Computing SyncCharts Reactions

Charles ANDRÉ

I3S Laboratory (UMR 6070)
University of Nice Sophia Antipolis / CNRS
BP 121, Sophia Antipolis cedex, 06903, FRANCE
{andre@unice.fr}

ISRN I3S/RR-2003-09-FR**

Abstract. SYNCCHARTS are a state-based visual synchronous model. Though using a simple graphical syntax, SYNCCHARTS may exhibit complex instantaneous behavior, mixing concurrent evolutions, preemptions and state re-incarnations. This paper explains such reactions in terms of *microsteps*. The underlying semantics is a *constructive semantics*, fully compatible with the ESTEREL's semantics. The semantics is presented in a semi-formal way, as resulting from the cooperation of concurrent *reactive cells*.

1 Introduction

The reactive systems are *discrete-event systems*. They are mostly event-driven, which means that they perform little processing on their own and that their behavior can be represented as a sequence of *reactions to stimuli*.

A classical way of representing sequential evolutions is to resort to state-transition models. However the simplest forms of automata are not convenient to cope with the complexity of modern applications. Representing the behavior of such systems implies hierarchical description, support of concurrency and synchronization, and communication between the different parts of the system. Synchronous languages have been introduced to address the issues of reactive system programming. In the imperative synchronous language ESTEREL[1], communication and synchronization are unified under the concept of *signals*. The stimuli that provoke reactions are associated with emission and reception of signals. Signals are also the actors of preemption, making it possible to suspend (to freeze) or to abort the behavior of some parts of the system. Synchronous models have adopted a simplifying hypothesis: the system evolves only during discrete phase (instant), the duration of which is 0. Thus, the computation of a reaction, which may result from complex interactions among parts of the program, is supposed to be instantaneous. This strong hypothesis, augmented with the hypothesis of instantaneous broadcasting of signals, allows deterministic behaviors even in the presence of concurrency and preemption. ESTEREL excels in expressing complex reactive behaviors, but it does not directly support specifications in terms of hierarchical communicating finite state machines. SYNCCHARTS [2] have been introduced as a graphical form of the Esterel language, which adopts states as first-class citizens. Being direct descendants of Esterel, SyncCharts have inherited the mathematical semantics of the language, along with its advantages and its drawbacks. On the other hand, SyncCharts owe their look (syntax) to Harel's STATECHARTS [3]. This double inheritance may be misleading for the beginner: a syncChart[1] looks like a statechart but its behavior may be different.

Sections 2 and 3 of this paper are mostly educative. They aim at sensitizing the reader about SyncCharts semantics through simple examples. SyncCharts—an apparently simple model—may have complex (instantaneous) reactions, which strictly respect the synchronous hypotheses. These reactions reflect the underlying

** This report is an extended version of the paper presented at SLAP'03 (Synchronous Languages, Applications and Programming), July 1st, 2003 – Porto (P) – URL: http://www.elsevier.nl/locate/entcs/volume88.html

[1] SYNCCHARTS is the model. A *syncChart* is a particular instance of the model.

semantics of SyncCharts, introduced in Section 4. The semantics is described in terms of *microsteps* that respect the *constructive causality*: before being tested,the presence status of a signal must be determined at a prior microstep. This is the essence of the *constructive semantics* [4] introduced by G. Berry for the Esterel language. This semantics is presented in a semi-formal way, heavily relying on the structure of SyncCharts. This structure is precisely defined by an UML model. The semantics is given in an operational style (detailed in the Appendix) suitable for simulation at the microstep level, leading to a better understanding of SyncCharts reactions.

2 A Tour of SyncCharts

2.1 Illustrating Example

SyncCharts consist of states and transitions for their structure, and signals for their dynamics. A typical syncChart is drawn in Fig.1

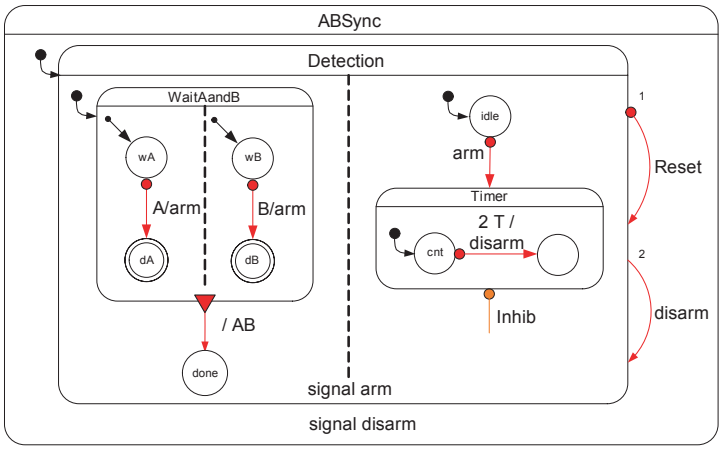


Fig. 1. A Typical SyncChart: This syncCharts specifies a detector of synchronized occurrences. Signal AB must be emitted when A and B have occurred, in any order, but "close enough" in time. "Close enough" means that there are at most two occurrences of T between the occurrences of A and B. Reset is a signal re-initializing signal. Finally, Inhib is a signal that enables "time suspension": when Inhib is present, possible occurrences of T are ignored.

