sigma_i}(\pi_i^{\sigma}(s))$ and φ is FIFO consistent.

Proof: The direct implication is obvious according to the definitions of $\otimes$ and $\pi_i^{\alpha}()$. Conversely, consider φ a consistent trace such that $\forall i : \varphi_i = \pi_i^{\alpha}(\varphi) \in Traces_{\Sigma_i}(\pi_i^{\sigma}(s))$. Then, the consistency of φ allows us to prove, by induction over $length(\varphi)$, that the interleaving of the φ_i's into φ is possible under the composition contraints imposed by the asynchronous FIFOs that take part in $\|_{i=1}^{n}\Sigma_i$. This implies $\varphi \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$. □

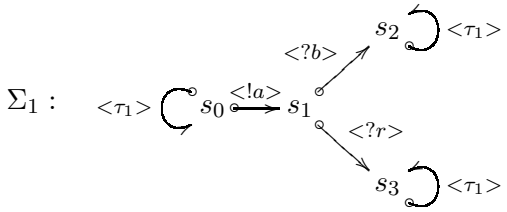
As a corollary, if we are given $\varphi_i \in Traces_{\Sigma_i}(\pi_i^\sigma(s))$, $1 \leq i \leq n$, and if we can order their non-clock transitions in a FIFO-consistent way, then there exists $\varphi \in Traces_{\|_{i=1}^n \Sigma_i}(\iota(s))$ such that $\pi_i^\alpha(\varphi)$ and φ_i are identical upto void transitions for all $1 \leq i \leq n$.

3. Modelling and correctness of GALS implementations

This section starts by illustrating our definitions with a number of small, but intuitive examples. Based on this intuition, we define in section 3.2 the formal correctness criterion. Section 3.3 explains why our model is useful in solving the GALS implementation problem.

3.1. Examples

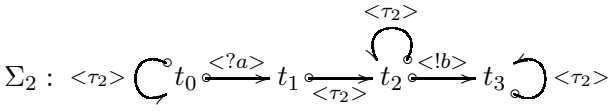
The following μSTS represents a system that emits a message on channel a and then awaits for one message from either channel b or r (*e.g.* for whichever comes first). Data is uninterpreted (not important), therefore not represented. The clock of the system is τ_1, and we shall assume that the directed variable set of Σ_1 is $\{!a, ?b, ?r\}$:



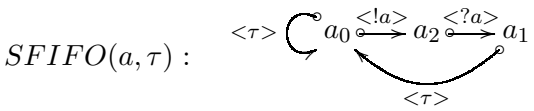
In a more classical macrostep framework, like that of [11], this system would be represented by:

The correspondence between the microstep and macrostep representations of a system is straightforward: The states of the macrostep system are the synchronizing states of the microstep one. The macrostep transitions correspond to full reactions connecting synchronizing states (after forgetting the direction of the signals and the causality between successive labels).

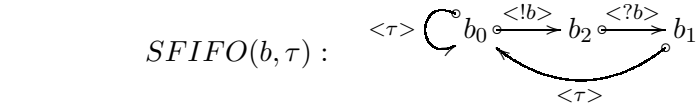
We compose Σ_1 with the μSTS Σ_2, which has the clock τ_2 and the directed variable set $\{?a, !b\}$:



The synchronous composition $\Sigma_1 \mid \Sigma_2$ is done using two synchronous FIFOs, corresponding to the variables/channels a and b:

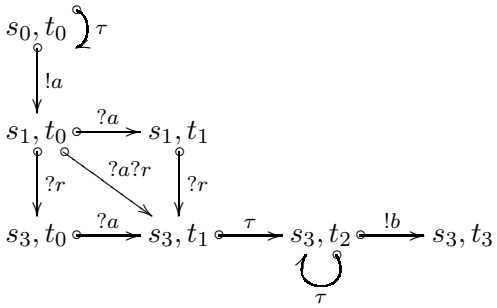


$SFIFO(b,\tau)$:



In this example, data is uninterpreted, only write/read causality and clock synchronization is considered. The composed synchronous system is (we simplified for space reasons the label notations, as explained in section 2.1):

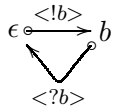
$\Sigma_1 \mid \Sigma_2$:



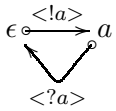
Note that we simplified the notation by not representing the state of the two FIFOs (the initial state having void FIFOs, the status of the FIFOs is fully determined in each state). However, note that the composed system is blocked in state (s_3, t_3) because $SFIFO(b,\tau)$ cannot take a clock transition (data has been written on it, but not read). The system $\Sigma_1 \mid \Sigma_2$ is blocking, thus incorrect.

The asynchronous composition $\Sigma_1 \parallel \Sigma_2$ is done using the two asynchronous FIFOs, figured below:

$AFIFO(b)$:

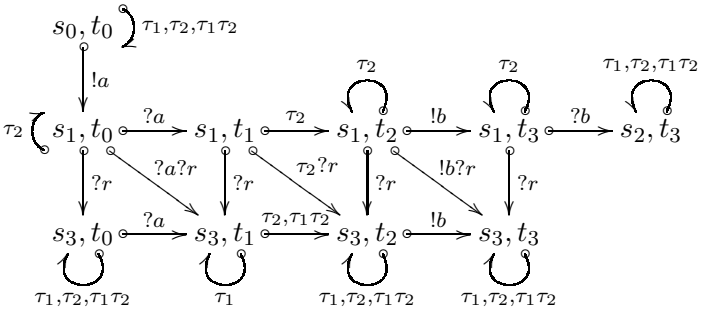


$AFIFO(a)$:



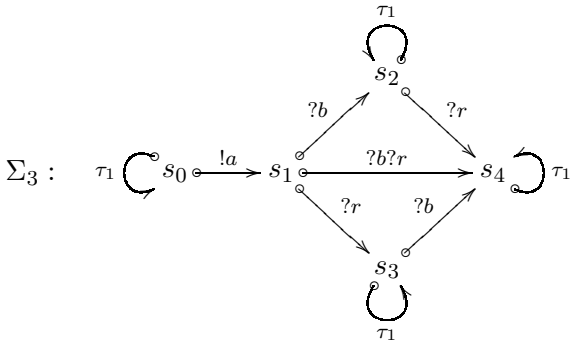
Recall that in the general case the asynchronous FIFO models are infinite. However, Σ_1 and Σ_2 can emit at most one message on any of the two channels, so our choice does not affect the result of the composition:

$\Sigma_1 \parallel \Sigma_2$:

