Adding Time to Functional Logic Programs

-Extended abstract-

Jérémie Blanc and Rachid Echahed

Laboratoire LEIBNIZ – Institut IMAG, CNRS
46, avenue Felix Viallet, F-38031 Grenoble, France
Tel: (+33) 4 76 57 48 91; Fax: (+33) 4 76 57 46 02
Jeremie.Blanc@imag.fr Rachid.Echahed@imag.fr

Abstract. We introduce a new framework which constitutes a conservative extension of functional logic programming together with time features such as signals, timed terms, timed atoms and timed clauses (rules). The resulting formalism combines in a very high level and uniform way the main features of functional and logic programming with new features regarding the use of time thanks to some temporal operators. We define first our framework by its syntax and semantics. Then, we tackle the problem of the operational semantics by defining an efficient calculus which is proven to be sound. Finally, we show how our framework compares to related work such as tccp, temporal logic programming and synchronous languages.

1 Introduction

Functional-logic programming languages have been intensively studied during the last years and realistic implementations are available nowadays, e.g. Curry [3]. However, the description of real-world applications requires, in general, more than classical features of pure functional and logic languages such as concurrency, state changes, communicating processes, mobility etc., see [10, 6].

In this paper, we extend functional-logic languages in order to deal with applications using quantitative notion of time (e.g., real-time reactive programming) or qualitative notion of time (e.g., temporal programming). Consider for instance two functions Alarm and Warning of the same profile device_sort$\longrightarrow$bool such that Alarm(device) (respectively, Warning (device)) holds (equals true) whenever device has been broken for the last ten seconds (respectively, has been broken at least once during the last ten seconds). Such simple programs cannot be described rigorously in the current frameworks underlying functional-logic languages.

We propose in this paper, a conservative extension of conditional equational logic [4] with time. Our approach is new and departs from the proposals already made in order to add time into (constraints) logic languages such as `tcc` [14], `Templog` [1], and `Chronolog` [16].

Roughly speaking, timed programs are provided, as in synchronous languages, e.g. [8], with a canonical signal or clock, noted SREF which serves as a reference for other possible user-defined or input signals. In general, operator definitions within a program $\mathcal{P}$ may depend on time. Thus, at each instant t, of

SREF, a new store consisting of operator definitions is updated in the same way as in data-flow languages. We obtain then, a stream of stores, $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_t, \ldots$. At any instant t, store $\mathcal{F}_t$ constitutes a functional-logic program which may be used to solve goals as usual, e.g., solutions of the equation $\mathsf{Alarm(m)} == \mathsf{true}$ consist of the current devices which have been broken for the last ten seconds.

In order to make easier the description of such timed programs, we enrich first-order term 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. 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 program clauses we consider in this paper have the following general shape $t_1 == t_2 \Leftarrow Body$ **when** $Tail$ or $t_1 == t_2 \Leftarrow Body$ **as soon as** $Tail$ where terms can be timed, $Body$ and $Tail$ are conjunctions of possibly timed atoms or equations. The role of tails is to filter at every instant, t, the clauses that constitute the store, $\mathcal{F}_t$.

The rest of the paper is organized as follows. The next section introduces timed functional logic programs. Then, we briefly introduce a denotational semantics of our programs based on labeled transition systems. Our intention is to define efficient and realistic programs which may be interactive and reactive. For that, we describe in section 4 an optimized 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.

2 Timed Functional Logic Programs

A timed program in our framework is a pair $\mathcal{P} = <\Sigma, E>$ where Σ is a timed first order signature and E is a set of timed conditional equations. 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., [8, 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 SREF : $\longrightarrow$ nat. More precisely, SREF is the finest signal i.e. for all signals S in $\mathcal{S}$, the ticks of S are mapped into those of SREF. At the beginning, the value of SREF 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` [8]. 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 expression* 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 E are timed conditional equations of the following forms:

1. $lhs == trhs \Leftarrow B$
2. $lhs == trhs \Leftarrow B$ **when** C
3. $lhs == trhs \Leftarrow B$ **as soon as** C

where lhs is a first order term, $trhs$ is a timed term, B and C are possibly empty conjunctions of timed atoms. $lhs == trhs$ 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 equation 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 of the interval are closed or open, and b_1 and b_2 represent pairs noted $-n\,\mathrm{S}$ 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 equations to be considered at each instant. A timed program $\mathcal{P}$ generates an infinite sequence of non-temporal programs, $\mathcal{F}_t$ as depicted in Figure 1. Intuitively, at each instant t, $\mathcal{F}_t$ includes:

- all the conditional equations e without tail in $\mathcal{P}$,
- all the conditional equations e such that e **when** C is in $\mathcal{P}$ and C holds at t,
- all the conditional equations e such that e **as soon as** C is in $\mathcal{P}$ and C holds at t, whereas it does not hold at the previous instant, i.e, in $\mathcal{F}_{t-1}$.

