Semantics of SyncCharts

Charles André
I3S Laboratory – UMR 6070
University of Nice-Sophia Antipolis / CNRS
BP 121
F 06903 Sophia Antipolis cédex

andre@unice.fr

April 2003

This report has been written for Esterel-Technologies
(Available at //www.esterel-technologies.com,
Download>Scientific Papers>Semantics of S.S.M)

1 Table of Contents

2 List of Figures

3 Introduction

SyncCharts are a visual synchronous model. They were conceived in the mid nineties [André 1996a] as a graphical notation for the Esterel language [Boussinot and De Simone 1991, Berry 2000]. As such, SyncCharts were given a mathematical semantics fully compatible with the Esterel semantics. A technical report [André 1996b] explained this semantics. This is a valuable document for people familiar with formal semantics but may be difficult to read for most potential users. Since SyncCharts were also devised as a graphical model, akin to finite state machine, intended for engineers, an informal presentation of the model and its semantics was missing. This paper is an attempt to fill this need.

Like Esterel, SyncCharts are devoted to *programming control-dominated* software or hardware *systems*. These systems are *reactive*, that is, they continuously react to stimuli coming from their environment by sending back other stimuli. They are purely *input-driven*: they react at a pace imposed by their environment. A reactive application usually performs both data handling and control handling. Esterel and SyncCharts are imperative languages especially well-equipped to deal with *control-handling*: they produce discrete output signals in reaction to incoming signals. At a given instant, a signal is characterized by its *presence status*. Besides its presence status, a signal may convey a value of a given type. Such a signal is called a *valued signal*. A signal that conveys no other information than its presence is called a *pure signal*.

This paper mostly focuses on *Pure SyncCharts*, which are restricted to pure signaling. Pure SyncCharts are enough to explain most typical reactions that are easily expressed by graphical notations. Since a syncChart[1] may include any "in-line" Esterel code, a comprehensive presentation of the semantics of SyncCharts should include a presentation of the Esterel semantics. This is definitely beyond the scope of this paper. Interested readers should refer to two papers written by Gérard BERRY: "The Primer" for the Esterel language [Berry 1997], which presents the language and its semantics in a precise but informal way, and "The Constructive Semantics of Pure Esterel" [Berry 1999], which presents the reference semantic framework for the language. Note that this presentation is also restricted to the "pure" subset of the language.

While Esterel adopts a textual form to express the control, SyncCharts rely on a graphical representation made of a hierarchy of communicating and concurrent finite state machines (FSMs). Our intuitive presentation of the semantics of SyncCharts will explain why, given a current configuration (a set of active states) and a stimulus (a set of input signals), a syncChart changes its configuration and generates output signals. Since SyncCharts are deterministic, the new configuration and the set of emitted signals are perfectly defined for any correct syncChart.
The observed reactions result from instantaneous interactions among finite state machines. With simple examples, progressively enriched, we will introduce structural elements of SyncCharts and explain their interactions. Not surprisingly, our informal but precise descriptions of the behavior are visual representations.

[1] "SyncCharts" is the model. A "syncChart" is a particular instance of this model.

Organization of the paper

- After this introduction, a chapter (Section 4) introduces the SyncCharts model and the synchronous hypotheses. The basic concepts of signals, state, and transition are illustrated with simple examples. In this chapter SyncCharts are seen as another variant of Finite State Machines.
- The next chapter (Section 5) explains why SyncCharts are much more than Finite State Machines: they support hierarchy, concurrency, and preemption. The various kinds of preemption are introduced, they can be combined with hierarchy and concurrency, while preserving deterministic evolutions.
- After these informal presentations of the SyncCharts and their behavior, a chapter (Section 6) deals with an operational semantics of SyncCharts. The syntax of the model is precisely defined and a way to compute a reaction of a syncChart is given. This computation relies on the structure.
- Synchronous reactions, which allow emitted signals to participate to the reaction itself, may result in paradoxical or even incorrect behavior. Section 7 explains such an erroneous behavior known as a "causality cycle".
- Advanced constructs of SyncCharts are presented in the next chapter (Section 8). The first two constructs capture powerful concepts rarely supported in state-based models. The first one is the "immediate" transition: transition firings can be explicitly chained during a reaction, so that transient states can be compiled out. The second one is the "suspension", a temporary form of preemption, useful to freeze evolutions of parts of the model. Instantaneous actions to perform when entering or leaving a state are other model extensions. The computation of a reaction is then revisited to integrate these model enhancements. This chapter ends with a short introduction to valued SyncCharts, a presentation of the pre operator, and finally two illustrations of signal and state reincarnations.

Most explanations are precise and yet easy to understand. Some points are not so simple. They are pointed out by the  symbol. The reader may skip them for a first reading. A few points need deep insight in the model semantics or describe reactions at a very fine grain. They are indicated by the  symbol. They should be reserved for a second reading.

4 A First Look at SyncCharts

4.1 Abstract

In this section we introduce the SyncCharts model and the synchronous hypotheses.
The main concepts are signal, state, and transition. In this first approach, only "flat" SyncCharts are considered. They can be seen as a variant of Finite State Machines. Their behavior is represented by executions traces (for particular evolutions) or by Mealy machines.

4.2 Reaction of a SyncChart

In the synchronous approach, signals are the unique abstraction for modeling information exchange between the reactive system and its environment. The signals sent by the environment to the reactive system are called *input signals*; the signals generated by the reactive system are called *output signals* (Figure 4-1). In a control application, input signals are often associated with sensors, while output signals are associated with actuators. The input and output signals define the interface of the reactive system. The black-box view of the reactive system consists of a box, incoming arrows that represent input signals, and outgoing arrows for outgoing arrows. A syncChart describes the behavior of a reactive system, that is, how sequences of output signals are related to sequences of input signals.

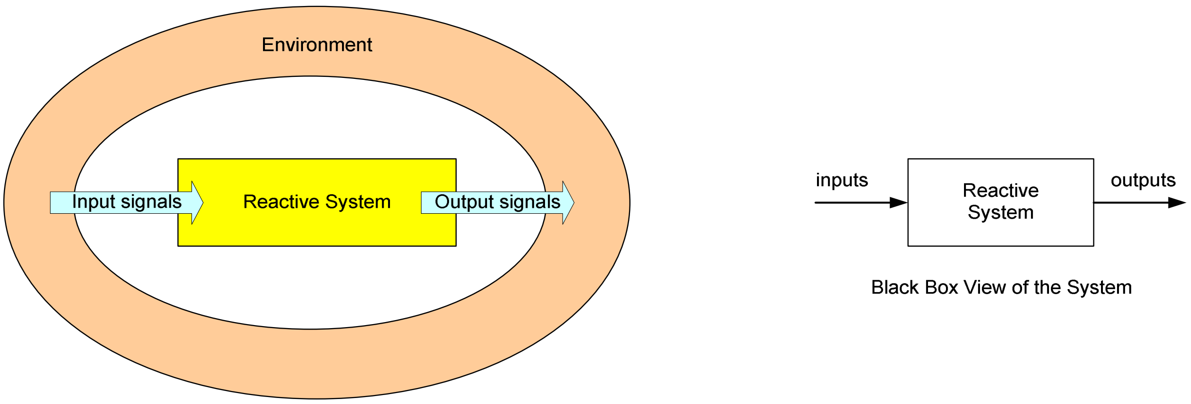


Figure 4-1: Input and output signals.

A second hypothesis of the synchronous approach is that the reactive system evolves by successive reactions taking place at discrete *instants*. This results in a *cyclic evolution* model (Figure 4-2). A reaction consists of three phases:

1. Reading input signals
2. Computing the reaction
3. Performing outputs.

The first phase collects the presence status and the possible value of each input signal. The second phase computes the reaction (i.e., the next internal state of the syncChart, and the presence status and the value, if any, of each output signal). The third phase issues output signals to the environment. The set of all present input signals has been called the *input event* in Esterel. We adopt this term though the reader must keep in mind that an event is set of signals instead of a simple change-of-state of some condition (meaning given in Petri nets, UML...). Of course, an *output event* is a set of emitted signals.

In order to satisfy the strict synchronous hypothesis, which assumes that a *reaction is instantaneous* (0-duration), these three phases are supposed to be executed on a hypothetical infinitely fast machine. This machine acts as a transformer of input histories to output histories.

This execution model will refined later. It is sufficient to explain the behavior of the simplest SyncCharts that are simple finite state machines.

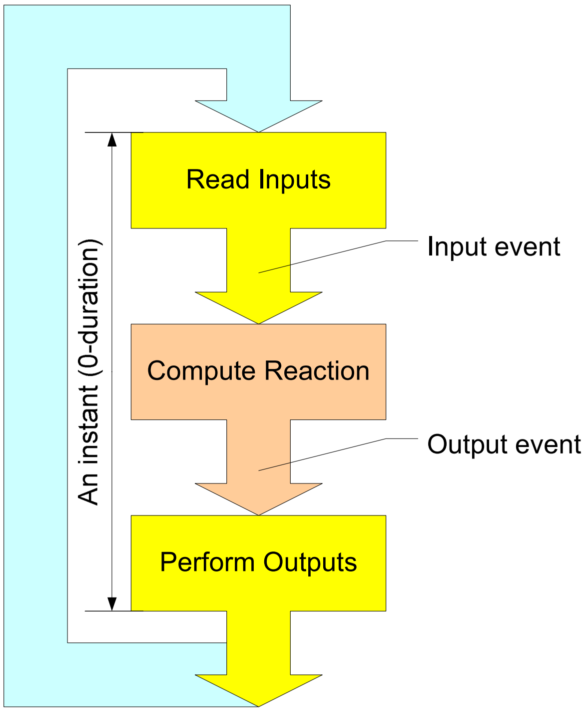


Figure 4-2: Cyclic evolution.

4.3 Finite State Machine

Finite State Machines (FSMs) are widely used in many domains, with possible different interpretations. A FSM is made of states and transitions. When used in control applications, a FSM represents the expected behavior of the system.

Interpretations may differ on

- how to trigger a transition,
- when leaving a state,
- when entering a state,
- when performing actions (effects) associated with a transition,
- when performing actions associated with a state,
- …

SyncCharts will give a precise answer to all these questions.

“A finite state machine (FSM) 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”

[Douglass 2003, chap.1]. This definition given by B.P DOUGLASS 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 a *transition* to) other states based on the signals it accepts.

4.4 FSM with outputs associated with transitions

Consider a simple "frequency divider", that is, a system that waits for a first occurrence of a signal **T,** and then emits a signal **C** at every other occurrence of **T**. This behavior can be represented by the syncChart in Figure 4-3.

4.4.1 Model

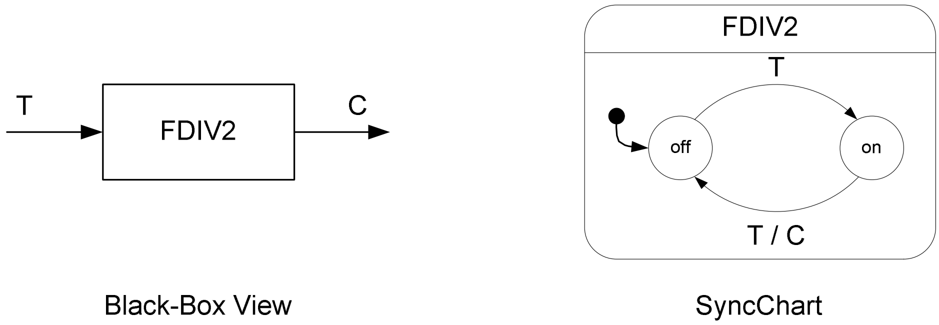


Figure 4-3: A Simple Frequency Divider.

Graphically, a state is drawn as a circle (or an ellipse). An optional identifier, written inside the state, may be given to a state. SyncChart **FDIV2** has two states named **off** and **on**. The ways to exit a state and to enter another one are represented by transitions from the source state to the target state. The label associated with the transition indicates the trigger and the effect, according to the following syntax: *trigger / effect*. The simplest trigger is a single triggering signal. Complex triggers consist of several signals combined with the and, or, and not operators. With Pure SyncCharts, effects are restricted to signal emissions. The trigger and the effect are optional, the interpretation of a trigger-less transitions will be given later (See Section 5.4.4).

- Transition from state **off** to state **on** is triggered by an occurrence of signal **T**.
- Transition from state **on** to state **off** is triggered by an occurrence of signal **T** and signal **C** is emitted while the transition is taken.

SyncCharts being a deterministic model, a state must be selected as the *initial state*. The initial state is denoted by a arrow pointing to the state. State **off** is the initial state of syncChart **FDIV2**.

4.4.2 Behavior

A simple trigger is said to be *satisfied* when the associated signal is present. The satisfaction of a complex trigger is computed by giving to the and, or, and not operators their usual meaning. For instance, not **S**, where **S** is a signal, is satisfied if and only if **S** is absent; for **S** and **T** two signals, **S** and **T** is satisfied if and only if **S** is present and **T** is present.

When a state is entered (*activation* of the state) the outgoing transition is not immediately checked: only a strictly future satisfaction of the trigger can enable the transition. Stated in other words: As soon as a state is activated, this state waits for a *strictly future satisfaction* of the trigger of its outgoing transition. When the trigger is satisfied, the transition is said to be *enabled*. The transition is immediately taken and emits associated signals, if any. The *firing* of a transition takes no time.

The behavior of the system can be represented by *execution traces*. An execution trace is a record of successive reactions, indexed by natural numbers. Each reaction is characterized by an input event and an output event. Table 4-1 contains an execution trace for **FDIV2**.

Notation:

With Pure SyncCharts, it is sufficient to mention present signals. When the set is a singleton, the curly braces are omitted (i.e., **T** stands for {**T**}, and is interpreted as signal **T** is present).

Instant	Input	Output
1		
2	**T**	
3		
4	**T**	**C**
5		
6	**T**	
7	**T**	**C**
8	**T**	
9		

Table 4-1: An execution trace for FDIV2.

4.5 Associating outputs with states

A machine that associates outputs with transitions is known as a "Mealy Machine". Sometimes, it may be interesting to know the current state of the syncChart. This can be done by associating output signals with state (Moore Machine).

4.5.1 Example: Toggle Flip-Flop

We modify the previous example by adding two new output signals: **OFF** and **ON**. **OFF** is emitted when in state **off**, whereas **ON** is emitted when in state **on** (see Figure 4-4). The new system is known as a "Toggle Flip-Flop" (T Flip-Flop). "A Toggle Flip-Flop has a single input that causes the stored state to be complemented when the input is asserted" [Katz 1995].

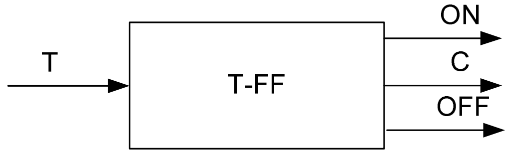


Figure 4-4: Toggle Flip-Flop—Black-Bow view.

In SyncCharts, signals associated with a state are denoted by a label attached to the state. The syntax is / *effect*. In Pure SyncCharts, *effect* is a set of signals.

Now, we are faced with the task of deciding when precisely the output signals must be emitted. There are three different cases to analyze: when *entering* a state, when *in* a state, when *exiting* a state. For now, we consider *entering*, *in*, and *exiting* as exclusive. That is, at an instant, a state is *in* if and only if it is active, it has been entered in a previous instant (not entering), it will stay active (not exiting).

4.5.2 Strong and Weak Abortion Transitions

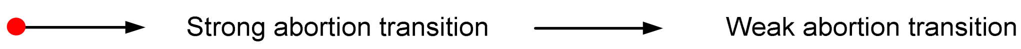
Obviously, when *in* a state, the associated output signals must be emitted. SyncCharts have two types of transitions (*strong abortion* transitions and *weak abortion* transitions) specifying different behaviors. Table 4-2 defines the behavior. The third case is given for information. It is the behavior observed for circuits running in the clocked (synchronous) mode, an usual mode for sequential circuits (see [Katz 1995]).

	entering	*in*	*exiting*
Weak abortion	Yes	Yes	**Yes**
Strong abortion	Yes	Yes	**No**
(clocked mode)	No	Yes	Yes

Table 4-2: Emitting output signals associated with states.

From Table 4-2 we deduce that synchronous models are “faster” than classical models: they perform actions associated with the target state of a transition at the very instant when the transition is taken. Weak and strong abortion transitions differ only on what is done when exiting a state. Weak abortion performs actions associated with the exited state, while strong abortion does not. A more general characterization of abortions will be given later, after introducing hierarchy in SyncCharts.

Notation



SyncCharts

The behavior of the Toggle Flip-Flop is specified in Figure 4-5. Two versions using strong and weak abortions are presented. The former is called **Tsa** (Toggle strong abort), the latter **Twa** (Toggle weak abort).

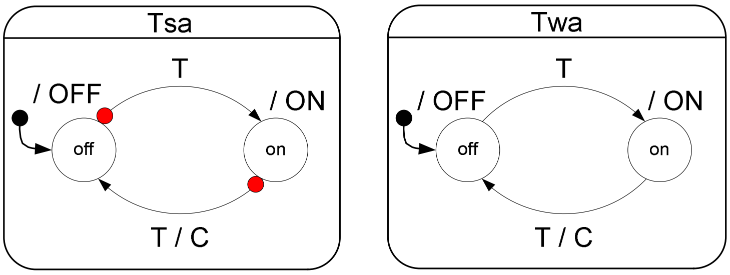


Figure 4-5: SyncCharts for the Toggle Flip-Flop—Strong and weak abortion versions.

Behavior

Table 4-3 contains an execution trace for the two versions of the Toggle Flip-Flop. Signals emitted when exiting a state by weak abortion are written in red letters.

Instant	Input	Output	
		Tsa	Twa
1		**OFF**	**OFF**
2	**T**	**ON**	**OFF,ON**
3		**ON**	**ON**
4	**T**	**C,OFF**	**C,OFF,ON**
5		**OFF**	**OFF**
6	**T**	**ON**	**OFF,ON**
7	**T**	**C,OFF**	**C,OFF,ON**
8	**T**	**ON**	**OFF,ON**
9		**ON**	**ON**

Table 4-3: An execution trace for the Toggle Flip-Flop.

4.6 Alternative Representation to Execution Traces

The representation of execution traces given in Table 4-3 makes no direct reference to internal states. In order to explain how an ssm works, explicit references to states are needed. Using the notations introduced in Figure 4-6 makes it possible.

Figure 4-7 contains the first five instants of the execution trace already presented in Table 4-3 for **Tsa**. The advantage of this representation is to show the active state of the ssm and its dynamic evolutions. A more concise representation (Figure 4-8) only mentions the active state but loses the information about the control path (transitions through which the control passed).

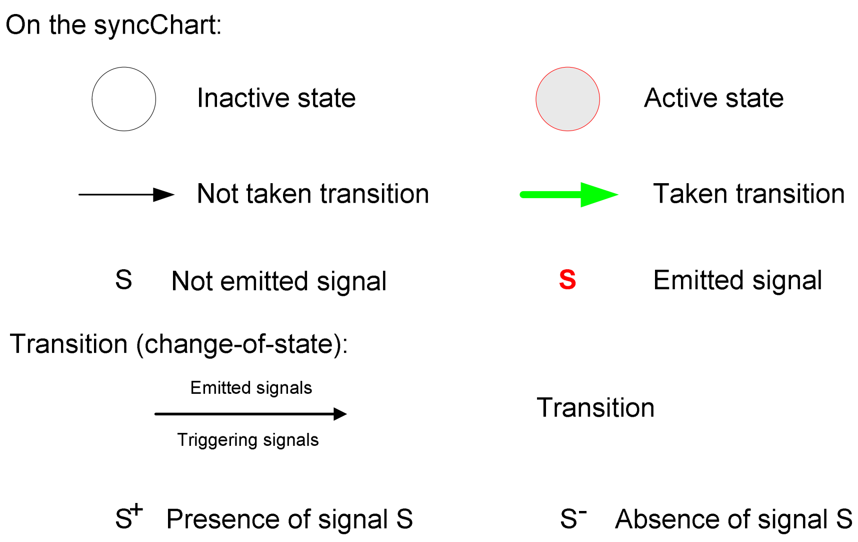


Figure 4-6: Notations.

Executions traces, in all the previously described forms, represent only particular behaviors. There are very useful to *understand* the system behavior. In order to represent all possible behaviors, we need some "trace folding" technique. For finite state models, FSMs can do that. The Mealy machine (Figure 4-9) is equivalent to syncChart **Tsa**. The reader may wonder why to introduce a new model, namely the SyncCharts, if we have recourse to well-known FSMs or Mealy machines. The answer to this question will be given later.

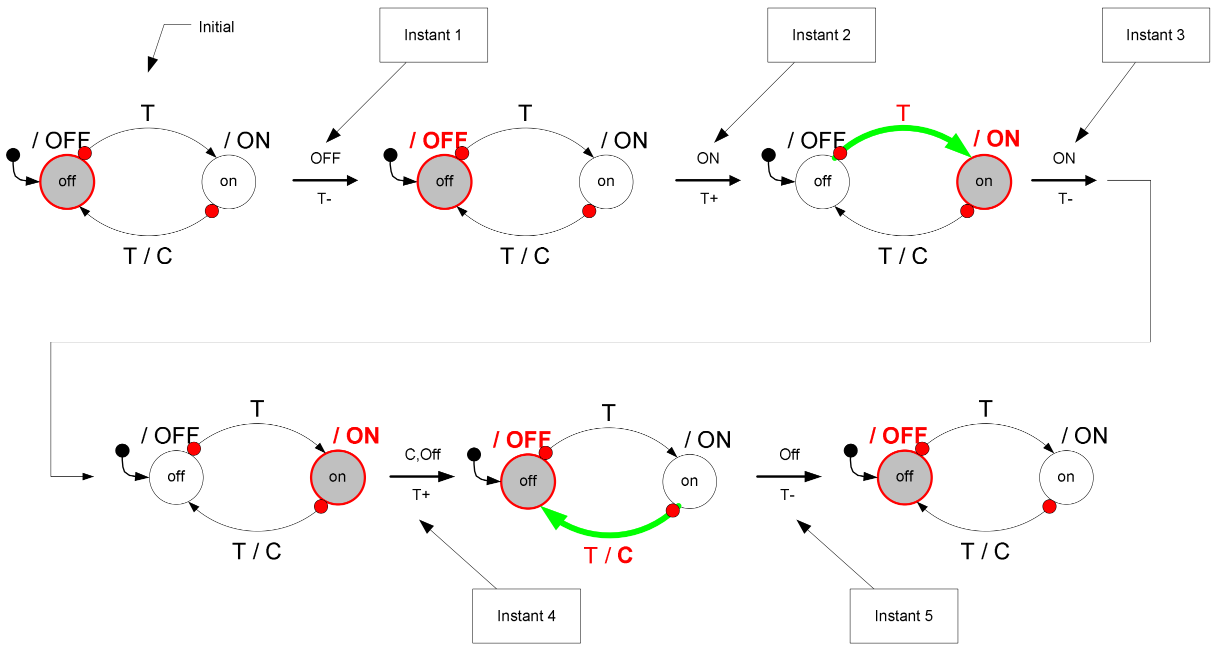


Figure 4-7: An Execution Trace for Tsa.

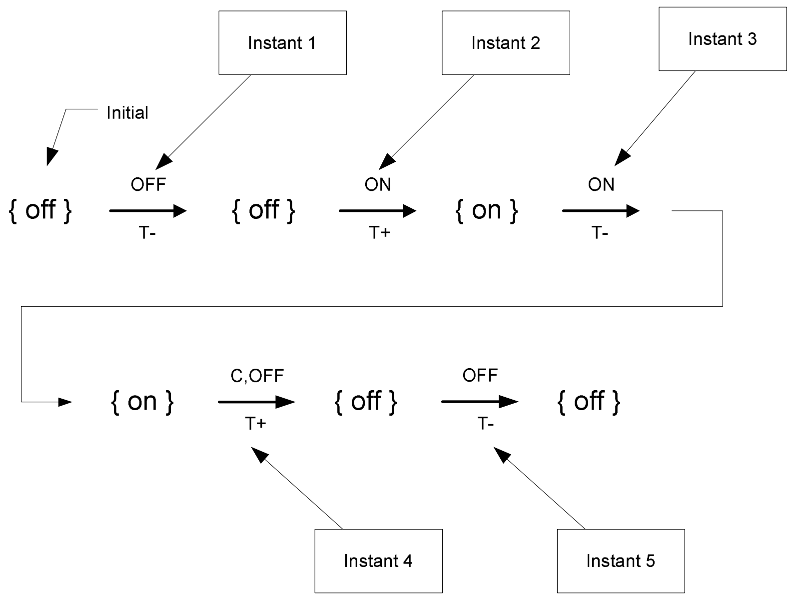


Figure 4-8: An Execution Trace for Tsa—Concise form.

