SyncCharts: A Visual Representation of Reactive Behaviors

Charles ANDRE

October 1995, revision April 27, 1996

Abstract

This report is an introduction to SyncCharts (Synchronous Charts) a new graphical representation of reactive behaviors based on the synchronous paradigm. Syntactically this model is close to Statecharts and Argos. It offers enhanced preemption capabilities. Its semantics is formally defined and any *Synchronous Charts* can be translated into an equivalent Esterel program.

SyncCharts combines state-oriented descriptions, hierarchy and powerful preemption mechanisms. It is fully compatible with the imperative synchronous language Esterel and it encourages programming with both textual and graphical reprensentations.

1 Introduction

A *reactive system* maintains permanent interactions with its environment. Usually, reactive systems are concurrent systems. Their global behavior results from the cooperation of their components (subsystems). In order to carry out an expected behavior, the evolutions of subsystems must be coordinated. Communication (information exchange) plays a central role in this coordination, and consequently, reactive systems are often viewed as communicating processes. This approach relegates *preemption* to a position of secondary importance, which is prejudicial to many reactive applications. "A process preemption ... consists in denying the right to work to a process, either permanently (abortion) or temporarily (suspension)" (G. Berry [Ber92]). Real-time operating systems, interrupt-driven systems, and more generally, control-oriented systems heavily rely on preemption. The specification of such systems needs preemption as a first class concept; their programming requires preemption primitives. Few models can deal with preemption. Languages that support preemption, to some extent, do not offer primitives tailored to reactive applications. A reason for this lack, is that most semantics dealing with both concurrency and preemption are complex or vague.

Faced with this problem, we adopt a *synchronous approach* [BB91]. The "*perfect synchrony hypothesis*" assumes on the one hand that cause (stimuli) and effect (reaction) are simultaneous, and on the other hand that information is instantaneously broadcast. These assumptions lead to an abstract view of temporal behaviors, in which sequencing, concurrency and preemption can be considered as orthogonal concepts. Within the framework of synchronous programming, clear mathematical semantics can be given to several forms of preemption. This point of view is advocated in a G. Berry's paper entitled "Preemption in Concurrent Systems" [Ber92]. In what follows, we shall often refer to this paper which brings the theoretical foundation of our contribution.

Engineers in the field of control and manufacturing systems are somewhat reluctant to synchronous languages. As a rule, they prefer *graphical approaches*. The Grafcet [IEC88] (Sequential Function Charts) is widely used for industrial logic control. StateCharts [Har87], a hierarchical visual model which is part of the Statemate environment, allows design of complex reactive system and it takes advantages from its industrial support. StateCharts is convenient but, as stated in a review of the various extensions of StateCharts [Bee94], its semantics has to be clarified. Argos [Mar90] is another synchronous and graphical model. It is based on a clear semantics, but it has not been widely distributed. In this report, we introduce SyncCharts (an acronym for Synchronous Charts) which inherits from StateCharts and Argos. Compared to its predecessors, SyncCharts deals with preemptions in a more rational way. It uses a restricted set of powerful graphical primitives.Its semantics is founded on a process calculus and a "synchronous charts" can be automatically translated into an Esterel program.

Graphical description has its own limits. We advocate using a *multiformalism approach.* SyncCharts allows graphical and textual refinements. Both are possible thanks to the underlying formal semantics: Macrostate graphs and Esterel's modules can be abstracted into a common model.

Plan

The paper is organized as follows:

- In a first part, we identify the basic needs for describing reactive behaviors. With the help of simple examples, we explain how SyncCharts can fulfill these demands.
- We then introduce a process calculus. This is a convenient way to express complex behaviors and to formally compare them.
- The third part is devoted to the syntax of SyncCharts.
- The expected behavior of a SyncCharts is formally expressed in the fourth section.
- Translations from SyncCharts into Esterel are given in the annex.

2 Main features of SyncCharts

This section is an *informal introduction* to SyncCharts. With the help of a progressively enriched example, we identify needs for a graphical representation of reactive behaviors. For each requirement, we introduce the solution adopted by SyncCharts.

A well-known basic model for discrete systems is the sequential machine (including Moore and Mealy machines). For industrial applications Grafcet (Sequential Function Charts) is widely used. More recent contributions are StateCharts and Argos. All these models are based on the notion of state, so is SyncCharts which inherits from them.

2.1 An example: a SR flip-flop

A Set-Reset flip-flop is a reactive system with two inputs: `set, reset` and two outputs: `OFF, ON` (Fig.1a). We assume that `set` and `reset` never occur at the same instant. Figures 1b, 1c and 1d are respectively Mealy machine, Moore machine and Grafcet representations of the behavior of a SR flip-flop.

2.2 State

A state represents a (temporary) invariant behavior of a part of a system. This is analogous to *condition* in condition/event systems (C.A Petri [Rei85]). *Steps* in Grafcet and states in Moore machines, adhere to this interpretation: When in a state, the associated outputs hold. In synchronous models, this behavior is expressed by emitting the associated signals at each instant (e.g., see the `sustain` statement in Esterel).

SyncCharts adopts this association of outputs with states. Graphically, a state is represented as a circle or an ellipse. The signal(s) to be emitted when the state is active, is (are) written inside the state symbol.

2.3 Transition

A transition is a change in active state. Usually, the graphical representation of a transition is an arc labelled by triggerring events. Mealy and Moore machines have self-loop arcs to express that the system may remain in a state. Grafcet and other synchronous models (StateCharts, Argos) make non-changes implicit[1]: Arcs express effective changes (notion of receptivity in Grafcet). SyncCharts, as a synchronous model,

[1] Of course, it is possible to have *explicit* self-loops, but the semantics is not simply staying in a state (see further).

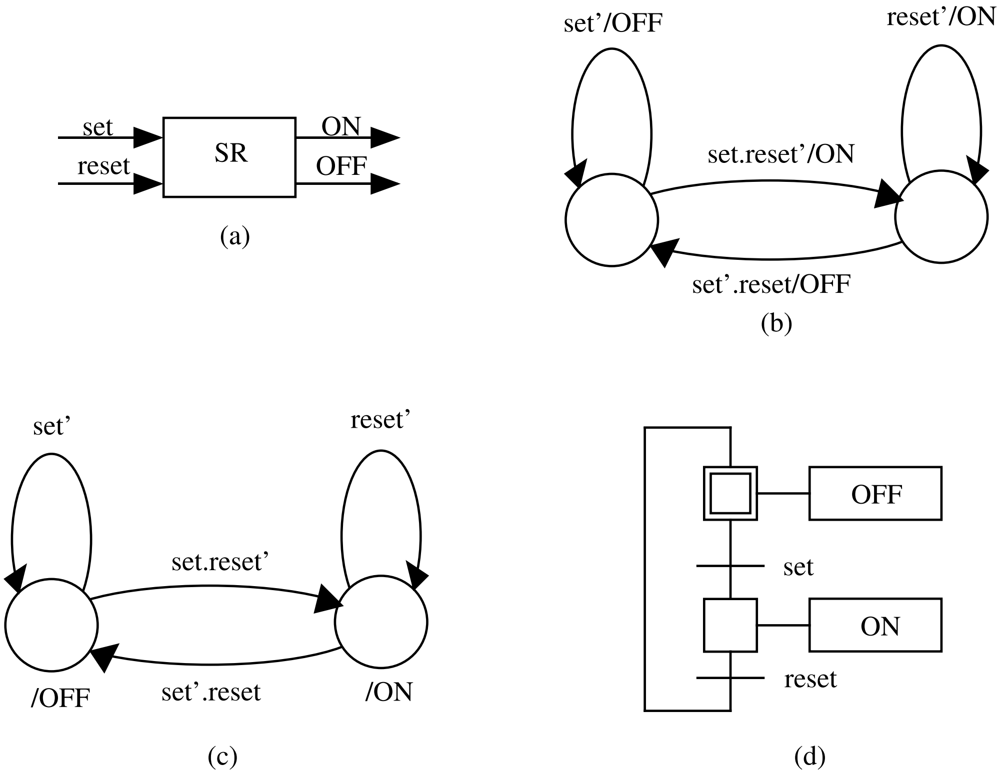

Figure 1: Modelling SR flip-flop

applies this convention: Arcs are labelled by triggerring signals. Signals can be combined (boolean expressions on signals using "+" as disjunction, "." as conjunction, and a prime ($'$) as negation).

In discrete time models, we have to be precise about when exactly changes in state and outputs occur. Let Y_t be the state, X_t the input and Z_t the output at instant t. Next state and outputs can be defined by a system of recurrent equations. Table 1 expresses dependencies for Mealy, Moore and GRAFCET models. Note the shift in time concerning states in Mealy and Moore machines.

Mealy			Moore			GRAFCET		
Y_{t+1}	$=$	$F(Y_t, X_t)$	Y_{t+1}	$=$	$F(Y_t, X_t)$	Y_t	$=$	$F(Y_{t-1}, X_t)$
Z_t	$=$	$G(Y_t, X_t)$	Z_t	$=$	$G(Y_t)$	Z_t	$=$	$G(Y_t, X_t)$

Table 1: Time dependencies in models

2.4 Preemption

In SYNCCHARTS, since a state behaves like an infinite loop (never ending emission), we need an *abortion* mechanism to leave a state. Abortion is a form of *preemption*. It can be either *weak* or *strong*. In SYNC-CHARTS, there are two types of arcs: Weak abortion is represented by an ordinary arrow ($\longrightarrow$) from the source state to the target state, whereas strong abortion is denoted by an arrow with a small circle ($\circ\!\!\rightarrow$) at the source state[2]. GRAFCET and STATECHARTS ignore this distinction, ARGOS suggested it, SYNCCHARTS is the first graphical model that encourages the use of both types of abortion arcs.

[2]This alteration in the drawing of a transition is not an arbitrary choice, it reflects the semantics of the strong abortion, as explained in the session devoted to semantics.

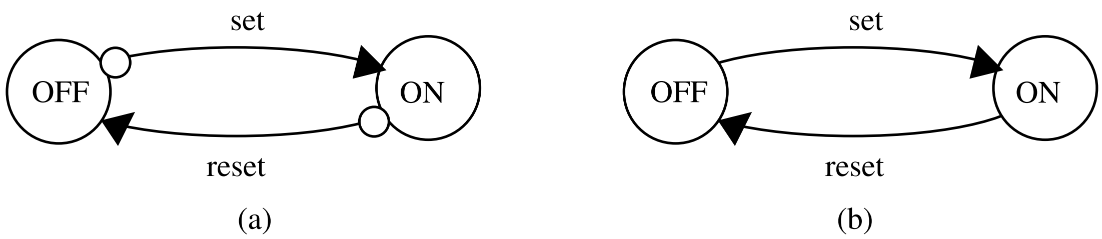

Figure 2: SYNCCHARTS for a SR flip-flop

instant	...	$k-1$	k	$k+1$	...
inputs	...		set		...
strong abortion	...	OFF	ON	ON	...
weak abortion	...	OFF	OFF.ON	ON	...

Table 2: Strong and weak abortions

Roughly speaking, strong abortion kills the process before executing it at the instant, whereas weak abortion lets the process execute at the current instant before killing it. A comparison of the behaviors with strong (Fig 2a) and weak (Fig 2b) abortions is given in Table 2. Obviously, in this case, only the strong abortion leads to the expected behavior.

2.5 Concurrency

GRAFCET, STATECHARTS and ARGOS support parallel composition. For SYNCCHARTS we adopt the notation introduced in STATECHARTS: Orthogonal state graphs are drawn in a box (a Macrostate) and they are separated by dotted lines. Each component graph has an initial state pointed to by an arrow. Each component evolves independently but synchronously.

Note that a macrostate in SYNCCHARTS is drawn as a rounded-corner box, with an optional name written in the header of the box (Fig.3).

2.6 Synchronization by local signals

In order to ensure a global behavior, concurrent processes are usually synchronized. All synchronous models allow synchronization by signals that are *instantaneously broadcast* system-wide. STATECHARTS, ARGOS and SYNCCHARTS support the declaration of *local signals*. Fig.3b is a SYNCCHARTS specifying a 3-bit binary counter. c_0 is an input signal, b_0, b_1, b_2 are three (boolean) output signals, c_1, c_2 (written in a cartouche) are local signals. Three T-type[3] flip-flops are composed in parallel. Synchronization is done by emission of local signals during a transition (e.g., c_1 is emitted by the transition from the state b_0 to the state b'_0). An emitted local signal may trigger other transition at the same instant (e.g., c_1 triggers transition from state b'_1 to state b_1). Like Mealy machine, STATECHARTS and ARGOS, SYNCCHARTS denotes signals emitted during a transition by the signal name preceded by a slash in the label associated with the transition (e.g., c_0/c_1).

2.7 Normal termination

A macrostate can terminate because the process inside this macrostate terminates. We call this termination mode "*normal termination*", in opposition to *forced terminations* induced by abortions. SYNCCHARTS provides a special arc to denote the normal termination. The head of the arrow is a triangle (⊳→). To deal with normal termination of concurrent processes, we define *final states* (distinguished by double concentric circles). A parallel composition normally terminates at an instant if and only if, each component is in a final

[3] T-type flip-flop = Toggle-type flip-flop, i.e., flip-flop that toggles each time its input T is set to 1 and memorizes otherwise.

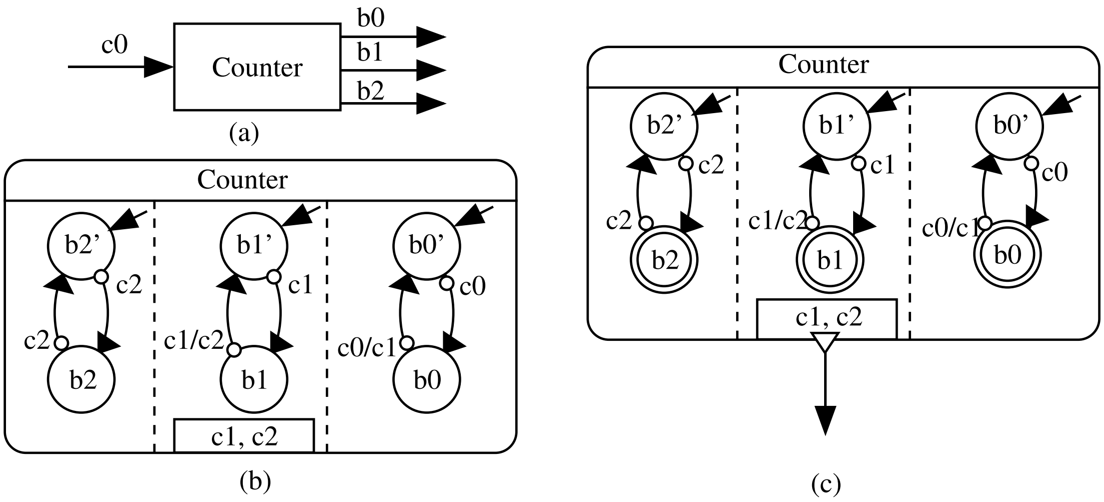

Figure 3: SyncCharts for a 3-bit binary counter

state at this instant. In Fig.3c, the macrostate counter is supposed to normally terminate when the code 111 is reached, i.e., after 7 occurrences of c_0.

2.8 Refinement

In StateCharts a state may be refined as a state-graph. Argos and SyncCharts take up this idea. This is a convenient way to handle hierarchy. In SyncCharts the refinement can be a textual description (an Esterel module), as well.

In Fig.4, the `ON`-state of the SR flip-flop, named `isON` has been refined as a state-graph with two states. The first state, named `Counter` is also refined. The system `Watchdog` starts up-counting the occurrences of `c0` as soon as `set` occurs. If the count reaches 8 before `reset` occurs, then `Alarm` is emitted. The occurrence of `reset` disables the counting. Note that each time state `isON` is entered, the counter is set to 000.