We call *atemporal operator*, an operator ω which is defined classically without any reference to timed syntactic entities. Notice that a classical functional-logic program consists only of atemporal operator definitions.

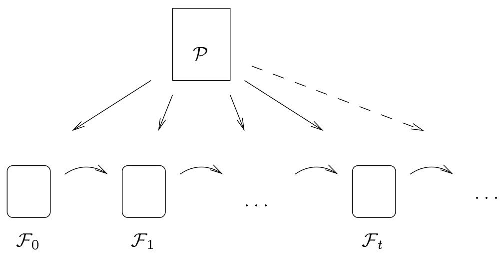

Fig. 1. Store Generation

Below we give some toy examples in order to illustrate our framework. First, we define a counter, Counter, which is incremented at each tick of a given signal S. Then, we define the operators Alarm and Warning mentioned in section 1. In addition, 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 equational declarative style (fib2). However, fib1 will be a timed operator whose value will change during the execution and fib2 will be an atemporal operator. Finally, we define a signal n-Bit-Register: [0..n-1]$\longrightarrow$bool which simulates a register of n bits. The elements of [0..n-1] represent the bit significances. n-Bit-Register is updated whenever a tick of a boolean signal S occurs and is incremented whenever S is true. We define also a temporized operator n-Carry: [0..n-1]$\longrightarrow$bool which stands for the set of carry flags and helps to compute the current value of n-Bit-Register.

Example 1.

Counter $==$ 0 **when** SREF $==$ 0
Counter $==$ at $(-1$ SREF$)$ Counter $+$ 1 **when** !S $==$ true

Alarm(device) $==$ true
  as soon as $\Box$ $[-10\,s$ .. now$]$ Broken(device) $==$ true
Warning(device) $==$ true
  as soon as $\Diamond$ $[-10\,s$ .. now$]$ Broken(device) $==$ true

fib1 $==$ 0 **when** SREF $==$ 0
fib1 $==$ 1 **when** SREF $==$ 1
fib1 $==$ at $(-1$ SREF$)$ fib1 $+$ at $(-2$ SREF$)$ fib1 **when** SREF$>$ 1 $==$ true

fib2(0) $==$ 0

$\mathsf{fib2(1)} == 1$
$\mathsf{fib2(x)} == \mathsf{fib2(x\text{-}1)} + \mathsf{fib2(x\text{-}2)} \Leftarrow x > 1 == \mathsf{true}$

n-Bit-Register(i) == false **when** $\textsc{Sref} == 0$
n-Bit-Register(0) == $\textsc{s}$ xor at $(-1\ \textsc{Sref})$ n-Bit-Register(0)
when !$\textsc{s}$ == true
n-Bit-Register(succ(i)) == n-Carry(succ(i)) xor
at $(-1\ \textsc{Sref})$ n-Bit-Register(succ(i)) **when** !$\textsc{s}$ == true

n-Carry(succ(i)) == at $(-1\ \textsc{Sref})$ n-Bit-Register(i) and not(n-Bit-Register(i))

3 Labeled Transition System

We present an easily comprehensible way of executing a program. We assume in the sequel that a timed program $\mathcal{P}$ is given with signature $< S\,,\, \mathcal{S}\,,\, \Omega >$. As depicted in Figure 1, a run of a program $\mathcal{P}$ consists of a stream of stores, $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_t, \ldots$. At any instant t, store $\mathcal{F}_t$ is a functional-logic program which may be used to solve goals as usual. If we consider the example 1, the solutions of the goal "Alarm(x) == true" asked at a given instant t are the devices, considered at the same instant t, which have been broken for the last 10 seconds. $\mathcal{F}_t$ consists of first-order conditional equations (without any reference to timed operators or atoms). We describe in this section how a functional-logic program, $\mathcal{F}_t$, is produced at every instant. Notice that our semantics combines labeled transition systems with goal solving.