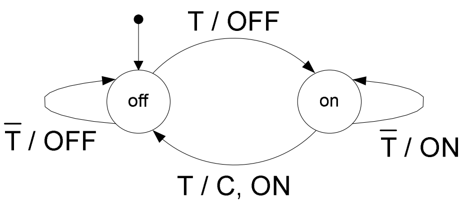


Figure 4-9: Mealy machine with the same input-output behavior as Tsa.

4.7 FSM with Choice: Introducing Priority

4.7.1 An Example of Resource Management

This system allows two users to access a resource, while ensuring exclusive access to the resource. The access controller (**ResMgr**) consists of

- Two user's dialog controller (**UCtrl1** and **UCtrl2**)
- And an arbitration controller (**Arbiter**).

More details about this application are given in Annex 2.

4.7.2 User's Dialog Controller

Figure 10-6 shows the interface (left side) and the syncChart of the user's dialog controller (**UCtrl**). The syncChart uses both strong and weak abortions. This example is almost as simple as the T flip-flop. An equivalent Mealy machine (Figure 4-11) can be easily proposed. Just notice that the transition from state **Wg** to state **Busy**, caused by a weak abortion, emits both **Rq** and **Rn**. Signal **Rq** stands for "request" and is associated with state **Wg** (Waiting for Grant). Signal **Rn** stands for "running" and is associated with state **Busy**.

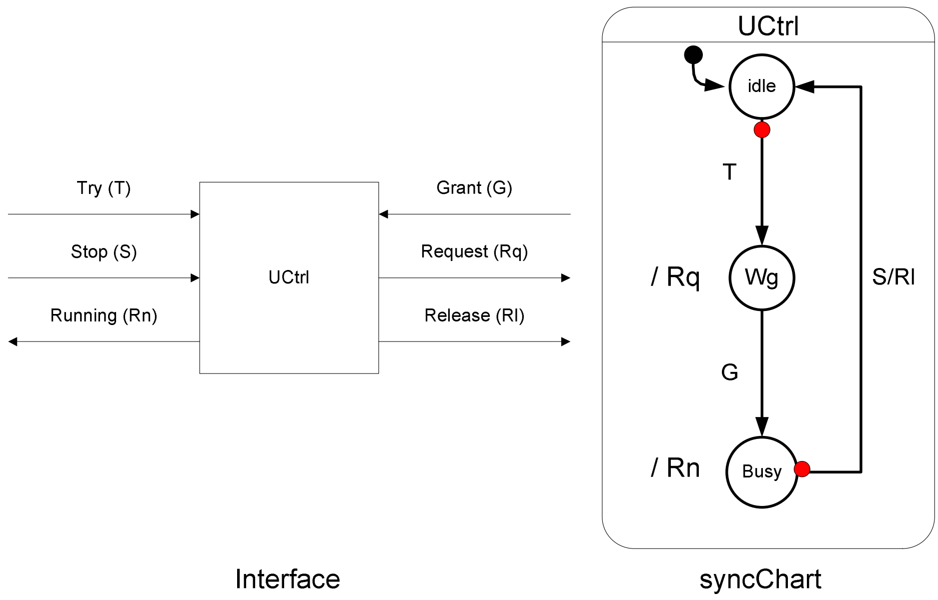


Figure 4-10: User's Dialog Controller.

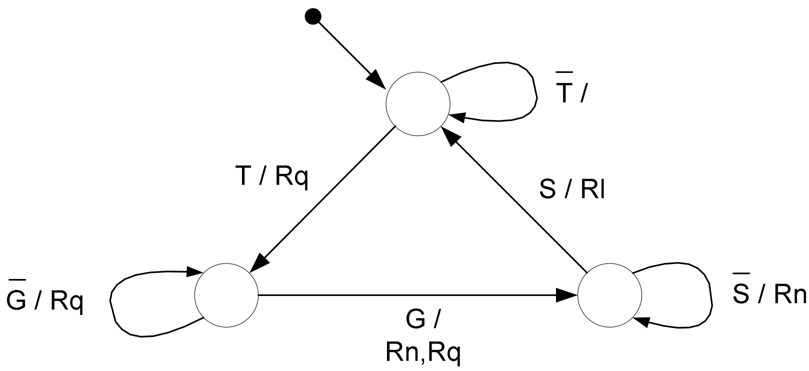


Figure 4-11:Mealy machine equivalent to UCtrl.

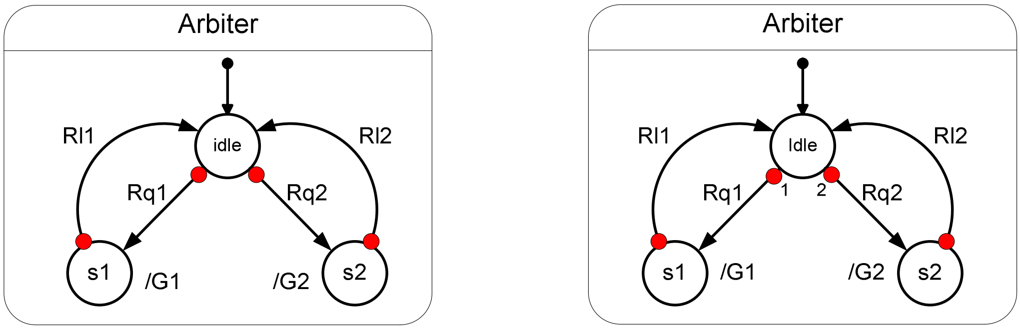
4.7.3 Arbitration Controller

Up to now, each state had at most one outgoing transition. Usually, there exist several ways to exit a state, and therefore, several outgoing transitions. Since SyncCharts are *deterministic* models, we have to resolve a choice when several transitions are simultaneously enabled , and therefore, candidate for firing.

Consider the simple syncChart that expresses the behavior of the arbiter in the Resource Manager application (Figure 4-12).

The external signals of the **Arbiter** and their interpretation are:

input **Rq1**; (User1 requests the resource)
input **Rl1**; (User1 releases the resource)
output **G1**; (Arbiter grants the resource to User1)
input **Rq2**; (User2 requests the resource)
input **Rl2**; (User2 releases the resource)
output **G2**; (Arbiter grants the resource to User2)



Illegal syncChart Legal syncChart

Figure 4-12: SyncChart with Choice.

The syncChart on the left is incorrect because when in state **Idle**, if **Rq1** and **Rq2** get present at the very same instant, both outgoing transitions can be taken, but only one is actually taken. This results in a non deterministic choice. A correct syncChart avoids this situation by imposing a *deterministic choice*. A *priority* attached to each outgoing transition (an integer number written by the origin of the transition) resolves the potential conflict (decreasing priority for increasing number).

The previous syncChart with priority given to **Rq1** over **Rq2** behaves like the Mealy machine below (Figure 4-13).

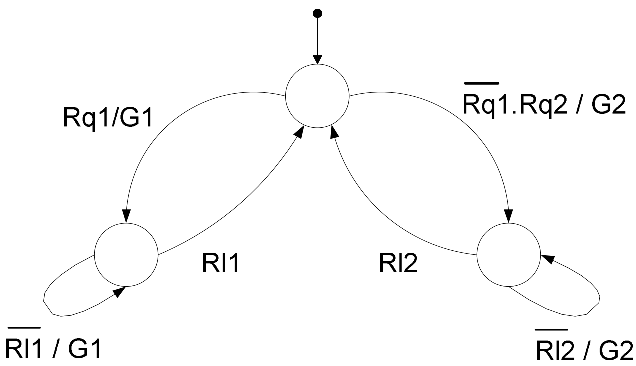


Figure 4-13: Mealy machine for Arbiter.

4.8 Summary Statement of SyncCharts and FSM

In its simplest form, a syncChart is a variant of the FSM model. At each instant there is one and only one *active* state. The *initial state* is the first activated state. Transitions between a *source* state to a *target* state are of two kinds: *weak abortion* transitions and *strong abortion* transitions.

Labels are optionally attached to transitions and states. A transition label has two optional fields: a *trigger*, and an *effect*. A trigger may be a single signal or a combination of signals using the and, or, and not operators. An effect may be a single signal or a set of signals. A state label has only an effect field, which is a set of signals.

A distinct static *priority* is attached to each outgoing transition of a state. Figure 4-14 sums up the various notations.

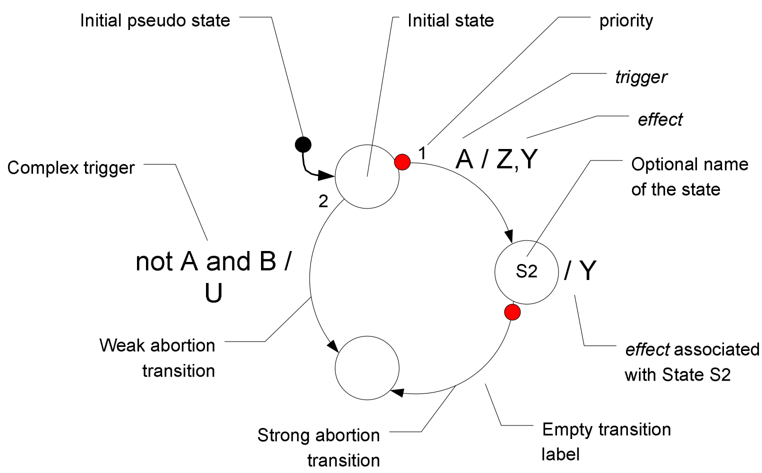


Figure 4-14: FSM notations

An *active* state waits for the *satisfaction of the trigger* of one of its outgoing transition, at an instant strictly posterior to its entering (activation). The satisfaction of a trigger enables the associated transition. An *enabled transition* must be immediately taken.
The change-of-state, caused by the *firing* of a transition is *fully deterministic*. It takes no time. Signals may be emitted as a consequence of the transition firing. Whether a signal associated with state is to be emitted when exiting the state depends on the kind of transition: only weak abortion permits emission.

Execution traces, possibly showing *active* states, can be used to represent particular behaviors of a syncChart.

As we will see in the next section, SyncCharts are generally made of several FSMs. These machines have concurrent evolutions, and moreover they may be nested. Their behavior will greatly differ from usual FSM behavior—behavior upon which users not always agree. So, we prefer to use another term: *State Transition Graph* (STG) to designate connected labeled graphs made of states connected by transitions, with an initial state.

5 Hierarchy, Concurrency, Preemption

5.1 Abstract

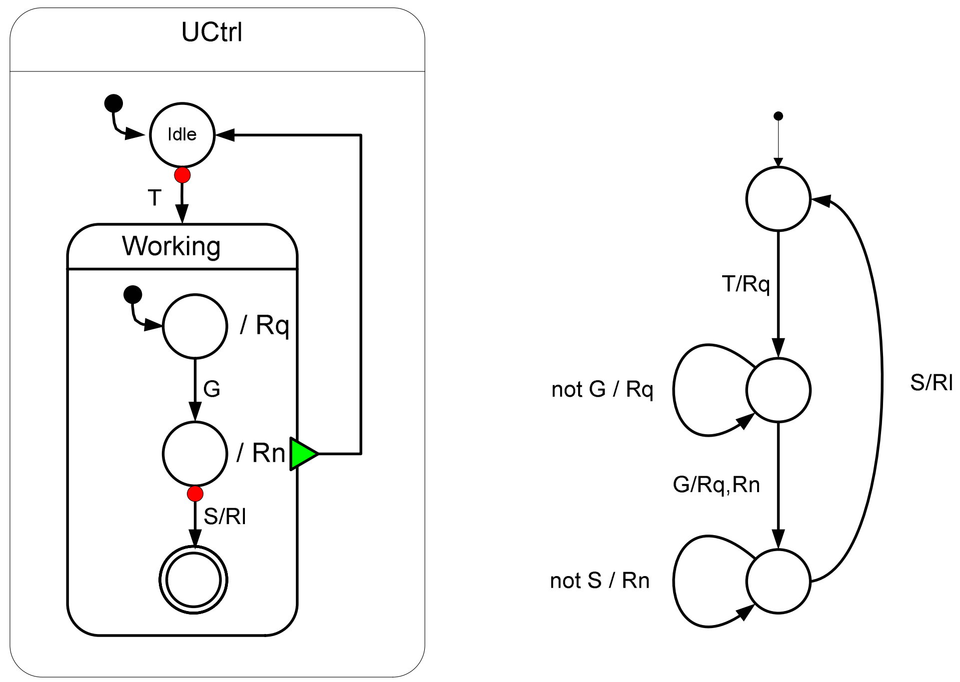
SyncCharts are more than FSMs. They support hierarchy, concurrency, and preemption. This section shows how to model hierarchy (macrostate), concurrency (concurrent STGs), and preemptions (strong and weak abortion). A reaction is explained in terms of microsteps. A striking feature of synchronous models is that evolutions are still deterministic even when concurrency, communication, and preemption are mingled.

5.2 Hierarchy

Hierarchy can be seen as a grouping facility, or as a support for refinement.

5.2.1 Hierarchy seen as state grouping

The User's Dialog Controller can be either **Idle**, or **Working**. The latter status corresponds to either sustaining a request while waiting for **Grant**, or being running and waiting for **S**. This is captured by the notion of *macrostate*. A macrostate is a state that contains one (or several) state transition graph(s). In contrast, a classical state, which is not refined, will be called a *simple-state*.

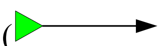


Macrostate with normal termination

Equivalent Mealy machine

Figure 5-1: Macrostate as state grouping.

In the macrostate named **Working** (Figure 5-1), there exists a special state (with a double outline), called a *final state*. When entering this final state, the control is instantaneously passed through a normal termination transition to the **Idle** state. Thus, the behavior is the same as the "flat" model previously studied (Figure 4-10).

The tail of a normal termination transition is a small green triangle ().

This way of leaving a macrostate, without an explicit triggering event is called a *normal termination*.

The use of final states and normal termination is more interesting in the presence of concurrency. This will be illustrated after introducing concurrent evolutions.

Remark: Macrostates can be nested at any depth. Showing too deep a hierarchy in a syncChart may hamper readability and understanding. Fortunately, there exists a modularity notion (*reference macrostates* presented in Section 8) that allows better organization of deep hierarchy.

5.3 Concurrency

5.3.1 Example of a Binary Counter

In hardware, starting with 2 T flip-flops, a 2-bit binary counter is easily obtained by cascading the two flip-flops: the carry output of the first flip-flop is connected to the triggering input of the second flip-flop. The diagram structure (Figure 5-2) explicitly shows these connections.

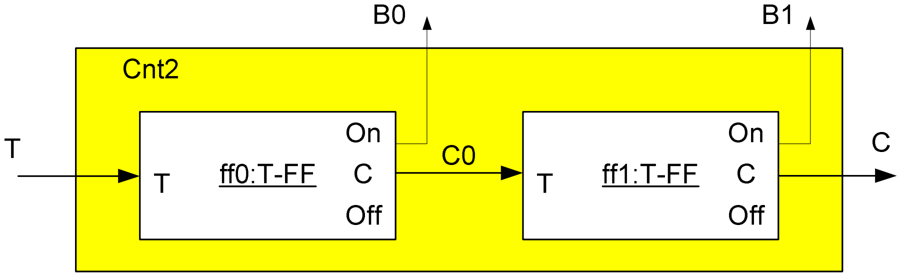


Figure 5-2: A 2-bit binary counter.

The syncChart for the 2-bit binary counter named **Cnt2** is obtained by a *parallel composition* of two syncCharts for T flip-flop (Figure 5-3). Dashed lines are used to separate concurrent STGs contained in a macrostate. STGs are coupled thanks to shared signals: an STG may emit the *local signal* **C0**, which is a triggering signal for the other STG. A local signal is declared with the keyword signal, and its scope is the containing macrostate.

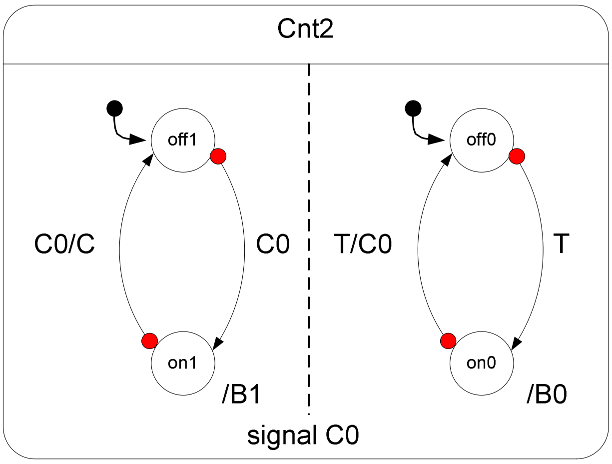


Figure 5-3: SyncChart for a 2-bit binary counter.

5.3.2 Behavior of Cnt2

The interface of **Cnt2** is:

input **T**;
output **B0**, **B1**, **C**;

Consider the input sequence **T-;T+;T+;T+;T+**. The associated execution trace is detailed in Figure 5-4. The first two steps involve only one STG, so the reactions are similar to the ones studied in Section 2.

Reaction 3 is more complex: two STGs are concerned. In the syncChart two states are active at the same time (one per STG). The "internal state" of the syncChart is no longer defined by one active state, but by a set of active states, instead. A set of (concurrent) active state is called a *configuration*. This word is the one used in Statechart semantics [Harel and Naamad 1996]. A more formal presentation will be given in Section 6. The configuration of **Cnt2** is {**off1, on0**}. Since **T** is present, the transition from **on0** to **off0** is taken. As a result **C0** is emitted (effect associated with the transition) and **B0** is not emitted (strong abortion of state **on0**). Now, **C0** being the trigger of the transition from **off1** to **on1**, this transition is taken and state **on1** is entered, causing emission of **B1**. The reaction has been computed as a sequence of *microsteps*, all executed during the same instant, but in an order that respects causality (the cause precedes the effect). An external observer sees the reaction as a whole: **Cnt2** instantaneously passes from the configuration {**off1, on0**} to the configuration {**on1, off0**} while emitting **B1**. Of course, **C0**, which is a local signal, is not visible to the outside. Figure 5-5 shows the microsteps that compose the third reaction. Reaction 5 is also a reaction that results from a sequence of microsteps.

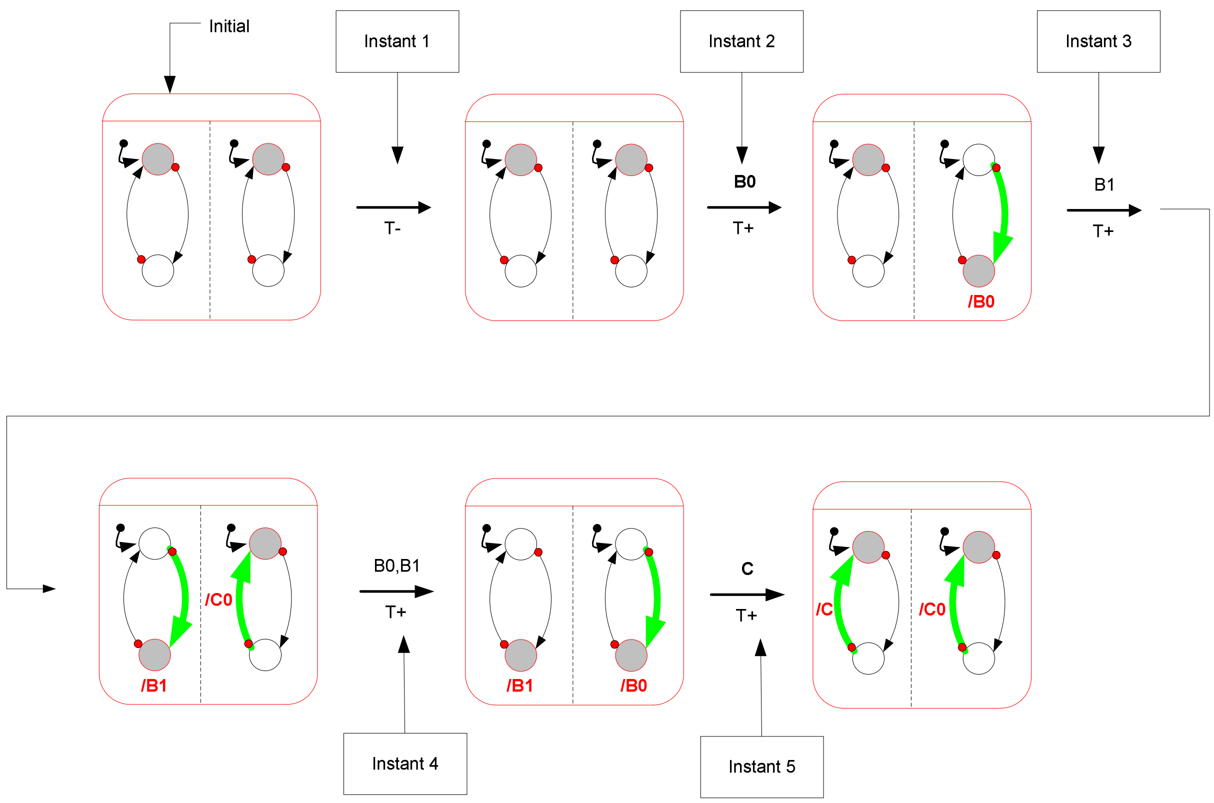


Figure 5-4: A Detailed Execution Trace for Cnt2.

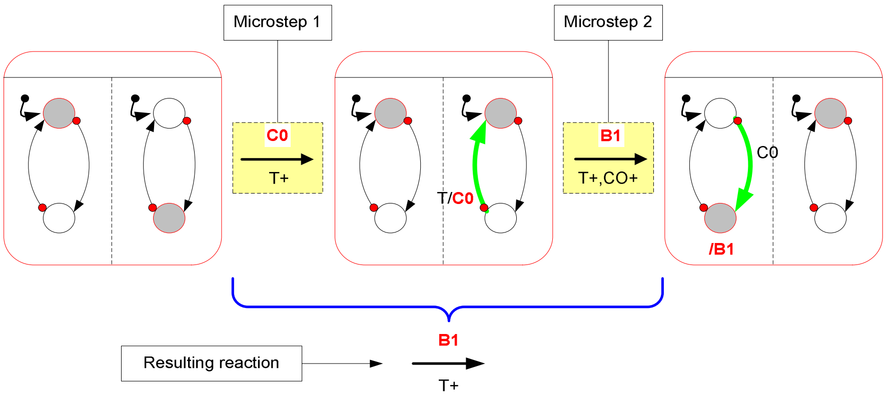


Figure 5-5: Microsteps.

Remark: The microstep evolutions are given to facilitate understanding of reactions. The user only sees the whole (instantaneous) reaction from a configuration to another one, with the concomitant emitted output signals.

5.3.3 Another example: Resource Manager

Consider the full controller **ResMgr** (Figure 5-6) composed of the 2 user's dialog controllers and the arbiter. Their cooperation is modeled by a *parallel composition* of the individual syncCharts. This example involves several local signals and *instantaneous dialogs*. While in the previous example communication was unidirectional (i.e., from one STG to another one), communication is now bidirectional. An instantaneous dialog is the manifestation of bidirectional communication among concurrent STGs.

Consider the behavior when **Arbiter** is in the state granting the resource to **User2**, while **User1** is requesting the resource by sustaining signal **Rq1** (configuration = {**Wg1, s2, Busy2**}).
Let **k** be the instant when **S2** occurs. Figure 5-7 represents reactions **k** and **k+1**. At instant **k**, **S2** triggers a transition so that **Rl2** is emitted, causing the **Arbiter** to come back to its **Idle** state (configuration = {**Wg1, Idle, Idle2**}).
At instant **k+1**, since **User1** sends **Rq1**, **Arbiter** leaves the **Idle** state, enters state **s1**, and emits **G1**. Now, **G1** being present, state **Wg1** is exited, state **Busy1** is entered, and signal **Rn1** is emitted (configuration = {**Busy1, s1, Idle2**}).

