From Synchrony to Asynchrony*

Albert Benveniste, Benoît Caillaud, and Paul Le Guernic

Irisa/Inria, Campus de Beaulieu, 35042 Rennes cedex, France
firstname.lastname@irisa.fr
Web: http://www.irisa.fr/

Abstract. We present an in-depth discussion of the relationships between synchrony and asynchrony. Simple models of both paradigms are presented, and we state theorems which guarantee *correct desynchronization*, meaning that the original synchronous semantics can be reconstructed from the result of this desynchronization. Theorems are given for both the desynchronization of single synchronous programs, and for networks of synchronous programs to be implemented using asynchronous communication. Assumptions for these theorems correspond to proof obligations that can be checked on the original synchronous designs. If the corresponding conditions are not satisfied, suitable synchronous mini-programs which will ensure correct desynchronization can be composed with the original ones. This can be seen as a systematic way to generate "correct protocols" for the asynchronous distribution of synchronous designs. The whole approach has been implemented, in the framework of the SACRES project, within the SILDEX tool marketed by TNI, as well as in the SIGNAL compiler.

1 Introduction

Synchronous programming [5, 10, 14] has been proposed as an efficient approach for the design of reactive and real-time systems. It has been widely publicized, using the idealized picture of "zero time" computation and instantaneous broadcast communication [9]. Efficient techniques for code generation and verification have resulted [10, 20, 5, 15].

Criticisms have been addressed to this approach. It has been argued that, very frequently, real-life architectures do not obey the ideal model of perfect synchrony. Counter-examples are numerous: operating systems with multi-threading or multitasking, distributed architectures, asynchronous hardware, etc.

However, similarities and formal links between synchrony and asynchrony have already been discussed in the literature, thus questioning the oversimplified vision of "zero time" computation and instantaneous broadcast communication. Early paper [6] informally discussed the link between perfect synchrony and token-based asynchronous data-flow networks, see in particular section V therein. The first formal and deep study can be found in [13]. It establishes a precise

* This work is or has been supported in part by the following projects: Esprit R&D-SACRES (Esprit project EP 20897), Esprit LTR-SYRF (Esprit project EP 22703).

Jos C.M. Baeten, Sjouke Mauw (Eds.): CONCUR'99, LNCS 1664, pp. 162–177, 1999.

relation between so-called well-clocked synchronous functional programs and the subset of Kahn networks amenable to "buffer-less" evaluation.

Distributed code generation from synchronous programs requires to address the issue of the relationship between synchrony and asynchrony in a different way. Mapping synchronous programs to a network of automata, communicating asynchronously via unbounded FIFOs, has been implemented for the Lustre language and formalized in [12]. Mapping SIGNAL programs to distributed architectures was proposed in [19, 4], based on an early version of the theory we present in this paper. The SYNDEX tool [22, 21] also implements a similar approach. Recent work [11] on the POLIS system proposes to reuse the "constructive semantics" approach for the ESTEREL synchronous language, with CFSM (Codesign Finite State Machines) as a model for synchronous machines which can be desynchronized; this can be seen as a refinement of [13], although the referred model of asynchrony is not fully stated.

Independently, another approach relating synchrony and asynchrony has been followed. In [7, 18] it is shown how *nondeterministic* SIGNAL programs can be used to model asynchronous communication media such as queues and buffers. *Reactive Modules* [1] were proposed as a synchronous language for hardware modeling, in which asynchrony is emulated by the way of nondeterminism. Although this is of interest, we believe this approach is not suited to the analysis of true asynchrony, in which no notion of global synchronization state is available, unlike for synchrony.

In this paper we provide an extensive, in depth, analysis of the links between synchrony and asynchrony. Our vision of asynchrony encompasses distributed systems, in which no global synchronization state is available, and communications/actions are not instantaneous. This extension allows us to handle incomplete designs, specifications, properties, architectures, and executable programs, in a unified framework, for both synchronous and asynchronous semantics.

In section 2 we informally discuss the essentials of synchrony and asynchrony. Synchronous Transition Systems are defined in section 3, and their asynchronous counterpart is defined in section 4, where also desynchronization is formally defined. The rest of the paper is devoted to the analysis of desynchronization and its inverse, namely resynchronization.

2 The Essentials of the Synchronous Paradigm

There have been several attempts to characterize the essentials of the synchronous paradigm [5, 14]. With some experience and after many attempts to address the issue of moving from synchrony to asynchrony (and back), we feel the following features are indeed essential for characterizing this paradigm:

1. Programs progress via an infinite sequence of *reactions*: $P = R^\omega$, where R denotes the family of possible reactions[1].

[1] In fact, "reaction" is a slightly restrictive term, as we shall see in the sequel that "reacting to the environment" is not the only possible kind of interaction a synchronous system may have with its environment.

2. Within a reaction, decisions can be taken on the basis of the *absence* of some events, as exemplified by the following typical statements, taken from ESTEREL, LUSTRE, and SIGNAL respectively:
```
present S else 'stat'
y = current x
y := u default v
```
The first statement is self-explanatory. The "`current`" operator delivers the most recent value of `x` at the clock of the considered node, it thus has to test for absence of `x` before producing `y`. The "`default`" operator delivers its first argument when it is present, and otherwise its second argument.
3. Parallel composition is given by taking the pairwise conjunction of associated reactions, whenever they are composable: $P_1 \| P_2 = (R_1 \wedge R_2)^\omega$. Typically, if specifying is the intention, then the above formula is a perfect definition of parallel composition. In contrast, if programming is the intention, then the need for this definition to be compatible with an operational semantics complicates very much the "when it is defined" prerequisite[2].

Of course, such a characterization of the synchronous paradigm makes the class of synchronous formalisms much larger than usually considered. But it has been our experience that these were the key features for the techniques we have developed so far. Clearly, this calls for a simplest possible formalism with the above features, on which fundamental questions should be investigated: The design of the STS formalism[3] described in the next section has been guided by these objectives.