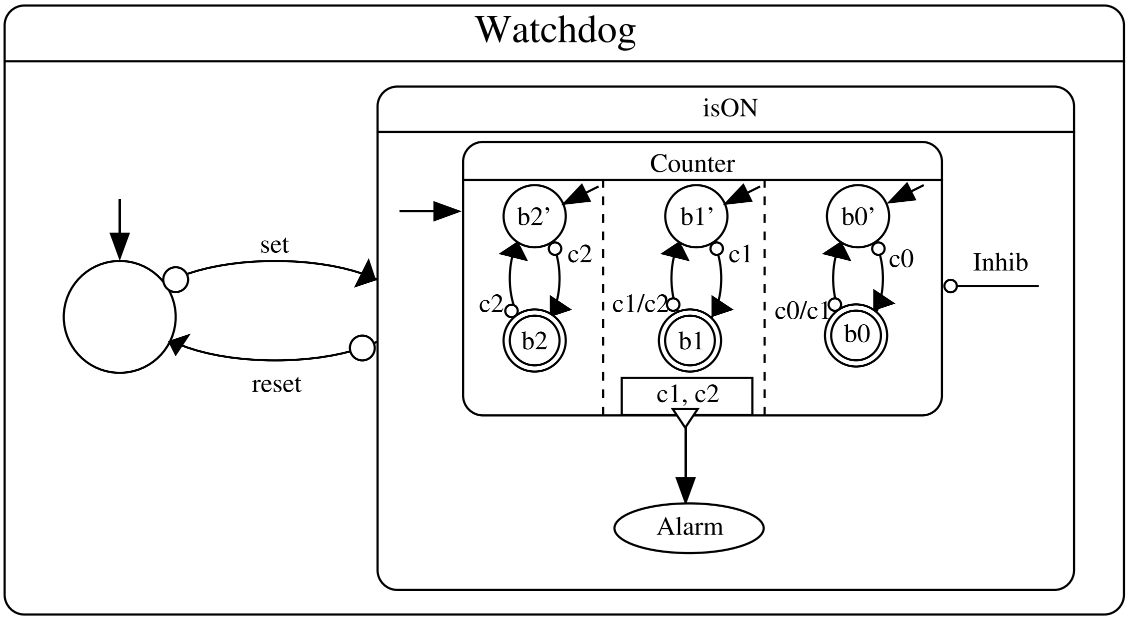

Figure 4: SyncCharts for a watchdog

2.9 Suspension

Often the evolutions of a subsystem has to be "frozen". In the above example, we inhibit up-counting by the presence of the signal `Inhib`. SyncCharts represents a *suspension* by a special arrow ending with a circle touching the state to be suspended. A label specifies the suspending event.

2.10 Transient states

Up to now, when entering a state, the control remains in this state for at least one instant. A state is said to be *transient* when it can be entered and left at the same instant. In Grafcet this behavior is not uncommon (transient activation of steps during a reaction). Transient states can be caused by *immediate* preemptions. An immediate preemption by a signal s takes account of the strict future occurrences of s, as well as the possible present occurrence.

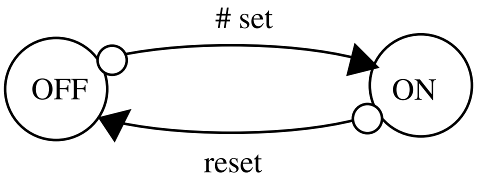

Figure 5: Immediate preemption

SyncCharts allows both delayed and immediate occurrences. The former is the default option, the latter requires a special prefix "#" before the triggering expression.

Suppose that `set` and `reset` may occur simultaneously and that, in this case, `set` has priority over `reset`. Fig.5 captures this expected behavior.

2.11 Other features

SyncCharts has other possibilities not shown in the above examples. When a state can be preempted in various ways, a *priority* is assigned to each outgoing arc, so that the behavior can be deterministic even if triggering events are not exclusive.

3 Process Algebra

In this section we introduce an abstract and formal description of reactive behaviors. This terse form allows precise and concise expression of complex behaviors. People familiar with the Esterel's semantics are used to this approach (see G. Berry's papers, e.g., [Ber92]).

3.1 Signals

A *reactive system* maintains permanent interactions with its environment by means of input and output *signals*. Let $\mathcal{I}$ be the set of input signals, and $\mathcal{O}$ the set of output signals of a reactive system. An *input event* is a subset I of $\mathcal{I}$, an *output event* O is a subset of $\mathcal{O}$. A *Synchronous Reactive System* (SRS) instantly reacts to input stimuli (input event I) by producing an output event O. An input history is a sequence $I_1, I_2, \ldots, I_n, \ldots$ of input events, and an output history is a sequence of $O_1, O_2, \ldots, O_n, \ldots$ of output events. The behavior of a SRS of sort $\{\mathcal{I}, \mathcal{O}\}$ can be characterized by a mapping from input histories into output histories:

$$B : I^* \longrightarrow O^*$$

Let $\mathbb{S}$ be a set of signals. $\perp$ is a distinguished element of $\mathbb{S}$; $\perp$ stands for a signal which is *never present* (the never occurring signal).

Definition 1 (Process) *A process on* $\mathbb{S}$ *is defined recursively by:*

1. 0 *(null)*
2. $s \in (\mathbb{S} - \{\perp\})$ is a process on $\mathbb{S}$, *(emission)*
3. If p and q are processes on $\mathbb{S}$, then $p \mid q$ is a process on $\mathbb{S}$, *(parallel)*
4. If p is a process on $\mathbb{S}$, then $p*$ is a process on $\mathbb{S}$ *(loop)*
5. If p is a process on $\mathbb{S}$ and $s \in (\mathbb{S} - \{\perp\})$, then $p \setminus s$ is a process on $\mathbb{S}$, *(restriction)*
6. If p is a process on $\mathbb{S}$, $s \in (\mathbb{S} - \{\perp\})$, and $t \in \mathbb{S}$, then $p\ [t/s]$ is a process on $\mathbb{S}$, *(renaming)*
7. If p is a process on $\mathbb{S}$ and $s \in \mathbb{S}$, then $s \supset p$ is a process on $\mathbb{S}$, *(suspension)*
8. If p, n, q are processes on $\mathbb{S}$ and $s \in \mathbb{S}$, then $s \nearrow p \rhd n, q$ is a process on $\mathbb{S}$, *(abortion)*

The first process is useful to build derived constructors. The next four constructors are classical imperative constructors. The sixth constructor is usual in process calculus. Note that $\perp$ can't be renamed, but an input signal can be renamed into $\perp$. The last two constructors are typically reactive.

Sort Transformations

Given p a process on $\mathbb{S}$, we associate three disjoint subsets of $\mathbb{S}$ with p: $\mathcal{I}_p, \mathcal{O}_p, \mathcal{L}_p$ respectively called the input, the output, the local sorts of p. Two auxiliary sets are introduced:

$$\mathcal{X}_p = \mathcal{I}_p \cup \mathcal{O}_p \quad \text{and} \quad \mathcal{S}_p = \mathcal{I}_p \cup \mathcal{O}_p \cup \mathcal{L}_p$$

$\mathcal{X}_p$ is the interface set of p, $\mathcal{S}_p$ is the sort of p.

These sets are defined inductively as follows:

pp	$\mathcal{X}_{pp}$	$\mathcal{O}_{pp}$	$\mathcal{L}_{pp}$
0	$\emptyset$	$\emptyset$	$\emptyset$
s	$\{s\}$	$\{s\}$	$\emptyset$
$p \mid q$	$\mathcal{X}_p \cup \mathcal{X}_q$	$\mathcal{O}_p \cup \mathcal{O}_q$	$\emptyset$
$p*$	$\mathcal{X}_p$	$\mathcal{O}_p$	$\emptyset$
$p \setminus s$	$\mathcal{X}_p - \{s\}$	$\mathcal{O}_p - \{s\}$	$\mathcal{L}_p \cup \{s\}$
$p\ [t/s]\ (s \in \mathcal{X}_p)$	$\mathcal{X}_p\ [t/s]$	$\mathcal{O}_p\ [t/s]$	$\emptyset$
$s \supset p$	$\mathcal{X}_p \cup \{s\}$	$\mathcal{O}_p$	$\emptyset$
$s \nearrow p \rhd n, q$	$\mathcal{X}_p \cup \mathcal{X}_q \cup \mathcal{X}_n \cup \{s\}$	$\mathcal{O}_p \cup \mathcal{O}_q \cup \mathcal{O}_n$	$\emptyset$

and for each operation:

$$\mathcal{I}_{pp} = \mathcal{X}_{pp} - \mathcal{O}_{pp} \text{ and } \mathcal{S}_{pp} = \mathcal{X}_{pp} \cup \mathcal{L}_{pp}$$

Note that our renaming is restrictive: only an input or output signal of a process can be renamed, not local signals.

Definition 2 (Event) *An event E is a subset of $\mathbb{S}$ without the never occurring signal: $E \subseteq \mathbb{S} - \{\perp\}$. Given a process p on $\mathbb{S}$, $I \subseteq \mathcal{I}_p$ is an input event, $O \subseteq \mathcal{O}_p$ is an output event.*

3.2 Semantics

The semantics is expressed by a behavioral semantics. The behavior of a process is a deterministic mapping from input sequences to output sequences. A reaction is interpreted as a process rewriting.

Given a process p and an input sequence $I_1, I_2, \cdots, I_n, \cdots$, the output sequence $O_1, O_2, \cdots, O_n, \cdots$ is computed as a chain of individual reactions:

$$p = p_1 \xmapsto[O_1]{I_1} p_2 \xmapsto[O_2]{I_2} \cdots p_n \xmapsto[O_n]{I_n} p_{n+1} \cdots$$

A transition $p_n \overset{I_n}{\underset{O_n}{\longmapsto}} p_{n+1}$ represents a single reaction. The $p \overset{I}{\underset{O}{\longmapsto}} p'$ relation is defined using an auxiliary relation

$$p \xrightarrow[E]{E_p,\, b} p'$$

defined by structural induction over p. E is the set of signals that p sees as being present, E_p is the set of signals that p emits when receiving E, and b is a boolean (termination bit) such that $b = \mathtt{tt}$ if p terminates and $b = \mathtt{ff}$ otherwise (p is said to wait).

The *broadcasting invariant* $E_p \subseteq E$ must be maintained during all the derivations.

Given a process p, an input event I:

$$p \overset{O}{\underset{I}{\longmapsto}} p' \text{ iff } p \xrightarrow[O \cup I]{O,\, b} p' \text{ for some } b$$

Rewriting Rules

$$0 \xrightarrow[E]{\emptyset,\, \mathtt{tt}} 0 \qquad \textit{(null)}$$

$$s \xrightarrow[E]{\{s\},\, \mathtt{tt}} 0 \qquad \textit{(emission)}$$

$$\frac{p \xrightarrow[E]{E_p,\, b_p} p' \qquad q \xrightarrow[E]{E_q,\, b_q} q'}{p \mid q \xrightarrow[E]{E_p \cup E_q,\, b_p \wedge b_q} p' \mid q'} \qquad \textit{(parallel)}$$

$$\frac{p \xrightarrow[E]{E_p,\, \mathtt{ff}} p'}{p* \xrightarrow[E]{E_p,\, \mathtt{ff}} \bot \nearrow p' \rhd (p*), 0} \qquad \textit{(loop)}$$

$$\frac{p \xrightarrow[E \cup \{s\}]{E_p,\, b} p' \qquad s \in E_p}{p \setminus s \xrightarrow[E]{E_p - \{s\},\, b} p' \setminus s} \qquad \textit{(restr1)}$$

$$\frac{p \xrightarrow[E - \{s\}]{E_p,\, b} p' \qquad s \notin E_p}{p \setminus s \xrightarrow[E]{E_p,\, b} p' \setminus s} \qquad \textit{(restr2)}$$

$$\frac{p \xrightarrow[E]{E_p,\, b} p' \qquad s \in \mathcal{X}_p}{p\ [t/s] \xrightarrow[E\ [t/s]]{E_p\ [t/s],\, b} p'\ [t/s]} \qquad \textit{(renam)}$$

$$\frac{s \in E}{s \supset p \xrightarrow[E]{\emptyset,\, \mathtt{ff}} s \supset p} \qquad \textit{(susp1)}$$

$$\frac{s \notin E \qquad p \xrightarrow[E]{E_p,\, \mathtt{tt}} p'}{s \supset p \xrightarrow[E]{E_p,\, \mathtt{tt}} 0} \qquad \textit{(susp2)}$$

$$\frac{s \notin E \qquad p \xrightarrow[E]{E_p,\,\mathtt{ff}} p'}{s \supset p \xrightarrow[E]{E_p,\,\mathtt{ff}} s \supset p'} \qquad \textit{(susp3)}$$

$$\frac{p \xrightarrow[E]{E_p,\,\mathtt{tt}} p' \qquad n \xrightarrow[E]{E_n,\,b} n'}{s \nearrow p \rhd n, q \xrightarrow[E]{E_p \cup E_n,\,b} n'} \qquad \textit{(abort1)}$$

$$\frac{s \in E \qquad p \xrightarrow[E]{E_p,\,\mathtt{ff}} p' \qquad q \xrightarrow[E]{E_q,\,b} q'}{s \nearrow p \rhd n, q \xrightarrow[E]{E_p \cup E_q,\,b} q'} \qquad \textit{(abort2)}$$

$$\frac{s \notin E \qquad p \xrightarrow[E]{E_p,\,\mathtt{ff}} p'}{s \nearrow p \rhd n, q \xrightarrow[E]{E_p,\,\mathtt{ff}} s \nearrow p' \rhd n, q} \qquad \textit{(abort3)}$$

Comments: We comment only three rules:

- (*renam*) short for renaming. $p\ [t/s]$ is the process p in which each occurrence of s is replaced by t. $E\ [t/s] = \big(E - \{s\}\big) \cup \{t\}$. As mentioned above, we rename only interface signals ($s \in \mathcal{X}_p$). There exists another restriction: if t is $\perp$ then s must not be an output signal of p. In other words:

$$p\ [t/s] \text{ is defined for } (s,t) \in \big(\mathcal{I}_p \times \mathbb{S}\big) \cup \big(\mathcal{O}_p \times (\mathbb{S} - \{\perp\})\big)$$

- (*abort1*) states that if p terminates then n is started at the same instant: the normal termination of p is followed by n. If p does not terminate we have two cases:
  - Either s is present (*abort2*): p is aborted after its execution and q starts executing at the same instant; q can be seen as an exception handler.
  - Or s is absent (*abort3*): then p normally proceeds.
- (*loop*). Note that p (the body of the loop) should not terminate instantaneously.

Remark: This set of operators is not minimal. For instance, "0" could have been defined as "$(s \setminus s)$". However, taking 0 as a primitive is simpler than deriving it from the somewhat complex restriction.

3.3 Derived constructors

For convenience and for compatibility with Esterel, new constructors are derived from the previous:

Derived imperative constructors

1	(pause)
$p; q$	(sequence)
$s?p, q$	(conditional)

They can be defined as:

$$\begin{aligned} 1 &\equiv (s \mid (s \supset 0)) \setminus s \\ p; q &\equiv \perp \nearrow p \rhd q, 0 \\ s?p, q &\equiv s \nearrow (s \supset 0) \rhd q, p \end{aligned}$$

with the following semantics obtained by composition of the rewriting rules of the primitives (Section 3.2):

$$1 \xrightarrow[E]{\emptyset,\,\mathtt{ff}} 0 \qquad \textit{(pause)}$$

$$\frac{p \xrightarrow[E]{E_p,\,\mathtt{ff}} p'}{p;\ q \xrightarrow[E]{E_p,\,\mathtt{ff}} p'\ ;\ q} \qquad \textit{(seq1)}$$

$$\frac{p \xrightarrow[E]{E_p,\,\mathtt{tt}} p' \qquad q \xrightarrow[E]{E_q,\,b} q'}{p;\ q \xrightarrow[E]{E_p \cup E_q,\,b} q'} \qquad \textit{(seq2)}$$

$$\frac{s \in E \qquad p \xrightarrow[E]{E_p,\,b_p} p'}{s?p, q \xrightarrow[E]{E_p,\,b_p} p'} \qquad \textit{(cond1)}$$

$$\frac{s \notin E \qquad q \xrightarrow[E]{E_q,\,b_q} q'}{s?p, q \xrightarrow[E]{E_q,\,b_q} q'} \qquad \textit{(cond2)}$$

1 waits for the next instant. $p;q$ executes p and then q, in sequence. The conditional $s?p, q$ executes either p or q according to the presence or the absence of s.

