On the Operational Semantics of Timed Rewrite Systems

Jérémie Blanc
Jeremie.Blanc@imag.fr

Rachid Echahed
Rachid.Echahed@imag.fr

Laboratoire LEIBNIZ – Institut IMAG, CNRS
46, avenue Felix Viallet, F-38031 Grenoble, France

Abstract

We propose an efficient operational semantics for a new class of rewrite systems, namely Timed Rewrite Systems. This class constitute a conservative extension of first-order conditional term rewrite systems together with time features such as clocks, signals, timed terms, timed atoms and timed rules. We define first Timed Rewrite Systems and illustrate them through some examples. A naive approach to the operational semantics is very costly in space. We propose, for a large class of programs, an improved calculus with a linear space complexity. Finally, we show how our framework compares to related work.

1. Introduction

Rewrite Systems constitute the foundation of several declarative (functional and/or logic) programming languages. They benefit from several analysis and proof techniques as well as efficient implementations. However, such term or graph rewrite systems fail to specify in a natural way real-world applications where *time* should be described. Consider for instance the boolean operator Alarm with the following profile device_sort$\longrightarrow$bool such that Alarm(device) is true whenever device has been broken for the last ten seconds. Unfortunately, such simple programs cannot be described rigorously using classical rewrite systems.

In this paper, we define timed rewrite systems as a class of rewrite systems which allows one to specify declaratively applications where the notion of time, should it be qualitative or quantitative, is involved. Our approach is new and departs from the proposals already made in order to add time into some declarative languages such as `tcc` [12], `Templog` [1], and `Chronolog` [14]. Roughly speaking, a timed rewrite system is provided with user-defined and incoming signals like in synchronous languages, e.g. [7]. A signal is a stream of pairs (ticks, values) that happen over time. We assume given a canonical signal or clock, noted REF which serves as a reference for other signals.

In general, operator definitions within a timed rewrite system or program $\mathcal{P}$ may depend on time. Thus, at each instant i, of REF, a new rewrite system consisting of operator definitions is updated in the same way as in data-flow languages. We obtain then, a stream of rewrite systems, $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_i, \ldots$. At any instant i, $\mathcal{F}_i$ constitutes a classical rewrite system which may be used as usual, e.g., simplification of expressions or resolution of goals.

In order to make easier the description of such timed rewrite systems, we enrich first-order terms by a temporal operator at which is used to refer to past definitions. For example the expression $at(-3\ seconds)f(0) + f(1)$ is meant to add the value of $f(0)$ three seconds before now and the value of $f(1)$ at the current instant. On the other hand, we also enrich classical atoms by allowing the use of the box $\Box$ and diamond $\Diamond$ modal operators over intervals. A timed atom $\Box_I B$, respectively $\Diamond_I B$, holds iff the atom B holds at every instant, respectively at one instant at least, of the time interval I. The rewrite rules we consider in this paper have the following general shape $lhs \rightarrow trhs \Leftarrow Body$ **when** $Tail$ where the term $trhs$ can be timed, $Body$ and $Tail$ are conjunctions of possibly timed atoms or equations. The role of tails is to filter at every instant, i, the rewrite rules that constitute the rewrite system $\mathcal{F}_i$.

The rest of the paper is organized as follows. The next section introduces timed rewrite systems. Then, we briefly sketch a first semantics based on labelled transition systems the details of which can be found in [4]. Our intention is to define efficient and realistic programs which may be interactive and reactive. For that, we describe in section 4 an optimised operational semantics. In section 5, we compare our proposal to related work and conclude the paper. Due to lack of space, proofs have been omitted as well as some formal definitions. The missing definitions and proofs could be consulted in a more detailed version [3].

Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)
1530-1311/02 $17.00 © 2002 **IEEE**

2. Timed Rewrite Systems

We define a *timed rewrite system* or *program* as a pair $\mathcal{P} = < \Sigma , R >$ where Σ is a timed first order signature and R is a set of timed conditional rewrite rules.

A *timed first order signature* is a triple $< S , \mathcal{S} , \Omega >$ where S is a set of sorts, $\mathcal{S}$ is an S^+-sorted family of signals and Ω is an S^+-sorted family of timed operators. S^+ stands for the set of non empty strings over S. We assume that S contains at least the sort of booleans, bool, and the sort of naturals, nat.

The meaning of a signal in our approach is very close to the ones introduced in some synchronous languages, e.g., [7, 5]. A *signal* S of sort $s_1 , \ldots , s_n , s$, noted S : $s_1 , \ldots , s_n \longrightarrow s$ is a stream of operator denotations. These denotations are associated implicitly to some instants over time. These instants constitute a set called ticks of S. Hence, our notion of time is linear and multi-form. Every signal S is associated to a boolean operator noted !S which is equal to true on the ticks of the considered signal S.

The family $\mathcal{S}$ contains a distinguished signal we call *reference signal* of profile REF : $\longrightarrow$ nat. More precisely, REF is the finest signal i.e. for all signals S in $\mathcal{S}$, the ticks of S are mapped into those of REF. At the beginning, the value of REF is 0, then it increments by one at each instant. Its value may be useful as a clock which gives the number of the current instant.

Starting from a timed signature, we introduce the notion of *timed terms* in order to take into account the "temporization" of operators thanks to the presence of signals. The definition of an operator, say ω, may change at every instant. We refer to a past definition of an operator by using a special temporal operator denoted *at* which extends the *pre* operator of `Lustre` [5] and $ operator of `Signal` [7]. We write $at\,(-n\ \mathrm{S})\ \omega$ to refer to the operator ω at the instant corresponding to n ticks of signal S before the current instant. In general, we can use several at-expressions to define the right instant in the past of an operator. For that we use *timed operator expressions* or *toe* for short, as defined below: $\phi \ ::= \ \omega \mid at\,(-n\ \mathrm{S})\ \phi$ where ω is an operator, n is a positive natural number, S is a signal, ϕ is a toe. The toes of the form $at\,(-n\ \mathrm{S})\ \phi$ are called past toes. *Timed terms* extend classical ones by referring to past definitions of operators. They are defined as follows : $tt \ ::= \ x \mid \phi\,(u_1 , \ldots , u_m)$ where x is a variable, m is a natural number ($m \geq 0$), ϕ is a toe and $u_1 , \ldots , u_m$ are well-sorted timed terms.

