Form Method Syst Des (2006) 28: 111–130
DOI 10.1007/s10703-006-7844-8

Concurrency in synchronous systems

**Dumitru Potop-Butucaru · Benoît Caillaud ·
Albert Benveniste**

Published online: 3 May 2006

Abstract In this paper we introduce the notion of weak endochrony, which extends to a synchronous setting the classical theory of Mazurkiewicz traces. The notion is useful in the synthesis of correct-by-construction communication protocols for globally asynchronous, locally synchronous (GALS) systems. The independence between various computations can be exploited here to provide communication schemes that do not restrict concurrency while still guaranteeing correctness. Such communication schemes are then lighter and more flexible than their latency-insensitive or endo/isochronous counterparts.

Keywords Synchronous · Distribution · Desynchronization · Globally asynchronous locally synchronous (GALS) · Concurrency · Trace theory

1. Introduction

The *notion of time* has been a crucial aspect of electronic system design for years. Dealing with concurrency, time and causality has become increasingly difficult as the complexity of the design grew. The *synchronous programming model* [4, 9, 16] has had major successes at the specification level because it provides a simpler way to access the power of concurrency in functional specification. Synchronous languages like ESTEREL [5], LUSTRE [10], and SIGNAL [13], the quasi-synchronous STATECHARTS [11] modeling methodology, and design environments like SIMULINK/STATEFLOW [1] all benefit from the simplicity of the *synchronous assumption*, *i.e.*: (1) the system evolves through an infinite sequence of successive atomic reactions indexed by a *global logical clock*, (2) during a reaction each

Work supported by the ARTIST and COLUMBUS IST European projects

D. Potop-Butucaru (✉) · B. Caillaud · A. Benveniste
IRISA, Campus de Beaulieu, 35042 Rennes, France
e-mail: Dumitru.Potop@irisa.fr

B. Caillaud
e-mail: Benoit.Caillaud@irisa.fr

A. Benveniste
e-mail: Albert.Benveniste@irisa.fr

component of the system computes new events for all its output signals based on its internal state and on the values of its input signals, and (3) the communication of all events between components occur *synchronously* during each reaction.

However, if the synchronous assumption simplifies system specification and verification, the problem of deriving a correct physical implementation from it does remain [4]. 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 (e.g. CAN for automotive applications, ARINC for avionics, and Ethernet for industrial automation). To use a synchronous approach in the development of such applications, one solution is to replace the asynchronous buses with communication infrastructures[1] that comply with a notion of global synchronization. However, such a fully synchronous implementation must be conservative, forcing the global clock to run as slow as the slowest computation/communication process. Consequently, the overhead implied by time-triggered architectures and synchronous implementations is often large enough to convince designers to use the asynchronous buses mentioned above.

Gathering advantages of both the synchronous and asynchronous approaches, the Globally Asynchronous Locally Synchronous (GALS) architectures are emerging as the 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 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 designs over GALS architectures.

1.1. Informal discussion of the issues

In the *synchronous paradigm* [4, 9, 16], execution and communication progress along a sequence of *reactions.* A synchronous run, also called *trace*, is a sequence $r_1, r_2, \dots$, where each reaction r_i assigns values to the set of variables U of the considered 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.

In contrast, no global clock exists in the *asynchronous* paradigm, and therefore no notion of reaction. Asynchronous runs of a program, also called *histories*, are tuples of signals, each signal giving the sequence of values of a variable. Absence ($\perp$) has no meaning and cannot be sensed.

Relaxing the synchronous communication to obtain an asynchronous or GALS implementation—an operation called *desynchronization*—consists of removing the signal absence events ($\perp$) and the synchronization boundaries of the reactions. The

[1] Like the family of Timed-Triggered Architectures introduced and promoted by H. Kopetz [12].

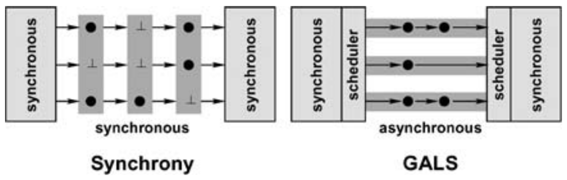


Fig. 1 From synchrony to GALS. Bullets represent informative values (messages). Vertical gray boxes represent reactions, horizontal ones represent asynchronous signals

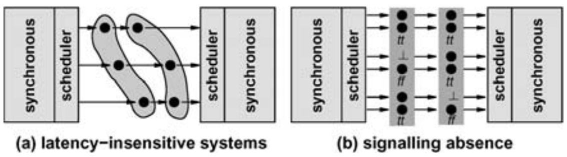


Fig. 2 Existing solutions

desynchronization problem addressed in this paper is informally illustrated in Fig. 1, which shows how a channel of a small synchronous model is substituted in the implementation process by three asynchronous communication lines. Immerging the synchronous modules in an asynchronous environment requires here the use of *schedulers* that (1) decide when reactions are computed by each component (e.g. when all locally-needed input is available) and (2) reconstruct synchronous inputs from their desynchronized counterparts, and feed them to the associated components (an operation called *resynchronization*)

The problem is that relaxing synchrony is clearly not invertible: the three asynchronous signals in Fig. 1(b) can be reorganized in successive reactions in many ways. In other words, resynchronization is not possible in the general case.

Metric time is not so much an issue, as revealed by Fig. 2(a), which depicts a run of a synchronous program in which all variables are present in all reactions. The effect of an asynchronous medium typically results in offsets between the dates of arrival of the variables that are part of the same reaction. However, this can be easily corrected at the receiver end thanks to a proper buffering mechanism. Such a technique has been used in [6] for hardware circuits, where it is referred to as *latency insensitive design*.

Unfortunately, this simple method does not extend to the case of Fig. 1: Since some variables can be absent in a nondeterministic way in each reaction, the buffering mechanism used in [6] does not ensure the correct reconstruction of reactions. A first solution to the general problem, illustrated in Fig. 2(b), would consist in attaching to each variable v an additional Boolean variable $[v]$ which indicates at each reaction whether v is present or absent. However, this simple solution has two drawbacks: It results in twice as many communications and, more importantly, the components of the resulting distributed system are running at the same pace. This is inappropriate whenever components have dissimilar activation rates.

The main issues concerning desynchronization are *correctness* and *efficiency*. Correctness is important because the advantages of synchrony lie with specification and verification. We would therefore like that each asynchronous history of the GALS implementation is covered by the verification of the initial synchronous model. Consequently, we require that each history is the desynchronization of a trace of the initial synchronous specification. We also expect a GALS implementation *not* to restrict the functionality of the system through tough scheduling policies aimed at ensuring correctness in a simple way. The correct desynchronization criterion is therefore:

*Criterion 1 (*semantics preservation, informal). Desynchronizing the traces of the initial synchronous system produces the exact set of asynchronous executions (histories) of the GALS implementation.

Efficiency, in terms of speed or number of exchanged messages, is also important, because GALS embedded systems often run on architectures with few resources. The issue is obviously subject to many trade-offs, but exploiting the independence between computations or identifying execution modes to minimize communication, power consumption, or to allow multi-rate computation proves useful in most cases (thus offering criteria for comparing solutions).

1.2. Previous work

The most general **analysis** of the distributed implementation problem is due to Benveniste et al. [3]. Heterogeneous models such as the GALS architectures are formalized there along with parallel composition operators. A comprehensive notion of correct deployment of a heterogenous system (a form of semantics preservation) is then defined, which covers the distributed implementation of synchronous specifications.

