

# Chapter 1

## A Discrete Model for Real-Time Environments

*Dan Ionescu and Cristian Lambiri*

### 1.1 Introduction

The notion of real-time system, as it is currently accepted, describes a computational process that has to respond to internal or external stimuli in determined periods of time. Such processes have to consider time as the independent variable as in all theoretical system approaches. In real-time systems the time variable has discrete values and it is considered in a different manner than their non real-time counterparts, where time is implied but never stated. This chapter tries to establish an abstract model for real time systems, which blends the discrete nature of computational processes with the continuous nature of the time constraints.

Real-time systems have been studied in the realm of control systems and/or computer related processes for a very long time. The models and solutions for control systems, brought forward by various recent researchers which consider it a domain of differential equations, are based on the mathematical apparatus that has been a staple of control theory since its inception: functional analysis [Curtain 97]. The rise of discrete computational processes has given way to new methods of investigating this class of very important systems. Logic based (temporal logic [Ostroff 89b]), as well algebraic based (*I/O automata* [Gawlick 93; Sogaard-Anderson 93; Segala 95], real-time process algebra [Baeten 91]) have been proposed and studied in extensively.

What this chapter tries to achieve is to combine the control theory insight within a computationally viable model. An early attempt to bridge the gap between control theory and computer science has been done by Arbib [Arbib 75] who pointed out that the notion of finite automaton is just a particular case of his and Kalman's [Kalman 69] notion of dynamical system. Control theory based works provided tools for a powerful insight in the modeling, analysis, and synthesis of continuous or discrete systems using abstract mathematical models which can span from complex or functional analysis to category theory. This can be leveraged by combining it with the well-established computational models that have been in use in the computer science world which lately arrived also at the category theory abstractions.

## 1.2 Time and Events

A starting point for the study is represented by the general properties of real-time systems. It is a consensus in all research work that all models should respect the principle of causality, that the system should react to events or stimuli and that it should have a discrete representation. For this property to be well founded the time is of essence.

### *Time*

The notion of time has been formalized in several ways, depending on the properties that we consider important. In the classical automata theory it is modeled as the set of natural numbers  $N$  and it is considered implicit in the definition of the automata. Using this mapping one can appreciate the running time complexity of programs by expressing it in number of transitions. Time can be made explicit by describing it as a special, infinitely recurring, event called *clock* [Ostroff 89b; Ostroff 90b; Brandin 94b], or by explicitly presenting it in the system definition as another component of the system [Kalman 69]. The former view is a natural extension of the classical automata theory. Its drawback stems from the special rules that have to be introduced in order to ensure the acceptance of the clock event. This chapter considers the second view and identifies time with the set of positive real numbers  $R^+$ , together with the usual binary relation  $\leq$ , as a model for time. It is easy to see that  $R^+$  has a cpo (complete partial order) structure.

Time is relative to the system of reference in which it is measured and so each system is considered to have its own *local time* ( $LT$ ). In order to be able to synchronize various parts of a distributed system a global entity called *global time* ( $GT$ ) has also to be considered. Between any  $LT$  and  $GT$  a Newtonian time relation exists. If  $\tau$  is the  $GT$  moment when system  $S$  started, a time moment in the two reference systems,  $t_g$  in  $GT$  and  $t_l$  in  $LT$  is in the following relation:  $t_g = t_l + \tau$ .

Let us consider two parts  $P_1$  and  $P_2$  of the same system  $S$ . Each part has its own local time  $T_1$  and  $T_2$  respectively. Considering that both parts started to function at the same  $GT$  moment  $\tau$ , there is obviously an isomorphism between  $T_1$  and  $T_2$ :  $id : T_1 \rightarrow T_2$ . This isomorphism implies that both systems measure the time in the same manner.

*Discrete event systems* [Lin 93] have the property that events appear at discrete moments in time, but these moments can be anywhere on the time axis.  $\mathbb{R}^+$  is continuous from zero to  $\infty$  and can, therefore, handle events that appear at any time moments.

Time moments are not the only kind of temporal structures. Of equal importance are the time intervals. We will denote a time interval  $[t_1, t_2)$  by  $\sigma_{t_1}^{t_2}$ . Depending on the values of  $t_1$  and  $t_2$  several special cases that are more important will have separate denotations. A time interval that has only an upper limit  $[0, t)$  will be denoted by  $\sigma^t$ . Similarly for a time interval that has only a lower limit,  $[t, \infty)$ , the following notation will be used  $\sigma_t$ . Single point intervals of the form  $[t, t]$  will be denoted by  $\bar{\sigma}_t$ .

Let us consider now the set  $\Sigma$  of all time intervals over  $\mathbb{R}$ . Over this set, let us define a partial operation, (denoted by  $+$ ), called *interval concatenation*, in the following manner:

$$\sigma_{t_1}^{t_2} + \sigma_{t_2}^{t_3} = \sigma_{t_1}^{t_3},$$

$$\sigma_{t_1}^{t_2} + \sigma_{t_3}^{t_4} = \begin{cases} \sigma_{\min(t_1, t_3)}^{\max(t_4, t_2)}, & \text{if } [t_1, t_2) \cap [t_3, t_4) \neq \emptyset \\ \text{undefined} & , \text{ if } [t_1, t_2) \cap [t_3, t_4) = \emptyset \end{cases}$$

## Events

Based on their time existence, events can be classified in *discrete* and *duration* events. In our model we consider the *discrete* variety as the base event type, with the *duration* events modeled as strings of discrete events. We will denote *discrete events* with  $dt$  and with  $it$  the interval events.

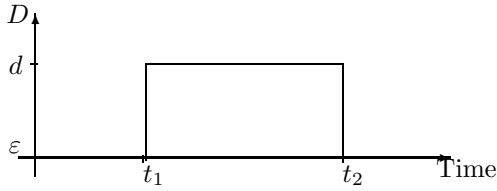


Fig. 1.1 Timed events

Events are defined by two time constants  $t_1$  and  $t_2$  that give a lower and an upper bound for the event's validity as shown in Fig. 1.1. An event is therefore a triple  $(e, t_1, t_2)$ . The first component  $e \in D$ , where  $D$  is a set called the domain (Fig. 1.1), and  $t_1, t_2 \in R^+$  specify the time value when the event starts to have an impact in the system while  $t_2$  specifies the time value when the event ends to have an impact on the system. Events are described in their own reference system. To be more precise, if an event  $(e, t_1, t_2)$  appears at a moment  $t_d$  in a system's local time, it has to be executed in the interval  $[t_d + t_1, t_d + t_2)$  in order to have an effect. The same interval in  $GT$  is given by  $[\tau + t_d + t_1, \tau + t_d + t_2)$ .