Keeping in mind the essentials of the synchronous paradigm, we are now ready to discuss informally how asynchrony relates to synchrony. Referring to points 1, 2, and 3 above, the following can be stated about asynchrony:

1. Reactions cannot be observed any more: since no global clock exists, global synchronization barriers which indicate the transition from one reaction to the next one are no more observable. Instead, a reliable communication medium is assumed, in which messages are not lost, and for each individual channel, messages are sent and received in the same order. We call a *flow* the totally ordered sequence of values sent or received on a given communication channel.
2. Absence cannot be detected, and thus cannot be used to exercise control.
3. Composition occurs by means of unifying each flow shared between two processes. This models in particular the communications via asynchronous unbounded FIFOs, such as those in Kahn networks. Rendez-vous type of communication can also be abstracted in this way.

Synchrony and asynchrony are formalized in sections 3 and 4, respectively. Section 7 details how these results can be put into practice.

[2] For instance, most of the effort related to the semantics of ESTEREL has been directed toward solving this issue satisfactorily [10].

[3] We thank Amir Pnueli for having proposed this formalism, in the course of the SACRES research project, as a minimal framework capturing the paradigm of perfect synchrony.

3 Synchronous Transition Systems (STS)

Synchronous Transition Systems (STS). We assume a vocabulary $\mathcal{V}$ which is a set of typed variables. All types are implicitly extended with a special element $\perp$, interpreted as *absence*. Among the types we consider, there are the type of *pure signals* with domain $\{\mathrm{T}\}$, and the *boolean* type with domain $\{\mathrm{T}, \mathrm{F}\}$ (recall both types are extended with the distinguished element $\perp$). We define a *state* s to be a type-consistent interpretation of $\mathcal{V}$, assigning a value to each variable. We denote by S the set of all states. For a subset of variables $V \subseteq \mathcal{V}$, a V-state is a type-consistent interpretation of V. Thus a V-state s assigns a value $s[v]$ to each variable v in set V; the tuple of values assigned to the set of variables V is denoted by $s[V]$.

We define a *Synchronous Transition System* (STS) to be a tuple $\Phi = \langle V, \Theta, \rho \rangle$ consisting of the following components: V is a finite set of typed *variables*, Θ is an assertion on V-states characterizing the set of *initial states* $\{s | s \models \Theta\}$ and ρ is the *transition relation* relating past and current V-states, s^- and s, by referring to both past[4] and current values of variables in V. For example the assertion $x = x^- + 1$ states that the value of x in s is greater by 1 than its value in s^-. If $(s^-, s) \models \rho$ then we say that state s^- is a *ρ-predecessor* of state s.

Runs. A *run* $\sigma : s_0, s_1, s_2, \ldots$ is a sequence of states such that $s_0 \models \Theta \ \bigwedge \ \forall i > 0\ ,\ (s_{i-1}, s_i) \models \rho$.

Composition. The *composition* of two STS $\Phi = \Phi_1 \parallel \Phi_2$ is defined as follows: $\Phi = \langle V = V_1 \cup V_2,\ \Theta = \Theta_1 \wedge \Theta_2,\ \rho = \rho_1 \wedge \rho_2 \rangle$. The composition is thus the pairwise conjunction of initial and transition relations. It should be noticed that, in STS composition, interaction occurs through common variables only.

Notations for STS. For the convenience specification, STS will have a set of *reactive* variables written $V_{\mathbf{r}}$, implicitly augmented with associated *auxiliary* variables: the whole constitutes the set V of variables. We shall use the following generic notations in the sequel:

- $b, c, v, w, \ldots$ denote reactive variables, and b, c are used to refer to variables of boolean type.
- for v a variable, $h_v \in \{\mathrm{T}, \perp\}$ denotes its *clock*: $[h_v \neq \perp] \Leftrightarrow [v \neq \perp]$
- for v a reactive variable, ξ_v denotes its associated *state* variable, defined by:

  $$\textbf{if } h_v \textbf{ then } \xi_v = v \textbf{ else } \xi_v = \xi_v^-$$

  Values can be given to $s_0[\xi_v]$ as part of the initial condition. Then, ξ_v is always present after the first occurrence of v. Finally, $\xi_{\xi_v} = \xi_v$, therefore "state variables of state variables" need not be considered.

[4] Usually, variables and *primed* variables are used to refer to current and *next* states. This is equivalent to our present notation. We have preferred to consider s^- and s, just because the formulas we shall write mostly involve current variables, rather than past ones. Using the standard notation would have resulted in a burden of primed variables in the formulas.

As modularity is desirable, every STS should be permitted to do nothing while its environment is possibly working. This feature has been yet identified in the literature and is known as *stuttering* invariance or robustness [16, 17]. For a STS Φ, stuttering invariance is defined as follows: If $\sigma = s_0, s_1, s_2, \ldots$ is a run of Φ, so is

$$\sigma' = s_0, \underbrace{\perp_{s_0}, \ldots, \perp_{s_0}}_{0 \leq \#\{\perp_{s_0}\} < \infty}, s_1, \perp_{s_1}, \ldots, \perp_{s_1}, s_2, \perp_{s_2}, \ldots, \perp_{s_2}, \ldots$$

where, for s an arbitrary state, symbol $\perp_s$ denotes the *silent state* associated with s, defined by

$$\forall v \in V_{\mathbf{r}} : \begin{cases} \perp_s[v] = \perp \\ \perp_s[\xi_v] = s[\xi_v] \end{cases}$$

meaning that state variables are kept unchanged whenever their associated reactive variables are absent. It should be noticed that stuttering invariance allows for runs possessing only a finite number of present states. We shall require in the sequel that all STS we consider are stuttering invariant. They should indeed satisfy: $(s^-, s) \models \rho \Rightarrow (s^-, \perp_{s^-}) \models \rho$ and $(\perp_{s^-}, s) \models \rho$. When this condition is not satisfied, we extend ρ minimally so that stuttering invariance is satisfied. By convention, we shall simply write $\perp$ instead of $\perp_s$ when mentioning a particular state s is not required.