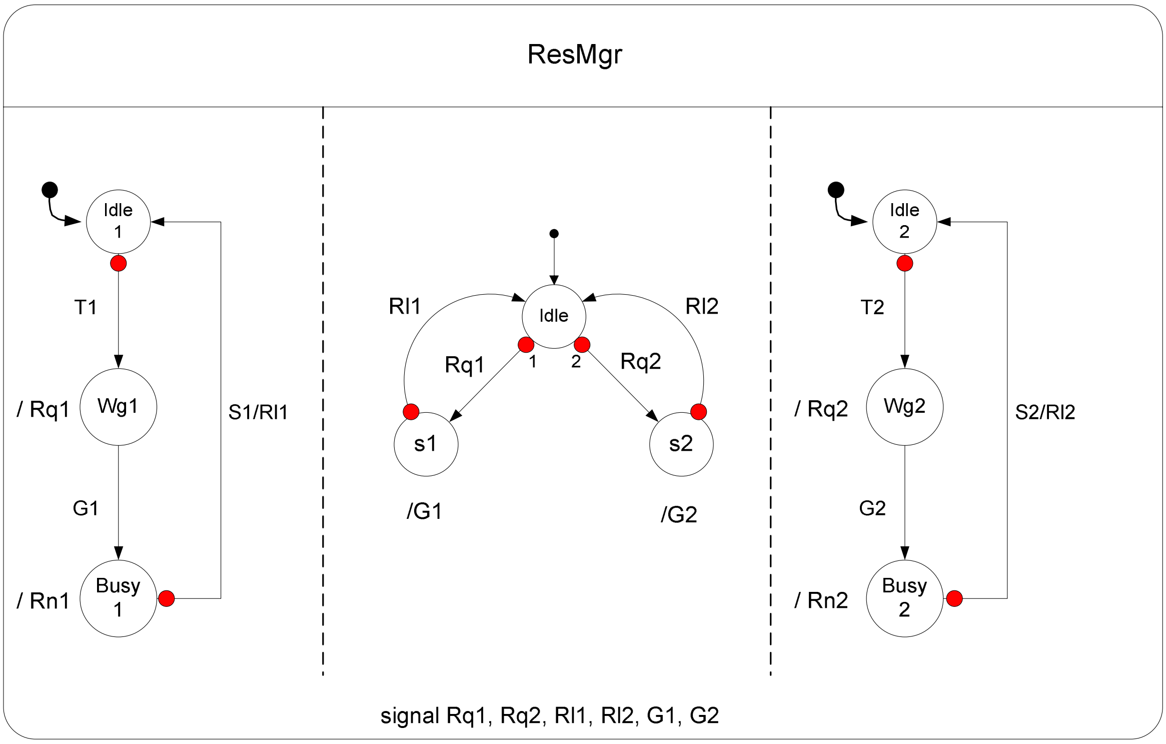


Figure 5-6: Controller of the Resource Manager.

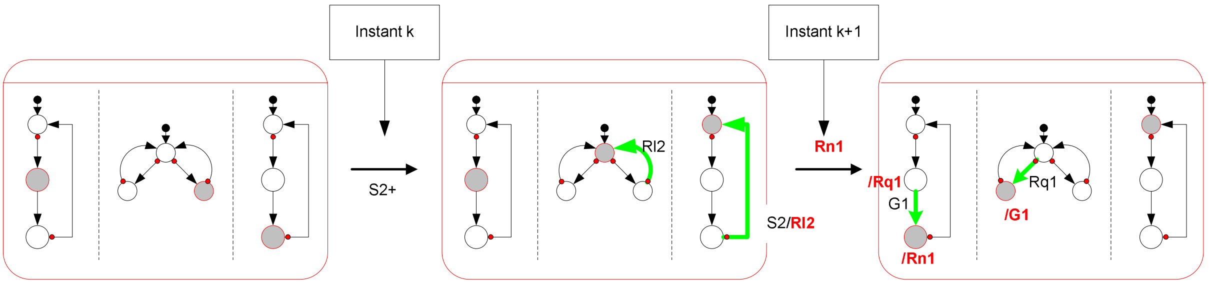


Figure 5-7: Partial Execution trace of the Resource Manager Controller.

Figure 5-8 depicts the microsteps of reaction **k+1**, clearly showing how STGs influence each other. Instantaneous dialogs enable powerful instantaneous communication protocols. A drawback of this expressiveness is that mutual influence may be source of instantaneous cyclic communications. A special section will be devoted to such faulty behaviors.

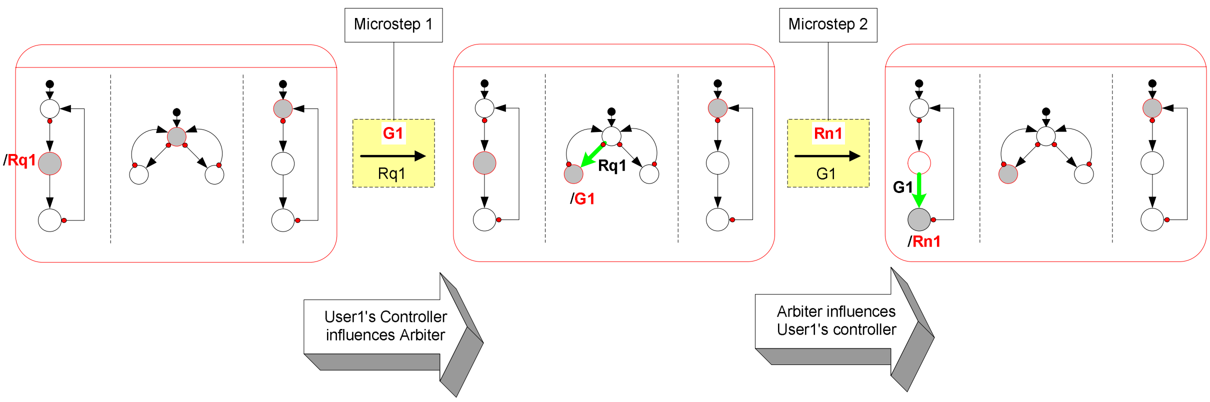
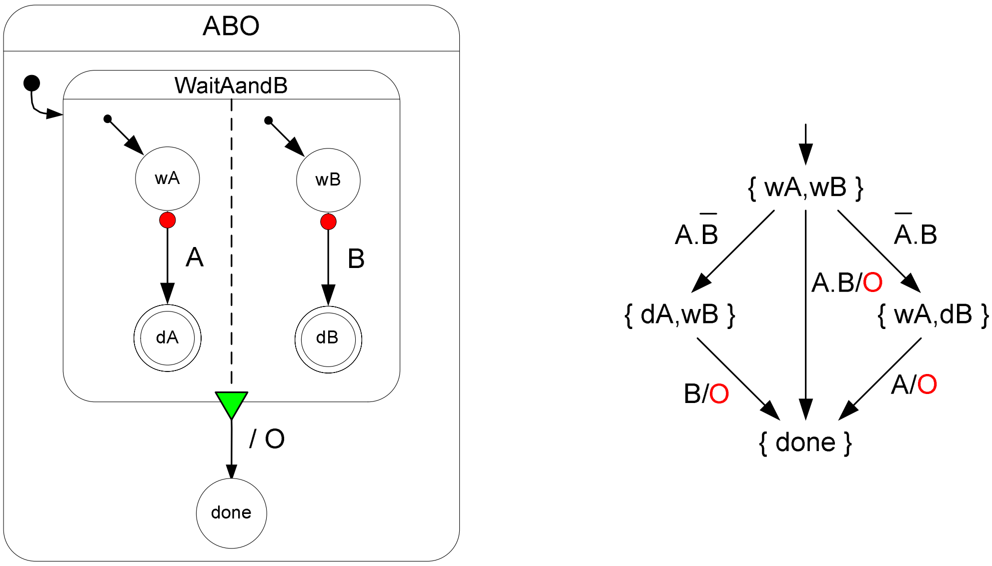


Figure 5-8: Microsteps in an Instantaneous Dialog.

5.3.4 Concurrency and Normal Termination

To illustrate the combined use of concurrency and normal termination, we choose to model a Memory Transaction System. This system waits for two concurrent events: availability of an Address (input signal **A**), and availability of Data (input signal **B**). As soon as both events have occurred, the system performs a Memory Write (output signal **O**). Note that **O** is emitted at the very instant when the last of **A** and **B** becomes present.

SyncChart **ABO** in Figure 5-9 specifies this behavior. As clearly shown on the syncChart, the system is making two concurrent waits. Suppose that **A** has occurred and that we are waiting for **B**. What happens when **B** occurs is traced in Figure 5-10. This behavior results from the following rule: When each (concurrent) STG in a macrostate reaches a final state, then the macrostate is immediately exited by its normal termination transition. This behavior generalizes the one presented in Section 5.2, where the macrostate contains only one STG. Note that the lack of normal termination transition for a macrostate with final states reveals a ill-structured syncChart.



SyncChart Equivalent Mealy Machine

Figure 5-9: Synchronized Termination.

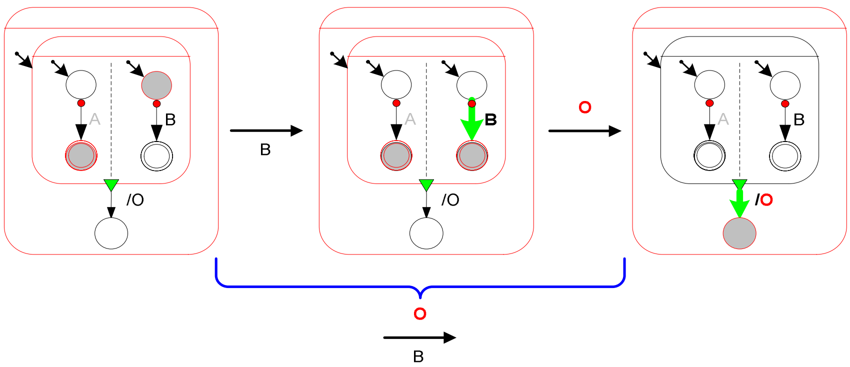


Figure 5-10: Execution of a synchronized termination.

An equivalent Mealy machine is given in Figure 5-9. It seems even simpler than the syncChart. In fact, this is no longer the case when the system waits for n > 2 independent signals. The number of states and transitions of the Mealy machine increases exponentially with respect to n (2^n states), whereas the complexity of the syncChart is linear (n concurrent STGs). Figure 5-11 is a syncChart that waits for three signals. The corresponding Mealy machine is left as an exercise for the reader.

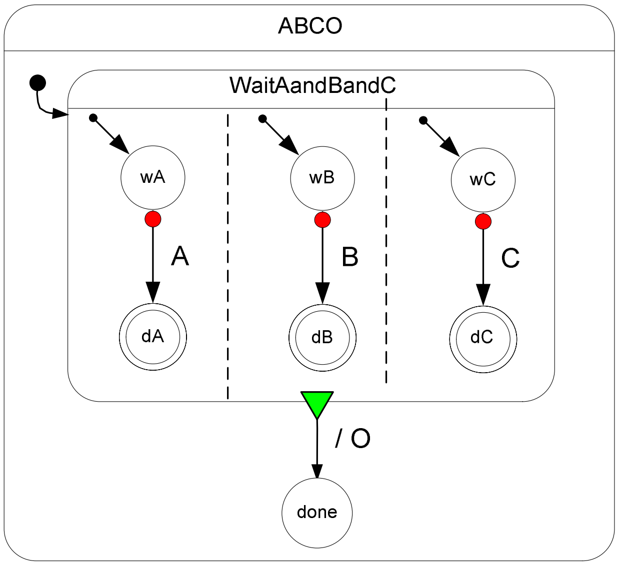


Figure 5-11: Waiting for three signals.

Another drawback of the state machine representation is *the lack of structure*: it is a flat model, and the same signal appears on many transitions. This is against the good software engineering principle "*Write Things Once*". SyncCharts, like Esterel (as explained in the Esterel Language Primer [Berry 1997]), often replaces replication by structure.

5.4 Preemption

The *preemption* is the possibility given to an agent to interrupt the execution of another agent. This interruption may be either definitive (*abortion*) or temporary (*suspension*). SyncCharts support both kinds of preemption. In this section we analyze abortion, suspension is presented later (Section 8.3).

Abortion has been presented as the way to exit a state (Section 4.5.2). It can apply to macrostate as well. When a macrostate is exited by abortion, a *strong abortion* forbids any reaction within the aborted macrostate prior to the abortion. On the contrary, a *weak abortion* lets the macrostate react before exiting. This explains why output signals associated with a state are not emitted in case of a strong abortion, and emitted with a weak abortion. Below are examples of abortions applied to macrostates.

5.4.1 ABRO: Strong Abortion on a Macrostate

The Memory Transaction System is augmented with a possibility to abort a transaction (signal **R**). **R** is a reset signal that erases previously received occurrences. If **R** occurs simultaneously with the second awaited signal, the transaction is also aborted.

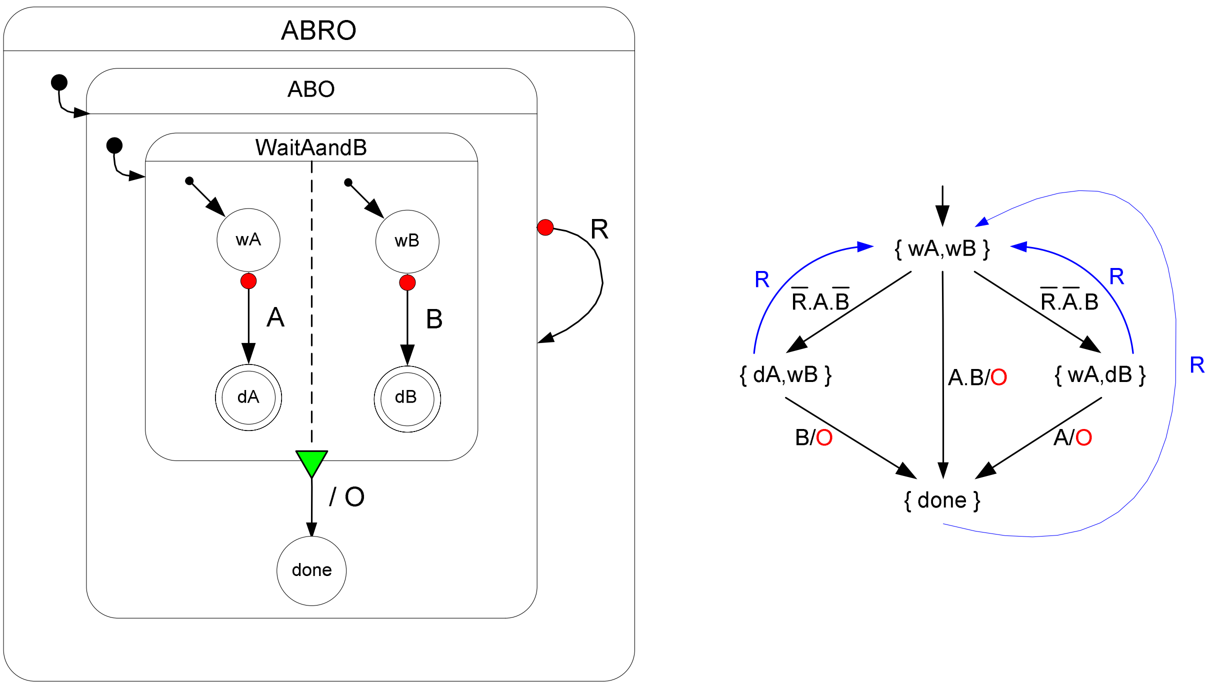
This behavior is easily expressed with a syncChart: it is sufficient to exit macrostate **ABO** as soon as **R** occurs. This is done by a strong abortion transition whose source is **ABO** (Figure

5-12). Nothing has to be changed within the macrostate. As for the re-initialization of the transaction, it is enforced by the target of the abortion transition that is macrostate **ABO** itself.

Example of Reaction with Preemption and Hierarchy

Consider **ABRO** when states **dA** and **wB** are active in macrostate **WaitAandB**. Containing macrostates **ABO** and **ABRO** are also active. What is the reaction of the syncChart when **R** and **B** occur simultaneously?

The triggers of two strong abortion transitions are satisfied: **R** enables the preemption of macrostate **ABO**, **B** enables the preemption of simple state **wB**. Since a strong abortion prevents any execution in the preempted state, the preemption caused by **R** is taken, while **B** preemption is ignored. **ABO** and all contained states are exited without any internal execution. Since the target of the abortion transition is macrostate **ABO**, this macrostate is instantaneously (re-)entered. Its initial state **WaitAandB** is also instantaneously entered. And finally, the initial state of both STGs in **WaitAandB** is immediately entered. Newly activated simple state **wB** is not preempted by **B**: only a strictly future occurrence of **B** can do that. Figure 5-13 shows the reaction. The equivalent Mealy machine (Figure 5-12) is cluttered with transitions labeled by **R**. The number of such transitions increases exponentially with the number of awaited signals.



SyncChart Equivalent Mealy Machine

Figure 5-12: SyncChart for ABRO.

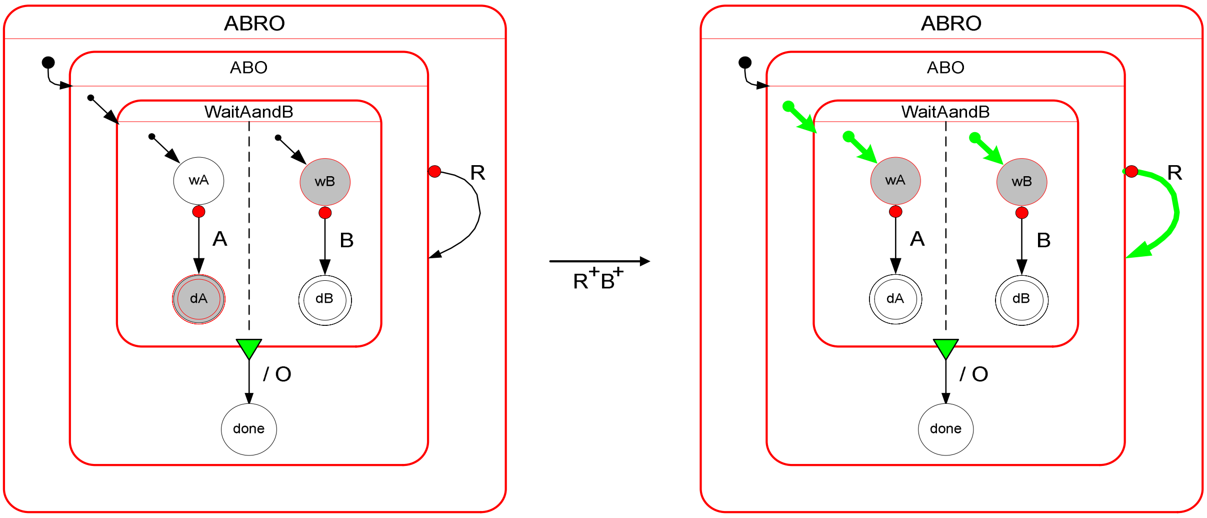
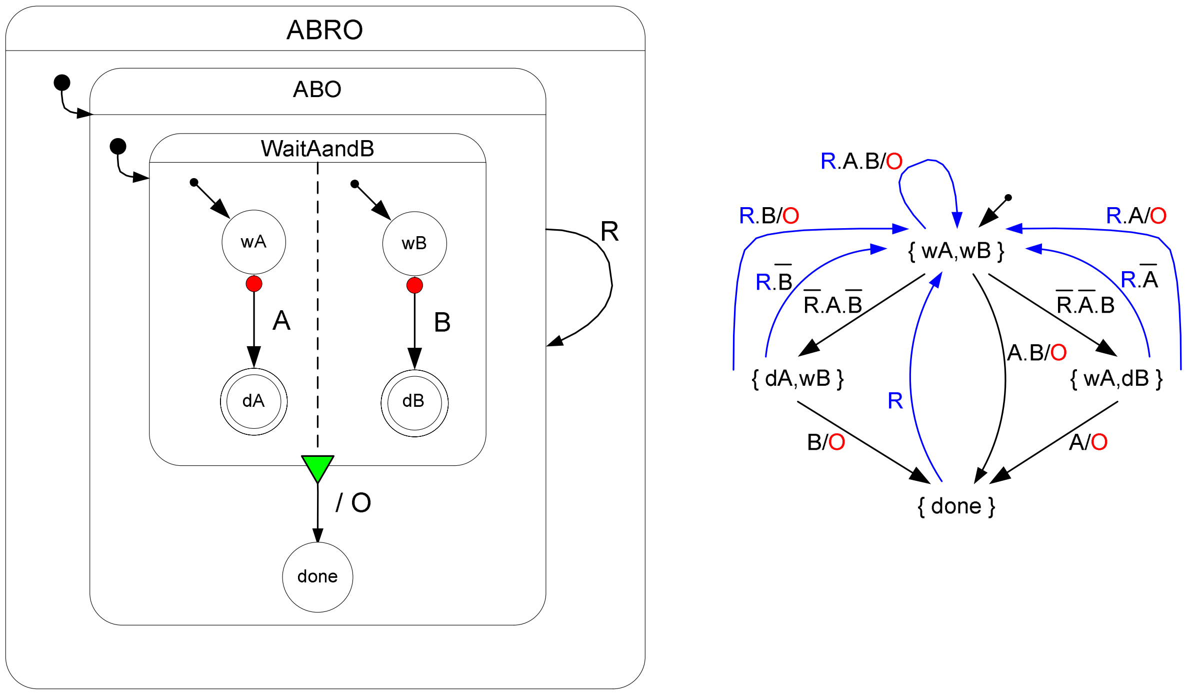


Figure 5-13: A Reaction involving preemption and hierarchy.

5.4.2 ABRO Variant: Weak Abortion on a Macrostate



SyncChart　　　　　Equivalent Mealy Machine

Figure 5-14: SyncChart (variant) for ABRO.

As with the previous example, we consider **ABRO** when states **dA** and **wB** are active in macrostate **WaitAandB**. Containing macrostates **ABO** and **ABRO** are also active. Suppose **B** and **R** present. Figure 5-15 describes the microsteps of the reaction. The triggers of two abortion transitions are satisfied: **R** enables a weak abortion of macrostate **ABO**, **B** enables a strong preemption of simple state **wB**.
Since the outermost preemption is weak, the reactions within macrostate **ABO** are performed: the strong abortion triggered by **B** takes place, resulting in a configuration in which states **dA** and **dB** are active (microstep 1). These states being final, the normal termination is taken,

signal **O** is emitted, and the configuration contains now **done** as an active state(microstep 2). There is not any more possible evolution in macrostate **ABO**. Now, the weak abortion transition triggered by R is taken, causing re-entering of macrostate **ABO**, and the nested macrostate **WaitAandB**. This results in a configuration with states **wA** and **wB** active (microstep 3). Thus the reaction has emitted signal **O** and has re-initialized the system.

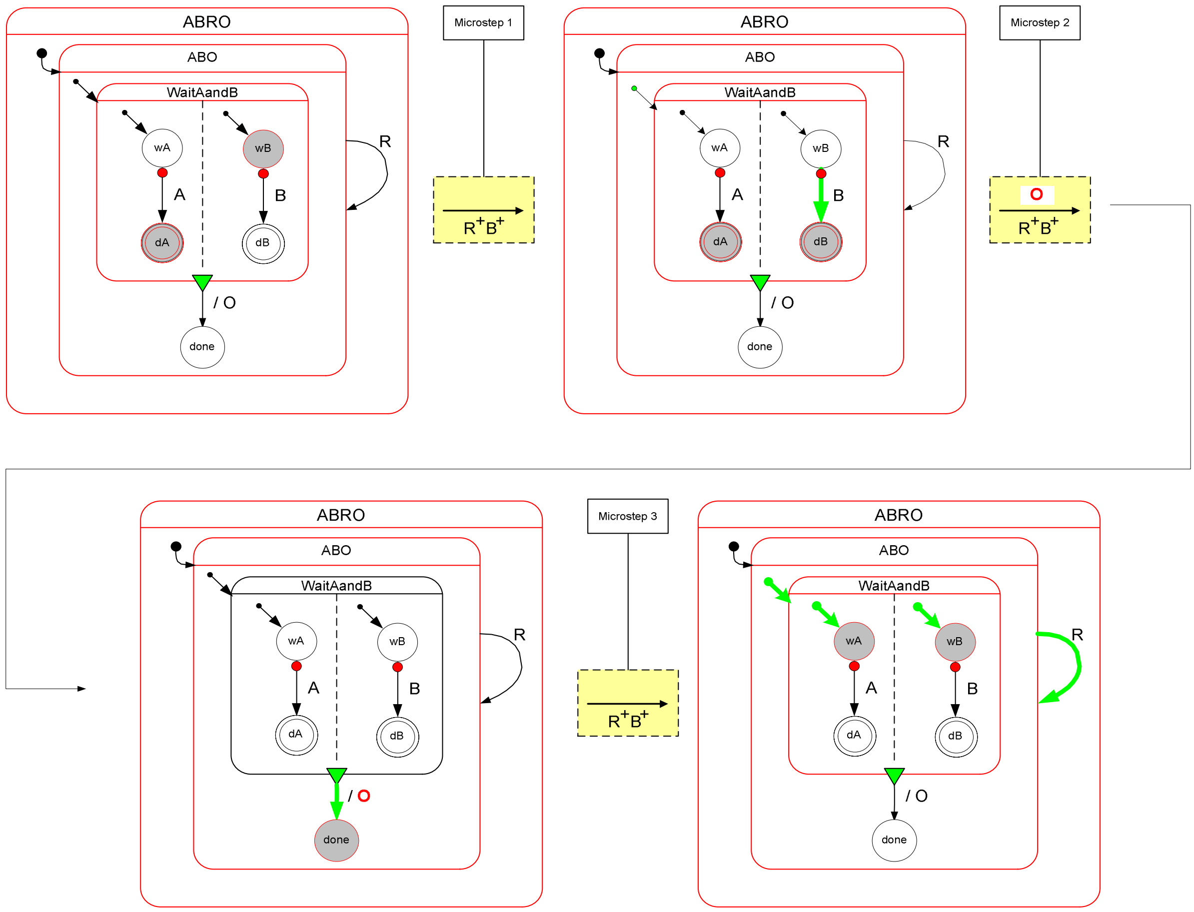


Figure 5-15: Microsteps in a case of weak abortion.