Roughly speaking, the computation of $\mathcal{F}_t$ is done in three stages. First, we generate from $\mathcal{P}$, a new program $\mathcal{P}^\alpha$ which is equal to $\langle\mathcal{P}\rangle^t$ where the function $\langle . \rangle^t$ transforms a timed program $\mathcal{P}$ at the instant t, into another one without past toes nor past atoms. A past toe of the form $at\,(-n\,\textsc{s}\,)\,\omega$ is replaced with a new operator noted $\lfloor\omega\rceil^{t'}$ where t' denotes the absolute instant corresponding to $at\,(-n\,\textsc{s}\,)$ with respect to instant t. We can define a function *absolute_instant* such that $t' = absolute\_instant(\,n\,,\,\textsc{s}\,,\,t\,)$. When a past toe is of the general form $at\,(-n_1\,\textsc{s}_1\,)\,\ldots at\,(-n_m\,\textsc{s}_m\,)\,\omega$, we proceed likewise by iterating recursively the function *absolute_instant* to obtain the instant corresponding to $at\,(-n_1\,\textsc{s}_1\,)\,\ldots at\,(-n_m\,\textsc{s}_m\,)$. Whenever less than n ticks of $\textsc{s}$ occurred since the beginning of the execution, *absolute_instant* is not defined for $(\,n\,,\,\textsc{s}\,,\,t\,)$. Then, the timed conditional equation in which the past toe $at\,(-n\,\textsc{s}\,)\,\omega$ occurs is eliminated.

Example 2. Assume that the ticks of a signal $\textsc{s}$ happened at instants 2 and 4. The term $\omega\,+\,\mathrm{at}\,(-2\,\textsc{s}\,)\;\omega$ considered at the instant 5 will be transformed into $\langle\omega + \mathsf{at}\,(-2\,\textsc{s}\,)\;\omega\rangle^5 = \lfloor\omega\rceil^5 + \lfloor\omega\rceil^2$. The same term considered at 4 is not defined because there is only one tick of $\textsc{s}$ between 0 and 3. Thus, at 4, $\mathcal{P}^\alpha$ does not contain any transformed timed conditional equation in which $\omega\,+\,\mathrm{at}\,(-2\,\textsc{s}\,)\;\omega$ occurred.

The function $\langle . \rangle^t$ replaces the past atoms by conjunctions or disjunctions of atoms considered at different instants. Assume $B = \Diamond\ |_1\ -n_1\ \mathrm{S}_1\ ..\ -n_2\ \mathrm{S}_2\ |_2\ B'$. $\langle B\rangle^t = \bigvee_{t' \in |_1\, t_1\, ;\, t_2\, |_2} \langle B'\rangle^{t'}$ where $t_1 = \mathit{absolute\_instant}(n_1, \mathrm{S}_1, t)$ and $t_2 = \mathit{absolute\_instant}(n_2, \mathrm{S}_2, t)$. The definition is recursive because B' may still contain other past atoms or past toes. If *absolute_instant* is not defined for (n_1, S_1, t), then we consider $t_1 = 0$. In addition, if *absolute_instant* is not defined for (n_2, S_2, t), then B is replaced by an equation 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 is similar to past-diamond atoms: disjunctions become conjunctions and the equation FALSE becomes an equation noted TRUE which is always true.

Example 3. Consider the previous example. $\langle \Box[\text{ -2 S..-1 S }]\ \omega == \mathsf{true}\rangle^4$ is replaced by the conjunction $\langle \omega == \mathsf{true}\rangle^0 \wedge \langle \omega == \mathsf{true}\rangle^1 \wedge \langle \omega == \mathsf{true}\rangle^2$ which is equivalent to $\lfloor\omega\rceil^0 == \mathit{true} \wedge \lfloor\omega\rceil^1 == \mathit{true} \wedge \lfloor\omega\rceil^2 == \mathit{true}$. Likewise, $\langle \Box[\text{ -2 S..-1 S }]\ \omega == \mathsf{true}\rangle^5$ is replaced by the conjunction $\langle \omega == \mathsf{true}\rangle^2 \wedge \langle \omega == \mathsf{true}\rangle^3 \wedge \langle \omega == \mathsf{true}\rangle^4$ which is equivalent to $\lfloor\omega\rceil^2 == \mathit{true} \wedge \lfloor\omega\rceil^3 == \mathit{true} \wedge \lfloor\omega\rceil^4 == \mathit{true}$.