4 Desynchronizing STS, and Two Fundamental Problems

From the definition of a run of a STS, we can say that a run is a sequence of tuples of values in domains extended with the extra symbol $\perp$. Desynchronizing a run amounts to discarding the synchronization barriers defining the successive reactions. Hence, for each variable $v \in V$, we only know the ordered sequence of *present* values. Thus desynchronizing a run amounts to mapping a *sequence of tuples* of values in domains extended with the extra symbol $\perp$, into a *tuple of sequences* of present values, one sequence per variable. This is formalized below.

For $\sigma = s_0, s_1, s_2, \ldots$ a run of Φ, we decompose state s_k as $s_k = (s_k[v])_{v \in V}$. Thus we can rewrite run σ as follows: $\sigma = (\sigma[v])_{v \in V}$, where $\sigma[v] = s_0[v]$, $s_1[v]$, $\ldots$, $s_k[v]$, ... Now, each $\sigma[v]$ is compressed by deleting those $s_k[v]$ that are equal to $\perp$. Formally, let $k_0, k_1, k_2, \ldots$ be the subsequence of $k = 0, 1, 2, \ldots$ such that $s_k[v] \neq \perp$. Then we set: $\sigma^a = (\sigma^a[v])_{v \in V}$, where $\sigma^a[v] = s_{k_0}[v]$, $s_{k_1}[v]$, $s_{k_2}[v]$, ... This defines our *desynchronization mapping* $\sigma \longmapsto \sigma^a$, and each $\sigma^a[v] = s_{k_0}[v], s_{k_1}[v], s_{k_2}[v], \ldots$ is called a *flow* in the sequel.

The asynchronous abstraction of a STS $\Phi = \langle V, \Theta, \rho \rangle$, is defined as follows:

$$\Phi^a =_{\text{def}} \langle V, \Sigma^a \rangle \;, \tag{1}$$

where Σ^a is the family of all (asynchronous) runs σ^a, with σ ranging over the set of (synchronous) runs of Φ. For $\Phi_i = \langle V_i, \Theta_i, \rho_i \rangle, i = 1, 2$, we define:

$$\Phi_1^a \parallel_a \Phi_2^a =_{\text{def}} \langle V, \Sigma^a \rangle \text{ , where } \begin{cases} V = V_1 \cup V_2 \\ \Sigma^a = \Sigma_1^a \;\wedge^a\; \Sigma_2^a \end{cases} \tag{2}$$

and $\wedge^a$ denotes conjunction of sets of asynchronous runs, which we define now. For $\sigma_i^a \in \Sigma_i^a, i = 1, 2$, we say that σ_1^a and σ_2^a are *unifiable,* written $\sigma_1^a \bowtie^a \sigma_2^a$, if the following condition holds: $\forall v \in V_1 \cap V_2 \; : \; \sigma_1^a[v] = \sigma_2^a[v]$. If σ_1^a and σ_2^a are unifiable, then we define $\sigma^a =_{\text{def}} \sigma_1^a \;\wedge^a\; \sigma_2^a$ as:

$$\begin{aligned} \forall v \in V_1 \cap V_2 : \sigma^a[v] &= \sigma_1^a[v] = \sigma_2^a[v] \\ \forall v \in V_1 \setminus V_2 : \sigma^a[v] &= \sigma_1^a[v] \\ \forall v \in V_2 \setminus V_1 : \sigma^a[v] &= \sigma_2^a[v] \end{aligned}$$

Finally, Σ^a is the set of the so defined σ^a. Thus asynchronous composition proceeds via unification of shared flows.

Synchrony vs. Asynchrony? At this point two natural questions arise, namely:

Question 1 (Desynchronizing a Single STS*).* **Is resynchronization feasible and uniquely defined?** More precisely, is it possible to reconstruct uniquely a synchronous run σ of our STS from a desynchronized run σ^a?

Question 2 (Desynchronizing a Communication). **Does communication behave equivalently for both the synchronous and asynchronous compositions?** More precisely, does the following property hold:

$$\Phi_1^a \parallel_a \Phi_2^a = (\Phi_1 \parallel \Phi_2)^a \;\; ? \tag{3}$$

If question 1 had a positive answer, then we could desynchronize a run of the considered STS, and then still recover the original synchronous run. Thus a positive answer to question 1 would guarantee that the synchronous semantics is preserved when desynchronization is performed on a single STS.

On the other hand, if question 2 had a positive answer, then we could interpret our STS composition equivalently as synchronous or asynchronous.

Unfortunately, neither 1 nor 2 have positive answers in general, due to the possibility of exercising control by the way of absence in synchronous composition $\parallel$. In the following section, we show that questions 1 and 2 have positive answers under certain sufficient conditions, in which the two notions of *endochrony* (for point 1) and *isochrony* (for point 2) play a central role.

5 Endochrony and Re-synchronization

5.1 Formal Results

In this section, we use notations from section 3. For an STS $\Phi = \langle V, \Theta, \rho \rangle$, and s a reachable state of Φ, the clock-abstraction of s (denoted by s^h) is defined as follows:

$$\forall v \in V : \;\; s^h[v] \in \{\bot, \top\}, \text{ and } s^h[v] = \bot \Leftrightarrow s[v] = \bot \tag{4}$$

For a STS $\Phi = \langle V, \Theta, \rho \rangle$, s^- a reachable state for Φ, and $W' \subseteq W \subseteq V$, we say that W' is a *clock inference of* W *given* s^-, written $W' \hookrightarrow_{s^-} W$, if for each state s of Φ, reachable from s^-, knowing the presence/absence and actual value carried by each variable belonging to W', allows us to determine exactly the presence/absence of each variable belonging to W. In other words $s[W']$ uniquely determines $s^h[W]$.