**Definition 1.1** The set of timed events over the domain  $D$  is the set of triples  $(e, t_1, t_2)$ , where  $t_1, t_2 \in R^+$ , with the property that  $t_1 \leq t_2$ .  $E = \{(e, t_1, t_2) | e \in D, t_1 \in R^+, t_2 \in R^+, t_1 \leq t_2\}$

The dynamics of the system is given by the interaction with the environment through its inputs and outputs. When an event is received, the time of entry in the system becomes part of the event. As an example, let us consider a set of events  $E$  as defined above. The set of possible event strings that can be generated by the environment is a subset of  $(E \times T)^*$ , with the obvious condition that if  $i = \dots(e_\alpha, t_\alpha)(e_\beta, t_\beta)\dots$ , then  $t_\alpha < t_\beta$ . Let us denote by  $\omega_{t_1}^{t_2}$  an input string that starts at  $t_1$  and ends at  $t_2$ . If  $t_2$  is finite then the length of such a string has to be finite.

The above condition ensures that the so-called Zeno executions, over a finite interval, do not appear.

Events are described in their own reference system. Thus, if an event  $e_{t_1}^{t_2}$  appears at moment  $t_d$  in the local time of a system, it has to be executed in the interval  $[t_d + t_1, t_d + t_2)$  in order to affect the system. The same interval in  $GT$  is given by  $[\tau + t_d + t_1, \tau + t_d + t_2)$ . Of course, an even more sophisticated interpretation can be brought up, where the event has a

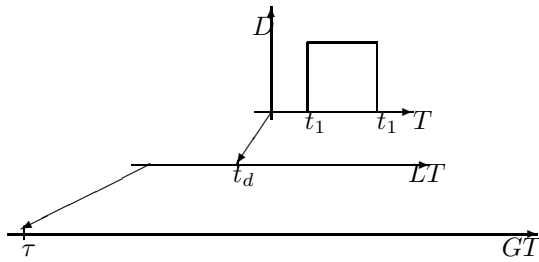


Fig. 1.2 Relation between time referentials

“good” behavior in the enabling interval and a “bad” one outside it. If the time interval is  $[0, \infty)$  then the events are untimed. The interval addition can be used to simplify the event description. Let  $e_{d_{t_1}^{t_2}}$  and  $e_{d_{t_3}^{t_4}}$  be two timed events for the same symbol  $d$ . If the operation  $\sigma_{t_1}^{t_2} + \sigma_{t_2}^{t_3}$  is defined, then  $e_{d_{\min(t_1, t_3)}^{\max(t_4, t_2)}}$  (Figure 1.2) represents both events.

The set of admissible input functions is defined over subsets of  $T$  and with the co-domain in  $E$ . Events on the other hand are formalized as discussed above. The set of all events that appear on any *finite* time interval has to have finite cardinality. This is true because a real-time system reacts in a finite amount of time to any event. In the case when an *infinite* number of events are allowed to appear over a finite interval, the system will not finish processing those events over a finite interval. Such a behavior is considered unacceptable and hence, the above condition is imposed. If  $\omega_{t_1}^{t_0}$  is an acceptable input function defined over a time interval, with  $n$  the number of time points where  $\omega$  is defined, the system will process all the inputs in a  $n * \mu$  period of time.  $\Omega$  is composed of functions  $\omega$  defined in the following manner:  $\Omega = \{\omega : T \rightarrow E \mid \omega = e_k(t_{e_k})\}$ . Thus, any function  $\omega$  has to be defined in a *countable* set of time points. If  $T_d \in T$  is the set of time points where the function  $\omega$  is defined there exists a morphism  $f : T_d \rightarrow N$ . If  $\text{Card}(T_d) = \infty$  then  $f$  is an isomorphism.

The above condition ensures that the so-called Zeno executions [Sogaard-Anderson 93], over a finite interval, do not appear. In such a behavior, the processing time of a set of events that appear over a finite interval takes an infinite amount of time. There is also another type of Zeno behavior. This occurs when the number of events that appear over any finite interval is greater than the number of entries in the input buffer

plus the number of events that can be processed in that interval. If  $K$  is the maximum number of events that can be buffered, the number of events that can appear over an interval  $\sigma_{t_1}^{t_2}$  is given by:  $N \leq K + \lfloor (t_2 - t_1)/\mu \rfloor$ . If the number of events that appear over a period of time is greater than  $K + \lfloor (t_2 - t_1)/\mu \rfloor$  then some of them will be lost. Over an infinite interval this means that the arrival rate of the events has to be less than or equal to  $1/\mu$ .

The acceptable input functions can be viewed as words in a language  $L_E$ . The language is defined over an *infinite* alphabet  $E \times T$  where  $E$  is a set of events (timed and untimed) and  $T$  is the time set. A word in the language  $w$  is defined as follows:  $\forall \omega \in \Omega, \forall t \in T$  where  $w = \omega(t_0)\omega(t_1)\dots\omega(t_n)$ . The language  $(L_E)$  is composed of the set of all words  $w$  such that  $w = e_0(t_0)e_1(t_1)\dots e_n(t_n)$  and  $t_0 < t_1 < \dots < t_n$ , where  $e_0, e_1, \dots, e_n \in E$  and  $t_0, t_1, \dots, t_n \in T$ . Hence,  $L_E$  is a subset of  $(E \times T)^*$ .

Another way of introducing timed events is by relating a *timer* with each event [Brandin 94b; Alur 90c]. There, the authors present similar approaches in the way they introduce the timed events, with the difference that in [Brandin 94b] timers for events are integrated in the notion of state of the system as in [Alur 90c] they are considered a separate entity.

### 1.3 Discrete Real-Time Systems

The system model that we consider is a discrete one, in the sense that the states set is countable. The duration of a state transition depends on the type of event that the system is processing. Thus the *transition interval* is given by the function  $\mu : D \rightarrow T$ . This means that on any finite interval, the system will make a countable number of steps of different lengths.