Previous approaches to **implementing** synchronous specifications over GALS architectures are respectively based on *latency-insensitive systems*, on *endo/isochronous systems*, and on *Kahn process networks* (KPN). All of them follow a general pattern by trying to transform the components of the initial specification into "equivalent" synchronous components that have been "normalized" (by modifying their interface) in such a way as to make trivial schedulers (like those needed in Fig. 2) correct.

Every such approach is basically defined by two properties, which in turn determine the scope and complexity of the "normalizing" synthesis methods and the exact type of schedulers to consider: **Scheduling-independence** characterizes the "normalized" synchronous components that know how to read their inputs, so that we can consider them self-clocked. The scheduling-independence properties of the previously-mentioned approaches are respectively latency-insensitivity, endochrony, and *I/O* determinism. The second property is the actual **semantics-preservation criterion**, which ensures that enough signaling has been added between the self-clocked components as to prohibit asynchronous executions not corresponding to synchronous ones. Such a property is isochrony; the KPN-based approach does not specify one, not covering correctness aspects; the highly constrained latency-insensitivity ensures by itself the preservation of semantics.

In the *latency-insensitive systems* of Carloni et al. [6], each synchronous component reads each input and writes every output at each reaction (see Fig. 2(a)). The communication protocols effectively simulate a single-clocked system, which is inefficient, but simplifies implementation.

An essential improvement is brought by previous work by Benveniste et al. on *endo/isochronous systems* [2]. Informally speaking, a synchronous component is endochronous when the presence or absence of each variable can be inferred incrementally during each reaction from the values of state variables and present input variables. A pair of synchronous components is isochronous if there exists no pair of reactions that are not synchronizable, yet not contradictory (i.e. both present with different value on some common variable). Both endochrony and isochrony can be model-checked and even synthesized. Unlike latency-insensitivity, the endo/isochronous approach is able to take into account execution modes and independence between components in order to minimize communication and allow multi-rate computation. The problem of the approach is that endochrony is not compositional, mainly due to poor handling of concurrency within components. This leads to inefficient synthesis for systems formed of more than 2 components.

While incomplete from the point of view of the semantics preserving criterion, we cite here the approach based on *Kahn process networks* [14] because it is the only one formulated in a causal framework. Here, by requiring that each component has a deterministic input/output function, the determinism of the global system (and thus the independence from the scheduling scheme) is guaranteed. 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.

From another point of view (detailed in Section 5), our work seems closely related to results concerning the design of asynchronous [7] and burst-mode [17] circuits. Scheduling independence, in particular, can be seen as a speed-independence property guaranteeing correctness and determinism regardless of the relative speeds of different computations and communications.

1.3. Contribution

While following the same pattern, our work brings an essential improvement over latency-insensitive design and endo/isochrony by allowing operations within a component to run independently when synchronization is not necessary. Being formulated in a non-causal framework, our approach is also less constrained than the KPN-based one, allowing non-determinism in the less abstract causal model. The scheduling-independence criterion is in our case *weak endochrony*, while *weak isochrony* ensures the preservation of semantics. The two properties form together a correct desynchronization criterion that is decidable on finite synchronous systems. Moreover, transforming a general synchronous system to satisfy them is easy (although making it in an efficient way is a difficult, yet unsolved problem).

Our main contribution is the definition of weak endochrony—a non-trivial extension to a synchronous setting of the classical trace theory [8]. The notion supports signalization schemes that are simpler and more efficient than their latency-insensitive and endo/isochronous counterparts. This is due to the fact that weak endochrony is compositional (simpler synthesis methods) and able to represent concurrency (lighter communication schemes).

1.4. Outline

The remainder of the paper is organized as follows: Section 2 defines the formal framework used throughout the paper. Section 3 is on *weak endochrony*. It introduces the notion and the normal form results (which show the strong relation with trace theory). Section 4 formally defines our desynchronization problem, introduces weak isochrony, and shows that weak endochrony and weak isochrony form a decidable criterion that guarantees correct desynchronization. Section 5 gives a technical comparison with existing results.

2. Definitions

We formally define in this section the notions used throughout the paper: reactions, traces, histories, the model of synchronous system, and parallel composition.

2.1. Variables, values, and reactions

A *finite set* $\mathcal{V}$ of *variables* represents the communication channels connecting the synchronous components. Without losing generality, we assume that all the values exchanged between components belong to a single domain $\mathcal{D}$. Each variable $v \in \mathcal{V}$ has at every moment a value $x \in \bar{\mathcal{D}} = \mathcal{D} \cup \{\bot, *\}$. A value $x \in \mathcal{D}$ represents a communication on the channel associated with v, $x = \bot$ represents the absence of communication, and $x = *$ tells us that the status of the channel is not known.

The interaction between a synchronous component and its environment is a sequence of *reactions*, which are mappings $r \in \mathit{Reactions} = \mathcal{V} \to \bar{\mathcal{D}}$. The *signature* of a reaction r is $sig(r) = \{v \mid r(v) \neq *\}$. Reactions of a synchronous component will all have the same signature, given by the set of variables of the component. We denote with $\mathit{Reactions}(U)$ the set of reactions of signature $U \subseteq \mathcal{V}$. The *image* of a reaction r through the signature U is the reaction $r_{|U} \in \mathit{Reactions}(U)$ which equals r over variables common to U and $sig(r)$ and equals $\bot$ over $U \setminus sig(r)$. The *support* of a reaction r is $supp(r) = \{v \mid r(v) \in \mathcal{D}\}$. Note that $supp(r) \subseteq sig(r)$ for any reaction r. The *stuttering reaction* $\bot_U$ has empty support and signature U.

On $\bar{\mathcal{D}}$ we define two partial orders. The first, denoted with $\leq$, is the least partial order such that $\forall x \in \mathcal{D} : * \leq \bot \leq x$. The second, denoted with $\sqsubseteq$, is the least partial order such that $* \sqsubseteq \bot$ and $\forall x \in \mathcal{D} : * \sqsubseteq x$. Each of the two relations induces *least upper bound* and *greatest lower bound* operators. We denote them, respectively, with $\vee$ and $\wedge$ (for $\leq$), and $\sqcup$ and $\sqcap$ (for $\sqsubseteq$). The operators $\wedge$ and $\sqcap$ are totally defined, while $\vee$ and $\sqcup$ are only partial. Note that $r_1 \vee r_2$ is defined whenever $r_1 \sqcup r_2$ is defined, and in this case the two values are equal. The order relations $\leq$ and $\sqsubseteq$ are extended variable-wise to reactions, and we denote with $\vee$, $\wedge$, $\sqcup$, and $\sqcap$ the associated least upper bound and greatest lower bound operators on reactions. For any $r_1, r_2 \in \mathit{Reactions}$ and $\theta \in \{\vee, \wedge, \sqcup, \sqcap\}$, $r_1 \theta r_2$ exists whenever $r_1(v)\theta r_2(v)$ exists for all $v \in \mathcal{V}$, and in this case $\forall v \in \mathcal{V} : (r_1 \theta r_2)(v) = r_1(v)\theta r_2(v)$.

We will say that the reactions r_1 and r_2 are *synchronizable* whenever $r_1 \sqcup r_2$ exists. When $r_1 \vee r_2$ exists, we say that r_1 and r_2 are *non-contradictory*. For any two non-contradictory reactions $r_1, r_2 \in \mathit{Reactions}(U)$ we define $r_1 \setminus r_2 \in \mathit{Reactions}(U)$ by $\forall v \in U$, $(r_1 \setminus r_2)(v) = r_1(v)$ if $r_2(v) = \bot$ and $(r_1 \setminus r_2)(v) = \bot$ otherwise. We also define in this case $r_1 \Delta r_2 = (r_1 \setminus r_2) \vee (r_2 \setminus r_1)$.