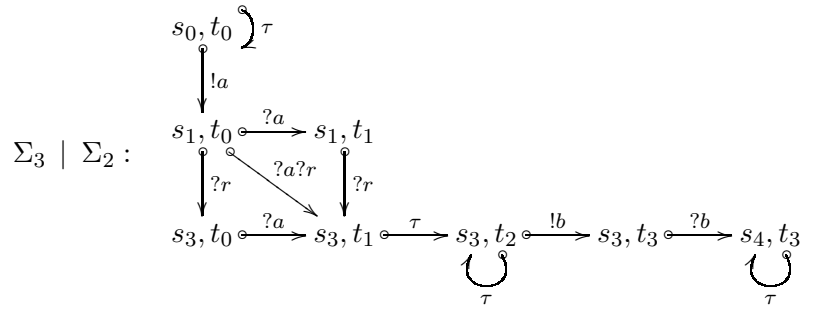


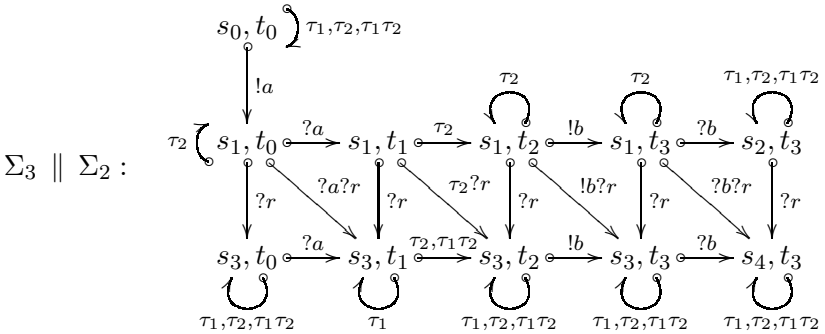
It is essential to note that $\Sigma_1 \parallel \Sigma_2$ has traces, like $< !a >< ?a >< \tau_2 >< !b >< ?b >$, that are not asynchronously equivalent to any of the synchronous traces of $\Sigma_1 \mid \Sigma_2$. Such traces are not covered by the verification done on the synchronous model, meaning that the GALS implementation does not preserve the semantics of the specification.

It is also important to note that requiring a one-to-one correspondence between synchronous and asynchronous traces is not a good idea, because for large classes of systems it can be highly inefficient. Consider, for instance, the following system:



and its synchronous and asynchronous composition with Σ_2:





As expected, the synchronous composition binds tighter than the asynchronous one, but for any trace of $\Sigma_3 \parallel \Sigma_2$ going from (s_0, t_0) to (s_4, t_3) we can find an asynchronously equivalent trace in $\Sigma_3 \mid \Sigma_2$. Such a GALS implementation is obviously correct, because it does not introduce new behaviors. Exploiting the concurrency between different computations (as we do here) to allow the systems to evolve at different rates is a desirable feature because it minimizes communication and consumption. The difference between Σ_1 and Σ_3 is that in Σ_3 the transitions $< ?b >$ and $< ?r >$ are concurrent in state s_1, while in Σ_1 there is a non-deterministic choice between them (meaning that if messages come on both channels, only one will be read, in an unpredictable fashion).

3.2. Formal correctness criterion

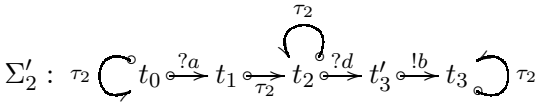
We already presented, in section 1.1, the intuition covering the notion of correctness of a GALS implementation with respect to its microstep synchronous specification. We give here the corresponding formal correctness criterion:

Criterion 1. (correct desynchronization)
Let $\Sigma_i, i = \overline{1,n}$ be composable μSTSs. Then, we shall say that the GALS implementation $\|_{i=1}^{n}\Sigma_i$ is correct *w.r.t.* the synchronous specification $|_{i=1}^{n}\Sigma_i$ if for all synchronizing state s of $|_{i=1}^{n}\Sigma_i$ and for all trace $\varphi \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ there exist $\tilde{\varphi} \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ and $\overline{\varphi} \in Traces_{|_{i=1}^{n}\Sigma_i}(s)$ such that $\varphi \preceq \tilde{\varphi}$ and $\tilde{\varphi} \sim \iota(\overline{\varphi})$.

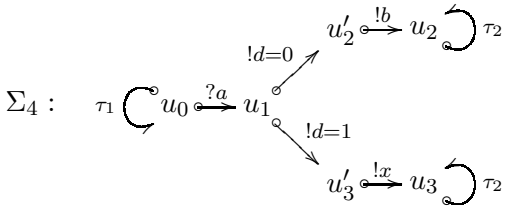
In other words, the GALS implementation is correct if any of its traces can be completed with a finite number of transitions to a trace that is asynchronously equivalent to a complete synchronous trace.

Our criterion is akin to previous correctness criteria [9, 11] defined in a macrostep setting. Most important, criterion 1 allows us to exploit (like in $\Sigma_3 \parallel \Sigma_2$) the concurrency of the synchronous specification to support GALS implementations that are weakly synchronized, yet correct. Important differences exist, though, as our criterion is formulated in a micro-step operational framework that simplifies, as we shall see in section 4.4, the definition of sufficient conditions for correctness.

As explained in the introduction, our purpose is now to find sufficient conditions for correctness (in the formal sense of criterion 1) that cover large classes of implementations. We do not cover here the synthesis problem of transforming given systems to satisfy the correctness criterion. However, we use two examples to give the intuition of future synthesis techniques: First, to correct the composition of Σ_1 with Σ_2, we can simply prevent Σ_2 from firing the transition labeled $< !b >$ by guarding it with a condition that is never fulfilled:

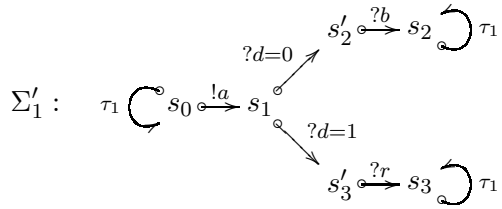


More interesting is the case where we compose Σ_1 with a process Σ_4 that non-deterministically chooses between emitting b or doing something else. In this case, the solution is to signal the non-deterministic choice to Σ_1, so that it can adapt its behavior:



Here, we assumed that the non-deterministic choice between $< !b >$ and $< !x >$ is an essential feature of the specification Σ_4, which must be preserved. To make the composition correct we need to make this choice visible from its asynchronous environment, under the form of a choice over the value of a new channel, named d. Then, we can modify Σ_1 into Σ_1', which uses this signal to decide which message to

wait for.



3.3. Modeling issues

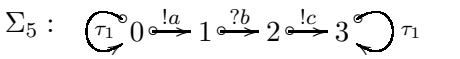
The I/O transitions systems can be viewed either as microstep specifications, or as asynchronous implementation models. A sub-class of I/O transition systems satisfy the synchronous hypothesis – they have a single clock variable, which determines clock transitions, and no variable is assigned twice between successive clock transitions. Thus, they can be seen as microstep synchronous specifications. The only hypothesis that departs from the classical synchronous model is stuttering-invariance. However, we see stuttering-invariance as a prerequisite for the efficient multi-rate GALS deployment.

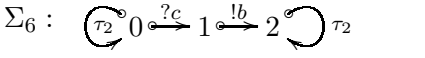
If we compare our model to macro-step models like those of [9, 11], every macrostep specification (automaton) has (at least) a microstep implementation. Like many macro-step models, our formalism does not explicitly represent the reaction to signal absence. This does not influence the expressivity of the model, as reaction to signal absence can be represented using non-deterministic choice. The composition through point-to-point links is not an essential restriction, as it is easy to define FIFO models that cover multicast.

The synchronous and asynchronous composition operators reflect the assumption that an emitted signal must not be left unread by the receiver. This hypothesis reflects in an operational fashion the rendez-vous-like synchronized product composition from macro-step formalisms.