In the second stage, we compute a new program $\mathcal{P}^\beta$. $\mathcal{P}^\beta = \mathcal{P}^\alpha \cup \mathit{In} \cup D$ where In is the set of equations defining the incoming signals and D consists of conditional equations defining the new operators, of the form $\lfloor\omega\rceil^{t'}$, in $\mathcal{P}^\alpha$, where $0 \leq t' < t$. The axioms in D are copied from the stores previously computed. Notice that $\mathcal{P}^\beta$ may include operator definitions relative to the first instant, $\lfloor\omega\rceil^0$. For example, an operator, f, defined by f(x) == at $(-1\,\textsc{Sref})$ f(x) + g(x) where g is an input signal, needs at every instant, t, all the conditional equations which have been used to define $\lfloor \mathsf{f}\rceil^{t'}$ since the first instant, i.e., for $0 \leq t' < t$.

Finally, at a third stage, we compute a new program $\mathcal{P}^\gamma$ which is $\mathcal{P}^\beta$ in which all tails have been eliminated. $\mathcal{P}^\gamma$ consists of conditional equations e such that:

- e occurs without tail in $\mathcal{P}^\beta$,
- there exists a timed equation e **when** C in $\mathcal{P}^\beta$ where C holds in $\mathcal{P}^\beta$.
- or there exists a timed equation e **as soon as** C in $\mathcal{P}^\beta$ where C holds in $\mathcal{P}^\beta$ and C does not hold in $\mathcal{F}_{t-1}$. (If $t = 0$, $\mathcal{F}_{t-1}$ is not defined. Thus, we consider that C does not hold in $\mathcal{F}_{t-1}$).

Whenever all the tails of equations defining an operator $\lfloor\omega\rceil^t$ in $\mathcal{P}^\beta$ are false, the definition of $\lfloor\omega\rceil^t$ in $\mathcal{P}^\gamma$ is obtained from the one of $\lfloor\omega\rceil^{t-1}$ in $\mathcal{F}_{t-1}$, by replacing $\lfloor\omega\rceil^{t-1}$ by $\lfloor\omega\rceil^t$. If $\lfloor\omega\rceil^t$ represents the value of a signal S at t, we have to define the tick of S, too. In this case, if one of the tails of the equations defining an operator $\lfloor\omega\rceil^t$ in $\mathcal{P}^\beta$ is true, a tick of S occurs and the equation !S $== \mathit{true}$ is added to $\mathcal{P}^\gamma$. Otherwise, no tick of S occurs and the equation !S $== \mathit{false}$ is added to $\mathcal{P}^\gamma$. The store $\mathcal{F}_t$ is equal to $\mathcal{P}^\gamma$.

Example 4.
Consider the definition of the Counter in example 1. S is the only incoming signal. We give, below, samples of $\mathcal{P}^\beta$ and $\mathcal{P}^\gamma$ at different instants.

– At instant 0: Suppose no tick of S occurs at 0. $\mathcal{P}^\beta = \mathcal{P}^\alpha \cup In \cup D$. $\mathcal{P}^\alpha$ consists of the two following timed conditional equations:
  1. $\lfloor \mathsf{Counter} \rceil^0 == \mathsf{0}$ **when** $\lfloor \text{SREF} \rceil^0 == \mathsf{0}$
  2. $\lfloor \text{SREF} \rceil^0 == \mathsf{0}$

  Since no tick of S occurs at 0, the conditional equation which defines the tick of the incoming signal S is:

  3. $\lfloor !\text{S} \rceil^0 == \mathsf{false}$.

  Thus, In includes the conditional equation 3. Moreover, since 0 is the first instant, D is empty. Hence, $\mathcal{P}^\beta$ includes the timed conditional equations 1, 2 and 3. The only timed conditional equation with a tail is the first one. Its tail, $\lfloor \text{SREF} \rceil^0 == 0$, holds in $\mathcal{P}^\beta$. Therefore, $\mathcal{F}_0$ includes the timed conditional equations $1'$, 2 and 3, where:

  1'. $\lfloor \mathsf{Counter} \rceil^0 == \mathsf{0}$

– At instant 1: Suppose no tick of S occurs at 1. $\mathcal{P}^\beta$ includes the following timed conditional equations:
  1. $\lfloor \mathsf{Counter} \rceil^1 == \mathsf{0}$ **when** $\lfloor \text{SREF} \rceil^1 == \mathsf{0}$
  2. $\lfloor \mathsf{Counter} \rceil^1 == \lfloor \mathsf{Counter} \rceil^0 + 1$ **when** $\lfloor !\text{S} \rceil^1 == \mathsf{true}$
  3. $\lfloor \text{SREF} \rceil^1 == 1$
  4. $\lfloor !\text{S} \rceil^1 == \mathsf{false}$.
  5. $\lfloor \mathsf{Counter} \rceil^0 == \mathsf{0}$