**Definition 1.2** A Discrete Real Time System **DRTS** is a tuple  $A = (T, Q, E, Y, \phi, \beta, \mu)$  where:

- $T = \mathbb{R}^+$  is a set of **time values**
- $E$  is a set of **events** and  $I = E \times T$  is the input alphabet.
- $Q$  is a set of symbols called **states**.
- $Y = (Y_j \mid j \in J)$  is an indexed set of **output values** with  $Y_j \in D \times T \times T$ .
- $\mu : prj_1(e) \rightarrow T$  is the transition function.
- $\phi$  is a function  $\phi : T \times Q \times I \rightarrow Q$ , called the **next state function**.

If the system is in the state  $q$  at  $t$  it will be in state  $q_1$  at  $t + \mu(\text{prj}(e))$ , where  $e_{t_1}^{t_2} \in E$  is the event,  $t_d$  is the moment when the event appears and  $t \geq t_d$  is the moment when the event is processed.

- $\beta$  is a set of total functions  $\beta = \{\beta_j | \beta_j : Q \rightarrow Y_j\}$ , called **output functions**.

The definition above is missing the initial and final states. It is easy to particularize the DRTS definition to make it accommodate such states.

**Definition 1.3** An iDRTS is an DRTS together with  $q_0 \in Q$ , a special state called initial state, and  $F \in Q$  a set of final states.

The class of *reactive systems* also fall into the above definition, by considering the set  $F$  to be empty. Such a system will be denoted by *iDRTS*.

An input-output run of a DRTS is a string of triples  $(e, t, y)$  where  $e$  is an event,  $t$  the moment when it appears, and  $y$  the output of the system. All the runs will be considered in  $LT$  and therefore will start at  $t = 0$ . For reactive systems runs have infinite length, while for other types of DRTS' might have finite runs. Considering a DRTS  $A$ , we will denote by  $\Omega_q$  the set of all possible runs of  $A$  when it starts in state  $q$ . For *iDRTS*, where the initial state is known, the set will be simply denoted by  $\Omega$ . Two states  $q_1$  and  $q_2$  are said to be equivalent if  $\Omega_{q_1} = \Omega_{q_2}$ . Since *iDRTS*' always starts from the same state,  $q_0$  we will call two iDRTS'  $A$  and  $B$  equivalent if  $\Omega_A = \Omega_B$ .

## 1.4 Composition and Decomposition of iDRTS Structures

The main point of interest, in this section, is the way we can combine simpler structures in order to obtain more complicated ones. Another way of obtaining new systems from known ones is by transforming them. The following definition gives the restrictions needed in order to build a morphism between two iDRTS.

**Definition 1.4** Let  $M_1 = (T, E_1, Q_1, Y_1, \delta_1, \beta_1, q_{01})$  and  $M_2 = (T, E_2, Q_2, Y_2, \delta_2, \beta_2, q_{02})$  be two iDRTS. A **morphism** between  $M_1$  and  $M_2$  is a set of functions  $\{tm : T \rightarrow T, h : Q_1 \rightarrow Q_2, ev : E_1 \rightarrow E_2\} \cup \{g_j : Y_{1j} \rightarrow Y_{2j} \mid \forall j \in J\}$  satisfying the following conditions:  $f(\tau_1) = \tau_2$ ,  $f(\mu_1) = \mu_2$ ,  $h(q_{01}) = q_{02}$ ,  $h(\delta_1(t, q_1, (e_1, t_d))) = \delta_2(tm(t), h(q_1), (ev(e_1), tm(t_d)))$  and  $\forall j \in J, g_j(\beta_{1j}(q_1)) = \beta_{2j}(h(q_1))$ .

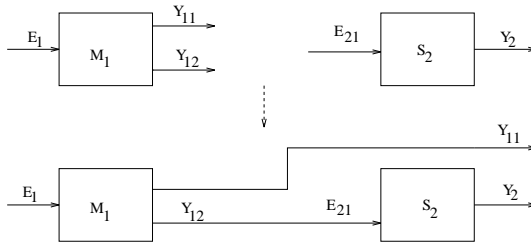


Fig. 1.3 Series composition of iDRTS

Using three types of compositions, *series*, *parallel* and *feedback*, we can obtain complex systems from simpler ones. Although used a lot by system theory, composition is weakly represented in the domain of formal models. Most of the authors seem concerned with the monolithic description of the system and except for parallel composition [Alur 90c; Sogaard-Anderson 93; Lin 93; Manna 92; Brandin 94b] none of the other is considered. Parallel composition provides an fairly accurate description of the behavior for *non interacting* systems functioning simultaneously. For systems that also interact with each other more sophisticated constructions are needed. Some define [Ostroff 89b] *parallel compositions* although the system is a series construction with feedback. Following classical work in system theory [Mesarovic 75] and automata theory [Arbib 69] the *series*, *parallel* and *feedback* composition will be defined.

In the *series* configuration (Figure 1.3) two automata are linked so that outputs from one automaton are connected to inputs of the other automaton. Of course the connection can be made if and only if the set  $Y_{12} \subset E_{21}$ . Without this connection compatibility the second system will not make any transitions. In the composed system both components start at the same time  $\tau$  and thus, the set  $T$  is the same on both systems. Furthermore the output of the first system has to be event driven in order to have the same structure as the other events. Without this, the output would generate an uncountable number of events and this would be a Zeno behavior.

The sets of admissible inputs are, usually, different between iDRTS as a result of different transitions times and input queue lengths. To have a resulting system with non-Zeno behaviors, it is necessary to impose restrictions on the admissible input runs for the composed system. The admissible runs are determined by the transition time. To obtain them for

the composed system let us compare the transition times for the parts. If  $\mu_1 \geq \mu_2$  then the actions generated by the first system are at least  $\mu_1$  apart and this second system will synchronously process them as they appear. Hence, the set of acceptable inputs for  $M_1$  is the set of acceptable inputs for the composed system. If  $\mu_1 < \mu_2$  the set of admissible inputs for the composed system should be reduced such that the arrival rate is smaller than  $1/\mu_2$ . Therefore the arrival rate for the composed system should be:  $1/\max(\mu_1, \mu_2)$ .

In the sequel only the case when  $\mu_1 > \mu_2$  is considered. The transition time of the composed system is then  $\mu = \mu_1 + \mu_2$ . The above equation states that the outputs are considered to change only after the effect of an input event had propagated through the system.