Composing the μSTSs $\Sigma_i, i = \overline{1,n}$ using the $\|$ operator intuitively corresponds to implementing $|_{i=1}^{n}\Sigma_i$ as a GALS system where all the communication lines have been replaced with asynchronous FIFOs. The components are still clocked, but individual clocks are independent, and the components are only synchronized by the FIFO causality rules. In the GALS implementation the clock of one component can be triggered concurrently with another clock or an assignment of another component. The GALS implementation can function in a multi-rate fashion, as no constraint relates the occurrence of clock transition in different components.

Compared to classical macro-step approaches, our model brings a level of detail which is essential in deciding the correctness of actual implementations. Composing Σ_5 and Σ_6 results in a blocking system:





However, this problem cannot be observed in macro-step settings, where the system does not block and can even fire the transition of label abc. Indeed, the microstep model is better suited for analysis akin to causality checks performed in synchronous languages like Esterel. In fact, we shall see in section 4.4, that non-blocking correctness and semantics preservation are truly related.

4. Correct desynchronization criteria

Following the goal fixed in the introduction, we now define criteria that characterize a large class of synchronous components for which small, simple wrappers produce deterministic, efficient, and semantics-preserving GALS implementations.

4.1. Microstep weak endochrony

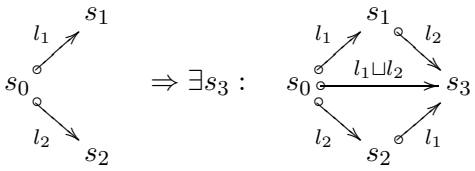
Microstep weak endochrony (or, simply, weak endochrony) is the property guaranteeing that a given synchronous component (μSTS) knows how to read its inputs, so that no asynchronous wrapper is needed. Weak endochrony requires that all internal choice of the component is visible as a choice over the value (and not presence/absence status) of a directed variable (either input or output). Thus, the behavior of the system becomes predictable in *any asynchronous environment*, because choices can be observed.

With this requirement, the implementation space delimited by weak endochrony is nonetheless very large: Concurrent behaviors are not affected by the previous rule, so that independent system parts can evolve at different speeds. Weak endochrony does not require I/O determinism. Instead, a weakly endochronous component must inform the environment about non-deterministic decisions (the variable used to do so behaves like an oracle that is visible from outside).

Definition 4.1. (weak endochrony)
We say that the μSTS $\Sigma = (S, \hat{s}, V, \tau, \circ\!\!\rightarrow)$ is weakly endochronous if it satisfies the following axioms:

μWE1 (determinism): $s \xrightarrow{l} s_i, i = 1,2 \Rightarrow s_1 = s_2$ (from now on, we shall denote with $s.\varphi$ the unique state of Σ having the property $s \xrightarrow{\varphi} s.l$, and the notation is extended to traces).

μWE2 (independence): if the labels l_1 and l_2 are disjoint and if $l_1, l_2 \neq \tau$, then:



μWE3 (clock properties): assume that $s_0 \xrightarrow{<\tau>} s_1$ and $\varphi \in Traces_\Sigma(s_0)$ with $\tau \notin supp(\varphi)$. Then:

1. $\varphi \in Traces_\Sigma(s_1)$
2. if $\varphi < \tau > \in Traces_\Sigma(s_0)$, then $\varphi < \tau > \in Traces_\Sigma(s_1)$ and $s_0.\varphi < \tau > = s_1.\varphi < \tau >$
3. if $\varphi\psi < \tau > \in Traces_\Sigma(s_1)$, then there exists $\psi' \leq \psi$ such that $\varphi\psi' < \tau > \in Traces_\Sigma(s_0)$.
4. if $\varphi < \tau >, \theta < \tau > \in Traces_\Sigma(s_0)$ and $\varphi \bowtie \theta$, then $\varphi(\theta \setminus \varphi) < \tau > \in Traces_\Sigma(s_0)$

μWE4 (choice): if $\varphi_i < v = x_i > \in Traces_\Sigma(s)$, $i = 1,2$ and $\varphi_1 \bowtie \varphi_2$, then $\varphi_1 < v = x_2 > \in Traces_\Sigma(s)$.

Similar in intuition and in function to its macrostep counterpart [11], weak endochrony is nevertheless specific to our more concrete causal, microstep framework. Thus, while choice can only occur at the level of atomic variable assignments, concurrency (more precisely confluence) must also deal with full reactions and clock transitions (through axioms μWE2 and μWE3, and the consequences of lemma 4.2). Axiom μWE4 insures that a choice between two concurrent execution paths does not hide a "real" choice between non-concurrent assignments.

Lemma 4.1. (independence)
Let $\Sigma = (S, \hat{s}, V, \tau, \circ\!\!\rightarrow)$ be a weakly endochronous μSTS, let $s \in S$, and let $\varphi_1, \varphi_2 \in Traces_\Sigma(s)$ with $supp(\varphi_1) \cap supp(\varphi_2) \subseteq \{\tau\}$. Then:

1. If $\tau \notin supp(\varphi_i), i = 1, 2$, then $s.\varphi_1\varphi_2$ and $s.\varphi_2\varphi_1$ are defined and equal.

2. If φ_i complete, $i = 1, 2$, then $s.\varphi_1\varphi_2$ and $s.\varphi_2\varphi_1$ are defined and equal.

Proof:
part 1: From $supp(\varphi_1) \cap supp(\varphi_2) \subseteq \{\tau\}$ and $\tau \notin supp(\varphi_i), i = 1, 2$, we obtain $supp(\varphi_1) \cap supp(\varphi_2) = \emptyset$. Then the labels $\varphi_1[k]$ and $\varphi_2[l]$ are disjoint, for all k and l. Based on this remark, the result is easily obtained by induction over $length(\varphi_1) + length(\varphi_2)$, the induction step using axiom μWE2.

part 2: We shall give here the proof for the case where φ_1 and φ_2 comprise each one step. This will prove that independent steps commute, and this result can then be easily applied to prove that any two complete traces commute. Assume then $\varphi_1 = Step_0(\varphi_1) < \tau >$ and $\varphi_2 = Step_0(\varphi_2) < \tau >$. Let $s' = s.Step_0(\varphi_1)$.

According to the first part of this lemma, $s.Step_0(\varphi_1)Step_0(\varphi_2)$ and $s.Step_0(\varphi_2)Step_0(\varphi_1)$ exist and are equal.

By applying axiom μWE3.4, $s.Step_0(\varphi_1)Step_0(\varphi_2) < \tau >$ exists. Then, by applying axiom μWE3.2 in s', we obtain that $s'.< \tau > Step_0(\varphi_2) < \tau >$ exists and is equal to $s'.Step_0(\varphi_2) < \tau >$. By using the definition of s', this implies

$$s.Step_0(\varphi_1) < \tau > Step_0(\varphi_2) < \tau > = s.Step_0(\varphi_1)Step_0(\varphi_2) < \tau >$$

Similarly:

$$s.Step_0(\varphi_2) < \tau > Step_0(\varphi_1) < \tau > = s.Step_0(\varphi_2)Step_0(\varphi_1) < \tau >$$

Given that the second terms of the two equalities are equal, the proof is completed. □

Lemma 4.2. (confluence)
Let $\Sigma = (S, \hat{s}, V, \tau, \circ\!\!\rightarrow)$ be a weakly endochronous μSTS, let $s \in S$, and let $\varphi_i \in Traces_\Sigma(s), i = 1, 2$ such that $\varphi_1 \bowtie \varphi_2$. Then:

1. If $\tau \notin supp(\varphi_i), i = 1, 2$, then $s.\varphi_1(\varphi_2 \setminus \varphi_1)$ and $s.\varphi_2(\varphi_1 \setminus \varphi_2)$ are defined and equal.

