Hierarchic Normal Forms for Desynchronization

Jean-Pierre Talpin, Albert Benveniste, Benoit Caillaud, Paul Le Guernic

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

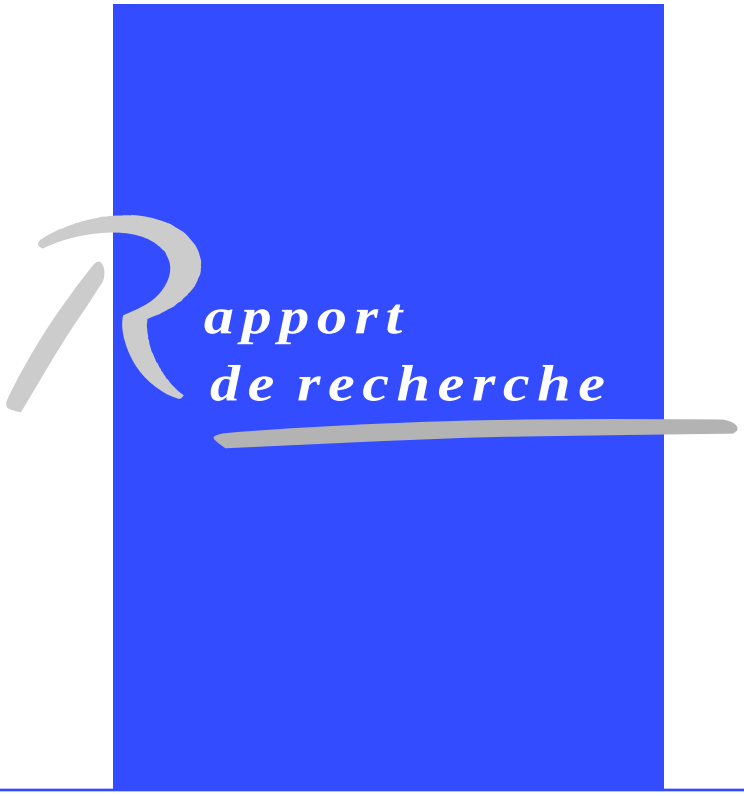
Hierarchic Normal Forms for desynchronization

Jean-Pierre Talpin , Albert Benveniste , Benoît Caillaud , Paul Le Guernic

N°3822

December 1999

THÈME 1



ISSN 0249-6399 ISRN INRIA/RR--3822--FR+ENG

Hierarchic Normal Forms for desynchronization

Jean-Pierre Talpin * , Albert Benveniste , Benoît Caillaud , Paul Le Guernic†

Thème 1 — Réseaux et systèmes
Projets Ep-Atr, Pampa

Rapport de recherche n ° 3822 — December 1999 — 37 pages

Abstract: Based on an earlier work, 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. This theory can be used as a basis for correct distributed code generation. The present paper presents a new data structure, the *hierarchic normal form* for a transition relation, which is instrumental in implementing this theory. We illustrate this on a Statecharts example. The whole approach is implemented in the SIGNAL compiler.

Key-words: synchronous languages, desynchronization, distributed code generation.

(Résumé : tsvp)

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

† This work is or has been supported in part by the following projects: REUTEL funded by Alcatel, Esprit LTR-SYRF (Esprit project EP 22703).

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Hiérarchies d'horloges pour la désynchronisation

Résumé : Nous présentons une étude approfondie des relations entre formalismes synchrones et asynchrones, et nous étudions la désynchronisation de programmes synchrones. Dans nos travaux précédents nous avions donné des critères effectifs garantissant la préservation de la sémantique lors de la désynchronisation et la génération de code distribuée. Ce rapport se concentre sur les structures de données adéquates pour ce problème, à savoir les hiérarchies d'horloges. Ce travail apporte des éléments théoriques nouveaux sur les techniques utilisées dans le compilateur SIGNAL.

Mots-clé : langages synchrones, désynchronisation, génération de code distribuée.

Contents

1 Introduction

Synchronous programming [4, 9, 13] 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 [8]. 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.

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 [5] 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 [12]. It establishes a precise relation between so-called well-clocked synchronous functional programs and the subset of Kahn networks amenable to "buffer-less" evaluation.

Since then, the issue of synchronous program "desynchronisation" has been investigated by several authors [11, 21, 3, 24, 23, 10], see [7] for a more detailed discussion.

In [7, 6] we have provided an extensive, in depth, analysis of the links between synchrony and asynchrony. The proposed 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. The above papers investigated in particular the following two issues related to desynchronisation, namely : *1/ when is desynchronisation revertible, for a single synchronous program ?,* and, *2/ when is a network of synchronous programs robust to the desynchronisation of communications ?* The new notions of *endochrony* and *isochrony* were introduced to answer questions 1 and 2 respectively. Endo/Isochrony are properties which can be checked on the synchronous specifications, prior to desynchronisation and code distribution. Enforcing Endo/Isochrony in a given design is an efficient way to automatically synthesize the schedulings and protocols needed for correct asynchronous code distribution.

In this paper we introduce suitable data structures for checking Endo/Isochrony, and we provide corresponding algorithms. This data structure, called *hierarchic normal form* (HNF) is a new kind of decision diagram. It is a normal form for transition relations, it does not require any prior variable ordering. It deeply exploits the multiclock nature of Synchronous Transition Systems (STS), a simple model for synchronous programs suggested by Amir Pnueli and discussed in [7, 6]. In STS, signals can be present or absent in a given reaction. The present/absent status of a signal is abstracted as its *clock.* A HNF provides a canonical form for signals in which corresponding clocks are nested. Using HNF, Endo/Isochrony is efficiently checked or can be synthesized.

The paper is organised as follows. In section 2 we introduce a simple Statecharts example to motivate our approach. The STS model is briefly recalled in section 3, and the results of [7, 6] are summarized in section 4. HNF are introduced in section 5, for the simpler case of boolean STS. As STS are equipped with synchronous parallel composition, we provide in section 6 compositional algorithms for computing them. In section 7 we discuss how HNF can be used in Endo/Isochrony. HNF for general STS is discussed in section 8. The Statecharts example is re-discussed in section 9, and then conclusions and perspectives are drawn.

2 Discussing a Statecharts example

The motivations for choosing Statecharts for our example are the following. First, Statecharts are a widely known and visual formalism [16]. And, second, we wish to convince the reader that our technique for distributing synchronous programs extends beyond our own SIGNAL formalism [20], and even beyond the "french" synchronous ESTEREL [9] and LUSTRE [14, 15].

The Statecharts example shown in figure 1 will support our discussion throughout this paper. The Statecharts has four and-states working in parallel : an arbiter, an emitter, and

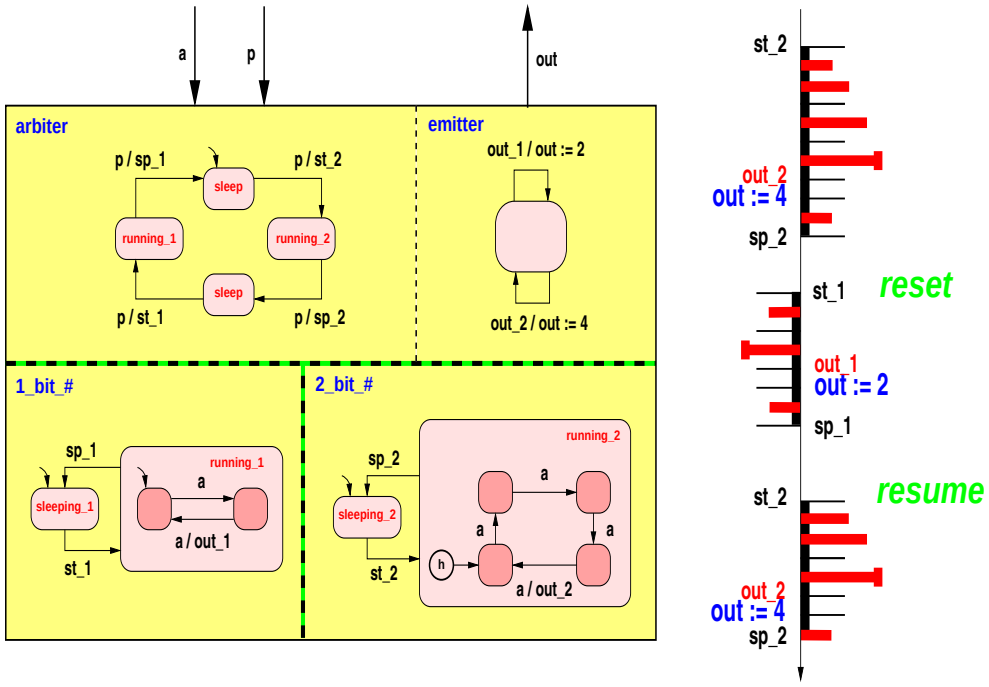


Figure 1: *A Statecharts example, and a scenario.*

two counters (1_bit– and 2_bit–counter). When receiving the push-button message `p`, the arbiter switches to the next state, in a circular way, and emits the start (`st`) or stop (`sp`) messages toward the two counters. Thus the two counters are activated according to an interleaving mode. When activated, each counter counts the occurrences of event `a`, and returns an `out` message when reaching his zero state. The emitter emits the result to the environment. Note the different reactivation mode for the two counters, with and without history (i.e., resume and restart mode). A possible scenario is depicted. It shows two active phases for the 2_bit–counter, and one for the 1_bit–counter. Counting the `a` is figured by the length of the horizontal thick bars.

2.1 First attempts

Brute force distribution. In figure 2 we show a distribution scenario with three commu-

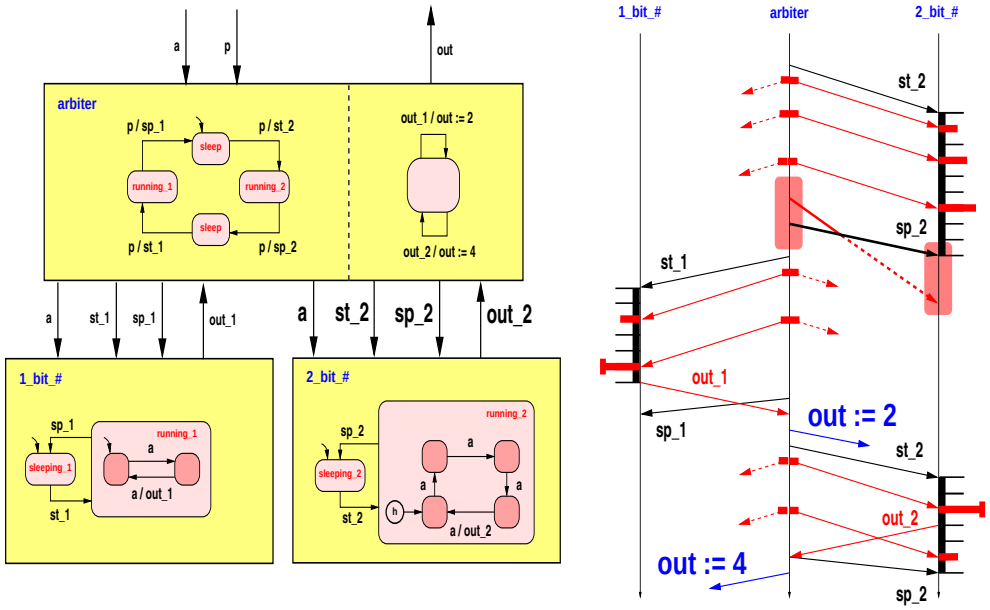


Figure 2: *Brute force distribution.* Loss of semantics.

nicating sequential processes, in which we have glued together the arbiter and the emitter to form a single, synchronous, component. Focus on the grey patches on `arbiter` and `2_bit_#`: they exhibit a change in the interleaving, causing in turn a change in the history of process `arbiter` (in the scenario of figure 1 we first observe `out:=4`, while in the scenario of figure 2 we first observe `out:=2`). Whence the loss of so-called serial semantics.