Derived reactive operators

They allow specific preemptions.

$$\begin{array}{lcll}
p : s \mathrel{\dot{>}} q & \equiv & s \nearrow p \rhd 0, q & \text{(weak abortion)} \\
p : s \mathrel{\dot{\gg}} q & \equiv & (s \mathrel{\dot{\supset}} p) : s \mathrel{\dot{>}} q & \text{(strong abortion)} \\
s \mathrel{\dot{\Rightarrow}} p & \equiv & (1*) : s \mathrel{\dot{>}} p & \text{(trigger)}
\end{array}$$

Comments: Because of the suspension of p by s, the strong abortion prevents p from executing at the instant when it is preempted.

Delayed operators

Up to now, we have considered immediate and future occurrences of signal, sometimes only strict future occurrences are desired. Suspension and abortion operators have their "delayed" counterparts:

$$\begin{array}{lccl}
\delta_s^d & \equiv & \big(1; (s?d, 0)\big)* & \text{(strict future)} \\
s \Rightarrow p & \equiv & 1; (s \mathrel{\dot{\Rightarrow}} p) & \text{(delayed trigger)} \\
s \supset p & \equiv & \Big(\big(\big(((d \mathrel{\dot{\supset}} p)\,;t) \mid \delta_s^d\big) \setminus d\big) : t \mathrel{\dot{>}} 0\Big) \setminus t & \text{(delayed suspension)} \\
p : s > q & \equiv & \Big(\big(\big((s \Rightarrow d) : d \mathrel{\dot{>}} 0\big) \mid (p : d)\big) : d \mathrel{\dot{>}} q\Big) \setminus d & \text{(delayed weak abortion)} \\
p : s \gg q & \equiv & (s \supset p) : s > q & \text{(delayed strong abortion)}
\end{array}$$

These formulations are adapted from the ESTEREL calculus. Simpler expressions will be given later on.

Starting and termination signals

Two local signals "α, ω"will be often associated with a process p: $p \equiv (\alpha; p; \omega) \setminus \alpha \setminus \omega$ provided $\alpha, \omega \notin \mathcal{X}_p$.

- α occurs at the very first instant of p activation,
- ω occurs at the termination of p.

For instance, the process that executes, after the termination of p, either q_α if p terminates at its first instant, or q_η otherwise, can be expressed as follows: $\big(\alpha; p; (\alpha?q_\alpha, q_\eta)\big)$.

3.4 Extending to compound signals

Definition 3 (Compound signal) *A compound signal on $\mathbb{S}$ is defined recursively as follows:*

- $\forall s \in \mathbb{S}$, s is a compound signal on $\mathbb{S}$,
- if σ_1 and σ_2 are compound signals on $\mathbb{S}$, so are $\sigma_1 + \sigma_2$ and $\sigma_1 \bullet \sigma_2$
- if σ is a compound signal on $\mathbb{S}$, so is σ'

Notation 1 *Let $\mathcal{C}_\mathbb{S}$ be the set of the compound signals on $\mathbb{S}$.*

Definition 4 (Truth value) *The truth value associated with a compound $\sigma \in \mathcal{C}_\mathbb{S}$ for a given $e \subseteq \mathbb{S}$ (noted $[\![\,\sigma\,]\!]_e$) is defined recursively as follows:*

- $\forall s \in \mathbb{S} : [\![\,s\,]\!]_e = (s \in e)$,
- $\forall \sigma_1, \sigma_2 \in \mathcal{C}_\mathbb{S} :\ [\![\,\sigma_1 + \sigma_2\,]\!]_e = [\![\,\sigma_1\,]\!]_e \vee [\![\,\sigma_2\,]\!]_e$ and $[\![\,\sigma_1 \bullet \sigma_2\,]\!]_e = [\![\,\sigma_1\,]\!]_e \wedge [\![\,\sigma_2\,]\!]_e$
- $\forall \sigma \in \mathcal{C}_\mathbb{S} : [\![\,\sigma'\,]\!]_e = \neg [\![\,\sigma\,]\!]_e$

Definition 5 (Satisfaction of a compound signal) *An event $e \subseteq \mathbb{S} - \{\bot\}$ satisfies $\sigma \in \mathcal{C}_\mathbb{S}$ (noted $e \models \sigma$) if $[\![\,\sigma\,]\!]_e$ evaluates to* `tt`*.*

Definition 6 (Support of a compound signal) *The support of $\sigma \in \mathcal{C}_\mathbb{S}$ (noted $\|\sigma\|$) is the set of the signals occurring in σ.*

The rules for *emission* and *restriction* are extended to subset of signals; the rules for *conditional*, *suspension* and *abortion* are extended to compound signals:

S	for $S \subseteq (\mathbb{S} - \{\bot\})$	(emission')
$p \setminus S$	for $S \subseteq (\mathbb{S} - \{\bot\})$	(restriction')
$\sigma?p, q$	for σ compound signal on $\mathbb{S}$	(conditional')
$\sigma \supset p$	for σ compound signal on $\mathbb{S}$	(suspension')
$\sigma \nearrow p \rhd n, q$	for σ compound signal on $\mathbb{S}$	(abortion')

Let $S = \{s_1, \cdots, s_n\}$. $p \setminus S \equiv \Big(\cdots (p \setminus s_1) \cdots\Big) \setminus s_n$.

Other extensions behave according to the following rules:

$$S \xrightarrow[E]{S,\,\mathtt{tt}} 0 \qquad \textit{(emission')}$$

$$\frac{E \models \sigma \qquad p \xrightarrow[E]{E_p,\,b_p} p'}{\sigma?p, q \xrightarrow[E]{E_p,\,b_p} p'} \qquad \textit{(cond1')}$$

$$\frac{E \not\models \sigma \qquad q \xrightarrow[E]{E_q,\, b_q} q'}{\sigma ?p, q \xrightarrow[E]{E_q,\, b_q} q'} \qquad \textit{(cond2')}$$

$$\frac{E \models \sigma}{\sigma \supset p \xrightarrow[E]{\emptyset,\, \mathtt{ff}} \sigma \supset p} \qquad \textit{(susp1')}$$

$$\frac{E \not\models \sigma \qquad p \xrightarrow[E]{E_p,\, \mathtt{tt}} p'}{\sigma \supset p \xrightarrow[E]{E_p,\, \mathtt{tt}} 0} \qquad \textit{(susp2')}$$

$$\frac{E \not\models \sigma \qquad p \xrightarrow[E]{E_p,\, \mathtt{ff}} p'}{\sigma \supset p \xrightarrow[E]{E_p,\, \mathtt{ff}} \sigma \supset p'} \qquad \textit{(susp3')}$$

$$\frac{p \xrightarrow[E]{E_p,\, \mathtt{tt}} p' \qquad n \xrightarrow[E]{E_n,\, b} n'}{\sigma \nearrow p \triangleright n, q \xrightarrow[E]{E_p \cup E_n,\, b} n'} \qquad \textit{(abort1')}$$

$$\frac{E \models \sigma \qquad p \xrightarrow[E]{E_p,\, \mathtt{ff}} p' \qquad q \xrightarrow[E]{E_q,\, b} q'}{\sigma \nearrow p \triangleright n, q \xrightarrow[E]{E_p \cup E_q,\, b} q'} \qquad \textit{(abort2')}$$

$$\frac{E \not\models \sigma \qquad p \xrightarrow[E]{E_p,\, \mathtt{ff}} p'}{\sigma \nearrow p \triangleright n, q \xrightarrow[E]{E_p,\, \mathtt{ff}} s \nearrow p' \triangleright n, q} \qquad \textit{(abort3')}$$

Remark: Compound signals allow simpler expressions of delayed operators:

$$\begin{array}{lcll} s \supset p & \equiv & \Big((\alpha' \bullet s) \supset (\alpha; p) \Big) \setminus \alpha \text{ provided } \alpha \notin \mathcal{X}_p & \text{(delayed suspension)} \\ p : s > q & \equiv & \Big((\alpha; p) : (\alpha' \bullet s) \gtrdot q \Big) \setminus \alpha \text{ provided } \alpha \notin \mathcal{X}_p & \text{(delayed weak abortion)} \end{array}$$

3.5 Generalized termination

3.5.1 Termination and Priority

A process p may either (normally) terminate, or be (weakly or strongly) aborted. A priority order can be assigned to those terminations.

We denoted the generalized termination by: $p : \sigma_1\, \rho_1\, q_1, \cdots, \sigma_n\, \rho_n\, q_n$ where $\rho_1, \cdots, \rho_n \in \{\gtrdot, >, \ggg, \gg\}$

For at most one j, $\sigma_j\, \rho_j\, q_j$ may be replaced by $\triangleright\ q_j$, standing for the normal termination. If p can normally terminate and no "$\triangleright q_j$" term is provided, then we assume that there is an implicit "$\triangleright 0$" term at the front of the termination list. This convention meets the one adopted for abortion in Sec.3.3:

$$s \nearrow p \triangleright 0, q \equiv p : s \gtrdot q \text{ is interpreted as } p : \triangleright\, 0, s \gtrdot q$$

Notation 2 (Highest priority) *Given a list of compounds on $\mathbb{S}$ arranged according to descending priority:*

$\{\tau_1, \cdots, \tau_n\}$, and an event E on $\mathbb{S}$, the predicate $E \underset{\{\tau_1,\cdots,\tau_n\}}{\widehat{\models}} \tau_k$ denotes that k is the smallest integer such that E satisfies an element of the list.

$$E \underset{\{\tau_1,\cdots,\tau_n\}}{\widehat{\models}} \tau_k \stackrel{def}{=} (\exists k : 1 \leq k \leq n) \ \big((E \models \tau_k) \wedge (\forall j : 1 \leq j < k) \ (E \not\models \tau_j)\big)$$

Where no ambiguity can arise, the subscript list is omitted.

3.5.2 Expected behavior

Let $GT_p = p : \sigma_1 \, \rho_1 \, q_1, \cdots, \sigma_n \ \rho_n \, q_n$.

Assume that only immediate terminations are used $\big((\forall k) \, \rho_k \in \{\gtrdot, \dot{\gg}, \rhd\}\big)$. Delayed abortions are easily treated by substituting $\alpha' \bullet \tau$ for τ (see below). Assume also that j is the priority of the normal termination. The expected behavior of the generalized termination is[4]:

$$\frac{(\widehat{E \models \sigma_k}) \ \ (op_k = \gtrdot) \ \ \begin{pmatrix} (k < j) \ \ (p \xrightarrow[E]{E_p, \mathtt{tt}} p') \\ p \xrightarrow[E]{E_p, \mathtt{ff}} p' \end{pmatrix} \ \ (q_k \xrightarrow[E]{E_q, b_q} q'_k)}{GT_p \xrightarrow[E]{E_p \cup E_q, b_q} q'_k} \qquad (GT\ 1)$$

$$\frac{(\widehat{E \models \sigma_k}) \ \ (op_k = \dot{\gg}) \ \ \begin{pmatrix} (k < j) \ \ (p \xrightarrow[E]{E_p, \mathtt{tt}} p') \\ p \xrightarrow[E]{E_p, \mathtt{ff}} p' \end{pmatrix} \ \ (q_k \xrightarrow[E]{E_q, b_q} q'_k)}{GT_p \xrightarrow[E]{E_q, b_q} q'_k} \qquad (GT\ 2)$$

$$\frac{\begin{pmatrix} (\forall k : 1 \leq k \leq n) \ \ (op_k \in \{\gtrdot, \dot{\gg}\}) \ \ (E \not\models \sigma_k) \\ (\widehat{E \models \sigma_k}) \ \ (k > j) \ \ (op_k \in \{\gtrdot, \dot{\gg}\}) \end{pmatrix} \ \ (p \xrightarrow[E]{E_p, \mathtt{tt}} p') \ \ (q_j \xrightarrow[E]{E_q, b_q} q'_j)}{GT_p \xrightarrow[E]{E_p \cup E_q, b_q} q'_j} \qquad (GT\ 3)$$

$$\frac{(\forall k : 1 \leq k \leq n) \ \ (op_k \in \{\gtrdot, \dot{\gg}\}) \ \ (E \not\models \sigma_k) \ \ (p \xrightarrow[E]{E_p, \mathtt{ff}} p')}{GT_p \xrightarrow[E]{E_p, \mathtt{ff}} GT_{p'}} \qquad (GT\ 4)$$

Process associated with the generalized termination

To deal with the more general case, we first transform this expression into an homogenous flattened form using the weak preemption only.

Definition 7 (Flattening) *Let $p, q_1, \ldots, q_n$ be processes on $\mathbb{S}$, $\sigma_1, \ldots, \sigma_n \in \mathcal{C}_{\mathbb{S}}$, and $\rho_1, \ldots, \rho_n \in \{\gtrdot, >, \dot{\gg}, \gg, \rhd\}$, the flattening of $p : \sigma_1 \ \rho_1 \ \ q_1, \cdots, \sigma_n \ \rho_n \ q_n$ is the process:*

$$\mathrm{Flat}(p : \sigma_1 \ \rho_1 \ \ q_1, \cdots, \sigma_n \ \rho_n \ q_n) \equiv \Big(\alpha; \big(\iota \supset (p; \omega)\big) : \tau_1 \ \gtrdot \ \kappa_1, \cdots, \tau_n \ \gtrdot \kappa_n\Big) \setminus \iota \setminus \alpha \setminus \omega$$

[4]Notation: The numerator of a rule is a conjunction of predicates. Stacked predicates denote disjunction of predicates.

provided $\iota, \alpha, \omega \notin \left(\mathcal{X}_p \cup \left(\bigcup_{k=1}^{k=n} \mathcal{X}_{q_k} \right) \right)$ and where

$< \rho_k >$	$< \tau_k >$	$< \kappa_k >$
$\gtrdot$	σ_k	q_k
$>$	$\alpha' \bullet \sigma_k$	q_k
$\mathrel{\cdot\!\!\gg}$	σ_k	$(\iota; q_k)$
$\gg$	$\alpha' \bullet \sigma_k$	$(\iota; q_k)$
$\triangleright$	ω	q_k

The generalized termination can be expressed as:

$$
\begin{array}{l}
p \ : \ \sigma_1 \ \rho_1 \ q_1, \cdots, \sigma_n \ \rho_n \ q_n \quad \equiv \\
\quad (\zeta \nearrow (\\
\qquad\qquad \left(\alpha; (\xi \nearrow (\iota \supset p) \triangleright \omega, 0) \right) \\
\qquad\qquad | \\
\qquad\qquad (\\
\qquad\qquad\quad (\\
\qquad\qquad\qquad (\tau_1 \, ? \, (\xi; \kappa_1; \zeta) \, , 0) \, ; \\
\qquad\qquad\qquad\qquad\qquad \vdots \\
\qquad\qquad\qquad (\tau_n \, ? \, (\xi; \kappa_n; \zeta) \, , 0) \, ; \\
\qquad\qquad\qquad 1 \\
\qquad\qquad\quad) \, * \\
\qquad\qquad) \\
\qquad) \triangleright 0, 0 \\
\quad) \setminus \alpha \setminus \omega \setminus \iota \setminus \xi \setminus \zeta
\end{array}
$$

with the above notations and provided $\xi, \zeta \notin \mathcal{X}_p \cup \bigcup_{k=1}^{k=n} \mathcal{X}_{q_k}$.

Proof: Left as an instructive but boring exercise[5].

Comments: An informal description of the behavior is:

- As soon as a triggering compound (e.g. τ_k) turns to present ξ is emitted and q_k begins execution. ξ causes the abortion of p which rewrites to 0. When q_k completes its execution, ζ is emitted and aborts the generalized termination.
- Priority is enforced by the sequential evaluation of the triggering compounds.
- If p normally terminates then ω is emitted so that the continuation associated with the normal termination (e.g. q_j) executes. When q_j completes its execution, ζ is emitted and aborts the generalized termination.
- If at the current instant, no trigger occurs then p normally executes in the first branch of the parallel, whereas the second branch waits (1). At the next instant, triggers are reevaluated thanks to the loop in the second branch.

4 The Graphical Model

SyncCharts is a state-based description of the reactive behaviors. It must support states, hierarchy of states, concurrency, transitions of several types and even textual annotations. Roughly, a macrostate is either a (textual description of a) process or a parallel composition of state-graphs. Each state graph is a collection of one or several interconnected states, with initial and, possibly, final states. In turn, a macrostate can be substituted for a state. Instead of this top-down approach to SyncCharts, we adopt a bottom-up presentation of the various components.