2. If φ_i complete, $i = 1, 2$, then $s.\varphi_1(\varphi_2 \setminus \varphi_1)$ and $s.\varphi_2(\varphi_1 \setminus \varphi_2)$ are defined and equal.

Proof:
part 1:
Case a: When each of the two traces are reduced to one label $\varphi_1 = l_1$, $\varphi_2 = l_2$. From $\varphi_1 \bowtie \varphi_2$ we have $l_1 \bowtie l_2$. Then, from axiom GCTS2, we can decompose l_2 into $l_2 \setminus l_1$ and $l_2 \sqcap l_1$. By applying axiom μWE2 to $l_2 \setminus l_1$ and l_1 in state s, we obtain that $s.l_1(l_2 \setminus l_1)$ and $s.(l_2 \sqcup l_1)$ exist and are equal. Similarly, $s.l_2(l_1 \setminus l_2)$ exists, and is equal to $s.(l_2 \sqcup l_1)$, which implies the needed result.
Case b: when only φ_2 *is reduced to a single transition*. We deduce that the desired result holds by induction over $length(\varphi_1)$, and by applying case (a).
Case c: the general case. Induction over $length(\varphi_2)$ allows us to prove the first point of the lemma in the general case.

part 2: We shall give here the proof for the case where φ_1 and φ_2 comprise one step each. This will prove that non-contradictory steps can be merged into confluent derivations. The general case is proved by iterating this result. Assume then $\varphi_1 = Step_0(\varphi_1) < \tau >$ and $\varphi_2 = Step_0(\varphi_2) < \tau >$. Let $s' = s.Step_0(\varphi_1)$.

Using the first point of the lemma, $s.Step_0(\varphi_1)(Step_0(\varphi_2) \setminus Step_0(\varphi_1))$ and $s.Step_0(\varphi_2)(Step_0(\varphi_1) \setminus Step_0(\varphi_2))$ exist and are equal. From axiom μWE3.4, $s.Step_0(\varphi_1)(Step_0(\varphi_2) \setminus Step_0(\varphi_1)) < \tau >$ exists, and then, by applying axiom μWE3.2 in state s' we obtain that

$$s.Step_0(\varphi_1) < \tau > (Step_0(\varphi_2) \setminus Step_0(\varphi_1)) < \tau > = s.Step_0(\varphi_1)(Step_0(\varphi_2) \setminus Step_0(\varphi_1)) < \tau >$$

Similarly,

$$s.Step_0(\varphi_2) < \tau > (Step_0(\varphi_1) \setminus Step_0(\varphi_2)) < \tau > = s.Step_0(\varphi_2)(Step_0(\varphi_1) \setminus Step_0(\varphi_2)) < \tau >$$

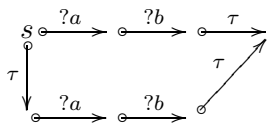
which implies $s.\varphi_1(\varphi_2 \setminus \varphi_1) = s.\varphi_2(\varphi_1 \setminus \varphi_2)$. □

Note that the proofs of lemma 4.1(1) and lemma 4.2(1) are only based on the axioms μWE3 and μWE4.

Lemma 4.3. (completion)
Let $\Sigma = (S, \hat{s}, V, \tau, T)$ be a weakly endochronous μSTS, and consider a state $s \in S$ and two traces $\varphi_1, \varphi_2 \in Traces_\Sigma(s)$. If φ_2 is complete and $\varphi_1 \leq \varphi_2$, then there exists φ_3 complete such that $s.\varphi_1\varphi_3 = s.\varphi_2$ and $\varphi_1\varphi_3 \sim \varphi_2$. In addition, if φ_1 is complete then we can take $\varphi_3 = \varphi_2 \setminus \varphi_1$.

Proof: The case where φ_1 is complete is a mere corollary of lemma 4.2(2). The case where $\tau \notin \varphi_1$ is proved using axiom μWE3.4. The general case is a simple combination of the two previous cases. □

Note that we do not require confluence for arbitrary (incomplete) traces. The intuition behind this restriction is that the atomicity of reactions must be preserved, and therefore the clock transitions cannot follow the simple commutation rule of axiom μWE2. In the following weakly endochronous μSTS, for instance (initial state s), $s.< ?a >$ and $s.< \tau >< ?a >$ are different:



Also note how lemma 4.1 gives the classical independence (full commutation) results, for the case where $supp(\varphi_1)$ and $supp(\varphi_2)$ share no directed variable. However, the finer microstep notion allows us to consider systems like Σ_3 where the classical macrostep independence does not apply (in state s_0, the macrostep transitions ab and ar do not commute, yet the system is I/O deterministic).

The confluence properties of an endochronous system are even stronger, as stated by the following:

Theorem 4.1. (determinism)
Let $\Sigma = (S, \hat{s}, V, \tau, T)$ be a weakly endochronous μSTS, $s \in S$, and let φ_1, φ_2 be traces of $Traces_\Sigma(s)$ such that $\varphi_1 \sim \varphi_2$. Then

1. if $\tau \notin supp(\varphi_i)$, $i = 1, 2$, then $s.\varphi_1 = s.\varphi_2$
2. if φ_1, φ_2 are complete, then $s.\varphi_1 = s.\varphi_2$.

Proof: Point 1 is a corollary of lemma 4.2(1). Point 2 is a simple corollary of lemma 4.3. □

Note that the last lemma (point 1) tells us that we can non-ambiguously label the states reachable from a given state in one instant by the signals emitted or received to reach it.

In fact, these strong confluence properties allow us to put any trace of of a weakly endochronous system in *normal form*, in which every transition is maximal and the number of reactions minimal. The main result is:

Theorem 4.2. (maximal steps/normal form)
Let $\Sigma = (S, \hat{s}, V, \tau, T)$ be a weakly endochronous μSTS, $s \in S$, and $\varphi \in Traces_\Sigma(s)$, complete. Then, there exists $\overline{\varphi} \in Traces_\Sigma(s)$, complete, with $\overline{\varphi} \sim \varphi$ and such that $< Step_0(\overline{\varphi}) >$ is maximal (for label inclusion).

Proof: Let L be the set of all labels l of non-clock transitions starting in s such that $l \leq \varphi$. Then, for all $l_1, l_2 \in L$ we have $l_1 \bowtie l_2$. By using the same reasoning as in the proof of lemma 4.2(1), we obtain $l_1 \sqcup l_2 \in L$, for all $l_1, l_2 \in L$. The maximal transition $\overline{\varphi}[0]$ is the union of all the labels in L.

This process can be iterated to construct maximal non-clock, non-void transitions $\overline{\varphi}[j]$, $j \geq 0$, until for a given $j_0 + 1$ no such transition can be built. The process is finite, for each variable $v \in V$ can be assigned at most once by $\overline{\varphi}_1 = \overline{\varphi}[0 \ldots j_0]$.