Preserving step semantics. In figure 3 we remove the undesirable effect observed in figure 2 by fully distributing the synchronous semantics: the step is broadcast and is used to activate each communicating StateChart. By doing this we restore the semantics, even globally. However, as revealed by the scenario shown in figure 3, each step is blocking. This is often unacceptable as a mode of execution. And it turns out that this is unnecessary as we show next.

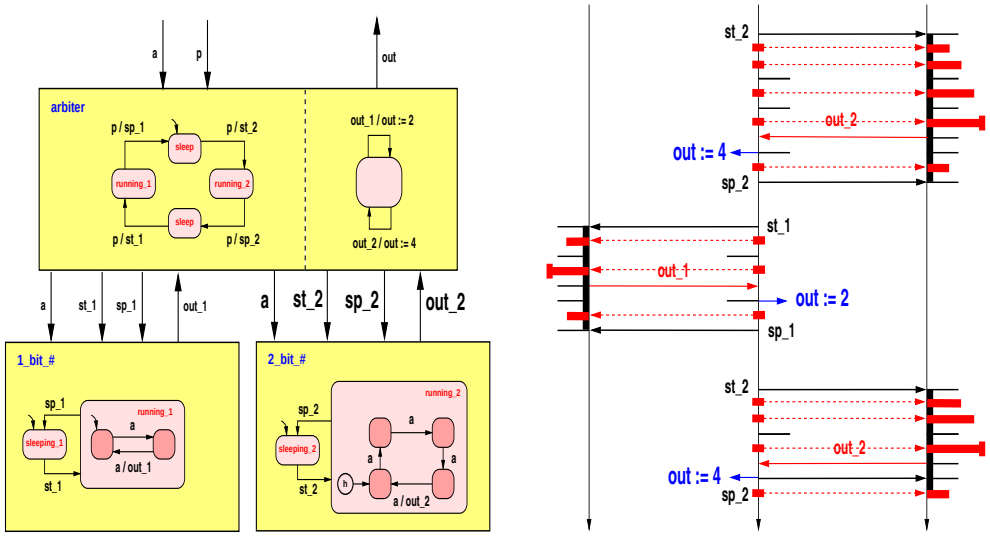


Figure 3: *Preserving step semantics.* blocking at each step : not acceptable !!

2.2 Preserving semantics : discussion and proposed solution

1. Referring to figure 2, focus on the downward communications `a, st_2, sp_2`: they are awaited by `2_bit_#` *asynchronously,* hence we need to explicitly preserve the global interleaving from the emitter down to the receiver.

   This is achieved by the way of introducing a *"local superstep"*, which has as a clock the supremum of the clocks of `a, st_2, sp_2`; this "local superstep" is emitted by the arbiter and it indicates to the `2_bit_#` which signal it is supposed to await in the considered reaction.

2. Then, focus on the `out_2` communication; again the arbiter is awaiting for it asynchronously, and we again need a "local superstep"; but it already exists, as we can use for this purpose the immediate step following the emission of the `a` signal. The "step" we refer to here is the local step of the arbiter.

   Thus we only need to convert the `out_2` event into a boolean signal with clock identical to that of the immediate step following the emission of `a`, and encodes `out_2` via the `true` occurrences of this boolean signal, which we keep calling `out_2`.

Comparing with figure 3, we see that, with our new solution, the `arbiter` is blocked only when waiting for the echo (boolean) signal `out_2`. We have proposed *an application-dependent protocol generation,* this is illustrated in the figure 4.

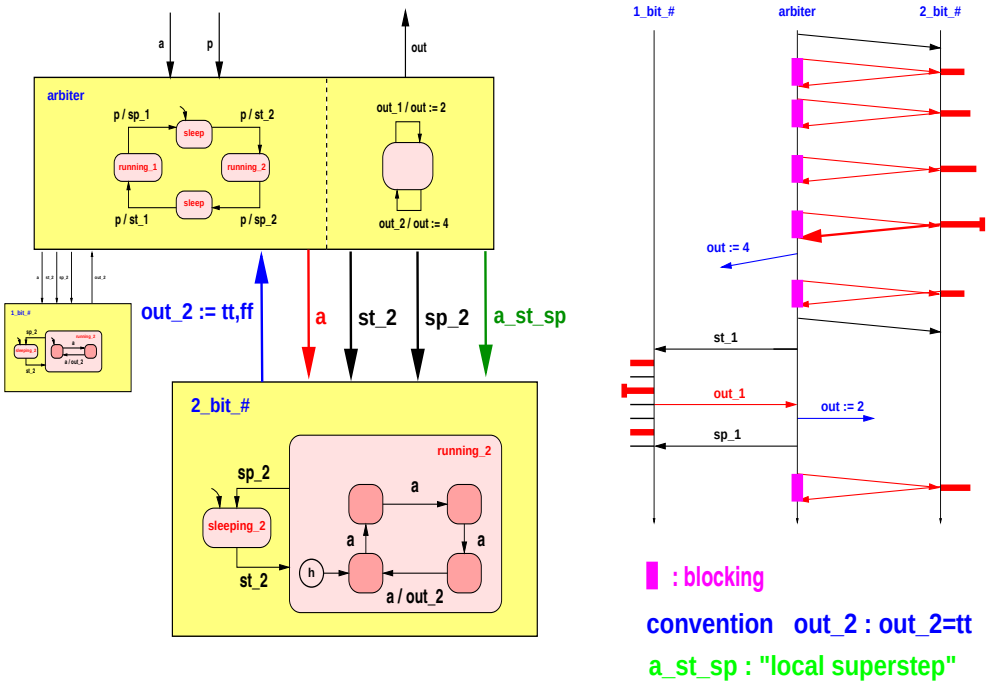


Figure 4: *Desynchronizing* `2_bit_#`. The new "local superstep" `a_st_sp` has been introduced, it has as a clock the union of the instants of `a`, `st`, `sp`, and it indicates which of these three events are present in the considered reaction. Therefore it synchronizes the reception of the triple `a`, `st`, `sp`. Also, `out_2` has been converted from pure signal to boolean. For simplicity, we did not modify the Statecharts, so the production of `a_st_sp` in the `arbiter` has not been made explicit.

2.3 The key idea

This was just an example, but it indicates what kind of tool we need to make this systematic. When a process is waiting for stimuli from the environment in an asynchronous mode, we need to ensure that a sort of a "*local* superstep" is transmitted first, from which the proper synchronization of the original stimuli can be inferred by the receiver without resourcing to any global clocking mechanism. As indicated in figure 4, we had to *add* such a local superstep in order to synchronize the reception of messages `a`, `st`, `sp`, by the `2_bit_#`. On the other hand, such a local superstep to synchronize the reception of `out_2` by the `arbiter` was already available, as `arbiter` knows that `out_2` can be received only when he had emitted an `a`. From this quick analysis the following conclusions emerge :

- Reasoning on the nesting of clocks (implication of presence/absence between different signals) is the basic tool. To this end we shall introduce a so-called *hierarchic normal form* for synchronous specifications, which makes the nested structure of clocks fully explicit. This nesting of clocks can be formally manipulated to automatically synthesize the "local supersteps".
- To prove the correctness of our approach, we first need a semantic framework in which synchrony and asynchrony can be jointly handled and the preservation of semantics can be proved.

This task is the subject of the rest of this paper.

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[1] 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. Transitions are pairs $(s^-, s) \in S \times S$. For $t = (s^-, s)$ a transition and $v \in V$ a variable, we write $t[v] = (s^-[v], s[v])$.

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:

$$\Phi_1 \parallel \Phi_2 = \langle V_1 \cup V_2, \Theta_1 \wedge \Theta_2, \rho_1 \wedge \rho_2 \rangle \quad (1)$$

[1] 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.

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 of 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, $h_v \in \{\mathrm{T}, \bot\}$ denotes its *clock*: $[h_v \neq \bot] \Leftrightarrow [v \neq \bot]$
- 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.

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 [18, 19]. 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

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

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

$$\forall v \in V_{\mathbf{r}} \quad : \quad \left\{ \begin{array}{rcl} \bot_s[v] & = & \bot \\ \bot_s[\xi_v] & = & s[\xi_v] \end{array} \right.$$

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: $(s^-, s) \mid\!= \rho \Rightarrow (s^-, \bot_{s^-}) \mid\!= \rho$ and $(\bot_{s^-}, s) \mid\!= \rho$. When this condition is not satisfied, we extend ρ minimally so that stuttering invariance is satisfied. By convention, we shall simply write $\bot$ instead of $\bot_s$ when mentioning a particular state s is not required.

Examples for subsequent reuse. We now introduce several simple examples to illustrate our algorithms. The Statecharts example is rediscussed in section 9.

Example. Our first example is an STS with transition relation specified by the statement

$$\begin{array}{l} h_b = h_u = h_v \\ \| \\ \textbf{if } \; b = \mathrm{T} \; \textbf{then } \; y = u \; \textbf{else } \; y = v \end{array} \tag{2}$$

where b is boolean. This is a selector guarded by the boolean b. ◇

Example. Our second example is an STS with transition relation specified by the statement

$$\textbf{if } h_u \neq \bot \; \textbf{then } \; y = u \; \textbf{else } \; y = v \tag{3}$$

This selector is not guarded. ◇

4 A theory of desynchronisation

The synchronous programming paradigm is essentially characterized as follows :

1. Programs progress via an infinite sequence of *reactions*: $P = R^\omega$, where R denotes the family of possible reactions.
2. Within a reaction, decisions can be taken on the basis of the *absence* of some events.
3. Parallel composition is given by taking the pairwise conjunction of associated reactions, whenever they are composable: $P_1 \| P_2 = (R_1 \wedge R_2)^\omega$.