2.2. Traces and histories

A *synchronous trace*, simply called *trace* in the sequel, is a sequence of reactions having the same signature. For $U \subseteq \mathcal{V}$, we denote with $\mathit{Traces}(U)$ the set of traces of signature U: $\mathit{Traces}(U) = \mathit{Reactions}(U)^\omega = \{(r_i)_{1 \leq i < n} \mid n \leq \infty \wedge \forall i : sig(r_i) = U\}$. We denote with Traces the set of all traces. The length of a trace $\tau = (r_i)_{1 \leq i < n} (n \leq \infty)$ is $length(\tau) = n - 1$; its ith element r_i is denoted $\tau[i]$. The finite sub-trace $r_i r_{i+1} \ldots r_j$ of τ is denoted with $\tau[i, j]$ (if $i > j$, then $\tau[i, j] = \epsilon$). The suffix $(r_j)_{i \leq j < n}$ of τ is denoted with $\tau[i \ldots]$. Any reaction is also a trace of length 1. Two traces τ_1 and τ_2 can be concatenated if they have the same signature and if $length(\tau_1) < \infty$. The trace τ_1 is a prefix of τ_2 (written $\tau_1 \preceq \tau_2$) if, by definition, $\tau_2 = \tau_1 \tau_3$ for some τ_3. The prefix relation is a partial order over traces.

The order relation $\sqsubseteq$ and the operators $\sqcup$, and $\sqcap$ are component-wise extended to pairs of traces of the same length. Given two traces τ_1 and τ_2 we shall say that they are *synchronizable* if $\tau_1 \sqcup \tau_2$ exists. We also extend component-wise to traces the image operator $\tau_{|U}$.

A *history* is an asynchronous observation of one or more synchronous components. In a history, the synchronization constraints are forgotten, so that only the communications/values can be observed. Formally, a history is any mapping $\chi \in \mathit{Histories} = \mathcal{V} \to \mathcal{D}^\omega$. The concatenation operator over *Histories* is the variable-wise extension of the concatenation operator over $\mathcal{D}^\omega$. The associated prefix order, denoted with $\preceq$, induces a greatest lower bound $\sqcap$ and a least upper bound $\sqcup$ operator. The *support* of a history χ is $supp(\chi) = \{v \in \mathcal{V} \mid \chi(v) \neq \epsilon\}$, where ϵ denotes the empty word over $\mathcal{D}$. In the sequel, we shall also denote with ϵ the trace of length 0 and the empty history. The length of a history χ is $length(\chi) = \max_{v\in\mathcal{V}} length(\chi(v))$.

The *desynchronization morphism* $\delta : Traces \to Histories$ associates an asynchronous observation to every synchronous trace by forgetting the synchronization values $*$ and $\perp$: (i) $\delta(\epsilon) = \epsilon$; (ii) $\forall r \in Reactions,\ v \in \mathcal{V}$, $\delta(r)(v) = r(v)$, if $r(v) \in \mathcal{D}$, and $\delta(r)(v) = \epsilon$, otherwise; (iii) $\delta(\tau_1\tau_2) = \delta(\tau_1)\delta(\tau_2)$. The morphism preserves the concatenation and least upper bound operators. For $\chi \in Histories$ and $U \subseteq \mathcal{V}$, we denote with $head_U(\chi)$ the maximal reaction of signature U such that its desynchronization is a prefix of χ.

2.3. Synchronous transition systems

A *labeled synchronous transition system* (LSTS) is a synchronous automaton $\Sigma = (U, S, \to, \hat{s})$, where $U \subseteq \mathcal{V}$ is the set of variables of Σ (its signature), S is the set of *states*, $\to \subseteq S \times Reactions(U) \times S$ is the transition relation, and $\hat{s}$ is the initial state of the system. The notation $s \xrightarrow[\Sigma]{r} s'$ shall be used in the sequel to represent the transition $(s, r, s') \in \to$. For $s \in S$ and $\rho \in Reactions(U)$ we denote $Reactions_\Sigma(s) = \{r \mid \exists s' : s \xrightarrow[\Sigma]{r} s'\}$ and $Reactions_\Sigma(s, \rho) = \{r \mid \exists s' : s \xrightarrow[\Sigma]{r} s' \wedge r \le \rho\}$. When confusion is not possible, the name of the LSTS can be omitted from these notations.

The LSTS $\Sigma = (U, S, \to, \hat{s})$ *accepts the trace* $\tau \in Traces(U)$ *in the state* s if there exist the states $s_1 = s, s_2, \ldots$ such that for all i $s_i \xrightarrow{\tau[i]} s_{i+1}$. When $n = length(\tau)$ is finite, we also write $s \overset{\tau}{\Rightarrow} s_n$. We denote with $Traces_\Sigma(s)$ the language formed by the traces accepted by

```
module Crossbar:
input  I1,I2,R; output O1,O2;
relation I1#R I2#R;
loop
  abort
    loop await I1; emit O1 end
  ||
    loop await I2; emit O2 end
  when R ;
  abort
    loop await I1; emit O2 end
  ||
    loop await I2; emit O1 end
  when R
end
end module
```

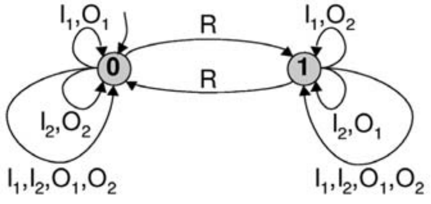
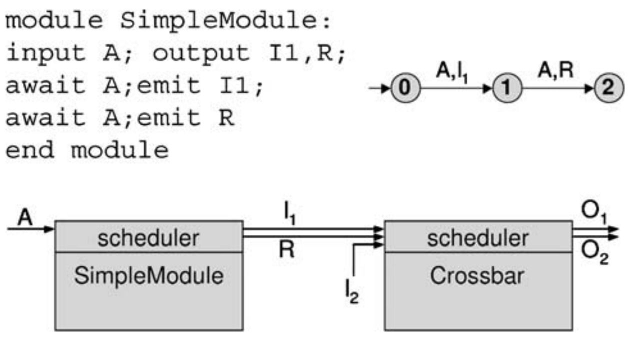


Fig. 3 A small synchronous program, meant to model a small communication crossbar, and its LSTS. We omitted stuttering reactions and $\perp$ values



```
module SimpleModule:
input A; output I1,R;
await A;emit I1;
await A;emit R
end module
```

Fig. 4 The SimpleModule synchronous module, its LSTS (top left), and a small GALS system (bottom) formed by composing SimpleModule and Crossbar

Σ in state s. State $s' \in S$ is reachable from $s \in S$ if $s \overset{\tau}{\Rightarrow} s'$ for some τ. We denote with $RSS_s(\Sigma)$ the set of states that are reachable from s. The *reachable state space* of Σ is $RSS(\Sigma) = RSS_{\hat{s}}(\Sigma)$.

We only consider in the sequel *stuttering-invariant* systems that can take time while doing nothing. Formally, $\Sigma = (U, S, \rightarrow, \hat{s})$ is stuttering-invariant if $s \xrightarrow{\perp_U} s$ is a transition of Σ for all $s \in RSS(\hat{s})$. Later in the paper we shall also require that systems do not change their state without interacting with their environment, so that $\perp_U$ only labels stuttering transitions.

Figure 3 gives a small synchronous program, written in Esterel [9], and its LSTS representation. Meant to model a very simple communication crossbar, our program has two operation modes, corresponding to the states of the automaton. In state 0, data from input `I1` is output on `O1`, and data from input `I2` is output on `O2`. In state 1, data from `I1` is sent through `O2`, and data from `I2` is sent through `O1`. Mode changes are triggered by the reset signal `R`. Values are abstracted away, so that we are only interested in the presence/absence of signals. The program is stuttering-invariant, but the stuttering transitions have been omitted, for the sake of clarity.