Formulas in R are timed conditional rewrite rules of the following form: $lhs \rightarrow trhs \Leftarrow B$ **when** C where lhs is a first order term, $trhs$ is a timed term, B and C are possibly empty conjunctions of timed atoms. $lhs \rightarrow trhs$ is called *head*, B *body* and C *tail*. If lhs is of the form $\omega(u_1 , \ldots , u_m)$, we say that the considered timed conditional rewrite rule defines ω. A timed atom is either an equation $u_1 == u_2$ where u_1 and u_2 are two timed terms, a formula $\Diamond|_1\ b_1\ ..\ b_2\ |_2\ B'$ which holds whenever the timed atom B' is true at least once during the interval $|_1 b_1\ ..b_2|_2$, or a formula $\Box|_1\ b_1\ ..\ b_2\ |_2\ B'$ which holds whenever the timed atom B' is true at every instant of the interval $|_1\ b_1\ ..\ b_2\ |_2$. Atoms of the form $\Box|_1\ b_1\ ..\ b_2\ |_2\ B'$ (respectively, $\Diamond|_1\ b_1\ ..\ b_2\ |_2\ B'$) are called *past-box atoms* (respectively, *past-diamond atoms*). $|_1$ and $|_2$ stand either for the symbol [or for the symbol] indicating if the bounds are included or not in the interval, and b_1 and b_2 represent pairs noted $-n$ S where n is a non null natural number and S is a signal. We also use "now" as a particular form of b_2 to indicate the current instant. The role of tails is to filter the rewrite rules to be considered at each instant. A timed program $\mathcal{P}$ generates an infinite sequence of classical (atemporal) rewrite systems, $\mathcal{F}_t$, also called *stores* in the sequel, as depicted in Figure 1. Intuitively, at each instant t, $\mathcal{F}_t$ includes all the rewrite rules cl such that cl **when** C is in $\mathcal{P}$ and C holds at t. We write $\mathcal{F} \vdash B$ to note that atom B holds in the classical rewrite system $\mathcal{F}$. A timed operator ω is *atemporal* or *static*, if ω is defined classically without any reference to timed syntactic entities. Otherwise, ω is called *dynamic*. Notice that a classical term rewrite systems consists only of atemporal operator definitions.

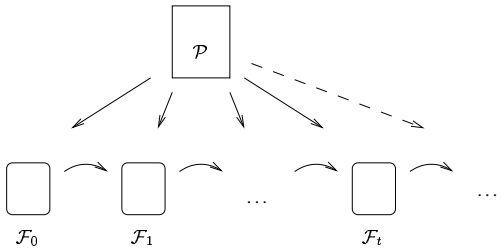


Figure 1. Store Generation

Below we give some toy examples in order to illustrate our framework.

Example 1 In this example, we give two different definitions of Fibonacci function. This function is often used to show the abilities of synchronous languages. In our framework, we can define Fibonacci function either by following synchronous style (fib1) or a pure declarative style (fib2). However, fib1 is a timed operator whose value will change during the execution whereas fib2 is an atemporal operator.

1. fib1 $\rightarrow$ 0 **when** REF == 0
2. fib1 $\rightarrow$ 1 **when** REF == 1
3. fib1 $\rightarrow$ at $(-1$ REF$)$ fib1 + at $(-2$ REF$)$ fib1
   when REF> 1 == true

1. fib2(0) $\rightarrow$ 0
2. fib2(1) $\rightarrow$ 1

Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)
1530-1311/02 $17.00 © 2002 **IEEE**

3. fib2(x) $\rightarrow$ fib2(x-1) + fib2(x-2) $\Leftarrow x > 1$ == true

Example 2 In this example, we consider the case of a controller which has to decide automatically whether an item, sliding over a belt, is to be discarded (because of any fault) or not. A decision concerning an item is taken when the item arrives at a particular position, p_0, detected by a laser ray (laser3). An item is to be discarded if a fault is detected (!reject == true) while the item was sliding between two particular positions p_1 and p_2. The controller detects an item entering position p_1 (resp. p_2) thanks to a laser ray laser1 (resp. laser2). Below we give a definition of the constant discard_item whose value is computed every time an item reaches position p_0, i.e. a tick of signal laser3 occurs.

discard_item $\rightarrow$ true
$\Leftarrow \Diamond[\ -1$ laser1 .. -1 laser2 $[$!reject == true
when !laser3 == true
discard_item $\rightarrow$ false
$\Leftarrow \Box[\ -1$ laser1 .. -1 laser2 $[$!reject == false
when !laser3 == true

3. Labeled Transition System

We assume in the sequel that a timed rewrite system $\mathcal{P}$ is given with signature $< S\ ,\ \mathcal{S}\ ,\ \Omega >$. As depicted in Figure 1, a run of a program $\mathcal{P}$ generates a stream of stores, $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_t, \ldots$. In this section, we give a brief overview of a first semantics based on a labelled transition system. The details of this semantics can be found in [4]. To each instant t, corresponds a state of such a transition system which contains, among other technical entities, a classical term rewrite system $\mathcal{F}_t$. Roughly speaking, the computation of $\mathcal{F}_t$ is done in four stages: $\mathcal{P}^\alpha$, $\mathcal{P}^\beta$, $\mathcal{P}^\gamma$ and $\mathcal{P}^\delta$.