In contrast, the following can be stated about the kind of asynchrony we consider in this work :

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 individual 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.

In the rest of this section we summarize the formal results from [7, 6].

4.1 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:

$$\sigma = (\sigma[v])_{v \in V}, \quad \text{where} \quad \sigma[v] = s_0[v], s_1[v], \ldots, s_k[v], \ldots.$$

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 $s_k[v] \neq \perp$ (this subsequence depends on v). Then we set:

$$\sigma^a = (\sigma^a[v])_{v \in V} \quad \text{where} \quad \sigma^a[v] = s_{k_0}[v], s_{k_1}[v], s_{k_2}[v], \ldots.$$

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:

$$\Phi^a \quad =_{\text{def}} \quad \langle V, \Sigma^a \rangle \ , \tag{4}$$

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:

$$\Phi_1^a \parallel_a \Phi_2^a \quad =_{\text{def}} \quad \langle V, \Sigma^a \rangle \ , \text{where} \left\{ \begin{array}{rcl} V & = & V_1 \cup V_2 \\ \Sigma^a & = & \Sigma_1^a \sqcup^a \Sigma_2^a \end{array} \right. \tag{5}$$

and $\sqcup^a$ 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

$$\sigma_1^a \quad \bowtie^a \quad \sigma_2^a \tag{6}$$

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 $\sigma^a =_{\text{def}} \sigma_1^a \sqcup^a \sigma_2^a$ as:

$$(\sigma_1^a \sqcup^a \sigma_2^a)[v] \quad =_{\text{def}} \quad \textbf{if} \ \ v \in V_i \ \textbf{then} \ \ \sigma_i^a[v]$$

Then

$$\Sigma^a \quad = \quad \{\sigma_1^a \sqcup^a \sigma_2^a \ : \ \sigma_i^a \in \Sigma_i^a \ \wedge \ \sigma_1^a \bowtie^a \sigma_2^a\}$$

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:*

$$\Phi_1^a \parallel_a \Phi_2^a \quad = \quad (\Phi_1 \parallel \Phi_2)^a \ ? \tag{7}$$

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.

4.2 Endochrony and Re-synchronization

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:

$$\forall v \in V : \ s^h[v] \in \{\bot, \mathrm{T}\}, \text{ and } s^h[v] = \bot \Leftrightarrow s[v] = \bot \tag{8}$$

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

$$W' \quad \hookrightarrow_{s^-} \quad W,$$

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^-,

$$\emptyset = V(0) \hookrightarrow_{s^-} V(1) \hookrightarrow_{s^-} V(2) \hookrightarrow_{s^-} \ldots \tag{9}$$

in which, for each $k > 0$, $V(k)$ is the greatest set of variables such that $V(k-1) \hookrightarrow_{s^-} V(k)$ holds. In particular, in (9), $V(1)$ consists of the subset of those variables that are either always present or always absent in all successor states of s^-. Of course sequence (9) 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 (9) 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:*

$$\emptyset = V(0) \hookrightarrow_{s^-} V(1) \hookrightarrow_{s^-} V(2) \hookrightarrow_{s^-} \ldots \text{ converges to } V \tag{10}$$

Condition (10) 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, we refer the reader to [7, 6] for a proof:

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*

$$(\Phi')^a = \Phi^a \quad \Rightarrow \quad \Phi' = \Phi \tag{11}$$

4.3 Isochrony, and Synchronous/Asynchronous Compositions

The next result addresses the question of when property (7) 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. We use the notations of the beginning of section 3. We need to weaken the definition of STS parallel composition (1) as follows[2].

Desynchronized conjunction of transition relations. The *desynchronized conjunction* of two transition relations $\rho_1 \wedge_a \rho_2$ is defined as follows. For t_1 and t_2 two transitions, we define *asynchronous unifiability* $t_1 \bowtie^a t_2$ by:

$$t_1 \bowtie^a t_2 \quad \text{iff} \quad (\, t_1[v] \neq \perp \text{ and } t_2[v] \neq \perp \,) \Rightarrow (\, t_1[v] = t_2[v] \,) \tag{12}$$

Note that $t_1 \bowtie^a t_2$ means that transitions t_1 and t_2 are unifiable on their common *present* ports, regardless of absence (this is just the restriction to transitions of the definition (6)

[2] We follow here a presentation which is slightly different from, but equivalent to that of [7, 6].

which was formulated for flows). Definition (12) is in contrast to *synchronous unifiability,* or unifiability for short, $t_1 \bowtie t_2$ defined by:

$$t_1 \bowtie t_2 \quad \text{iff} \quad (\, v \in V_1 \cap V_2 \,) \Rightarrow (\, t_1[v] = t_2[v] \,) \tag{13}$$

which means that transitions t_1 and t_2 are unifiable on their common ports, including presence/absence. Condition (13) corresponds to the conjunction of transition relations introduced in the definition of STS composition.

If $t_1 \bowtie^a t_2$, we can define $t_1 \sqcup^a t_2$ by

$$\begin{array}{lll} (t_1 \sqcup^a t_2)[v] & =_{\text{def}} & \textbf{if } \exists i = 1,2 \ : \ (\, v \in V_i \textbf{ and } \ t_i[v] \neq \bot \,) \textbf{ then } \ t_i[v] \\ & & \textbf{else } \ \bot \end{array}$$

With this in mind, we define $\rho_1 \wedge_a \rho_2$ as follows:

$$\rho_1 \wedge_a \rho_2 \quad = \quad \{t_1 \sqcup^a t_2 \ : \ t_i \models \rho_i \ \wedge \ t_1 \bowtie^a t_2\} \tag{14}$$

Definition 2 (Isochrony) *Let (Φ_1, Φ_2) be a pair of* STS *and $\Phi = \Phi_1 \parallel \Phi_2$ be their parallel composition. The pair (Φ_1, Φ_2) is called* ***isochronous*** *if*

$$\rho_1 \wedge \rho_2 \quad = \quad \rho_1 \wedge_a \rho_2 \tag{15}$$

holds, restricted to the set of reachable states for Φ.

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.

The following theorem justifies introducing this notion of isochrony, we refer the reader to [7, 6] for a proof.

Theorem 2

1. *If the pair (Φ_1, Φ_2) is isochronous, then it satisfies property (7).*
2. *Conversely, we assume in addition that Φ_1 and Φ_2 are both endochronous. If the pair (Φ_1, Φ_2) satisfies property (7), then it is isochronous.*

Thus, isochrony is a sufficient condition of property (7), and it is also in fact necessary when the components are endochronous.

Now we collect from reference [6] the following consequences of theorem 2, we refer the reader to the above reference for proofs:

1. If pairs (Ψ, Φ_1) and (Ψ, Φ_2) are isochronous, then so is pair $(\Psi, \Phi_1 \parallel \Phi_2)$.

2. We are given a finite family $(\Phi_k)_{k=1,\ldots,K}$ of STS. Assume that each pair $(\Phi_k, \Phi_{k'})$ is isochronous. Then
   (a) For each disjoints subsets I and J of set $\{1,\ldots,K\}$, the pair
   $$\left(\ \|_{k\in I}\ \Phi_k \quad , \quad \|_{k'\in J}\ \Phi_{k'} \ \right) \tag{16}$$
   is isochronous. Thus *isochrony is compositional.*
   (b) Also, desynchronization extends to the network:
   $$(\Phi_1 \parallel \ldots \parallel \Phi_K)^a \quad = \quad \Phi_1^a \ \|_a \ \ldots \ \|_a \ \Phi_K^a \ . \tag{17}$$
3. Assume pair (Φ_1, Φ_2) is isochronous, and pair (Ψ_1, Ψ_2) is such that Ψ_1 has no common variable with $\Phi_2 \parallel \Psi_2$ and Ψ_2 has no common variable with $\Phi_1 \parallel \Psi_1$. Then pair $(\Psi_1 \parallel \Phi_1 \, , \, \Phi_2 \parallel \Psi_2)$ is also isochronous. Thus *isochrony is a local property.*

5 Hierarchic normal form: the case of boolean STS

To simplify our presentation, we first consider the case of STS in which only pure or boolean variables are considered[3], we call them boolean STS. Boolean STS have finite state.

While BDD's or related decision diagrams have proved useful for symbolic manipulations of boolean relations, they are of little help to analyze the properties of endochrony or isochrony efficiently. Here we introduce an alternative decision diagram as a canonical form for transition relations, which we call the *hierarchic normal form.* As will be seen, this new type of decision diagram directly matches the notion of endochrony[4], and it allows an efficient checking of isochrony.

Example. To motivate our new notion of hierarchy we first discuss examples (2) and (3). In particular, example (2) is endochronous, as boolean b is present in all activations of the considered STS, and its value T/F determines which branch of the alternative is selected. To capture the fact that (2) is endochronous, we suggest to rewrite it as the following tree structure:

$$\begin{array}{ccc} & (h_b = \mathrm{T}, u, v) & \\ \swarrow & & \searrow \\ (b = \mathrm{T}, y = u) & & (b = \mathrm{F}, y = v) \end{array} \tag{18}$$

which reads as follows. 1/ When b is present, read u and v. 2/ Assuming b present, then if $b = \mathrm{T}$ select the left branch to produce y, otherwise select the right branch. Structure (18) is a simple case of a guarded hierarchy: each branch is guarded by a predicate whose value selects the branch, and this predicate can be evaluated prior to selecting the branch, this is figured by the fact that h_b ("b is present", hence it can be evaluated) is a father of the two branches in this hierarchy.

On the other hand, no such guarded selection occurs in example (3): the presence/absence of the two variables u, v must be checked prior to select the way y is produced. This STS is not endochronous, and it cannot be rewritten in a form like (18). ◇

[3] still augmented with symbol $\perp$ for absence.

[4] It was proposed by Paul Le Guernic in 1990 for this purpose [1], although the present formalization was not available until now.

5.1 Hierarchies

The type of decision diagram we introduce is a collection of trees, we call it a hierarchy, and define it now.