2.2 Simplified Syntax of SyncCharts

Structure. A *state* can be either a simple state, drawn as a simple circle or ellipse, or a macrostate drawn as a rounded rectangle. A *macrostate* is refined: it contains other states. A *simple state* does not. A state may have a name: written inside for a simple state, written in a cartouche for a macrostate.

A *transition* is a directed link between two states (from a *source* state and a *target* state). There are three types of transitions: *strong abortion*, *weak abortion*, and *normal termination* transitions. The transition from state wA to state dA is a strong abortion transition. The self-loop transition from state Detection to itself, and

labeled disarm is a weak abortion transition. Finally the transition from state WaitAandB to state done is a normal termination one.

A *State-Transition Graph* (STG) is a connected set of states. A STG must have an *initial state*, graphically denoted by an arc pointing to it. States Detection, wA, wB... are initial states for different STGs. A STG may also have *final states*, denoted by double circles. States dA and dB are final states. STGs are necessarily contained in a macrostate. When several STGs are in the same macrostate, they are separated by dashed lines, and they are said to be concurrent. Note that contrary to statecharts, SYNCCHARTS respect a strict containment policy: there is no inter-level transition. Macrostate Detection contains two concurrent STGs, and so is macrostate WaitAandB. Outgoing transitions from a state are ordered: an integer called the priority is attached to the origin of the transition. For instance, two transitions leave state Detection. The strong abortion transition has priority 1, while the weak abortion transition has priority 2.

The "lollipop" at the bottom of macrostate Timer is a *suspension arc*. A label can be associated with a transition, an initial arc, or a suspension arc. These labels refer to signals presented in the next subsection.

Signals. *Signal* is our unique abstraction for handling communication and synchronization. Emitting or receiving a signal meets with the classical notion of event. A signal has a *presence status*: present(+), absent(-), or unknown($\perp$). It may convey a *value* of a given type. If a signal can be emitted several times within one reaction, a combination function is also required. A signal that conveys no value is called a *pure signal*. Finally, a signal has a *scope*: either external or local to a macrostate. External signals are further classified in input signals and output signals. Local signals are bi-directional but used only for internal communication.

Local signals are explicitly mentioned on a syncChart: arm is signal local to macrostate Detection; disarm is local to macrostate ABSync. External signals are not explicitly declared in the syncChart. The input signals of the outermost macrostate are A, B, Reset, T, and Inhib. There is only one output signal: AB.

Label Associated with a Transition. The general syntax for a label is *trigger [guard] / effect*. A *trigger* may be a single signal the presence of which is expected to fire the transition. An optional integer factor indicates that several consecutive occurrences of the signal are expected (e.g., the trigger "2 T" of the transition in macrostate Timer). More complex triggers are expressions on signals using logical operators (`and`, `or`, `not`). A *guard* is an expression that evaluates to true or false. The expression uses values of signals and constants. An *effect* is a set of (instantaneous) actions. Emitting a signal is a possible action. Trigger, guard, and effect are optional.

Label associated with a simple state. An *effect* can be associated with a simple state. For pure SyncCharts (i.e., a syncChart with pure signal only) this effect consists of a possibly empty set of emitted signals. The syntax of the label is */ effect*.

2.3 Informal Semantics

Up to now, we have only presented syntactical aspects of SYNCCHARTS. Since SYNCCHARTS are intended to represent reactive behavior, the main thing is to understand the dynamics of the model. In this section, explanations are kept informal. Moreover, we focus on *pure syncCharts*, that is, syncCharts with pure signals only. Reactivity has strong connection with the presence/absence of signals. Pure SYNCCHARTS are sufficient to explain the essence of a reaction. This simplified approach has already been successfully adopted for the ESTEREL's semantics [4].

Reaction. A syncChart, like other synchronous models, is "executed" cyclically. An evolution cycle is as follows:

1. Read the inputs: in our restricted presentation, this means "get the presence status of each input signal, yielding an input image".
2. Compute the reaction: according to the internal state of the syncChart and the input image, compute the new internal state and the output image (i.e., find for each output signal its new presence status).
3. Perform the outputs (i.e., effectively deliver output signals to the environment).

This process is supposed to take no time. An *instant* is fully characterized by the associated reaction. Note that the reaction must be *deterministic*. Thus, computing the reaction of a syncChart in a fully deterministic way is a central problem.