$\mathcal{P}^\alpha$ is obtained from $\mathcal{P}$ by replacing all toes and past atoms by classical (atemporal) terms. A past toe of the form $at(-n\ \mathrm{s})\,\omega$ is replaced by a new operator noted $\lfloor\omega\rceil^{t'}$ where t' denotes the absolute instant corresponding to $at(-n\ \mathrm{s})$ with respect to instant t. We can define a function $absolute\_instant$ such that $t' = absolute\_instant(n\ ,\ \mathrm{s}\ ,\ t)$. When a past toe is of the general form $at\,(-n_1\ \mathrm{s}_1)\ \ldots at\,(-n_m\ \mathrm{s}_m)\,\omega$, we proceed likewise by iterating recursively the function $absolute\_instant$ to obtain the instant corresponding to $at\,(-n_1\ \mathrm{s}_1)\ \ldots at\,(-n_m\ \mathrm{s}_m)$. Whenever less than n ticks of s occurred since the beginning of the execution, $absolute\_instant$ is not defined for $(n\ ,\ \mathrm{s}\ ,\ t)$. Then, the timed rewrite rule in which the past toe $at\,(-n\ \mathrm{s})\,\omega$ occurs is eliminated. The elimination of a past atom, say B, of the form $B = \Diamond\ |_1 - n_1\ \mathrm{s}_1\ ..\ - n_2\ \mathrm{s}_2|_2\ \ B'$ consists in replacing B by a disjunction $\bigvee_{t' \in |_1 t_1\ ;\ t_2\ |_2} \langle B'\rangle^{t'}$ where $t_1 = absolute\_instant\ (n_1\ ,\ \mathrm{s}_1\ ,\ t)$, $t_2 = absolute\_instant\ (n_2\ ,\ \mathrm{s}_2\ ,\ t)$ and $\langle B'\rangle^{t'}$ is the result of the elimination of toes and past atoms in B' w.r.t. instant t'. If $absolute\_instant$ is not defined on $(n_1\ ,\ \mathrm{s}_1\ ,\ t)$, then we consider $t_1 = 0$. In addition, if $absolute\_instant$ is not defined on $(n_2\ ,\ \mathrm{s}_2\ ,\ t)$, then B is replaced by an atom noted FALSE which is always false. If the disjunction is empty then we consider B is replaced by FALSE too. The case of the past-box atoms, $\Box\ |_1 - n_1\ \mathrm{s}_1\ ..\ - n_2\ \mathrm{s}_2\ |_2\ B'$, is similar to past-diamond atoms: disjunctions become conjunctions and the atom FALSE becomes an atom noted TRUE which is always true.

Notice that $\mathcal{P}^\alpha$ contains neither the definitions of the introduced operators, nor the definitions of the incoming signals. The second stage which leads to $\mathcal{P}^\beta$ consists to complete $\mathcal{P}^\alpha$ with additional rewrite rules defining the newly introduced operators and incoming signals. The third stage consists in filtering the timed rewrite rules. If a timed rewrite rule ϕ **when** C of $\mathcal{P}^\beta$ is such that C holds, then ϕ occurs in $\mathcal{P}^\gamma$; otherwise ϕ is erased. $\mathcal{P}^\gamma$ is built stepwise according to a certain stratification of timed operators in Ω. $\mathcal{P}^\delta$ is $\mathcal{P}^\gamma$ in which the definitions of the timed constants (i.e. timed operators without arguments) have been simplified. All these stages are performed under the following assumptions:

Requirement 1 The tails of each timed rewrite rule of a given stratum are expressed through the operators of lower strata.

Requirement 2 Simplification of timed constants ω must be terminating in $\mathcal{P}^\gamma$, at each instant. At each instant, the definition of a timed function (i.e. timed operator with arguments) ω must not depend on a past definition of ω.

The execution model of a timed program $\mathcal{P}$ can be given by a labelled transition system the state of which are pairs $< \mathcal{H}_t\ ,\ \mathcal{F}_t >$ where $\mathcal{F}_t$ is the store computed at instant t and $\mathcal{H}_t$, called *history*, consists of the stores computed before t. Elements of $\mathcal{H}_t$ are used to bring into $\mathcal{P}^\beta$ the definitions of the new operators, introduced in $\mathcal{P}^\alpha$, of the form $\lfloor\omega\rceil^{t'}$ with $t' < t$. The transitions of the labelled transition system are of the form $< \mathcal{H}_t\ ,\ \mathcal{F}_t > \stackrel{In_{t+1}}{\longrightarrow} < \mathcal{H}_{t+1}\ ,\ \mathcal{F}_{t+1} >$ where In_{t+1} consists of the definition of the incoming signals. The history being not bounded, the definition of the labelled transition system as sketched above is not realistic. So, we propose in the following section a new tractable semantics.

4. Actual Operational Semantics

In this section, we propose some sufficient conditions under which we can develop an operational semantics using a bounded space memory. As in the previous section, we will show how current stores are generated following four stages, $\mathcal{P}^{\alpha\prime}$, $\mathcal{P}^{\beta\prime}$, $\mathcal{P}^{\gamma\prime}$ and $\mathcal{P}^{\delta\prime}$.

Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)
1530-1311/02 $17.00 © 2002 **IEEE**

The first stage generates a timed program $\mathcal{P}^{\alpha\prime}$ without any reference to toes or past atoms just like $\mathcal{P}^{\alpha}$. The difference with $\mathcal{P}^{\alpha}$ is that the past toes such as $\phi = at\,(-n\ \mathrm{S})\,\omega$ are replaced by a new operator noted $\lfloor at\,(-n\ \mathrm{S})\,\omega \rceil$ of the same profile as ω and the past atoms B of the form $\Box\ |_1\ -n_1\ \mathrm{S}_1\ ..\ -n_2\ \mathrm{S}_2\ |_2\ u_1 == u_2$ or $\Diamond\ |_1\ -n_1\ \mathrm{S}_1\ ..\ -n_2\ \mathrm{S}_2\ |_2\ u_1 == u_2$, are replaced by equations $\lfloor B \rceil == true$ where $\lfloor B \rceil$ is a new boolean operator. The calculus of the definition of $\lfloor B \rceil$ will be possible at every instant, if the following requirement is verified by B.

Requirement 3 Every past atom of the form $\Box\ |_1\ -n_1\ \mathrm{S}_1\ ..\ -n_2\ \mathrm{S}_2\ |_2\ B'$ or $\Diamond\ |_1\ -n_1\ \mathrm{S}_1\ ..\ -n_2\ \mathrm{S}_2\ |_2\ B'$ is closed (i.e. it contains no free variable) and is such that B' can be decided in a sensible computational duration at each instant.