[5] Wanted! a friendly rewriting system for making proofs in a reasonable amount of time.

4.1 Graphical elements

4.1.1 Star

The graphical basic block is the *state*. In fact, this block is not only the expression of some local invariant behavior (its *body*) but also a full description of the ways to leave this state. So, the basic block is both a classical state and its outgoing arcs (Fig. 6). We would rather call it a *star* (with outgoing arcs seen as beams)[6].

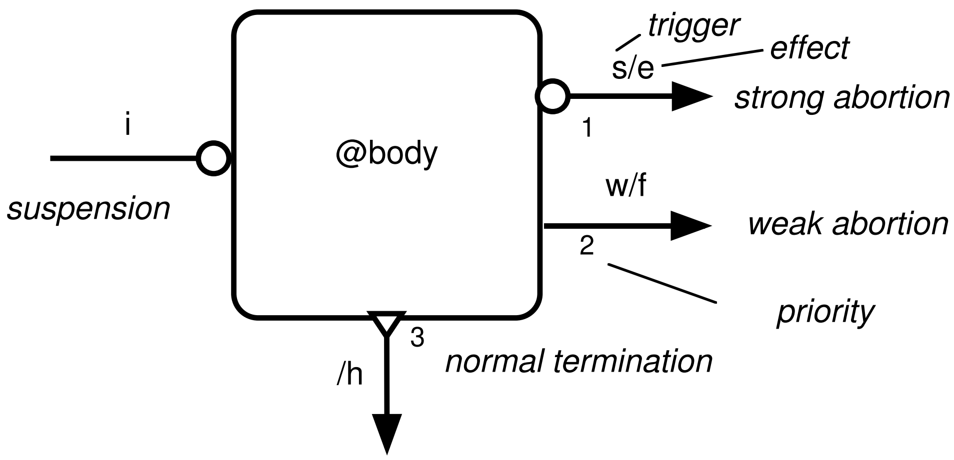

Figure 6: Basic Block: a Star

A star is drawn as a rounded rectangle with its "beams". Arcs are numbered according to a priority ordering (the less, the highest priority). The weak abortion is expressed by a plain arrow ($\longrightarrow$). The normal termination, i.e., leaving the star because its body terminates, is drawn as an arrow with a leading triangle ($\triangleright\!\!\rightarrow$). The suspension is denoted by a special dangling incoming arc with a circle head ($-\!\!\circ$). The strong abortion which is a combination of weak abortion and suspension, is specified by an hybrid arrow ($\circ\!\!\rightarrow$).

A star with n outgoing arcs (n beams) is said to be a n-star. There exists a special kind of star with no beam. It is a 0-star. There is no way to leave a 0-star but by a higher level abortion.

4.1.2 Constellation

States are interconnected to make a state graph; Stars are interconnected to make a *constellation*. Figure 7 shows an instance of a constellation with one initial star and one final star. There may be 0, 1 or several final stars. Final stars are distinguished by double-line rounded rectangles. There must be at least one initial star. Initial stars are identified by a dangling incoming arrow. A constellation need not be a connected graph. A constellation can be made of a single 0-star which is implicitly an initial star.

A constellation made of n stars is said to be a n-constellation.

4.1.3 Macrostate

A parallel composition of constellations is a *firmament* or a *macrostate*[7]. The components are delimited by dashed lines. There may be a single constellation in a macrostate. In this case, dashed lines are omitted. A macrostate may have local signals. A macrostate has an optional *name*. It is drawn as a rounded rectangle with a *header* in which the optional name is written. See Fig. 8.

A macrostate with n orthogonal constellation is said to be a n-macrostate.

[6] Reactive systems have been compared to *cacti*, in our opinion stars are less aggressive and more poetical.

[7] Firmament is a nice word whose etymology "*firmamentum*" means "support" (the vault on which stars are fixed). Contrary to star and constellation used elsewhere in mathemathics, firmament is not usual; We shall refer to as macrostate instead.

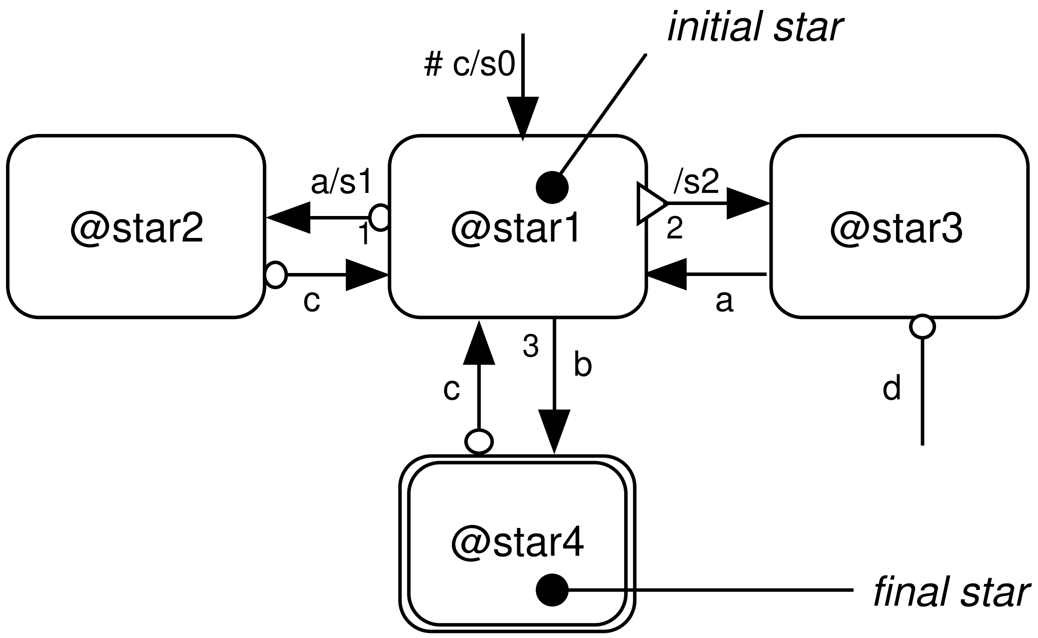

Figure 7: A Constellation

4.1.4 Terminal states

In a Moore machine, outputs are associated with states. In our synchronous approach we have to emit these signals at each instant. A Moore-like state is a special star whose body is a simple process: $(S;1)*$, where $S \subseteq \mathbb{S}$. For convenience, we introduce a short hand notation: an oval with the set of the signals to be emitted written inside. We even allow conditional emission: the annotation "$s?A,B$" inside an oval, stands for a star whose body is the process $\big((s?A,B);1\big)*$. Either the `then` clause or the `else` clause can be omitted. Finally, an oval with no inscription inside stands for a never ending process, doing nothing forever $(1*)$.

4.1.5 Arc labelling

An arc is labelled by a pair composed of a compound signal and a subset of signals. The first component is the *trigger*, the second component is the *effect*. They are separated by a "/". One or both components can be omitted. Note that normal termination ($\rhd\!\!\rightarrow$ arcs) has not explicit trigger.

The label can be prefixed by the symbol "#". In this case, we use *immediate* preemption, instead of *delayed* preemption which is the default option.

4.2 Formal Description

In this subsection we describe the internal structures of a SyncCharts.

Definition 8 (Hierarchy) *Given a SyncCharts Σ, its* hierarchical structure *is described by a tree T_Σ where vertices belong to three disjoint sets of components: M_Σ, C_Σ, and S_Σ, called its macrostate set, its constellation set, and its star set, respectively.*

This tree has a specificity: macrostate, constellation, and star repeatedly appear in this order, while going down to the leaves. A SyncCharts is a macrostate. This macrostate is the root of a tree. A macrostate vertex is a AND-node: its successors are orthogonal constellations. A constellation node is a OR-node: its successors are exclusive stars. A star node has only one successor: its body which is a macrostate. Of course, an AND-node or an OR-node may degenerate to a single-successor node. Leaves of the tree are macrostates.

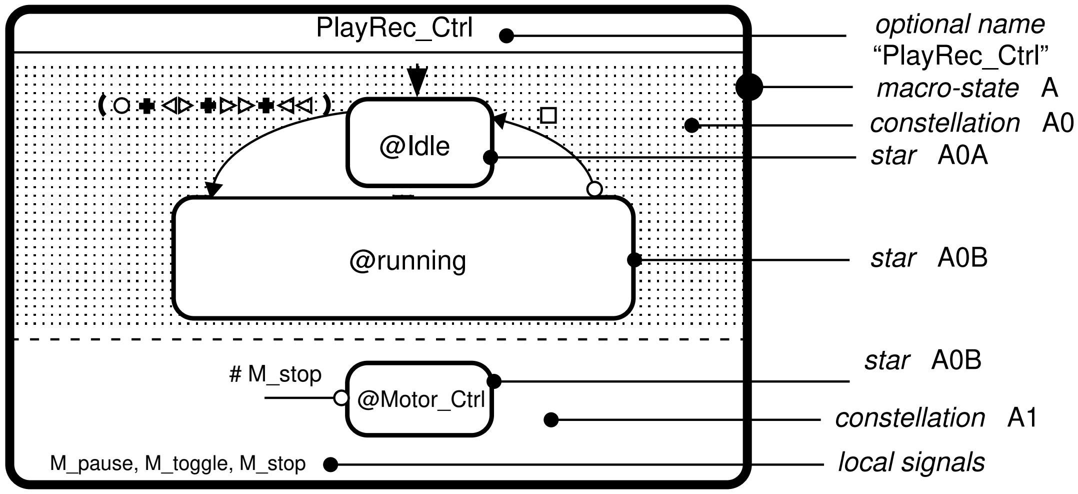

Figure 8: A macrostate

4.2.1 Naming Conventions

A name can be given to a macrostate. This is an external or user's identifier. We need a non ambiguous *internal name* for each star, constellation and macrostate. We adopt a naming convention that reflects the hierarchy of the SyncCharts. Moreover, states in a constellation have to be ordered (see below). So, the naming will also reflect this ordering. We use two distinct alphabets: the latine letters ([A..Z]) for stars and the decimal digits ([0..9]) for constellations. We use the lexicographical order on [A..Z0..9]. Since a star is followed by a unique macrostate we assign the same name to the star and its body macrostate. By convention, if id is the internal name of a star, (id) is the internal name of its body macrostate. According to these conventions, the root name is (A). The highest level constellation names are $A0, A1, \cdots, A9, A10, \cdots$. The stars in the $A3$ constellation are $A3A, A3B, \cdots, A3Z, A3AA, \cdots$. The first colomn in Table. 3 shows the names of the components of the "Watchdog" SyncCharts.

Notation 3 (Internal name) *Let v be a component of a SyncChart (a macrostate, a constellation, or a star),* $\texttt{i-name}_v$ *denotes the internal name associated with v.*

If v is a star then $\texttt{i-name}_v \in ([A..Z]^+[0..9]^+)^*[A..Z]^+$.
If v is a constellation then $\texttt{i-name}_v \in ([A..Z]^+[0..9]^+)^+$.

Notation 4 (Successor) *Given a SyncChart Σ, let $\mathcal{T}_\Sigma$ be the tree associated with Σ. Let v be a vertex of $\mathcal{T}_\Sigma$. v° denotes the set of the immediate successors of v in $\mathcal{T}_\Sigma$.*

Remark: if $v \in M_\Sigma$ then $v^\circ \subset C_\Sigma$, if $v \in C_\Sigma$ then $v^\circ \subset S_\Sigma$, and if $v \in S_\Sigma$ then $v^\circ \subset M_\Sigma$ and $v^\circ = \{M\}$ for some M.

4.2.2 Structure of an Arc

An arc has a type (weak or strong abortion, normal termination, suspension, or initial), a modifier (immediate or delayed), a triggering compound signal and an effect subset of signals not including $\perp$. We introduce new types:

$$\begin{aligned}
\texttt{ARC_TYPE} &::= \{\ \texttt{weak_abortion, strong_abortion,} \cdots \\
&\qquad\qquad \cdots \texttt{normal_termination, suspension, initial}\ \} \\
\texttt{ARC_MOD} &::= \{\ \texttt{immediate, delayed}\ \} \\
\texttt{ARCS}_{\mathbb{S}} &::= \texttt{ARC_TYPE} \times \texttt{ARC_MOD} \times (\mathcal{C}_{\mathbb{S}} \cup (\texttt{void})) \times \left(2^{\mathbb{S}-\{\perp\}} \cup (\texttt{void})\right)
\end{aligned}$$

Notation 5 (Arc) *Let* $a \in \texttt{ARCS}$. $a = < a_{typ}, a_{mod}, a_{trig}, a_{eff} >$.

Subtypes are distinguished for $\texttt{ARCS}_{\mathbb{S}}$:

- $\texttt{ARCS}_{\mathbb{S}}^{susp}$ for *suspension arcs*, such that $\forall a \in \texttt{ARCS}_{\mathbb{S}}^{susp}$:

$$a_{typ} = \texttt{suspension}, a_{trig}\ \texttt{is not (void)}\ , a_{eff}\ \texttt{is (void)}$$

- $\texttt{ARCS}_{\mathbb{S}}^{norm}$ for *normal termination arcs*, such that $\forall a \in \texttt{ARCS}_{\mathbb{S}}^{norm}$:

$$a_{typ} = \texttt{normal\_termination}, a_{mod} = \texttt{immediate}, a_{trig}\ \texttt{is (void)}\ , a_{eff}\ \texttt{is not (void)}$$

- $\texttt{ARCS}_{\mathbb{S}}^{init}$ for *initial arcs*, such that $\forall a \in \texttt{ARCS}_{\mathbb{S}}^{init}$:

$$a_{typ} = \texttt{initial}, a_{mod} = \texttt{immediate}$$

- $\texttt{ARCS}_{\mathbb{S}}^{abort}$ for *transition arcs*, such that $\forall a \in \texttt{ARCS}_{\mathbb{S}}^{abort}$:

$$a_{typ} \in \{\texttt{weak\_abortion, strong\_abortion}\}, a_{trig}\ \texttt{is not (void)}\ , a_{eff}\ \texttt{is not (void)}$$

4.2.3 Structure of a Macrostate

Let M be a macrostate of a SyncCharts Σ. Either M is the root and its internal name is (A), or M is the body of some star S and $\texttt{i-name}_M = (\texttt{i-name}_S)$.

Information attached to a macrostate M

- $\texttt{Name}_M$ (optional identifier)
- $\texttt{Local}_M \subseteq (\mathbb{S} - \{\perp\})$ (local signals)
- $M^\circ \subseteq C_\Sigma$ (set of constellation components)

Constraint

- $M^\circ \neq \emptyset$

Signal sets of M

$$\begin{aligned}
\mathcal{L}_M &= \texttt{Local}_M \cup \left(\bigcup_{c \in M^\circ} \{\omega_c\} \right) \text{ provided } \left(\bigcup_{c \in M^\circ} \omega_c \right) \cap \left(\bigcup_{c \in M^\circ} \mathcal{X}_c \right) = \emptyset \\
\mathcal{X}_M &= \left(\bigcup_{c \in M^\circ} \mathcal{X}_c \right) - \mathcal{L}_M \\
\mathcal{O}_M &= \left(\bigcup_{c \in M^\circ} \mathcal{O}_c \right) - \mathcal{L}_M \\
\mathcal{I}_M &= \mathcal{X}_M - \mathcal{O}_M
\end{aligned}$$

Remark: Signals ω_c are emitted by the constellation components and are used for the termination of the macrostate. ω_c is present whenever constellation c is in a final star. These internal signals are hidden from the user.

4.2.4 Structure of a Constellation

Let C be a constellation of a SyncCharts Σ.