From the maximality of $\overline{\varphi}_1$ and from lemma 4.3, we deduce that $\overline{\varphi} = \overline{\varphi}_1 < \tau > (\varphi \setminus \overline{\varphi}_1)$ is a trace of $Traces_\Sigma(s)$ with $\varphi \sim \overline{\varphi}$. The maximality of $< Step_0(\overline{\varphi}) >$ is easily proved by *reduction ad absurdum*, which completes our proof. □

We conclude the presentation of weak endochrony by stating the very important compositionality result that allows to incrementally build complex weakly endochronous systems.

Theorem 4.3. (compositionality)
Let $\Sigma_i, i = \overline{1, n}$ be composable weakly endochronous μSTSs. Then, $|_{i=1}^{n} \Sigma_i$ is weakly endochronous.

Proof: Direct application of the previous results, by taking into account the definition of the synchronous composition. □

Weak endochrony is illustrated by the μSTSs Σ_2, Σ_3, and $\Sigma_3 \mid \Sigma_2$ of section 3.1, and by all the examples of the sections 3.2 and 3.3. The μSTS Σ_1 is not weakly endochronous because the non-deterministic choice in state s_1 makes Σ_1 unpredictable, so that other components, like Σ_4, cannot adjust their behavior to preserve the synchronous semantics. The transformation of Σ_1 in Σ_1' illustrates the type of instrumentation required to transform a general μSTS into a weakly endochronous one.

4.2. Comparison with macrostep Weak Endochrony

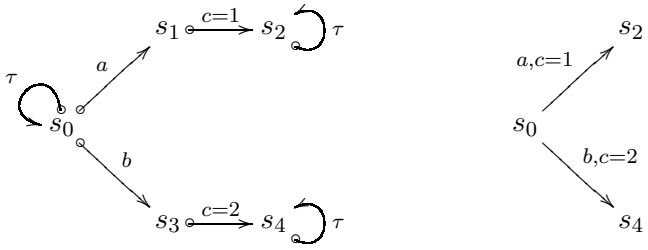
The fundamental difference between macrostep Weak Endochrony and this microstep version is that the former can make decisions involving the value of several signals received during a reaction. In our microstep framework, each decision is based on the value of only one input signal.

It is easy to associate a macrostep synchronous representation – an LSTS in the spirit of [11] – to any STS. More exactly, given $\Sigma = (S,\ \hat{s},\ V,\ \tau,\ T)$, we associate the LSTS $[\Sigma] = (S, \hat{s}, T')$, where:

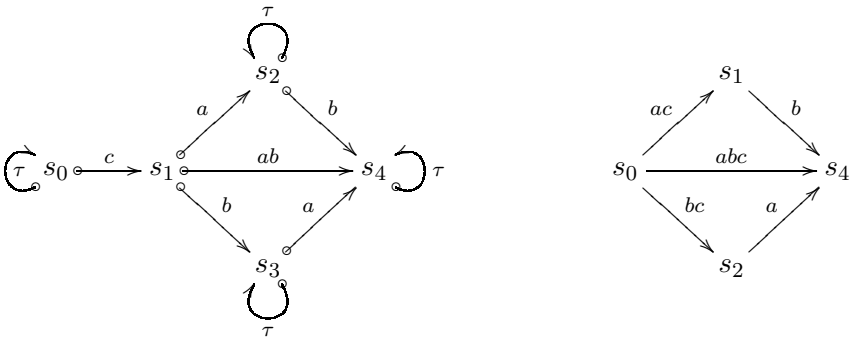
$$s \xrightarrow[[\Sigma]]{l} s' \ \Leftrightarrow \exists \varphi : \begin{cases} s \overset{\varphi}{\Longrightarrow} s' \\ \varphi = Step_0(\varphi) < \tau > \\ l =< Step_0(\varphi) > \end{cases}$$

In other words, the macrostep version considers only transitions from a synchronizing state to another synchronizing state, the other states being invisible at this level of abstraction.

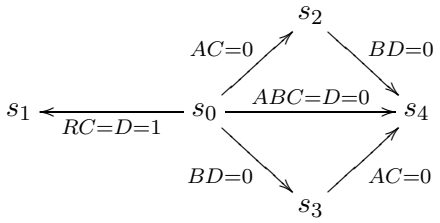
Unfortunately, the relation between microstep and macrostep weak endochrony is not simple. Given a μSTS Σ, such as the one below (at left) the fact that $[\Sigma]$ (below, at right) is weakly endochronous does not imply that Σ is microstep weakly endochronous. In our case, it is not.



At the same time, a microstep weakly endochronous system Σ is not necessarily macrostep weakly endochronous, as the following example shows:



One extra problem is that there exist macrostep weakly endochronous systems that have no microstep weakly endochronous encoding. One of them is the following:



It appears that for each macrostep weakly endochronous system there exists a microstep one over the same variables and with the same asycnhronous traces. As explained, earlier, this is due to the fact that macrostep weak endochrony can rely on tests involving several variables at a time, which is impossible in our microstep framework.

4.3. Comparison with related models

Weak endochrony belongs to a family of properties whose goal is to preserve concurrency while ensuring the correct operation of a system in an untimed asynchronous environment. We refer here to the work of Keller [15]. In this paper, Keller shows that 3 properties – *determinism*, *commutativity*, and *persistence* – ensure global confluence in a very general form of asynchronous transition system. The determinism requirement is quite common, but commutativity and persistence are the key point of the approach. They roughly correspond to axioms μWE2 and μWE3, ensuring that independent labels in a given state are concurrent and non-interfering, and remain available while not taken.

Weak endochrony follows the same principles, but in a much more specific setting:

- Our communication lines can transmit data, not mere arrival notifications. This allows us to refine our correctness criteria to take into account *choice* in the system-environment synchronization protocol (axiom μWE4).

- Weak endochrony deals with *synchronous systems*. The most natural way of ensuring persistency of transitions that are not taken is to ensure an intra-instance persistency, concerning microstep transitions (μWE2), and a macrostep persistency, covering full reactions (μWE3 states that clock transitions cannot disable other transitions). It is interesting here to recall that the macrostep weak endochrony of [11] needed only a macrostep persistency property. But here we need the microstep aspect, as well. Macrostep persistency can be seen as covering non-interfering transactions instead of elementary communications.

- Weak endochrony defines a normal, most compact form for system behaviors, something not provided by the general confluence results of Keller. This allows reasoning on convergence speed. For instance, if two synchronous reactions starting in a state are non-contradictory, then convergence between them can be attained in at most one reaction.

- Finally, our systems are input/output systems, which are predictable, but not deterministic as such (they are deterministic only if we forget about the direction of signals).

These supplementary aspects determine the complexity of the theory and the difficulty of the proofs. A Major difference with Keller's work is that he is interested in the confluence of a single system (which corresponds, in our setting, to lemma 4.3). Our work aims at finding conditions under which the semantics of a system of components does not change when we replace a strongly synchronized composition mechanism with a purely asynchronous one.

The work of Keller provides the link with two approaches used in asynchronous circuit design: speed independence and delay insensitivity. Speed independence [23, 19, 15] (which usually implies the hypothesis of semimodularity) ensures that the behavior of a circuit does not depend on the speed of its basic computing elements. Delay insensitivity [16, 17] ensures that the behavior of a circuit does not depend on the delays of its internal or external communication lines. These two properties are

important because they support the definition of circuits whose functionality remains unchanged when the fabrication process changes.

Like weak endochrony, speed independence and delay insensitivity support specializations of Keller's fundamental theorem (as noted, for instance, in [15]), but the hypothesis on the systems and communication lines are different from those of weak endochrony.