Communicating FSMs. A first approach is to consider a syncChart as a set of communicating finite state machines (a state machine for each STG contained in the syncChart). "A finite state machine is a machine specified by a finite set of conditions of existence (called states) and a likewise finite set of transitions among states triggered by events". This definition given by B. P. Douglass [5] applies to SYNCCHARTS, provided events are replaced by signals. As usual, a state characterizes a condition that may persist for a significant period of time. When in a state, the system is reactive to a set of signals and can reach (take or fire a transition to) other states based on the signals it accepts. Suppose that states wA, wB, and idle are active in syncChart ABSync. If A is present, then the transition from state wA to state dA is taken. The associated effect (emission of local signal arm) is executed. Now, this signal is instantaneously broadcast. State idle was waiting for the presence of arm. The presence of arm triggers the transition from state idle to macrostate Timer. Entering macrostate Timer, causes the activation of the initial state of the STG in Timer. Further evolution is no longer possible in this reaction. Therefore, signal AB is not emitted during this reaction (presence status set to absent). Thus, a reaction appears as a sequence of instantaneous transitions or *microsteps* driven by causality relationship. The global result is an instantaneous change of active states. The causality chain may be cyclic: its instantaneous execution is obviously unacceptable. In such a case, the syncChart is rejected.

The previous reaction implies only strong abortion transitions on simple states. *Strong abortion* applies to macrostates as well. When Reset is present, whatever the presence status of the other input signals, macrostate Detection is exited without any prior internal evolution of the macrostate. The target state being the same macrostate, Detection is re-entered during the reaction and recursively activates initial states of the enclosed STGs. A *weak abortion* is performed differently: before exiting the macrostate origin of the weak abortion transition, the internal evolutions of the macrostate are executed (see example below).

A *normal termination* transition has no explicit trigger. Such a transition is fired as soon as each STG of the source macrostate is in a final state. For instance, if states dA and wB are active, and B is present, the transition from wB to dB is taken. Now the two STGs in macrostate WaitAandB are in a final state. Instantaneously, the normal termination transition is taken, signal AB is emitted, and state done becomes active.

We can now illustrate the firing of the unique weak abortion transition of the example. Suppose that states dA, wB, and cnt are active. What is the reaction of the syncChart if B and T are present, and T is the second occurrence of T since cnt has been active? The transition whose trigger is "2 T" is taken. The local signal disarm is emitted. The weak abortion is triggered, but before exiting macrostate Detection, possible internal evolutions must be executed. This is the case for the normal termination of macrostate WaitAandB previously presented. Thus, signal AB is emitted. Now, no further evolution in Detection is possible; macrostate Detection is effectively exited, and then re-entered as already explained for the strong abortion.

Note that, when several transitions issued from the same state are simultaneously eligible for firing, only the one with the highest priority (the smallest integer value) is taken. In order to avoid inconsistent decisions,

SYNCCHARTS impose the following restriction: strong abortions have priority over weak abortions, which have priority over a normal termination.

Normal termination is not strictly necessary: the same behavior can be obtained by a weak abortion triggered by a conjunction of local signals (see Fig.2). We keep this convenient construct that is more readable than its weak abortion counterpart.

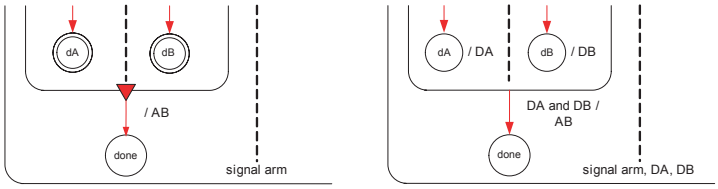


Fig. 2. Normal termination can be replaced by weak abortion and local signals.

Finally, *suspension* is used to "freeze" evolutions within a macrostate. When Inhib is present, Timer will ignore possible occurrences of T, and, of course, can not emit disarm.

These few examples show the complexity of a reaction, which may result from partially ordered transition firings. In fact, communicating FSMs are not well-adapted to hierarchy, preemption, and above all, instantaneous chaining of transition firings. Instead of communicating FSMs, we choose cooperating *reactive cells* as active agents.

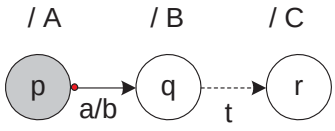
Reactive Cell. In SYNCCHARTS, the default behavior is to stay in a state for ever. A state can be exited only by firing an abortion transition (recall that normal termination is a special case of weak abortion), or indirectly by exiting a containing macrostate. Thus, active agents in a syncChart are states with their outgoing transitions. We call *reactive cell* a state with its outgoing transitions. A reactive cell can be *active* (alive) or *idle* (doing nothing at all). An active reactive cell is permanently testing the triggers associated with its transitions. As soon as a transition can be taken, the reactive cell is deactivated and the reactive cell target of the transition is activated. A reaction now appears as a propagation of activations/deactivations among a collection of cells. This execution model has been inspired by Boussinot's reactive objects [6], though reactive objects address distributed applications, and are not subject to strict synchronous hypotheses.

Reactive Cells will be used in our more formal presentation of the model (Sec.4).