In the following, we call *new operators* the operators $\lfloor \phi \rceil$ and $\lfloor B \rceil$ in $\mathcal{P}^{\alpha\prime}$. Note that the past toes of the form $at\,(-n_1\ \mathrm{S}_1)\ \ at\,(-n_2\ \mathrm{S}_2)\ \ \omega$ are transformed into $\lfloor at\,(-n_1\ \mathrm{S}_1)\,\omega' \rceil$ where ω' is $\lfloor at\,(-n_2\ \mathrm{S}_2)\,\omega \rceil$, and the past atom of the form $\Diamond_{I_1}\ \Diamond_{I_2}\ B$ are transformed into $\lfloor \Diamond_{I_1} B' \rceil == true$ where B' is $\lfloor \Diamond_{I_2} B \rceil == true$. So, all the new operators are of the form $\lfloor at\,(-n\ \mathrm{S})\,\omega \rceil$, $\lfloor \Diamond_I B \rceil$ or $\lfloor \Box_I B \rceil$ where ω is an operator and B an atom.

Example 3 The application of the first stage to the specification of fib1 given in Example 1 has the following result: $\mathcal{P}^{\alpha\prime}$ contains the timed rewrite rules 1, 2 and 3′. 1 and 2 are copied without any change from the timed program because they contain neither past toes, nor past atoms. On the other hand, 3′ is the transformation of 3 into

3'. fib1 → ⌊at(−1 REF) fib1⌉ + ⌊at(−2 REF) fib1⌉
when REF> 1 == true

Example 4 In the same way as in the latter example, the rewrite rules defining discard_item given in Example 2 becomes:

discard_item → true
⇐ ⌊◇[−1 laser1 .. −1 laser2 [!reject == true⌉
== true **when** !laser3 == true

discard_item → false
⇐ ⌊□[−1 laser1 .. −1 laser2 [!reject == false⌉
== true **when** !laser3 == true

Generation of $\mathcal{P}^{\beta\prime}$ consists in adding to $\mathcal{P}^{\alpha\prime}$ the rewrite rules defining the new operators and the incoming signals. Now, we show how the rules concerning the new operators are synthesized. We tackle at first the new operators of the form $\lfloor at\,(-n\ \mathrm{S})\,\omega \rceil$ corresponding to past toes. For such a new operator, we associate a past toe memory, denoted $\mathcal{M}em_{at\,(-n\ \mathrm{S})\,\omega}$, defined as follows.

Definition 1 Let $\phi = at\,(-n\ \mathrm{S})\ \omega$ be a past toe and t be the current instant. The *past toe memory* $\mathcal{M}em_{\phi}$ at t for the new operator $\lfloor \phi \rceil$ is a queue (FIFO) of length n of sets of rewrite rules defining ω noted $[\ d_n \ldots d_2\ d_1\ ]$.

d_n should contain at each instant, a set of rewrite rules from which we can define the new operator $\lfloor at\,(-n\ \mathrm{S})\,\omega \rceil$. For that, we require that past toe memories satisfy the following invariant.

Definition 2 Let $\phi = at\,(-n\ \mathrm{S})\ \omega$ be a past toe. Assume that $\mathcal{M}em_{\phi} = [d_n\ \ldots\ d_2\ d_1]$. The *invariant* for $\mathcal{M}em_{\phi}$ is the following assertion: "For all $i \in \{\ 1\,;\ldots;n\ \}$, if at least i ticks of signal S have occurred since instant 0, d_i is the definition of ω, i ticks of S ago, otherwise d_i is empty".

In order to preserve the invariant, we give in Figure 2 the inference rules to be applied on past toe memories at every instant.

Since the past toes are not defined at the first instant, the memories initialized by Rule $(\mathsf{at}-0)$ verify the invariant. The two other inference rules Rule $(\mathsf{at}-1)$ and Rule $(\mathsf{at}-2)$ are rules which update the past toe memories in a rather straightforward manner according to the cases whether a tick of signal S occurs or not at the previous instant $t-1$, i.e., whether $\mathcal{F}_{t-1} \vdash !\mathrm{S} == true$. All the sets D are finite since Requirement 2 is verified. The addition of a temporal annotation to the operators of D does not change the value which represents ω. It allows one to make the distinction between $\lfloor \omega' \rceil^{t'}$ in $\mathcal{M}em_B$ which represents the value of ω' at some instant $t' < t$ and ω' in $\mathcal{P}^{\beta}$ which represents the current value of ω' i.e. the value of ω' at instant t.

Example 5 Consider the memory associated to the new operator ⌊at(−2 REF) fib1⌉ occurring in the timed rewrite rule, say 3', of Example 3. Figure 3 indicates all the states of the memory associated to the new operator ⌊at(−2 REF) fib1⌉ for the instant 0 until 3. From instant 1, $(\mathsf{at}-2)$ is always applied because !REF == true is always true. At each

$$(\mathsf{at}-0)\ \frac{}{[\ \emptyset\ \ \ldots\ \ \emptyset\ \ \emptyset\ ]}\quad \text{if } t = 0$$

$$(\mathsf{at}-1)\ \frac{[\ d_n\ \ \ldots\ \ d_2\ \ d_1\ ]}{[\ d_n\ \ \ldots\ \ d_2\ \ d_1\ ]}\quad \text{if } \begin{cases} t > 0 \\ \mathcal{F}_{t-1} \vdash \\ !\mathrm{S} == false \end{cases}$$

$$(\mathsf{at}-2)\ \frac{[\ d_n\ \ \ldots\ \ d_2\ \ d_1\ ]}{[\ d_{n-1}\ \ \ldots\ \ d_1\ \ D\ ]}\quad \text{if } \begin{cases} t > 0 \\ \mathcal{F}_{t-1} \vdash \\ !\mathrm{S} == true \end{cases}$$

where D stands for the complete definition of ω at the previous instant $t-1$ in which we replace all the timed operators ω' without temporal annotation and different from ω by new operators of the form $\lfloor \omega' \rceil^{t-1}$ and $[\ d_n\ \ldots\ d_2\ d_1\ ]$ is the memory of the past toe $at\,(-n\ \mathrm{S})\,\omega$ at instant $t-1$.

