Synchronous Interface Behavior

Syntax and Semantics

Charles ANDRÉ
I3S Research Report #00–11

August 1, 2000 (Version 0.1)
December 5, 2000 (Version 0.2)

Abstract

This report presents the syntax and the semantics of "Sequential Interface Behavior" (SIBs) proposed as a substitute for "Sequence Diagrams" of the UML. A sib expresses an input/output behavior in an unambiguous way. It supports quantitative timing constraints useful for real-time applications. The SIB's semantics is defined in terms of a synchronous process algebra, detailed in this report.

Keywords: Synchronous Programming, Process algebra, Sequence Diagrams, Reactive Programming.

1 Introduction

"Synchronous Interface Behavior" (SIB) extends "Sequence Diagrams" of the UML. It also borrows some features from "Message Sequence Charts". The novelty is in the synchronous approach adopted for SIB[1].

This technical report presents a formal semantics for SIB. The semantics is defined in terms of a synchronous process algebra. This technique has been successfully applied to the Esterel language [Ber93] and to the SyncCharts [And96]. Equipped with this semantics, a sib is an unambiguous description of an input/output behavior of a reactive system. Moreover, every sib can be translated into a semantically equivalent Esterel program, and thus, takes advantage of the platform (compiler, debugger, model-checker) developed for Esterel.

A sib is a special synchronous program. On the one hand, it is simpler than, say, an Esterel program. There is no instantaneous feedback from output to input: a SIB is only an observer [Hal93]. On the other hand, a sib offers high-level constructs dedicated to the expression of timing constraints. Last but not least, a siballows a limited form of nondeterminism, useful in specifications.

[1] SIB is the model. A sib is an instance of the model.

A sib is used to check properties of a reactive program, standing for a *controller*. The sib is translated into a pure Esterel module (i.e., an Esterel program with only pure signals) with counters (i.e., with integer constants, known at compile-time, used in repetitive statements). When the controller is programmed in Esterel, or SyncCharts, the controller program and the sib module are composed in parallel, giving a synchronous program on which properties can be checked.

2 Notations

2.1 Integers

$\mathbb{N}$ denotes the set of the non negative integers. Let $\mathbb{N}'$ be the set of non negative integer augmented with ω: $\mathbb{N}' = \mathbb{N} \cup \{\omega\}$.

$$\forall n \in \mathbb{N} : n < \omega \text{ and } \omega \pm n = \omega$$

2.2 Syntax

We use a BNF-notation. A terminal keyword is written in bold font (e.g., **Observed**). A terminal symbol is written in bold font, between accents (e.g., '**;**'). Non terminals are written in slant font (e.g., *list-of-signals*). Non terminals like *signal*, *integer*, ... are defined by classical regular expressions.

3 Textual Syntax

Note that this concrete syntax is only an intermediate format, the user should use a graphical one, not yet fully defined.

3.1 Grammar

Table 1 contains the rules for usual terms. Most rules are classical. The synchronous foundation of the model induces specific rules:

- Simultaneous occurrences are perfectly defined in a synchronous model: all the signals present at the same instant are simultaneous. The *signal-conj* non terminal denotes simultaneous occurrences.
- By default, when waiting for a signal occurrence, one waits for a strictly future occurrence. In some cases, the possible present occurrence may be significant. The modifier **immediate** means awaiting a present or future occurrence.

Table 1: Syntax of Basic SIBs

sib	::=	*declarations seq-of-actions*

Table 1: Syntax of Basic SIBs

declarations	::=	*observed timebase*
observed	::=	**Observed** '**=**' '**{**' *list-of-signals* '**}**'
timebase	::=	| **Timebase** '**=**' '**{**' *list-of-signals* '**}**'
action	::=	*expect* | *within* | *before* | *upto* | *parallel*
seq-of-actions	::=	*action* | *seq-of-actions* '**;**' *action*
parallel	::=	*seq-of-actions* '||' *seq-of-actions* | *parallel* '||' *seq-of-actions*
imm	::=	| **immediate**
optional	::=	**opt** *seq-of-actions* **end opt**
expect	::=	*expect-and*
expect-and	::=	**expect** *imm signal-conj*
within	::=	**do** *seq-of-actions* **within** *integer positive-inf signal-conj*
before	::=	**do** (*seq-of-actions* | *seq-of-actions* '**;**' *optional* | *optional* |) **before** *integer integer signal-conj*
upto	::=	**do** (| *optional*) **upto** *seq-of-cases* **end upto**
case	::=	**case** *imm signal-conj* **do** (| *seq-of-actions*)
seq-of-cases	::=	*case* | *seq-of-cases case*
signal	::=	[A-Za-z][A-Za-z_0-9]*
list-of-signals	::=	*signal* | *list-of-signal* '**,**' *signal*
signal-conj	::=	*signal* | *signal-conj* **and** *signal*
times	::=	| *positive*
times-inf	::=	| *positive-inf*
integer	::=	[0-9]+
positive	::=	[1-9][0-9]*
positive-inf	::=	*positive* | **infinity**