3 Advanced Features

3.1 Immediate Preemption

When entering state wA in macrostate WaitAandB, if A is present at this very instant, the outgoing transition is not taken. This is the usual behavior: a trigger waits for a *strictly future* occurrence. This default behavior can be modified using the *immediate* qualifier for a transition (denoted by the # symbol prefixing the trigger). With immediate transitions some states can possibly become transient, that is, activated and then deactivated during the same reaction. Note that they are still genuine states, because the control can stay in them, under some circumstances. Immediate reaction combined with preemption results in different behaviors, as illustrated by Fig.3. Note that, in this example, the immediate strong abortion makes state q to be bypassed.



transition t	new state	emitted signals
$\xrightarrow{b}$	$\{q\}$	b, B
$\xrightarrow{\#b}$	$\{r\}$	b, B, C
$\circ\!\!\xrightarrow{\#b}$	$\{r\}$	b, C

Fig. 3. Immediate Reactions: Behavior according to the type of preemption applied to state q, when p is active, and a occurs.

3.2 Re-incarnation

Loop, immediate preemptions, and priority can lead to amazing, but perfectly consistent behaviors. Fig.4 shows an example especially devised to illustrate these complex interactions. For this example, we use valued signals for a better traceability, but the succession of microsteps would have been the same with pure signals. The signal v is an integer with the multiplication as the combination function. The value emitted by each transition is a different prime number, so that, the value conveyed by v faithfully reflects the transitions fired during the reaction[2].

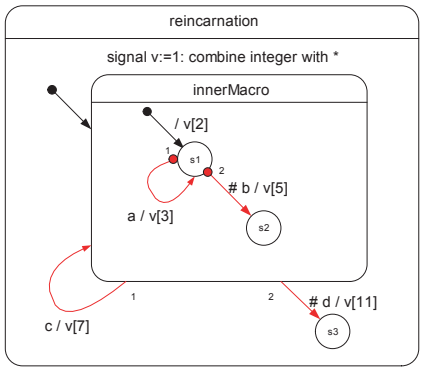


Fig. 4. State Reincarnation.

Suppose state s1 is active. If signals a, b, c, and d are present, then v is present and its value is $11550 = 2 * 3 * 5^2 * 7 * 11$. The new active state is s3. This reaction can be informally decomposed as follows:

1. Macrostate innerMacro must be weakly aborted by the transition labeled c, which has priority over the one labeled by d.
2. Before firing the transition, the inside of the macrostate must be executed. Inside, it is the strong abortion triggered by a that is taken, emitting v(3).
3. State s1 is the target, so s1 is re-entered. In fact, this is a *fresh* instance (re-incarnation) of s1.

[2] The ordering cannot be preserved by a combination function, which must be commutative!

4. This fresh instance is receptive to a strictly future occurrence of a, and to a present or future occurrence of b. Hence, the transition triggered by b is taken, and v is emitted with 5. State s2 is activated.
5. Since state s2 has no outgoing transition, no more evolution is possible in innerMacro. The transition triggered by c is then fired. v is emitted with value 7.
6. The target of the transition is macrostate innerMacro, which is re-entered. Again, it is a re-incarnation. This fresh instance is receptive to strictly future occurrences of c, and to a present or future occurrence of d. The weak abortion triggered by d is then to be taken, but before, the inside of innerMacro must react.
7. The execution of innerMacro starts with emitting v with value 2 (initial arc) and enters state s1.
8. This fresh instance of s1 is receptive to a strictly future occurrence of a, and to a present or future occurrence of b. Hence, the transition triggered by b is taken, and v is emitted with 5. State s2 is activated.
9. Since state s2 has no outgoing transition, no more evolution is possible in innerMacro. The transition triggered by d is then fired. v is emitted with value 11.
10. State s3 is activated, and the reaction stops.

Hence, v conveys the value $3 * 5 * 7 * 2 * 5 * 11$. To recapitulate, a fully explainable behavior, all but obvious.

4 Formal Presentation of SyncCharts

SyncCharts are based on states and instants. Usually, the semantics of instant-based models is given as the set of possible stimuli/reaction sequence pairs (input sequences / output sequences). This set is given implicitly by an acceptation procedure or by an inductive construction mechanism. In this paper, we choose the latter. So, we need a formal definition of the structure of a syncChart (Sec.4.1). Then, we have to characterize the *state* of a syncChart and of its environment. For this, the concepts of *configuration* and *signal context* are defined (Sec.4.2). Finally, we have to give a semantics: in Sec.4.3, we introduce an operational constructive semantics.

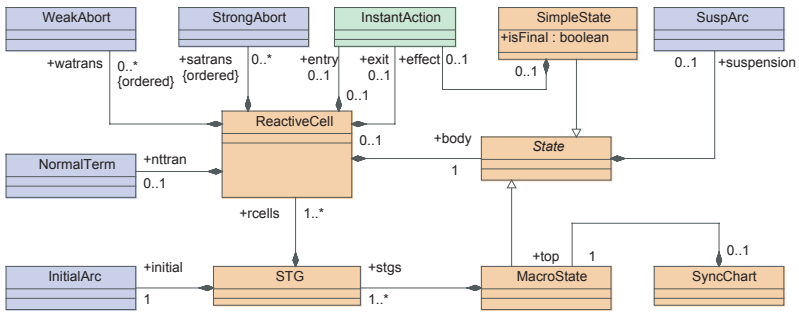


Fig. 5. SyncCharts's Metamodel: States.