If both $W' \hookrightarrow_{s^-} W_1$ and $W' \hookrightarrow_{s^-} W_2$ hold, then $W' \hookrightarrow_{s^-} (W_1 \cup W_2)$ follows, thus there exists a greatest W such that $W' \hookrightarrow_{s^-} W$ holds. Hence we can consider the unique maximal increasing sequence of subsets of V, for a given s^-,

$$\emptyset = V(0) \hookrightarrow_{s^-} V(1) \hookrightarrow_{s^-} V(2) \hookrightarrow_{s^-} \dots \tag{5}$$

in which, for each $k > 0$, $V(k)$ is the greatest set of variables such that $V(k-1) \hookrightarrow_{s^-} V(k)$ holds. As $\emptyset = V(0)$, $V(1)$ consists in the subset of variables that are present as soon as the considered STS gets activated or which are always absent in successor states of s^-. Of course sequence (5) must become stationary at some finite $k_{\max}$: $V(k_{\max}+1) = V(k_{\max})$. In general, we only know that $V(k_{\max}) \subseteq V$. Sequence (5) is called the *synchronization sequence* of Φ in state s^-.

Definition 1 (Endochrony). *A* STS Φ *is said to be* endochronous *if, for each reachable state* s^- *of* Φ, $V(k_{\max}) = V$, *i.e., if the synchronization sequence:*

$$\emptyset = V(0) \hookrightarrow_{s^-} V(1) \hookrightarrow_{s^-} V(2) \hookrightarrow_{s^-} \dots \text{ converges to } V \tag{6}$$

Condition (6) expresses that presence/absence of all variables can be inferred *incrementally* from already known values carried by present variables and state variables of the STS in consideration. Hence no test for presence/absence on the environment is needed. The following theorem justifies our approach:

Theorem 1. *Consider a* STS $\Phi = \langle V, \Theta, \rho \rangle$.

1. *Conditions (a) and (b) given below are equivalent:*
   (a) Φ *is endochronous.*
   (b) *For each* $\delta \in \Sigma^a$, *we can reconstruct the corresponding synchronous run* σ *such that* $\sigma^a = \delta$, *in a unique way up to silent reactions.*
2. *Let us assume* Φ *is endochronous and stuttering invariant. If* $\Phi' = \langle V, \Theta, \rho' \rangle$ *is another endochronous and stuttering invariant* STS *then*

$$(\Phi')^a = \Phi^a \Rightarrow \Phi' = \Phi \tag{7}$$

Proof. We prove successively points 1 and 2.

1. We consider a previous state s^- and prove the result by induction. We pick out a $\delta \in \Sigma^a$, and assume for the moment that it can be decomposed in:

$$\underbrace{s_1\,,\, s_2\,,\, \dots\,,\, s_n}_{\text{initial segment of } \sigma \text{ of length } n} \qquad \delta_n \tag{8}$$

i.e., into a sequence of length n, made of non-silent states s_i (the head of the synchronous run σ we wish to reconstruct), followed by the tail of the asynchronous run δ, which we denote by δ_n, and we assume that such a decomposition is unique. Then we claim that

$$\text{(8) is also valid with } n \text{ substituted by } n+1. \tag{9}$$

To prove (9), we note that, whenever STS Φ is activated in the considered state, the presence/absence of each variable belonging to $V(1)$ is known. By assumption, the state $s^h_{n+1}[V(1)]$ resulting from clock-abstraction, having $V(1)$ as variables, is uniquely determined. In the sequel we write $s^h_{n+1}(1)$ for short instead of $s^h_{n+1}[V(1)]$. Thus, presence/absence of variables for state $s_{n+1}(1)$ is known, the values carried by present variables still have to be determined.
For any $v \in V_1$, we simply take the value carried by the minimal element of the sequence associated with variable v in δ_n. Values carried by corresponding state variables are updated accordingly. Thus we know the presence or absence and the value of each individual variable in state $s_{n+1}(1)$.
Next we move on constructing $s_{n+1}(2)$. From $s_{n+1}(1)$ we know $s^h_{n+1}(2)$. Thus we know how to split V_2 into present and absent variables for the considered state. We pick up the present ones, and repeat the same argument as before to get $s_{n+1}(2)$.
Repeating this argument until $V(k) = V$ for some finite k (by endochrony assumption), proves claim (9).
Given the initial condition for δ, we get from (9), by induction, the desired proof that *(a)* $\Rightarrow$ *(b)*.
We shall now prove *(b)* $\Rightarrow$ *(a)*. We assume that Φ is not endochronous, and show that condition (b) cannot be satisfied. If Φ is not endochronous, there must be some reachable state s^- for which sequence (6) does not converge to V. Thus, again, we pick out a $\delta \in \Sigma^a$, decomposed in the same way as in formula (8):

$$\underbrace{s_1\,,\, s_2\,,\, \ldots\,,\, s_n}_{n-\text{initial segment of } \sigma}\ \ \delta_n$$

and we assume in addition that $s_n = s^-$, the given state for which endochrony is violated. We now show that (9) is not satisfied. Let $k_* \geq 0$ be the smallest index such that $V(k) = V(k+1)$, we know $V_{k_*} \neq V$. Thus we can apply the algorithm of case 1 for reconstructing the reaction, until variables of V_{k_*}. Then presence/absence for variables belonging to $V \setminus V_{k_*}$ cannot be determined based on the knowledge of variables belonging to V_{k_*}. This means that there exist several possible extensions of $s^h_{n+1}(k_*+1)$ and the $(n+1)$-th reaction is not determined in a unique way. Hence condition (b) does not hold.

2. Let us assume Φ is endochronous, and consider Φ' as in point 2 of the theorem. As both Φ and Φ' are stuttering invariant, point 2 is an immediate consequence of point 1. ◇

Comments.

1. Endochrony is not decidable in general. However, it is decidable for sts only involving variables with finite domains of values, and model checking can be used for that. For general sts, model checking can be used, in combination with abstraction techniques. The case of interest is when the chain $V(0), V(1), \ldots$ does not depend upon the particular state s^-, and we write simply $V(k) \hookrightarrow V(k+1)$ in this case. This abstraction yields to a sufficient condition of endochrony.
2. The proof of this theorem in fact provides an effective algorithm for the on-the-fly reconstruction of the successive reactions from a desynchronized run of an endochronous program.

(Counter-)examples.