Figure 2. Past toe memory rules

Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)
1530-1311/02 $17.00 © 2002 **IEEE**

t	Rule	Memory state
0	$(\text{at} - 0)$	[$\emptyset$ $\emptyset$]
1	$(\text{at} - 2)$	[$\emptyset$ { fib1 == 0 }]
2	$(\text{at} - 2)$	[{ fib1 == 0 } { fib1 == 1 }]
3	$(\text{at} - 2)$	[{ fib1 == 1 } { fib1 == 1 }]

Figure 3. States of $\mathcal{M}em_{at\,(-2\ \text{REF}\ )\,fib1}$

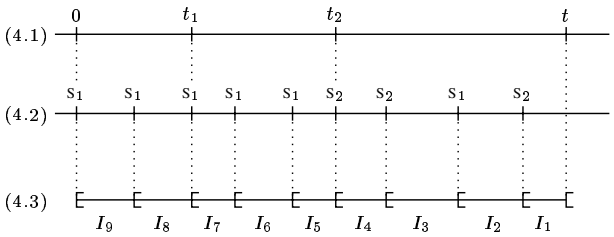


Figure 4. Interval partition

instant, we can verify that element d_2 is the definition of fib1 two instants ago, and element d_1 is the definition of fib1 at the previous instant.

For a memory [d_n ... d_2 d_1] associated to a past toe $\phi = at\,(-n\ \mathrm{s}\,)\,\omega$ which verifies the invariant, the definition of ϕ is d_n in which we replace the timed operator ω by the new operator $\lfloor at\,(-n\ \mathrm{s}\,)\,\omega \rceil$. Since the memory verifies the invariant at every instant, the calculus of the definition of ϕ is always possible.

Now, we consider how it is possible to compute the truth-values of past atoms within a bounded space. We illustrate our algorithm on the case of formulas, B, of the form $\Diamond[\ -n_1\ \mathrm{s}_1\ ..\ -n_2\ \mathrm{s}_2\ [\ B'$. Recall that, according to the semantics given in the previous section, the truth-value of $\lfloor B \rceil^t$ is equal to the disjunction $\bigvee_{t' \in [t_1,t_2[} \lfloor B' \rceil^{t'}$ where t_1 (resp. t_2) is the absolute instant corresponding to $at\,(-n_1\ \mathrm{s}_1\,)$ (resp. $at\,(-n_2\ \mathrm{s}_2\,)$).

The length of $[t_1, t_2[$ is not bounded, and so is the number of elements occurring in the disjunction $\bigvee_{t' \in [t_1,t_2[} \lfloor B' \rceil^{t'}$. To overcome this issue, we propose a way to split the interval $[t_1, t_2[$ into subintervals so that the computation of the value of $\lfloor B \rceil^t$ becomes feasible using a bounded space memory. Let $I_1, \ldots, I_k$ be a partition of $[t_1, t_2[$. We associate a constant w_i for every interval I_i. w_i corresponds to the value $\bigvee_{t' \in I_i} \lfloor B' \rceil^{t'}$. The value of $\lfloor \mathrm{B} \rceil^t$ is thus equal to $\bigvee_{i \in [1,k]} w_i$.

Such a partition changes over time and should be available at every instant. Hence, we should keep preparing the forthcoming partitions. Our proposal consists in partitioning at every instant t, the interval $[0, t[$ as follows:

Definition 3 Let τ and τ' be two instants less than t. Assume $\tau' < \tau$. τ and τ' belong to the same interval iff

1. either $\tau, \tau' \geq t_2$ (with $t_2 = absolute\_instant(\,n_2\,,\,\mathrm{s}_2\,,\,t\,)$) and for all instants τ'' such that $\tau' < \tau'' \leq \tau$, $\mathcal{F}_{\tau''} \vdash !\mathrm{s}_1 == false$ and $\mathcal{F}_{\tau''} \vdash !\mathrm{s}_2 == false$.
2. or $\tau, \tau' < t_2$ and for all instants τ'' such that $\tau' < \tau'' \leq \tau$, $\mathcal{F}_{\tau''} \vdash !\mathrm{s}_1 == false$.
3. or t_2 does not exist (i.e., not enough s_2 ticks) and for all instants τ'' such that $\tau' < \tau'' \leq \tau$, $\mathcal{F}_{\tau''} \vdash !\mathrm{s}_1 == false$ and $\mathcal{F}_{\tau''} \vdash !\mathrm{s}_2 == false$.

In the previous definition, item (1) suggests to partition the interval $[t_2, t[$ according to the ticks of s_1 or s_2. That is to say, $[t_2, t[= [t_2, \tau_1[\cup [\tau_1, \tau_2[\cup \cdots \cup [\tau_m, t[$ such that the set $\{\tau_1, \ldots, \tau_m\}$ consists of all the instants which coincide with a tick of s_1 or s_2 in the interval $[t_2, t[$. Item (2) suggests to partition the interval $[0, t_2[$ according to the ticks of s_1. That is to say, $[0, t_2[= [0, \tau_1[\cup [\tau_1, \tau_2[\cup \cdots \cup [\tau_m, t_2[$ such that the set $\{\tau_1, \ldots, \tau_m\}$ consists of all the instants which coincide with a tick of s_1 in the interval $[0, t_2[$. Whenever t_2 is not defined, item(3) suggests to partition the interval $[0, t[$ according to the ticks of s_1 or s_2.

In the sequel we consider that the intervals, $(I_i)_i$, defined by Definition 3 are numbered from the most recent to the oldest as depicted in Figure 4. Thus, 1 is the index of the interval which has the form $[b1, t[$ where $b1$ is an instant of the past, i.e. $b1 \in [0, t[$. We will denote by z, l and r the indices of the following intervals of the partition: I_z is of the form $[0, bz[$, I_l is of the form $[t_1, bl[$ and I_r is of the form $[t_2, br[$ where bz, bl and br are instants of $[0, t[$.

We can show (see [3]) that at most the last $n_1 + n_2$ intervals, say, $I_{n_1+n_2}, \ldots I_1$ are sufficient to compute the truth-value of the past atom B.

In practice, we introduce a data structure called past atom memory and denoted $\mathcal{M}em_B$, which allows us to evaluate B at every instant. Informally, this data structure encodes either the last $n_1 + n_2$ intervals of the partition defined above as well as the evaluation of $\Diamond_{I_i} B'$ for every interval, I_i, of the considered partition.