4.1 Metamodel

The abstract syntax for SyncCharts is expressed using the standard UML notations. Figs.5 and 6 cover all basic concepts. A ReactiveCell consists of a body and possibly empty sets of outgoing transitions. The body is

a state: either a Macrostate or a SimpleState. satrans is the set of strong abortion transitions; watrans the set of weak abortion transitions; nttran is a the set of normal termination transitions that contains one transition at most.

Well-formed SyncCharts must also respect a few constraints:

- An STG must be a connected graph.
- A Transition links two reactive cells in the same STG.
- A SimpleState, which is final, has neither outgoing transition nor associated effect.

The structure of a syncChart can be represented by a tree that reflects the state containment hierarchy. More precisely, a macrostate contains STGs, a STG contains reactive cells, a reactive cell contains one and only one state. In the tree, a ReactiveCell has one and only one successor. It can be omitted without loss of information about the structure. The resulting tree has two types of nodes (state and STG) that alternate on any path of the tree. Fig. 7 gives our notation and the tree associated with ABSync. Note that solid circles, which represent states, have three variants: macrostate without suspension, macrostate with suspension, and simple state.

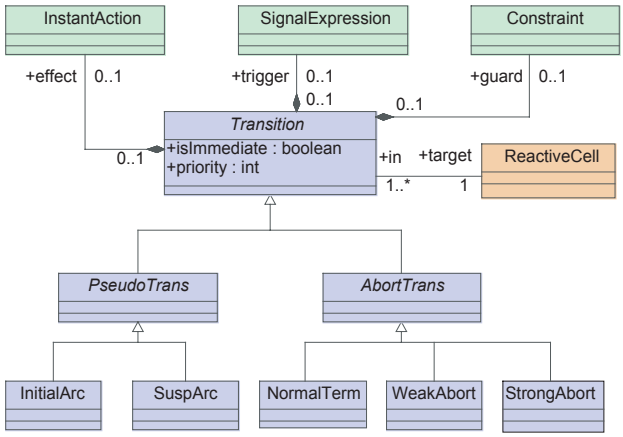


Fig. 6. SYNCCHARTS's Metamodel: Transitions.

4.2 Configuration and Signal Context

Configuration. Harel and Naamad [7] defined a configuration as a maximal set of states that could be simultaneously active. This definition must be adapted to take account of suspensions.

Let T be the top macrostate associated with a syncChart. A configuration C for T (and thus, for the syncChart) must satisfy the following rules:

1. T is in C.
2. If a macrostate without suspension M is in C, then C must also contain for each STG G directly contained in M, exactly one state directly contained in G.

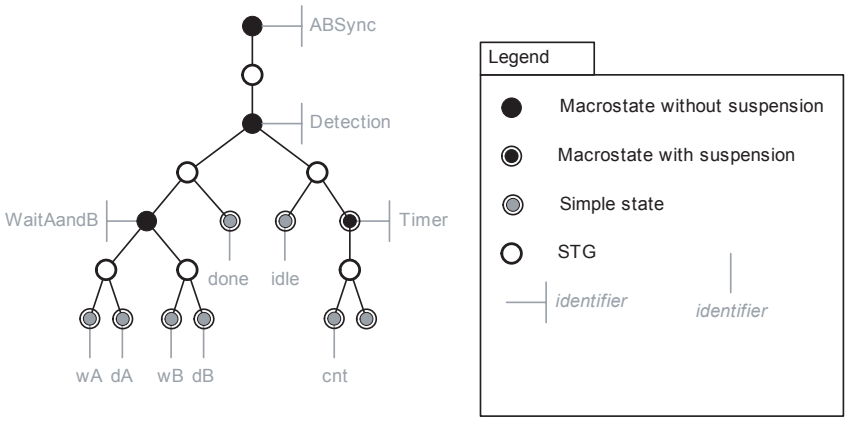


Fig. 7. Tree associated with a syncChart.

3. If a macrostate with suspension M is in C, then
   - Either C must not contain state descendant of M,
   - Or C must also contain for each STG G directly contained in M, exactly one state directly contained in G.
4. C contains only states satisfying rules 1 to 3.

Configurations can be derived from the tree associated with the syncChart. This tree has to be considered as a AND/OR tree, where AND-nodes are solid circles (states) and OR-nodes are hollow circles (STGs). {ABSync, Detector, WaitAandB, wA, wB, idle}, {ABSync, Detector, WaitAandB, wA, wB, Timer}, and {ABSync, Detector, WaitAandB, wA, wB, Timer, cnt} are instances of configurations of syncChart ABSync.

Signal Context. Since reactions in SYNCCHARTS may result from instantaneous cooperation of several subsystems, signals, which support communication, are the cornerstone in the SYNCCHARTS semantics.

With each reaction is associated a set of signals E called the *signal context*. E is partitioned into two blocks: E^+ and E^- (i.e., $E = E^+ \cup E^-$ and $E^+ \cap E^- = \emptyset$)[3]. E^+ is the set of the present signals at the current instant, E^- the set of the absent signals. During an instant, a signal must be either present or absent. When computing a reaction, only input signals, which are imposed by the environment, have a definite status. The presence status of all other signals must be determined. Like in the ESTEREL language, we assume that a non-input signal is present if and only if it is emitted during the instant.