Information attached to a constellation C

- $C^\circ \subseteq S_\Sigma$ (set of the component stars)
- $\texttt{in-degree}_C \in \mathbb{N}$ (number of initial arcs)
- $\Gamma_C : \left(\bigcup_{s \in C^\circ} (\{s\} \times [1..\texttt{out-degree}_s]) \right) \longrightarrow C^\circ$ (connections)
- $\Gamma_C^i : [1..\texttt{in-degree}_C] \longrightarrow \texttt{ARCS}_{\mathbb{S}}^{init} \times C^\circ$ (initial arcs)
- $\Gamma_C^f \subseteq C^\circ$ (final stars)

Constraint

- $\texttt{in-degree}_C \geq 1$

Signal sets of C

$$\begin{array}{rcl}
\mathcal{L}_C & = & \bigcup_{s \in C^\circ} \{\gamma_s\} \;\; \text{provided } \gamma_s \notin \bigcup_{k \in C^\circ} \mathcal{X}_k \\
\mathcal{X}_C & = & \left(\bigcup_{s \in C^\circ} \mathcal{X}_s \right) \cup \{\omega_C\} \;\; \text{provided } \omega_C \notin \bigcup_{s \in C^\circ} \mathcal{X}_s \\
\mathcal{O}_C & = & \left(\bigcup_{s \in C^\circ} \mathcal{O}_s \right) \cup \{\omega_C\} \\
\mathcal{I}_C & = & \mathcal{X}_C - \mathcal{O}_C
\end{array}$$

Remark: The signals ω_C and γ_s are emitted by C. ω_C is used for the synchronized termination of parallel constellation. γ_s is used to pass the control from a star to another star. γ_s can be interpreted as "go to star s". These internal signals are hidden from the user.

4.2.5 Structure of a Star

Let S be a star of a SyncCharts Σ.

Information attached to a star S

- $\texttt{Body}_S \in M_\Sigma$ such that $S^\circ = \{\texttt{Body}_S\}$ (body of S)
- $\texttt{out-degree}_S \in \mathbb{N}$ (number of beams)
- $\texttt{Inhibit}_S \in \texttt{ARCS}_{\mathbb{S}}^{susp}$ (suspension)
- $\texttt{Beams}_S : [1..\texttt{out-degree}_S] \longrightarrow \left(\texttt{ARCS}_{\mathbb{S}}^{abort} \cup \texttt{ARCS}_{\mathbb{S}}^{norm} \right)$ (outgoing arcs)

Constraint

- There is at most one $k \leq \texttt{out-degree}_S$ such that $\texttt{Beams}_S(k) \in \texttt{ARCS}_{\mathbb{S}}^{norm}$.

Signal sets of C Let p be the body of S ($\{p\} = S^{\circ}$), h be $\texttt{Inhibit}_S$, and n be $\texttt{out-degree}_S$. S is a n-star.

$$
\begin{aligned}
\mathcal{L}_S &= \{\alpha, \omega, \iota\} \text{ provided } \alpha, \omega, \iota \notin \mathcal{X}_p \\
\mathcal{X}_S &= \mathcal{X}_p \cup \|h_{trig}\| \cup \left(\bigcup_{\substack{k \in [1..n] \\ a = \texttt{Beams}_S[k]}} \left(\|a_{trig}\| \cup \|a_{eff}\| \cup \{\xi_k\} \right) \right) \text{ provided } \xi_k \notin \mathcal{X}_p \\
\mathcal{O}_S &= \mathcal{O}_p \cup \left(\bigcup_{\substack{k \in [1..n] \\ a = \texttt{Beams}_S[k]}} \left(\|a_{eff}\| \cup \{\xi_k\} \right) \right) \\
\mathcal{I}_S &= \mathcal{X}_S - \mathcal{O}_S
\end{aligned}
$$

Remark: The signals α, ω_C, ι, and ξ_k are emitted by S and they are used for internal synchronization. ι is a signal that causes the inhibition (the suspension) of the body. ξ_k (eXit signal) is the signal emitted when leaving the star S through the beam whose priority is k. These internal signals are hidden from the user.

4.3 Example

Fig.9 shows the structuration of the Watchdog application (Section.2.8).

`Table 3: Components of the SyncCharts that specifies Watchdog`

i-name	type	modif	$\mathcal{I}_N$	$\mathcal{O}_N$	$\mathcal{L}_N$
A0B0A0A	1-star	I	$\{c_2\}$	$\{b'_2\}$	$\emptyset$
A0B0A0B	1-star	F	$\{c_2\}$	$\{b_2\}$	$\emptyset$
A0B0A0	2-const.		$\{c_2\}$	$\{b'_2, b_2\}$	$\emptyset$
A0B0A1A	1-star	I	$\{c_1\}$	$\{b'_1\}$	$\emptyset$
A0B0A1B	1-star	F	$\{c_1\}$	$\{b_1, c_2\}$	$\emptyset$
A0B0A1	2-const.		$\{c_1\}$	$\{b'_1, b_1, c_2\}$	$\emptyset$
A0B0A2A	1-star	I	$\{c_0\}$	$\{b'_0\}$	$\emptyset$
A0B0A2B	1-star	F	$\{c_0\}$	$\{b_0, c_1\}$	$\emptyset$
A0B0A2	2-const.		$\{c_0\}$	$\{b'_0, b_0, c_1\}$	$\emptyset$
(A0B0A)	3-macro		$\{c_0\}$	$\{b'_2, b_2, b'_1, b_1, b'_0, b_0\}$	$\{c_2, c_1\}$
A0B0A	1-star	I,S	$\{c_0, Inhib\}$	$\{b'_2, b_2, b'_1, b_1, b'_0, b_0\}$	$\emptyset$
A0B0B	0-star		$\emptyset$	$\{Alarm\}$	$\emptyset$
A0B0	2-const.		$\{c_0, Inhib\}$	$\{b'_2, b_2, b'_1, b_1, b'_0, b_0, Alarm\}$	$\emptyset$
A0B	1-star		$\{c_0, Inhib, reset\}$	$\{b'_2, b_2, b'_1, b_1, b'_0, b_0, Alarm\}$	$\emptyset$
A0A	1-star	I	$\{set\}$	$\emptyset$	$\emptyset$
A0	2-const.		$\{c_0, Inhib, reset, set\}$	$\{b'_2, b_2, b'_1, b_1, b'_0, b_0, Alarm\}$	$\emptyset$
A	0-star	I	$\{c_0, Inhib, reset, set\}$	$\{b'_2, b_2, b'_1, b_1, b'_0, b_0, Alarm\}$	$\emptyset$

4.4 Grammar

`Table 4: Grammar of SyncCharts`

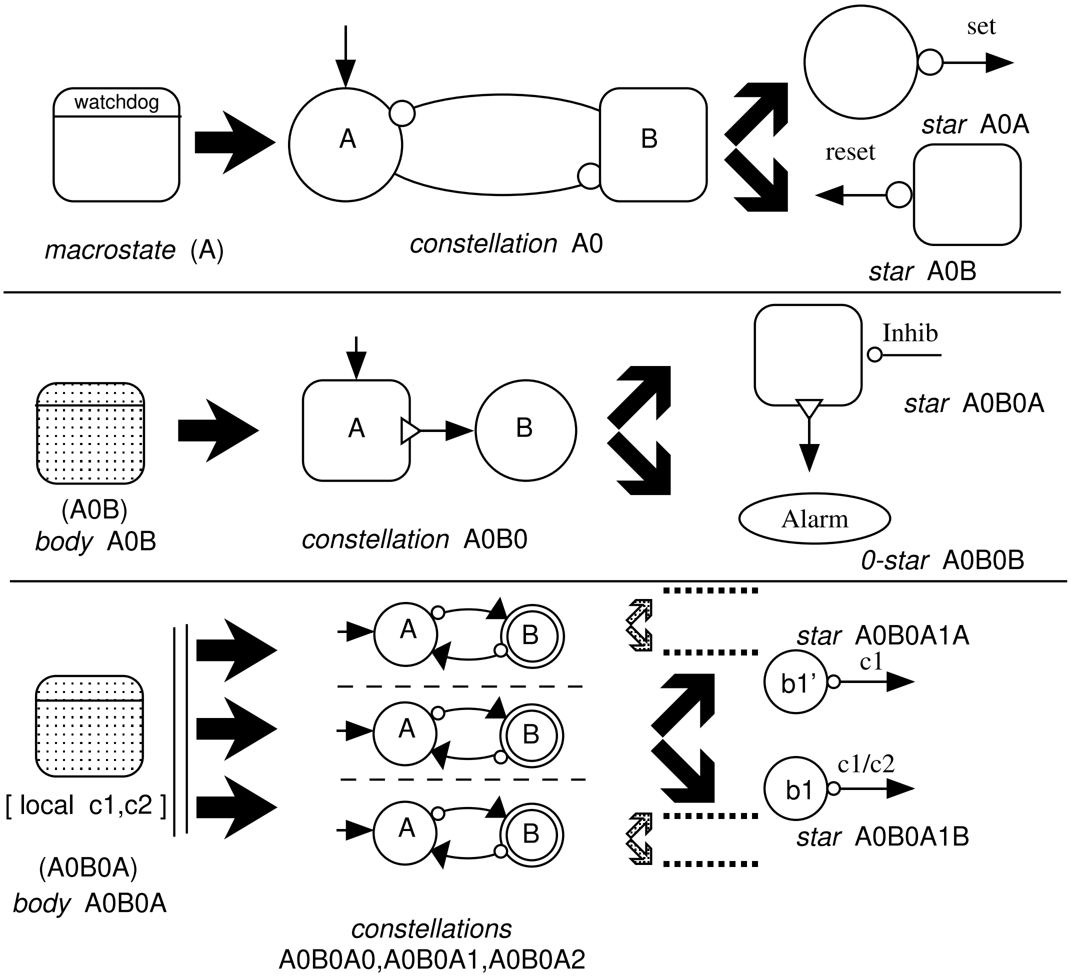

Figure 9: Structuration of Watchdog

```
SyncCharts       ::=  macrostate

macrostate       ::=  [ident]
                      [local_signals]
                      (
                           constellation
                           |
                           parallelcomp ⋮ constellation
                      )

parallelcomp     ::=  constellation
                      | parallelcomp ⋮ constellation

constellation    ::=  (
                           ( stars
                             initial_arcs )
                           |
                           0_star
                      )
                      [final_stars]

0_star           ::=  [ suspension_arc ]
                      body

star             ::=  exit_arcs
                      [ suspension_arc ]
                      body

stars            ::=  star
                      | stars star

body             ::=  process | macrostate

suspension_arc   ::=  [#] trigger

initial_arc      ::=  [# trigger] [/ effect]

abort_arc        ::=  ( strong_abortion | weak_abortion)
                      [ [#] trigger ] [/ effect]

norm_arc         ::=  normal_termination
                      [/ effect]

exit_arc         ::=  ( abort_arc | norm_arc )

initial_arcs     ::=  initial_arc
                      | initial_arcs initial_arc

exit_arcs        ::=  exit_arc
```

continued on next page

continued from previous page

		| exit_arcs exit_arc
strong_abortion	::=	$\circ\!\!\rightarrow$
weak_abortion	::=	$\longrightarrow$
normal_termination	::=	$\triangleright\!\!\rightarrow$
suspension	::=	$-\!\!\circ$
trigger	::=	compound
effect	::=	signal | effect , signal
factor	::=	signal | signal' | (compound)'
term	::=	factor | term • factor
compound	::=	term | compound + term
local_signals	::=	signal | local_signals , signal
signal	::=	ident

5 Process Associated with a SyncCharts

The previous section defined the syntax of SyncCharts, but not its semantics. We now explain how to generate the process that expresses the meaning of a SyncCharts. This process is built inductively. We shall describe star, constellation and macrostate processes successively. First, we have to interpret triggers and effects.

5.1 Triggers and Effects

Trigger

Syntactically, the trigger of an initial arc or an abortion arc may be omitted. We interpret this omission as a compound that is always satisfied.

Definition 9 *The default for the trigger field of an initial arc or an abortion arc is interpreted as* $\perp'$

Therefore, the default for a trigger is the always present signal. For convenience, we adopt a new symbol for an always present signal[8]: "$\top$".

Recall that a suspension arc always has an explicit compound trigger and that a normal termination has a `(void)` trigger field.

Effect

An effect is a subset of $\mathbb{S} - \{\perp\}$. The semantics of an effect E is the emission of all signals in E. The default effect is the empty set.

Definition 10 *The default for the effect field is interpreted as* $\emptyset$

5.2 Star

5.2.1 Presentation

A star is characterized by its body and its beams. The body of a star describes the *activity* of a star (what it does). A special case is that of a star that may burn out. Such a "dark" star loses all visible activity, but still

[8] In Esterel `tick` is the always present signal.

exists. On the contrary, a star can be destroyed (either externally enforced destruction, or self-destruction). Beams represent potential destructions of the star.

The activity of a star may also be suspended (it is only an eclipse, not a permanent extinction).

5.2.2 Expected behavior of a star

The behavior of a star is akin to a generalized termination (Section 3.5) in which handlers are instantaneous processes that, possibly, emit signals. Nevertheless, there are two differences: the activity of a star can be suspended by a triggering compound, and the continuation after the normal termination of the body of the star can be absent.

Let p be the body of the star, τ_0 be its suspending triggering compound, $\tau_1\,\rho_1\,\upsilon_1, \cdots, \tau_n\,\rho_n\,\upsilon_n$ be its termination list. The terms $\tau_k\,\rho_k\,\upsilon_k$ are arranged according to descending priority of the beams. τ_k (υ_k) is the trigger (the effect, resp.) of the k^{th} beam. $\rho_1, \cdots, \rho_n \in \{\gtrdot, >, \dot{\gg}, \gg, \rhd\}$ and finally, $\rho_0 \in \{\cdot\!\supset, \supset\}$ characterizes the type of suspension.

Given a star S, $STAR_S(p)$ denotes the process associated with S, where p is the body of S. Unlike the τ's, the ρ's and the υ's which are static information, p changes during the life of the star. That is the reason why p is used as a parameter.

Without loss of generality, assume that only immediate operators are used. If there is no normal termination j is set to $n+1$, otherwise j is assigned the priority index of the unique normal termination. The expected behavior of the star is:

$$\frac{(\widehat{E \models \tau_k})\;\;(\rho_k = \gtrdot)\;\;\left(\begin{array}{ll} (E \not\models \tau_0) & \left(\begin{array}{c} (k<j)\;\;(p \xrightarrow[E]{E_p,\mathtt{tt}} p') \\ \\ (p \xrightarrow[E]{E_p,\mathtt{ff}} p') \end{array}\right) \\ \\ (E \models \tau_0) & (E_p = \emptyset) \end{array}\right)}{STAR_S(p) \xrightarrow[E]{E_p \cup \upsilon_k \cup \{\xi_k\},\mathtt{tt}} 0} \qquad \textit{(Star 1)}$$

$$\frac{(\widehat{E \models \tau_k})\;\;(\rho_k = \dot{\gg})\;\;\left(\begin{array}{c} (E \not\models \tau_0)\;\;\left(\begin{array}{c} (k<j)\;\;(p \xrightarrow[E]{E_p,\mathtt{tt}} p') \\ \\ (p \xrightarrow[E]{E_p,\mathtt{ff}} p') \end{array}\right) \\ \\ (E \models \tau_0) \end{array}\right)}{STAR_S(p) \xrightarrow[E]{\upsilon_k \cup \{\xi_k\},\mathtt{tt}} 0} \qquad \textit{(Star 2)}$$

$$\frac{\left(\begin{array}{c} (\forall k:1..n)\;(\rho_k \in \{\gtrdot, \dot{\gg}\})\;(E \not\models \tau_k)\;(j \le n) \\ \\ (\widehat{E \models \tau_k})\;(k>j)\;(\rho_k \in \{\gtrdot, \dot{\gg}\}) \end{array}\right)\;\;(E \not\models \tau_0)\;\;(p \xrightarrow[E]{E_p,\mathtt{tt}} p')}{STAR_S(p) \xrightarrow[E]{E_p \cup \upsilon_j \cup \{\xi_j\},\mathtt{tt}} 0} \qquad \textit{(Star 3)}$$

$$\frac{(\forall k:1..n)\;\;(\rho_k \in \{\gtrdot, \dot{\gg}\})\;\;(E \not\models \tau_k)\;\;(E \models \tau_0)}{STAR_S(p) \xrightarrow[E]{\emptyset,\mathtt{ff}} STAR_S(p)} \qquad \textit{(Star 4)}$$

$$\frac{(\forall k:1..n)\ (\rho_k \in \{\gtrdot, \gg\})\ (E \not\models \tau_k)\ (E \not\models \tau_0)\ (p \xrightarrow[E]{E_p,\mathtt{ff}} p')}{STAR_S(p) \xrightarrow[E]{E_p,\mathtt{ff}} STAR_S(p')} \qquad \textit{(Star 5)}$$

$$\frac{(\forall k:1..n)\ (\rho_k \in \{\gtrdot, \gg\})\ (E \not\models \tau_k)\ (E \not\models \tau_0)\ (p \xrightarrow[E]{E_p,\mathtt{tt}} p')\ (j > n)}{STAR_S(p) \xrightarrow[E]{E_p,\mathtt{ff}} STAR_S(1*)} \qquad \textit{(Star 6)}$$

Comments:

- Leaving the star: Rule *(Star 1)* expresses the weak abortion, rule *(Star 2)* the strong abortion, and rule *(Star 3)* the normal termination. Note that for the first two rules, we have to check for the normal termination of the body of the star that may be of higher priority. Suspension cannot prevent from aborting.
- Suspending the activity of the star: Rule *(Star 4)* captures the suspension of the activity of the star, in the absence of abortions.
- Normal activity of the star: Rule *(Star 5)* applies in the absence of any preemption on the star; only its body changes. The last rule is used if the normal termination of the star is not provided ($j > n$). In this case, when the body of the star terminates, Rule *(Star 6)* rewrites the body of the star into the never ending process ($1*$). From now on, we have a dark star waiting for its possible destruction.