Partial transitions and other notations. We use the notations of section 3. Transitions are pairs $(s^-, s) \in S \times S$. For $t = (s^-, s)$ a transition and $v \in V$ a variable, we write $t[v] = (s^-[v], s[v])$. For $W \subseteq V$, we write $t[W] = (s^-[W], s[W])$; if $W = \emptyset$ we take the convention that $t[W] = *$, where $*$ is a distinguished symbol. *Partial transitions* are defined now :

$$\begin{array}{rcl} \text{a transition} & ::= & t \\ \text{a partial transition} & ::= & t \quad | \quad t[W] \ , \ W \subseteq V \end{array}$$

For $t[W]$ a partial transition, we call W its *associated set of variables.* Notation $t[W]$ is inconvenient as it explicitly refers to the, non unique, underlying transition t. Thus we shall instead generically denote by t a partial transition, and denote by V_t its associated set of variables. Hence, for t a partial transition, there exists some transition $\widetilde{t}$ such that $t = \widetilde{t}[V_t]$. Note that such a $\widetilde{t}$ is non unique. For $t = \widetilde{t}[V_t]$ a partial transition, we decompose it into its previous partial state and its current partial state: $t = (s^-, s)$, where $\widetilde{t} = (\widetilde{s}^-, \widetilde{s})$ and $s^- = \widetilde{s}^-[V_t], s = \widetilde{s}[V_t]$, and $\widetilde{t}$ is a (total) transition such that $t = \widetilde{t}[V_t]$. Finally, for ρ a transition relation, we denote by V_ρ its set of variables.

For t, t' two partial transitions such that $V_t \cap V_{t'} = \emptyset$, we denote by $t \cup t'$ the join of t and t', defined by

$$(t \cup t')[v] \quad = \quad \begin{cases} v \in V_t & : \quad t[v] \\ v \in V_{t'} & : \quad t'[v] \end{cases} \tag{19}$$

Definition 3 (hierarchy) ***Hierarchies*** *are defined over the following inductive grammar :*

a partial transition	t	$::=$	(s^-, s)	
a tree	Θ	$::=$	t	$\mid \ [\, t \triangleright \Phi \ : \ V_t \cap V_\Phi = \emptyset \ , \ s[V_t] \neq \bot \,]$
a forest	Φ	$::=$	Θ	$\mid \ \Phi \vee \Phi'$

where

- $V_\Phi = \cup_t V_t$*, where* t *ranges over the set of the partial transitions that are the nodes of hierarchy* Φ*.*
- $\triangleright$ *points to the set of children, and* $\vee$ *is the disjunction.*
- *The bracket* $[\, t \triangleright \Phi \ : \ V_t \cap V_\Phi = \emptyset \ , \ s[V_t] \neq \bot \,]$ *indicates that, in term* $t \triangleright \Phi$*, the attached conditions must be satisfied, and* $s[V_t] \neq \bot$ *means that* $\forall v \in V_t, s[v] \neq \bot$*. On the other hand, no condition is attached to the singleton term* t *in the same line.*

Using this notation, the informally introduced hierarchy (18) rewrites as follows: $(h_b = \mathrm{T}, u, v) \triangleright ((b = \mathrm{T}, y = u) \vee (b = \mathrm{F}, y = v))$. Transition relations are just flat hierarchies, i.e., hierarchies which do not involve the symbol $\triangleright$, thus we shall freely regard transition relations as hierarchies in the sequel.

Maximal paths $t_1 \triangleright t_2 \triangleright \ldots \triangleright t_n$ in Φ are generically denoted by τ, and we denote by $\mathbf{path}(\Phi)$ the set of maximal paths of Φ.

Definition 4 (associating transition relations to hierarchies) *Every hierarchy Φ defines a unique transition relation ρ_Φ as follows. For each maximal path $\tau = t_1 \triangleright t_2 \triangleright \ldots \triangleright t_n \in \mathbf{path}(\Phi)$, set*

$$t(\tau) \quad = \quad t_1 \cup t_2 \cup \ldots \cup t_n \ , \tag{20}$$

and, if $\cup_{i=1}^{n} V_{t_i} \overset{\neq}{\subset} V_\Phi$, extend $t(\tau)$ to the remaining set of variables of V_Φ by assigning to them the value $\perp$. Then, ρ_Φ is defined as the set of the so defined transitions $t(\tau)$.

Partial transition t is well defined thanks to the condition $V_t \cap V_\Phi = \emptyset$ attached to the term $t \triangleright \Phi$ in the definition 3. Also, using condition $s[V_t] \neq \perp$ attached to the term $t \triangleright \Phi$, we get the following property: referring to definition 4, pick a pair $(v_i, v_j), v_i \in V_{t_i}, v_j \in V_{t_j}$ and assume $i < j$. Then, decomposing $t = (s^-, s)$, we have:

$$s[v_j] \neq \perp \quad \Rightarrow \quad s[v_i] \neq \perp \tag{21}$$

Property (21) expresses that, if variables v_i and v_j are associated to partial transitions t_i and t_j such that t_i sits above t_j in hierarchy Φ, then the presence of v_j implies the presence of v_i. Thus, in a hierarchy, clocks are nested.

We have introduced hierarchies, and we have shown how hierarchies uniquely define transition relations. In the next section we show that the converse can be done as well, namely that every transition relation uniquely defines a hierarchy, we call it its *hierarchic normal form* (HNF).

5.2 Getting the *hierarchic normal form* Φ_ρ for a transition relation ρ

Consider the following function $\mathcal{H}$, mapping transition relations to hierarchies:

1. For ρ a transition relation defined over a set V of variables, and $t = (s^-, s)$ a transition satisfying ρ, we define

$$t^\triangle \quad = \quad t[W] : W = \left\{ v \in V \ : \ s[v] \neq \perp \wedge \left(\exists \widetilde{t}, \ \begin{array}{ll} & \widetilde{t} \models \rho \\ \wedge & \widetilde{t} \neq t \\ \wedge & \widetilde{t}[v] = t[v] \end{array} \right) \right\} \tag{22}$$

$$t^\triangledown \quad = \quad t[V \setminus W]$$

where $\setminus$ denotes set difference. Note that $t^{\triangle}$ and $t^{\triangledown}$ are partial transitions, and t decomposes as $t = t^{\triangle} \cup t^{\triangledown}$. Also, if $W = \emptyset$ or $V \setminus W = \emptyset$, then convention $t[\emptyset] = *$ applies.

2. Denote by $\rho^{\triangle}$ the set of partial transitions $t^{\triangle}$, where t ranges over ρ.

3. Pick a $t' \in \rho^{\triangle}$. Define $\rho^{\triangledown}_{t'}$ as the set of partial transitions $t^{\triangledown}$, where t ranges over the set of transitions of ρ such that $t^{\triangle} = t'$.

4. Define $\mathcal{H}$ by the formula

$$\mathcal{H}(\rho) \quad = \quad \bigvee_{t' \in \rho^{\triangle}} t' \triangleright \rho^{\triangledown}_{t'} \tag{23}$$

$\mathcal{H}(\rho)$ is a hierarchy.

In (23) we take the following conventions:

Convention 1

1. *If $t' = *$ we substitute $\rho^{\triangledown}_{t'}$ for the tree $(t' \triangleright \rho^{\triangledown}_{t'})$.*

2. *Similarly, if $t^{\triangledown} \in \rho^{\triangledown}_{t'}$ is such that $t^{\triangledown} = *$, we substitute the single partial transition t' for the branch $(t' \triangleright t^{\triangledown})$.*

3. *Finally, if $\rho^{\triangledown}_{t'}$ is a singleton, we collapse the branch $(t' \triangleright \rho^{\triangledown}_{t'})$ into the single partial transition $(t' \cup \rho^{\triangledown}_{t'})$ (this avoids having nodes with a single child).*

With these conventions, hierarchy $\mathcal{H}(\rho)$ has at most two levels. Thanks to (22) we have

$$\rho_{\mathcal{H}(\rho)} \quad = \quad \rho \tag{24}$$

i.e., applying definition 4 to hierarchy $\mathcal{H}(\rho)$ reproduces transition relation ρ.

The hierarchic normal form of a transition relation ρ is obtained by inductively re-applying $\mathcal{H}$ to $\rho^{\triangle}$ in (23). This is illustrated in figure 5 and is formalized next. For

$$\Phi \quad = \quad \bigvee_{t \in \rho} t \triangleright \Phi_t$$

a hierarchy, set

$$\mathcal{H}(\Phi) \quad = \quad \bigvee_{\tau \in \mathbf{path}(\mathcal{H}(\rho))} \tau \triangleright \Phi_t \tag{25}$$

where $\mathcal{H}(\rho)$ was defined in (23), and $\mathbf{path}(\mathcal{H}(\rho))$ is the set of maximal paths of hierarchy $\mathcal{H}(\rho)$. As $\mathcal{H}(\rho)$ generally possesses two levels, formula (25) generally increases the number of levels by one. In (25), we have $\mathcal{H}(\Phi) = \Phi$ iff $\mathcal{H}(\rho) = \rho$, i.e., applying $\mathcal{H}$ to ρ returns a flat hierarchy.

Iterating $\mathcal{H}$ ultimately yields a fixpoint, i.e., $\exists n < \infty : \mathcal{H}^{n+1}(\Phi) = \mathcal{H}^n(\Phi) =_{\text{def}} \mathcal{H}^\star(\Phi)$, i.e., we denote by $\mathcal{H}^\star$ the function mapping Φ to the fixpoint of the orbit of $\mathcal{H}$ containing Φ.

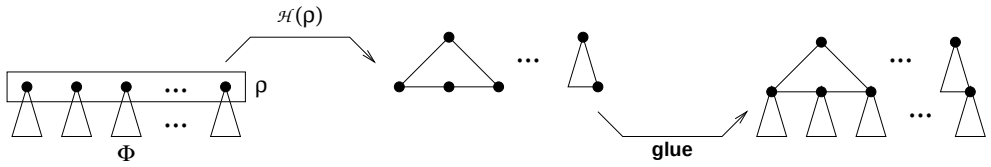


Figure 5: *Illustrating induction step (25).* The 1st diagram depicts Φ. Extract from Φ its top nodes, this defines a transition relation ρ. Apply $\mathcal{H}$ to ρ, this is depicted in the 2nd diagram. Then we glue the bottom part of the 1st diagram to the 2nd diagram, this yields the 3rd diagram, and illustrates formula (25).

Definition 5 (hierarchic normal form) *Since transition relations are just particular cases of hierarchies, for ρ a transition relation, we can define*

$$\Phi_\rho \quad =_{\text{def}} \quad \mathcal{H}^\star(\rho) \tag{26}$$

The following result is a consequence of property (24):

Theorem 3 *For every transition relation ρ, we have*

$$\rho_{\Phi_\rho} \quad = \quad \rho\,.$$

In section 7 we show how hierarchies can be used to handle endochrony and isochrony efficiently. The algorithm we just introduced for constructing the hierarchic normal form is not practical, as it requires that the transition relation, and hence the state space, be at hand in an enumerative form. In section 6 we provide more practical algorithms.

Discussion: BDD's. As revealed by definition 4, the hierarchic normal form is a decision diagram to represent transition relations, this is a feature common with BDD. In BDD, however, one first need to fix an ordering of the variables, and *then* the BDD becomes a canonical representation. In our case, the hierarchic normal form does not require any variable ordering prior to its definition. In some sense, a (partial) ordering of the variables is canonically derived as part of the algorithm, this ordering is provided by the tree structures building the hierarchy.