For convenience, extra rules are defined and some are modified (Table 2). Using them, makes the SIB more concise, maybe at the price of a slight lost of legibility.

Table 2: Syntax of Extended SIBs

action	::=	... | *watchdog* | *repetitive*
expect	::=	*expect-and* | *expect-or*
expect-or	::=	**expect** *signal-disj*
signal-disj	::=	*imm signal-conj* | *signal-disj* **or** *imm signal-conj*
repetitive	::=	**repeat** *positive-inf* **times** *list-of-actions* **end repeat**
watchdog	::=	**do** (*seq-of-actions* | *seq-of-actions* '**;**' *optional* | *optional*) **watchdog** *integer integer signal-conj*

3.2 Derived Constructs

The set of constructs in the SIB syntax is not minimal. For instance, **expect** can be derived from **upto**. In the next section, only the primitive constructs should be given a formal semantics. Nevertheless, non primitive constructs are useful in SIB descriptions, for they are concise forms.

3.2.1 Expect

expect is a degenerated form of **upto**.

Table 3: Expansions of the expect statement

expect S	≡	**do upto case** S **do end upto**
expect immediate S	≡	**do upto case immediate** S **do end upto**

Expect immediate: Iteration can be dangerous in synchronous programming in the case of instantaneous statements. The immediate multiple expect must be understood as: **expect immediate** n S ≡ **expect immediate** S; **expect** n-1 S.
This point will be clarified by the formal semantics (section 5).

Expect one among several events: this is another form of the expect.

```
expect S1 or S2 or S3  ≡  do
                          upto
                                  case S1 do
                                  case S2 do
                                  case S3 do
                          end upto
```

3.2.2 Upto

The **upto** construct is *deterministic* (this is a deliberate choice of the author). When several expected events are present, the first (textual order) matching case is taken. Thus, the order of the case-clauses indicates a priority.

The or-list of signals in an expect with disjunction statement is commutative only because actions in the case-clauses are void. This commutativity must not be generalized to other upto constructs.

4 Graphical Syntax

4.1 Basic Constructs

The flow of control is top-down in a sib. A vertical red line represents this flow. *Expected events* are denoted by solid red dots on this line. Fig. 1 shows that a signal may be received (input) or sent (output). The distance between two consecutive expects is meaningless, only the ordering is relevant, in accordance with the logical time used by synchronous models.

The SIB is a structured block-oriented model. Expects are the bricks. The main construct is the *sequence*. In the figures below, sequences are drawn as green vertical rectangles. A sequence is an ordered set of actions. Expect is the simplest action. Any block described below is also an action.

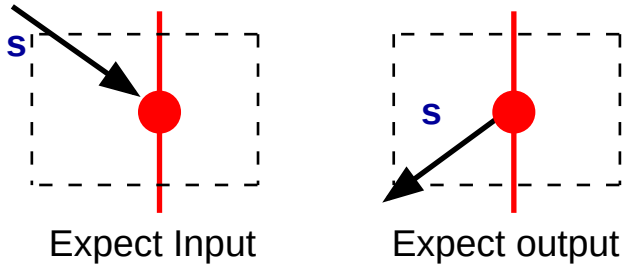

Figure 1: Expect.

4.2 Temporal behaviors

In real-time applications, event occurrences are temporally constrained. Binding logical time to real time can be done by relevant events (e.g., a 1 kHz physical clock that generates signal "`ms`"). Multi-form (discrete) timing constraints are expressed by two special constructs: `within` and `before` (Fig. 2).