5.4.3 Abortion and Priority

Priority has been introduced in Section 4.7 to enforce a deterministic choice when several outgoing transitions of an active state are simultaneously enabled. May the priority be arbitrary assigned to transitions whatever the type? For flexibility, the user would like a positive answer, and yet, SyncCharts impose a constraint.

For any state,

- every outgoing transition has a different priority,
- any strong abortion transition has priority over any weak abortion transition,
- any weak abortion transition has priority over a normal termination transition.

This ordering is not the only sensible choice. In the Esterel language, for instance, normal termination has priority over weak abortion (weak abort statement in Esterel). The above rules make code generation easier, without reducing the expressiveness of the model. Imposing another priority ordering is possible by state nesting, which induce structural priorities. See Figure 5-16 for an example in which the normal termination is given priority over a weak abortion. The solution resorts to an extra level of nesting and a local signal **nt** (supposed not already defined within the scope of the outermost macrostate).

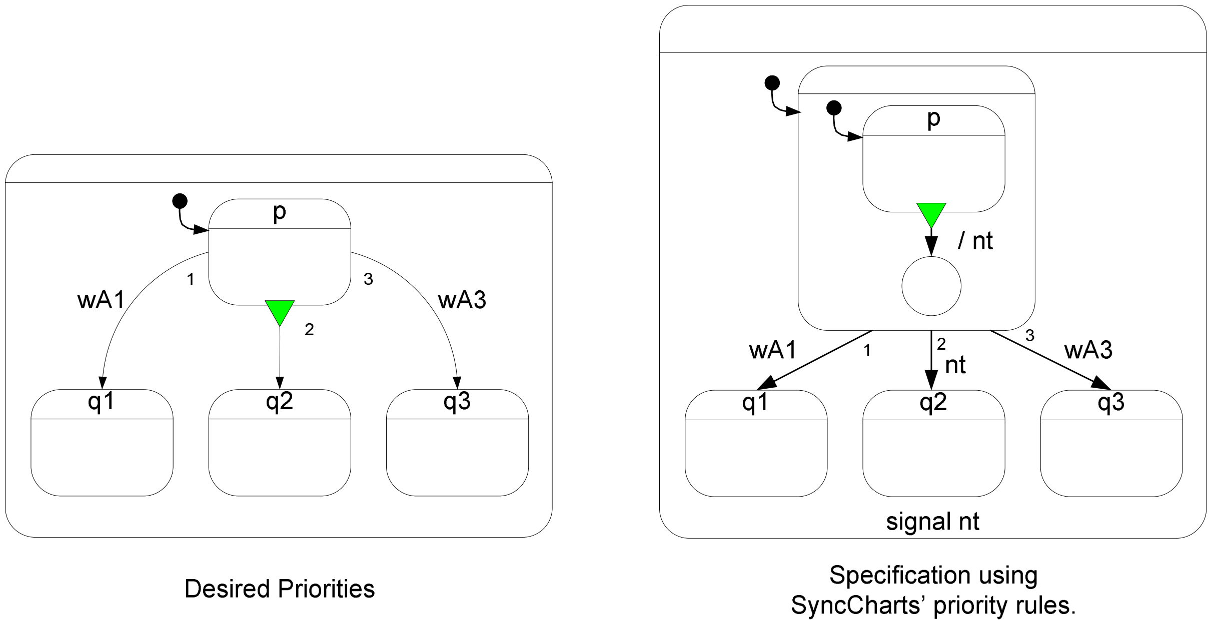


Figure 5-16: Imposing an arbitrary priority ordering.

Giving priority over strong abortion is more dangerous. The reason is that a weak abortion can be caused by the execution of the body of the state, while a strong abortion requires that the body of the state is not executed at all. This may cause incorrect behavior known as Causality Cycle and studied in Section 7. Figure 5-17 shows a syncChart that gives priority to the weak abortion triggered by **wA** over the strong abortion triggered by **sA**. This priority is enforced by a complex trigger: **sA** and not **wA**, not by the structure.

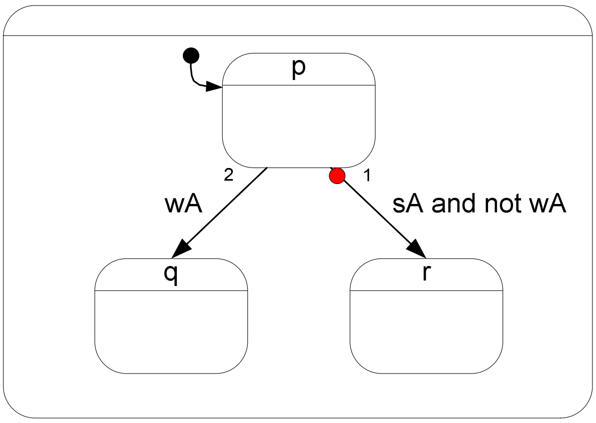


Figure 5-17: Imposing higher priority to weak abortion.

5.4.4 Trigger-less Transitions

The trigger field in a transition label is optional. A trigger-less transition becomes enabled at the instant just after the activation of its source state. An interpretation is that the default trigger is the special signal tick whose occurrence is expected. tick is a reserved word that denotes an implicit signal present at every instant. Thus the behavior consists in waiting for the first strictly future occurrence of tick, which, by definition, occurs at the next instant.

5.5 Summary

In this section we have explored the constructs for hierarchy, concurrency, and preemption. A reaction generally involves several microsteps and results from instantaneous dialogs among concurrent parts of the syncCharts. Thanks to the instantaneous broadcast of signals, and priority enforcement (through the structure and explicit declarations) these reactions are kept deterministic. Moreover, through their rich structuring possibilities, SyncCharts make it possible to apply the good software principle of Write Things Once.

6 Computation of a Reaction: A First Approach

6.1 Abstract

The two previous sections have described in an informal way the reactions of a syncCharts. A more formal approach is necessary in order to deal with more complex examples. This section starts with a precise definition of the syntax of SyncCharts, so that the entities that compose a syncChart will be known and referred to without any ambiguity.
An operational semantics, relying on the structure, is then proposed.

6.2 SyncCharts Structure: Associated Tree

The behavior of a syncChart results from the cooperation of simple functional units we call *reactive-cells*. A reactive cell is a state (either a simple-state, or a macrostate), with all its outgoing transitions. Signal broadcasting is the unique communication medium among reactive-cells. Since a syncChart is a hierarchical model, its structure should be exploited to compute its reactions. So far, an informal presentation of the structure has been sufficient. In order to explain how to compute reactions of SyncCharts, we have adopted a more formal presentation.

The syncChart's structure respects a strict state containment policy. A tree representation can be easily attached to any syncChart. This tree alternates macrostates and state-transition graphs (STGs). The leaves of the tree are simple-states.

6.2.1 Syntax for SyncCharts

The ABRO example will illustrate the definitions of the abstract syntax. This is a simplified version, restricted to pure SyncCharts.

Macrostate

With a *syncChart* is associated a unique *macrostate* called its top. Top designates the top-level state that is the root of the state containment hierarchy. A macrostate is composed of a non empty set of STGs, and three possibly empty sets of signals: *input signals*, *output signals*, *local signals*. For a macrostate **M**, these sets are denoted **M.G**, **M.I**, **M.O**, **M.L**, respectively.

Reactive-Cell

An STG is a non empty set of *reactive-cells*. One of these reactive-cells is referred to as *initial*. A reactive-cell has a *body* and a possibly empty set of *outgoing transitions* of different kinds (strong abort, weak abort, normal termination).
The body is either a simple-state or a macrostate. A simple-state is not refined: it is a leaf of the tree.
Graphically, no special picture is defined for a reactive-cell. Its body and its outgoing transitions are drawn, instead (see Figure 6-2).

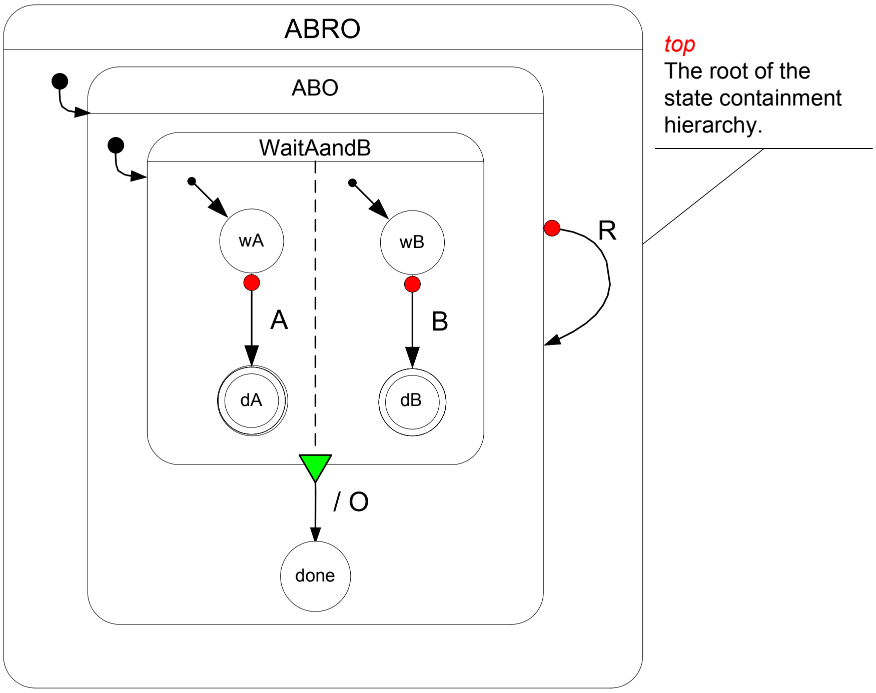


Figure 6-1: A SyncChart.

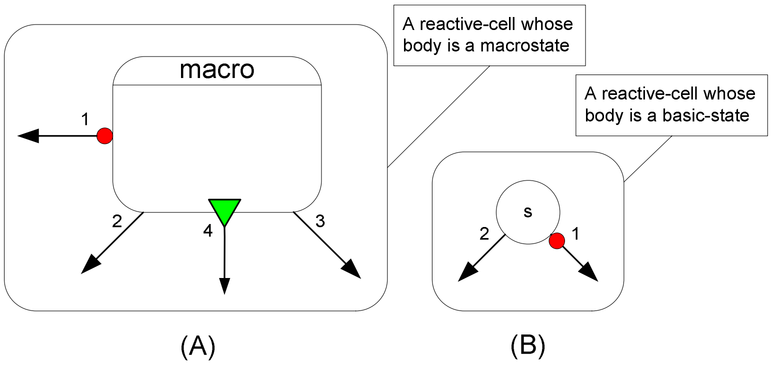


Figure 6-2: Reactive Cells.

Outgoing Transitions

An outgoing transition has a *destination cell* and a *label*. The destination cell is a reactive-cell. Graphically the arrow end of the transition points to the body of the destination cell drawn as a simple-state or a macrostate. A label is composed of three optional fields: a *trigger*, a *guard*, an *effect*.

Remark: depending of the kind of transition, some fields may be forbidden. Details are omitted at this level.

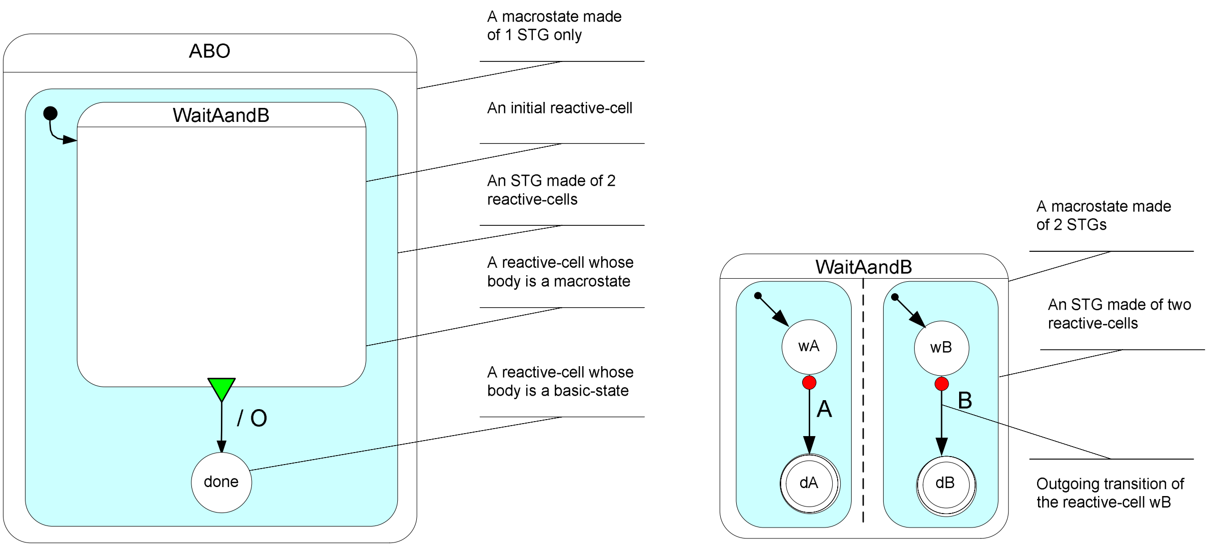


Figure 6-3: Macrostate and STGs.

Naming convention

STGs and reactive-cells cannot be named by the user. Assigning automatic identifiers to STGs and reactive-cells will make algorithm easier to express.

Let **M** be a macrostate. Each STG directly contained in **M** is given a unique identifier **M.G_k**, where **k** is an integer between 1 and n (the number of concurrent STGs in **M**). As for reactive-cells, they are called by the name of their body, which is unique.

For a STG **G**, its set of reactive-cells is denoted by **G.S**, and its initial state by **G.ini**.

Example

For syncChart "**ABRO**" (Section 5.4.1)

The macrostate named **ABRO** is the top.

ABRO.I = {**A, B, R**}
ABRO.O = {**O**}
ABRO.L = ∅

Macrostate **ABRO** is composed of one STG named **ABRO.G_1**, by convention, and **ABRO.G** = {**ABRO.G_1**}.

STG **ABRO.G_1** is made of only one reactive-cell whose body is macrostate **ABO**. The set of reactive-cells **ABRO.G_1.S** = {**ABO**}. With our convention, this reactive-cell is also named **ABO**. The context easily resolves possible ambiguity between the macro-cell and its body.

ABRO.G_1.S = {**ABO**}
ABRO.G_1.ini = **ABO**

Macrostate **ABO** is composed of one STG.

ABO.I = {**A, B**}
ABO.O = {**O**}
ABO.L = ∅
ABO.G = {**ABO.G_1**}

STG **ABO.G_1** is made of two reactive-cells, one with macrostate **WaitAandB** as its body, and another the body of which is a simple state named **done**.

ABO.G_1.S = {**WaitAandB, done**}
ABO.G_1.ini = **WaitAandB**

Macrostate **WaitAandB** is composed of two STGs.

WaitAandB.I = {**A, B**}
WaitAandB.O = ∅
WaitAandB.L = ∅
WaitAandB.G = {**WaitAandB.G_1**, **WaitAandB.G_2**}

Finally, each STG is composed of two reactive-cells with simple states as bodies:

WaitAandB.G_1.S = {**wA, dA**}
WaitAandB.G_1.ini = **wA**
WaitAandB.G_2.S = {**wB, dB**}
WaitAandB.G_2.ini = **wB**

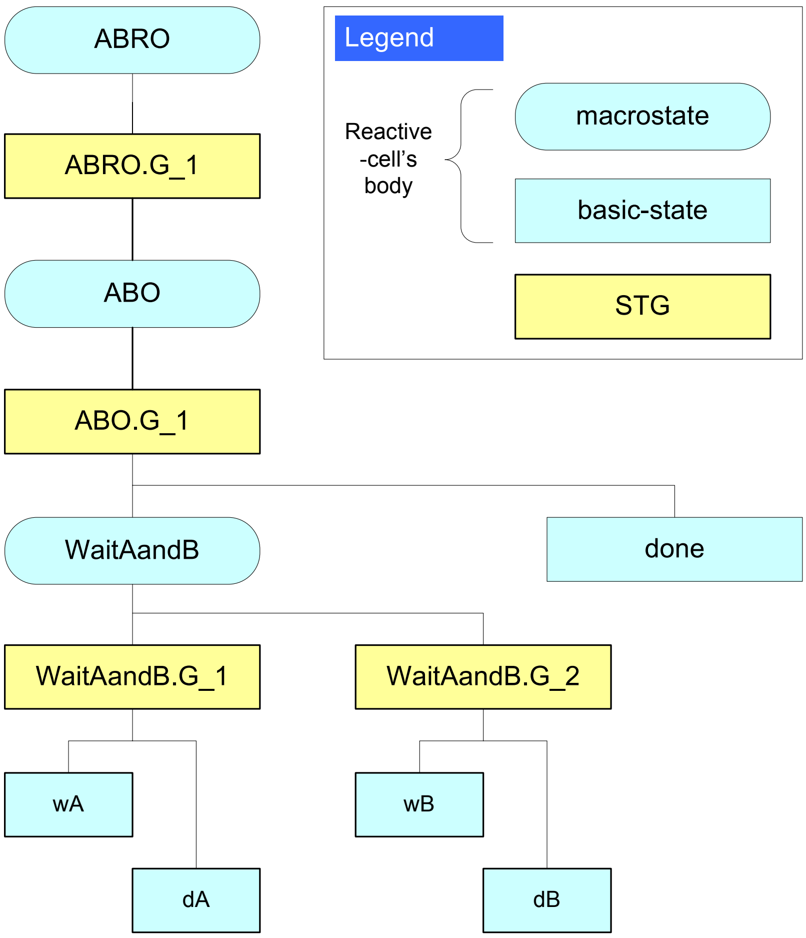


Figure 6-4: Tree associated with ABRO.

An outgoing transition is denoted as a 4-tuplet: <*type-of-arc*, *trigger*, *effect*, *target-identifier*>. Empty fields are left blank. *Type-of-arc* can be sA for strong abortion, wA for weak abortion, nT for normal termination. For a reactive-cell **R**, its outgoing transition set is denoted by **R.out**.

> **ABO.out** = {<sA, **R**, , **ABO**>}
> **WaitAandB.out** = {<nT, , **O**, **done**>}
> **done.out** = ∅
> **wA.out** = {<sA, **A**, , **dA**>}
> **dA.out** = ∅
> **wB.out** = {<sA, **B**, , **dB**>}
> **dB.out** = ∅

The tree associated with syncChart **ABRO** is represented in Figure 6-4.

6.3 Behavior

6.3.1 Configuration

A *configuration* is a maximal set of states (macrostates or simple-states) that the system could be in simultaneously. Any subset of states is not a legal configuration.
Let **T** be the top macrostate associated with a syncChart. A *legal* configuration **C** for **T** (and for the syncChart) must satisfy the following rules:

1. **T** is in **C**,
2. If a macrostate **M** is in **C**, then **C** must also contain for each STG **G** directly contained in **M**, exactly one state directly contained in **G**,
3. **C** is maximal and contains only states satisfying rules 1 and 2.

The legal configurations of **ABRO** are:

> {**ABRO**, **ABO**, **done**}
> {**ABRO**, **ABO**, **WaitAandB**, **wA**, **wB**}
> {**ABRO**, **ABO**, **WaitAandB**, **wA**, **dB**}
> {**ABRO**, **ABO**, **WaitAandB**, **dA**, **wB**}
> {**ABRO**, **ABO**, **WaitAandB**, **dA**, **dB**}

The legality of a configuration relies on structural considerations only. SyncCharts represent dynamic behaviors that are not simply characterized by the structure. Only a subset of the legal configurations is of interest for the user: the set of stable configurations.
A *stable* configuration is a legal configuration that the syncChart can reach after a sequence of reactions. As shown in Figure 5-12, {**ABRO**, **ABO**, **WaitAandB**, **dA**, **dB**} is not a stable configuration for **ABRO**.

Macrostates and simple-states in a configuration are said to be *active*. By extension, a reactive-cell the body of which is active, is said to be active. An STG with an active reactive-cell is also qualified as active.

How to compute stable configurations and signals emitted during a reaction is explained in the next two sections.

6.3.2 Computation of a Reaction: Overview

For the sake of simplicity, only pure syncCharts are considered. Moreover, we assume simple triggers consisting of a single signal whose presence is expected. These limitations will be relaxed later. Even for this restricted class of syncCharts, microstep construction may be not straightforward. In order to make it easier, we decompose a reaction according to the hierarchy. The reaction of a macrostate relies on the reactions of its STGs. The reaction of an STG relies on the reaction of its reactive cells. The reaction of a reactive-cell relies on the reaction of its body (a macrostate or a simple-state). Of course, this approach is recursive and has to be applied down to the leaves of the tree which are simple-states. Figure 6-5 summarizes the process of computing a reaction.

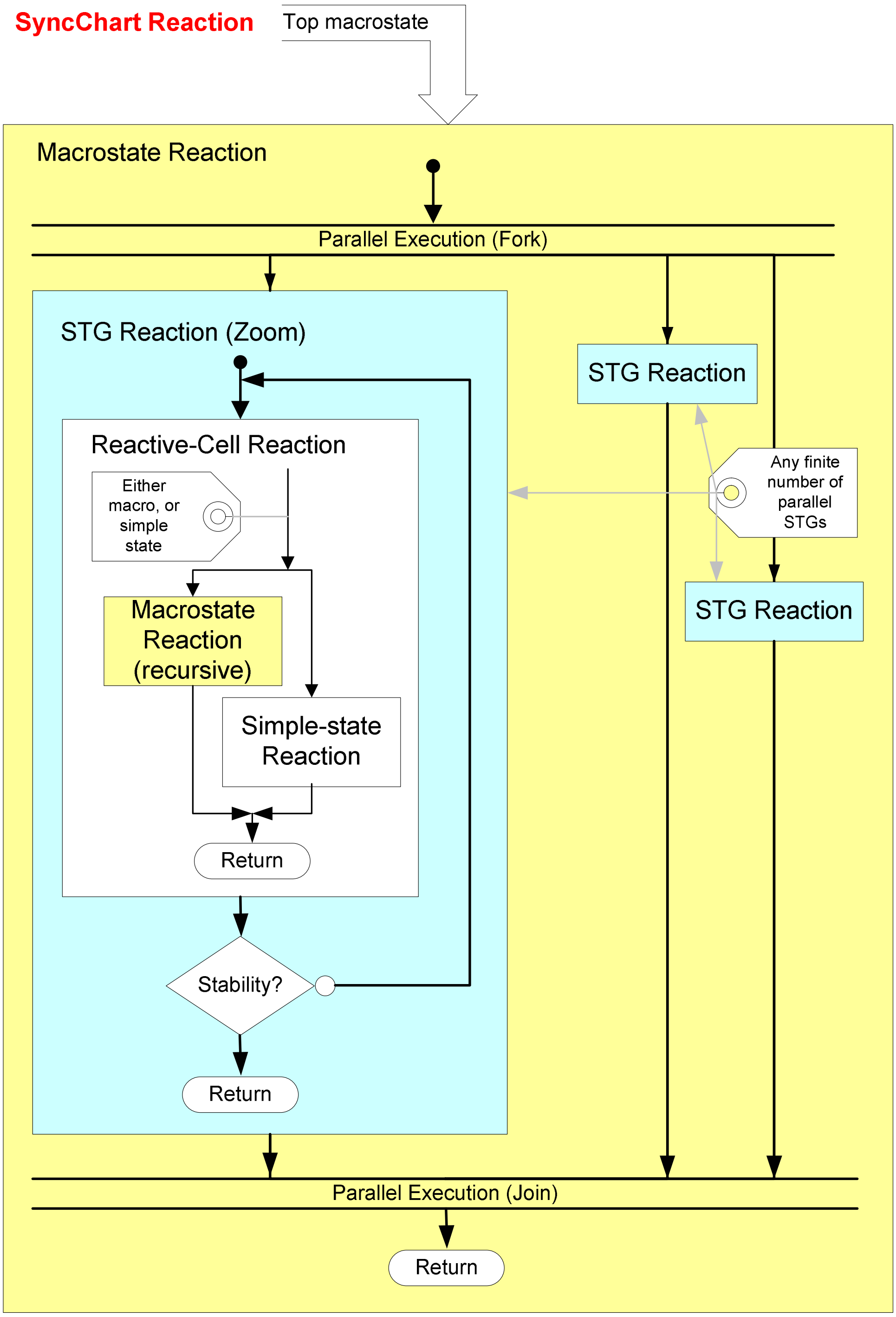


Figure 6-5: Overview of a Reaction.