Example. Consider example (2). To simplify our discussion, we abstract the values carried by u, v, y (but not b). With this simplification, and introducing the clock of b, the enumerated form for this transition relation is:

$$\rho \quad = \quad \left\{ \begin{array}{ll} & h_b = \text{T}, b = \text{T}, y = u, v \\ \vee & h_b = \text{T}, b = \text{F}, u, y = v \\ \vee & \bot \end{array} \right.$$

where disjunct $\bot$ denotes the silent transition, in which every variable is absent. Denote by t_1, t_2, t_3 the corresponding three transitions. We have

$$\begin{array}{ll} {t_1}^{\triangle} = (h_b = \mathrm{T}) & {t_1}^{\triangledown} = (b = \mathrm{T}, y = u, v) \\ {t_2}^{\triangle} = (h_b = \mathrm{T}) & {t_2}^{\triangledown} = (b = \mathrm{F}, u, y = v) \\ {t_3}^{\triangle} = * & {t_3}^{\triangledown} = \bot \end{array}$$

Hence, applying $\mathcal{H}$ and then convention 1.1 to ${t_3}^{\triangle}$ yields:

$$\Phi_\rho \quad = \quad \left(\begin{array}{ll} & (h_b = \mathrm{T}) \triangleright \left(\begin{array}{ll} & (b = \mathrm{T}, y = u, v) \\ \vee & (b = \mathrm{F}, u, y = v) \end{array} \right) \\ \\ \vee & \bot \end{array} \right)$$

◇

Example. Consider example (3). Again, we abstract the values carried by u, v, y. With this convention, the enumerated form for this transition relation is:

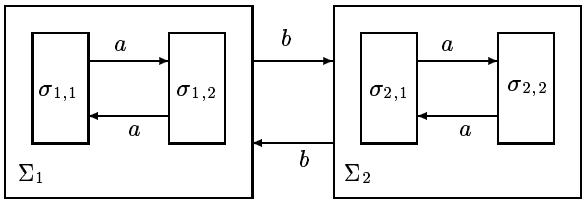
$$\rho \quad = \quad \left\{ \begin{array}{ll} & y = u, u \neq \bot, v \\ \vee & u = \bot, y = v \end{array} \right.$$

Denote by t_1, t_2 the corresponding two transitions. We have

$$\begin{array}{ll} {t_1}^{\triangle} = * & {t_1}^{\triangledown} = t_1 \\ {t_2}^{\triangle} = * & {t_2}^{\triangledown} = t_2 \end{array}$$

Hence $\Phi_\rho = \rho$ in this case. ◇

Example. Here we show how a hierarchy of states can be revealed by the HNF. Consider the following simple Statechart:



Its STS specification is:

$$\left(\begin{array}{ll} & \sigma_1^- = 1 \wedge a \wedge \sigma_1 = 2 \wedge \Sigma^- = \Sigma = 1 \\ \vee & \sigma_1^- = 2 \wedge a \wedge \sigma_1 = 1 \wedge \Sigma^- = \Sigma = 1 \\ \vee & \sigma_2^- = 1 \wedge a \wedge \sigma_2 = 2 \wedge \Sigma^- = \Sigma = 2 \\ \vee & \sigma_2^- = 2 \wedge a \wedge \sigma_2 = 1 \wedge \Sigma^- = \Sigma = 2 \end{array} \right)$$

$$\vee \qquad (27)$$

$$\begin{array}{l} \left(\begin{array}{ll} & \Sigma^- = 1 \wedge b \wedge \Sigma = 2 \\ \vee & \Sigma^- = 2 \wedge b \wedge \Sigma = 1 \end{array} \right) \\ \vee \\ \quad \bot \end{array}$$

In these formulas, statement $\bot$ is a shortage to indicate that all variables are absent in the considered reaction, i.e., the reaction is silent. For the sake of clarity, we have dropped expressions of the form $\sigma_1 = \bot$ etc, i.e., not mentioning a variable within a given disjunct indicates that the considered variable is absent in this disjunct. Also, for variables of pure type (modelling events), we write a for short instead of statement $a = \mathrm{T}$, and similarly for b.

This STS has two components of an overall choice. The first component models the four exclusive states named $\sigma_{i,j},\ i,j = 1,2$ in the figure. To this end we use two variables $\sigma_i,\ i = 1,2$ to indicate which local state is currently active ($\sigma_1 = 2$ indicates that state $\sigma_{1,2}$ is active in the diagram of the figure). We introduce the variables $\sigma_i^-,\ i = 1,2$ to indicate the previous state. The second component of the overall choice models the top level transition. State variables Σ, Σ^- are introduced to encode the top states in the diagram. Note that state variables are not persistent, but they return their current value when they are sensed.

Note that b-transitions are strongly preemptive, as they are exclusive from low level transitions. Introducing clock variables h_Σ, etc., to denote the clocks of Σ, etc., yields the following HNF for (27) (we have omitted the additional branch $\bot$ for clarity):

$$(h_\Sigma = \mathrm{T}) \triangleright \left(\begin{array}{l} (\Sigma^- = \Sigma = 1 \wedge h_{\sigma_1} = \mathrm{T}) \triangleright \left(\begin{array}{ll} & \sigma_1^- = 1 \wedge a \wedge \sigma_1 = 2 \\ \vee & \sigma_1^- = 2 \wedge a \wedge \sigma_1 = 1 \end{array} \right) \\ \vee \\ (\Sigma^- = \Sigma = 2 \wedge h_{\sigma_2} = \mathrm{T}) \triangleright \left(\begin{array}{ll} & \sigma_2^- = 1 \wedge a \wedge \sigma_2 = 2 \\ \vee & \sigma_2^- = 2 \wedge a \wedge \sigma_2 = 1 \end{array} \right) \\ \vee \\ \Sigma^- = 1 \wedge b \wedge \Sigma = 2 \\ \vee \\ \Sigma^- = 2 \wedge b \wedge \Sigma = 1 \end{array} \right)$$

In writing this HNF, we have taken advantage of the relations $h_{\Sigma_i^-} = h_{\Sigma_i}$, etc. This HNF again reveals the hierarchy depicted in the diagram. Thus nested states and transitions are reflected in the HNF. $\diamond$

6 Compositional algorithms

Algorithm (23,25,26) uses an enumerated form for the transition relation. This is clearly not practical. As actual transition relations are obtained by inductively composing transition relations or performing hiding, we provide incremental algorithms for these two operations. Then, we indicate how concatenation of transition relations translates into their corresponding hierarchies.

6.1 Hiding in a hierarchy

Consider a transition relation ρ defined over the set V of variables, and let $v \in V$ be a given variable, for hiding. Hiding v in ρ is denoted $(\exists v.\rho)$.

Now, for Φ a hierarchy defined over set V of variables, and $v \in V$, define $\exists v.\Phi$ as follows:

Algorithm 1 (hiding)

1. *Hide v in each node t of hierarchy Φ: by doing this some nodes of the resulting hierarchy become equal to $*$.*
2. *Having this done for each node of Φ, apply the conventions 1.*

Theorem 4 *The following holds, for any transition relation ρ and associated hierarchic normal form Φ_ρ:*

$$\exists v.\Phi_\rho \quad = \quad \Phi_{\exists v.\rho}$$

6.2 Composition of hierarchies

We want an incremental formula for computing $\Phi_{\rho_1 \parallel \rho_2}$ given Φ_{ρ_1} and Φ_{ρ_2}. This will be obtained by introducing the parallel composition of hierarchies, denoted $\overset{\mathcal{H}}{\parallel}$, and defined in such a way that formula $\Phi_{\rho_1} \overset{\mathcal{H}}{\parallel} \Phi_{\rho_2} = \Phi_{\rho_1 \parallel \rho_2}$ will hold.

Preliminaries.

1. For t_1 and t_2 two transitions, we have considered *asynchronous unifiability* $t_1 \bowtie^a t_2$ in (12). If $t_1 \bowtie t_2$ holds we can unify the two considered transitions and construct transition $t_1 \sqcup t_2$ by setting:

$$(t_1 \sqcup t_2)[v] \quad =_{\text{def}} \quad \textbf{if } v \in V_i \textbf{ then } t_i[v]$$

If $t_1 \bowtie^a t_2$ holds, decompose $t_i, i = 1, 2$ as follows. Define transitions t_1^u and t_1^r by

$$\begin{array}{rcl} V_{t_1^u} & = & (V_1 \setminus V_2) \cup \{ v \in V_1 \cap V_2 : t_1[v] = t_2[v] \} \\ v \in V_{t_1^u} & \Rightarrow & t_1^u[v] = t_1[v] \end{array} \qquad (28)$$

$$\begin{array}{rcl} V_{t_1^r} & = & V_{t_1} \setminus V_{t_1^u} \\ v \in V_{t_1^r} & \Rightarrow & t_1^r[v] = t_1[v] \end{array}$$

and similarly for t_2^u and t_2^r. These transitions satisfy

$$t_i = t_i^u \cup t_i^r \quad \text{and} \quad t_1^u \bowtie t_2^u\,, \qquad (29)$$

and pair (t_1^u, t_2^u) is maximal [5] having this property. The unification of t_1^u and t_2^u is well defined and is denoted $t_1^u \sqcup t_2^u$.

[5] meaning that $V_{t_i^u}, i = 1, 2$ are maximal for set inclusion.

2. For Φ a hierarchy, and t_0 a transition such that $V_{t_0} \cap V_\Phi = \emptyset$, we define

$$t_0 \cup \Phi \tag{30}$$

by substituting in Φ each root node $t_{\min}$ by $t_0 \cup t_{\min}$. Note that $t_0 \cup \Phi$ is indeed a hierarchy.

Algorithm 2 (composition of hierarchies)

1. *For two transition relations ρ_1 and ρ_2, we regard them as flat hierarchies and define*

$$\rho_1 \ {}^{\mathcal{H}}\!\parallel \rho_2 \quad \equiv \quad \rho_1 \parallel \rho_2 \tag{31}$$

i.e., ${}^{\mathcal{H}}\!\parallel$ and $\parallel$ coincide over pairs of flat hierarchies[6].

2. *Then we recursively define*[7]

$$\begin{aligned} &\Big(\bigvee_{t_1} t_1 \triangleright \Phi_{t_1}\Big) \ {}^{\mathcal{H}}\!\parallel \Big(\bigvee_{t_2} t_2 \triangleright \Phi_{t_2}\Big) \\ = &\bigvee_{t_1 \bowtie^a t_2} \big((t_1^u \sqcup t_2^u) \triangleright \big((t_1^r \cup \Phi_{t_1}) \ {}^{\mathcal{H}}\!\parallel (t_2^r \cup \Phi_{t_2})\big)\big) \end{aligned} \tag{32}$$

where $\bowtie^a, t_i^u, t_i^r$ are defined in (12,29).