5.2.3 Process associated with a star

The following process behaves like a star:

$$
\begin{array}{l}
STAR_S(\mathtt{Body}_S)\ \equiv \\
\quad (\\
\quad\quad (< g_1 > \nearrow \\
\quad\quad\quad (< g_2 > \nearrow \\
\quad\quad\quad\quad (\\
\quad\quad\quad\quad\quad\quad \vdots \\
\quad\quad\quad\quad\quad (< g_n > \nearrow (1*) \rhd 0, < e_n > \\
\quad\quad\quad\quad\quad) \rhd 0, < e_{n-1} > \\
\quad\quad\quad\quad) \rhd 0, < e_{n-2} > \\
\quad\quad\quad\quad \vdots \\
\quad\quad\quad) \rhd 0, < e_1 > \\
\quad\quad) \\
\quad\quad | \\
\quad\quad \left(\bigvee_{k=1}^{k=n} \xi_k\right) \nearrow \left(\alpha; (\iota \supset (< g_0 > \supset \mathtt{Body}_S))\right) \rhd \omega, 0 \\
\quad) \setminus \alpha \setminus \iota \setminus \omega
\end{array}
$$

provided $\xi_k (k = 1..n), \alpha, \iota, \omega \notin \left(\mathcal{X}_p \cup \bigcup_{k=1}^{k=n} (\|\tau_k\| \cup \|\upsilon_k\|)\right)$

$\mathcal{X}_{p\ :\ \tau_1\ \rho_1\ \upsilon_1, \cdots, \tau_n\ \rho_n\ \upsilon_n} = \mathcal{X}_p \cup \bigcup_{k=1}^{k=n} (\|\tau_k\| \cup \|\upsilon_k\|)$

$\mathcal{O}_{p\ :\ \tau_1\ \rho_1\ \upsilon_1, \cdots, \tau_n\ \rho_n\ \upsilon_n} = \mathcal{O}_p \cup \bigcup_{k=1}^{k=n} \|\upsilon_k\|$

$\mathcal{I}_{p\ :\ \tau_1\ \rho_1\ \upsilon_1, \cdots, \tau_n\ \rho_n\ \upsilon_n} = \mathcal{X}_{p\ :\ \tau_1\ \rho_1\ \upsilon_1, \cdots, \tau_n\ \rho_n\ \upsilon_n} - \mathcal{O}_{p\ :\ \tau_1\ \rho_1\ \upsilon_1, \cdots, \tau_n\ \rho_n\ \upsilon_n}$

$\mathcal{L}_{p\ :\ \tau_1\ \rho_1\ \upsilon_1, \cdots, \tau_n\ \rho_n\ \upsilon_n} = \mathcal{L}_p$

where

ρ_0	$< g_0 >$	
$\dot{\supset}$	τ_0	
$\supset$	$\alpha' \bullet \tau_0$	
ρ_k	$< g_k >$	$< e_k >$
$\dot{>}$	$\tau'_1 \bullet \cdots \bullet \tau'_{k-1} \bullet \tau_k$	$\upsilon_k ; \xi_k$
$>$	$\alpha' \bullet \tau'_1 \bullet \cdots \bullet \tau'_{k-1} \bullet \tau_k$	$\upsilon_k ; \xi_k$
$\dot{\gg}$	$\tau'_1 \bullet \cdots \bullet \tau'_{k-1} \bullet \tau_k$	$\iota ; \upsilon_k ; \xi_k$
$\gg$	$\alpha' \bullet \tau'_1 \bullet \cdots \bullet \tau'_{k-1} \bullet \tau_k$	$\iota ; \upsilon_k ; \xi_k$
$\triangleright$	$\tau'_1 \bullet \cdots \bullet \tau'_{k-1} \bullet \omega$	$\upsilon_k ; \xi_k$

Comments: When S is destroyed (normal termination or abortion) an exit signal (ξ_k) is emitted to let know which kind of termination has occurred. The priority is ensured by the conjunction of the negations of the higher priority triggering compounds.

Hint: In order to avoid causality cycles, strong abortions should preceed weak abortions and normal terminations. This is not mandatory.

Remark: A star must be either left by a beam or preempted by a higher level abortion. When the normal termination of p is not captured by a beam, the star *must* not be destroyed on the completion of p. The above process meets this requirement: the second branch of the parallel constructor emits ω and terminates, but the first branch does not. The star becomes a dark star. The star will be destroyed as soon as a trigger $< g_k >$ is satisfied.

5.3 Constellation

5.3.1 Star ordering

Because of immediate abortions or normal terminations, a star may exit at its instant of activation. Thus, several stars in a constellation can be activated and de-activated at the same instant. Of course, instantaneous loops are not allowed. A solution could be to forbid 0-duration activations. This would discard one of the most interesting feature of the synchronous programming. So, we allow transient activations, but the price to pay is an ordering of stars, at compile-time.

Given a constellation C,

1. From Γ_C and Γ^i_C deduce a *dependence graph* G_C such that
   - If there are n stars in C, G_C has $n+1$ vertices: $\{v_0, v_1, \cdots, v_n\}$.
   - There is a bijection $\{v_1, \cdots, v_n\} \leftrightarrow C^\circ$. Without loss of generality, assume that $v_k \leftrightarrow s_k$.
   - For $k \neq 0$, there is an arc from v_0 to v_k if and only if s_k is an initial star.
   - For $i \neq 0$ and $j \neq 0$, there is an arc from v_i to v_j if and only if the control can pass from s_i to s_j
2. In G_C delete all arcs corresponding to delayed abortions. Recall that normal terminations can be immediate: don't discard them!. Let G'_C be the new graph: It represents the skeleton of the possible instantaneous reactions.
3. If G'_C is acyclic, then perform a *topological sorting.* v_0 is always the first node, don't name it, it stands for an implicit initialization node. Assign to each vertex v_k, $k \neq 0$ a name $l_k \in [A..Z]^+$ such that the lexicographical order respects the topological sorting. Star s_k is given the internal name $\texttt{i-name}_C \# \# l_k$, where $\#\#$ stands for catenation.
4. If there is a cycle, then,
   - Either reject the SyncCharts for *presumption of instantaneous loop*,

- Or consider further details in order to erase an arc involved in a cycle. For example you can deduce from the inspection of the body of a star that its normal termination is never at its activation time.

See example of application in Fig. 10. Vertex v_0 is a solid circle. Since the translation process heavily depends on the order of stars in a constellation, be sure not to make unjustified deletions.

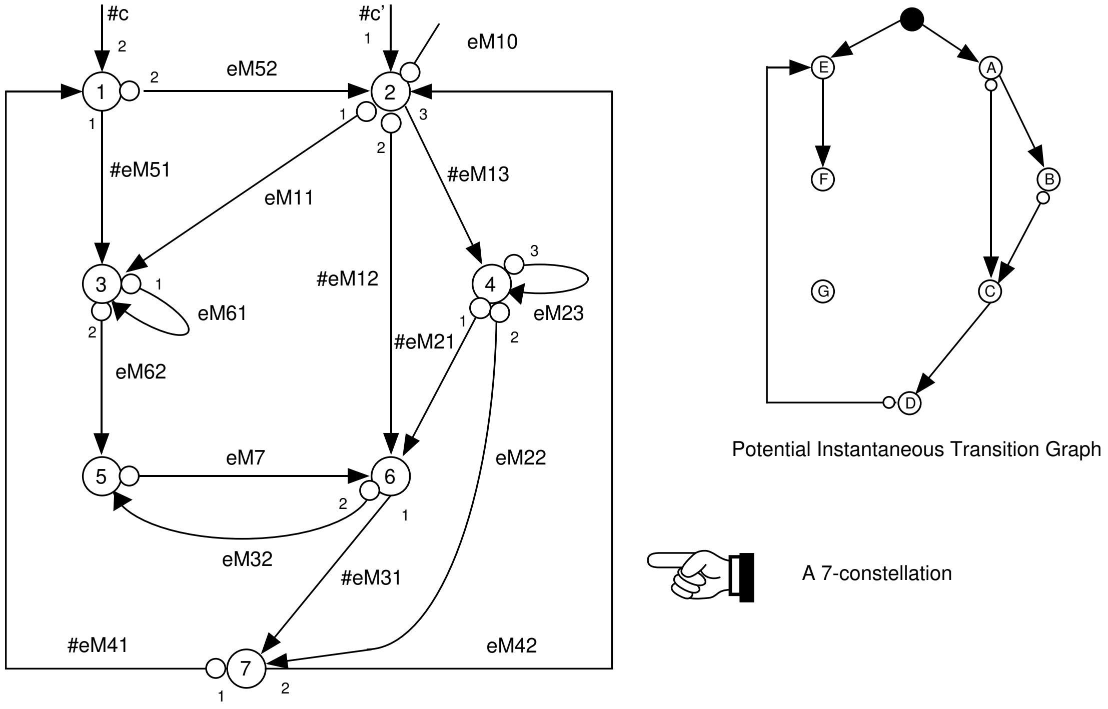

Figure 10: Ordering stars in a constellation

5.3.2 Expected behavior of a constellation

Given a constellation C,

- Let n be the number of stars in C (cardinality of C°).
- Let l be the number of initial stars in C ($\texttt{in-degree}_C$). Recall that $l \geq 1$.
- Let $\Gamma^i_C(k) = < I_k \ , \ s_k > \quad (k = 1..l)$ where $I_k \in \texttt{ARCS}^{init}_{\mathbb{S}}$ and $s_k \in C^{\circ}$. Note that I_{k_1} has priority over I_{k_2} if $k_1 < k_2$.
- Let $\Gamma_C(s,k) = \ll s, k \gg$ where $s \in C^{\circ}, k \in [1..\nu_s]$ where $\nu_s = \texttt{out-degree}_s$, and $\ll s, k \gg \in C^{\circ}$.

We impose a new semantic constraint on a constellation:

Constraint

- $\big(\forall E \subseteq (\mathbb{S} - \perp)\big) \quad [\![\bigvee_{k=1}^{k=l} (I_k)_{trig}]\!]_E = \texttt{tt}$

This constraint means that at least one initial arc has a trigger evaluated to $\texttt{tt}$, so that there is always at least one eligible initial star.

Hint: Use a void trigger for the default initial arc. The semantic constraint on the initial arc labeling will be always fulfilled.

The process $CONST_C(k,p)$ associated with C has two parameters: the index of the active star and the current body of this star. Initially, we have $CONST(0,0)$. After an initialization phase, $CONST_C$ enters a never ending loop.

Initialization phase: The first active star in the constellation C is the elegible star with the highest priority. Let k be this star ($1 \leq k \leq l$). The expected behavior is:

$$\frac{(\exists k : 1..l)\ \big(E \models (I_k)_{trig}\big) \wedge \big((\forall j < k)\ E \not\models (I_j)_{trig}\big)}{CONST_C(0,0) \xrightarrow[E]{\{\gamma_k\},\,\mathtt{tt}} CONST_C(k, \mathtt{Body}_k)} \quad \textit{(Const 1)}$$

Comments: $\gamma_1, \cdots, \gamma_n$ are signals local to C. γ_k is interpreted as `goto` star k.

Comments:

- The initialization emits one and only one signal: γ_k such that $k \in [1..l]$ is the least integer such that $E \models (I_k)_{trig}$ where E is the current event.
- The least priority initial arc ($\Gamma^i_C(l)$) is used as the default initial arc: If for all $k < l$, $E \not\models (I_k)_{trig}$, then $(I_l)_{trig}$ *must* be satisfied. The constraint imposed on the initial arc labeling ensures that, in this case, $E \models (I_l)_{trig}$ and there is no need for evaluating it, at run-time.

A process for the initialization part :

$$
\begin{array}{l}
CONST_C(0,0) ::= \\
\quad (I_1)_{trig}?\big(\gamma_1; (I_1)_{eff}\big), \big(\\
\quad\quad (I_2)_{trig}?\big(\gamma_2; (I_2)_{eff}\big), \big(\\
\quad\quad\quad\quad \ddots \\
\quad\quad\quad\quad (I_{l-1})_{trig}?\big(\gamma_{l-1}; (I_{l-1})_{eff}\big), \big(\\
\quad\quad\quad\quad\quad \gamma_l; (I_l)_{eff} \\
\quad\quad\quad\quad\quad \big) \\
\quad\quad\quad\quad \vdots \\
\quad\quad\quad \big) \\
\quad\quad \big); \\
\quad LOOP_C
\end{array}
$$

The Loop part: The idea is to optionally execute stars ordered according to the topological sorting in sequence and repeatedly. The `goto`'s signals are used to decide whether a star has to be activated or not.

Beside the activity of a star, we have to emit the ω_C signal if and only if the active star is a final star that will not be aborted at the current instant.

Expected behavior of the loop part:

$$\frac{(\gamma_k \in E)\ \ \big(STAR_k(\mathtt{Body}_k) \xrightarrow[E]{E_k,\,\mathtt{tt}} 0\big)\ \ (\xi_i \in E_k)\ \ \big(\Gamma_C(k,i) = j\big)}{CONST_C(k, \mathtt{Body}_k) \xrightarrow[E]{E_k\ [\gamma_j/\xi_i],\,\mathtt{tt}} CONST_C(j, \mathtt{Body}_j)} \quad \textit{(Const 2)}$$

$$\frac{(\gamma_k \in E) \quad \big(STAR_k(\mathtt{Body}_k) \xrightarrow[E]{E_k,\,\mathtt{ff}} STAR_k(p)\big) \quad \begin{pmatrix} (k \in \Gamma_C^f)\ (F = \{\omega_C\}) \\ \\ (k \notin \Gamma_C^f)\ (F = \emptyset) \end{pmatrix}}{CONST_C(k, \mathtt{Body}_k) \xrightarrow[E]{F \cup E_k\ [\gamma_j/\xi_i],\,\mathtt{ff}} CONST_C^{\eta}(k, p)} \quad \textit{(Const 3)}$$

$$\frac{\big(STAR_k(p) \xrightarrow[E]{E_k,\,\mathtt{ff}} STAR_k(p')\big) \quad \begin{pmatrix} (k \in \Gamma_C^f)\ (F = \{\omega_C\}) \\ \\ (k \notin \Gamma_C^f)\ (F = \emptyset) \end{pmatrix}}{CONST_C^{\eta}(k, p) \xrightarrow[E]{F \cup E_k,\,\mathtt{ff}} CONST_C^{\eta}(k, p')} \quad \textit{(}\eta\textit{-Const 1)}$$

$$\frac{\big(STAR_k(p) \xrightarrow[E]{E_k,\,\mathtt{tt}} 0\big) \quad (\xi_i \in E_k) \quad \big(\Gamma_C(k,i) = j\big)}{CONST_C^{\eta}(k, p) \xrightarrow[E]{E_k\ [\gamma_j/\xi_i],\,\mathtt{tt}} CONST_C^{\eta}(j, \mathtt{Body}_j)} \quad \textit{(}\eta\textit{-Const 2)}$$

Comments:

- Rule *(Const 2)* captures the instantaneous execution of the star k that emits one and only one exit signal (say ξ_i). This signal is renamed in a goto signal (γ_j) according to the connections in the constellation C $(\Gamma_C(k,i) = j)$. The control is immediately passed to the star j.
- If the star k does not immediately terminate, then rule *(Const 3)* applies. $CONST_C(k, \mathtt{Body}_k)$ rewrites into $CONST_C^{\eta}(k,p)$, the superscript η denotes that it is no longer the first instant of $CONST_C(k, \mathtt{Body}_k)$. The parameter p keeps trace of the changes in the star k. Moreover, if k is a final star $(k \in \Gamma_C^f)$, then signal ω_C is emitted.
- Rule *(η-Const 1)* stands for the normal activity of star k. Here again, if k is a final star then signal ω_C is emitted.
- Finally, rule *(η-Const 2)* corresponds to the abortion of star k not in its first instant. One and only one exit signal ξ_i is emitted, seen as a goto signal γ_j. The star j is immediately activated. Note that in the case of a final star k, ω_C is no longer emitted since the star k is aborted.