2.4. Parallel composition

The composition of LSTSs is defined by means of *synchronized product*. Given the LSTSs $\Sigma_i = (U_i, S_i, \rightarrow_i, \hat{s}_i)$, $i = \overline{1,2}$, their product is the LSTS:

$$\Sigma_1 \times \Sigma_2 = (U_i \cup U_2, S_1 \times S_2, \rightarrow, (\hat{s}_1, \hat{s}_2))$$

where $(s_1, s_2) \xrightarrow[\Sigma_1 \times \Sigma_2]{r} (s_1', s_2')$ *iff* $\forall i : s_i \xrightarrow[\Sigma_i]{r|_{U_i}} s_i'$. The product operator is associative and commutative, and for all $s_i \in S_i$, $i = \overline{1,2}$ we have:

$$Traces_{\Sigma_1 \times \Sigma_2}(s_1, s_2) = \{\tau_1 \sqcup \tau_2 \mid \tau_i \in Traces_{\Sigma_i}(s_i)\}$$

For instance, the product of the LSTSs Crossbar (of Fig. 3) and SimpleModule (of Fig. 4) is:

The function of SimpleModule is here to produce one `I1` signal and then emit `R`, thus requiring a change of communication mode in Crossbar. One trace of the product is, for

$$\begin{tikzcd} {} \arrow[r] & (0,0) \arrow[loop above, "\mathtt{I2,O2}"] \arrow[r, bend left, "\mathtt{A,I1,O1}"] \arrow[r, bend right, "\mathtt{A,I1,I2,O1,O2}"'] & (0,1) \arrow[loop above, "\mathtt{I2,O2}"] \arrow[r, "\mathtt{A,R}"] & (1,2) \arrow[loop above, "\mathtt{I2,O1}"] \end{tikzcd}$$

instance:

$$(\mathtt{A},\mathtt{I1},\mathtt{O1})(\mathtt{A},\mathtt{R})(\mathtt{I2},\mathtt{O1}) \in \mathit{Traces}_{\mathtt{Crossbar}\times\mathtt{SimpleModule}}((0,0))$$

3. Weakly endochronous systems

The notion of weak endochrony extends the theory of Mazurkiewicz traces to our synchronous setting. Weakly endochronous (WE) systems have trace languages that are closed under commutation of independent reactions, where independence is given by the non-overlapping of supports. To fully take into account the synchronous setting: (1) trace languages are also closed under unification ($\sqcup$) of independent reactions and (2) overlapping non-contradictory reactions can be decomposed into atomic reactions that commute. WE systems satisfy the classical commutation and normal form properties of the Mazurkiewicz traces. Unfortunately, the existing theory could not be used to prove the new results, difficulties arising mainly from the interpretation of the independence alphabet over synchronous reactions. **Hence, the complexity of the proofs, given in [15].**

Definition 1 (weak endochrony). We say that LSTS $\Sigma = (U, S, \rightarrow, \hat{s})$ is weakly endochronous if the following properties are satisfied for all $s, s_1, s_2 \in RSS(\Sigma)$, and for all $r, r_1, r_2 \in \mathit{Reactions}(U)$:

W1. Determinism: $s \xrightarrow{r} s_1 \wedge s \xrightarrow{r} s_2 \Rightarrow s_1 = s_2$

W2. Transitions with disjoint signal support commute and can be joined to form larger transitions. Formally, if $supp(r_1) \cap supp(r_2) = \emptyset$ then:

1. $\left.\begin{array}{l} s \xrightarrow{r_1} s_1 \\ s \xrightarrow{r_2} s_2 \end{array}\right\} \Rightarrow \exists s' : s \xrightarrow{r_1 \vee r_2} s'$
2. $s \xrightarrow{r_1} s_1 \xrightarrow{r_2} s_2 \Rightarrow \exists s' : s \xrightarrow{r_2} s'$

W3. Non-contradictory overlapping reactions can be fragmented into reactions of smaller supports:

$$\left.\begin{array}{l} s \xrightarrow{r_1} s_1 \\ s \xrightarrow{r_2} s_2 \\ r_1 \vee r_2 \textit{ exists} \end{array}\right\} \Rightarrow \exists s' : \begin{tikzcd} & & s_1 \\ s \arrow[r, "r_1 \wedge r_2"] & s' \arrow[ur, "r_1 \backslash r_2"] \arrow[dr, "r_2 \backslash r_1"'] & \\ & & s_2 \end{tikzcd}$$

The **intuition** is that we are looking for systems where (1) all causality is implied by the sequencing of messages on communication channels and (2) all choices are visible as choices over the value (and not present/absent status) of some communication channel. All internal decisions of such a system can therefore be observed or controlled from the exterior even after desynchronization, meaning that there is no ambiguity concerning the construction of the schedulers (the system is self-clocked). Moreover, unlike latency-insensitive and endochronous systems, which ensure scheduling-independence by prohibiting all internal concurrency, WE components only expose meaningful causality and choices. Latency-insensitive systems [6] and endochronous systems [2] satisfy the axioms of weak endochrony. Moreover, unlike classical endochrony, weak endochrony is compositional:

Theorem 1 (Composition). *If Σ_1 and Σ_2 are weakly endochronous LSTSs, then $\Sigma_1 \times \Sigma_2$ is weakly endochronous.*

The remainder of this section is organized as follows: Section 3.1 presents some basic properties of the weakly endochronous systems and defines atomic reactions. Section 3.2 introduces the normal form operator. Intuitive examples are given in the last subsection.

3.1. Atoms

The first step in establishing the relation with Mazurkiewicz trace theory is to define the alphabet of letters and the independence relation. In our case, the letters are the *atomic* reactions, of which all the other reactions are composed. Indeed, *atomic* reactions are those least, but not silent, reactions enabled in a given state of the system. Axiom W3 is the means by which atoms are constructed, by fragmenting larger reactions. The independence relation is the non-overlapping of supports of atoms. Formally, the set of atomic reactions enabled in the state s of Σ is:

$$Atoms_\Sigma(s) = \{r \in Reactions_\Sigma(s) \mid \forall r' \in Reactions_\Sigma(s),\ r \not> r'\}$$

and the set of atoms smaller than a given reaction ρ is:

$$Atoms_\Sigma(s, \rho) = \{r \in Atoms_\Sigma(s) \mid r \leq \rho\}$$

(we will omit Σ from notations whenever possible).

The commutation of independent reactions (and in particular atoms) is governed by the following:

Proposition 1 (Full diamond). *Let $\Sigma = (U, S, \rightarrow, \hat{s})$ be a weakly endochronous LSTS, $s \in RSS(\Sigma)$, and $s \xrightarrow{r_i} s_i$, $i = 1, 2$. Then:*

$$supp(r_1) \cap supp(r_2) = \emptyset \Rightarrow \exists s' : \begin{tikzcd} & s_1 \arrow[dr, "r_2"] & \\ s \arrow[ur, "r_1"] \arrow[rr, "r_1 \vee r_2"] \arrow[dr, "r_2"'] & & s' \\ & s_2 \arrow[ur, "r_1"'] & \end{tikzcd}$$

More generally, the property holds for any number of mutually independent transitions or atoms in a given state, which form a full diamond (with diagonals).