4.3 Introduction to Constructive Semantics

Problem: Given a pure syncChart, one of its configuration, and the presence status of all its input signals, compute *the* reaction, (i.e., the next configuration, and the presence status of all output signals).

[3] In ESTEREL, the set of present input signals is called an input *event*. We avoid this term, usually associated with occurrence of a single signal.

Principle of the Computation of a Reaction. We consider the syncChart as a collection of *interacting* reactive cells. Each cell receives signals that trigger evolutions, which possibly emit new signals. All the signals are instantaneously broadcast.

Conceptually, reactive cells *run concurrently*. Each active reactive cell locally determines its behavior (i.e., performs actions, takes a transition and thus becoming idle, or staying active). The presence status of the signals is the deciding information. To ensure the *determinism* of the reaction of the syncChart, all reactive cells must agree on the *actual* presence status of each signal. This suppose finding a fixpoint solution through dialogs.

To solve this problem, we propose that each reactive cell *suspends its evolution* when in doubt about the value of a triggering signal expression. The corresponding evaluation is left pending. When a still running concurrent cell emits a signal, this *fact* about the presence status of the emitted signal is broadcast to other cells. This reliable fresh information may assert or refute a pending evaluation, and thus resume the evolution of a cell. This process is applied till stability, when each active cell has terminated its evolutions, or is suspended. What is done in the latter case depends on the chosen semantics. Some semantic variants will be described below. Beforehand, we have to precisely define the interpretation of signal expression.

Remark: This approach is akin to the one advocated by G. Berry [4] for the ESTEREL language. He named it the *constructive semantics* in reference to the fact that values are computed by explicit proofs, not by a "trial and error" procedure.

Signal Algebra. The partition of the *signal context* in present and absent signals is effective when the reaction has been successfully computed. However, during the computation itself, the presence status of a signal can be unknown. Now, E^+ is the set of the *certainly present* signals at the current instant, E^- the set of the *certainly absent* signals, and $E^\perp$ the set of signals, the status of which is not yet known. Instead of dealing with Boolean values, we resort to a Scott Boolean domain $B_\perp = \{\perp, -, +\}$ where $\perp < +$ and $\perp < -$.

The technique adopted for determining the context E is to *propagate facts* and build the solution incrementally, if one exists. The computation relies on monotonic functions. The order relation is defined by

$$E \leq E' \iff E^+ \subseteq E'^+ \wedge E^- \subseteq E'^-$$

Signals are combined in *signal expressions* used, for instance, in triggers. A signal expression is an expression similar to a Boolean expression, made of signals, operators (not , or , and), and parentheses.

Given a signal context E, a signal expression Φ evaluates in $B_\perp$:
For $\Phi ::= \sigma$ (for a signal σ) $\mid$ not $\phi \mid \phi$ or $\psi \mid \phi$ and ψ. ϕ is evaluated as follows:

$$\begin{aligned} \text{eval}(\sigma, E) &= + \text{ if } \sigma \in E^+,\ - \text{ if } \sigma \in E^-,\ \perp \text{ otherwise} \\ \text{eval}(\text{not } \phi, E) &= \text{not eval } (\phi, E) \\ \text{eval}(\phi \text{ or } \psi, E) &= \text{eval } (\phi, E) \text{ or eval } (\psi, E) \end{aligned}$$

where operators not , or , and and are defined in the tables below.

	not
$\perp$	$\perp$
-	+
+	-

or	$\perp$	-	+
$\perp$	$\perp$	$\perp$	+
-	$\perp$	-	+
+	+	+	+

and	$\perp$	-	+
$\perp$	$\perp$	-	$\perp$
-	-	-	-
+	$\perp$	-	+

Reaction of Reactive Cells. The reaction of a reactive cell relies on the reaction of its components. A react() method is defined for the different classes. This method returns a *termination status* taking values in the enumeration {DONE, DEAD, PAUSE}.

Returning DONE means that the object has terminated its reaction and has nothing left to do at the next instant. Returning PAUSE means that its reaction is over for the current instant, but will proceed at the next instant. Finally, DEAD means that the object has terminated its reaction and has nothing left to do at the next instant except waiting for a normal termination. Once entered in a final state, react() returns DEAD till the effective normal termination.

The pseudo-code for the react() methods and comments are presented in the appendix. Note that, our objective is not to make an efficient compilation of SyncCharts, but only *to explain microsteps*. Our solution heavily relies on concurrent executions, while most compilations (of ESTEREL programs) try to serialize concurrent evolutions (e.g., see S. Edwards's compiler [8], and SAXO-RT [9]).