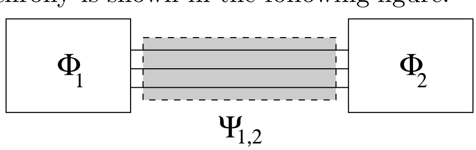
Examples:

- a single-clocked sts.
- sts "**if** b = T **then** *get* u", where b, u are the two inputs, and b is boolean. The clock of b coincides with the activation clock for this sts, and thus $V(1) = \{b\}$. Then, knowing the value for b indicates whether or not u is present, thus $V(2) = \{b, u\} = V$.

Counter-example: sts "**if** ([**present** a] $\|$ [**present** b]) **then** ..." is not endochronous, as the environment is free to offer any combination of presence/absence for the two inputs a, b. Thus $\emptyset = V(0) = V(1) = V(2) = \ldots \subsetneq V$, and endochrony does not hold.

5.2 Practical Consequences

A first use of endochrony is shown in the following figure:

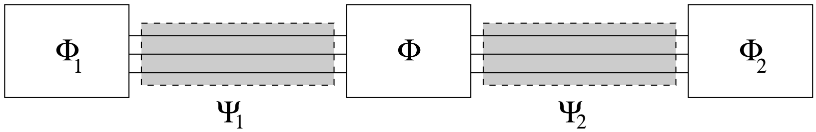


In this figure, a pair (Φ_1, Φ_2) of sts is depicted, with W as the set of shared variables. Their composition is rewritten as follows: $\Phi_1 \| \Phi_2 = \Phi_1 \| \Psi_{1,2} \| \Phi_2$, where $\Psi_{1,2}$ is the restriction of $\Phi_1 \| \Phi_2$ to W, hence $\Psi_{1,2}$ models a synchronous communication medium. We obtain by using property $\Phi \| \Phi = \Phi$ for every sts Φ:

$$\Phi_1 \| \Phi_2 = \underbrace{(\Phi_1 \| \Psi_{1,2})}_{\widetilde{\Phi}_1} \| \underbrace{(\Psi_{1,2} \| \Phi_2)}_{\widetilde{\Phi}_2} = \widetilde{\Phi}_1 \| \widetilde{\Phi}_2 \quad (10)$$

This model of communication medium $\Psi_{1,2}$ is endochronous, and composition $\Phi_1 \| \Phi_2$ is implemented by the (equivalent) composition $\widetilde{\Phi}_1 \| \widetilde{\Phi}_2$. Since all runs of $\Psi_{1,2}$ are also runs of $\widetilde{\Phi}_1$ and the former is endochronous, then communication can be equivalently implemented according to perfect synchrony or full asynchrony.

This answers question 2, however it does not extend to networks of STS involving more than two nodes. The following figure shows a counter-example:



Transition systems Ψ_1 and Ψ_2 are assumed to be endochronous. Then communication between Φ_1 and Φ on the one hand, and Φ and Φ_2 on the other hand, can be desynchronized. Unfortunately, communication between Φ_1 and Φ_2 via Φ cannot, as it is not true in general that $\Psi_1 \parallel \Phi \parallel \Psi_2$ is endochronous. The problem is that endochrony is not compositional, hence even ensuring in addition that Φ itself is endochronous does not work out. Thus we would need to ensure that Ψ_1, Ψ_2 as well as $\Psi_1 \parallel \Phi \parallel \Psi_2$ are all endochronous. This cannot be considered as an adequate solution when networks of processes are considered. Therefore we move on introducing the alternative notion of *isochrony,* which focusses on communication, and is compositional.

6 Isochrony, and Synchronous/Asynchronous Compositions

The next result addresses the question of when property (3) holds. We are given two STS $\Phi_i = \langle V_i, \Theta_i, \rho_i \rangle$, $i = 1, 2$. Let $W = V_1 \cap V_2$ be the set of their common variables, and $\Phi = \Phi_1 \parallel \Phi_2$ their synchronous composition. For each reachable state s of Φ, we denote by $s_1 =_{\text{def}} s[V_1]$ and $s_2 =_{\text{def}} s[V_2]$ the restrictions of state s respectively to Φ_1 and Φ_2. It should be reminded that, for $i = 1, 2$, s_i is a reachable state of Φ_i. Corresponding notations s^-, s_1^-, s_2^- for past states are used accordingly.

Definition 2 (Isochrony). *Let* (Φ_1, Φ_2) *be a pair of* STS *and* $\Phi = \Phi_1 \parallel \Phi_2$ *be their parallel composition. Transitions of* $\Phi_i, i = 1, 2$, *are written* (s_i^-, s_i). *The following conditions* **(i)** *and* **(ii)** *are defined on pairs* $((s_1^-, s_1), (s_2^-, s_2))$ *of transitions of* (Φ_1, Φ_2)*:*

(i) *1.* $s_1^- = s^-[V_1]$ *and* $s_2^- = s^-[V_2]$ *holds for some reachable state* s^- *of* Φ, *in particular* s_1^- *and* s_2^- *are unifiable;*
2. none of the states $s_i, i = 1, 2$ *are silent on the common variables, i.e., it is not the case that, for some* $i = 1, 2$*:* $s_i[v] = \bot$ *holds for all* $v \in W$ *;*
3. s_1 *and* s_2 *coincide over the set of* present *common variables*[5], *i.e.:*

$$\forall v \in W \ : \ (\, s_1[v] \neq \bot \text{ and } s_2[v] \neq \bot \,) \Rightarrow s_1[v] = s_2[v] \ ;$$

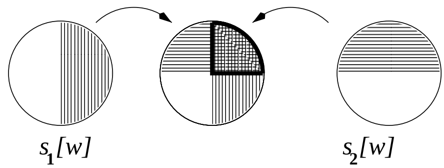
(ii) *States* s_1 *and* s_2 *coincide over the* whole set *of common variables, i.e., states* s_1 *and* s_2 *are unifiable, i.e.,*

$$s_1 = s[V_1] \text{ and } s_2 = s[V_2] \text{ holds for some reachable state } s \text{ for } \Phi \ .$$

The pair (Φ_1, Φ_2) *is said to be* isochronous *if and only if for each pair* $((s_1^-, s_1), (s_2^-, s_2))$ *of transitions of* (Φ_1, Φ_2), *condition* **(i)** *implies condition* **(ii)**.