Corollary 1 (Commutation). *Let $\Sigma = (U, S, \rightarrow, \hat{s})$ be a weakly endochronous LSTS, $s \in RSS(\Sigma)$, and $a, b, r_i \in Reactions(U)$, $i \geq 1$ s.t. $supp(a) \cap supp(b) = \emptyset$. Then:*

- $r_1 \ldots r_k a b r_{k+1} \ldots \in Traces_\Sigma(s) \Leftrightarrow$
  $r_1 \ldots r_k b a r_{k+1} \ldots \in Traces_\Sigma(s)$
- $r_1 \ldots r_k a b r_{k+1} \ldots \in Traces_\Sigma(s) \Rightarrow$
  $r_1 \ldots r_k (a \vee b) r_{k+1} \ldots \in Traces_\Sigma(s)$

Let, in addition, $\tau = r_1 r_2 \ldots \in Traces_\Sigma(s)$ and $a \in Reactions_\Sigma(s)$ such that $\forall i : supp(a) \cap supp(r_i) = \emptyset$. Then, $a\tau \in Traces_\Sigma(s)$.

In a given state, the set of atoms indeed generates all possible reactions:

Proposition 2 (Generation). *Let $\Sigma = (U, S, \rightarrow, \hat{s})$ be a weakly endochronous LSTS, and $s \in RSS(\Sigma)$. Then: $Reactions_\Sigma(s) = \{\bigvee_{i=1}^{n} a_i \mid n \geq 0 \wedge \forall i : a_i \in Atoms_\Sigma(s) \wedge \forall i \neq j : supp(a_i) \cap supp(a_j) = \emptyset\}$ Moreover, the decomposition of a reaction into atoms is unique upto permutation.*

Moreover, changing the state without giving additional input does not enable new atoms (or transitions), nor disable existing ones:

Proposition 3 (Atom preservation). *Let $\Sigma = (U, S, \rightarrow, \hat{s})$ be a weakly endochronous LSTS, let $s \in RSS(\Sigma)$, $\rho \in Reactions(U)$, and $r \leq \rho$ such that $s \xrightarrow{r} s'$. Then, $Atoms_\Sigma(s', \rho \setminus r) = Atoms_\Sigma(s, \rho \setminus r)$.*

Finally, if the restriction of a trace to a set of variables corresponds to a valid history, then the restriction to the complementary set of variables is a trace of the given LSTS:

Theorem 2 (Disjoint support). *Let $\Sigma = (U, S, \rightarrow, \hat{s})$ be a weakly endochronous LSTS, $s \in RSS(\Sigma)$, and $V \subseteq U$. Suppose that $\tau, \theta \in Traces_\Sigma(s)$ such that:*

$$\delta(\theta)(v) = \begin{cases} \delta(\tau)(v), & \text{if } v \in V \\ \epsilon, & \text{otherwise} \end{cases}$$

Then, the traces $\tau', \tau'' \in Traces(U)$ defined for all $i \geq 1$ by $\tau'[i] = (\tau[i]_{|V})_{|U}$ and $\tau''[i] = (\tau[i]_{|U \setminus V})_{|U}$ are traces of $Traces_\Sigma(s)$.

Note that, under the hypothesis of the theorem, there exists $\phi \in Traces_\Sigma(s)$ such that:

$$\delta(\phi)(v) = \begin{cases} \delta(\tau)(v), & \text{if } v \in U \setminus V \\ \epsilon, & \text{otherwise} \end{cases}$$

3.2. Normal form

Given an execution of a weakly endochronous system, we can put it in *normal form*, where the atomic operations are re-combined to form largest transitions, so that each atom is executed as early as possible. Like for the Cartier-Foata normal form, putting a trace in normal form keeps unchanged the causal relations between atomic operations (determined by support overlapping). Unlike in the classical trace theory, however, our synchronous setting facilitates the understanding and the manipulations, as the normal form is a synchronous trace, and not a sequence of cliques of atoms/letters. The normal form can be computed from the desynchronized version of an execution. Even more, constructing a normal form execution from a general history χ (one that does not necessarily correspond to a trace) results in an execution τ which is maximal such that $\delta(\tau) \preceq \chi$. This maximal execution is unique upto commutation of independent atoms and any smaller execution can be "completed" to one that is maximal.

3.2.1. Definition

We explain here how the normal form associated with a history is constructed. Let $\Sigma = (U, S, \rightarrow, \hat{s})$ be a weakly endochronous LSTS. Then, for all $s \in RSS(\Sigma)$ and $\rho \in Reactions$ we denote the *unique* maximal reaction smaller than ρ with $\lfloor \rho \rfloor_s = \bigvee_{a \in Atoms(s,\rho)} a$.

Let $\chi \in Histories$ such that $supp(\chi) \subseteq U$. We shall denote with $NF_\Sigma(s, \chi)$ the trace of Σ obtained by starting in state s and performing at each step the maximal reaction enabled by the remainder of the history χ. When no confusion is possible, we will also write $NF(s, \chi)$. Formally, $NF_\Sigma(s, \chi) = r_1 r_2 \ldots$, where $s_1 = s$, $\chi_1 = \chi$, and for all $i \geq 1$: $r_i = \lfloor head_U(\chi_i) \rfloor_{s_i}$, $\chi_{i+1} = \chi_i \setminus \delta(r_i)$, and $s_i \rightarrow^{r_i} s_{i+1}$.

3.2.2. Properties

Our main result states the existence of the normal form, its maximality, and the equifinality of the execution for a given history.

Theorem 3 (Normal form and confluence). *A. Let $\Sigma = (U, S, \rightarrow, \hat{s})$ be a weakly endochronous LSTS. For all $\chi \in$ Histories, $s \in RSS(\Sigma)$, $\tau \in Traces_\Sigma(s)$ such that $\delta(\tau) \preceq \chi$, we have $\delta(\tau) \preceq \delta(NF_\Sigma(s, \chi))$.*

B. Suppose, in addition, that χ is of finite length and $s \overset{NF_\Sigma(s,\chi)}{\Rightarrow} s'$. Then, if τ is chosen such that $\delta(\tau)$ is maximal, we have $\delta(\tau) = \delta(NF_\Sigma(s, \chi))$ and $s \overset{\tau}{\Rightarrow} s'$.

Moreover, any trace that is not maximal under a given history can be completed with transitions:

Proposition 4 (Completion). *Let $\Sigma = (U, S, \rightarrow, \hat{s})$ be a weakly endochronous LSTS, $s \in RSS(\Sigma)$, $\tau \in Traces_\Sigma(s)$, finite, and $\chi \in$ Histories such that $\delta(\tau) \preceq \chi$. Then, if $\delta(\tau) \neq \delta(NF_\Sigma(s, \chi))$, there exists a reaction $r \in Reactions(U)$, $r \neq \bot_U$ such that $\tau r \in Traces_\Sigma(s)$ and $\delta(\tau r) \preceq \chi$.*

Finally, the normal form operator is monotonous and commutes with the limit operator on histories:

Proposition 5 (Monotonicity and limit on histories). *Let $\Sigma = (U, S, \rightarrow, \hat{s})$ be a weakly endochronous LSTS, $s \in RSS(\Sigma)$, and $\chi, \chi', \chi_i \in$ Histories, $i \geq 1$. Then:*

1. *If $\chi' \preceq \chi$ then $\delta(NF_\Sigma(s, \chi')) \preceq \delta(NF_\Sigma(s, \chi))$*
2. *If $(\chi_k)_{k \geq 1}$ is monotonous such that $\bigsqcup_{k \geq 1} \chi_k = \chi$ then $\bigsqcup_{k \geq 1} \delta(NF_\Sigma(s, \chi_k)) = \delta(NF_\Sigma(s, \chi))$*