**Definition 1.5** Let  $M_1 = (T, E_1, Q_1, Y_1, \delta_1, \beta_1, q_{01})$  and

$$M_2 = (T, E_2, Q_2, Y_2, \delta_2, \beta_2, q_{02})$$

be two iDRTS', where  $Y_1 = (Y_{11}, Y_{12})$ ,  $\beta = (\beta_{11}, \beta_{12})$ . The series composition of  $M_1$  and  $M_2$  denoted by  $(M_1 \odot M_2)$  is a system

$$M = M_1 \odot M_2 = (T, E, Q, Y, \delta, \beta, q_0)$$

where

- $Q = Q_1 \times Q_2$  is the state set.
- $E = E_1$  is the event set and  $I = E_1 \times T$  is the input alphabet.
- $Y = (Y_{11}, Y_2)$  is the output set.
- $\delta : T \times Q \times I \longrightarrow Q$  is the next state function.

$$\delta(t, (q_1, q_2), (e_1, t_d)) = (\delta_1(t, q_1, (e_1, t_d)), \delta_2(t, q_2, (\beta_{11}(q_1), t_d))).$$

- $\beta = (\beta_{11}, \beta_2)$  is the set of output functions.
- $q_0 = (q_{01}, q_{02})$  is the *initial state*.

Let us consider the case when several inputs are applied to the same system. There are two possible points of views for these kinds of structured inputs. The first one considers the events to be synchronized [Ionescu 94].

To explain this, let us consider two iDRTS  $M_1$  and  $M_2$  working in parallel. Let  $E_1$  is the set of admissible events for  $M_1$ , and  $E_2$  for  $M_2$ . Then the set of events for the resulting system is  $E' = \{(e_1, \varepsilon) \mid e_1 \in E_1\} \cup \{(\varepsilon, e_2) \mid e_2 \in E_2\} \cup \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2\}$ . The set of

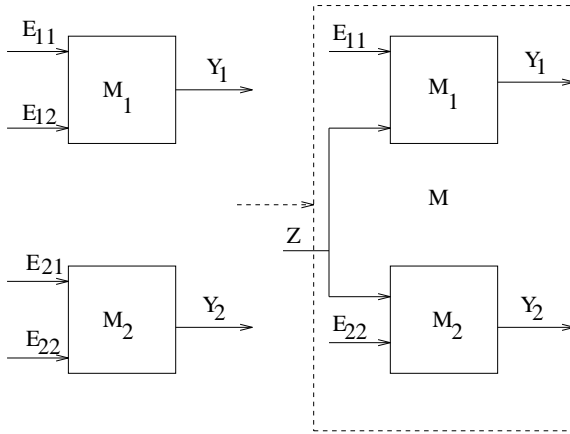


Fig. 1.4 Parallel composition of iDRTS

events for the parallel composition denotes that the events can appear at the same moment in time. Therefore they can be synchronized, or appear at different moments and be desynchronized. In each case the composed events are the doubles  $(e, \varepsilon)$  or  $(\varepsilon, e)$  where  $\varepsilon$  represents a “non-event”. This point of view considers all the events queued in a single unit and processed synchronously. If  $I_j = E_j \times T$  is the input alphabet for one input then because  $T \times T \times \dots \times T \cong T$  the alphabet of the composed inputs is  $E_1 \times E_2 \dots E_n \times T$ .

A second point of view is to consider the inputs to be independently processed. Considering the inputs independent though, contradicts our intuition that systems behave like monoliths.

In the *parallel* configuration the set of events  $E$  of the resulting machine is a subset of the Cartesian product of the sets of events of the composing machines  $E \subset E_1 \times E_2$ . As with the *serial connection*, both machines have the same time reference, with the same initial moment. This ensures that the elements of the Cartesian product are valid events for the composed machine.

**Definition 1.6** Let  $M_1 = (T, E_1, Q_1, Y_1, \delta_1, \beta_1, q_{01})$  and

$$M_2 = (T, E_2, Q_2, Y_2, \delta_2, \beta_2, F_2, q_{02})$$

be two iDRTS' (Figure 1.4), with  $E_1 = (E_{11})$  and  $E_2 = (E_{21} \times E_{22})$ . The parallel composition of  $M_1$  and  $M_2$  denoted by  $(M_1 \oplus M_2)$  is a system

$$M = M_1 \oplus M_2 = (T, E, Q, Y, \delta, \beta, q_0)$$

where

- $T$  is the set of time values.
- $Q = Q_1 \times Q_2$  is the set of states and  $q_0 = (q_{01}, q_{02})$ .
- $E = (E_{11} \times Z \times E_{22})$ , with  $Z \subseteq E_{12} \cap E_{21}$ , is the set of events and  $I = E_{11} \times Z \times E_{22} \times T$  is the input alphabet.
- $Y = (Y_1, Y_2)$  is the set of output values.
- $\delta : T \times Q \times I \rightarrow Q$  is the next state function, defined by:

$$\delta(t, (q_1, q_2), (e_1, z, e_2, t_d)) = (\delta_1(t, q_1, (e_1, z, t_d)), \delta_2(tq_2, (e_2, z, t_d))),$$

- $\beta = \{\beta_1, \beta_2\}$  is the set of output functions.

The transition time of the composed system is equal to  $\max(\mu_1, \mu_2)$ .

In the *feedback* connection some of the system's outputs are connected with its inputs thus forming another system. The feedback system can make more than one transition for certain inputs, because the output that is generated will be another event for the system. The iDRTS of the Definition 1.2 makes only one transition after it receives an event. In this case we define the transition of the feedback system to take place only when it reaches a state that will not change unless it receives an input event. Evidently the system can make several internal transitions in order to enter such a state. The set of admissible input functions for the feedback system is only a subset of that of the original system. This is due to the increased processing time.

**Definition 1.7** Let  $M_1 = (T, E_1, Q_1, Y_1, \delta_1, \beta_1, F_1, q_{01})$  be a iDRTS (Figure 1.5), with  $E_1 = (E_{11} \times E_{12})$  and  $Y_1 = (Y_{11}, Y_{12})$ . The **feedback** connection of  $M_1$  is a system

$$M = (T, E, Q, Y, \delta, \beta, q_0, F),$$

where:

- $T$  is the set of time values.
- $Q = Q_1$  is the set of states, with  $q_0 = q_{01}$ .
- $E = E_{12}$  is the set of events and  $I = E \times T$  is the input alphabet.