A second important difference with Keller's work is that our results are based on the assumption that an underlying communication infrastructure provides a lossless message-passing mechanism. Under this assumption, weak endochrony implies a very permissive persistency property. By comparison, speed independence and delay insensitivity ensure, among other things, that signals are not lost (in a sense, they cover at the same time the correctness of the message-passing protocol under given hypothesis[4], and the persistency property).

Wires in speed independent or delay insensitive circuits can only transmit events, not values: An event consist in the wire changing its value from 0 to 1, or from 1 to 0. Thus, value choices (as found in weakly endochronous systems) cannot be directly expressed in Keller's formalism, as the only possible choices are among different events, occurring on different wires. Microstep weak endochrony is not meant to express such choices, which depend on temporal assumptions on the environment (no input is produced by the environment until the system is ready to read it). With an appropriate introduction of clock transitions, weak endochrony should be able to directly represent delay-insensitive systems with no choice (*cf.* axiom R'_3 in [16]).

More work is needed to understand the precise relation between weak endochrony, on one side, and speed independence and delay insensitivity, on the other, particularly by defining a notion of *circuit realization* for weakly endochronous systems, along the lines of [19].

Our work bears some relations with that of Yun and Dill on burst-mode circuits [20]. Their goal is to deal with multiple-signal interactions, instead of single signal events. The approach is oriented towards circuit synthesis, and strict operation conditions are required, which basically exclude true concurrency.

4.4. Correctness results

Weak endochrony is compositional. However weak endochrony of all components does not guarantee the correctness (non-blocking) of the global synchronous specification, nor the correctness (semantics preservation) of the GALS implementation model. This can be easily checked on the systems formed by composing Σ'_1 and Σ_2 — defined in Sections 3.1 and 3.2.

The most important result of this paper is the following theorem, which states that the correctness of the synchronous composition implies the correctness of the GALS implementation.

In fact, the strong confluence and determinism properties of the weakly endochronous systems will allow to prove an even stronger result, that also insures state determinism in addition to observational behavior equivalence:

Criterion 2. (correct desynchronization for weakly endochronous systems)
Let $\Sigma_i, i = \overline{1,n}$ be composable μSTSs. Then, we shall say that the GALS implementation $\|_{i=1}^{n}\Sigma_i$ is correct *w.r.t.* the synchronous specification $|_{i=1}^{n}\Sigma_i$ if for all synchronizing state s of $|_{i=1}^{n}\Sigma_i$ and for all trace $\varphi \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ there exist $\overline{\varphi} \in Traces_{|_{i=1}^{n}\Sigma_i}(s)$ complete and $\tilde{\varphi} \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ such that $\varphi \preceq \tilde{\varphi}$, $\tilde{\varphi} \sim \iota(\overline{\varphi})$, and $\iota(s).\tilde{\varphi} = \iota(s).\iota(\overline{\varphi})$.

[4] such as the fact that forks are isochronic

Quite interestingly, Criterion 2 implies Criterion 1 (the former has extra requirements).

Theorem 4.4. (correctness)
Let $\Sigma_i, i = \overline{1,n}$ be composable weakly endochronous μSTSs. If $|_{i=1}^{n}\Sigma_i$ is non-blocking, then $\|_{i=1}^{n}\Sigma_i$ is correct *w.r.t.* $|_{i=1}^{n}\Sigma_i$ in the sense of criterion 2.

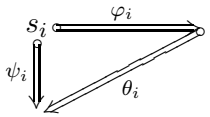
Two technical lemmas are needed to prove the theorem.

Lemma 4.4. (completion, GALS)
Let $\Sigma_1, \ldots, \Sigma_n$ be composable weakly endochronous μSTSs, let s be a synchronizing state of $|_{i=1}^{n}\Sigma_i$, and let $\psi \in Traces_{|_{i=1}^{n}\Sigma_i}(s)$, complete, and $\varphi \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ such that $\varphi \leq \iota(\psi)$. Then, there exists $\theta \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s).\varphi)$ such that $\varphi\theta \sim \iota(\psi)$ and $\iota(s).\varphi\theta = \iota(s).\iota(\psi)$

Proof: We can assume, without losing generality, that all the labels of ψ are atomic (assign exactly one variable). By projecting φ and ψ on the components Σ_i, we obtain:

$$\pi_i^{\alpha}(\varphi), \pi_i^{\sigma}(\psi) \in Traces_{\Sigma_i}(\pi_i^{\sigma}(s)) \text{ with } \begin{cases} \pi_i^{\sigma}(\psi) \text{ complete} \\ \pi_i^{\alpha}(\varphi) \leq \pi_i^{\sigma}(\psi) \end{cases}$$

We denote with $\varphi_i = \pi_i^{\alpha}(\varphi)$, $\psi_i = \pi_i^{\sigma}(\psi)$, $s_i = \pi_i^{\sigma}(s)$. By applying lemma 4.3(2), we find a complete trace θ_i such that the following holds in Σ_i:



Now recall that the construction process used by lemma 4.3 (based on the constructions of lemma 4.2) insures that in each θ_i the atomic communication operations (non-clock labels) are ordered in the same fashion as they are in ψ_i. More precisely, let r_i be the rank in ψ_i of the non-clock label that has rank i in θ_i. Whenever $r_i > r_j$, we have $i > j$. The same relation is preserved by the projection of ψ onto ψ_i. Then, the ordering of the operations of the θ_i in ψ can be used to interleave the labels of the traces θ_i into θ, and our lemma is proved. □

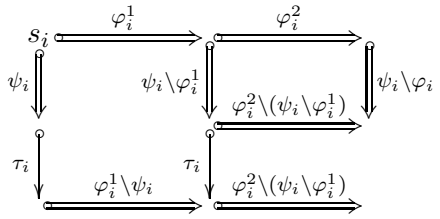
Lemma 4.5. (technical)
Let $\Sigma_1, \ldots, \Sigma_n$ be composable weakly endochronous μSTSs, let s be a synchronizing state of $|_{i=1}^{n}\Sigma_i$, and let $\varphi, \psi < \tau_1 \ldots \tau_n > \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ such that $\varphi \bowtie \psi$ and $\forall i : \tau_i \notin supp(\psi)$. Then:

$$\varphi(\psi \setminus \varphi), \psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi) \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$$

Proof: By projecting φ and ψ on the components Σ_i, we obtain:

$$\pi_i^{\alpha}(\varphi), \pi_i^{\alpha}(\psi) < \tau_i > \in Traces_{\Sigma_i}(\pi_i^{\sigma}(s)) \text{ with } \begin{cases} \pi_i^{\alpha}(\varphi) \bowtie \pi_i^{\alpha}(\psi) \\ \tau_i \notin supp(\pi_i^{\alpha}(\psi)) \end{cases}$$

We denote with $\varphi_i = \pi_i^{\alpha}(\varphi)$, $\psi_i = \pi_i^{\alpha}(\psi)$, $s_i = \pi_i^{\sigma}(s)$. Also let $\varphi_i = \varphi_i^1\varphi_i^2$ with φ_i^1 complete and $\tau_i \notin supp(\varphi_i^2)$. All these elements are pictured in fig. 2, which also contains the other transitions that will be constructed during this proof.



where $\begin{cases} (\varphi_i^1 \setminus \psi_i)(\varphi_i^2 \setminus (\psi_1 \setminus \varphi_i^1)) = \varphi_i \setminus \psi_i \\ \varphi_i^1 \varphi_i^2 = \varphi_i \end{cases}$

Figure 2. Diagram with the transitions used in the proof of lemma 4.5

By applying lemma 4.2(2), $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1) < \tau_i >$ and $s_i.\psi_i < \tau_i > (\varphi_i^1 \setminus \psi_i)$ exist and are equal.