Definition 4 Let $B = \Diamond\ [\ -n_1\ \mathrm{s}_1\ ..\ -n_2\ \mathrm{s}_2\ [\ B'$ be a past atom and t be the current instant. The *past atom memory* $\mathcal{M}em_B$ at t is a triple $< v\,,\ x^1\,,\ x^2 >$ of sequences of $n_1 + n_2$ truth values of a 3-valued logic ($\top$ -true-, $\bot$ -false-, u -undefined-). We represent a past atom memory $\mathcal{M}em_B$ by the following figure:

$$\begin{bmatrix} v_{n_1+n_2} \ldots v_2 & v_1 \\ x^1_{n_1+n_2} \ldots x^1_2 & x^1_1 \\ x^2_{n_1+n_2} \ldots x^2_2 & x^2_1 \end{bmatrix}$$

The interpretation of elements v_i, x^1_i, x^2_i is defined by the following invariant over the data structure $\mathcal{M}em_B$.

Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)
1530-1311/02 $17.00 © 2002 **IEEE**

Definition 5 Let $B = \Diamond [\ n_1\ \mathrm{s}_1\ ..n_2\ \mathrm{s}_2\ [\ B'$ be a past atom and $\mathcal{M}em_B$ be the past atom memory for B at t. Let $\{ I_z ; \ldots ; I_1 \}$ be the interval partition at t. The *invariant* for $\mathcal{M}em_B$ is the conjunction of the following assertions:

1. "(if t_1 is defined and $i \leq l$) or (if t_1 is not defined and $i \leq z$), then
   - v_i is the disjunction of the truth value of B' on the interval I_i,
   - $x_i^1 = \top$ if I_i begins with a tick of s_1 and $x_i^1 = \bot$ otherwise,
   - and $x_i^2 = \top$ if I_i begins with a tick of s_2 and $x_i^2 = \bot$ otherwise."
2. "$i > z$ iff $v_i = x_i^1 = x_i^2 = \mathsf{u}$."

Example 6 Consider Example 4 and suppose that the ticks of laser1 happened only at the instants 5, 11 and 16, the ticks of laser2 happened only at 7 and 14, a tick of laser3 happened only at 12. $absolute\_instant(\,1\,,\,\mathsf{laser2}\,,\,16\,) = 14$. Thus, the interval partition is $\{\,[1,4]\,;\,[5,10]\,;\,[11,13]\,;\,[14,15]\,\}$. Since the interval partition has to be verified, the memory for $\Diamond[\ -1$ laser1 .. -1 laser2 [!reject == true must be:

$$\begin{bmatrix} \top & \bot \\ \top & \bot \\ \bot & \top \end{bmatrix} \tag{1}$$

From the definition of $\mathcal{M}em_B$, it is easy to compute at each instant the indices l and r by using the sequences x^1 and x^2 and then deduce the value of B from the sequence v. We give in Definition 6 a formal definition of the computation of the value of B. For that, we introduce first a couple of technical definitions. We note by $Index(\,n\,,\,x\,,\,i\,)$ where $n, i \in \mathbf{N}, n > 0, 1 \leq i \leq n_1 + n_2$ and x is either x^1 or x^2, the lowest index i' over x such that there are exactly n occurrences of $\top$ in $x_{i'}, \ldots, x_i$. Roughly speaking, $Index(\,n\,,\,x^1\,,\,1\,)$ is the index of the interval beginning with $absolute\_instant(\,n\,,\,\mathrm{s}_1\,,\,t\,)$ where t stands for the current instant. Symmetrically, $Index(\,n\,,\,x^2\,,\,1\,)$ is the index of the interval beginning with $absolute\_instant(\,n\,,\,\mathrm{s}_2\,,\,t\,)$.

The formal calculus of the truth value of the past atoms uses an operation, say $\vee$, of the 3-valued logic. We note $\vee$, the unique extension of the boolean disjunction where u is the identity.

Definition 6 Let $B = \Diamond [\ n_1\ \mathrm{s}_1\ ..n_2\ \mathrm{s}_2\ [\ B'$ be a past atom and $\mathcal{M}em_B$ be the past atom memory for B at instant t which verifies the invariant (Definition 5). The value of B is defined by

- $B = \bigvee_{i \in [Index(\,n_2\,,\,x^2\,,\,1\,)+1, Index(\,n_1\,,\,x^1\,,\,1\,)]} v_i$ if $Index(\,n_1\,,\,x^1\,,\,1\,)$ and $Index(\,n_2\,,\,x^2\,,\,1\,)$ are defined.
- $B = \bigvee_{i \in [Index(\,n_2\,,\,x^2\,,\,1\,)+1, n_1+n_2]} v_i$ if $Index(\,n_1\,,\,x^1\,,\,1\,)$ is not defined and $Index(\,n_2\,,\,x^2\,,\,1\,)$ is defined,
- $B = false$ if $Index(\,n_2\,,\,x^2\,,\,1\,)$ is not defined.

The calculus of the definition of the new operator $\lfloor \mathsf{B} \rceil$ generated from the past atom B can easily be performed using Definition 6. The definition which we add to $\mathcal{P}^{\beta'}$ is $\lfloor \mathsf{B} \rceil \rightarrow$ true if the truth value of B is true. Otherwise, we add $\lfloor \mathsf{B} \rceil \rightarrow$ false.

Example 7 Consider the memory of

$$\Diamond[\ -1\ \mathsf{laser1}\ ..\ -1\ \mathsf{laser2}\ [\ \mathsf{!reject} == \mathsf{true}$$

given in Example 6. $Index\ (\,1\,,\,x^1\,,\,1\,) = 2$ and $Index\ (\,1\,,\,x^2\,,\,1\,) = 1$. Thus, the truth value of

$$\Diamond[\ -1\ \mathsf{laser1}\ ..\ -1\ \mathsf{laser2}\ [\ \mathsf{!reject} == \mathsf{true}$$