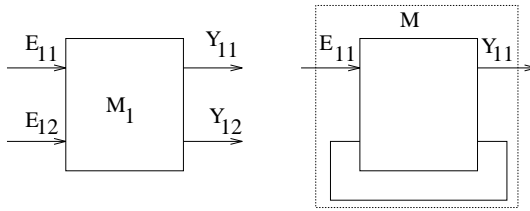


Fig. 1.5 Feedback composition of iDRTS

- $Y = Y_{12}$  is the set of outputs.
- $\beta = \{\beta_1\}$  is the set of output functions.
- $\delta : T \times Q \times I \longrightarrow Q$  is the next state function:

$$q_1(t + n * \mu) = \delta(t, q, (e_{11}, t_d)) = \delta_1(\dots(\delta_1(t, q, (e_{11}, t_d))))).$$

Due to their nature, some feedback systems can have an infinite transition time. This is because the system will make internal transitions forever. This suggests that the class of all iDRTS' is not closed under the feedback operation. Thus, such an operation has to be applied with caution.

## 1.5 DRTS as Algebras

The relation between automata theory and algebra, an idea that seems almost natural now, has been first pointed out by Büchi [Buchi 89], who studied automata as *unary algebras*. By reducing the class of abstract automata to a unary algebra, only one set of the structure can be studied and therefore simplifications have to be made. Pointing out the problem Büchi [Buchi 89] also noted that means of a more general approach can study the same structures. He proposed *multi sorted algebra* which, presents the advantage of allowing an indexed carrier for the algebra. Therefore, the totality condition imposed for algebra operators can be satisfied over subsets of the carrier. Noticing the relation of universal algebra with another mathematical field of study, category theory, automata theorists and mathematicians tried to link the two fields together. Manes and Arbib [Arbib 75] have started a categorical approach by proving that the class of automata is a category and also, that the automaton itself can be viewed as a category. In this case automata properties are derived using categorical con-

cepts. Adamek [Adamek 89] used an interesting variant to the categorical approach, by studying automata as *functors* between categories.

The algebraic methods of study are at help when one tries to make the transition from the automata as models of computations to logic theories that describe them. The relation of universal algebra with logic, by the means of model theory, means that a link between logical theories and the abstract models can be made. This is the reason of approaching the DRTS and therefore distributed systems through an algebraic theory.

Let us now consider the **DRTS** framework in the light of universal algebra. The multi set nature of the framework makes it a natural candidate for their study using *multi sorted* algebra.

Let  $M = (T, Q, E, Y, \delta, \beta)$  be a **DDES**. The condition  $t \leq t_d$  makes  $\delta$  partial. To make a **DDES** suitable for study using the universal algebra framework, it is needed to make  $\delta$  a total function. This can be done by considering an extension of  $\delta$  such that, for all the doubles  $(t, t_d)$  where the function is undefined,  $\delta(t, q, (e, t_d)) = q$ . In the remainder of the section  $\delta$  will be considered to be the extended *next state function*  $\bar{\delta}$ . As previously stated, for **DRTS** only the reactive case with  $F = \phi$  is considered where  $F = \{f|f : Td \rightarrow N\}$  Hence, the set  $F$  does not appear explicitly. As expected the *DRTS*, with the extended transition function, have the structure of universal algebras as proven by the following theorems.

**Theorem 1.1** *A DDES is a multi-sorted algebra.*

**Proof.** Let  $M = (T, Q, E, Y, \delta, \beta)$  be a **DDES**. Let us attach a sort to each of the sets. Hence, we'll consider  $T$  of having sort  $s_1$  and  $Q$  of having sort  $s_2$ . The set of output values is made of several subsets:  $Y = (Y_1, \dots, Y_j)$ . Each  $Y_j$  is considered to have sort  $j$ . Finally consider  $E$  of having sort  $i$ . Thus, the set  $S$  of sorts is composed of:  $\{s_1, s_2, i\} \cup \{j | midj \in J\}$ . Over  $S$  consider the signature  $\Sigma = (\delta, q_0, \beta_j | j \in J)$  with the following arities:  $ar(\delta) = (s_1 s_1 s_2 i, s_2)$ ,  $ar(\beta_j) = (s_2, j)$ .

We consider the set  $C = (T_{s_1}, Q_{s_2}, E_i, \{Y_j | j \in J\})$  to be the carrier and the functions  $\delta$  and  $\beta_j \in \beta$  from  $M$  to be realizations of the operation symbols from  $\Sigma$ . Then by the definition of multi sorted algebras,  $A = (C, \delta, \{\beta_j | j \in J\})$  is a multi sorted algebra. Therefore,  $M$  is a *multi sorted algebra*.  $\square$

**Theorem 1.2** *An iDRTS is a multi-sorted algebra.*

**Proof.** Let  $M = (T, Q, E, Y, \delta, \beta, q_0)$  be a **DRTS**. As in the case of **DDES** let us consider the set  $C = \{T_{s_1}, Q_{s_2}, E_i, \{Y_j \mid j \in J\}\}$  with the arities specified by the indexes.

If  $q_0$  is considered to be an operation symbol, then we obtain the signature  $\Sigma = (\delta, \{\beta_j \mid j \in J\}, q_0, )$ . The signature has the following arities:  $ar(\delta) = (s_1 s_1 s_2 i, s_2)$ ,  $ar(\beta_j) = (s_2, j)$ ,  $ar(q_0) = s_2$ . Hence, the set  $C$  with the operation defined by  $\sigma$  form, is a multi sorted algebra. Therefore,  $M$  is a *multi sorted algebra*.  $\square$

**DRTS** are therefore *multi-sorted* algebras. In this context, it is interesting to notice that morphisms between DRTS are homomorphisms in the universe of multi sorted algebras.