Use of a Potential Function. If the recursive computation returns, then the syncChart is now in a new configuration, and all signals used during the reaction have been given a definite presence status. By default, all other signals are set to absent. The computation of the reaction is successful. However, the computation may stall with suspended reactive cells. For ESTEREL programs and reactive objects, F. Boussinot [10] has studied different ways to resume the computation. These ideas apply to SYNCCHARTS as well. The more drastic solution, in use in reactive objects, is to set to absent all not yet assigned signals, to resume the evaluation of pending expressions, and to defer the issue of the decisions to the next instant. SYNCCHARTS, like ESTEREL does not defer decisions. Instead, they try to enrich their knowledge about the signal context. A most interesting information is about signals that are certainly absent (cannot be emitted) during the reaction. To know whether a signal shall not be emitted seems to be relevant of clairvoyance. In fact, using the structure (syntactic analysis), we construct a monotonic decreasing set of potentially emitted signals, called the *potential*. Any signal not in this set will certainly not be emitted, provided that this set is *correct* (i.e., all signals possibly emitted *are in* the potential).

Taking account of the potential, a stalled reaction may resume, proceed by microsteps, and then stall again. This process is repeated, and eventually, either the process stops with all presence status defined, or there still exists suspended evaluation and the potential cannot help the pending evaluations. In the first case, the reaction completes successfully. In the latter case the syncChart is rejected as *non constructive*. The choice of the potential function is critical. A rough potential is easy to compute but leads to many unjustified rejections of syncCharts; a fine potential is difficult to construct but accepts a large class of SYNCCHARTS.

The commercial version of SYNCCHARTS used in Esterel Studio [11] translates the syncChart into an equivalent Esterel program, which is then compiled. Thus the potential function is the one used in the Esterel V5 compiler, based on the "must" and "cannot" sets [4]. Research on efficient potential functions exploiting the structure of the syncCharts are still in progress.

5 Conclusion and Perspectives

This paper has shown that the apparent simplicity of SYNCCHARTS may hide complex behavior. In fact, SYNCCHARTS are the graphical counterpart of ESTEREL, a textual language. Any syncChart can be translated into an equivalent ESTEREL program, and this is the usual way to compile SYNCCHARTS. For people familiar with the semantics of ESTEREL, the behavior of SYNCCHARTS is easy to understand. For instance, the amazing state re-incarnation example can be seen as a simple graphical variant of the ESTEREL signal re-incarnation.

Instead of exploiting this close relationship between SYNCCHARTS and ESTEREL, we have chosen to present SYNCCHARTS as an independent state-based synchronous model. The structure of SYNCCHARTS has been formally defined by a metamodel, using the UML notations. The computation of a reaction has been explained

in terms of cooperating *reactive cells*. The principles of the constructive semantics have been also explained, but not formalized[4].

We believe that a graphical model like SYNCCHARTS may be a good introduction to synchronous programming for many engineers more familiar with state graph models than with programming languages. However, being as expressive as the ESTEREL language, SYNCCHARTS may be too powerful for "standard" users. The re-incarnation example is a bright illustration. Moreover, even advanced users, may use only a subset of the SYNCCHARTS possibilities.

We plan to develop a platform dedicated to SYNCCHARTS that will allow the user

- to select the set of constructs he/she wants to use (customizing),
- to trace microstep execution (understanding semantics),
- to trace reactions (simulating).

Thus, the user could adapt his/her model to his/her needs. A simpler model is, of course, easier to learn, and may lead to more efficient compilations. Fig.8 is a good starting point for selecting constructs and see the possible simplifications. For instance, the *immediate* modifier is very interesting for instantaneous dialogs, but it may easily introduce *causality cycles*. Discarding *immediate* makes the first instant reaction far simpler: neither strong nor weak abortion can occur on a just-entered state. Other possibilities, not presented on the figure, concern the choice of a potential function. It is even possible to introduce variations on signal handling: In some circumstances, for example to eliminate a causality cycle, it would be convenient to differ the emission of some signals to the next instant, as it is done in Statecharts. All those variations are easily formalized and implemented. Additional constraints are added to the metamodel (syntax). As for semantics, it is sufficient to modify the react() methods.

References

1. G. Berry. The foundations of Esterel. In C. Stirling G. Plotkin and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.
2. 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.
3. D. Harel. STATECHARTS: A visual formalism for complex systems. *Science of computer programming*, 8:231–274, 1987.
4. G. Berry. *The Constructive Semantics of pure Esterel*. (revision 1999), available on the web, `www.esterel-technologies.com`, Sophia Antipolis (F), July 1999.
5. B. P. Douglass. *Real-Time Design Patterns*. Object Technology Series. Addison-Wesley, Reading, Massachusetts, 2003.
6. F. Boussinot, G. Doumenc, and J-B. Stefani. Reactive objects. *Ann. Telecommunication*, 51(9–10):459–473, 1996.
7. D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Trans. Soft. Eng. Method.*, 5(4):477–498, October 1996.
8. Stephen A. Edwards. An esterel compiler for large control-dominated systems. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21(2):169–183, 2002.
9. E. Closse, M. Poize, J. Pulou, P. Venier, and D. Weil. SAXO-RT: Interpreting ESTEREL semantics on a sequential execution structure. In *Electronic Notes in Theoretical Computer Science*, volume 65 (5), Grenoble (F), 2002. Slap'2002, Synchronous Languages, Applications and Programming.
10. F. Boussinot. Sugarcubes implementation of causality. Technical Report 3487, INRIA, September 1998.
11. Esterel Technologies, Guyancourt (F), `//www.esterel-technologies.com`. *Esterel Studio, V4*, 2002. Reference Manual.
12. Charles André. Syncharts: a visual representation of reactive behaviors. Technical Report RR 95–52, rev. RR (96–56), I3S, Sophia-Antipolis, France, Rev. April 1996.