  Since the tails of the timed conditional equations 1 and 2 are false and the definition of $\lfloor \mathsf{Counter} \rceil^0$ in $\mathcal{F}_0$ is $\lfloor \mathsf{Counter} \rceil^0 == \mathsf{0}$, the definition of $\lfloor \mathsf{Counter} \rceil^1$ in $\mathcal{P}^\gamma$ is:

  6. $\lfloor \mathsf{Counter} \rceil^1 == \mathsf{0}$

  Therefore, $\mathcal{F}_1$ includes the timed conditional equations 3, 4, 5 and 6.
– At instant 2: Suppose a tick of S occurs at 2. $\mathcal{P}^\beta$ includes the following timed conditional equations:
  1. $\lfloor \mathsf{Counter} \rceil^2 == \mathsf{0}$ **when** $\lfloor \text{SREF} \rceil^2 == \mathsf{0}$
  2. $\lfloor \mathsf{Counter} \rceil^2 == \lfloor \mathsf{Counter} \rceil^1 + 1$ **when** $\lfloor !\text{S} \rceil^2 == \mathsf{true}$
  3. $\lfloor \text{SREF} \rceil^2 == 2$
  4. $\lfloor !\text{S} \rceil^2 == \mathsf{true}$.
  5. $\lfloor \mathsf{Counter} \rceil^1 == \mathsf{0}$

  The tail of 1 is false and the tail of 2 is true because of the timed conditional equation 4. Hence, $\mathcal{F}_2$ includes the timed conditional equations $2'$, 3, 4 and 5 where:

  2'. $\lfloor \mathsf{Counter} \rceil^2 == \lfloor \mathsf{Counter} \rceil^1 + 1$

Now, we are ready to define a labeled transition system associated to a timed program.

Definition 1. *(labeled transition system)*
Let $\mathcal{P}$ be a timed program. The associated labeled transition system *is a tuple* $< Q\,,\, L\,,\, \longrightarrow\,,\, Init >$ *such that :*

- *Q is the set of* states. *A state is a pair ($\mathcal{H}$, $\mathcal{F}$) where $\mathcal{H}$, called history, is a list of stores and $\mathcal{F}$ is a store. The stores define at different instants operators occurring in $\mathcal{P}$.*
- *L is the set of* labels *where a label is a set of equations defining the incoming signals.*
- *$\longrightarrow \subseteq Q \times L \times Q$ is the transition relation defined as follows. We write $\mathcal{C}_1 \xrightarrow{In} \mathcal{C}_2$ for a triple $< \mathcal{C}_1\,,\, In\,,\, \mathcal{C}_2 >$. A transition $< \mathcal{H}\,,\, \mathcal{F}_t > \xrightarrow{In} < [\mathcal{F}_t \mid \mathcal{H}]\,,\, \mathcal{F}_{t+1} >$ occurs iff $\mathcal{F}_{t+1}$ is the store computed at the instant $t+1$ starting from the program $\mathcal{P}$, the history $[\mathcal{F}_t \mid \mathcal{H}]$ and the input signals given by In.*
- *$Init$ is a function $L \longrightarrow Q$ of* initialization *such that if In_0 represents the definitions of the incoming signals at 0 then $Init(In_0) = < []\,,\, \mathcal{F}_0 >$.*

4 Operational semantics

The computation of the transition system described above is not feasible in general. So, the aim of this section is to propose a tractable operational semantics for our framework. We start by an overview of the main drawbacks before presenting the changes which make the operational semantics tractable.

The problem of the history: Since the history is one of the components of states, the space memory of a state increases over time.

The problem of the number of formulae in a store: Consider the specification of Counter given in Example 1. Suppose the ticks of S happened at 2, 6, 8. At 11, obviously Counter == 3. However, the whole definition for Counter according to the first semantics (transition system) is:

$\lfloor \mathsf{Counter} \rceil^{11} == \lfloor \mathsf{Counter} \rceil^{7} + 1$
$\lfloor \mathsf{Counter} \rceil^{7} == \lfloor \mathsf{Counter} \rceil^{5} + 1$
$\lfloor \mathsf{Counter} \rceil^{5} == \lfloor \mathsf{Counter} \rceil^{1} + 1$
$\lfloor \mathsf{Counter} \rceil^{1} == 0$

In fact, each time a tick of S occurs, the store space increases by (at least) one equation. Since the stores are parts of the states, this is a new reason for which the space memory of a state can increase over time.