Basically, to be accepted, a sequence of actions must be completely matched within a temporal window for the `within` construct, and `before` a deadline for the before construct. These constraints can be too restrictive for high-level specifications. So, we weaken them. First, we introduce a slack in the deadline occurrence: the deadline can occur at random within a given interval. Second, we permit departure from the rule of full matching of the sequence of actions. The trailing part of the sequence can be *optional*, that is, when the deadline occurs, not fully matching the sequence in the optional part does not make the sib not applicable. Graphically, the optional part of the sequence is denoted by a dashed control flow line. Fig. 2 shows the graphical notation for the `within` and the `before` constructs. Below are examples with their (informal) semantics.

- "`do p within 1 .. 4 ms`" says that the sequence actions "`p`" must take place between the first and the fourth future occurrence of `ms`. Note that replacing ms with meter is semantically perfectly correct and allows generalized forms of time constraints: "`stop_the_car within 20 .. 30 meter`".
- "`do p before 8 .. 10 ms`" expresses another behavior: the sequence of actions "`p`" must terminate before a delay of 8, 9 or 10 occurrences of ms has elapsed. The deadline is clearly denoted by an (implicit) expect, at the exit of the before block. Taking a lower bound different from the upper bound of the delay is a way to introduce a restricted form of non determinism. At each instant between the lower and the upper bound, the deadline can or cannot occur, arbitrarily.

4.3 Synchronous specific constructs

Since instants are discrete, one can expect a stricly future occurrence of an event, or consider a possible present occurrence.

Synchronous modeling induces subtle issues, generally irrelevant to traditional approaches. Since simultaneous occurrences are possible, one can expect a conjunction of event occurrences.

Our notation captures these nuances (see Fig. 3).

4.4 Advanced constructs

A strict sequential representation of concurrent evolutions needs interleaving of events and induces "parasitic" ordering. A `parallel` construct is better at expressing partial ordering. Fig. 4 shows the parallel construct made of at least two sequences. This is a restricted form of concurrency (fork-join).

The `upto` construct, in Fig. 4, expresses alternative: One out of several sequences is taken. The taken sequence is the one whose guarding event occurs first. If several guarding events occurs at the same instant,

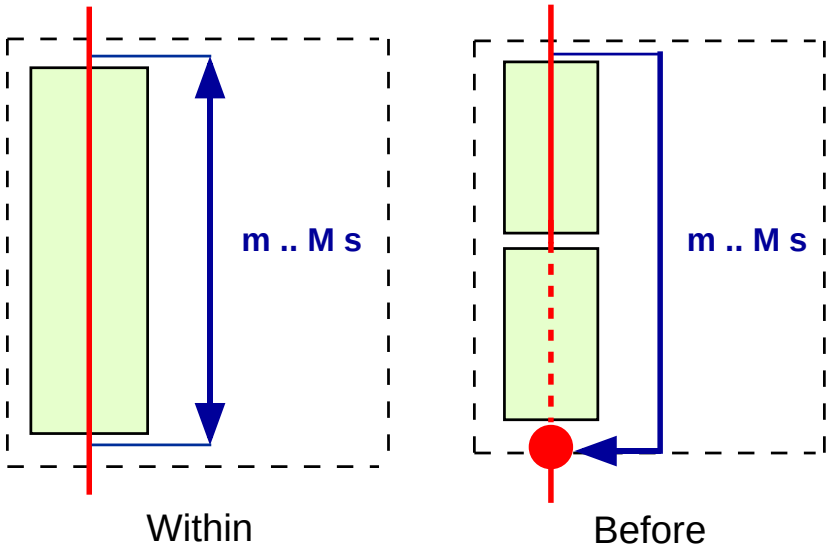

Figure 2: Temporal constraints.

which is perfectly possible in a synchronous model, the left-most opened sequence is taken. Thus, we have a deterministic choice. This construct has been called upto, because before the occurrence of the selecting event, the sib was awaiting in an optional sequence, and stays there up to the occurrence of a selecting event.

Like with MCS, subsequences can be iterated. The `repeat` construct (Fig. 5) allows folding of sequences. This is only "syntactic sugar" that denotes the unfolding of the loop.

A last construct is also very useful, especially in real-time systems and in protocols. We call it the `watchdog` construct (Fig. 5). This block is left on the occurrence of a deadline event, the solid red dot on the exit of the block. As suggested by the picture, the `watchdog` construct is akin to the before block. The difference is that the former uses a disarming of the deadline: if the sequence in the watchdog block terminates before the deadline, then, the deadline is "re-armed". For flexibility, the number of occurrences may vary within an interval, and the sequence may have one optional part.