Theorem 5 *The following holds:*

$$\Phi_{(\rho_1 \parallel \rho_2)} \quad = \quad \Phi_{\rho_1} \ {}^{\mathcal{H}}\!\parallel \Phi_{\rho_2} \tag{33}$$

Proof: see appendix A. ◇

6.3 Discussion

As discussed when introducing this section, algorithm (23,25,26) for computing the hierarchic normal form is not practical, as it uses an enumerated form for the transition relation.

An alternative algorithm, based on theorem 5, is the following. Given a transition relation ρ, inductively composed of the parallel composition $\parallel$ of flat transition relations $\rho_i, i \in I$:

1. Apply algorithm (23,25,26) to derive hierarchy Φ_{ρ_i}.
2. Pick a pair $(i,j) \in I \times I$ to form $\Phi_{i,j} = \Phi_{\rho_i} \parallel \Phi_{\rho_j}$.
3. Inductively pairwise compose such partial hierarchies to finally derive Φ_ρ.

[6] by abuse of notation, we write $\rho_1 \parallel \rho_2$ to mean $\rho_1 \wedge \rho_2$.
[7] in (32) we write $\bigvee_{t_1 \bowtie^a t_2}$ to mean $\bigvee_{(t_1,t_2)\,:\,t_1 \bowtie^a t_2}$.

As they involve selection procedures, steps 2 and 3 may have a very different performance depending on the way pairs are selected. The number of nodes $|\Phi_{i,j}|$ of $\Phi_{i,j}$ (we call it the *size* of the hierarchy) is at worst equal to $|\Phi_i| \times |\Phi_j|$. Carefully selecting the pair (i,j) can result in a drastic reduction of the actual size $|\Phi_{i,j}|$ as compared to worst case [8]. The objective of the selection procedure is to avoid intermediate explosion of the size of the partial hierarchies built in this way. Such a selection procedure involves heuristics. Such a heuristics, kown as the *clock calculus,* is implemented as a main step of the SIGNAL compiler [2].

6.4 Concatenation of hierarchies

So far, in defining hierarchies and constructing the HNF of a transition relation, we just ignored the initial condition. This implies that we might have considered transitions $t = (s^-, s)$ such that s^- is not reachable. As revealed by the theorems 1 and 2, reachability should be taken into account. The natural way of taking reachability into account is to provide incremental algorithms for computing reachable states. We do this by providing an algorithm for computing the concatenation $\Phi^- \ {}^{\mathcal{H}}\!\circ\ \Phi$ of hierarchies Φ^- and Φ in such a way that the following theorem holds, where $\rho^- \circ \rho$ denotes the concatenation of transition relations ρ^- and ρ [9]:

Theorem 6

$$\Phi_{(\rho^- \circ \rho)} \quad = \quad \Phi_{\rho^-} \ {}^{\mathcal{H}}\!\circ\ \Phi_\rho \tag{34}$$

It turns out that the new algorithm needed is a variation of the ones we provided just before. Therefore we shall only sketch it and shall not give full details. The idea is simple. Regard transition concatenation as the composition of the following maps:

1. Project transition relation ρ on its set of new states. Equivalently, encode previous states using "previous variables", i.e., for each variable $v \in V$, add a variable v^- such that $s[v^-] = s^-[v]$, this defines v^-. Rewrite ρ in terms of the augmented set of variable $V^- \cup V$, where V^- is the set of variables v^- where v ranges over V, and without explicitly referring to previous state s^-. Then the desired projection amounts to hiding V^- in ρ, and we have the algorithm 1 for performing this on the corresponding hierarchy. In the next step, V^- shall generically denote the set of previous variables of a transition relation with variable set V.

2. Consider concatenation $\rho^- \circ \rho$, and denote by W the set of variables of ρ^-. Using step 1 we can compute hierarchy $\exists W^-.\Phi_{\rho^-}$ from hierarchy Φ_{ρ^-}. Concatenating Φ_ρ to $\exists W^-.\Phi_{\rho^-}$ is just a composition ${}^{\mathcal{H}}\!\|$ of these two hierarchies in which variables W of $\exists W^-.\Phi_{\rho^-}$ have been relabelled by superscript $^-$. Again we have algorithm 2 for computing this incrementally.

[8] In particular, one may choose Φ_i and Φ_j of maximal $|V_{\Phi_i} \cap V_{\Phi_j}|$ to compose $\Phi_{i,j} = \Phi_{\rho_i} \parallel \Phi_{\rho_j}$

[9] the concatenation of two transition relations ρ_1 and ρ_2 is defined by: $(s^-, s) \models \rho_1 \circ \rho_2$ iff $\exists s' : (s^-, s') \models \rho_1 \wedge (s', s) \models \rho_2$.

To summarize

$$\Phi_{\rho^-} \ {}^{\mathcal{H}}\!\circ \ \Phi_\rho \quad =_{\text{def}} \quad \underbrace{\left(\underbrace{(\exists W^-.\Phi_{\rho^-})}_{\text{algorithm 1}}[W/W^-] \right) \ {}^{\mathcal{H}}\!\| \ \Phi_\rho}_{\text{algorithm 2}}$$

Clearly, these two steps can be collapsed into a more compact algorithm, but we prefer not giving the details here.

7 Hierarchies and endo/isochrony

Hierarchies are tightly related to endochrony. The idea is that the successive levels of a hierarchy Φ correspond to the nested sequence $V(0) \hookrightarrow_{s^-} V(1) \hookrightarrow_{s^-} V(2) \hookrightarrow_{s^-} \ldots$ introduced in (10). To really achieve this we first need to investigate the structure of *guards* in a hierarchy.

7.1 Guards

We wish to regard a hierarchy as a data structure to support an operational semantics. Assume the considered hierarchy is a tree Θ. Variables sitting in the root note of Θ will be present in any nonsilent transition of ρ_Θ : their common clock is the activation clock of the related STS. Assuming ρ_Θ is activated, we wish to know which transition will occur. Using (20), the transition will be incrementally constructed by selecting a path in tree Θ. Thus at each node t of Θ we will have to choose a child of t. In the good case, this selection is triggered by a guard, i.e., by the observed value of some boolean predicates. The material for formalizing this machinery is introduced next.

Definition 6 (guards) *Let Φ be a hierarchy and t a node of Φ. We assume that t is not a root node, and we denote by τ_t the downgoing path to t in hierarchy Φ. We say that t is* ***guarded*** *if*

> *there exists a nonempty subset $B_t \subseteq V_t$ of* boolean *variables such that, for each $b \in B_t$, its clock h_b belongs to $V_{t'}$ for some $t' \in \tau_t$.*

(Clock h_b sits "above" b in hierarchy Φ). The conjunction of the predicates $t[b]$ where b ranges over B_t is called the ***guard*** *of t.*

In the sequel, guards are generically denoted by the symbol α. For example, if $B_t = \{b, c\}$, then $\alpha = t[b] \wedge t[c]$.

Definition 7 (guarded hierarchy) *Let Φ be a hierarchy.*

1. *A subtree Θ of Φ is* guarded *by α_Θ if its root node is guarded in Φ by α_Θ.*

2. *A tree* $t \triangleright (\vee_{i=1}^{n} \Theta_i)$ *is* well-guarded *if each* Θ_i *is guarded, and if corresponding guards* α_{Θ_i} *form a set of mutually exclusive guards.*

3. *The hierarchy* Φ *is* ***fully-guarded*** *if is it an inductively well-guarded tree.*

7.2 Characterizing endochrony

Theorem 7 *The following two conditions are equivalent:*

(a) *The hierarchy* Φ *is fully-guarded.*

(b) *The transition* $t \models \rho_\Phi$ *is recursively determined by the value of variables involved at the successive nodes of* Φ *whose union form* t *: the corresponding* STS *is endochronous (definition 1).*

Proof:

(a)⇒(b) Using decomposition (20), each transition $t \models \rho_\Phi$ decomposes as $t = \cup_{i=0}^{k} t_i$ where $t_0 \triangleright t_1 \triangleright \ldots \triangleright t_k$ is a maximal path in Φ. Since hierarchy Φ is fully-guarded, it is firstly a tree and its root is t_0. Then, for $i = 1, \ldots, k$, the successive nodes t_i of each path are guarded. Pick t_0 ; due to condition 2 of definition 7, the choice among the children of t_0 is entirely determined by the value of some boolean variables b such that $h_b \in V_{t_0}$. Assume some child t_1 of t_0 is selected. Then the choice between the children of t_1 is entirely determined by the value of some boolean variables b such that $h_b \in V_{t_0} \cup V_{t_1}$. And so on. This proves the statement.

(b)⇒(a) If Φ is not fully-guarded, i.e., **(a)** is not satisfied, at least one of the following cases must occur :

1. Φ is not a tree. Then the choice between the root nodes is not determined, and thus **(b)** is not satisfied.
2. Some node, say t_i, of Φ is not guarded. Then choosing t_i does not depend on the values of the variables involved in the path to t_i, and thus **(b)** is not satisfied.
3. Some node t_i of the hierarchy has two childrens with non exclusive guards. When both guards are true, then again choosing between the two childrens is not determined by the variables involved in the path to t_i, and thus **(b)** is not satisfied.

This proves the statement **(b)** ⇒ **(a)**, and the theorem follows. ◇

Example. Consider example (2). Its HNF is fully guarded, hence it is endochronous. On the other hand, example (3) possesses a flat HNF, hence it is not fully guarded and is not endochronous. ◇

Enforcing endochrony

Let us now assume some non-endochronous ρ, how can we proceed to enforce endochrony ? The hierarchic normal form of ρ, again, provides the adequate answer : it tells us where and how fewest possible oracles should be added to ρ in order to reconstruct a fully-guarded tree out of the initial hierarchy Φ_ρ. This is detailed next. Referring to the proof of case **(b)** $\Rightarrow$ **(a)** of theorem 7, the three cases 1,2,3 can occur. For each of them, we give a solution to enforce endochrony.

1. If Φ_ρ is not a tree, assume that it has k roots $t_{1,1}, \ldots, t_{1,k}$. Add to each $t_{1,i}, i = 1, \ldots, k$ a guard $\alpha_{0,i} \wedge_{j \neq i} \neg \alpha_{0,j}$, and add the root $t_0 : \forall i\,,\ h_{\alpha_{0,i}} = h$ on the top of Φ_ρ, so that $t_0 \triangleright \Phi_\rho$ is now a tree.