[5] By convention this is satisfied if the set of present common variables is empty.

Comment. Roughly speaking, condition of isochrony expresses that unifying over *present* common variables is enough to guarantee the unification of the two considered states s_1 and s_2. Condition of isochrony is illustrated on the following figure:



The figure depicts, for unifiable previous states s_1^-, s_2^-, the corresponding states s_1, s_2 where (s_i^-, s_i) is a valid transition for Φ_i. The figure depicts the interpretation of s_1 (circle on the left) and s_2 (circle on the right) over shared variables W. White and dashed areas represent absent and present values, respectively. The two left and right circles are superimposed in the mid circle. In general, vertically and horizontally dashed areas do not coincide, even if s_1 and s_2 unify over the subset of shared variables that are present for both transitions (double-dashed area). Pictorially, unification over double-dashed area does not imply in general that dashed areas coincide. Isochrony indeed requires that unification over double-dashed area does imply that dashed areas coincide, hence unification of s_1 and s_2 follows.

The following theorem justifies introducing this notion of isochrony.

Theorem 2.

1. *If the pair (Φ_1, Φ_2) is isochronous, then it satisfies property (3).*
2. *Conversely, we assume in addition that Φ_1 and Φ_2 are both endochronous. If the pair (Φ_1, Φ_2) satisfies property (3), then it is isochronous.*

Thus, isochrony is a sufficient condition of property (3), and it is also in fact necessary when the components are endochronous.

Comments:

1. We have already discussed the importance of enforcing property (3). Now, why is this theorem interesting? Mainly because it replaces condition (3), which involves infinite runs, by condition **(i)** $\Rightarrow$ **(ii)** of isochrony, which only involves pairs of reactions of the considered pair of sts.
2. Comment 1 about endochrony also applies for isochrony.

Proof. We successively prove points 1 and 2.

1. Isochrony Implies Property (3). The proof proceeds from two steps:

1. Let Φ^a be the desynchronization of Φ, defined in equation (1), and $\delta \in \Sigma^a$ be an asynchronous run of Φ^a. There is at least one corresponding synchronous run σ of Φ such that $\delta = \sigma^a$. Any such σ is clearly the synchronous composition of two unifiable runs σ_1 and σ_2 for Φ_1 and Φ_2, respectively. Hence associated asynchronous runs σ_1^a and σ_2^a are also unifiable, and their asynchronous composition $\sigma_1^a \wedge^a \sigma_2^a$ belongs to $\Sigma_1^a \wedge^a \Sigma_2^a$. Thus we always have the inclusion:

$$\Phi_1^a \parallel_a \Phi_2^a \supseteq (\Phi_1 \parallel \Phi_2)^a \tag{11}$$

Proving (3) now amounts to the proof of the converse inclusion. So far we have only used the definition of desynchronization and asynchronous composition, isochrony has not yet been used.

2. Proving the opposite inclusion, requires to prove that, when moving from asynchronous composition to the synchronous one, the additional constraints resulting from a reaction-per-reaction matching of unifiable runs will not result in rejecting pairs of runs that otherwise would be unifiable in the asynchronous sense. This is where isochrony is used.
A pair (δ_1, δ_2) of asynchronous runs is picked out such that $\delta_1 \bowtie^a \delta_2$: they can be combined with the asynchronous composition to form some run $\delta = \delta_1 \wedge^a \delta_2$ (cf. (2)). By definition of desynchronization (cf. section 4), there exist a (synchronous) run σ_1 of Φ_1, and a (synchronous) run σ_2 of Φ_2, such that δ_i is obtained by desynchronizing σ_i, $i = 1, 2$ (as we do not assume endochrony at this point, run σ_i is not uniquely determined). Thus each run σ_i is a succession of states. Clearly, inserting finitely many silent states between successive states of σ_i would also provide valid candidates for recovering δ_i after desynchronization. We shall show, by induction over the set of runs, that:

$$\text{properly inserting such silent states in the appropriate component produces two runs which are \textit{unifiable} in the synchronous sense.} \tag{12}$$

This means that, from a pair (δ_1, δ_2) such that $\delta_1 \bowtie^a \delta_2$, we can reconstruct (at least) one pair (σ_1, σ_2) of runs of Φ_1 and Φ_2 that are unifiable in the synchronous sense, and thus it proves the converse inclusion:

$$\Phi_1^a \parallel_a \Phi_2^a \subseteq (\Phi_1 \parallel \Phi_2)^a \ . \tag{13}$$

From (11) and (13) we then deduce property (3). Thus we move on proving (12) by induction over pairs of runs.
Let (σ_1, σ_2) be a pair of runs of Φ_1 and Φ_2. the induction hypothesis is:

$$\sigma_1^a \bowtie^a \sigma_2^a \Rightarrow \exists(\rho_1, \rho_2) \, \text{runs of} \, \Phi_1 \, \text{and} \, \Phi_2, \, \text{s.t.} \, \sigma_i^a = \rho_i^a \, \text{and} \, \rho_1 \bowtie \rho_2 \tag{14}$$

Let us assume that (14) holds for every pair of runs of ordinal strictly less than that of (σ_1, σ_2) and that σ_1^a and σ_2^a are asynchronously composable. These two runs may start with infinitely or finitely many silent states over the common variables W, therefore three cases may occur:

CASE 1 : Both runs contain some non silent state over W, therefore they can be decomposed as follows: $\sigma_1 = s_{1,1}, \ldots s_{1,k_1}, s_{1,k_1+1}, \sigma_1'$ and $\sigma_2 = s_{2,1}, \ldots s_{2,k_2}, s_{2,k_2+1}, \sigma_2'$, where the first k_1 states of σ_1 and the first k_2 states of σ_2 are all silent over W and s_{1,k_1+1}, s_{2,k_2+1} are both non-silent over W. We concentrate on those variables $v \in W$ that are present in both states s_{1,k_1+1} and s_{2,k_2+1}. As $\sigma_1^a \bowtie^a \sigma_2^a$ holds, then we must have $s_{1,k_1+1}[v] = s_{2,k_2+1}[v]$ for any such v. Thus points 1,2 and 3 of condition **(i)** of isochrony are satisfied. Hence, by isochrony, s_{1,k_1+1} and s_{2,k_2+1} are indeed unifiable in this case. Moreover $\sigma_1'^a \bowtie^a \sigma_2'^a$ and since the ordinal of (σ_1', σ_2') are strictly less than that of (σ_1, σ_2), induction hypothesis (14) holds, and there exists (ρ_1', ρ_2') a pair of composable runs such that $\sigma_i'^a = \rho_i'^a, i = 1, 2$. We now define two runs by inserting silent states in σ_1 and σ_2:

$$\rho_1 = s_{1,1}, \ldots s_{1,k_1}, \underbrace{\bot, \ldots \bot}_{h_1 \text{ silent states}}, s_{1,k_1+1}, \rho_1'$$

$$\rho_2 = s_{2,1}, \ldots s_{2,k_2}, \underbrace{\bot, \ldots \bot}_{h_2 \text{ silent states}}, s_{2,k_2+1}, \rho_2'$$

Where $h_1 = \max(0, k_2 - k_1)$, $h_2 = \max(0, k_1 - k_2)$. The first $\max(k_1, k_2)$ states of ρ_1 and ρ_2 are composable because they are silent over W. Recall that s_{1,k_1+1} and s_{2,k_2+1} are composable states and that $\rho_1' \bowtie \rho_2'$. Therefore ρ_1 and ρ_2 are composable and $\rho_i^a = \sigma_i^a$.

CASE 2 : Both runs $\sigma_1 = s_{1,1}, \ldots s_{1,i}, \ldots$ and $\sigma_2 = s_{2,1}, \ldots s_{2,i}, \ldots$ are silent over W. Therefore they are synchronously composable.

CASE 3 : One of the two runs σ_1, σ_2 is silent over W, while the other contains a non-silent state. This violates the left-hand part of the implication in the induction hypothesis (14): $\sigma_1^a \bowtie^a \sigma_2^a$ does not hold.

This proves that induction hypothesis (14) holds for runs (σ_1, σ_2). By induction principle it also holds for every pair of runs.

2. Under Endochrony of the Components, Property (3) Implies Isochrony. From Theorem 1 we know that, in our proof of point 1 of theorem 2, the synchronous runs σ_i are uniquely defined, up to silent states, from their desynchronized counterparts σ_i^a. If isochrony is not satisfied, then, for some pair (σ_1^a, σ_2^a) of unifiable asynchronous runs, and their decompositions $\sigma_i = (s_{i,j})_{j>0}, i = 1, 2$, of them, it follows that points 1,2,3 of condition **(i)** of isochrony are satisfied, and there exists $n > 0$ such that states $s_{1,n}$ and $s_{2,n}$ are *not* unifiable. As our only possibility is to try to insert silent states in the two components our process of incremental unification on a per reaction basis fails. Thus (13) is violated, and so is property (3). This finishes the proof of the theorem. ◇

An interesting immediate byproduct is the extension of these results on desynchronization to networks of communicating synchronous components:

Corollary 1 (Desynchronizing a Network of Components). *Let $(\Phi_k)_{k=1,\ldots,K}$ be a family of* STS. *Let us assume that each pair $(\Phi_k, \Phi_{k'})$ is isochronous, then:*

1. *For each disjoint subsets I and J of set $\{1, \ldots, K\}$, the pair*

$$\left(\ \|_{k \in I} \ \Phi_k \ , \ \ \|_{k' \in J} \ \Phi_{k'} \ \right) \tag{15}$$

is isochronous.

2. *Also, desynchronization extends to the network:*

$$(\Phi_1 \parallel \ \ldots \ \parallel \Phi_K)^a = \Phi_1^a \parallel_a \ \ldots \ \parallel_a \Phi_K^a \ . \tag{16}$$

Proof. 1. It is sufficient to prove the following restricted case of (15):

$$(\Psi, \Phi_1) \text{ and } (\Psi, \Phi_2) \text{ are isochronous} \Rightarrow (\Psi, \Phi_1 \parallel \Phi_2) \text{ is isochronous} \tag{17}$$

as (15) follows via obvious induction on the cardinality of sets I and J. Thus we focus on proving (17). Let (s^-, s) and (t^-, t) be two pairs of successive states of Ψ and $\Phi_1 \parallel \Phi_2$ respectively, which satisfy condition **(i)** of isochrony, in definition 2. Let t be the composition (unification) of the two states s_1 and s_2 of Φ_1 and Φ_2, respectively. By point 2 of **(i)**, at least one of these two states is not silent, and we assume s_1 is not silent. From point 3 of **(i)**, s and s_1 coincide over the set of *present* common variables, and thus, since pair (Ψ, Φ_1) is isochronous, states s and s_1 coincide over the *whole* set of common variables of Ψ and Φ_1. Thus s and s_1 are unifiable. But, on the other hand, s_1 and s_2 are also unifiable since they are just restrictions of the same global state t of $\Phi_1 \parallel \Phi_2$. Thus states s and t are unifiable, and pair $(\Psi, \Phi_1 \parallel \Phi_2)$ is isochronous. This proves (17).

2. The second statement is proved via induction on the number of components: $(\Phi_1 \parallel \ldots \parallel \Phi_K)^a = ((\Phi_1 \parallel \ldots \parallel \Phi_{K-1}) \parallel \Phi_K)^a = (\Phi_1 \parallel \ldots \parallel \Phi_{K-1})^a \parallel_a \Phi_K^a$, and the induction step follows from (15). ◇

(COUNTER-)EXAMPLES.

Examples:

- a single-clocked communication between two STS.
- the pair $(\widetilde{\Phi}_1, \widetilde{\Phi}_2)$ of formula (10).

Counter-example: Two STS communicating with one another through two unconstrained reactive variables x and y. Both STS exhibit the following reactions: x present and y absent, or alternatively x absent and y present.