6.3.3 Computation of a Reaction: Algorithms.

Termination Code

For computational purpose, the reaction of a component (reactive-cell, STG, macrostate, simple-state) returns a *termination code* taking its value in {**DONE**, **DEAD**, **PAUSE**}. This code is for internal use only and does not appear as a result of the reaction of the syncChart. Returning **PAUSE** means that the component has nothing left to do until the next instant. Returning **DONE** means that the component has terminated its execution, and that there is nothing left to do at the next instant. Returning **DEAD** means that the component has nothing left to do at the current instant and in the future (final state), and that it is candidate to join a normal termination. If this normal termination does not take place, then the component will have nothing to do at the next instant, but returning **DEAD** again.

The algorithms are given in a pseudo algorithmic language. Comments are allowed. We adopt the C language notation for comments.

Reaction of the syncChart

This is the upper level.
Given a stable configuration, a reaction is computed by:

Reaction of a syncChart	
	1 - Read input signals /* the presence status of all input signals is known */ 2 - Set all output signals to the “unknown” presence status (⊥) 3 - Compute reaction of the top macrostate associated with the syncChart /* yields emitted signals and the next (stable) configuration */

Reaction of a Macrostate

This reaction returns either **DEAD** or **PAUSE**.

Reaction of macrostate **M**	
	1 - Set all local signals to the “unknown” presence status (⊥) 2 - For each STG **G** (directly contained) in **M** do in parallel /* Fork */ Compute the reaction of STG **G** Return the termination code in c(**G**) 3 - When all parallel executions are done, /* Join */ Compute C = maximum of c(**G**) for all STGs in **M** 4 - Return C

For the calculation of C, consider **DEAD < PAUSE**, so that a macrostate reaction returns:

1. **PAUSE** if and only if some concurrent STG in **M** returns **PAUSE**,
2. **DEAD** otherwise.

Comments:
There will be still something to do at the next instant if at least one of the parallel branches has something to do at the next instant (Rule 1). Conversely, when all the parallel branches are **DEAD**, the macrostate returns **DEAD**, that is, is ready for a normal termination.

Reaction of a Simple-state

This reaction returns either **DEAD** or **PAUSE**.

Reaction of simple-state **S**	
	1 - if **S** is a final state then return **DEAD**. 2 - If an effect is associated with the simple-state then emit all the signals in "effect". 3 - Return **PAUSE**

This is a very simple behavior. A final state has nothing to do but returning **DEAD**. A non final state can emit signals, and then returns **PAUSE**.

Reaction of an STG

This reaction returns either **DEAD** or **PAUSE**.

Reaction of STG **G**	
	1 - If there is no current state in **G** then set current state to the initial state 2 - Compute the reaction of the reactive-cell whose body is the current state 3 - Let r be the termination code. If r is equal to **DONE** then set current state to nextState, and go to 2 4 - Return r /* here r cannot be **DONE** */

Comments:
When entering a macrostate, the current state of each STG is undefined. Take the initial state as the current one (Step1). If the STG is already active the current-state is its (unique) currently active state.
Step 2 computes the reaction of the active reactive-cell. If this reaction returns **DONE**, this means that the state passes the control instantaneously to its successor. In this case, the new active state must also react. Since several instantaneous reactions can be chained in an STG, the algorithm uses a while-loop (steps 2 and 3). After a finite number of iterations, step 4 is executed with a termination code different from **DONE**. A non terminating loop indicates a syncChart with infinite instantaneous loop: The syncChart must be rejected.
The choice of the successor state (nextState) is done in the reactive-cell reaction (see below) according to the outgoing transition taken to exit the active state.

Reaction of a Reactive-Cell

This reaction is the heart of the reaction. It is at the cell-level that presence status of signals is tested and abortion decisions are taken. It is also the place where the analysis goes deeper in the hierarchy.
The test of the trigger is especially important. We propose a special function **testAbortion** that, given a set of abortion transitions, returns the first passing transition, if any, or null otherwise. Transitions are considered in a decreasing order of priority. When testing the presence of a triggering signal, its status may be unknown. If so, the *execution is suspended* till another concurrent execution thread will fix the status of the tested signal.

TestAbortion on a set **A** of abortion transitions	
	for each transition t in **A**, considered in the decreasing order of priority do Let **S** be the trigger of t Wait till **S** can be evaluated if **S** is satisfied then return t

	end for return null

testSA (**testWA**, respectively) is the **testAbortion** function applied to strong (weak, respectively) abortion transitions of the considered reactive-cell.

The outline of the computation of a reactive-cell reaction is as follows:

1. Strong abortion test:
   - If a strong abortion transition is enabled then take this transition
   - /* don't execute the body */
   - Return **DONE**
2. Execute the body:
   - If a macrostate, then recursive call
   - If a simple-state, then terminal call
3. Weak abortion test:
   - /* note that, at this point, the body has completed its execution */
   - If a weak abortion transition is enabled then take this transition
   - Return **DONE**
4. Normal termination test:
   - If the body has returned **DEAD** then take the normal termination transition
   - Return **DONE**
5. End of the reaction:
   - If you reach this point, then return **PAUSE**

A more precise description of the algorithm is represented by a flowchart (Figure 6-6).

Comments:
The status of a reactive-cell may be either **IDLE** or **ACTIVE**. This status is persistent information, initially set to **IDLE** and then possibly modified during a reaction.
Upon the activation of a reactive-cell, a Boolean named **firstInstant** is asserted. This flag allows the behavior to be different at the first instant and at the following instants: the triggers are not tested at the first instant.
Usually, the control stays in a state for more than one instant. At the end of the first reaction the status of the reactive-cell is set to **ACTIVE**. From now on, **firstInstant** is false, and the triggers are tested.
A "standard" reaction (not the first one) is as follows:

1. Check strong abortions. If the trigger of a strong abortion transition is satisfied, then exit the state, take the corresponding transition and set the status of the reactive-cell to **IDLE**. Note that, in case of strong abortion, the body of the reactive-cell is not executed at all.
2. If no strong abortion is possible, then compute the reaction of the body of the reactive-cell. This is a recursive call. When this call returns, save the termination code in a variable (**B**).
3. Now check for weak abortions. The behavior is then the same as for a strong abortion. Note that, with weak abortion the body of the reactive-cell has already done a complete reaction.
4. If no weak abortion is possible, then test for normal termination. The normal termination occurs if **B** is equal to **DEAD**.

5. Finally, if no abortion or normal termination is possible, then the reaction of the reactive-cell is over for the current instant. Return **PAUSE**.

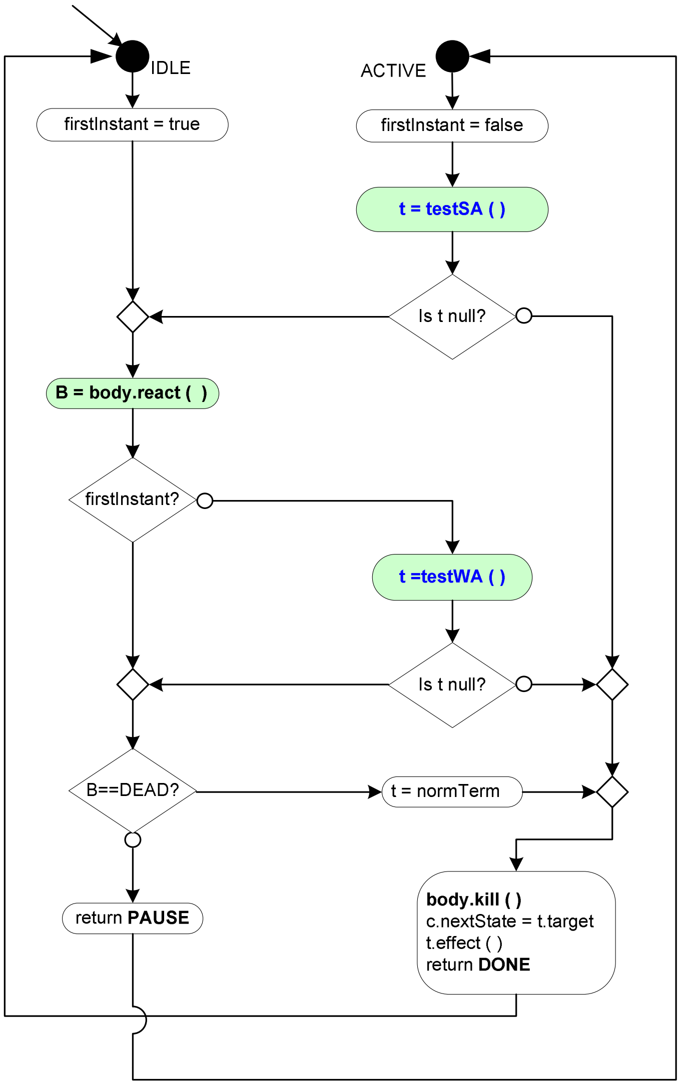


Figure 6-6: Reaction of a Reactive-Cell.

The capsules with colored background, in Figure 6-6, are places where the execution of the reaction can be suspended, waiting for extra information about the presence status of some signal. Computing the reaction of a reactive-cell usually requires *concurrent executions*.

When the state is exited, due to abortion or normal termination, a sequence of actions is performed (rectangle with rounded corners in the right lower side of the picture):

1. "Kill" the body of the reactive-cell. This means a recursive de-activation of all the components contained in the state. Because of the proposed algorithm, all these components have already reacted, or not react at all (when strongly aborted), so that their de-activation will cause no trouble.
2. Exit from the state by taking the transition **t**. Thus, execute the associated effect, and set the target of the transition as the new current state of the STG.
3. Set the persistent reactive-cell status to **IDLE**.

4. Return the termination code **DONE** to notify that this reactive-cell terminates its reaction.

Final Reactive-Cell

This is a very special case: the body of the reactive-cell is a final (simple) state. Of course, there is no need for transition checking. The flowchart degenerates to the one shown in Figure 6-7.

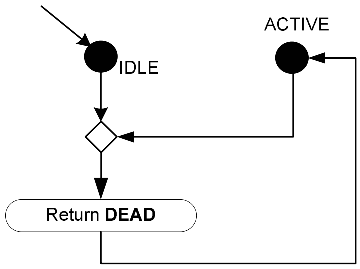


Figure 6-7: Reaction of a "final" Reactive-Cell.

6.4 Examples of Computation of a Reaction

This section illustrates the execution of a reaction step by step in great details. The reader may skip this section during the first reading.

6.4.1 Application to ABRO

Consider **ABRO** in the stable configuration {**ABRO, ABO, WaitAandB, dA, wB**}. What is the reaction of the syncChart when **R** and **B** occur simultaneously? This question has already been answered using an informal semantics (Section 5.4.1). The reaction is:

{ **ABRO, ABO, WaitAandB, dA, wB** } $\xrightarrow[\{R,B\}]{}$ { **ABRO, ABO, WaitAandB, dA, dB** }

Now, we derive this reaction by applying the above procedures. Comments, line numbering and indentation clearly show successive calls and their depth.

1 - Read input signals: **R+, A-, B+**
2 - Set all output signals to unknown: $\mathbf{O}^{\perp}$
3 - Reaction of top.
 /* Macrostate **ABRO** */
 1 – Set local signals to unknown: empty set, so nothing to do
 2 – For each STG do: only one STG **ABRO.G_1**
  /* Reaction of STG **ABRO.G_1***/
  1 – **ABO** is already active
  2 – Compute the reaction of the active reactive-cell
   /* Reaction of the reactive-cell **ABO** */
   1 – **firstInstant** is set to false
   2 – check for strong abortion: there is only one outgoing arc.
   The trigger is **R**. Since **R** is present the transition must be taken.
    /* abortion procedure */
    1 – Kill
     /* recursive kill of the body of **ABO** */
      /* recursive kill of the body of **WaitAandB** */
      set the status of **dA** to **IDLE**
      set the status of **wB** to **IDLE**

set the status of **WaitAandB** to **IDLE**
2 – nextState = t.target = **ABO**
/* a special case: the source and the target are the same */
3 – effect (void)
4 – set the status of **ABO** to **IDLE**
3 – return **DONE**
3 – Since termination code is **DONE**, current state is set to **ABO**, and go to 2
2 – Compute the reaction of nextState (**ABO**)
/* Reaction of the reactive-cell **ABO** */
1 – **firstInstant** is set to true /* **ABO** was **IDLE** */
2 – Compute the reaction of the body, i.e., macrostate **ABO**
/* Macrostate **ABO** */
1 – Set local signals to unknown: empty set, so nothing to do
2 – For each stg do: only one STG **ABO.G_1**
/* Reaction of STG **ABO.G_1***/
1 – no current state: **WaitAandB** becomes the current state
2 – Compute the reaction of the active reactive-cell
/* Reaction of the reactive-cell **WaitAandB** */
1 – **firstInstant** is set to true /* **WaitAandB** was **IDLE** */
2 – Compute the reaction of the body, i.e., macrostate **WaitAandB**
/* Macrostate **WaitAandB** */
1 – Set local signals to unknown: nothing to do
2 – For each STG do:
/* Reaction of stg **WaitAandB.G_1** */
1 – no current state: **wA** becomes the current state
2 – Compute the reaction of the current reactive-cell
/* Reaction of the reactive-cell **wA** */
1 – **firstInstant** is set to true
2 – Compute the reaction of the body
/* reaction of reactive-cell **wA** */
1 – no effect associated
2 – return **PAUSE**
3 – set the status of **wA** to **ACTIVE**
4 – return **PAUSE**
3 – r = **PAUSE** which not equal to **DONE**
4 – return **PAUSE**

/* Reaction of STG **WaitAandB.G_2** */
1 – no current state: **wB** becomes the current state
2 – Compute the reaction of the active reactive-cell
/* Reaction of the reactive-cell **wB** */
1 – **firstInstant** is set to true
2 – Compute the reaction of the body
/* reaction of simple-state **wB** */
1 – no effect associated
2 – return **PAUSE**
3 – set the status of **wB** to **ACTIVE**
4 – return **PAUSE**
3 – r = **PAUSE** which not equal to **DONE**
4 – return **PAUSE**

3 – return **PAUSE** which is the max of **PAUSE** and **PAUSE**
4 – return **PAUSE**
3 – B = **PAUSE**
4 – set the status of **WaitAandB** to **ACTIVE**
5 – return **PAUSE**

3 – termination code is not equal to **DONE**
4 – Return **PAUSE**

3 – the only STG returns **PAUSE**
4 – return **PAUSE**

3 – B = **PAUSE**
4 – set the status of **ABO** to **ACTIVE**
5 – return **PAUSE**

3 – r = **PAUSE**, which is not equal to **DONE**
4 – return **PAUSE**
3 – the only STG returns **PAUSE**
4 – return **PAUSE**

The new stable configuration is {**ABRO, ABO, WaitAandB, wA, wB**}. Signal **O** has never been tested during this reaction. Moreover, it has not been emitted. Its presence status is then set to absent. Thus, the reaction is:

$$\{ \textbf{ABRO, ABO, WaitAandB, dA, wB} \} \xrightarrow[\{R,B\}]{\varnothing} \{ \textbf{ABRO, ABO, WaitAandB, wA, wB} \}$$

This execution trace, definitely shows that detailed computation of reactions are not suitable for human users. Of course, the process can be automated, even if concurrent executions and suspensions due to triggering signal tests, make it not easy.

In fact, a human user should resort to the above procedures when in doubt about the behavior of a syncChart, or merely to understand the reaction of a syncChart at a micro-step level.

The **ABRO** example has illustrated preemption on a hierarchy. We explain now an instantaneous dialog, already studied in Section 5.3.3. On the execution trace, housekeeping like nested calls are omitted, advantageously replaced by comments and effective actions. Moreover, concurrent threads are made explicit. The reader is encouraged to use this kind of description.

6.4.2 Application to ResMgr

SyncChart **ResMgr** (Figure 5-6) models the expected behavior of the Resource Manager Controller. Consider the stable configuration {**ResMgr, Wg1, Idle, Idle2**}. What is the reaction for all input signals absent?

Input Signals: **T1**$^-$, **T2**$^-$, **S1**$^-$, **S2**$^-$
Output Signals: **Rn1**$^\perp$, **Rn2**$^\perp$

ResMgr Local Signals: **Rq1**$^\perp$, **Rq2**$^\perp$, **Rl1**$^\perp$, **Rl2**$^\perp$, **G1**$^\perp$, **G2**$^\perp$		
ResMgr_stg_1	**ResMgr_stg_2**	**ResMgr_stg_3**
Current state = **Wg1**	Current state = **Idle**	Current state = **Idle2**
No strong abortion	Test of strong abortion by **Rq1**	Test of strong abortion by **T2**
Reaction of **Wg1**: **Rq1**$^+$	-- *suspend* --	**T2**$^-$, no abortion
Test of weak abortion by **G1**	-- *resume* --	Return **PAUSE**

-- *suspend* --	Strong abortion taken	
	Current state = **s1**	
	First instant: no strong abortion	
	Reaction of **s1**: **G1**$^+$	
-- *resume* --	First instant: no weak abortion	
Weak abortion taken	Return **PAUSE**	
Current state = **Busy1**		
First instant: no strong abortion		
Reaction of **Busy1**: **Rn1**$^+$		
First instant: no weak abortion		
Return **PAUSE**		
ResMgr Return **PAUSE**		

All non input signals not tested during the reaction are set to absent

Figure 6-8: Computation of a Reaction of ResMgr.

Therefore, the reaction is:

$$\{\ \textbf{ResMgr, Wg1, Idle, Idle2}\ \} \xrightarrow[\varnothing]{\{\textbf{Rn1}\}} \{\ \textbf{ResMgr, Busy1, s1, Idle2}\ \}$$

6.5 Summary

This section has introduced the notion of Reactive-Cell, which plays a central role in the semantics of SyncCharts. The full computation of a reaction resorts on many concurrent threads, which suspend their execution when a trigger cannot be evaluated and can resume when new signal statuses are broadcast. This reflects the underlying constructive semantics of SyncCharts: a transition is taken (i.e., a microstep is executed) only when its trigger is *surely* satisfied (no possibility of trial and back tracking).
Strong abortions are easier to understand because there is no need for recursive execution within the aborted macrostate. On the contrary, a weak abortion takes place after the body of the preempted macrostate has been executed, which may entail deep recursions. Another delicate point is that some states can be activated and de-activated during the same reaction.
Sometimes, newly emitted signals are not enough to resume suspended threads. In this case, the knowledge of certainly *not emitted signals* may be used. This information can be derived from the structure of the syncChart, however this a complex process not detailed in this report.

7 Causality Cycle

7.1 Abstract

In a synchronous reaction, emitted signals may participate to the reaction by causing new signal emission. This instantaneous feedback may cause cyclic dependency, leading to incorrect reaction. The example below illustrates such a behavior known as a causality cycle.

7.2 Example of a Causality Cycle

In the Resource Manager Controller studied in Section 5.3.3, suppose we replace the weak abortion transitions triggered by **G** by strong abortion transitions (Figure 7-1):

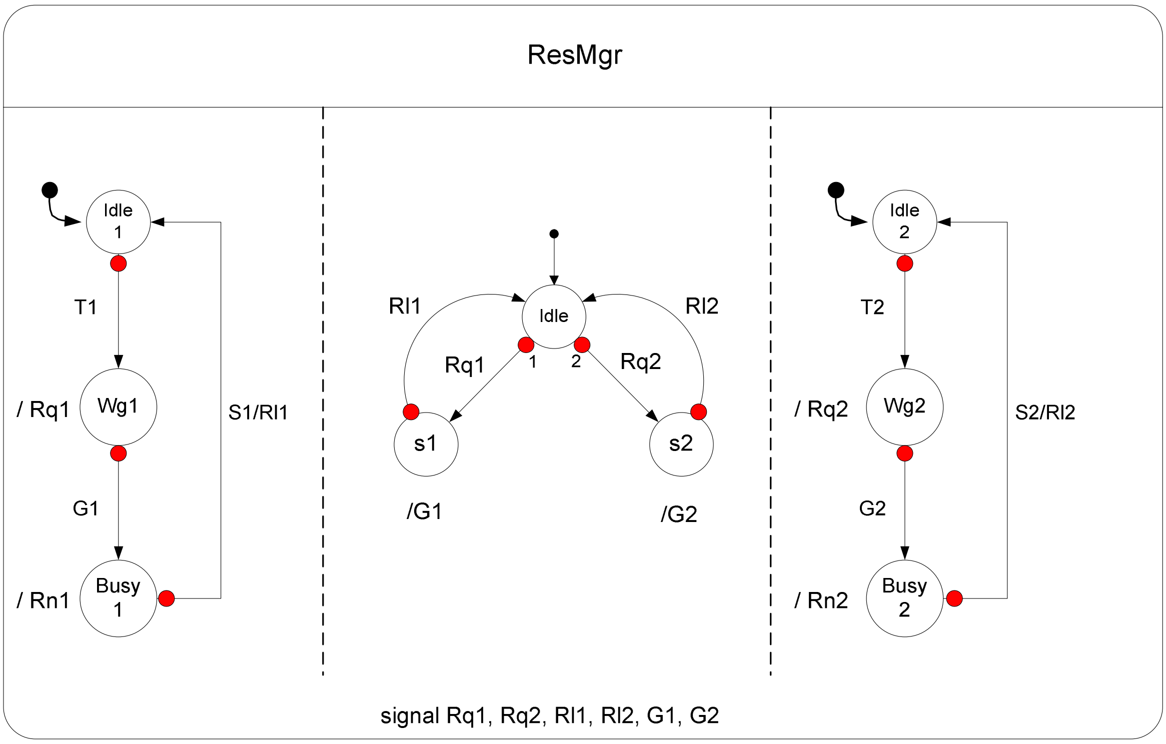


Figure 7-1: An Instance of Causality Cycle.

Starting with configuration = {**Wg1, Idle, Idle2**}, microsteps could be the same as in Figure 5-8. Since **User1** sends **Rq1**, **Arbiter** leaves the **Idle** state, enters state **s1**, and emits **G1**. Now, **G1** being present, state **Wg1** is exited, state **Busy1** is entered, and signal **Rn1** is emitted (configuration = {**Busy1, s1, Idle2**}). Unfortunately, this story is not consistent with the semantics of the strong abortion. The strong abortion of state **Wg1** should have prescribed any execution within **Wg1**. Therefore, **Rq1** should not have been emitted (Figure 7-2-A). This is an example of *causality cycle*: signal **Rq1** by a causality chain generates **G1**, which, in turn, forbids the emission of **Rq1**. The consequence has a direct influence on the cause! Adopting this kind of behavior as a legal one would lead to counter-intuitive semantics. Thus, SyncCharts with causality cycle are rejected as incorrect ones.

On the other side, there is not such a problem with weak abortion (Figure 7-2-B): the weak abortion does not forbid execution of the preempted state.

The causality problem should also be detected by the procedures given in Section 6.3.3. We try to compute the reaction in the same way as in Section 6.4.2

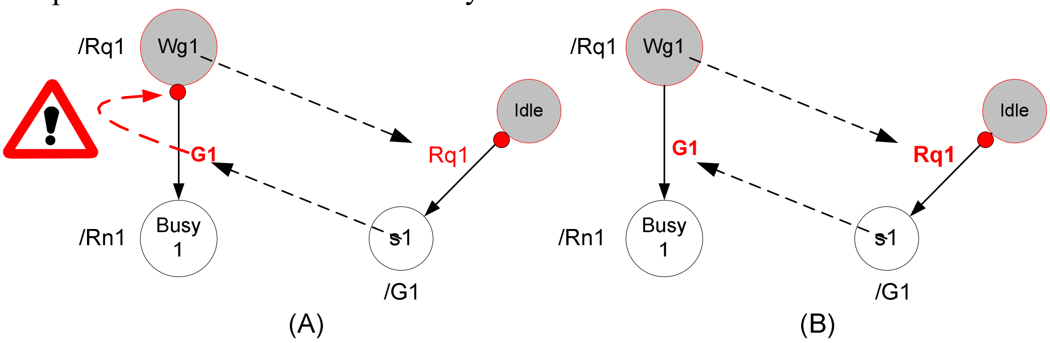


Figure 7-2: Analysis of the Causality Cycle.