By applying lemma 4.2(1) in state $s_i.\varphi_i^1$ and for the traces φ_i^2 and $\psi_i \setminus \varphi_i^1$, we conclude that $s_i.\varphi_i^1\varphi_i^2(\psi_i \setminus \varphi_i)$ and $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1)(\varphi_i^2 \setminus (\psi_i \setminus \varphi_i^1))$ exist and are equal.

The existence of $s_i.\varphi_i^1\varphi_i^2(\psi_i \setminus \varphi_i)$ means that $\varphi_i(\psi_i \setminus \varphi_i) \in Traces_{\Sigma_i}(s_i)$.

The existence of $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1)(\varphi_i^2 \setminus (\psi_i \setminus \varphi_i^1))$ means that $\theta_i = \varphi_i^2 \setminus (\psi_i \setminus \varphi_i^1)$ is a trace of Σ_i starting in $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1)$. Since $< \tau_i >$ is a trace starting in the same state, from axiom μWE1 we obtain that θ_i is a trace of Σ_i starting in state $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1) < \tau_i >$, and by the identity of $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1) < \tau_i >$ and $s_i.\psi_i < \tau_i > (\varphi_i^1 \setminus \psi_i)$ we have $\psi_i < \tau_i > (\varphi_i \setminus \psi_i) \in Traces_{\Sigma_i}(s_i)$.

We proved that for all $1 \leq i \leq n$ we have $\varphi_i(\psi_i \setminus \varphi_i), \psi_i < \tau_i > (\varphi_i \setminus \psi_i) \in Traces_{\Sigma_i}(s_i)$, meaning that

$$\forall i : \pi_i^\alpha(\varphi(\psi \setminus \varphi)), \pi_i^\alpha(\psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi)) \in Traces_{\Sigma_i}(\pi_i^\sigma(s))$$

On each channel, the projection of any of $\varphi(\psi \setminus \varphi)$ or $\psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi)$ is either a prefix of the projection of φ, or a prefix of the projection of ψ, which are themselves consistent. Therefore, traces $\varphi(\psi \setminus \varphi)$ and $\psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi)$ are also consistent. According to lemma 2.3, this implies that $\varphi(\psi \setminus \varphi), \psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi)$ is in $Traces_{\|_{i=1}^n \Sigma_i}(\iota(s))$. □

Thanks to these two technical lemmas, theorem 4.4 can now be proved.

Proof of theorem 4.4: Let s be a synchronizing state of $|_{i=1}^n \Sigma_i$ and let $\varphi \in Traces_{\|_{i=1}^n \Sigma_i}(\iota(s))$. We prove the existence of $\overline{\varphi}$ and $\tilde{\varphi}$ by induction over $\mid \varphi \mid$ (the number of variable assignments in φ). We can assume, without losing generality, that every label of φ assigns exactly one variable, either clock or directed. The reduction to this case is straightforward.

If $\mid \varphi \mid = 0$, then $\tilde{\varphi} = \varphi$ and $\overline{\varphi} =< \tau >$ clearly satisfy the conditions of criterion 2.

Consider now φ with $\mid \varphi \mid \geq 1$. If $\varphi[0] =< \tau >$, then we have:

$$\iota(s) \xrightarrow{\varphi[0]} \iota(s) \xRightarrow{\varphi[1 \ldots length(\varphi)-1]}$$

The induction hypothesis can be applied on $\varphi[1 \ldots length(\varphi) - 1]$ to determine $\tilde{\varphi}$ and $\overline{\varphi}$.

From now on, we assume that $\varphi[0]$ is not a clock transition.

To prove that $\tilde{\varphi}$ and $\overline{\varphi}$ exist, we shall first construct $\psi \in Traces_{|_{i=1}^n \Sigma_i}(s)$ such that $\tau \notin supp(\psi)$, $\psi < \tau > \in Traces_{|_{i=1}^n \Sigma_i}(s)$, $\psi[0] = \varphi[0]$, and $\iota(\psi) \bowtie \varphi$.

Construction of ψ: Consider the projections of φ onto components $\pi_j^\alpha(\varphi)$, $1 \leq j \leq n$. Since $\varphi[0]$ is not the empty transition, nor a clock transition, then there exists at least an j such that $\pi_j^\alpha(\varphi)[0]$ is not a clock transition, nor a void transition. Note that $\pi_j^\alpha(\varphi)[0]$ is fireable in state $\pi_j^\sigma(s)$.

Then, we start the iterative construction of ψ by setting the iteration counter i to 0 and setting $\psi_0 = \varphi[0]$. The iteration step:

continuation test: If $\psi_i < \tau > \in Traces_{|\stackrel{n}{i=1}\Sigma_i}(s)$, then we completed our construction.

construction step: If $\psi_i < \tau > \notin Traces_{|\stackrel{n}{i=1}\Sigma_i}(s)$, and since $|\stackrel{n}{i=1}\Sigma_i$ is non-blocking, there exists a label $< v = x >$ such that $\psi_i < v = x > \in Traces_{|\stackrel{n}{i=1}\Sigma_i}(s)$. If $\psi_i < v = x > \bowtie \varphi$, then consider $\psi_{i+1} = \psi_i < v = x >$ and go to the continuation test.

If not, then $\varphi = \varphi_0 < v = y > \varphi_1$ with $x \neq y$ and $v \notin supp(\varphi_0)$. However, by applying axiom μWE4, we have $\psi_i < v = y > \in Traces_{|\stackrel{n}{i=1}\Sigma_i}(s)$. By considering $\psi_{i+1} = \varphi_i < v = y >$, we also have $\psi_{i+1} \bowtie \varphi$, and we go to the continuation test.

The previous algorithm is finite, bounded by the number of variables in $|\stackrel{n}{i=1}\Sigma_i$. In the end we put the last ψ_i in ψ.

Construction of $\tilde{\varphi}$ and $\overline{\varphi}$: Let $s_1 = s.\psi < \tau >$. According to lemma 4.5 we have

$$\varphi(\iota(\psi) \setminus \varphi), \iota(\psi) < \tau_1, \ldots, \tau_n > \in Traces_{\|\stackrel{n}{i=1}\Sigma_i}(\iota(s))$$

Since $\varphi[0] = \iota(\psi)[0]$, we have $\mid \varphi \setminus \psi \mid < \mid \varphi \mid$. Then, we can apply the induction hypothesis in the synchronizing state s_1, and obtain $\overline{\theta} \in Traces_{|\stackrel{n}{i=1}\Sigma_i}(s_1)$, complete, and $\tilde{\theta} \in Traces_{\|\stackrel{n}{i=1}\Sigma_i}(\iota(s_1))$ such that:

$$\begin{cases} \varphi \setminus \iota(\psi) \preceq \tilde{\theta} \\ \iota(\overline{\theta}) \sim \tilde{\theta} \\ \iota(s_1).\iota(\overline{\theta}) = \iota(s_1).\tilde{\theta} \end{cases}$$