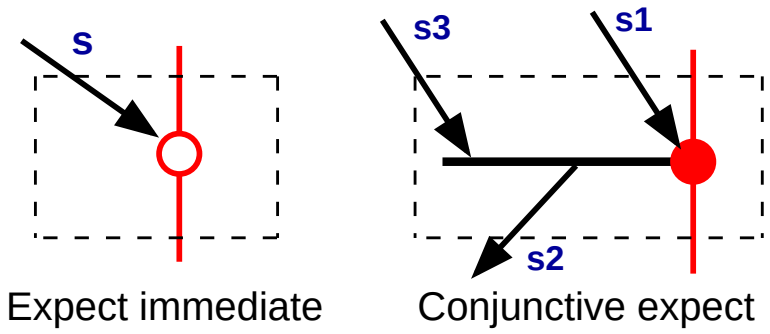

Figure 3: Variants of Expect.

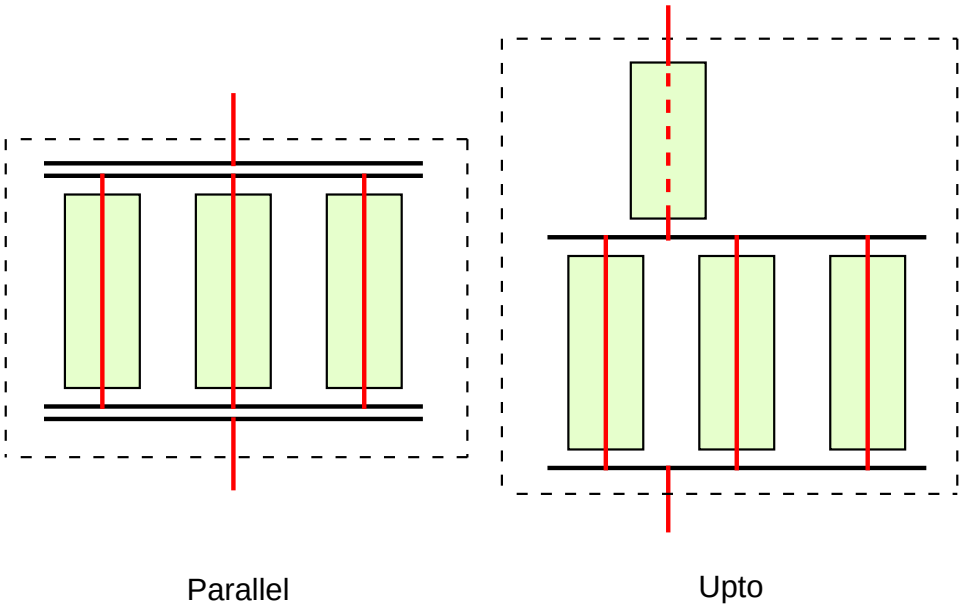

Figure 4: Advanced SIB (1).

5 Process Algebra

5.1 Reactions

Let $\mathcal{I}$ be the set of signals seen by the application (*i.e.*, the controller). $\mathcal{I} \supseteq$ Observed $\cup$ Timebase. The set of output signals is $\mathcal{O} = \{$Active, Accepted, Not_Applicable$\}$.

At the j^{th} instant, let I_j be the current input event: $I_j \subset \mathcal{I}$ such that

$$\forall S \in \mathcal{I}, S \in I_j \iff S \text{ is present at instant } j$$

For a given sequence of input events $I_1 \; ; I_2 \; ; \cdots$, the behavior of a SIB "p" is defined by a sequence of *reactions*:

$$\begin{aligned} p \;\; = \;\; & p_0 \xrightarrow[I_1]{O_1} p_1 \xrightarrow[I_2]{O_2} \cdots \xrightarrow[I_n]{O_n} p_n \xrightarrow[I_{n+1}]{\emptyset} 0 \xrightarrow[I_{n+2}]{\emptyset} 0 \cdots \\ & \text{for some } n \in \mathbb{N}', \\ & \text{and } O \in \{\{\text{Accepted}\}, \{\text{Not\_Applicable}\}, \{\text{Active}\}\}. \end{aligned}$$

If n is finite, the execution of p is said to terminate at instant n.