A process for the loop part: We need a loop in which we can enter at any point, then proceed on in sequence or restart from the beginning. A sequence of conditionals, weakly aborted by a local signal within a loop will do:

$$\Big(\Big(\big((\gamma_1 ? CSTAR_1, 0); \cdots; (\gamma_n ? CSTAR_n, 0)\big) : \lambda > 0\Big) \setminus \lambda\Big)*$$

where λ is emitted when a star terminates at an instant which is not its activation instant; $CSTAR_k$ is a process derived from $STAR_k$ augmented with the management of the ω_C signal.

A possible process is:

$$
\begin{array}{l}
LOOP_C ::= \\
\quad (\\
\quad\quad (\\
\quad\quad\quad (\\
\quad\quad\quad\quad \Big(\gamma_1?\big((\alpha; CSTAR_1; (\alpha?0,\lambda)) \setminus \alpha\big), 0\Big); \\
\quad\quad\quad\quad\quad\quad\quad \vdots \\
\quad\quad\quad\quad \Big(\gamma_n?\big((\alpha; CSTAR_n; (\alpha?0,\lambda)) \setminus \alpha\big), 0\Big); \\
\quad\quad) \setminus \lambda \\
\quad)*
\end{array}
$$

$$
\begin{array}{ll}
\text{where } CSTAR_k & \\
\quad ::= (& \\
\quad\quad\quad \Big(((\omega_C; 1)*) : \omega \gg 0\Big) & \\
\quad\quad\quad | & (\text{if } sk \in \Gamma_C^f) \\
\quad\quad\quad \Big(STAR_k(\texttt{Body}_k)\ [\gamma_{\ll k,1 \gg}/\xi_1, \cdots, \gamma_{\ll k,\nu_k \gg}/\xi_{\nu_k}]; \omega\Big) & \\
\quad\quad) \setminus \omega & \\
\quad ::= STAR_k(\texttt{Body}_k)\ [\gamma_{\ll k,1 \gg}/\xi_1, \cdots, \gamma_{\ll k,\nu_k \gg}/\xi_{\nu_k}] & (\text{otherwise})
\end{array}
$$

5.4 Macrostate

5.4.1 Presentation

Recall that a SyncCharts is a macrostate. A macrostate is composed of orthogonal constellations. A macrostate delimits the scope of user's defined local signals. A macrostate has also to manage the synchronized termination of its constellation components.

5.4.2 Expected behavior of a macrostate

Let M be a n-macrostate. Let $c1, c2, \ldots, cn$ be its constellation components. Let $L \subseteq \mathcal{L}_M$ a subset of the set of the local variables of M. Let $\omega_M = \widetilde{\omega_{c1}} \bullet \cdots \bullet \widetilde{\omega_{cn}}$, where $\widetilde{\omega_{ck}}$ is egal to either ω_{ck} or ω'_{ck}. We introduce auxiliary notations:

$$
\begin{array}{rcl}
\omega_M^+ & = & \{\omega_{ck} \mid \omega_{ck} \text{ appears in its direct form in } \omega_M\} \\
\omega_M^- & = & \{\omega_{ck} \mid \omega_{ck} \text{ appears in its complementary form in } \omega_M\} \\
\tilde{E} & = & (E \cup \omega_M^+) - \omega_M^- \\
\bar{L} & = & \mathcal{L}_M - L
\end{array}
$$

The process associated with the macrostate M has $2 * n$ parameters that characterize its state (i.e., the state of each constellation component): $MACRO_M\big((k_1,p_1), \cdots, (k_n,p_n)\big)$. The expected behavior of a macro is defined by the following rules:

$$
\frac{(L \subseteq \bigcup_{j=1}^{j=n} E_j)\ (\omega_M \neq \omega_{c1} \bullet \cdots \bullet \omega_{cn})\ \left(CONST_{cj}(k_j,p_j) \xrightarrow[(\tilde{E} \cup L) - \bar{L}]{E_j,\, b_j} CONST_{ck}(k'_j,p'_j)\right)_{j=1}^{j=n}}{MACRO_M\big((k_1,p_1), \cdots, (k_n,p_n)\big) \xrightarrow[E]{\left(\bigcup_{k=1}^{k=n} E_k\right) - L - \omega_M^+,\, \texttt{ff}} MACRO_M\big((k'_1,p'_1), \cdots, (k'_n,p'_n)\big)} \quad \textit{(Macro 1)}
$$

$$\frac{(L \subseteq \bigcup_{j=1}^{j=n} E_j)\ (\omega_M = \omega_{c1} \bullet \cdots \bullet \omega_{cn})\ \left(CONST_{cj}(k_j, p_j) \xrightarrow[(\tilde{E} \cup L) - \bar{L}]{E_j,\, b_j} CONST_{ck}(k'_j, p'_j) \right)_{j=1}^{j=n}}{MACRO_M\big((k_1,p_1),\cdots,(k_n,p_n)\big) \xrightarrow[E]{\left(\bigcup_{k=1}^{k=n} E_k\right) - L - \omega_M^{+},\, \mathtt{tt}} 0} \quad \textit{(Macro 2)}$$

Comments:

- Rule *(Macro 1)* applies when the macrostate does not terminate ($\omega_M \neq \omega_{c1} \bullet \cdots \bullet \omega_{cn}$). Signals ω_c's and signals in $\mathcal{L}_M$ are treated as local signals.
- Rule *(Macro 2)* is the termination of the macro M. Note that with respect to the immediate higher level (the star whose body is M), this termination is the normal termination.

5.4.3 Process associated with a macrostate

$$
\begin{aligned}
&MACRO_M ::= \\
&\quad (\\
&\qquad \Big(\big(CONST_{c1} \mid \cdots \mid CONST_{cn}\big) : \big(\omega_{c1} \bullet \cdots \bullet \omega_{cn}\big) \gg 0, \triangleright\ (1*)\Big) \setminus \{\omega_{c1}, \cdots, \omega_{cn}\} \\
&\quad) \setminus \mathtt{Local}_M
\end{aligned}
$$

Comments: This process shows that processes $CONST_c$'s execute in parallel. Their execution is weakly aborted when a final state is reached in each constellation ($\omega_{c1} \bullet \cdots \bullet \omega_{cn}$). If the parallel terminates (normal termination of all the branches), but not in the final state, the macro enters a never ending process ($1*$). Only a higher level abortion could exit this state. Signals ω_c's and signals in $\mathcal{L}_M$ are treated as local by the restriction constructor.

6 Translation into Esterel

The derived imperative operators are the same as those introduced by G. Berry; their translations into Esterel's statements are well-known. So is the (immediate) suspend operator. Only the (immediate) weak preemption is different.

6.1 Weak abortion operator

We propose a translation based on traps[9].

$p : s > q \iff$

```
trap P1 in
     trap P2 in
         run p; exit P1
         ||
         await immediate s; exit P2
     end trap;
     run q
end trap
```

In order to prove the equivalence between our weak preemption operator and the above Esterel program, we establish that both rewrite in the same way. The classical operators and rules are recalled in annex A.

Note that in our processes we have neither trap nor exit, thus "$\uparrow p \equiv p \equiv\uparrow p \equiv \{\uparrow p\}$" and the integer termination code k never exceeds 1 ($k = 0 \iff b = \mathtt{tt}$ and $k = 1 \iff b = \mathtt{ff}$). Recall that $\downarrow 1 = 1; \uparrow 0 = 0; \uparrow 1 = 1$.

According to these remarks, the Esterel statement is associated with the Berry's process: $WA ::= \{\{(p;3) \mid (s?2,1)*\}\,;q\}$.

[9] I'd rather translate $s \nearrow p \triangleright n, q$, easy to do but not written yet ...

- Prove that *(wa1)* $\Longleftrightarrow$ *(WA1)* where

$$\frac{p \xrightarrow[E]{E_p\,,\,0} p'}{WA \xrightarrow[E]{E_p\,,\,0} 0} \quad \textit{(WA1)}$$

proof :

$$\textit{(trap1)}\ \dfrac{\textit{(seq1)}\ \dfrac{\textit{(trap2)}\ \dfrac{\textit{(parallel)}\ \dfrac{\textit{(seq2)}\ \dfrac{p \xrightarrow[E]{E_p\,,\,0} p' \qquad \textit{(exit)}\ \ 3 \xrightarrow[E]{\emptyset\,,\,3} 0}{(p;3) \xrightarrow[E]{E_p\,,\,3} 0} \qquad \textit{(loop)}\ \dfrac{s?2,1 \xrightarrow[E]{\emptyset\,,\,k} \cdots \qquad k=1,2}{(s?2,1)* \xrightarrow[E]{\emptyset\,,\,k} \cdots \qquad k=1,2}}{(p;3) \mid (s?2,1)* \xrightarrow[E]{E_p\,,\,3} 0 \mid \cdots}}{\{(p;3) \mid (s?2,1)*\} \xrightarrow[E]{E_p\,,\,2} \{0 \mid \cdots\}}}{\{(p;3) \mid (s?2,1)*\};q \xrightarrow[E]{E_p\,,\,2} \{0 \mid \cdots\};q}}{WA \xrightarrow[E]{E_p\,,\,0} 0}$$

- Prove that *(wa2)* $\Longleftrightarrow$ *(WA2)* where

$$\frac{s \in E \qquad p \xrightarrow[E]{E_p\,,\,1} p' \qquad q \xrightarrow[E]{E_q\,,\,k} q'}{WA \xrightarrow[E]{E_p \cup E_q\,,\,k} q'} \quad \textit{(WA2)}$$

proof :

$$\textit{(seq2)}\ \dfrac{\textit{(trap1)}\ \dfrac{\textit{(parallel)}\ \dfrac{\textit{(seq1)}\ \dfrac{p \xrightarrow[E]{E_p\,,\,1} p'}{(p;3) \xrightarrow[E]{E_p\,,\,1} (p';3)} \qquad \textit{(loop)}\ \dfrac{\textit{(cond1)}\ \dfrac{s \in E \qquad 2 \xrightarrow[E]{\emptyset\,,\,2} 0}{(s?2,1)* \xrightarrow[E]{\emptyset\,,\,2} 0}}{(s?2,1)* \xrightarrow[E]{\emptyset\,,\,2} (s?2,1)*}}{(p;3) \mid (s?2,1)* \xrightarrow[E]{E_p\,,\,2} (p';3) \mid (s?2,1)*}}{\{(p;3) \mid (s?2,1)*\} \xrightarrow[E]{E_p\,,\,0} 0} \qquad q \xrightarrow[E]{E_q\,,\,k} q'}{\{(p;3) \mid (s?2,1)*\};q \xrightarrow[E]{E_p \cup E_q\,,\,k} q' \qquad k=0 \ or\ k=1}$$

There are two cases:

– $k = 0$: by *trap1* the last transition is rewritten as $WA \xrightarrow[E]{E_p \cup E_q\,,\,0} 0$

– $k = 1$: by *trap2* the last transition is rewritten as $WA \xrightarrow[E]{E_p \cup E_q\,,\,1} \{q'\}$. Now, $\{q'\}$ behaves like q', hence $WA \xrightarrow[E]{E_p \cup E_q\,,\,1} q'$

- Prove that *(wa3)* $\Longleftrightarrow$ *(WA3)* where

$$\frac{s \notin E \qquad p \xrightarrow[E]{E_p\,,\,1} p'}{WA \xrightarrow[E]{E_p\,,\,\mathtt{ff}} WA[p \leftarrow p']} \quad \textit{(WA3)}$$

proof :

$$\textit{(trap2)}\ \dfrac{\textit{(seq1)}\ \dfrac{\textit{(trap2)}\ \dfrac{\textit{(parallel)}\ \dfrac{\textit{(seq1)}\ \dfrac{p \xrightarrow[E]{E_p\,,\,1} p'}{(p;3) \xrightarrow[E]{E_p\,,\,1} (p';3)} \qquad \textit{(loop)}\ \dfrac{\textit{(cond2)}\ \dfrac{s \in E \qquad 1 \xrightarrow[E]{\emptyset\,,\,1} 0}{(s?2,1)* \xrightarrow[E]{\emptyset\,,\,1} 0}}{(s?2,1)* \xrightarrow[E]{\emptyset\,,\,1} (s?2,1)*}}{(p;3) \mid (s?2,1)* \xrightarrow[E]{E_p\,,\,1} (p';3) \mid (s?2,1)*}}{\{(p;3) \mid (s?2,1)*\} \xrightarrow[E]{E_p\,,\,1} \{(p';3) \mid (s?2,1)*\}}}{\{(p;3) \mid (s?2,1)*\}; q \xrightarrow[E]{E_p \cup E_q\,,\,1} \{(p';3) \mid (s?2,1)*\}; q}}{WA \xrightarrow[E]{E_p\,,\,1} WA[p \leftarrow p']}$$

Now, we propose systematic translations of SyncCharts'components into Esterel's programs.

6.2 Star module

Table 5: Star module

```
module STAR_S:
     % let p be the body of S
     % let a = Inhibit_S
     % assume S be a n-star
     input I_S; % see Section.4.2.5
     output O_S; % see Section.4.2.5
     trap T in
         signal α,ι,ω in
```

continued on next page

continued from previous page

```
          emit α;
          suspend
              suspend
                  run MACRO_p;
                  emit ω
              when < g0 >
          when immediate ι
          ||
          await
              case < g1 > do < e1 >
                                 ⋮
              case < gn > do < en >
          end await;
          exit T
      end signal
   end trap
end module

% where < g0 > = immediate [a_trig] (if a_mod = immediate)
%                          [a_trig] (otherwise)
```

Comments: In this translation priority is ensured by the `await ...case ...` statement: cases are written in a decreasing order of priority. The immediate or delayed modifier could have been directly expressed by the keyword `immediate` of Esterel, so that the local signal α would have been useless. Flattening of triggers has been prefered, instead. Non terminal terms are given below:

$< op_k >$	$< g_k >$	$< e_k >$
$\gtrdot$	τ_k	$\upsilon_k\,;\xi_k$
$>$	$\alpha' \bullet \tau_k$	$\upsilon_k\,;\xi_k$
$\ggg$	τ_k	$\iota;\upsilon_k\,;\xi_k$
$\gg$	$\alpha' \bullet \tau_k$	$\iota;\upsilon_k\,;\xi_k$
$\triangleright$	ω	$\upsilon_k\,;\xi_k$

0-stars have a simpler behavior (no way to leave), this makes it easier to translate.