7 Getting GALS Architectures

In practice, only partial desynchronization of networks of communicating STS may be considered. This means that system designers may aim at generating *Globally Asynchronous* programs made of *Locally Synchronous* components communicating with one another via asynchronous communication media — this is referred to as GALS architectures.

In fact, theorems 1 and 2 provide the adequate solution to this problem. Let us assume that we have a finite collection Φ_i of STS such that

1. *each Φ_i is endochronous, and*
2. *each pair (Φ_i, Φ_j) is isochronous.*

Then, from corollary 1 and theorem 1, we know that

$$(\Phi_1 \parallel \ \ldots \ \parallel \Phi_K)^a = \Phi_1^a \parallel_a \ \ldots \ \parallel_a \Phi_K^a$$

and each Φ_k^a is in one-to-one correspondence with its synchronous counterpart Φ_k. Here is the resulting execution scheme for this GALS architecture:

- For communications involving a pair (Φ_i, Φ_j) of STS, each flow is preserved individually, but global synchronization is loosened.
- Each STS Φ_i reconstructs its own successive reactions by just observing its (desynchronized) environment, and then locally behaves as a synchronous STS.
- Finally, each Φ_i is allowed to have an internal activation clock which is *faster* than communication clocks. Resulting local activation clocks evolve asynchronously from one another.

8 Conclusion

We have presented an in depth study of the relationship between synchrony and asynchrony. The overall approach consists in characterizing those networks of STS which can be safely desynchronized, without semantic loss. Actual implementation of the communications only requests that 1/ message shall not be lost, and 2/ messages on each individual channel are sent and delivered in the same order. This type of communication can be implemented either by FIFOs or by rendez-vous.

The next questions are: 1/ how to test for endo/isochrony? and, 2/ if such properties are not satisfied, how to modify the given network of STS in order to guarantee them? It turns out that both points are easily handled on abstractions of synchronous programs, using the so-called *clock calculus* which is part of the SIGNAL compiler. We refer the reader to [2, 3, 8] for additional details. Enforcing endo/isochrony amounts to equipping each STS with a suitable additional STS which can be regarded as a kind of "synchronization protocol". When this is done, desynchronization can be performed safely.

This method has been implemented in particular in the SILDEX tool for the SIGNAL language, marketed by TNI, Brest, France. It is also implemented in the SIGNAL compiler developed at Inria, Rennes.

References

[1] R. Alur and T. A. Henzinger. Reactive modules. In *Proceedings of the 11th IEEE Symposium on Logic in Computer Science, LICS'96*, pages 207–218, 1996. extended version submitted for publication.

[2] T. P. Amagbegnon, L. Besnard, and P. Le Guernic. Arborescent canonical form of boolean expressions. Technical Report 2290, Inria Research Report, June 1994.

[3] T. P. Amagbegnon, L. Besnard, and P. Le Guernic. Implementation of the dataflow language SIGNAL. In *Proceedings of the ACM SIGPLAN Conference on Programming Languages Design and Implementation, PLDI'95*, pages 163–173, 1995.

[4] P. Aubry. *Mises en œuvre distribuées de programmes synchrones*. PhD thesis, université de Rennes 1, 1997.

[5] A. Benveniste and G. Berry, editors. *Proceedings of the IEEE*, volume 79, chapter The special section on another look at real-time programming, pages 1268–1336. IEEE, September 1991.

[6] A. Benveniste and G. Berry. Real-time systems design and programming. *Another look at real-time programming, special section of Proc. of the IEEE*, 79(9):1270–1282, September 1991.

[7] A. Benveniste and P. Le Guernic. Hybrid dynamical systems theory and the signal language. *IEEE Transactions on Autom. Control*, 35(5):535–546, May 1990.

[8] A. Benveniste, P. Le Guernic, and P. Aubry. Compositionality in dataflow synchronous languages: specification & code generation. research report RR–3310, INRIA, November 1997. `http://www.inria.fr/RRRT/publications-eng.html`, see also a revised version co-authored with B. Caillaud, March 1998.

[9] G. Berry. Real time programming: Special purpose or general purpose languages. In *Proceedings of the IFIP World Computer Congress*, San Francisco, 1989.

[10] G. Berry. The constructive semantics of esterel. Draft book, `http://www.inria.fr/meije/esterel`, December 1995.

[11] G. Berry and E. M. Sentovich. An implementation of construtive synchronous programs in POLIS. manuscript, November 1998.
[12] B. Caillaud, P. Caspi, A. Girault, and C. Jard. Distributing automata for asynchronous networks of processors. *European Journal on Automated Systems*, 31(3):503–524, 1997.
[13] P. Caspi. Clocks in dataflow languages. *Theoretical Computer Science*, 94:125–140, 1992.
[14] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer Academic Pub., 1993.
[15] N. Halbwachs, F. Lagnier, and Ratel C. Programming and verifying real-time systems by means of the synchronous dataflow language lustre. *IEEE Trans. on Software Engineering*, 18(9):785–793, September 1992.
[16] L. Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, 1983.
[17] L. Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Proc. IFIP 9th World Congress*, pages 657–668. North Holland, 1983.
[18] P. Le Guernic, T. Gautier, M. Le Borgne, and Le Maire C. Programming real-time applications with SIGNAL. *Another look at real-time programming, special section of Proc. of the IEEE*, 79(9):1321–1336, September 1991.
[19] O. Maffeis and P. Le Guernic. Distributed implementation of Signal : scheduling and graph clustering. In *3rd International School and Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 149–169. Springer Verlag, September 1994.
[20] EP-ATR Project. Signal : a formal design environment for real time systems. In *6th International Joint Conference on Theory and Practice of Software Development, TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*, Aarhus, Denmark, 1995. Springer-Verlag.
[21] Y. Sorel. Sorel: Real-time embedded image processing applications using the a3 methodology. In *Proc. IEEE International Conf. on Image Processing*, Lausanne, September 1996.
[22] Y. Sorel and C. Lavarenne. Syndex v4.2 user guide. `http://www-rocq.inria.fr/syndex/.articles/doc/doc/SynDEx42.html`.