A reaction is computed by induction on the structure of the term. For this, we use an auxiliary relation (structural transition) defined by conditional rewriting rules. A structural transition is denoted by:

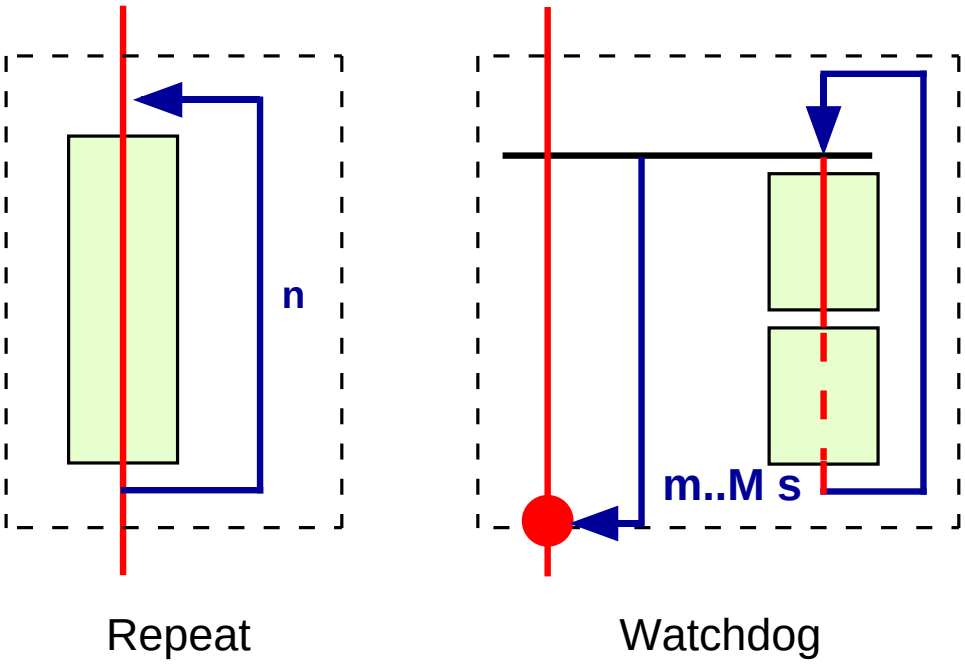

Figure 5: Advanced SIB (2).

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

where p is a term of the algebra, E the signal environment (the set of present signals), A the set of signals *accepted* by p under E. k is either an integer or ω, called the termination code, b is a Boolean that indicates whether the transition has been done in an optional process, or not. p' is the residue of p after the rewriting. $k = \omega$ means that p has got through a deadline.

$$p_j \xrightarrow[I_j]{O_j} p_{j+1} \quad \text{iff there exists a rewriting} \quad p_j \xrightarrow[I_j]{A,k,b} p_{j+1}$$

O_j and the continuation depends on A and k. Let $I'_j = I_j \cap$ Observed be the set of present signals at instant j restricted to the set of Observed signals.

$$\begin{aligned}
O_j &= \{\texttt{Not_Applicable}\} \text{ and the execution terminates if} \\
&\quad (I'_j \setminus A \neq \emptyset) \vee (k = \omega) \\
O_j &= \{\texttt{Accepted}\} \text{ and the execution terminates if} \\
&\quad (I'_j \setminus A = \emptyset) \wedge (k = 0) \\
O_j &= \{\texttt{Active}\} \text{ and the execution pauses till the next instant, otherwise}
\end{aligned}$$

5.2 Process

Instead of refering to the syntactic constructs of SIB, the process algebra adopts a terse notation. Some operators are already known to readers familiar with the process algebra of the Esterel language [Ber93]. Others are specific to SIB, they are mentioned in the lower part of the table.

Remark 1: Some classical Esterel constructs like the `suspend` are not used. The `trap` construct of Esterel, and its algebraic notation $\{p\}$ is not directly available. It is used to express the semantics of some derived statements. The same remark applies to the signal hiding ($p \setminus S$).

Remark 2: This set of terms is not minimal.

Table 4: Algebraic Terms

0	nothing	pass the control instantly
1	pause	wait for the next instant
!S	emit	$S \notin$ Observed $\cup$ Timebase
$S?p, q$	presence	test the presence of S and execute either p, or q
$p \,;\, q$	sequence	
$p \mid q$	parallel	
p^*	loop	
$\{p\}$	trap	
$\uparrow p$	shift	
$p \setminus S$	hiding of signal	
k	exit	where $k \geqslant 2$
$p < S$	strong abortion	
$p \times n$	repeat	finite iteration
[p]	optional	optional process
$p \sqsubset S_m^M$	within	p in within $[m..M[\ S$
$p \curvearrowright S_m^M$	before	p before $k \in [m..M]\ S$
$p \Cap_{i \in 1..n} \lessdot(S_i \,;\, q_i)$	upto	

5.3 Auxiliary rules

In the following rules, assume $m \leq M$.