**Corollary 1.1** *A morphism between two DRTS is a homomorphism between the underlying algebras.*

$$\begin{array}{ccccccc}
 T \times T \times Q_1 \times \Pi E_1 & \xrightarrow{\delta_1} & Q_1 & \xrightarrow{\beta_1} & Y_{1j} \\
 \downarrow tm_1 \quad \downarrow tm_2 \quad \downarrow h \quad \downarrow ev & & \downarrow h & & \downarrow g_j \\
 T \times T \times Q_2 \times \Pi E_2 & \xrightarrow{\delta_2} & Q_2 & \xrightarrow{\beta} & Y_{2j}
 \end{array}$$

**Proof.** Let  $M_1 = (T, E_1, Q_1, Y_1, \delta_1, \beta_1, q_{01})$  and  $M_2 = (T, E_2, Q_2, Y_2, \delta_2, \beta_2, q_{02})$  be two DRTS' with  $Y_1$  and  $Y_2$  defined over the same  $J$ . By Lemma 1.2  $M_1$  and  $M_2$  are algebras over the same signature  $\Sigma = (\delta, \{\beta_j \mid j \in J\}, q_0)$ .

The set of functions  $\{tm_1 : T \rightarrow T, tm_2 : T \rightarrow T, h : Q_1 \rightarrow Q_2, ev : E_1 \rightarrow E_2\} \cup \{g_j : Y_{1j} \rightarrow Y_{2j} \mid \forall j \in J\}$  satisfying the morphism conditions of Definition 1.4 also satisfy the conditions for multi sorted homomorphism.

Over the signature  $\Sigma$ , we have:

- $h(q_{01}) = q_{02}$  by hypothesis.
- $\forall f_1 \in F_1 \exists f_2 \in F_2$  such that  $h(f_1) = f_2$  by hypothesis.
- $h(\delta_1(t_1, t_2, q_1, e_1)) = \delta_2(tm(t_1), tm(t_2), h(q_1), ev(e_1))$  and
- $\forall j \in J, g_j(\beta_{1j}(q_1)) = \beta_{2j}(h(q_1))$ .

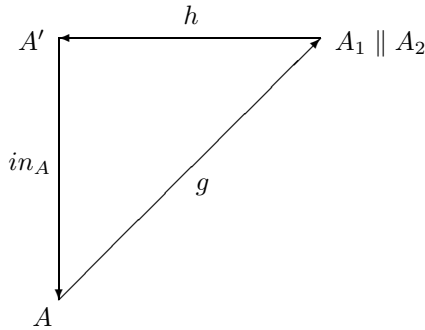
Therefore, by the definition of morphisms  $[tm_1, tm_2, h, ev, (g_j \mid j \in J)]$  is a  $\Sigma$ -homomorphism.  $\square$

The concept of **direct product** of algebras captures the behavior of parallel working automata. Let  $A_1 = (T, E_1, Q_1, Y_1, \delta_1, \beta_1, q_{01})$  and  $A_2 = (T, E_2, Q_2, Y_2, \delta_2, \beta_2, q_{02})$  be two iDRTS' and  $B$  their parallel composition with respect to Definition 1.6. The direct product  $A = A_1 \times A_2$  is the algebra  $A = (T \times T, E_1 \times E_2, Q_1 \times Q_2, Y_1 \times Y_2, \delta, \beta, (q_{01}, q_{02}))$ , where the functions  $\delta$  and  $\beta \in \beta$  are defined point wise. Let us study the relation between the Cartesian product of two algebras and the *parallel* composition of the iDRTS.

More precisely, let us first consider the parallel connection of two iDRTS' as in Figure 1.4. Let  $B = (T, E, Q, Y, \delta, \beta, q_0)$  be the multi sorted algebra obtained from the result of the parallel composition of the  $A_1$  and  $A_2$ . For the parallel connection to have meaning, we have to impose the condition  $E = E_{12} \cap E_{21}$ . Considering the subset  $E_s = \{(e, e) \mid e \in E\} \subset E_{12} \times E_{21}$  we can build a sub-algebra  $A'$  of  $A$  (the direct product of  $A_1$  and  $A_2$ ), formed with the same sets as  $A$  except for the event set, where we will take only a subset  $E_v = E_{11} \times E_s \times E_{22} \subset E_{11} \times E_{12} \times E_{21} \times E_{22}$ . Furthermore let us consider the following function  $f : E_{11} \times E \times E_{22} \longrightarrow E_{11} \times E_s \times E_{22}$  defined as  $f(e_{11}, e, e_{22}) = (e_{11}, e, e_{22})$ . As previously discussed for the time set  $T \times T \cong T$ . Let us consider the function  $id_{T \times T} : T \longrightarrow T \times T$ , defined as:  $id_{T \times T}(t) = (t, t)$ . Hence,  $h = [id_{T \times T}, id_{Q_1 \times Q_2}, f, id_{Y_j}]$  is a homomorphism between  $B$ , the algebra representing the parallel connection, and  $A'$ . Let us denote  $E_1 = E_{11} \times E_{12}$ ,  $E_2 = E_{21} \times E_{22}$  and  $Q_{1 \times 2} = Q_1 \times Q_2$ .

$$\begin{array}{ccccccc}
 T & \times & T & \times & Q_{1 \times 2} & \times & E_{11} \times E \times E_{22} & \xrightarrow{\delta} & Q_{1 \times 2} & \xrightarrow{\beta_j} & Y_{1j} \times Y_{2j} \\
 \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 id_{T \times T} & & id_{T \times T} & & id_{Q_1 \times Q_2} & & f & & id_{Q_1 \times Q_2} & & id_{Y_{1j} \times Y_{2j}} \\
 T \times T & \times & T \times T & \times & Q_{1 \times 2} & \times & E_1 \times E_s \times E_2 & \xrightarrow{\delta} & Q_{1 \times 2} & \xrightarrow{\beta_j} & Y_{1j} \times Y_{2j}
 \end{array}$$

However,  $A'$  is a sub algebra of  $A$ . Therefore, there exists a homomorphism  $in_A : A' \longrightarrow A$ . By applying the Universal Property of Algebra we obtain a homomorphism from  $B$  to  $A$ . Hence, there exists a homomorphism from the underlying algebra of the parallel connection and the Cartesian product of  $A_1$  and  $A_2$ .