Input Signals: **T1**$^-$, **T2**$^-$, **S1**$^-$, **S2**$^-$
Output Signals: **Rn1**$^\perp$, **Rn2**$^\perp$

ResMgr Local Signals: **Rq1**$^\perp$, **Rq2**$^\perp$, **Rl1**$^\perp$, **Rl2**$^\perp$, **G1**$^\perp$, **G2**$^\perp$		
ResMgr_stg_1	**ResMgr_stg_2**	**ResMgr_stg_3**
Current state = **Wg1**	Current state = **Idle**	Current state = **Idle2**
Test of strong abortion by **G1**	Test of strong abortion by **Rq1**	Test of strong abortion by **T2**
-- *suspend* --	-- *suspend* --	**T2**$^-$, no abortion
		Return **PAUSE**

The computation cannot proceed any further: the first two parallel branches are suspended and the third terminated. The only way to resume the computation is to know the presence status of **G1** or **Rq1**. Since no branch is still running, there is no possibility to emit **G1** or **Rq1**. However, if we might be sure that **G1** or **Rq1** or both cannot be emitted during the reaction, then this absence should negatively terminate the test of strong abortion and the computation should resume. Considering the structure of the syncChart and its configuration, signal **Rq2**, for instance, is certainly not emitted in this reaction. Unfortunately, we cannot be so categorical with signals **G1** and **Rq1**. In fact these signals are *potentially emitted signals*, i.e., if the reaction could proceed, then they could be emitted. Thus, the computation must be aborted, and the syncChart rejected. Note that computing potentially emitted signals is a complex task not detailed in this paper.

To sum up, when the computation of a reaction is stuck, with at least a suspended thread, we have to inject information about *certainly absent signals*. “Certainly absent” means that the absence can be proven. If this additional information is not sufficient to resume the computation, then the computation is aborted, the syncChart is said to be *not constructive* and is rejected. The constructive semantics has been defined by Gérard Berry [Berry 1999] for the Esterel language. The compatibility of SyncCharts with Esterel implies that the SyncCharts semantics is also a constructive semantics. A full treatment of this semantics is beyond the scope of this paper.

8 Advanced Constructs

8.1 Abstract

Other discrete state-transition models, like Statecharts, support hierarchy, concurrency and some limited forms of preemption. SyncCharts offer two additional concepts very useful in complex reaction specifications: the immediate transition and the suspension. A third extension concerns Entry and Exit actions. All are presented in this section. The computation of the reaction of a Reactive-Cell is then revisited to integrate them.
At this point the reader knows the essentials of SyncCharts.

Then, miscellaneous features follow. They are given for a second reading of the model: Valued SyncCharts, pre operator, reference macrostate, conditional pseudo-state, and the issue of signal and state reincarnations.

8.2 Immediate transition

Up to now, after entering a state, the state remains active till a *strictly future instant* when the trigger of an outgoing transition is satisfied. A modifier, denoted by a sharp symbol (#), modifies this behavior. When a trigger is prefixed by # (read immediate), the trigger may be satisfied as soon as the state is entered. Thus, with an immediate transition, the trigger is checked for present (immediate) or future satisfaction.

Using immediate transitions avoids delays in reactions. For instance, in the Resource Manager, we showed (Figure 5-7) that starting with configuration {**Wg1, s2, Busy2**}, when **S2** became present, configuration {**Busy1, s1, Idle2**} was reached in two reactions (two instants). Now considering immediate transitions from **Idle** to **s1** and to **s2** (Figure 8-1) results in a unique reaction from configuration {**Wg1, s2, Busy2**} to configuration {**Busy1, s1, Idle2**}. In this case, state **Idle** is ``bypassed'' during the reaction. Note that **Idle** is a genuine state, because under other circumstances (**Rl2** present but **Rq1** and **Rq2** absent) state **Idle** may stay active.

The immediate abortion is a powerful construct that eliminates unnecessary transient states during a reaction. The difference between weak and strong immediate abortions must be well understood. An immediate weak abortion may activate, execute and then de-activate a state during a reaction (see microsteps of the syncChart in Figure 8-2 when **a** and **b** are both present). Signal **Y**, which is the effect associated with state **q**, is emitted during the reaction. On the contrary, an immediate strong abortion (Figure 8-3) forbids any execution in the (immediately) preempted state, thus **Y** is not emitted during the reaction.

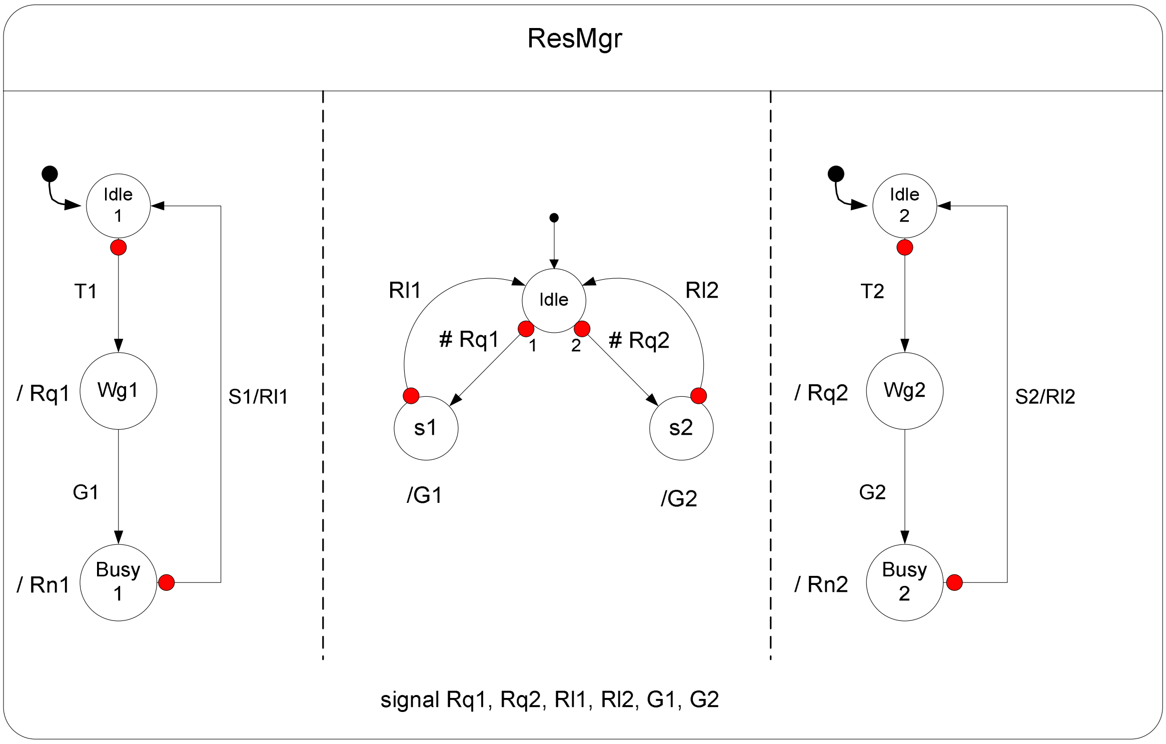


Figure 8-1: Controller of the Resource Manager using immediate transitions.

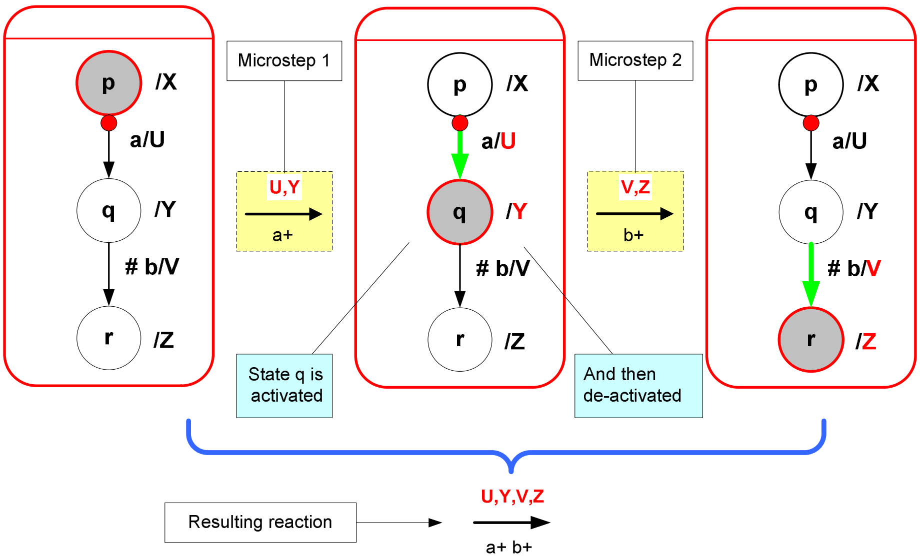


Figure 8-2: Immediate Weak Abortion.

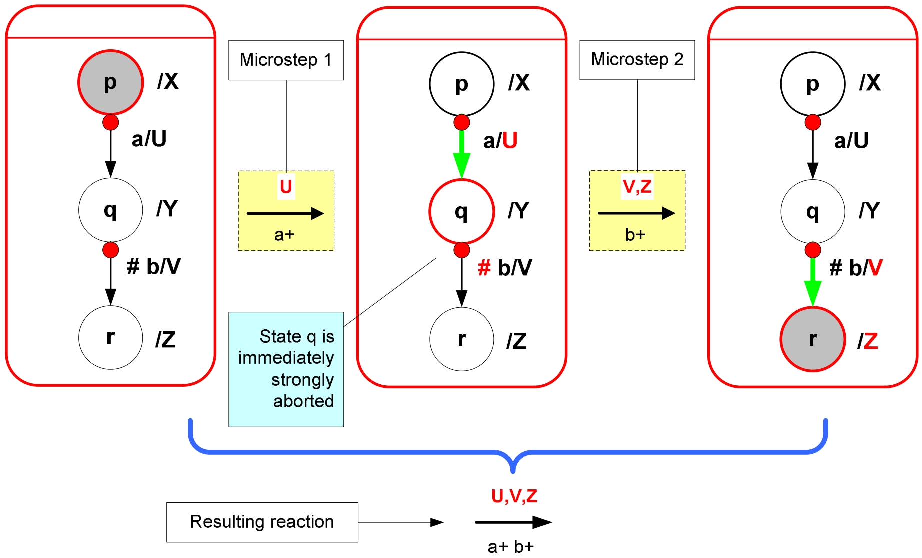


Figure 8-3: Immediate Strong Abortion.

8.3 Suspension

Suspension is a form of preemption. Contrary to abortion that forbids future execution, suspension only "freezes" the execution of the preempted state. Graphically a suspension appears as a "lollypop" labeled by a trigger. Whenever the trigger is valid, the reaction is suspended in the target state.

Suppose that the resource is accessed through a shared bus. When signal **D** is present a DMA steals bus cycles, so that **User1** can not effectively use the resource and signal **Rn** is not emitted while the DMA takes place. Figure 8-4 shows the modified STG for the User Dialog Controller.

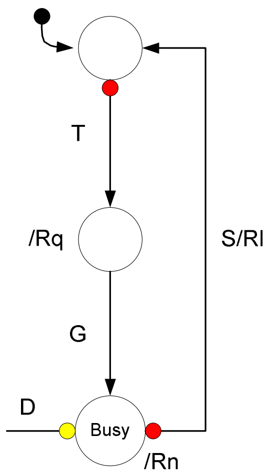


Figure 8-4: Suspension.

Suspension generally applies to macrostate, so that complex behaviors can be suspended by a single signal. For instance, the activity of the binary counter (Section 5.3.1) may be suspended by signal **inhib** (Figure 8-5). Whenever **inhib** is present, a possible presence of **T** is ignored, and neither **B0** nor **B1** can be emitted. As soon as **inhib** is no longer present, the counter resumes its activity. Note that the suspension does not prevent abortions. If **reset** and **inhib** are simultaneously present, the strong abortion is taken, and the initial configuration is reached ({**Cnt2withSuspension**, **Cnt2**, **off1**, **off2**}). Of course, a normal termination of a state cannot occur when the state is suspended.

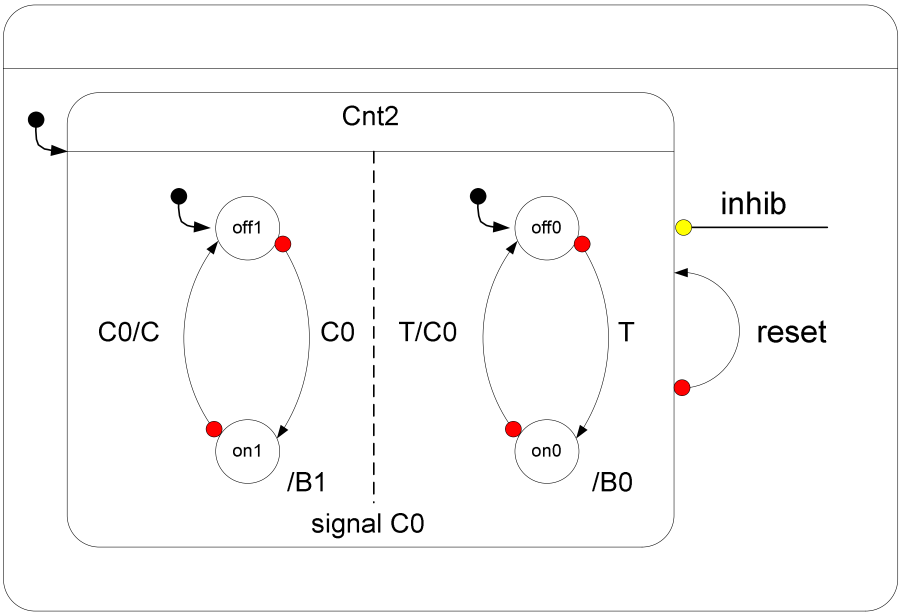


Figure 8-5: 2-bit Counter with Suspension and Reset.

As with abortion, a suspension is not effective when the state is entered: only a strict future satisfaction of the trigger will suspend the state. This behavior can be changed by the immediate modifier. In this case, the state is entered but its body is frozen.

Figure 8-6 is an example using immediate suspension. This example mimics a classical interruption mechanism. **irq** is the signal that requests interruption of a complex behavior encapsulated in the macrostate **aTask**. **ISR** is the Interrupt Service Routine. When the ISR terminates (normal termination on the **ISR** macrostate) the activity of **aTask** resumes. If **irq** is present when ISR terminates, then the ISR is instantaneously re-entered and **aTask** does not resume. Note that there is no need for context saving when **aTask** is suspended: it is only frozen.

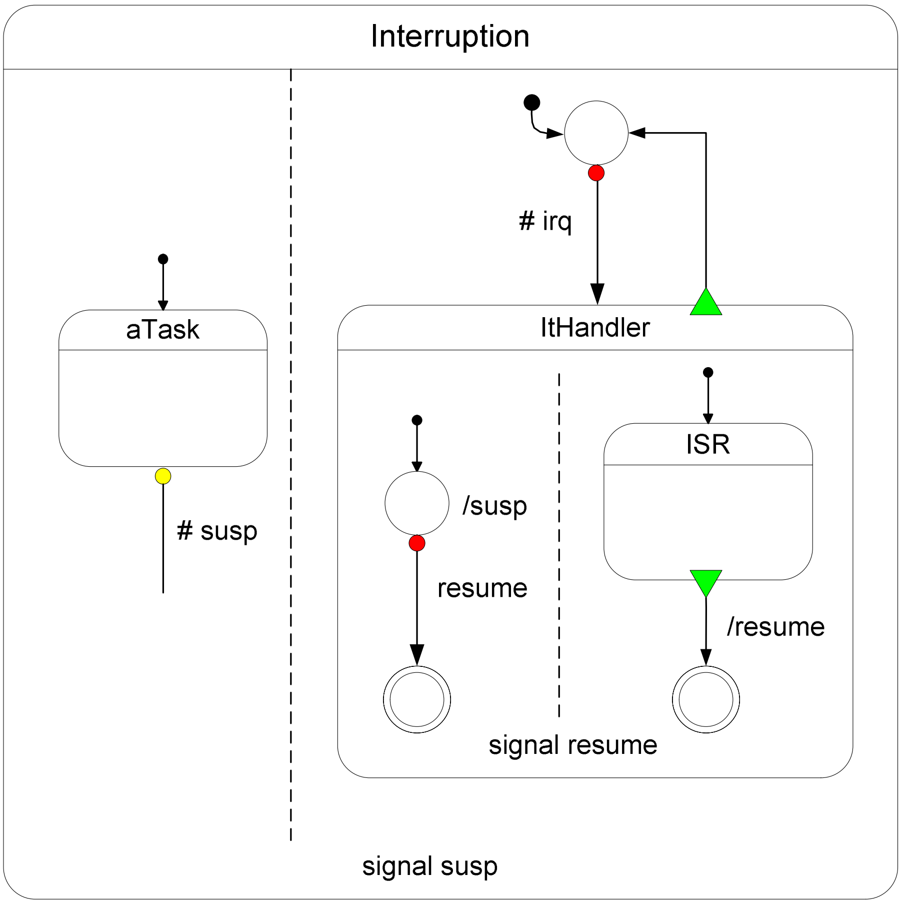


Figure 8-6: Interruption Mechanism.

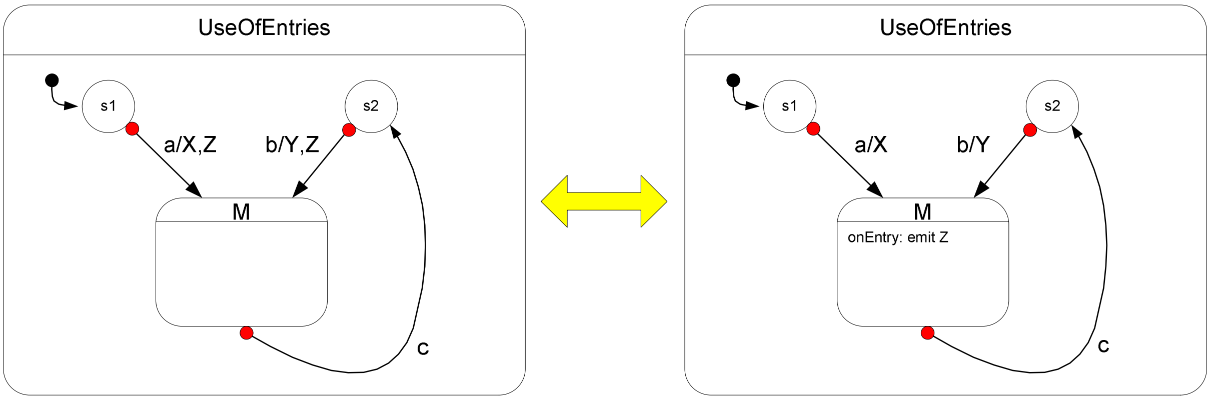
8.4 Entry and Exit Actions

Entering and exiting states play a central role in the SyncCharts semantics. In SyncCharts, there is a possibility to execute instantaneous actions when entering or exiting a macrostate. For pure SyncCharts, instantaneous action can only be signal emitting.

8.4.1 Entry Actions

Macrostate **M** in Figure 8-7-A can be entered coming from either **s1** or **s2**. In both cases, signal **Z** is emitted. The macrostate shown in Figure 8-7-B has the same behavior. Now, emitting **Z** is done when entering macrostate **M**. The actions to do when entering the macrostate are written as Esterel statements prefixed by the keyword onEntry. In this example the action is a simple signal emission, but any instantaneous Esterel statement can be used as well.

Entry actions can be seen as a kind of factorization. They are not strictly necessary, and yet they are advisable because they are applications of the WTO (Write Things Once) principle, already illustrated in Section 5.3.4.



(A) (B)

Figure 8-7: Entry Actions.

8.4.2 Exit Actions

Contrary to entry actions, exit actions are not simple factorizations of instantaneous actions. The idea is to execute instantaneous action(s) whenever a macrostate is exited. A macrostate can be exited for various causes illustrated in Figure 8-8:

- Normal termination (e.g., macrostate **M10**);
- Strong or weak abortion (e.g., macrostate **M2** strongly preempted by the transition whose trigger is **b**);
- Abortion of a containing macrostate (e.g., macrostates **M2**, **M10**, and **M11** preempted by the strong abortion of macrostate **M0** by signal **R**).

The actions to do when exiting the macrostate are written as Esterel statements prefixed by the keyword onExit. These exit actions are performed before taking the transition. When macrostates with exit actions are nested, the exit actions are executed in the innermost to outermost order. Note that strong and weak abortions have the same effect on exit actions. This explains why exit actions are primitive constructs: they cannot be expressed by a combination of the already studied constructs.

Here are some reactions with exit actions:

$\{\textbf{Exits}, \textbf{M0}, \textbf{M10}, \textbf{M2}, \textbf{M11}\} \xrightarrow[\textbf{b+}]{\textbf{Y2, X2, Y1, X11}} \{\textbf{Exits}, \textbf{M0}, \textbf{done}, \textbf{M11}\}$

$\{\textbf{Exits}, \textbf{M0}, \textbf{done}, \textbf{M11}\} \xrightarrow[\textbf{R+}]{\textbf{Z, Y0, X0}} \{\textbf{Exits}, \textbf{M0}, \textbf{M10}, \textbf{M2}, \textbf{M11}\}$

$\{\textbf{Exits}, \textbf{M0}, \textbf{M10}, \textbf{M2}, \textbf{M11}\} \xrightarrow[\textbf{a+}]{\textbf{Y2, Y1, X10}} \{\textbf{Exits}, \textbf{M0}, \textbf{done}, \textbf{M11}\}$

$\{\textbf{Exits}, \textbf{M0}, \textbf{M10}, \textbf{M2}, \textbf{M11}\} \xrightarrow[\textbf{R+}]{\textbf{Y2, Z, Y1, Y0, X0}} \{\textbf{Exits}, \textbf{M0}, \textbf{M10}, \textbf{M2}, \textbf{M11}\}$

The microsteps of the last reaction are given in Figure 8-9, Figure 8-10, and Figure 8-11. For simplicity, the top macrostate is not represented in these pictures.

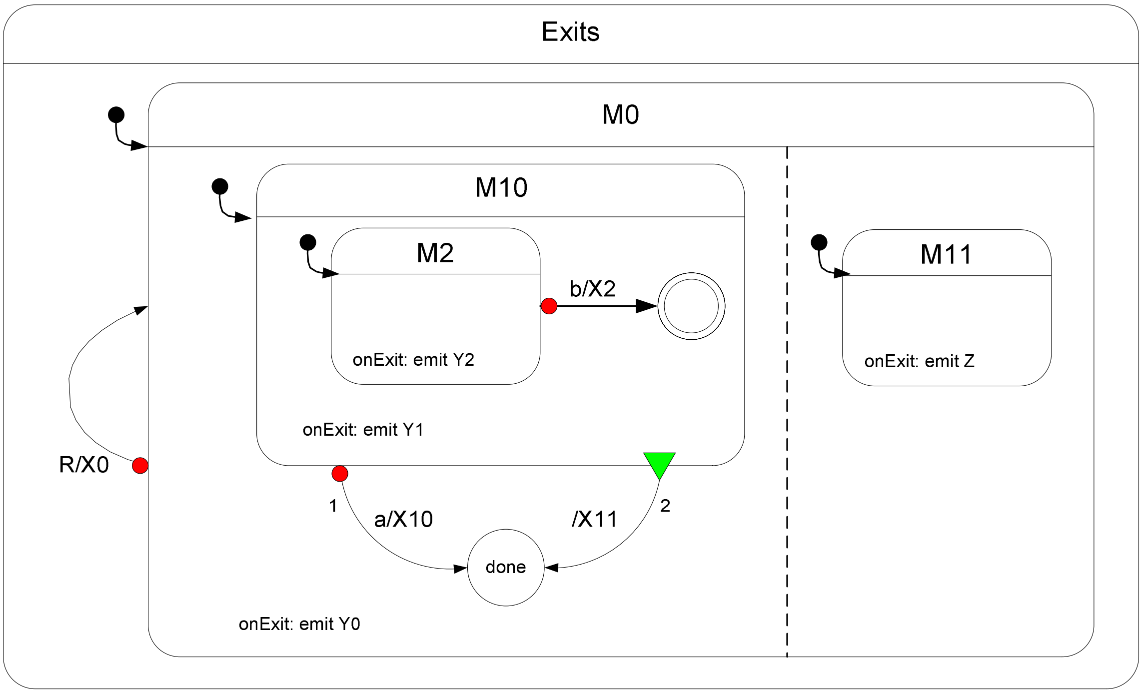


Figure 8-8: Exit Actions.