5.3.1 Classical rules

These rules are given without comments, see Esterel or SyncCharts semantics for details.

Terminaison :

$$k \xrightarrow[E]{\emptyset,k,\texttt{ff}} 0 \qquad \text{(term)}$$

Emit :

$$!S \xrightarrow[E]{\{S\},0,\texttt{ff}} 0 \qquad \text{(term)}$$

Presence :

$$\frac{S \in E \qquad p \xrightarrow[E]{A_p,k,b_p} p'}{S?p,q \xrightarrow[E]{A_p,k,b_p} p'} \qquad \text{(pres+)}$$

$$\frac{S \notin E \qquad q \xrightarrow[E]{A_q,k,b_q} q'}{S?p,q \xrightarrow[E]{A_q,k,b_q} q'} \qquad \text{(pres-)}$$

Sequence :

$$\frac{p \xrightarrow[E]{A_p,0,b_p} p' \qquad q \xrightarrow[E]{A_q,k,b_q} q'}{p\,;q \xrightarrow[E]{A_p \cup A_q,k,b_q} q'} \qquad \text{(seq1)}$$

$$\frac{p \xrightarrow[E]{A_p,k,b_p} p' \qquad k \neq 0}{p\,;q \xrightarrow[E]{A_p,k,b_p} p'\,;q} \qquad \text{(seq2)}$$

Parallel :

$$\frac{p \xrightarrow[E]{A_p,k_p,b_p} p' \qquad q \xrightarrow[E]{A_q,k_q,b_q} q'}{p \mid q \xrightarrow[E]{A_p \cup A_q,max\{k_p,k_q\},b_p \wedge b_q} p' \mid q'} \qquad \text{(par)}$$

Loop :

$$\frac{p \xrightarrow[E]{A_p,k,b_p} p' \qquad k \neq 0}{p^* \xrightarrow[E]{A_p,k,b_p} p' \,;\, p^*} \quad \text{(loop)}$$

Trap :

$$\frac{p \xrightarrow[E]{A_p,k,b_p} p' \qquad k = 0 \text{ or } k = 2}{\{p\} \xrightarrow[E]{A_p,0,b_p} 0} \quad \text{(trap1)}$$

$$\frac{p \xrightarrow[E]{A_p,k,b_p} p' \qquad k = 1 \text{ or } k > 2}{\{p\} \xrightarrow[E]{A_p,\downarrow k,b_p} \{p'\}} \quad \text{(trap2)}$$

$$\frac{p \xrightarrow[E]{A_p,k,b_p} p'}{\uparrow p \xrightarrow[E]{A_p,\uparrow k,b_p} \uparrow p'} \quad \text{(shift)}$$

Signal hiding : Without loss of generality, we assume that the signal name S is not used elsewhere. Apply a renaming of signals if it is not the case.

$$\frac{p \xrightarrow[E\cup\{S\}]{A_p\cup\{S\},k,b_p} p'}{p \setminus S \xrightarrow[E]{A_p,k,b_p} p' \setminus S} \quad \text{(sig+)}$$

$$\frac{p \xrightarrow[E\setminus\{S\}]{A_p\setminus\{S\},k,b_p} p'}{p \setminus S \xrightarrow[E]{A_p,k,b_p} p' \setminus S} \quad \text{(sig-)}$$

5.3.2 SIB specific rules

Up to here, we have only consider the presence or the absence of signals. This is sufficient for “pure” Esterel programs or “pure” SyncCharts. Now, we introduce counters. Counters are integer-valued variables used for the control of the program. Note that signals remain pure signals.

Repeat : This statement expresses finite iteration.

$$\frac{n > 1 \quad p \xrightarrow[E]{A_p,k,b} p' \quad k \neq 0}{p \times n \xrightarrow[E]{A_p,k,b} p' \,;\, p \times (n-1)} \quad \text{(rep)}$$

By convention, $p \times 1 \equiv p$.