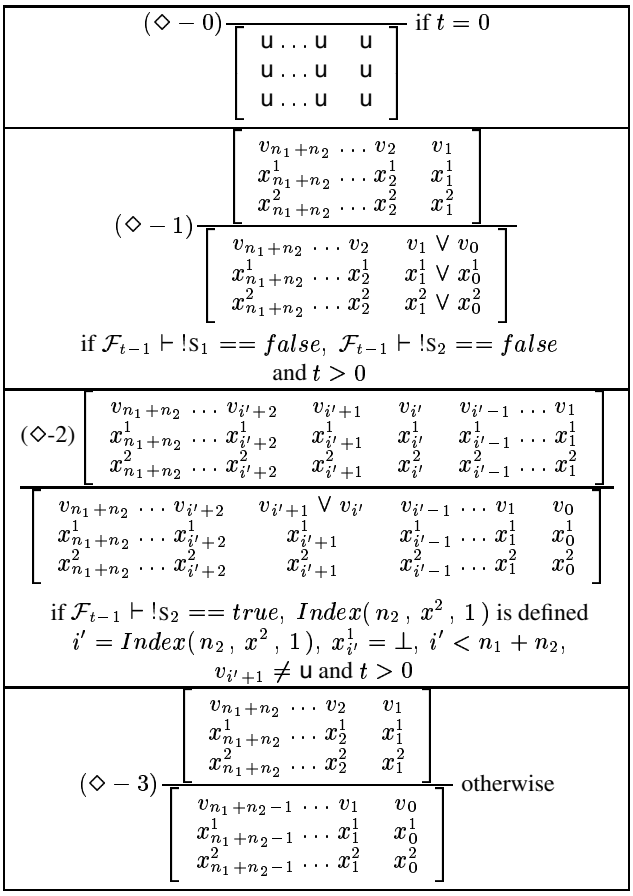
is v_2, i.e., true.

We focus now on the way $\mathcal{M}em_B$ is updated at each instant. In Figure 5, we give the inference rules which are applied at each instant. Rule $(\Diamond - 0)$ initialises the data structure with undefined values. Rule $(\Diamond - 1)$ is applied whenever both signal ticks of s_1 and s_2 are absent. In this case, interval I_1 is augmented by one instant, and thus the new value v_1 is equal to the value of v_1 at the previous instant augmented -using the $\vee$ operator- by the current value of B', say v_0. Rule $(\Diamond - 2)$ is a bit technical. Informally, Rule $(\Diamond - 2)$ is applied whenever a tick of signal s_2 occurs, t_2 shifts to next tick of s_2 to the right, and no tick of the signal s_1 happened at the previous t_2. Consider, for instance, the situation as depicted in Figure 4. Assume that t_2 moves to the left bound of interval I_3. In the new interval partition the previous intervals I_4 and I_5 are unified. i' and $i' + 1$ indicates the indices of such intervals to be unified. The new data corresponding to the unified interval is given by $v_{i'+1} \vee v_{i'}$, $x^1_{i'+1}$ and $x^2_{i'+1}$. Rule $(\Diamond - 3)$ is used whenever the three other rules cannot be applied. It creates a new interval, initialises the corresponding values, namely v_1, x_1^1 and x_1^2, and shifts all triples one step to the left. In this case, the values of the preceding instant corresponding to the $(n_1 + n_2)$th interval are lost.

Example 8 Consider Example 7. At 16, a tick of laser1 occurs. Thus, at 17, the rule $(\Diamond - 1)$ cannot be applied. In the same way, no tick of laser2 occurs at 16. Thus, at 17, the rule $(\Diamond - 2)$ cannot be applied. Hence, at 17, the rule $(\Diamond - 3)$ is applied to and transforms the past atom memory into

$$\begin{bmatrix} \bot & \bot \\ \bot & \top \\ \top & \bot \end{bmatrix}$$

Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)
1530-1311/02 $17.00 © 2002 IEEE



where v_0 stands for the truth value of B' in $\mathcal{F}_{t-1}$, x^1_0 for the truth value of $!s_1 == true$ in $\mathcal{F}_{t-1}$ and x^2_0 for the truth value of $!s_2 == true$ in $\mathcal{F}_{t-1}$. So, x^1_0 is true iff a tick of s_1 occurred at the previous instant, and symmetrically, x^2_0 is true iff a tick of s_2 occurred at the previous instant.

Figure 5. Past atom memory rules

$absolute\_instant(1, \text{laser2}, 17) = 14$. Thus, at 17, the interval partition is $\{ [1,4] ; [5,10] ; [11,13] ; [14,15] ; \{16\}\}$. Since $absolute\_instant\ (1, \text{laser1}, 17) = 16$, the invariant is still verified.

Now, suppose that, at 16, a tick of laser2 happens instead of a tick of of laser1. At 16, the rule $(\diamond - 1)$ cannot be applied. On the other hand, since $Index(1, x^2, 1) = 1$, $x^1_1 = \perp$ and $v_2 \neq$ u, the rule $(\diamond - 2)$ is applied and transforms the past atom memory into the one depicted in (1). $absolute\_instant(1, \text{laser2}, 17) = 16$. Thus, at 17, the interval partition is $\{ [1,4] ; [5,10] ; [11,15] ; \{16\}\}$. Since $absolute\_instant\ (1, \text{laser1}, 17) = 11$, the invariant is still verified.

The rules given in Figure 2 and Figure 5 must be applied to the corresponding memories at each instant t, before the store computation. Afterwards, the updated memories verify their invariants, and thus permit a sound computation of $\mathcal{P}^{\beta'}$ for the instant t. $\mathcal{P}^{\beta'}$ consists of the timed rewrite rules of $\mathcal{P}^{\alpha'}$ in addition to the definition of the incoming signals and the definition of the new operators which are calculated from their associated memories. $\mathcal{P}^{\gamma'}$ consists of the rewrite rules in $\mathcal{P}^{\beta'}$ whose tail is either true or absent. If all the tails of the timed rewrite rules defining a timed operator ω are false, we define ω in $\mathcal{P}^{\gamma'}$ by its remanent definition borrowed from the store of the previous instant. The rules of $\mathcal{P}^{\delta'}$ or $\mathcal{F}_t$ are obtained from $\mathcal{P}^{\gamma'}$ in the same way as $\mathcal{P}^{\delta}$ by simplifying the rules.

The soundness of this actual operational semantics is stated in following.

Theorem 1 Let $\mathcal{P}$ be a timed rewrite system satisfying requirements 1, 2 and 3. At each instant t, for all timed operators ω, $\lfloor\omega\rceil^t$ in $\mathcal{P}^{\delta}$ and ω in $\mathcal{P}^{\delta'}$ represent the same operation.

5. Conclusion and Related work

We gave the broad outlines of a new kernel framework allowing conservative extensions of declarative rule-based programs with time. We proposed an efficient operational semantics so that we can tackle real-world applications. The implementation of our formalism is under progress and will be available soon.