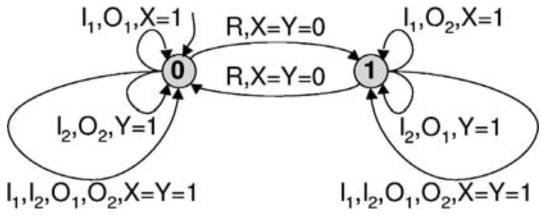


Fig. 5 A weakly endochronous version of the Crossbar module. We shall refer to it as CrossbarWE. Note the added signals X and Y

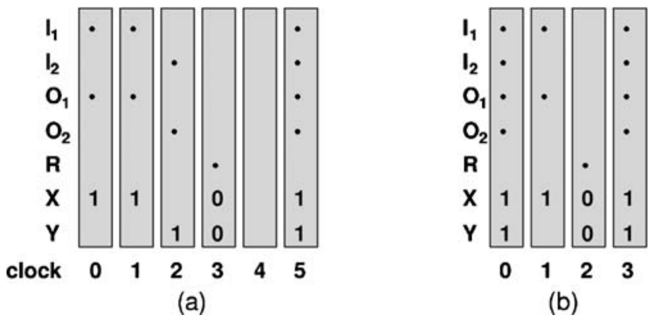


Fig. 6 A trace of the CrossbarWE LSTS (a), and its normal form (b)

3.3. Example

The LSTS of our Crossbar example (Fig. 3) is not weakly endochronous, because in any of the two states one cannot decide *based on the value of a signal* whether the automaton triggers a mode change or performs a "normal operation" transition (axiom W2 is not satisfied). Making the Crossbar weakly endochronous requires the use of supplementary signals, to make the choice/priority between R and I1 or I2 visible as a choice over a value. To this end, we use in Fig. 5 the extra signals X and Y. Note that the concurrency between the transitions I1,O1 and I2,O2 is preserved by the new signalling scheme in state 0 (the concurrency between I1,O2 and I2,O1 is preserved in state 1).

The atomic reactions of CrossbarWE are I1,O1,X=1, I2,O2,Y = 1, and R,X = Y = 0 (in state 0), respectively I1,O2,X = 1, I2,O1,Y = 1, and R,X = Y = 0 (in state 1). All the reactions in a given state are obtained as a union of atoms, for instance:

$$(\mathtt{I1}, \mathtt{I2}, \mathtt{O1}, \mathtt{O2}, \mathtt{X} = \mathtt{Y} = 1) = (\mathtt{I1}, \mathtt{O1}, \mathtt{X} = 1) \sqcup (\mathtt{I2}, \mathtt{O2}, \mathtt{Y} = 1)$$

Non-overlapping atoms are independent, and any trace can be put in normal form by commutation of independent atoms. As Fig. 6 shows, the normal form of a trace can be seen as the most compact execution of the atoms of the initial trace.

System SimpleModule is sequential, therefore weakly endochronous, so that SimpleModule × CrossbarWE is weakly endochronous.

4. Application to GALS systems

Traditionally, the synchronous paradigm has been used in hardware, where the clock-driven execution model and the instantaneous communication abstraction are natural. Later, it has been introduced in the development of safety-critical embedded software, where the deterministic concurrency of the model results in better verification capabilities. Synchronous languages have been developed, along with compilation methods able to generate efficient monolithic implementations in both software and digital circuits.

The implementation of a synchronous specification, however, may have to be *distributed* to some extent. This is obviously true when the target is distributed software. Less obvious, this is increasingly the case in complex microprocessors and Systems-on-a-Chip. As complexity and speed grow, controlling such a chip using a single global clock becomes increasingly difficult. Future large-scale circuits will likely be composed of several timing domains (each domain being either asynchronous, or locally clocked).

Globally Asynchronous Locally Synchronous (GALS) architectures are emerging as the architecture of choice for implementing complex specifications in both hardware and software. Starting from synchronous specifications, our objective is to derive GALS implementations where the synchronous components are connected through bounded lossless FIFOs (which can be easily implemented in both hardware and software). We exemplify on the small system of Fig. 4. The modules SimpleModule and Crossbar are connected here through FIFOs that transmit the signals `I1` and `R` as they are emitted by SimpleModule.[2] As the synchronous modules cannot be directly run in an asynchronous environment, small executives are used to read the inputs and schedule them into sequences of synchronous reactions (thus defining the local clock of each module).

The work presented in this paper aims at automatically synthesizing such executives that are both efficient (in terms of speed and number of exchanged messages) and correct. With the previous definitions, correctness criterion 1 can be rephrased as the following two properties: (1) for any asynchronous run (history) χ of the GALS system, there exists a trace τ of the synchronous model s.t. $\delta(\tau) = \chi$ (recall that the desynchronization morphism δ only retains the sequence of values of each variable) and (2) for any trace τ of the synchronous model, its desynchronization $\delta(\tau)$ is a history of the GALS system. Recall that these properties ensure that the verification of the synchronous model covers all the executions of the GALS system and that the executives (the schedulers) do not restrict the functionality w.r.t. the synchronous model.

Note that not any scheduler is a good one. In our example (Fig. 4), the module SimpleModule produces the outputs `I1` and `R` that are transmitted on separate channels. If Crossbar is scheduled so that `R` is read before an already emitted `I1`, then Crossbar will send the `I1` input over the wrong output channel (unless, of course, two such reading errors occur). Such an execution history does not correspond to a synchronous trace. In more complex examples, such errors may also lead to communication deadlocks where emitted messages remain unread on the FIFOs.

4.1. Semantics-preservation criteria

As the results of Section 3 show, weak endochrony is a scheduling-independence property characterizing a large class of self-clocked synchronous components that can be embedded into GALS systems using very simple schedulers. Weak endochrony generalizes existing scheduling-independence criteria (latency-insensitivity, endochrony) by allowing concurrency between computations of a synchronous component. Thus, it potentially supports the use of lighter, more efficient communication schemes in the development of GALS systems.

[2] For brevity, we considered here data-less programs, so that the FIFOs carry only *present* values. In real examples, however, the signals may also carry more complex data like integers. The signal presence acts in these cases as a *data ready* indicator.

Two more steps need to be taken in order to define development schemes based on weak endochrony. First, we need to define complementary correctness criteria characterizing the synchronous models (formed of weakly endochronous components) whose semantics is preserved in the GALS implementation. Such a semantics-preservation criterion is *weak isochrony*, defined later in this section. Second, we must develop synthesis algorithms able to put general synchronous systems in a weakly endo/isochronous form. This aspect is not covered in the current paper (work in progress).

Our contribution—the definition of weak endochrony and weak isochrony—is done at the level of LSTSs. In this non-causal framework, the operation representing the asynchronous composition through FIFOs of arbitrary length is the upper bound operator on histories $\sqcup$ (recall that $\chi_1, \chi_2 \in$ *Histories* are composable if for every variable v $\chi_1(v)$ is a prefix of $\chi_2(v)$ or $\chi_2(v)$ is a prefix of $\chi_1(v)$). Furthermore, we do not use $\sqcup$ to compose arbitrary executions. More exactly, we only consider deadlock-free executions that complete with empty FIFOs, and to identify such executions we introduce the notion of asynchronous composability:

Definition 2 (asynchronous composability). The synchronous traces τ_i of $\Sigma_i = (U_i, S_i, \rightarrow_i, \hat{s}_i)$, $i = 1, 2$ are asynchronously composable, denoted $\tau_1 \bowtie \tau_2$, if, by definition, $\delta(\tau_1)(v) = \delta(\tau_2)(v)$ for all $v \in U_1 \cap U_2$.