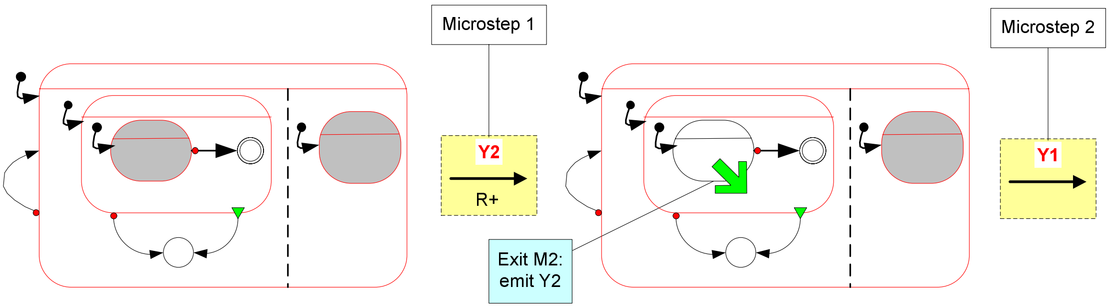


Figure 8-9: Microsteps of a reaction with exit actions (1).

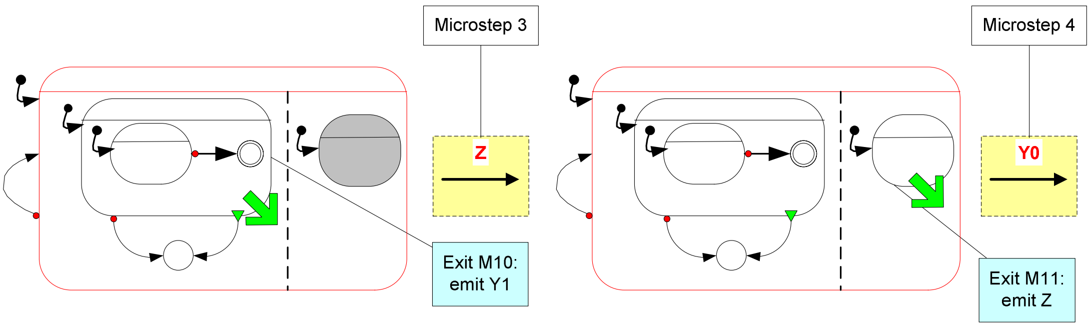


Figure 8-10: Microsteps of a reaction with exit actions (2).

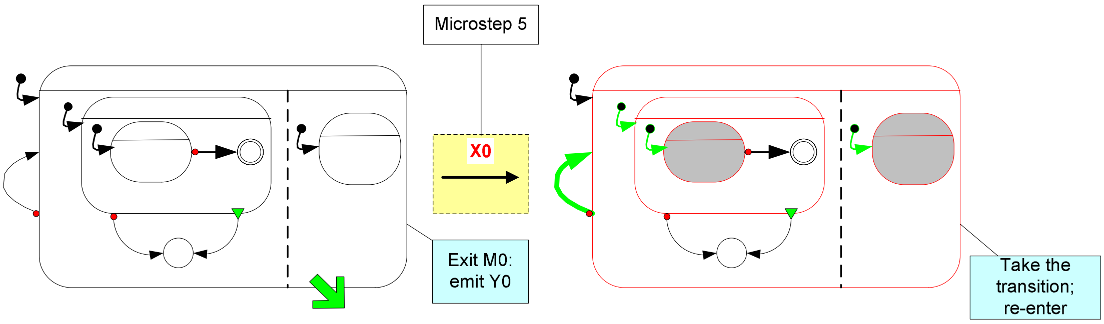


Figure 8-11: Microsteps of a reaction with exit actions (3).

8.5 Computation of a Reaction (Revisited)

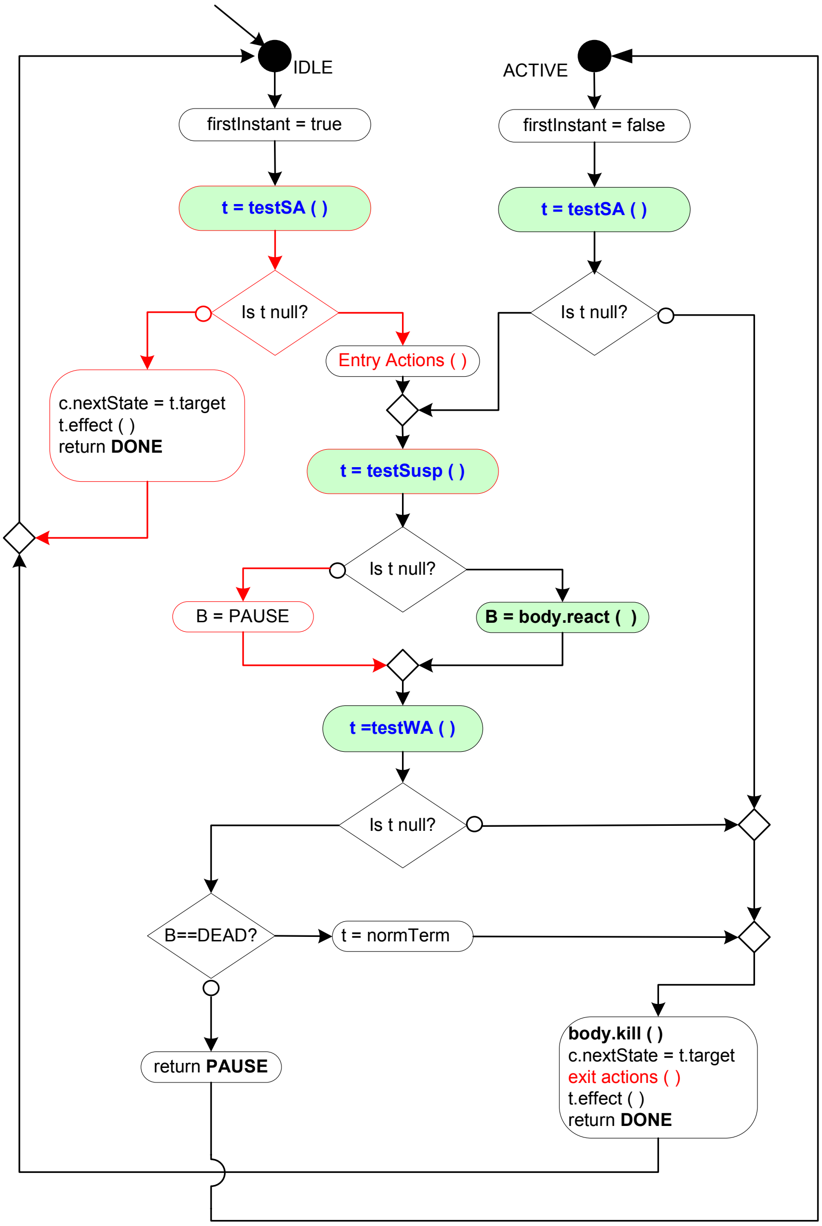


Figure 8-12: Reaction of a Reactive-Cell (extended version).

The computation of a reaction given in Section 6.3.3 has to be extended to support immediate preemptions, suspensions, and entry/exit actions. These new features affect only the reaction of a Reactive-Cell. Figure 8-12 is the new version. New elements in the diagram are colored red.

Comments

Immediate abortions may cause instantaneous abortion of the state. In this case the status stays **IDLE**, the transition is taken, **DONE** is returned: the state is "by-passed". If there is not an immediate strong abortion, then the state is effectively entered, and the *entry actions* are performed.
Function **testSusp ( )** checks for a possible *suspension* of the body of the macrostate. If the body must be suspended, it is not executed at all, and **B** is set to **PAUSE**, saying that there is nothing left to do in the macrostate at the current instant. If the body is not suspended, it reacts in the usual way, returning its termination code in **B**. The check for weak abortion must be done even at the first instant (possible immediate weak abortion).
When the state is exited (rounded rectangle in the right bottom corner of the diagram), *exit actions* are performed before taking the transition.

The **kill( )** function is also adapted: it recursively calls the kill function on its components, then performs its possible exit actions, and finally de-activates the caller.

8.6 Valued SyncCharts

Valued Signals

In the examples so far, signals do not convey a value. Since signals are the support for synchronization and communication, a value of a given type can associated with a signal. *Valued signals* are strongly typed.
For input signals the value of a signal is assigned by the environment, for all other signals, the value is given by a reaction. In any case, the value of a signal can change only when the signal is present.
The declaration **S:T** declares a valued signal **S** whose values are of type **T**. The variant **S:=t:T** declares signal **S** of type **T** with **t** as its initial value. In the absence of initialization, the value is undefined (⊥). Let **t** be an expression whose type is **T**, action **S(t)** emits signal **S** conveying the value of **t**. The value of signal **S** is accessed by **?S**. The value is *persistent*, that is a signal keeps the value assigned during the last presence instant. Thus, the value of a signal seems to behave like a variable in imperative languages. This interpretation forgets that SyncCharts are instant-based models. At each instant, a valued signal should have *one and only one value* (possibly ⊥ if not initialized and not yet emitted). So, what if a signal is emitted several times during a reaction? The answer is that multiple simultaneous emissions are forbidden, except for especially declared signals. The latter are called *combined valued signals*, the former *single valued signals*.
S: combine **T** with **f** declares a combined valued signal **S** of type **T**, with a combination binary function or operator **f**, which must be commutative and associative. **S**:= **t**: combine **T** with **f** is a variant with initialization.
For instance, consider a local signal **S** defined in a macrostate by

S:=3:combine **integer** with **+**.

A possible history for S is:

instant	Emissions of **S**	Value of **S**
k: entering the macrostate	none	3
k+1	none	3
k+2	**S**(5)	5
k+3	none	5
k+4	**S**(2), **S**(4), **S**(1)	7
k+5	none	7
k+6	**S**(0)	0

Other logical objects

Like Esterel, SyncCharts support types, functions, procedures, tasks,... The ready may refer to the Esterel's Primer [Berry 1997] for their definitions, which have been imported in SyncCharts without any change. We just explain the notion of variable used in examples below.

Variables are assignable objects that have a name and a type. The declaration of a variable is as follows: var **v**: **T** or the initialized variant var **v**:=**t**: **T**, where **v** is an identifier, **T** a type, **t** a value of type **T**. Concurrent writing of a variable are forbidden. So, the natural scope of a variable is an STG. Contrary to signals, a variable may hold different values during a reaction. This poses no problem with our semantics, which rejects possible concurrent writing of a variable.
To avoid the issue of concurrent writing, it is advisable to use local signals instead of variables whenever possible. The memorizing role of a variable can now be played by a valued signal using the pre operator (see Section 8.8).

Guard

In Valued SyncCharts, a transition may be *guarded*. The more general form of the label associated with a transition is *trigger* [*guard*] / *effect*. The guard is an expression that evaluates to true or false. This expression may use signal and variable values, operators, functions,...
The guard is evaluated when and only when the trigger is satisfied. If the guard is true, then the transition is *enabled*. In Pure SyncCharts, transition enabling and trigger satisfaction seem to be interchangeable concepts, whereas in Valued SyncCharts the enabling implies the satisfaction, not the converse. Thus, we cannot spare a concept: the satisfaction of a trigger is distinct from the enabling of the transition.

Remark on Count Delays

Triggers for preemptions are either simple (a single signal) or complex (combination of signals with and, or, and not operators) (Section 4.4.1). Instead of waiting for the next satisfaction of a trigger, one may wait for the nth next satisfaction. This is expressed by an integer factor written before the signal expression. For instance, 3 **S** waits for the third strictly future presence of **S**; 5 [**S** and not **T**] waits for the fifth strictly future satisfaction of the

conjunction of **S** present and **T** absent. Such triggers are called *count delays*. In order to avoid ambiguity, immediate count delays are not accepted.
Pure SyncCharts (using only pure signals) may have delays with constant counts known at compile-time. Valued SyncCharts may have integer expressions as counts for delays. The expression is then evaluated only once when the delay is initiated. If the (run-time) result is 0 or less, it is set to 1.

8.7 Reference Macrostate

A syncChart may use several instances of a macrostate defined elsewhere as another syncChart. Instead of in-line insertions of the macrostate, a better practice is to use *reference macrostates* (denoted with an @). For instance, the syncChart in Figure 8-13 specifies the behavior of a T Flip-Flop. The top macrostate of **Toggle** is instantiated 4 times in the 4-bit counter (Figure 8-14). The interface objects of the reference macrostate can be renamed when instantiated (e.g., **Cell0@Toggle**[signal **Tog/T**, **C0/C**, **B0/ON**] denotes an instance of macrostate **Toggle** in which the interface signals **T**, **C**, and **ON** are respectively renamed as **Tog**, **C0,** and **B0**. The instance itself is named **Cell0**. Other interface objects like type, functions,... can also be renamed.
Using reference macrostate is encouraged because it adheres to the WTO principle. A reference macrostate need not be as simple as the **Toggle** example. Moreover, a reference macrostate can contain other reference macrostate: for instance, the 4-bit counter might be defined as a reference macrostate.

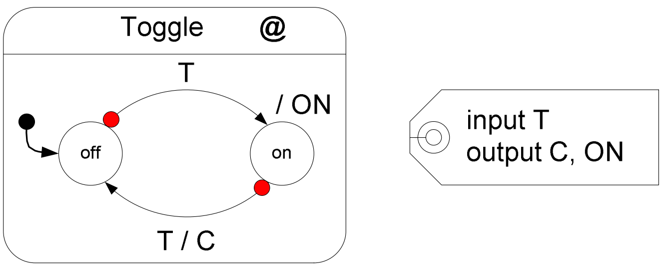


Figure 8-13: SyncChart used as a Reference.

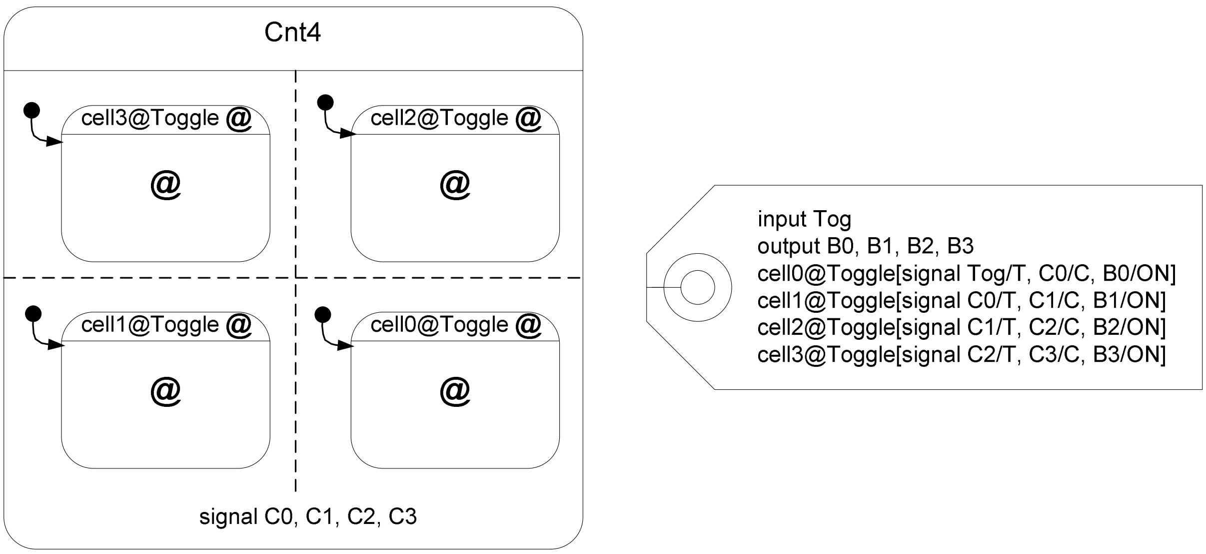


Figure 8-14: 4-bit Counter using Reference Macrostates.

8.8 Pre

Delaying signal occurrence

According to the synchronous semantics, the signal presence is a not persistent information, valid only at the current time. Sometimes, for instance to break a causality cycle, the effects of the presence of a signal have to be deferred to the next instant. The macrostates **Pre** and **ValuedPre** (Figure 8-15) offer this behavior in the case of pure signal for the former and of valued signal for the latter. State **wait** is exited as soon as **S** is present. The valued version memorizes the value of **S** in a variable (effect **vS** := **?S**, written between two back quotes). State **pause** has a trigger-less outgoing transition, so that at the next instant the transition is taken, and signal **preS** is emitted. For the valued version, **preS** is emitted with the value memorized in **vS**. The target of the transition is state **wait**. Because of the immediate abortion of **wait** by **S**, this state becomes active only if **S** is absent. Otherwise **wait** is by-passed, and **pause** is the new active state. See Figure 8-16 for an execution trace of **ValuedPre**. The third reaction is explained at the microstep level in Figure 8-17. This reaction illustrates the fact that a signal may have several values during a reaction: in the first microstep **vS** is equal to 3 (the memorized value), and during the second microstep value 5 is assigned to **vS**.
Note that if the preemption of **wait** is not immediate, occurrences of **S** may be lost.

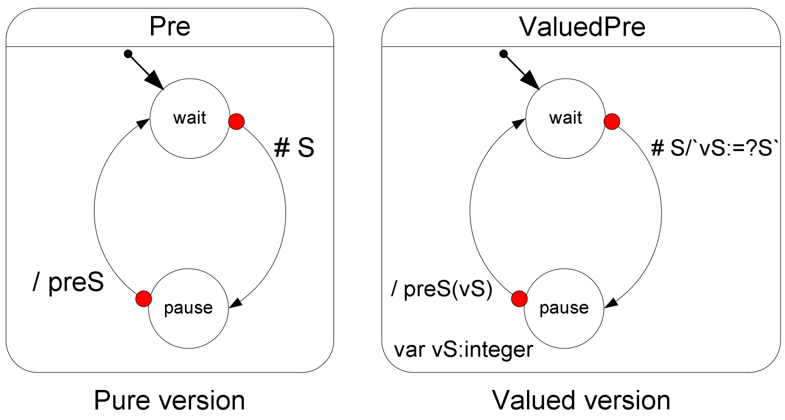


Figure 8-15: Macrostates Pre and ValuedPre.

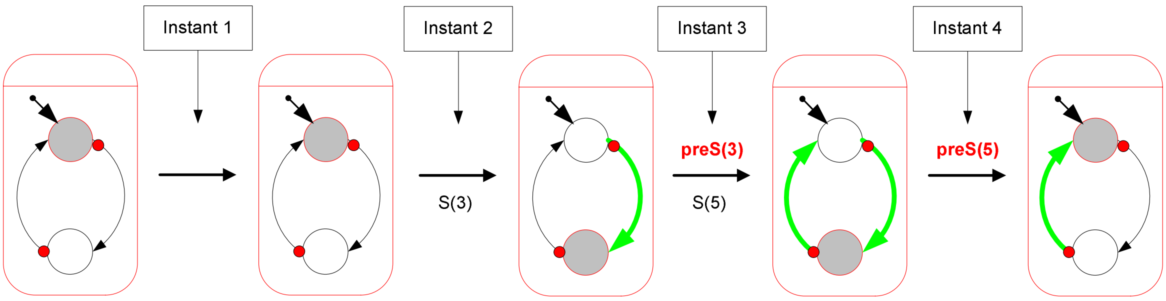


Figure 8-16: An Execution Trace of Pre.

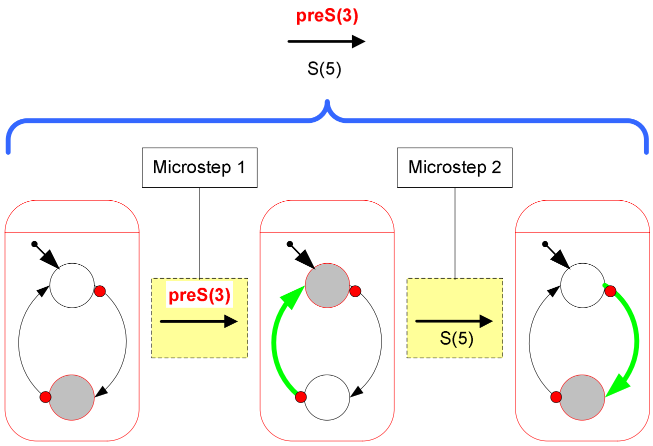


Figure 8-17: Microsteps in a Reaction of Pre.

Pre operators

Esterel in version 5.91 introduced new operators pre: pre(**S**) gives the presence status of signal **S** at the previous instant; pre(**?S**) returns the value of **S** at the previous instant. SyncCharts have adopted these operators whose implementation is more efficient than their equivalent macrostate representation (Figure 8-15). When entering the scope of a signal **S**, pre(**S**) is absent, and pre(**?S**) has the same value as **S**, if this value is defined, and ⊥ otherwise.

Examples with pre

FilteredSR (Figure 8-18) derives from a classical SR Flip-Flop. Its inputs are "filtered": an isolated presence of **S** or **R** is not sufficient to trigger a change of state: the presence must be confirmed at the next instant. Thus, instead of a simple trigger **S**, the transition from state **off** to state **on** is triggered by **S** and pre(**S**), that is, **S** is present and was present at the previous instant.

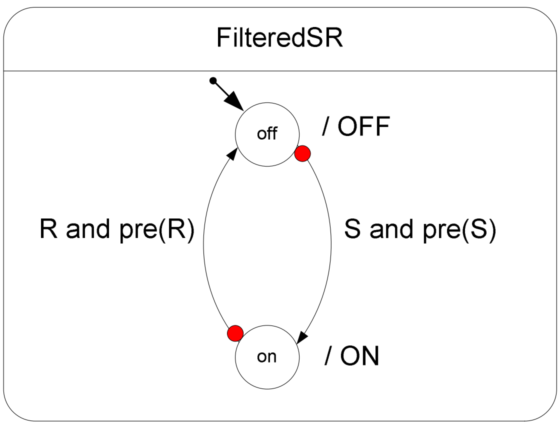


Figure 8-18: Filtered SR Flip-Flop.

Shifter3 (Figure 8-19) is an example using pre with valued signals. Whenever the input signal **I**:integer is present with a value **v**, signal **O**:integer will be emitted 3 instants later with value **v**. An execution trace is given in Table 8-1. The value of the signal is written between brackets, + denotes the presence, - the absence.

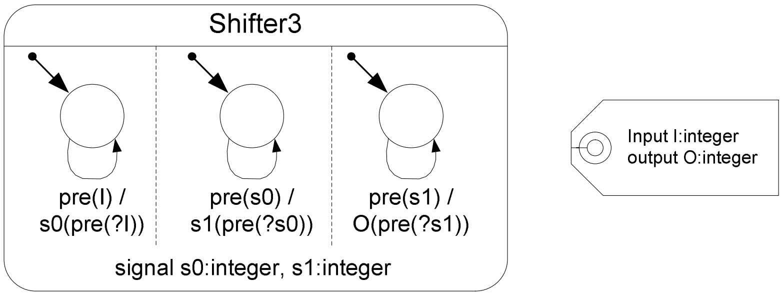


Figure 8-19: Shift Register.

instant	1	2	3	4	5	6	7	8
I	(⊥)-	(1)+	(2)+	(2)-	(3)+	(3)-	(3)-	(4)+
s0	(⊥)-	(⊥)-	(1)+	(2)+	(2)-	(3)+	(3)-	(3)-
s1	(⊥)-	(⊥)-	(⊥)-	(1)+	(2)+	(2)-	(3)+	(3)-
O	(⊥)-	(⊥)-	(⊥)-	(⊥)-	(1)+	(2)+	(2)-	(3)+

Table 8-1: An Execution Trace for Shifter3.

Local signal, pre and Suspension

Operators pre are sometimes misunderstood. pre(**S**) refers to the presence status of **S** in the previous instant when the scope of the signal was active; this is not necessarily the previous (absolute) instant. The syncChart in Figure 8-20 illustrates this situation.
Macrostate **Mod3Cnt** specifies the behavior of a modulo 3 binary counter. The carry signal **C** triggers a delayed abortion of macrostate **Cnt**. Since transitions of the right-hand STG in **Cnt** are trigger-less, the counter progresses at each instant. Suspending the evolutions of **Mod3Cnt** when signal **T** is absent makes a counter driven by **T**. Note that, according to the semantics of the suspension, **Mod3Cnt** executes at the first instant whatever the presence status of **T**.