Synchronous languages are specially designed for reactive programming [8]. The most representative synchronous languages are `Lustre` [5], `Signal` [7] and `Esterel` [2]. All of them are based on the notion of signal and enable to compile programs into finite state automata. However, these languages do not handle new abstract data types or functions as signals. Moreover, they do not provide powerful temporal operators such as $\Box$ and $\diamond$ over intervals. The closest languages to our formalism are the declarative ones, namely `Lustre` and `Signal`. The primitives of theses languages can be easily expressed in our formalism. As for `Esterel`, it rather offers an imperative style of programming and, as in our semantics, its signals are remanent unlike `Lustre` and `Signal`. Note that the compilation into a finite automaton is possible in our case too whenever the considered applications do not need more expressive machines.

The absence of time notion in `ccp` has led to the development of a new synchronous paradigm `tcc` [12] based on `ccp` and preserving its good properties. We share with `tcc` the principle of a very expressive evolving store. However, the way of expressing the store changes is completely different. In `tcc`, the stores are generated from processes and the process algebra is orthogonal to the formalism expressing formulas of the stores. Moreover, these formulas are

Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)
1530-1311/02 $17.00 © 2002 **IEEE**

atemporal and not remanent. Thus, the data transmission from an instant to another is performed explicitly by the combinator next of the process algebra. The timed rewrite system, we propose, is data-oriented. The mechanisms of data transmission, which are hidden from the user, could be rather complicated in the case of past atoms and they could manipulate definitions of operators which is not possible in `tcc`. A problem of `tcc` inherited from `ccp` is the negative information detection. At each instant, `tcc` may only detect an absence of information at the end of the store computation, and thus the reaction is put back one instant later. A solution was presented in [13]. The new paradigm is called `Default Timed cc` and enables to detect an absence of information and to react instantaneously. In compensation, the programs need to be verified statically to avoid causality loops, whereas it was not necessary for `tcc` programs ("paradox-free" property of [12]). Our store is a well-defined stratified rewrite system which may be adapted to a large scope of constraint. A boolean function may stand for a predicate solving the NOT problem in the logic programming paradigm and thus the negative information problem of `ccp`. On the other hand, `ntcc` [11] extends `tcc` by adding non-determinism. This paradigm has, among others, two operators noted ! and $\star$. !P means "P will be always true in the current store and in the future ones", $\star$P means "P will be true in the current store or in some future one". From a logic point of view, ! and $\star$ are the respective duals in the future of our operators $\Box$ and $\Diamond$. From a procedural point of view, ! and $\star$ can be considered as behavior generators, whereas $\Box$ and $\Diamond$ are behavior testers. ! and $\star$ can be applied just to a specific interval of time in future. However, this interval is expressed using the base time unit: no other granularity is provided.

Another class of programming languages related to our formalism is the temporal logic programming languages. An interesting overview is presented in [10]. The meaning of "temporal" for languages like `Templog` [1] and `Temporal Prolog` of Gabbay [6] is not the same as the one in the synchronous world. The time is indeed represented in a constraint system and does not constraint temporally the execution of the program. So, the causality of the condition in formulas is not preserved in these extensions of the logic programming, whereas it is in our paradigm. On the other hand, these languages have the temporal operators $\Box$ (always) and $\Diamond$ (eventually) but without scope over some time interval. `Chronolog` is a data-flow temporal logic language [14] designed to do efficient computation. However, this language only manages one clock. Since the need for a multi-granular time is very important in temporal programming, `Chronolog(MC)` [9] has been developed. Nevertheless, this language is not synchronous.

References

[1] M. Abadi and Z. Manna. Temporal logic programming. *J. of Symbolic Computation*, 8(3):277–295, September 1989.

[2] G. Berry and G. Gonthier. The Esterel programming language: Design, semantics and implementation. *Science of Computer Programming*, 19(2), 1992.

[3] J. Blanc and R. Echahed. Synchronous functional logic programming. Technical report, Laboratoire Leibniz, 2001. available at http://www-leibniz.imag.fr/PMP/TIME/Timed-programs.ps.

[4] J. Blanc and R. Echahed. Timed term rewrite systems. *ENTCS*, (64), 2002. URL: http://www.elsevier.nl/locate/entcs/volume64.html.

[5] C. Dumas-Canovas and P. Caspi. A PVS proof obligation generator for Lustre programs. In *7th Int. Conf. on Logic for Programming and Automated Reasoning*, volume 1955 of *LNAI*, 2000.

[6] D. Gabbay. *Modal and Temporal Logic Programming*, chapter 6, pages 197–237. Academic Press, 1987.

[7] P. L. Guernic, T. Gautier, M. L. Borgne, and C. L. Maire. Programming real time applications with signal. *Proceedings of the IEEE*, 79(9), 1991.

[8] N. Halbwachs. Synchronous programming of reactive systems. In *Computer Aided Verification (CAV'98)*, LNAI 1427, pages 1–16, 1998.

[9] C. Liu and M. A. Orgun. Dealing with multiple granularity of time in temporal logic programming. *J. of Symbolic Computation*, 22(5 and 6):699–720, 1996.

[10] M. A. Orgun and W. Ma. An overview of temporal and modal logic programming. In D. M. Gabbay and H. J. Ohlbach, editors, *First Int. Conf. on Temporal Logic*, LNAI 827, pages 445–479, July 1994.

[11] C. Palamidessi and F. Valencia. A temporal concurrent constraint programming calculus. Report RS-01-20, BRICS, University of Aarhus, June 2001.

[12] V. Saraswat, R. Jagadeesan, and V. Gupta. *Constraint Programming*, volume 131 of *the NATO Advanced Science Institute Series, Series F: Computer and System Sciences*, chapter Programming in Timed Concurrent Constraint Languages. Springer Verlag, 1994.

[13] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Timed default concurrent constraint programming. *Journal of Symbolic Computation*, 22(5 – 6):475 – 520, Nov – Dec 1996.

[14] K. Zhang and M. A. Orgun. Parallel execution of temporal logic programs using dataflow computation. In *Int. Conf. on Computing and Information*, pages 812–830, 1994.

Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)
1530-1311/02 $17.00 © 2002 **IEEE**