A characterization from  $A_1 \times A_2$  to  $B$  is also possible. In this case we need a function  $tm : T \times T \rightarrow T$  and a function  $g : E_{11} \times E_{12} \times E_{21} \times E_{22} \rightarrow E_{11} \times E \times E_{22}$ . The function  $tm$  can be defined as

$$tm(t_1, t_2) = \begin{cases} t & \text{if } t_1 = t_2 \\ 0 & \text{otherwise} \end{cases}$$

and the function  $g$  by the following description:

$$g(e_{11}, e_{12}, e_{21}, e_{22}) = \begin{cases} (e_{11}, e, e_{22}) & \text{if } e_{12} = e_{21} \\ (e_{11}, \varepsilon, e_{22}) & \text{otherwise} \end{cases}.$$

Thus,  $k = [tm, tm, g, id_{Q_1 \times Q_2}, id_{Y_j}]$  is a homomorphism from  $A_1 \times A_2$  to  $B$ .

## 1.6 A Model for Sequential Processes

This section examines the structure of computer processes, using the formal model that was built in the previous paragraphs. The abstraction process uses a “reversed engineering” method by trying to formalize a concept derived from a human made tool to produce other tools. Our technological perception of the “computational process” biases this type of abstraction.

Multitasking systems appeared after the sequential ones and tried to mimic their behavior, by creating the illusion of multiple parallel processes executed on a single physical unit. Thus the notion of “sequential process” is crucial to the formalization process.

In this chapter sequential processes are considered to be collections composed of two entities that interact with each other. One entity ensures the dynamics of the process and therefore, acts as a *controller*, while the other

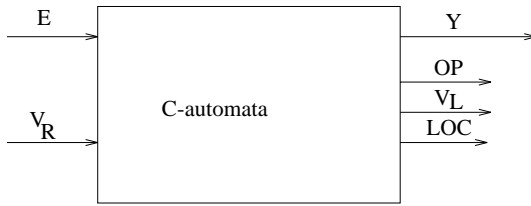


Fig. 1.6 C-automata

one acts as a *memory* that stores variables for the controller. Let us examine both parts from the perspective of the **iDRTS** model developed in the previous sections.

The *control* part of the process will be considered to be a **iDRTS**. The abstraction can be made because the controller has discrete states and accepts discrete events.

Let us consider a particular kind of **iDRTS** with the structure of inputs and outputs depicted in Figure 1.6. Let us call this type of system a *C-automaton*. The set of inputs for the system is structured, as visible in Figure 1.6. This is because some inputs ( $V_R$ ) are needed to interact with the *memory* part of the process, as the others are needed for external interaction  $E$ . The set of outputs is also composed of several subsets. Part of the outputs are inputs in the memory subsystem ( $OP, V_L, LOC$ ) and the rest are the outputs of the process. Thus, a C-automaton is an **iDRTS** with the following structure:

$$A_c = (T, \{E, V_R\}, Q, \{LOC, OP, V_L\}, Y, F, \delta, \{\beta_1, \beta_2\}, q_0)$$

where

- $E_c = V_R \times E$  is a set of events.
- $Q$  is a set of states.  $q_0 \in Q$  is the initial state.
- $\beta = V_L \times OP \times LOC \times Y$  is the a set of output events.
- $\delta : T \times T \times Q \times E_c \longrightarrow Q$  is the next state function,  $\delta(q, e_c) = q_{next}$
- $\beta = (\beta_1, \beta_2), \beta_1 : Q \longrightarrow Y, \beta_2 : Q \longrightarrow OP \times V_L \times LOC$  is the output function.

The system, as shown in Fig. 1.7 has two output functions,  $\beta_1$  and  $\beta_2$ , to make the distinction between the process output and the commands for the memory.

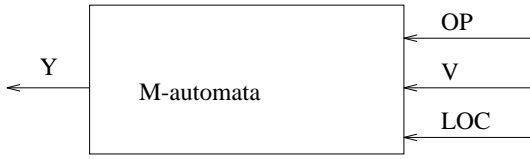


Fig. 1.7 M-automata

To be able to make a suitable abstraction for the memory we must analyze how a real memory behaves. The term can mean many things, but here we refer to a device that can store and access information by direct addressing. In the technology used for most of today’s computers a memory system is just a conglomerate of units called *memory cells*. A *memory cell* is a place where a quantity of information can be stored at times and retrieved afterwards. The number of different symbols that can be stored depends on the memory and is always finite. All the events that a memory receives are un-timed. Furthermore the memory system is time invariant and thus the time set can be omitted from the definition of the system. By considering the stored symbols as the *states* of the memory we can abstract it as an iDRTS with the following structure:  $(Q, Q, Q, \delta, \{\beta\}, F, q_0)$ . The notion of final state does not make sense for a storage facility. We will consider  $F = \phi$  and omit it from subsequent references.

We can expand the model by considering several memory units running in parallel. If we use parallel composition we can obtain a memory with  $n$  cells:  $(Q^n, Q^n, Q^n, \delta, \beta, q_0^n) = (Q^n, \delta, \beta, q_0^n)$  where  $\delta : Q^n \times Q^n \rightarrow Q^n$  and  $\beta : Q^n \rightarrow Q^n$ . Let  $\beta_m : Q^n \times E \rightarrow Q^n$ .

Let  $E_m = \{(-, -, \dots, q_i, \dots, -) \mid q_i \in Q\} \subset Q^n$  such that only one memory cell receives an event at a certain time moment. And let  $OP, V_L, LOC$  be three sets of symbols such that  $OP = \{R, WR\}, V_L \subset Q, LOC \subset \mathbb{N}$ . Let  $f : OP \times V_L \times LOC \rightarrow E_m$  be a function such that

$$f(x, y, z) = \begin{cases} \lambda & \text{if } x = R \\ \underbrace{(-, -, \dots, y, \dots)}_z & \text{if } x = WR \end{cases}$$

and  $\delta_M : OP \times V_L \times LOC, \delta_M = \delta \circ f$ . We can now define a model for the memory that we’ll call *M-automaton*.