[4] There exists a technical report [12] about a behavioral semantics of SYNCCHARTS (early version).

6 Appendix: Computation of a Reaction (Details)

6.1 SyncChart Reaction

```
SyncChart::react()
   read inputs
   for all s in outputs do
     s.reset()
   done
   top.react()
```

The `reset()` method, when applied to a signal, sets its presence status to $\perp$.

6.2 MacroState Reaction

The constants of the enumeration that type the *termination status* are ordered: `DEAD` $<$ `PAUSE`. This order is used to characterize the return of parallel executions of STGs in a MacroState. The `react` method applied to a macrostate returns either `DEAD` or `PAUSE`.

```
MacroState::react()
   for all s in locals do
     s.reset()
   done
   parallel for all g in stgs do
     r[k] = g.react()
   done
   return Max(r)
```

6.3 STG Reaction

```
STG::react()
   if curCell==null then
     curCell = initialCell
   endif
   while (r=curCell.react()) == DONE do
     curCell = nextCell
   done
   return r
```

The `react()` method of an STG calls the `react()` method of its current active reactive cell. This method returns either `DEAD` or `PAUSE`. Note that several reactive cells can be passed through, in a strictly sequential order, during the one instant.

6.4 SimpleState Reaction

This is a trivial case.

```
SimpleState::react()
   if isFinal then
     return DEAD
   else
     perform effect
     return PAUSE
   endif
```

6.5 ReactiveCell Reaction

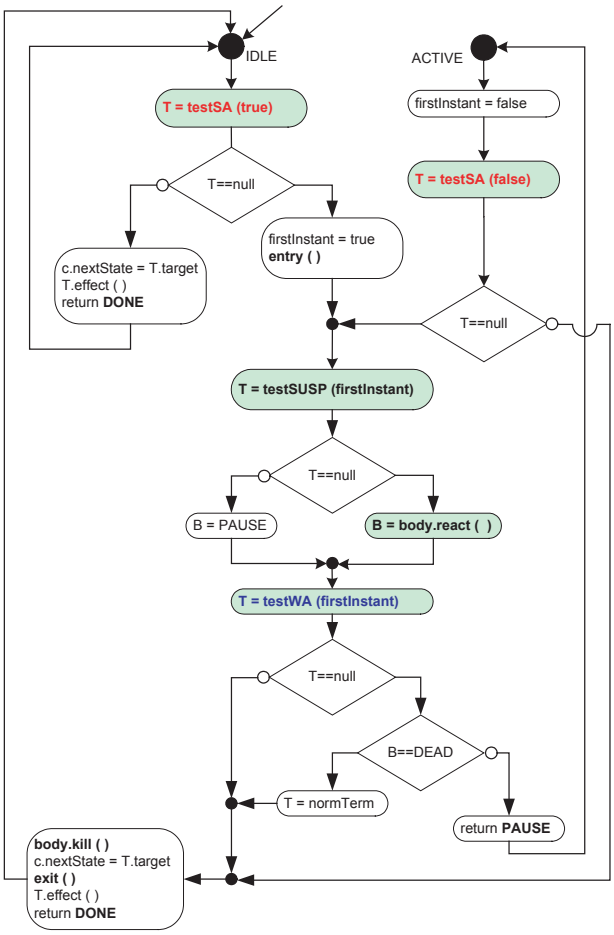


Fig. 8. ReactiveCell Reaction: `ReactiveCell::react()` method.

For the react() method of a ReactiveCell, see Fig. 8. Note that the behavior is different at the first instant and at the following instants. The kill() method makes recursive depth-first kills: ReactiveCells enter their IDLE state, and STGs forget their current ReactiveCell.

The heart of the computation is the testP (test presence) function that evaluates a signal expression. This function is executed by a thread that waits until the signal expression evaluates to either + or -.

```
boolean testP(expr,E)
   wait (r=eval(expr,E)) != ⊥
   return (r == +)
```

This function is used in trigger evolutions (capsules with colored background in Fig.8). Given an ordered set of transitions S and a signal context E, testTrans returns the first firable transition (i.e., the first transition whose trigger evaluates to true), if any, or null, otherwise.

```
Transition testTrans(S,E)
   for (t:Transition=S'first to S'last) do
     if testP(t.trigger,E) then
       return t
     endif
   endfor
   return null
```

Function testSA() (testWA(), testSUSP(), respectively) used in ReactiveCell :: react() is a special case of testTrans, where parameter S is the set of the strong abortion (weak abortion, suspension, respectively) transitions. These functions have a Boolean argument to indicate whether the evaluation concerns the first instant or not. At the first instant, all not immediate transitions are ignored.