2. Assume that some node of Φ_ρ, say t_i, is not guarded. Add to its parent node t_{i-1} a clock h_b and add to t_i the guard $b[\mathrm{T}]$.

3. Assume that some node t_i of the hierarchy has childrens with guards that are not pairwise mutually exclusive. Call *conflict set* a subset of this set of guards which can be simultaneously satisfied. A set of boolean variables with equal clocks can be introduced to solve this conflict. This generally yields a non unique solution. A unique solution can be found if enumerated types are used in addition to boolean ones.

7.3 Characterizing isochrony

In this subsection we express the property of isochrony in terms of the hierarchic normal forms for the two considered transition relations. To this end, we introduce a new composition operator for transition relations, and then for hierarchies. For ρ_1 and ρ_2 two transition relations, define

$$\rho_1 \parallel_a \rho_2 \quad = \quad \{t_1 \cup t_2 \mid t_1 \in \rho_1, t_2 \in \rho_2, t_1 \bowtie^a t_2\} \tag{35}$$

where $\bowtie^a$ was defined in (12). Using this new operation, we can reformulate the property of isochrony as follows :

Lemma 1 *A pair (ρ_1, ρ_2) of transition relations is isochronous iff*

$$\rho_1 \parallel_a \rho_2 \quad = \quad \rho_1 \parallel \rho_2 \tag{36}$$

holds.

We now indicate how condition (36) can be checked using hierarchies. We already defined the composition ${}^{\mathcal{H}}\!\parallel$ between hierarchies, as an extension of composition $\parallel$ on transition relations. Similarly, we now introduce composition ${}^{\mathcal{H}}\!\parallel_a$ between hierarchies, as an extension of composition $\parallel_a$ on transition relations. To this end, the following variation of algorithm 2 is introduced :

Algorithm 3 (variation of algorithm 2)

1. *For two transition relations ρ_1 and ρ_2, we define*

$$\rho_1 \ {}^{\mathcal{H}}\|_a \ \rho_2 \quad \equiv \quad \rho_1 \ \|_a \ \rho_2 \tag{37}$$

   i.e., ${}^{\mathcal{H}}\|$ and ${}^{\mathcal{H}}\|_a$ coincide over pairs of transition relations.

2. *Same as step 2 of algorithm 2.*

This algorithm is then used to check isochrony in the form of property (36) by checking whether

$$\Phi_{\rho_1} \ {}^{\mathcal{H}}\|_a \ \Phi_{\rho_2} \quad = \quad \Phi_{\rho_1} \ {}^{\mathcal{H}}\| \ \Phi_{\rho_2}$$

holds. Since their step 2 is identical, algorithms 2 and 3 only differ on leaves of hierarchies. Thus the marginal cost of computing $\Phi_{\rho_1} \ {}^{\mathcal{H}}\|_a \ \Phi_{\rho_2}$ in addition to $\Phi_{\rho_1} \ {}^{\mathcal{H}}\| \ \Phi_{\rho_2}$ is cheap. Therefore the additional cost of checking for isochrony in addition to computing the composition of the hierarchic normal forms of the two considered STS is cheap.

Enforcing isochrony

We first give a bruteforce algorithm directly working on transition relations, and then refine it, using hierarchies.

The bruteforce approach. We are given two transition relations ρ_1 and ρ_2. Introduce [10]

$$V^{4\text{iso}}_{\rho_1,\rho_2} \quad = \quad \{v \in V_{\rho_1} \cap V_{\rho_2} \ : \ \mathbf{Prop}(v)\}$$

where

$$\begin{aligned} \mathbf{Prop}(v) \ = \quad & \exists t_1 \models \rho_1 \ : \ [\, t_1[v] \neq \bot \,] \wedge [\, \exists t \models \rho_2 : t_1 \bowtie^a t \wedge t[v] = \bot \,] \\ \vee \quad & \exists t_2 \models \rho_2 \ : \ [\, t_2[v] \neq \bot \,] \wedge [\, \exists t \models \rho_1 : t \bowtie^a t_2 \wedge t[v] = \bot \,] \end{aligned}$$

Thus $V^{4\text{iso}}_{\rho_1,\rho_2}$ is the set of variables which cause isochrony to fail. Then, for each $v \in V^{4\text{iso}}_{\rho_1,\rho_2}$, add, in ρ_1, a boolean variable b_v which is present when ρ_1 is activated, and is true iff v is present, we call it a boolean clock. Add also such a b_v to ρ_2. One easily check that the so modified pair (ρ_1, ρ_2) is isochronous, as by unifying over the *value* of b_v, transitions must unify over *presence/absence* of v.

This algorithm is too expensive, as it forces to emit boolean control signals at the activation clock of each transition relation. The idea is to emit such a boolean control signal only when it is really needed. Using hierarchies provides the solution.

[10] Superscript $\ldots^{4\text{iso}}$ is reminiscent of "*for iso*chronizing".

Refinement using hierarchies. This is simple: just apply the bruteforce algorithm on the *leaves* of recursion (32) of algorithm 2 for computing $^{\mathcal{H}}\|$. More precisely, denote by Φ_i the HNF of $\rho_i, i = 1, 2$. Compute $\Phi_1 \ ^{\mathcal{H}}\| \ \Phi_2$ using recursion (32). The next step of this recursion is used to test whether $\Phi_1 \ ^{\mathcal{H}}\| \ \Phi_2 = \Phi_1 \ ^{\mathcal{H}}\|_a \ \Phi_2$ holds. If not, then apply the bruteforce approach to each leaf of $\Phi_1 \ ^{\mathcal{H}}\| \ \Phi_2$ which violates the desired condition $\Phi_1 \ ^{\mathcal{H}}\| \ \Phi_2 = \Phi_1 \ ^{\mathcal{H}}\|_a \ \Phi_2$. In this way, the minimum number of additional boolean clocks is created.

8 Hierarchic normal form in the general case: an abstraction technique

Section 5 discussed the boolean case. Here we discuss the general case, in which arbitrary domains for variables are considered.

Subsection 5.1 generalizes without any change, as the finite nature of state space is not used for introducing the notion of a hierarchy.

The situation is different, however, for subsection 5.2, as computing $t^{\triangle}$ in (22) involves a decision procedure. For general data types, testing for $\widetilde{t}[v] = t[v]$ may not be decidable. Also, the right hand side of formula (23) becomes an infinite structure if set $\rho^{\triangle}$ is infinite. Thus we need to implement some abstraction of the construction of subsection 5.2. We describe such an abstraction next (other constructions are possible).

To describe our abstraction procedure, we need to fix a syntax for STS. To this end we first use the objects introduced in the paragraph "notations" of section 3. In addition we consider the following set of primitive statements:

$$\textbf{if } b \textbf{ then } w = u \textbf{ else } w = v \tag{38}$$

$$\left.\begin{array}{r} R(v_1, \ldots, v_k) \\ h_{v_1} = \ldots = h_{v_k} \end{array}\right\} \tag{39}$$

Statement (38) is a selector: when boolean variable b is true, then w equals u, otherwise w equals v. Note that "otherwise" means here that b is false or absent. In the second statement, $(v_1, \ldots, v_k)$ is a tuple of variables of arbitrary types and of equal clocks, and R is a relation over the domain of $(v_1, \ldots, v_k)$, properly extended in such a way that $(\perp, \ldots, \perp)$ satisfies R. Note that (38,39) are both stuttering robust. As mentioned in [6], any transition relation can be constructed as the conjunct of the the objects introduced in the paragraph "notations" of section 3, together with primitive (38,39) above. We assume this mini syntax in the sequel.

To each statement (39) we associate an *assertion* α_R, which is a variable with domain $\{\mathrm{T}, \perp\}$ (it can be either true or absent), with clock equal to that of the arguments of R:

$$h_{\alpha_R} = h_{v_1} = \ldots = h_{v_k} \tag{40}$$

The abstraction $\widehat{\rho}$ of a transition relation ρ is obtained by replacing in ρ each conjunct of the form (39) by its corresponding abstract conjunct (40).

As our abstract transition relations only involves either boolean or pure data types, plus a finite number of equalities or inequalities over arbitrary data types guarded by predicates with boolean types, we know that, for $\widehat{t}, \widehat{t}' \models \widehat{\rho}$, the problem $\widehat{t}[v] = \widehat{t}'[v]$ is decidable. To show this, we refer the reader to [22]. In section 5.1 of this reference, a similar fragment of logic is shown to have the so-called "small-model" property, this means that the validity of tautologies involving this fragment of logic can be established by solely inspecting models up to a certain finite cardinality.

Hence computing $\widehat{t}^{\triangle}$ in (22) is now decidable. Also, formula (23) yields a finite structure. Then our whole apparatus applies. An abstraction technique of this kind is implemented in SIGNAL's clock calculus.

9 Back to the Statecharts example

Consider figure 3. Focus on signals `a, st_2, sp_2, out_2`. Abstract the behaviour of this system by masking all signals except `a, st_2, sp_2, out_2`. The corresponding STS has transition relation specified by :

$$\begin{array}{lll} \textbf{if } \texttt{step} = \text{T} & \textbf{then} & \textbf{if } \texttt{pre(a)} = \bot \textbf{ then } \texttt{out\_2} = \bot \\ & \textbf{else} & \tau \end{array} \tag{41}$$

In (41), `step` is the activation step of the considered Statecharts, it is a pure signal (this is examplified by the branch τ of the choice). Signal `pre(a)` denotes the value of `a` at the previous occurrence of `step` — for the sake of clarity, we did not include the statements relating `step, a, pre(a)` together. The rest of the statement indicates that `out_2` can only occur when `a` was present at the previous step. Besides the constraint that `step` is the activation step, the other variables `a, st_2, sp_2` are unconstrained. Note that we have discarded the counting of successive occurrences of `a`, this is an abstraction. Hence `out_2` is not fully determined by the other variables, unlike in the detailed Statecharts.

Now we take advantage of the side information that communication are directed. Partition the set { `a, pre(a) , st_2, sp_2, out_2` } into downgoing communications `a, st_2, sp_2` and upgoing ones `pre(a) , out_2` (we regard `pre(a)` as an "echo").

Consider first `a, st_2, sp_2` . As there is no constraint at all on this triple, this communication prevents the involved pair of Statecharts from being isochronous. To enforce isochrony we apply the trick of the end of subsection 7. As these three variable are not organized into a nontrivial hierarchy, we only can apply the bruteforce approach. This indeed consists in generating, as a guard, our so-called "local superstep" `a_st_sp` shown in figure 4.

Then focus on the pair `pre(a), out_2`. From the statement **if** `pre(a)` $= \bot$ **then** `out_2` $= \bot$, we know that `out_2` is less frequent than `pre(a)` . Thus again we need to generate a boolean variable which explicitly tells the `arbiter` when `out_2` was sent back by the

counter. This is equivalent to the conversion of `out_2` from pure to boolean type suggested in figure 4.