Optional : remind that optional sequences need not being fully matched when preempted.

$$\frac{p \xrightarrow[E]{A_p,0,b_p} p'}{[p] \xrightarrow[E]{A_p,0,\mathtt{tt}} 0} \quad \text{(opt1)}$$

$$\frac{p \xrightarrow[E]{A_p,k,b_p} p' \quad k \neq 0}{[p] \xrightarrow[E]{A_p,k,\mathtt{tt}} [p']} \quad \text{(opt2)}$$

within :

$$\frac{m > 0 \quad S \notin E}{p \sqsubset S_m^M \xrightarrow[E]{\emptyset,1,\mathtt{ff}} p \sqsubset S_m^M} \quad \text{(win1)}$$

$$\frac{m > 0 \quad S \in E}{p \sqsubset S_m^M \xrightarrow[E]{\emptyset,1,\mathtt{ff}} p \sqsubset S_{m-1}^{M-1}} \quad \text{(win2)}$$

$$\frac{M > 0 \quad p \xrightarrow[E]{A_p,0,b_p} p'}{p \sqsubset S_0^M \xrightarrow[E]{A_p,0,b_p} 0} \quad \text{(win3)}$$

$$\frac{M > 0 \quad S \notin E \quad p \xrightarrow[E]{A_p,k,b_p} p' \quad k \neq 0}{p \sqsubset S_0^M \xrightarrow[E]{A_p,k,b_p} p' \sqsubset S_0^M} \quad \text{(win4)}$$

$$\frac{M > 1 \quad S \in E \quad p \xrightarrow[E]{A_p,k,b_p} p' \quad k \neq 0}{p \sqsubset S_0^M \xrightarrow[E]{A_p,k,b_p} p' \sqsubset S_0^{M-1}} \quad \text{(win5)}$$

$$\frac{M > 1 \quad S \in E \quad p \xrightarrow[E]{A_p,k,b_p} p' \quad k \neq 0}{p \sqsubset S_0^M \xrightarrow[E]{A_p,\omega,b_p} 0} \quad \text{(win5')}$$

$$\frac{S \in E \quad p \xrightarrow[E]{A_p,k,b_p} p' \qquad k \neq 0}{p \sqsubset S_0^1 \xrightarrow[E]{A_p,\omega,\mathtt{ff}} 0} \quad \text{(win6)}$$

Comments :

- (win1): Not in the execution window. p should not react.
- (win2): Not in the execution window. p should not react. Since $S \in E$, decrement the bounds.
- (win3): Within the window. p terminates, so does the within.
- (win4): Within the window. p does not terminate.
- (win5): Within the window. Since $S \in E$, the deadline can occur. Here, the deadline is postponed. Let p react and decrement M.
- (win5'): Within the window. Since $S \in E$, the deadline can occur. Here, the deadline occurs. Let p react and terminate with a deadline violation ($k = \omega$).
- (win6)Within the window. Since $S \in E$ and $M = 1$, the deadline *must* occur. Let p react and terminate with a deadline violation ($k = \omega$).

Before : this construct leaves the block when the deadline occurs.

$$\frac{S \notin E \quad p \xrightarrow[E]{A_p,k,b_p} p'}{p \curvearrowright S_m^M \xrightarrow[E]{A_p,k,b_p} p' \curvearrowright S_m^M} \quad \text{(bef1)}$$

$$\frac{m > 0 \quad S \in E \quad p \xrightarrow[E]{A_p,k,b_p} p'}{p \curvearrowright S_m^M \xrightarrow[E]{A_p,k,b_p} p' \curvearrowright S_{m-1}^{M-1}} \quad \text{(bef2)}$$

$$\frac{S \in E \quad p \xrightarrow[E]{A_p,k,b_p} p' \qquad (k = 0) \vee (b_p = \mathtt{tt})}{p \curvearrowright S_0^1 \xrightarrow[E]{A_p,0,\mathtt{ff}} 0} \quad \text{(bef3)}$$

$$\frac{S \in E \quad p \xrightarrow[E]{A_p,k,b_p} p' \qquad k \neq 0 \qquad b_p = \mathtt{ff}}{p \curvearrowright S_0^1 \xrightarrow[E]{\emptyset,\omega,\mathtt{ff}} 0} \quad \text{(bef3')}$$

$$\frac{M > 1 \quad S \in E \quad p \xrightarrow[E]{A_p,k,b_p} p'}{p \curvearrowright S_0^M \xrightarrow[E]{A_p,k,b_p} p' \curvearrowright S_0^{M-1}} \quad \text{(bef4)}$$

$$\frac{M > 1 \quad S \in E \quad p \xrightarrow[E]{A_p,k,b_p} p' \quad (k = 0) \vee (b_p = \texttt{tt})}{p \curvearrowright S_0^M \xrightarrow[E]{A_p,0,b_p} 0} \quad \text{(bef5)}$$

$$\frac{M > 1 \quad S \in E \quad p \xrightarrow[E]{A_p,k,\texttt{ff}} p' \quad k \neq 0}{p \curvearrowright S_0^M \xrightarrow[E]{A_p,\omega,\texttt{ff}} 0} \quad \text{(bef5')}$$