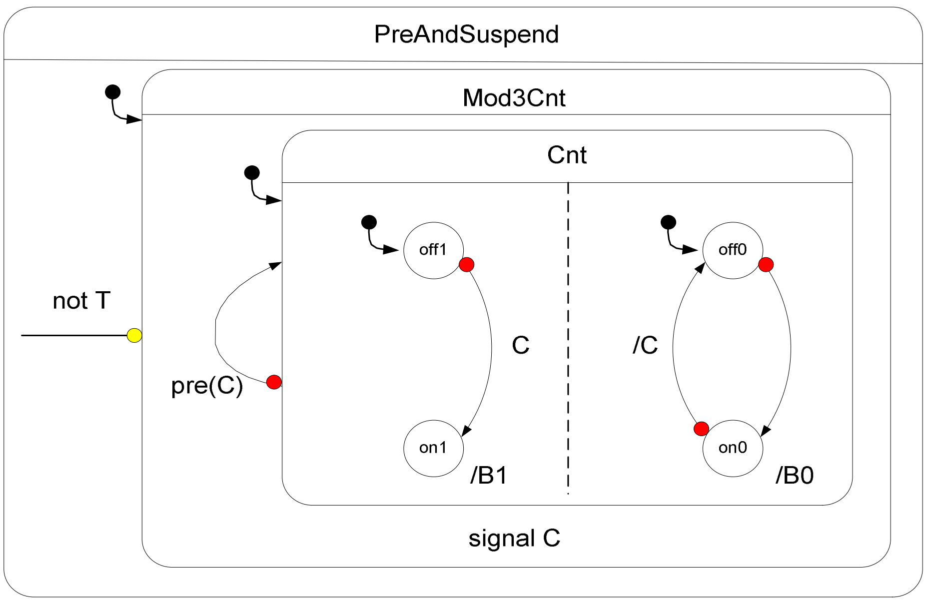


Figure 8-20: Local Signal, Suspension, and pre.

Table 8-2 contains an execution trace of the syncChart. **C** is local to macrostate **Mod3Cnt**. When **T** is absent, the body of Reactive-Cell **Mod3Cnt** is not executed (see Figure 8-12), and thus, time is frozen for signal **C** (Colored entries in the **C** row). Thus the absolute instant 6, is only the fourth instant with respect to **C**, and pre(**C**) at the absolute instant 6 is the presence status of **C** at the absolute instant 4 (i.e., present). Considering **C** absent at the absolute instant 5 will be a mistake, **C** just does not exit at this instant!

Instant	1	2	3	4	5	6	7	8	9	10	11	12	13
T	-	+	-	+	-	+	-	-	+	-	+	+	-
B0	-	+	-	-	-	-	-	-	+	-	-	-	-
B1	-	-	-	+	-	-	-	-	-	-	+	-	-
C	-	-		+		-			-		+	-	

Table 8-2: An Execution Trace of PreAndSuspend.

8.9 Conditional Pseudo-state

Sometimes a common trigger is shared by several outgoing transitions. Figure 8-21-A shows such a case. This syncChart is a variant of the **Arbiter**. It applies a turning priority policy: the last user is given a lower priority. Consider state **s1** active, which means that the resource is granted to User1. There exist two transitions to exit this state, respectively triggered by **Rl1** and **Rq2**, and **Rl1**. The former has priority over the latter. Both are triggered by **Rl1**, which indicates that User1 has just released the resource. This event is the primary cause of the preemption of state **s1**. The presence of **Rq2**, which is associated with a pending request from User2, enables the former transition, whereas its absence enables the latter.
Figure 8-21-B introduces a new notation that clearly shows the trigger common to several transitions (**Rl1**) and then the selecting triggers or guards. The intermediate node is a conditional pseudo-state (a grey circle with an inscribed C). Since a pseudo-state is not a state, it cannot be active. When a transition entering the pseudo-state is taken, there must always be an enabled transition leaving the pseudo-state. A good practice is to use an outgoing transition without trigger and guard as a “catch-all” transition. This transition is given the lowest priority, and it is taken when all the other transitions are disabled.

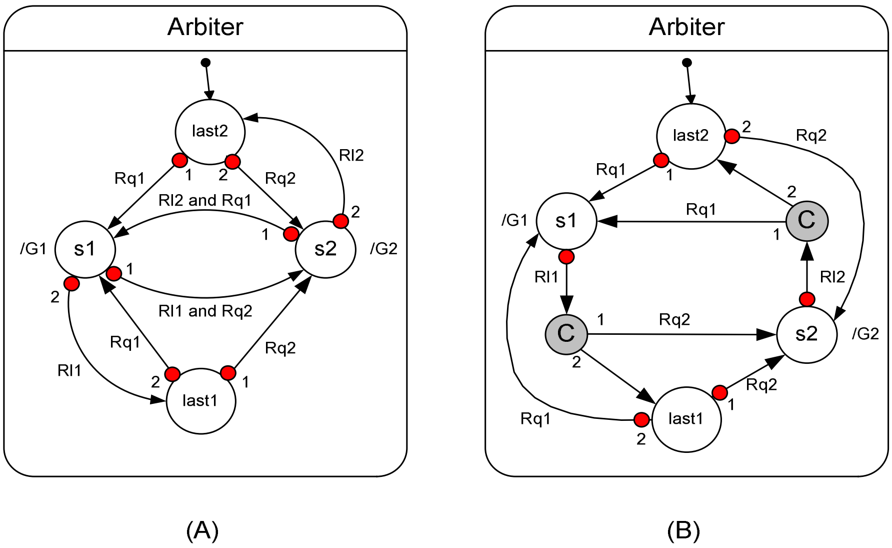


Figure 8-21: Arbiter with Turning Priority.

The two syncCharts in Figure 8-21 have the same behavior. Conditional pseudo-states do not increase the expressiveness of SyncCharts. They only make some charts more readable.
Remark: possible triggers on transitions from a conditional pseudo-state are implicitly immediate.

8.10 Reincarnation

A Simple Signal Reincarnation Example

A local signal has a well-defined scope: the macrostate in which it is declared. A loop may provoke the simultaneous existence of two different "incarnations" of a local signal. Figure 8-22 illustrates this situation.
The only place where local signal **S** can be emitted is the transition from state **q** to state **r**. This transition cannot be enabled at the initial instant, therefore **S** is absent. Macrostate **Reincarnation** is entered and since **S** is absent, the transition leading to state **q** is taken. The control stays in state **q** until a future occurrence of **A**. As soon as **A** is present the transition to state **r** is taken, and signal **S** is emitted. Now, **r** being a final state, the normal termination is taken and macrostate **Reincarnation** is re-entered. Instantaneously the presence of **S** is checked to choose between the two outgoing transitions of the conditional pseudo-state. Surprisingly, the transition to state **p** is not taken; the transition to **q** is taken instead. The reason for this is that a fresh instance of **S** has been created when entering macrostate **Reincarnation**. Since there is no way to emit signal **S** from the initial state, the new instance of **S** is absent. This presence status is independent from the presence status of the former instance. Figure 8-23 is a possible execution trace. Figure 8-24 contains the microsteps executed during the third instant (reincarnation).

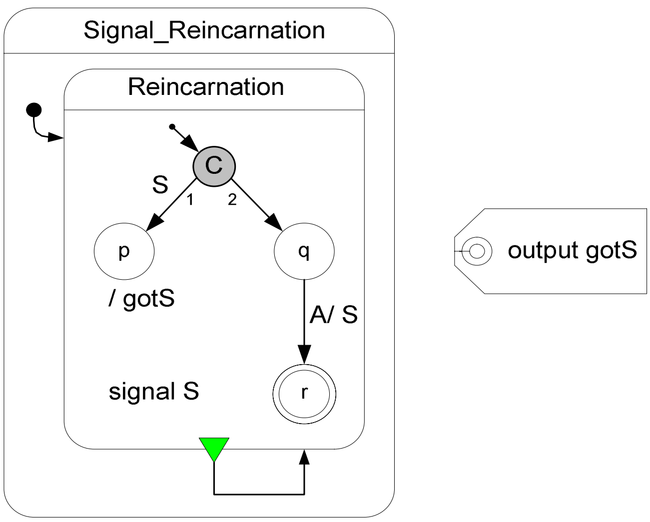


Figure 8-22: Signal Reincarnation.

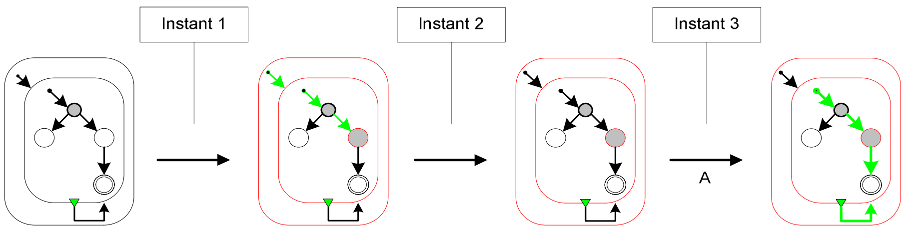


Figure 8-23: An execution trace of Signal_Reincarnation.

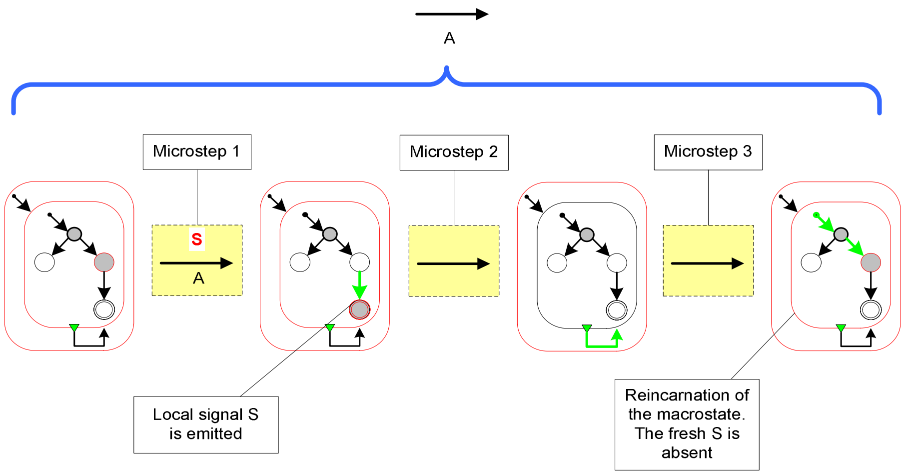


Figure 8-24: The microsteps of the third reaction.

Nested State Reincarnation

Loop, immediate preemptions, and priority can lead to amazing, but perfectly consistent behaviors. Figure 8-25 shows an example especially devised for illustrating these complex interactions.

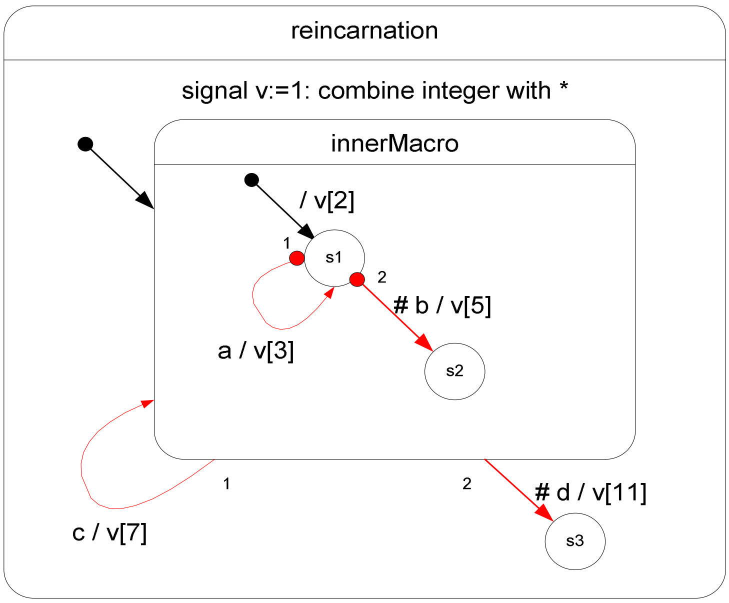


Figure 8-25: Nested Reincarnations.

Signal **v** is a combined integer signal with the multiplication as its 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.
Consider the configuration {**reincarnation**, **innerMacro**, **s1**}, and signals **a**, **b**, **c**, and **d** present. The reaction emits **v** with the value 11550 = $2\times3\times5^2\times7\times11$. The new configuration is {**reincarnation**, **s3**}. This reaction is explained as follows:

1. Macrostate **innerMacro** must be weakly aborted by the transition whose trigger is **c**, which has priority over the one triggered by **d**. Before firing the weak abortion transition, the body of the macrostate must be executed.
2. Reaction of the body of **innerMacro**: **s1** is strongly aborted by the transition triggered by **a**, which has priority over the transition triggered by **b**. While taking the transition, signal **v** is emitted with 3 for value.
3. State **s1** is the target of the transition, so **s1** is re-entered. In fact, this is a *fresh* instance (re-incarnation) of **s1**.
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**.
7. The weak abortion triggered by **d** is to be taken, but before, the inside of **innerMacro** must react.
8. The execution of **innerMacro** starts with emitting **v** with value 2 (initial arc) and enters state **s1**.
9. 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.
10. 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.
11. State **s3** is activated, and the reaction stops.

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

9 References

[André 1996a] Charles ANDRÉ, “Representation and Analysis of Reactive Behaviors: A Synchronous Approach”, Computational Engineering in Systems Applications (CESA), Lille (F), July 1996. Publisher: IEEE-SMC, pp 19–29.

[André 1996b] Charles ANDRÉ, “SyncCharts: a Visual Representation of Reactive Behaviors”, I3S Research Report # 96.56, Sophia Antipolis (F), April 1996.

[André et al. 2202] Charles ANDRÉ and Jean-Paul RIGAULT, “Variations on the Semantics of Graphical Models for Reactive Systems”, SMC'02, Hammamet (TN), October 2002. in IEEE Press, ISBN: 2-9512309-4-x, CD-ROM index TA2L2.

[Berry 1997] Gérard BERRY, “The Esterel v5 Language Primer”, 1997. (Revision v5_91, August 2000). Available on the web: http://www.esterel-technologies.com.

[Berry 1999] Gérard BERRY, “The Constructive Semantics of Pure Esterel”, 1999. (Version 3, July 1999). Available on the web: http://www.esterel-technologies.com.

[Berry 2000] Gérard BERRY, “The foundations of Esterel”. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, MIT Press, 2000.

[Boussinot and De Simone 1991] Frédéric BOUSSINOT, Robert De SIMONE, “The ESTEREL Language. Another Look at Real Time Programming”, Proceedings of the IEEE, 79:1293–1304, 1991.

[Douglass 2003] Bruce P. DOUGLASS, “Real-Time Design Patterns”, Object Technology Series, Addison-Wesley, 2003.

[Harel 1987] David HAREL, “Statecharts: A Visual Approach to Complex Systems”, Science of Computer Programming, 8:231–274, 1987.

[Harel and Naamad 1996] David HAREL and Amnon NAAMAD, “The Statemate Semantics of Statecharts”, ACM Trans. Soft. Eng. Method. 5:4, October 1996.

[Katz 1994] Randy H. KATZ, “Comtemporary Logic Design”, Benjamin/Cummings Publishing Company, Inc., 1995

10 Annex

10.1 Esterel-Studio notations

SyncCharts used as an input format in Esterel-Studio have a format slightly different from the one used in this paper. The following pictures show the correspondence between the two representations.

10.1.1 Initial state

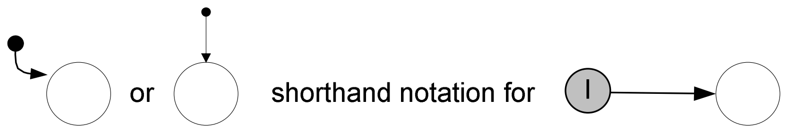


Figure 10-1: Initial state.

10.1.2 Effect associated with states

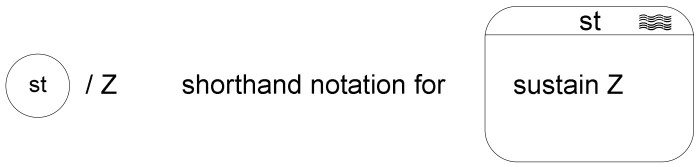


Figure 10-2: Effect associated with state.

10.1.3 Suspension

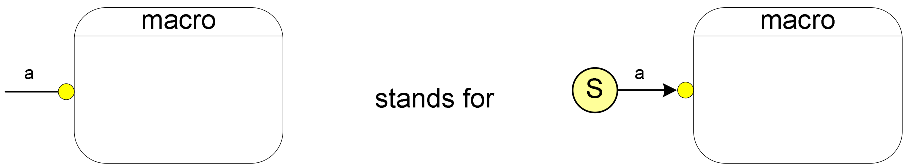


Figure 10-3: Suspension.

10.1.4 Entry and Exit Actions

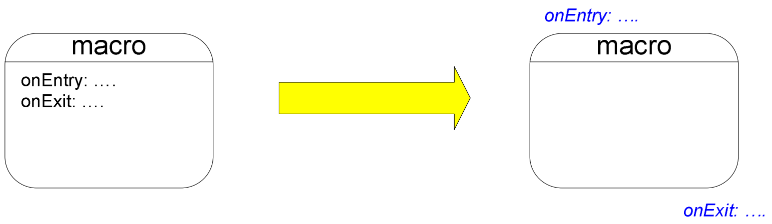


Figure 10-4: Entry and Exit Actions.

10.2 A Resource Management

This is a (simple) typical system, often referred to in this report.

10.2.1 The system

This system consists of

- A shared resource
- Two users that compete to access the resource
- An access controller (**ResMgr**)

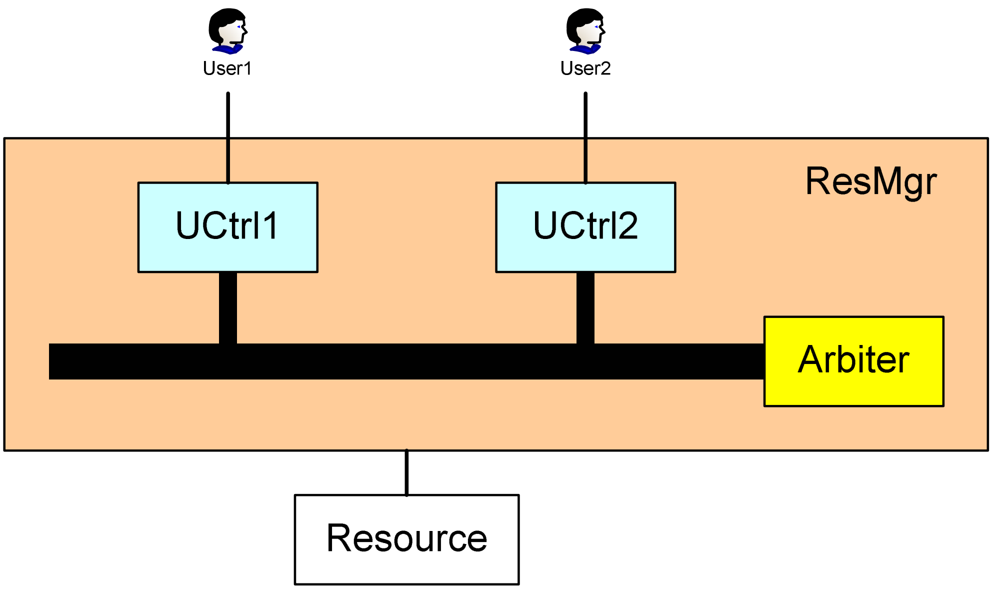


Figure 10-5: A Resource Management System.

The goal is to program the access controller. This controller is made of three cooperating controllers:

- Two interface controllers with the users: **UCtrl1** and **UCtrl2**
- An arbitration controller: **Arbiter**

10.2.2 Black-box view

UCtrl

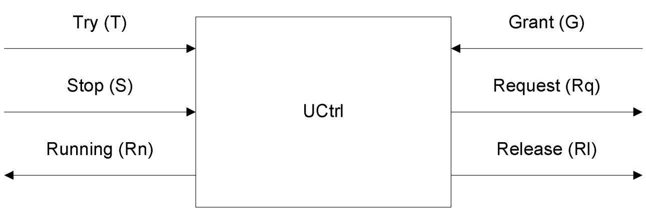


Figure 10-6: Interface of UCtrl.

Arbiter

Figure 10-7: Interface of Arbiter.

11 Glossary

Abortion

Strong Abortion

Form of *preemption* that forbids any reaction within the preempted state prior to the abortion. A strong abortion transition is drawn as

Weak Abortion

Form of *preemption* that lets the preempted state react prior to its abortion. A weak abortion transition is drawn as

Configuration

A configuration is a maximal set of states (macrostates or simple-states) that a syncChart could be in simultaneously.

Effect

An effect is a set of instantaneous actions that are associated with a transition or a simple-state. For Pure SyncCharts, such actions are only signal emissions. An effect associated with a transition is executed whenever the transition is taken. An effect associated with a simple-state is executed once at each instant when the state is active.

Execution traces

An Execution Trace is a representation of a particular behavior of a syncChart as a sequence of alternating *configuration* and *reaction*. A configuration can be described textually or graphically. A reaction is characterized by the set of present input signals and the set of emitted signals.

The notation for a reaction is $\xrightarrow[\text{present input signals}]{\text{emitted signals}}$

FSM: Finite State Machine

Discrete model made of states and transitions. Changes of state are modeled by transition firings, triggered by events. FSMs are used in many fields with various syntax and interpretation. Use in SyncCharts only as an informal model.

Normal Termination

A Normal Termination is a spontaneous exit from a macrostate, this exit occurs when each of STG of the macrostate is in a *final state*. A normal termination transition is drawn as . A normal termination transition should have no *trigger*.

Preemption

Preemption is the possibility to interrupt the activity of a state either definitively (*abortion*) or temporarily (*suspension*).

Signal

A signal is the unique abstraction for handling communication and synchronization. A signal has a presence status (present or absent). It may convey a value of a given type.

A signal has a scope: either external or local to a macrostate. External signals are further classified as input signals and output signals. Local signals are bidirectional.

Combined valued signal

A combined valued signal is a valued signal that can be emitted several times within one reaction. A combination function or operator is associated with this signal. The operation must be associative and commutative.

Pure signal

A pure signal conveys no value.

Single valued signal

A single valued signal is a valued signal that can be emitted only once during a reaction.

Valued signal

A valued signal conveys a value of a given type.

State

In SyncCharts, a state is either a simple-state or a macrostate.

Active state / Idle state

A state is either active (active point of control) or idle. An idle state may be activated. An active state may be de-activated.

Final state

A final state is a simple-state in which a STG waits for a normal termination. No effect is associated with a final state. Graphically, a final state is distinguished by its double outline.

Label associated with a state

An *effect* can be associated with a simple-state. The syntax of a label is "*/ effect*".

Macrostate

A macrostate is a state that is refined. A macrostate contains a non empty set of concurrent STGs.

Simple-state

A state that is not refined. An effect can be associated with a simple-state.

STG: State Transition Graph

An STG is a connected direct graph made of states and transitions. An STG has one initial state, and may have final states. At most one state is active in a STG. It is called its current state.

Suspension

A form of *preemption* that suspends the activity of a state while a *trigger* is satisfied. A suspension is represented by a "lollypop" . A suspension may be *immediate*.

Transition

A transition is a link between two states (its source and target states). In SyncCharts, a transition never crosses the macrostate boundary. A label may be associated with a transition. There are 3 types of transitions: strong abortion transition (sA), weak abortion transition (wA), and normal termination transition (nt).

Enabled transition

A transition whose source is active is enabled when its *trigger* is satisfied and its *guard* evaluates to true.

Firing/Taking a transition.

Taking a transition de-activates the source state, performs the associated effect, and activates the target state. The firing of a transition is instantaneous.

Immediate transition

For an immediate transition the trigger can be satisfied at the instant when the source state is activated, whereas for non immediate transitions the trigger can be satisfied only at a strictly future instant. The sharp symbol (#) denotes an immediate transition.

Label associated with a transition

The label of a transition has three optional fields: a trigger, a guard, an effect. Pure SyncCharts do not have a guard. The syntax of a label is "*trigger* [*guard*] / *effect* ".

Priority

A distinct integer value is associated with each transition leaving a state (the smaller integer, the higher priority). Strong abortions must be given higher priority than weak abortions, which in turn have higher priority than a normal termination.

Tick

tick is a predefined signal, present at each instant.

Trigger

A trigger is an expression on signals using operators and, or, and not. Some triggers may have a repetition factor (count delay), written "*integer-expression signal-expression*".

Complex trigger

A complex trigger is an expression effectively using operators and, or, or not.

Satisfaction

A trigger is satisfied when its expression evaluates to true. A simple trigger evaluates to true when the associated signal is present. A complex trigger is evaluated using the usual semantics of and, or, and not operators.
A trigger with a repetition factor of n is satisfied when the trigger has been satisfied at n different instants.

Simple trigger

A simple trigger consists of a single signal.

Trigger-less transition

The absence of a trigger for abortion transitions is interpreted as a simple trigger on the pre-defined signal tick.