With this notation, we can finally formalize the global semantics preservation criterion given in Section 1 (a variant of that defined by Benveniste et al. [2]):

Criterion 2 (Benveniste et al., 2000). For all $(s_1, s_2) \in RSS(\Sigma_1 \times \Sigma_2)$:

$$\delta(Traces_{\Sigma_1 \times \Sigma_2}(s_1, s_2)) = \{\delta(\tau_1) \sqcup \delta(\tau_2) \mid \tau_i \in Traces_{\Sigma_i}(s_i) \wedge \tau_1 \bowtie \tau_2\}$$

Note that the weakly endochronous LSTSs of Figs. 5 and 4 do not satisfy this criterion: The synchronous traces `(R, X = Y = 0)(I1, O2, X = 1)` of CrossbarWE and `(A, I1)(A, R)` of SimpleModule can be composed asynchronously, but not synchronously. This means that the GALS system of Fig. 4 exhibits more deadlock-free behaviors than the synchronous specification `SimpleModule` $\times$ `CrossbarWE`.

Unfortunately, criterion 2 cannot serve as an effective semantics preservation criterion, being undecidable for LSTSs that are weakly endochronous and finite (**the reader interested in the rather complex proofs of the results of this section is once more referred to [15]**).

Theorem 4 (Undecidability). *Criterion 2 is undecidable, even on finite weakly endochronous LSTS with variables taking their values in finite domains.*

Given this undecidability result, our goal has been to find decidable sufficient conditions for criterion 2 that characterize, at the same time, meaningful classes of LSTSs. The first step in this direction has been to note that criterion 2 can be largely simplified when we apply it (as we want) to systems whose components are weakly endochronous:

Criterion 3. For all $(s_1, s_2) \in RSS(\Sigma_1 \times \Sigma_2)$ and for all $\tau_i \in Traces_{\Sigma_i}(s_i), i = 1, 2$ such that $\tau_1 \bowtie \tau_2$ and $\delta(\tau_1) \sqcup \delta(\tau_2) \neq \epsilon$, the following holds:

$$\exists r_i \in Reactions_\Sigma(s_i), i = 1, 2 : \begin{cases} r_1 \vee r_2 \neq \bot_{U_1 \cup U_2} \\ \delta(r_i) \preceq \delta(\tau_i), i = 1, 2 \\ r_1 \sqcup r_2 \text{ exists} \end{cases}$$

The equivalence between criteria 2 and 3 is important, as it replaces the synchronization of full traces (impossible to compute for infinite traces) with the existence of a pair of synchronizable initial transitions. Nevertheless, the undecidability of criterion 2 on weakly endochronous systems also implies the undecidability of criterion 3 (which is therefore not effective).

4.2. Weak isochrony

Intuitively, the undecidability of criterion 3 is due to the quantification over asynchronously composable traces. To derive a decidable criterion, we over-approximate by quantifying over traces whose *asynchronous prefixes* of length 1 are synchronously composable. Formally, we start by denoting for all $\Sigma = (U, S, \rightarrow, \hat{s})$, and $s \in RSS(s)$ the set of asynchronous prefixes of depth 1: $head_\Sigma(s) = \{head_U(\delta(\tau)) \mid \tau \in Traces_\Sigma(s)\}$. Note that the elements of $head_\Sigma(s)$ are not necessarily reactions of Σ, but that $head_\Sigma(s)$ can be computed for any finite LSTS Σ, for instance through a depth-first search. With this definition, the semantics preservation property that will imply correctness criterion 3 shall be:

Criterion 4 (weak isochrony). Two LSTSs Σ_1 and Σ_2 are weakly isochronous if, by definition, for all $(s_1, s_2) \in RSS(\Sigma_1 \times \Sigma_2)$ and for all $r_i \in head_{\Sigma_i}(s_i), i = 1, 2$ such that $r_1 \sqcup r_2$ exists we have:

$$\exists \overline{r}_i \in Reactions(s_i), i = 1, 2 : \begin{cases} \overline{r}_i \leq r_i, i = 1, 2 \\ \overline{r}_1 \sqcup \overline{r}_2 \text{ exists} \\ \overline{r}_1 \vee \overline{r}_2 \neq \bot_{U_1 \cup U_2} \end{cases}$$

It is obvious that weak isochrony implies criterion 3, and therefore the main result of our paper:

Theorem 5 (correct desynchronization). *Let Σ_1 and Σ_2 be weakly endochronous LSTSs such that (Σ_1, Σ_2) is weakly isochronous. Then, Σ_1 and Σ_2 satisfy Benveniste's correct desynchronization criterion 2.*

Note that weak isochrony can only be used in conjunction with weak endochrony to form a correctness criterion. In the following example the LSTSs are weakly isochronous, but not weakly endochronous, and the semantics preservation criterion is not satisfied:

Figure 7 gives a possible solution to the toy problem defined in Fig. 4. As noted earlier, the LSTSs CrossbarWE and SimpleModule are both weakly endochronous, but not weakly

$$\Sigma_1 : \begin{tikzcd} {} \arrow[r] & \bullet \arrow[r, "a"] & \bullet \arrow[r, "b"] & \bullet \end{tikzcd} \qquad \Sigma_2 : \begin{tikzcd} {} \arrow[r] & \bullet \arrow[r, "b"] \arrow[dr, "a"] & \bullet \arrow[r, "a"] & \bullet \\ & & \bullet & \end{tikzcd}$$

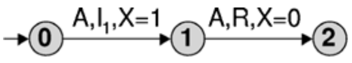


Fig. 7 A version of the SimpleModule LSTS that makes its composition with CrossbarWE weakly isochronous. We shall refer to it as SimpleModuleWI

isochronous. To make them weakly isochronous, we can enrich the interface of SimpleModule as pictured in Fig. 7. Note that the (non-causal) LSTSs do not specify the direction of the added signals `X` and `Y`. Signal `X`, for instance, can be produced by SimpleModuleWI (to inform CrossbarWE about the inputs to read), but can also be produced by `CrossbarWE` (to request a given input). Obviously, the decision is essential in the actual synthesis process, but the aspect is not covered in the current paper.

5. Comparison with existing work

5.1. Weak endochrony and trace theory

As the results of Section 3 show, weak endochrony is indeed an extension to a synchronous setting of the classical trace theory. More exactly, if we consider the formalization of [8], chapter 11, the dependence alphabet is interpreted in our case over synchronous reactions, while the dependence relation is given by a syntactic relation over such reactions (the non-overlapping of supports). The traces of a weakly endochronous system correspond to real dependency graphs having atoms as vertices, and the trace language of a system is closed under commutation of independent atoms.

Easily defined, the extension is nevertheless non-trivial. The new *synchrony* relation leads, when joined to the classical *independence* and *causality*, to stronger results, but also complexifies proofs. This is particularly obvious in the case of the normal form result: The formulation is in our case stronger, in the sense where the normal form of a trace is indeed another trace of the weakly endochronous system, and not a decomposition of the initial trace into a sequence of cliques of independent letters.[3] Moreover, weak endochrony also covers state-related determinism aspects through its strong confluence properties (Theorem 3). On the other side, the proof of the normal form theorem has been challenging, as the classical results of [8], which do not cover the synchrony relation, cannot be used.

Going beyond classical trace properties, our normal form operator is also continuous with respect to the complete partial order structures on the sets of traces and histories (cf. Proposition 5). We can therefore consider that networks of weakly endochronous components satisfy a relational form of the Kahn principle [14] where the determinism and continuity of the normal form operator on components implies the determinism (upto commutation of independent operations) of the system evolution for a given history.

[3] The relation between the two normal forms is straightforward: the decomposition as a sequence of cliques (the trace theory normal form) is given by the sequence of reactions of the weakly endochronous normal form, each reaction being transformed into the clique of atoms that generate it.