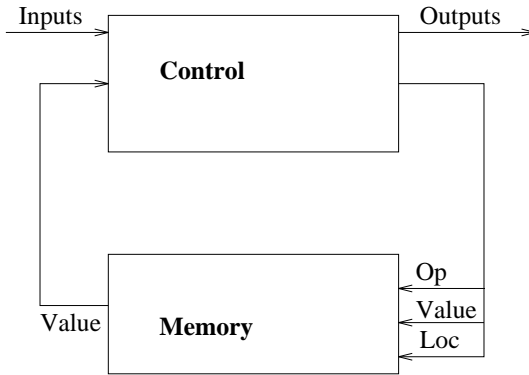


Fig. 1.8 Process abstraction

**Definition 1.8** An M-automaton is iDRTS,  $A_M = (\{OP, V_L, LOC\}, Q^n, Q, \delta, \beta, q_0^n)$  where

- E is the set of events  $E = OP \times V_L \times LOC$ , where:
  - $OP = \{R, W\}$  is a set of symbols called operations;
  - $V_L, LOC$  are finite sets of symbols called values and locations.
- $\delta$  is the next state function  $\delta : Q^n \times E \longrightarrow Q^n$
- $\beta$  is the output function  $\beta : Q^n \times LOC \longrightarrow Q$
- $q_0^n$  is the initial state:  $q_0^n \in Q^n$

We can now present a formal definition of the notion of a computational process. As previously said a computational process is considered to be composed of two parts that interact with each other: a control part that is described by a *C-automaton* and a memory part that is formalized by an *M-automaton*. Let us consider the two automata to be connected as depicted in Figure 1.8. This type of connection can be expressed as a combination of serial and feedback connections:  $P = \mathcal{F}(M \odot C)$ . Let us call the resulting system a *P-automaton*.

**Definition 1.9** Let

$$A_c = (T, \{E_c, V_R\}, Q_c, \{LOC, OP, V_L\}, Y_c, F_c, \delta_c, \{\beta_{1c}, \beta_{2c}\}, q_{0c})$$

be a C-automaton and

$$A_M = (\{OP, V_L, LOC\}, Q_M^n, Q_M, \delta_M, \beta_M, q_{0M}^n)$$

be a M-automaton. A **P-automaton** is a septuple

$$A_p = \mathcal{F}(A_C \odot A_M) = (T, E_p, Q_p, Y_p, \delta_p, \beta_p, q_{0p}, F_p, \{LOC, OP, V_L, V_R\})$$

resulting by the composition of the  $A_C$  and  $A_M$  where

- $Q_p = Q_C \times Q_M^n$  is the set of states, and  $q_{0p} = (q_{0C}, q_{0M}^n)$  is the initial state.
- $\beta_p$  is the output function.  $\beta_p((q_1, q_2)) = (\beta_{1C}(q))$
- $F_p = \{(f_C, q) | f_C \in F, q \in Q_M\} \subseteq Q_p$  is the set of final states.
- $E_p = E_C$  is the set of (external) events.
- $\delta_p$  is the next state function,

$$\delta : T \times T \times q_p \times E_p \longrightarrow Q_p,$$

$$\begin{aligned} \delta(t, t, (q_1, q_2), e) &= (\delta_C(t, q, ((e, v_R), t_e)), \delta_M(q_2, (op, v_L, loc))) \\ &= (\delta_C(t, q_1, ((e, \beta_M(q_2), t))), \delta_M(q_2, \beta_{2C}(q))) \end{aligned}$$

For a sequential process, its state is determined state by the values of its variables plus the state of the processor. For a *P-automaton* the state is the state of the control part plus the state of all the memory locations (values found in the memory cells). Therefore, using the above definition, the state of the process corresponds with the state of the *P-automaton*. This model is related with the *fair transition systems (FTS)* [Manna 89]. Let us consider the state set of *P-automata*  $Q_p = Q_c \times Q_M^n$ . A variable  $v \in Q_p$  is a tuple  $v = (v_c, v_M^1, \dots, v_M^n)$ . The composing elements of this variable are *state variables* of *FTS*. The transition  $\tau$  is the same as our function  $\delta$  and the *initial condition* from *FST* is our  $q_0$ . In [Manna 89], the authors add two more classes of restrictions to their models called *justice* and *fairness* in order to insure proper operation of the system.

From Definition 1.9 we can derive the definition of a reactive process.

**Definition 1.10** A **reactive process** is a sequential process that has an empty set of final states.

A typical concurrent system has a variable number of processes during its life-time that are created and killed at the request of other processes. The program that does this work is called *operating system (OS)*. After a process is created the relation between the processes that asked for the service and the new process is determined by the specifics of the system.

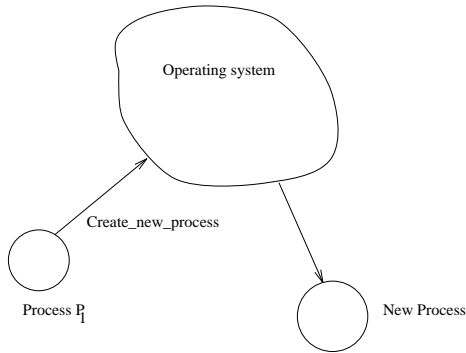


Fig. 1.9 Creation of new processes

From the point of view of process creation, the *operating system* can be abstracted by considering an entity that contains a set of *P-automaton*. This is the set of processes that the system can create. Furthermore, an automaton that has a special type of next state function is needed to start a new process when needed. When the *OS* automata receives an event requesting a new process it will make not only an internal transition, but will also start another *P-automata*. Let us call the abstraction of the *OS* an *S-automaton*.

**Definition 1.11** An S-automata is septuple

$$A_{s_i} = (T, E, Q, \delta, Y, \mathcal{A}_P, \beta, q_0)$$

where:

- $T$  is a set of time values,
- $E$  is a set of events,
- $Q$  is a set of states with  $q_0 \in Q$  the initial state,
- $\mathcal{A}_P$  is a set of P-automata and  $Y$  a set of output values,
- $\delta : T \times T \times Q \times E \rightarrow Q$  is the next state function,
- $\beta$  is the output function,  $\beta : Q \rightarrow Y \times \mathcal{A}_P$ ,

An *S-automaton* is thus, a DES whose set of output values is a set of *P-automata*. The *S-automaton* can output two types of actions: one is the same as in the case of iDRTS, a simple output, while the other is a new process. The set  $T$  is considered to be the set of real numbers as in all the

other formalization discussed before. The new process will start its work in its initial state. One major difference between this abstraction and a real system is that once it starts a new process the operating system loses control over it and cannot stop it. Although this might seem like a major drawback, in the authors' opinion it is not. To be more precise, once the process is started, if the process has a reactive behavior it should not stop.

## **1.7 Conclusions**

In this chapter we describe a discrete model for real-time systems and prove that it can be casted in the universal algebra framework. Furthermore the parallel composition operator can be described in the algebraic framework using direct product morphisms and sub-algebras.