Table 6: 0-star module

```
module STAR_S:
     % assume S be a 0-star
     % let p be the body of S
     % let a = Inhibit_S
     input I_S; % see Section.4.2.5
     output O_S; % see Section.4.2.5
         suspend
             run MACRO_p;
             halt
         when < g0 >
end module
```

continued on next page

continued from previous page

```
% where < g_0 > = immediate [a_trig] (if a_mod = immediate)
%                           [a_trig] (otherwise)
```

6.3 Macrostate module

Table 7: Macrostate module

```
module MACRO_M:
     input I_M; % see Section.4.2.3
     output O_M; % see Section.4.2.3
     signal Local_M ∪ {ω_c | c ∈ M°} in
         trap T in
              ||      run CONSTELLATION_c
            c∈M°
             || % wait for parallel termination
             await immediate [ ∧  ω_c]; exit T
                              c∈M°
         end trap
     end signal
end module
```

6.4 Constellation module

Table 8: constellation module

```
module CONSTELLATION_C:
     % let in-degree_C = l (l initial arcs)
     % let cardinality of C° = n (n-constellation)
     input I_C; % see Section.4.2.4
     output O_C; % see Section.4.2.4
     signal {γ_c | s ∈ C°} in
         % init: let (Γ^i_C[k])_trig = t_k
         % and (Γ^i_C[k])_eff = z_k
         present [t_1] then emit γ_1; emit z_1 else
             present [t_2] then emit γ_2; emit z_2 else
                 ⋱
                         emit γ_l; emit z_l % catchall
             end present
         end present;
         % main loop
         loop
             trap L in
                 % assume C° = {s_1, ···, s_n}
                 present γ_1 then
                     signal α in
                         emit α;
                         run STAR_s1[signal γ_.../ξ_..., ···];
                         present α else exit L end present
```

continued on next page

continued from previous page

```
                    end signal
                end present;
                % assume s_2 ∈ Γ_C^f
                present γ_2 then
                    signal α,ω in
                        emit α;
                        [
                            do sustain ω_C watching immediate ω
                            ||
                            run STAR_s_2[signal γ.../ξ..., ···]; emit ω
                        ];
                        present α else exit L end present
                    end signal
                end present;
                            ⋮
                present γ_n then
                    signal α in
                        emit α;
                        run STAR_s_n[signal γ.../ξ..., ···];
                        present α else exit L end present
                    end signal
                end present;
                halt % horrible trick to fake the compiler
            end trap
        end loop
    end signal
end module
```

Comments: Signals γ_k stand for `goto` star$_k$. When entering the constellation, we emit γ_k according to the satisfaction of the associated triggering events of the initial arcs. At least one γ is emitted (default initial star). We then enter the main loop. The idea is to test the presence of the γ_k in the topological order. If present, the corresponding star is executed.

Suppose we execute star$_k$. If it terminates instantly, then α is still present and the control passes to the next test (management of transient stars). Otherwise, the `trap L` is exited and we test again from the very beginning (management of transitions between steady states).

Terminal stars need a special handling: while the star is running, the signal ω_C is sustained.

Note that the `halt` statement at the end of the body of the loop should be never executed. We introduced it for technical reason (the compiler cannot understand that the body of the loop is never 0-duration long).

Since a star emits an exit signal (ξ) when leaving, this signal has to be renamed as a γ signal according to the structure of the constellation. This renaming is done by the `run` statement.

A A Summary of the Esterel Calculus

(Excerpt from G. Berry's paper on *Preemption in Concurrent Systems*.

A.1 Set of primitives

n	for $n \geq 0$	(termination code)
s	for $s \notin \mathcal{I}$	(emission)
$p; q$		(sequence)
$p*$		(loop)
$s?p, q$		(conditional)
$p \mid q$		(parallel)
$p \setminus s$	for $s \in \mathcal{L}$	(restriction)
$\{p\}$		(trap)
$\uparrow p$		(shift)
$s \supset p$		(immediate suspension)

A.2 Rules

$$n \xrightarrow[E]{\emptyset,\, n} 0 \qquad \textit{(termination code)}$$

$$s \xrightarrow[E]{\{s\},\, 0} 0 \qquad \textit{(emission)}$$

$$\frac{p \xrightarrow[E]{E_p,\, k} p' \qquad k \neq 0}{p;\ q \xrightarrow[E]{E_p,\, k} p';\ q} \qquad \textit{(seq1)}$$

$$\frac{p \xrightarrow[E]{E_p,\, 0} p' \qquad q \xrightarrow[E]{E_q,\, k} q'}{p;\ q \xrightarrow[E]{E_p \cup E_q,\, k} q'} \qquad \textit{(seq2)}$$

$$\frac{p \xrightarrow[E]{E_p,\, k} p' \qquad k \neq 0}{p* \xrightarrow[E]{E_p,\, k} p';\ (p*)} \qquad \textit{(loop)}$$

$$\frac{p \xrightarrow[E]{E_p,\, k_p} p' \qquad q \xrightarrow[E]{E_q,\, k_q} q'}{p \mid q \xrightarrow[E]{E_p \cup E_q,\, max(k_p, k_q)} p' \mid q'} \qquad \textit{(parallel)}$$

$$\frac{s \in E \qquad p \xrightarrow[E]{E_p,\, k_p} p'}{s?p, q \xrightarrow[E]{E_p,\, k_p} p'} \qquad \textit{(cond1)}$$

$$\frac{s \notin E \qquad q \xrightarrow[E]{E_q,\, k_q} q'}{s?p, q \xrightarrow[E]{E_q,\, k_q} q'} \qquad \textit{(cond2)}$$

$$\frac{p \xrightarrow[E \cup \{s\}]{E_p,\, k} p' \qquad s \in E_p}{p \setminus s \xrightarrow[E]{E_p - \{s\},\, k} p' \setminus s} \qquad \textit{(restr1)}$$

$$\frac{p \xrightarrow[E-\{s\}]{E_p,\,k} p' \qquad s \notin E_p}{p \setminus s \xrightarrow[E]{E_p,\,k} p' \setminus s} \qquad \textit{(restr2)}$$

$$\frac{s \in E}{s \supset p \xrightarrow[E]{\emptyset,\,1} s \supset p} \qquad \textit{(imm. susp1)}$$

$$\frac{s \notin E \qquad p \xrightarrow[E]{E_p,\,0} p'}{s \supset p \xrightarrow[E]{E_p,\,0} 0} \qquad \textit{(imm. susp2)}$$

$$\frac{s \notin E \qquad p \xrightarrow[E]{E_p,\,k} p' \qquad k \neq 0}{s \supset p \xrightarrow[E]{E_p,\,k} s \supset p'} \qquad \textit{(imm.3)}$$

$$\frac{p \xrightarrow[E]{E_p,\,k} p' \qquad k = 0 \; or \; k = 2}{\{p\} \xrightarrow[E]{E_p,\,0} 0} \qquad \textit{(trap1)}$$

$$\frac{p \xrightarrow[E]{E_p,\,k} p' \qquad k = 1 \; or \; k > 2}{\{p\} \xrightarrow[E]{E_p,\,\downarrow k} \{p'\}} \qquad \textit{(trap2)}$$

$$\frac{p \xrightarrow[E]{E_p,\,k} p'}{\uparrow p \xrightarrow[E]{E_p,\,\uparrow k} \uparrow p'} \qquad \textit{(shift)}$$

$$\downarrow k = \begin{cases} 1 \; if \; k = 1 \\ k-1 \; if \; k > 2 \end{cases}$$

$$\uparrow k = \begin{cases} 0 \; if \; k = 0 \\ 1 \; if \; k = 1 \\ k+1 \; if \; k \geq 2 \end{cases}$$

B Esterel translation of the Watchdog SyncCharts

This program has to be compiled with the -oldcausality option. The compiler detects static instantaneous loops which are not effective.

`Table 9: Translation of Watchdog`

```
% Charles ANDRE
% february 21, 1996
% compilation of the SyncCharts :  Watchdog

module A0B0A0A:
     input c2;
     output nb2,xi1;
     trap T in
         signal alpha, iota, omega in
             emit alpha;
             suspend
                 % suspend
                     sustain nb2;% run MACRO_p
                     emit omega
                 % when <g0>
             when immediate iota
             ||
             await
                 case c2 do emit iota; emit xi1
             end await;
             exit T
         end signal
     end trap
end module

module A0B0A0B:
     input c2;
     output b2,xi1;
     trap T in
         signal alpha, iota, omega in
             emit alpha;
             suspend
                 % suspend
                     sustain b2;% run MACRO_p
                     emit omega
                 % when <g0>
             when immediate iota
             ||
             await
                 case c2 do emit iota; emit xi1
             end await;
             exit T
         end signal
     end trap
end module

module A0B0A0:
     input c2;
     output nb2,b2,omega0;
```

continued on next page

continued from previous page

```
     signal gamma1,gamma2 in
         % init:
         emit gamma1;
         % main loop
         loop
             trap L in
                 present gamma1 then
                     signal alpha in
                         emit alpha;
                         run A0B0A0A [signal gamma2/xi1];
                         present alpha else exit L end present
                     end signal
                 end present;
                 % A0B0A0B is final
                 present gamma2 then
                     signal alpha, omega in
                         emit alpha;
                         [
                             do sustain omega0 watching immediate omega
                             ||
                             run A0B0A0B[signal gamma1/xi1]; emit omega
                         ];
                         present alpha else exit L end present
                     end signal
                 end present;
                 halt % horrible trick to fake the compiler
             end trap
         end loop
     end signal
end module

module A0B0A1A:
     input c1;
     output nb1,xi1;
     trap T in
         signal alpha, iota, omega in
             emit alpha;
             suspend
                 % suspend
                     sustain nb1;% run MACRO_p
                     emit omega
                 % when <g0>
             when immediate iota
             ||
             await
                 case c1 do emit iota; emit xi1
             end await;
```

continued on next page

continued from previous page

```
            exit T
        end signal
    end trap
end module

module A0B0A1B:
    input c1;
    output b1,c2,xi1;
    trap T in
        signal alpha, iota, omega in
            emit alpha;
            suspend
                % suspend
                    sustain b1;% run MACRO_p
                    emit omega
                % when <g_0>
            when immediate iota
            ||
            await
                case c1 do emit iota; emit c2; emit xi1
            end await;
            exit T
        end signal
    end trap
end module

module A0B0A1:
    input c1;
    output nb1,b1,c2,omega1;
    signal gamma1,gamma2 in
        % init:
        emit gamma1;
        % main loop
        loop
            trap L in
                present gamma1 then
                    signal alpha in
                        emit alpha;
                        run A0B0A1A [signal gamma2/xi1];
                        present alpha else exit L end present
                    end signal
                end present;
                % A0B0A1B is final
                present gamma2 then
                    signal alpha, omega in
                        emit alpha;
                        [
```

continued on next page

continued from previous page

```
                              do sustain omega1 watching immediate omega
                              ||
                              run A0B0A0B[signal gamma1/xi1]; emit omega
                          ];
                          present alpha else exit L end present
                      end signal
                  end present;
                  halt % horrible trick to fake the compiler
              end trap
          end loop
      end signal
end module

module A0B0A2A:
      input c0;
      output nb0,xi1;
      trap T in
          signal alpha, iota, omega in
              emit alpha;
              suspend
                  % suspend
                      sustain nb0;% run MACRO_p
                      emit omega
                  % when <g0>
              when immediate iota
              ||
              await
                  case c0 do emit iota; emit xi1
              end await;
              exit T
          end signal
      end trap
end module

module A0B0A2B:
      input c0;
      output b0,c1,xi1;
      trap T in
          signal alpha, iota, omega in
              emit alpha;
              suspend
                  % suspend
                      sustain b0;% run MACRO_p
                      emit omega
                  % when <g0>
              when immediate iota
              ||
```

continued on next page

continued from previous page

```
            await
                case c0 do emit iota; emit c1; emit xi1
            end await;
            exit T
        end signal
    end trap
end module

module A0B0A2:
    input c0;
    output nb0,b0,c1,omega2;
    signal gamma1,gamma2 in
        % init:
        emit gamma1;
        % main loop
        loop
            trap L in
                present gamma1 then
                    signal alpha in
                        emit alpha;
                        run A0B0A2A [signal gamma2/xi1];
                        present alpha else exit L end present
                    end signal
                end present;
                % A0B0A2B is final
                present gamma2 then
                    signal alpha, omega in
                        emit alpha;
                        [
                            do sustain omega2 watching immediate omega
                            ||
                            run A0B0A2B[signal gamma1/xi1]; emit omega
                        ];
                        present alpha else exit L end present
                    end signal
                end present;
                halt % horrible trick to fake the compiler
            end trap
        end loop
    end signal
end module

module M_A0B0A:
    input c0;
    output nb2,b2,nb1,b1,nb0,b0;
    signal c1,c2,omega0,omega1,omega2 in
        trap T in
```

continued on next page

continued from previous page

```
            run A0B0A0
            ||
            run A0B0A1
            ||
            run A0B0A2
            ||
            % wait for parallel termination
            await immediate [omega0 and omega1 and omega2]; exit T
        end trap
    end signal
end module

module A0B0A:
    input c0,Inhib;
    output nb2,b2,nb1,b1,nb0,b0,xi1;
    trap T in
        signal alpha, iota, omega in
            emit alpha;
            suspend
                suspend
                    run M_A0B0A;
                    emit omega
                when Inhib
            when immediate iota
            ||
            await
                case immediate omega do emit xi1
            end await;
            exit T
        end signal
    end trap
end module

module A0B0B:
    % input ;
    output Alarm;
    % suspend
        sustain Alarm
    % when
end module

module A0B0:
    input c0,Inhib;
    output nb2,b2,nb1,b1,nb0,b0,Alarm;
    signal gamma1,gamma2 in
        % init:
        emit gamma1;
```

continued on next page

continued from previous page

```
        % main loop
        loop
            trap L in
                present gamma1 then
                    signal alpha in
                        emit alpha;
                        run A0B0A[signal gamma2/xi1];
                        present alpha else exit L end present
                    end signal
                end present;
                present gamma2 then
                    signal alpha in
                        emit alpha;
                        run A0B0B;
                        present alpha else exit L end present
                    end signal
                end present;
                halt % horrible trick to fake the compiler
            end trap
        end loop
    end signal
end module

module A0B:
    input c0,Inhib,reset;
    output nb2,b2,nb1,b1,nb0,b0,Alarm,xi1;
    trap T in
        signal alpha, iota, omega in
            emit alpha;
            suspend
                % suspend
                    run A0B0;
                    emit omega
                % when
            when immediate iota
            ||
            await
                case reset do emit iota; emit xi1
            end await;
            exit T
        end signal
    end trap
end module

module A0A:
    input set;
    output xi1;
```

continued on next page

continued from previous page

```
    trap T in
        signal alpha, iota, omega in
            emit alpha;
            suspend
                % suspend
                    halt;
                    emit omega
                             when immediate iota
            ||
            await
                case set do emit iota; emit xi1
            end await;
            exit T
        end signal
    end trap
end module

module A0:
    input c0,Inhib,reset,set;
    output nb2,b2,nb1,b1,nb0,b0,Alarm;
    signal gamma1,gamma2 in
        % init:
        emit gamma1;
        % main loop
        loop
            trap L in
                present gamma1 then
                    signal alpha in
                        emit alpha;
                        run A0A[signal gamma2/xi1];
                        present alpha else exit L end present
                    end signal
                end present;
                present gamma2 then
                    signal alpha in
                        emit alpha;
                        run A0B[signal gamma1/xi1];
                        present alpha else exit L end present
                    end signal
                end present;
                halt % horrible trick to fake the compiler
            end trap
        end loop
    end signal
end module

module A:
```

continued on next page

continued from previous page

```
      input c0,Inhib,reset,set;
      output nb0,b0,nb1,b1,nb0,b0,Alarm;
      % suspend
          run A0
      % when
end module
```

References

[BB91] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceeding of the IEEE*, 79(9):1270–1282, September 1991.

[Bee94] M. von der Beeck. A comparison of statecharts variants. In *Proc. of Formal Techniques in Real Time and Fault Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 128–148. FTRTFT'94, Springer-Verlag, 1994.

[Ber92] G. Berry. Preemption in concurrent systems. *Proc FSTTCS, Lecture notes in Computer Science*, 761:72–93, 1992.

[Har87] D. Harel. STATECHARTS: A visual formalism for complex systems. *Science of computer programming*, 8:231–274, 1987.

[IEC88] IEC, Genève (CH). *Preparation of function charts for control systems*, december 1988. International standard IEC 848.

[Mar90] F. Maraninchi. ARGOS*: un langage graphique pour la conception, la description et la validation des systèmes réactifs.* PhD thesis, Université Joseph Fourier, Grenoble I, Janvier 1990.

[Rei85] W. Reisig. *Petri nets: an introduction.* Monograph on Theoretical Computer Science. Springer-Verlag, Berlin, 1985.