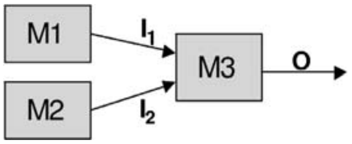


Fig. 8 Module `M3` multiplexes the inputs from `M1` and `M2` by alternating them onto `O`. This example is not handled well by the latency-insensitive approach

5.2. Desynchronization results

As mentioned in Section 1.2, the *latency-insensitive paradigm* [6] features no concurrency, nor execution modes. The communication protocols of a latency-insensitive system simulate a single-clocked system where all signals are transmitted during each reaction. This is inefficient, but easily supports causality and simple synthesis algorithms. By comparison, our approach takes into account execution modes and independence between computations, and allows multi-rate computation. This means that it potentially supports lighter communication protocols that minimize communication and power consumption. The intuition can be given on the example of Fig. 8, where we assume that `M3` multiplexes the inputs from `M1` and `M2` by alternating them onto `O`. The process is fully deterministic, but in a latency-insensitive encoding the communication lines `I1`, `I2`, and `O` must work at the same rate. This means that half of the communication effort on `I1` and `I2` is spent transmitting explicit *absent* events.

The *endo/isochronous systems* of Benveniste et al. [2, 3] take into account execution modes and independence between system components. Thus, the endo/isochronous encoding of the example in Fig. 8 requires no explicit *absent* events. On the other hand, the approach cannot handle the independence between computations at component level. It is therefore impossible to represent the concurrency of our initial Crossbar module (its endochronous encoding must remove all concurrency in each state, and therefore must completely synchronize the inputs `I1` and `I2` by using extra communication lines).

Our work improves over the endo/isochronous approach by allowing operations within a component to run independently when no synchronization is necessary. Like in our Crossbar/SimpleModule example, the following two systems are both weakly endochronous and weakly isochronous, meaning that no further synchronization is necessary:

However, making Σ_2 endochronous, and then isochronous with Σ_1 would require the removal of the non-determinism either by removing transitions of Σ_2 or by introducing supplementary signalization channels.

Weak endochrony generalizes endochrony, which in turn generalizes the notion of latency insensitivity. On the other hand, neither endochrony, nor weak endochrony take into account causality in the computation of reactions, and efficient synthesis algorithms have yet to be defined for both of them. Last, but not least, weak endochrony is compositional, while endochrony is not.[4]

In a *Kahn process network*, the input/output determinism of each component implies the determinism of the global system, and thus the independence from the scheduling scheme is guaranteed. 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. By comparison, weakly endochronous systems also guarantee the deterministic resynchronization upto commutation, but in a non-causal setting. Thus, non-determinism can be used (in the more concrete causal model, for instance in order to model some timing-

[4] In consequence, synthesizing endo-isochronous communication protocols for systems composed of more than 2 synchronous components involves "endochronization" steps resulting in heavy synchronization.

dependent behavior) as soon as the environment is informed about the non-deterministic internal decisions. As mentioned in the previous section, weak endochrony can be seen as a generalization of the Kahn processes to a relational (non-causal) setting, giving the actual implementations a supplementary degree of freedom that can be exploited by more flexible protocols.

5.3. Weak endochrony and speed independence

Weakly endochronous LSTSs being self-clocked, we can consider them not only as synchronous specifications, but also as asynchronous finite state machines allowing multiple variable changes on each transition. From this point of view, our work is closely related with previous work on (extended) burst-mode circuits, like that of Yun and Dill [17]. By comparison, our approach is not hardware-centric, nor takes into account I/O causality. On the other hand, weak endochrony can be viewed as a generalization of the extended burst-mode machines, because it allows concurrency between independent transitions (bursts), while not specifying/constraining the computation of the transitions.

In fact, weak endochrony can be considered as a generalization of the notion of speed-independence [7] to multiple-changes asynchronous automata. Indeed, one atomic transition of an LSTS can disable another only if the two transitions are contradictory (present with different values on some variable). An atomic transition that is enabled at some point under a given history will eventually fire (cf. Theorem 3, which states the equifinality of all maximal executions). This property is a form of semi-modularity.

6. Conclusion

In this paper we introduced the notion of weak endochrony and we explained how it can be used to advance the state of the art in the deployment of synchronous specifications over GALS architectures. Weak endochrony generalizes over previous work on endochronous and latency-insensitive systems by introducing a notion of independence (derived from the classical trace theory) between operations within a synchronous component. This potentially allows the use of lighter, more efficient synchronization schemes in the synthesis of GALS implementations. Weak endochrony and weak isochrony form a criterion guaranteeing the correct distribution of synchronous specifications in the sense of Benveniste et al. [3]. Moreover, the criterion can be decided on general finite LSTSs.

The current paper only represents a first step in our quest for *effective* deployment methods. Our future research directions will include (1) the definition of efficient synthesis algorithms ensuring the properties of weak endochrony and weak isochrony and (2) the extension of the model to take into account causality (and thus support the synthesis of actual GALS implementations).

References

1. MathWorks—Simulink. http://www.mathworks.com/products/simulink/
2. Benveniste A, Caillaud B, Le Guernic P (2000) Compositionality in dataflow synchronous languages: Specification and distributed code generation. Inform Computation 163:125–171
3. Benveniste A, Carloni L, Caspi P, Sangiovanni-Vincentelli A (2003) Heterogenous reactive systems modeling and correct-by-construction deployment. In: Proceedings of EMSOFT'03. Philadelphia, Pennsylvania, USA

4. Benveniste A, Caspi P, Edwards S, Halbwachs N, Le Guernic P, de Simone R (2003) The synchronous languages twelve years later. In: Proceedings of the IEEE
5. Berry G, Gonthier G (1992) The esterel synchronous programming language: Design, semantics, implementation. Sci Comp Program 19(2):87–152
6. Carloni C, McMillan K, Sangiovanni-Vincentelli A (2001) The theory of latency-insensitive design. IEEE Trans Comp-Aided Des Integ Cir Syst 20(9):1059–1076
7. Cortadella J, Kishinevsky M, Kondratiev A, Lavagno L, Yakovlev A (2002) Logic synthesis of asynchronous controllers and interfaces. Springer
8. Diekert V, Rozenberg G (eds.) (1995) The book of traces. World Scientific
9. Halbwachs N (1993) Synchronous programming of reactive systems. Kluwer Academic Publishers
10. Halbwachs N, Caspi P, Raymond P, Pilaud D (1991) The synchronous dataflow programming language lustre. In: Proceedings of the IEEE 79:1305–1320
11. Harel D (1987) Statecharts: A visual formulation for complex systems. Science of Computer Programming 8(3):231–274
12. Kopetz H (1997) Real-time systems, design principles for distributed embedded applications. Kluwer Academic Publishers
13. Le Guernic P, Gauthier T, Le Borgne M, Le Maire C (1991) Programming real-time applications with Signal. In: Proceedings of the IEEE, Vol. 79, pp. 1321–1336
14. Lynch N, Stark E (1989) A proof of the kahn principle for input/output automata. Inform Comput 82(1):81–92
15. Potop-Butucaru D, Caillaud B, Benveniste A (2004) Concurrency in synchronous systems. RR 5110, INRIA. http://www.inria.fr/rrrt/rr-5110.html
16. Potop-Butucaru D, de Simone R, Talpin J-P (2005) The synchronous hypothesis and synchronous languages. In: R. Zurawski (ed.), The Embedded Systems Handbook. CRC Press. to appear
17. Yun K, Dill D (1999) Automatic synthesis of extended burst-mode circuits. IEEE Trans Comp-Aided Des Integ Cir Syst 18(2):101–132