To summarize, our theory is a systematic extension of our proposed ad hoc solution for the Statecharts example.

10 Summary and perspectives

A theory of desynchronisation. Building on our previous work [6, 7] 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. We have illustrated this approach on a simple Statecharts example.

Data structures. Then we have focussed on algorithms and data structures needed to test for or enforce endo/isochrony. This paper can be seen as a mathematical reformulation of [1, 2], and a theoretical support for the techniques implemented in the SIGNAL compiler developed at Inria, Rennes, and the SILDEX tool for the SIGNAL language, marketed by TNI, Brest, France [11]. A central tool is the *Hierarchic Normal Form* (HNF) for STS, which is a decision diagram different from BDD's and their variants.

Assumptions on the communications layer. One may question the range of validity of the assumptions needed for our method to apply (requirements 1/ and 2/ above, and the resulting protocols which make use of blocking). It is certainly not applicable in general for all parts of embedded reactive system design, but we believe it can be useful in combination with other approaches. On the other hand, it seems well adapted to object oriented systems design methodologies, such as based on UML: both synchronous and asynchronous modes of interaction are useful and considered in this context, and our assumptions can be accepted in this case.

Static vs. dynamic. The traditional range of application for synchronous programming has been that of programs or systems with *no dynamic creation* of processes. Based on the above study, a resticted form of dynamicity can be handled, we discuss here an example. Consider an isochronous network of statically defined components $(\Phi_k)_{k=1,\ldots,K}$, see (16,17). Pick a pair (Φ_i, Φ_j) of components and perform a partial renaming using fresh names: the result is still isochronous. Do this finitely many times. Thanks to compositionality and locality of isochrony (see the consequences of theorem 2), we still get an isochronous network. This implies that an isochronous network of prototypes can be dynamically instanciated and deployed using asynchronous message communication. Of course, this is a restricted

[11] see `http://www.tni.fr/`

form of dynamic creation, as we assume a finite, statically defined, collection of prototype components. But the need for such an architecture is frequently encountered in practice. Work is under progress to establish this claim on a firm basis.

UML. From the previous results and remarks, it is natural to investigate the use of our techniques to establish a behavioural semantic backbone for UML, targeted to reactive systems [12]. To this end we have proposed a new formalism called BDL (Behavioural Description Language) [17, 25]. As compared to existing synchronous languages, BDL possesses the following features :

- It is targeted to architectures and partial designs. It aims at providing behavioural mechanisms of inheritance (a recognized difficult subject). To this end it is equipped with two composition operators : the parallel composition $\|$ (to specialize behaviours by adding constraints), and the nondeterministic choice $\vee$ (to enrich a given behaviour by adding new behaviours).
- It has a dual synchronous/asynchronous semantic.
- Statecharts as well as Sequence Diagrams and Message Sequence Charts can be translated into BDL, thus offering a blending of these formalisms at a semantic level.
- A visual syntax is under study, as a mild variation of Statecharts.

[12] This ongoing work is the central subject of the ongoing REUTEL project, funded by Alcatel.

A Appendix : proof of theorem 5

It is enough to prove the following two statements :

(i) We have

$$\rho_{\Phi_{(\rho \parallel \rho')}} \quad = \quad \rho_{(\Phi_\rho \ {}^{\mathcal{H}}\!\parallel \Phi_{\rho'})} \tag{42}$$

i.e., both sides of equality (33) generate the same transition relation. In fact, we shall instead prove the following, more general, result :

$$\rho_\Phi \parallel \rho_{\Phi'} \quad = \quad \rho_{(\Phi \ {}^{\mathcal{H}}\!\parallel \Phi')} \tag{43}$$

Clearly, (43) implies (42), but (43) can be formulated for arbitrary hierarchies Φ, Φ', not necessarily HNF. This will make the induction step in the proof easier.

(ii) Expression

$$\Phi \quad =_{\text{def}} \quad \bigvee_{t_1 \bowtie^a t_2} ((t_1^u \sqcup t_2^u) \triangleright (t_1^r \cup \Phi_{t_1} \ {}^{\mathcal{H}}\!\parallel t_2^r \cup \Phi_{t_2})) \tag{44}$$

is a hierarchic normal form (note that it is clearly a hierarchy, as the conditions of definition 3 are satisfied).

Both statements are proved by induction over the depth of the considered hierarchies. By (31) they trivially hold for flat hierarchies, so we only need to investigate the induction step, both for **(i)** and **(ii)**.

Induction step for (i). This is performed into two steps.

1. Consider the second line of equation (32). Assume we know that

   the desired property holds for hierarchies of depth at most equal to the maximal depth of hierarchies Φ_{t_i}.

   In particular,

$$\Phi_{t_1} \ {}^{\mathcal{H}}\!\parallel \Phi_{t_2} \quad = \quad \Phi_{(\rho_{t_1} \parallel \rho_{t_2})} \tag{45}$$

   where $\rho_{t_i}, i = 1, 2$ is the transition relation associated to hierarchy Φ_{t_i}. Using the definition (29) for t_i^r, formula (29) implies $V_{t_i^r} \cap V_{\Phi_{t_i}} = \emptyset$ and, by the induction hypothesis [13] :

$$(t_1^r \cup \Phi_{t_1}) \ {}^{\mathcal{H}}\!\parallel (t_2^r \cup \Phi_{t_2}) \quad = \quad \Phi_{((t_1^r \cup \rho_{t_1}) \parallel (t_2^r \cup \rho_{t_2}))} \tag{46}$$

[13] this is where we take advantage of replacing (42) by (43), as $(t_i^r \cup \Phi_{t_i})$, $i = 1, 2$ involved in (46) are hierarchies, but not HNF.

2. Transitions satisfying the transition relations respectively associated with hierarchies $\bigvee_{t_i} t_i \triangleright \Phi_{t_i}, i = 1,2$ have the form $t_i \cup t'_i$ where t'_i ranges over transition relation ρ_{t_i}. Pick such a pair and assume it is synchronously unifiable, we have

$$t_1^u \cup t_1^r \cup t'_1 \quad \bowtie \quad t_2^u \cup t_2^r \cup t'_2 \tag{47}$$

Thanks to decomposition (29), (47) is equivalent to

$$t_1^r \cup t'_1 \quad \bowtie \quad t_2^r \cup t'_2 \,. \tag{48}$$

But (46) and (48) together imply that both sides of (32) are associated to the same transition relation.

Induction step for (ii). Thanks to the inductive definition (25) of function $\mathcal{H}$ and the construction of the hierarchic normal form, it remains to prove the following:

$$\mathcal{H}(\rho_0) = \rho_0 \,, \quad \text{where} \quad \rho_0 =_{\text{def}} \bigvee_{t_1 \bowtie^a t_2} (t_1^u \sqcup t_2^u) \tag{49}$$

Define, for $i = 1,2 \; : \; \rho_{0,i} =_{\text{def}} \bigvee_{t_i} t_i^u$. Since Φ_{ρ_i} is a hierarchic normal form, we have $\mathcal{H}(\bigvee_{t_i} t_i) = \bigvee_{t_i} t_i$, hence

$$\mathcal{H}(\rho_{0,i}) = \rho_{0,i} \tag{50}$$

follows. Assume (49) is falsified. This implies that there exists two pairs (t_1, t_2) and (t'_1, t'_2) such that

$$\begin{aligned} t_1 \bowtie^a t_2 \quad &, \quad t'_1 \bowtie^a t'_2 \,, \text{ and} \\ \exists v \; : \; (t_1^u \sqcup t_2^u)[v] \quad &= \quad (t'^u_1 \sqcup t'^u_2)[v] \; \neq \; \bot \end{aligned} \tag{51}$$

Up to a permutation of the indices $1,2$, only two cases are possible if (51) holds:

$$\begin{aligned} \text{case } 1 \quad &: \quad t_1^u[v] = t'^u_1[v] \\ \text{case } 2 \quad &: \quad t_1^u[v] = t'^u_2[v] \end{aligned}$$

Case 1 contradicts (50). Case 2 is a bit more involved. Due to the maximality of the t_i^u in decomposition (29) we also have $t_1^u[v] = t_2^u[v]$, and we are back to case 1. This shows that (51) cannot hold, and thus (49) cannot be falsified. This finishes the proof of the induction step for **(ii)** and of the theorem 5. ◇

ACKNOWLEDGEMENT: Thierry Gautier is greatly indebted for fruitful discussions and comments on earler versions of the paper. Also, we thank Amir Pnueli for his comments on the abstraction technique.

References

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

[2] 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.

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

[4] 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.

[5] 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.

[6] A. Benveniste, B. Caillaud, and P. Le Guernic. Compositionality in dataflow synchronous languages : specification & distributed code generation. *Information and Computation*, to appear. http://www.irisa.fr/sigma2/benveniste/pub/compos-ic-final.ps.gz.

[7] A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. In J.C.M. Baeten and S. Mauw, editors, *CONCUR'99, Concurrency Theory, 10th International Conference*, volume 1664 of *Lecture Notes in Computer Science*, pages 162–177. Springer, August 1999.
http://www.irisa.fr/sigma2/benveniste/pub/compos-concur99.ps.gz.

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

[9] G. Berry. *The constructive semantics of* ESTEREL. Draft book,
http://www.inria.fr/meije/esterel, July 1999.

[10] G. Berry and E. M. Sentovich. An implementation of construtive synchronous programs in POLIS. manuscript, November 1998.

[11] 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.

[12] P. Caspi. Clocks in dataflow languages. *Theoretical Computer Science*, 94:125–140, 1992.

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

[14] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[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] D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts*. McGraw Hill, 1998.

[17] C. Jard, J-M. Jézéquel, A. Le Guennec, and B. Caillaud. Protocol engineering using uml. Draft paper, submitted, October 1999.

[18] L. Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, 1983.

[19] L. Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Proc. IFIP 9th World Congress*, pages 657–668. North Holland, 1983.

[20] 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.

[21] 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.

[22] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small-domains instantiations. In N. Halbwachs and D. Peled, editors, *Proceedings of CAV99*, Lecture Notes in Computer Science. Springer Verlag, 1999. to appear.

[23] Y. Sorel. Sorel: Real-time embedded image processing applications using the a3 methodology. In *Proc. IEEE International Conf. on Image Processing*, Lausanne, September 1996.

[24] Y. Sorel and C. Lavarenne. Syndex v4.2 user guide. http://www-rocq.inria.fr/syndex/.articles/doc/doc/SynDEx42.html.

[25] J-P. Talpin, A. Benveniste, and P. Le Guernic. Asynchronous deployment of synchronous pre-order transition systems. Technical report, Inria Research Report, October 1999. in preparation.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399