Comments :

- (bef 1): The deadline cannot occur (The time base signal is absent). Reaction of p only.
- (bef2): The deadline cannot occur ($m > 0$), but the time base signal is present: p reacts and bounds are decremented.
- (bef3): The deadline must occur ($m = 0, M = 1$, and the time base signal is present). p terminates or was in an optional sequence: The process terminates.
- (bef3'): The deadline must occur ($m = 0, M = 1$, and the time base signal is present). p does not terminates and was not in an optional sequence. Then the kill termination is set.
- (bef4): The deadline can occur ($m = 0, M > 1$, and the time base signal is present). The deadline is delayed, p reacts and M is deceremented.
- (bef5): The deadline can occur ($m = 0, M > 1$, and the time base signal is present). The deadline actually occurs, p reacts and terminates or was in a optional sequence. The process terminates
- (bef5'): The deadline can occur ($m = 0, M > 1$, and the time base signal is present). The deadline actually occurs, p reacts and does not terminate and was not in a optional sequence. The kill termination code is set.

Upto : An optional sequence must be preempted by guarding signals. The choice is deterministic.

The general form is $p \Uparrow_{i \in 1..n} \lhd (S_i \,;\, q_i)$, where $\lhd \in \{\lessdot, <\}$. We write rules for the immediate form only ($\lessdot$). Transforming a delayed guarding signal ($<$) into an immediate can be done with local signals:

$$p < S \,;\, q \equiv \{(((p \lessdot S' \,;\, q) \,;\, 2) \mid (1 \,;\, (S?!S', 1)^*)) \setminus S'\}$$

Let $S = \{S_1, \cdots, S_n\}$.

$$\frac{E \cap S = \emptyset \qquad p \xrightarrow[E]{A_p,k,b_p} p'}{p \Uparrow_{j\in 1..n}(\lessdot S_j\ ;\ q_j) \xrightarrow[E]{A_p,k,b_p} p' \Uparrow_{j\in 1..n}(\lessdot S_j\ ;\ q_j)} \quad \text{(upt1)}$$

$$\frac{\exists l \leq n\ \ S_l \in E \wedge (\forall i < l\ \ S_i \notin E) \qquad p \xrightarrow[E]{A_p,k,b_p} p'}{p \Uparrow_{j\in 1..n}(\lessdot S_j\ ;\ q_j) \xrightarrow[E]{(E\cap S)\cup A_p,0,\mathtt{ff}} q_l} \quad \text{(upt2)}$$

Comments :

- (upt1): No guarding signal is present. Let p react.
- (upt2): At least one guarding signal is present. Let S_l the left-most (the one with the highest priority). Let p react, and terminate. Possible deadline violation in p is "forgiven" since executed in the optional sequence of the upto construct.

The shift operator :

$$\downarrow k = \begin{cases} 0 & \text{if } k = 0 \text{ or } k = 2 \\ 1 & \text{if } k = 1 \\ k-1 & \text{if } k > 2 \end{cases}$$

$$\uparrow k = \begin{cases} k & \text{if } k = 0 \text{ or } k = 1 \\ k+1 & \text{if } k > 1 \end{cases}$$

Watchdog :

The watchdog statement is not a primitive. "**do** p **watchdog** $m..MS$" can be expressed by

$$\{\{(p\ ;\ 2 \curvearrowright S_m^M)\ ;\ 3\}^*\}$$

References

[And96] C. André. Representation and analysis of reactive behaviors: A synchronous approach. In *Computational Engineering in Systems Applications (CESA)*, pages 19–29, Lille (F), July 1996. IEEE-SMC.

[Ber93] G. Berry. Preemption in concurrent systems. In W. Brauer, editor, *FSTTCS 93*, volume 761 of *Lecture Notes in Computer Science*, pages 72–93. Springer-Verlag, 1993.

[Hal93] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, Amsterdam, 1993.