Let $\overline{\varphi} = \phi < \tau > \overline{\theta}$. Given that $\varphi \leq \iota(\psi) < \tau_1 \ldots \tau_n > (\varphi \setminus \iota(\psi))$ and that $\varphi \setminus \iota(\psi) \leq \iota(\overline{\theta})$, we deduce $\varphi \leq \iota(\overline{\varphi})$. Since $\overline{\varphi} \in Traces_{|\stackrel{n}{i=1}\Sigma_i}(s)$ is complete, lemma 4.4 can be applied to build $\varphi' \in Traces_{\|\stackrel{n}{i=1}\Sigma_i}(\iota(s).\varphi)$ such that $\varphi\varphi' \sim \iota(\overline{\varphi})$ and $\iota(s).\varphi\varphi' = \iota(s).\iota(\overline{\varphi})$. By considering $\tilde{\varphi} = \varphi\varphi'$, the proof is completed. □

Theorem 4.4 implies that for large classes of components for which simple wrappers exist, the correctness of the GALS implementation is implied by the correctness of the global synchronous specification. Thus, no extra signalization is needed to ensure semantics preservation (and no costly synthesis algorithms). The GALS implementation is correct by construction.

5. Conclusion. Future work

We introduced a new model for the representation of asynchronous implementations of synchronous specifications. The model covers implementations where a notion of global synchronization is preserved by means of signaling, and GALS implementations, where global synchronization is relaxed. The model takes into account computation and communication causality, and allows us to reason about semantics-preservation and absence of deadlocks in the GALS deployment of synchronous specifications. As the model captures the internal concurrency of the synchronous specification, our correctness criteria support implementations that are less constrained and more efficient than existing ones.

The results of section 4 suggest that our model offers a good abstraction level for reasoning about desynchronization. In particular, the level of detail is essential in revealing the intricate relation between (1) causal dependencies, concurrency and conflicts in the micro-step semantics of a synchronous specification and (2) the correctness (semantics preservation) of its GALS implementation.

5.1. Future work

Thanks to this new model, we are exploring the development of GALS circuits made of synchronous IPs. Our work aims at using asynchronous logic wrappers to encapsulate the components of a modular synchronous circuit into delay insensitive components. Our model seems well-suited to analyze designs involving both synchronous and asynchronous circuit specifications. Preliminary results are presented in [24], but we are only at the beginning of our work.

We are also considering symbolic analysis techniques that would allow us to translate the theory detailed in this paper to high-level synchronous languages like Signal or Esterel, instead of simple finite state automata. The objective is to derive efficient algorithms transforming general high-level specifications into weakly endochronous ones. Preliminary results in this direction are presented in [25].

A third research direction concerns the (still not sufficiently clear) relations between classical, macro-step synchronous models and more operational models like microstep synchronous transition systems (μSTS), or the ones covering the implementations of synchronous programming languages, especially when desynchronization is involved. For instance, it is important to understand how the notions of correct desynchronization and endochrony can be transposed into a constructive framework such as the one of Esterel [26].

References

[1] N. Halbwachs, *Synchronous programming of reactive systems*. Kluwer Academic Publishers, 1993.

[2] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone, "The Synchronous Languages Twelve Years Later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[3] D. Potop-Butucaru, R. de Simone, and J.-P. Talpin, "The synchronous hypothesis and synchronous languages," in *The Embedded Systems Handbook*, R. Zurawski, Ed., 2005, CRC Press.

[4] H. Kopetz, *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[5] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli, "The theory of latency-insensitive design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1059–1076, 2001.

[6] N. Lynch and E. Stark, "A proof of the Kahn principle for input/output automata," *Information and Computation*, vol. 82, no. 1, pp. 81–92, 7 1989.

[7] P. Caspi, A. Girault, and D. Pilaud, "Automatic distribution of reactive systems for asynchronous networks of processors," *IEEE Trans. on Software Engineering*, vol. 25, no. 3, pp. 416–427, 1999.

[8] J.-P. Talpin, P. L. Guernic, S. K. Shukla, R. Gupta, and F. Doucet, "Formal refinement checking in a system-level design methodology," *Fundamenta Informaticae*, vol. 62, no. 2, pp. 243–273, 2004.

[9] A. Benveniste, B. Caillaud, and P. L. Guernic, "Compositionality in dataflow synchronous languages: Specification and distributed code generation," *Information and Computation*, vol. 163, no. 1, pp. 125–171, 2000.

[10] M. Singh and M. Theobald, "Generalized latency insensitive systems for gals architectures," in *Proceedings FMGALS2003*, Pisa, Italy, 2003.

[11] D. Potop-Butucaru, B. Caillaud, and A. Benveniste, "Concurrency in synchronous systems," *Formal Methods in System Design*, vol. 28, no. 2, pp. 111–130, 2006.

[12] P. Caspi, A. Girault, and D. Pilaud, "Automatic distribution of reactive systems for asynchronous networks of processors," *IEEE Transactions on Software Engineering*, vol. 25, no. 3, pp. 416–427, 1999.

[13] T. Grandpierre and Y. Sorel, "From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations," in *Proceedings MEMOCODE'03*, Mont Saint-Michel, France, 2003.

[14] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou, "Handshake protocols for de-synchronization," in *Proceedings Async04*, Crete, Greece, 2004, pp. 149–158.

[15] R. Keller, "A fundamental theorem of asynchronous parallel computation," *Lecture Notes in Computer Science*, vol. 24, pp. 103–112, 1975.

[16] J. T. Udding, "A formal model for defining and classifying delay-insensitive circuits and systems," *Distributed Computing*, vol. 1, no. 4, pp. 197–204, 1986.

[17] A. Martin, "The limitations of delay-insensitivity in asynchronous circuits," Caltech, technical report CS-TR-90-02, 1990.

[18] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer, 2002.

[19] R. Keller, "Towards a theory of speed-independent modules," *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 21–33, January 1974.

[20] K. Yun and D. Dill, "Automatic synthesis of extended burst-mode circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 101–132, feb. 1999.

[21] E. Stark, "Concurrent transition systems," *Theoretical Computer Science*, vol. 64, no. 3, pp. 221–269, 1989.

[22] M. Mukund, "Petri nets and step transition systems," *International Journal of Foundations of Computer Science*, vol. 3, no. 4, pp. 443–478, 1992.

[23] D. E. Muller and W. S. Bartky, "A theory of asynchronous circuits," in *Proceedings of an International Symposium on the Theory of Switching*. Harvard University Press, 1959, pp. 204–243.

[24] S. Dasgupta, D. Potop-Butucaru, B. Caillaud, and A. Yakovlev, "From weakly endochronous systems to delay-insensitive circuits," in *Proceedings FMGALS'05*, Verona, Italy, 2005.

[25] J.-P. Talpin, D. Potop-Butucaru, J. Ouy, and B. Caillaud, "Compositional synthesis of latency-insensitive systems from multi-clocked synchronous specifications," in *Proceedings EMSOFT'05*, Jersey City, NJ, USA, 2005.

[26] G. Berry, "The constructive semantics of pure Esterel," July 1999, draft book available at http://www.esterel-technologies.com/.

[27] G. Kahn and G. Plotkin, "Concrete domains," *Theoretical Computer Science*, vol. 121, no. 1&2, pp. 187–277, 1993.