The problem of the size of the formulae: If we consider now the past atom $B = \Diamond\ [\ -1\,\mathrm{S} ..\ -1\,\mathrm{SREF}\ ]\ B'$ then $\langle B \rangle^{11} = \langle B' \rangle^{8} \vee \langle B' \rangle^{9} \vee \langle B' \rangle^{10}$. As far as no new tick of S occurs, the length of the disjunction will increase by one at each instant. Hence, the length of the timed conditional equation containing the disjunction will grow at the same rate.

The problem of the tail deletion: Consider the following sample of a program:

$A ==$ true **when** $B ==$ true
$B ==$ true **when** $A ==$ true

The computation of the definition of $\lfloor A \rceil^t$ requires the computation of the definition of $\lfloor B \rceil^t$ and conversely. Hence, the elimination of tails is not possible in this case. The problem of causality loop exists in most of synchronous programming languages [9]. In addition, for real-time applications, tails should be decidable and computable in a sensible amount of time.

All these problems lead to propose a new enhanced semantics which is sound w.r.t. the behavior of the previous transition system. The first change consists to replace the history by a finite set of bounded memories. They enable, under some restrictions, the computation of the definitions of the past toes and the past atoms within a bounded duration. This involves a change of the first stage. $\mathcal{P}^\alpha$ is calculated statically from $\mathcal{P}$ by replacing the past toes, respectively the past atoms by new atemporal operators, respectively equations involving new atemporal operators. $\mathcal{P}^\beta$ will contain the formulae of $\mathcal{P}^\alpha$ completed with the calculated definitions of these new operators. The third stage of the operational semantics will be the same as for the first one. It produces a program $\mathcal{P}^\gamma$ where tails have been removed. We also add a fourth stage computing a program $\mathcal{P}^\delta$ which consists in normalizing the definition of some operators, say array operators, so that, at the current instant t, the normalized definitions of these operators are bounded. For the operators which are not array operators, we propose a restriction which ensures directly that their definition is bounded.

The first stage of the store computation: The first stage is performed by a function $\langle . \rangle$. It consists like $\langle . \rangle^t$ in just replacing the past toes and the past atoms by atemporal expressions. In contrast to $\langle . \rangle^t$, $\langle . \rangle$ replaces the past toes $\phi = at\,(n_1\ \mathrm{s}_1\,)\ \ldots at\,(n_m\ \mathrm{s}_m\,)\,\omega$ by new operators noted $\lfloor at\,(n_1\ \mathrm{s}_1\,)\ \langle at\,(n_2\ \mathrm{s}_2\,)\ \ldots at\,(n_m\ \mathrm{s}_m\,)\ \ \omega \rangle \rceil$ of the same profile as ω and replaces the past atoms B by equations $\lfloor B \rceil == \mathit{true}$ where $\lfloor B \rceil$ is a new boolean operator. The resulting program is a timed program noted $\mathcal{P}^\alpha$ whose timed conditional equations are conditional equations with a tail. In the following, we call *new operators*, the operators of the form $\lfloor \phi \rceil$ and $\lfloor B \rceil$ in $\mathcal{P}^\alpha$.

The second stage of the store computation: The memories which replace the history are not the same according to the nature of the new operators with which they are associated. Thus, in the following, we separate the past atom memories from the past toe memories. However, all the memories verify an invariant at each instant which enables the calculus of the new operator definitions in a bounded duration. In addition, all the memories are space bounded and can be updated in a bounded duration. We will now present the past toe memories.

Definition 2. *(Past toe memory)*
Let $\phi = at\,(-n\ \mathrm{s}\,)\,\phi'$ *be a past toe and* t *be an instant. The* past toe memory $\mathcal{M}em_{\langle \phi \rangle}$ *at* t *for the new operator* $\lfloor at\,(-n\ \mathrm{s}\,)\,\langle \phi' \rangle \rceil$ *is a queue (FIFO) of* n *sets of conditional equations* d_i *defining* $\langle \phi' \rangle$*. This queue is noted:*

$$\left[\, d_n\ \ldots\ d_2\ d_1 \,\right]$$

Definition 3. *(Past toe memory invariant)*

Let $\phi = at\,(-n\,\text{s})\ \phi'$ be a past toe. Assume $\mathcal{M}em_{\langle\phi\rangle} = [d_n\ \ldots\ d_2\ d_1]$.

The invariant *for $\mathcal{M}em_{\langle\phi\rangle}$ is the following assertion: "For all $i \in \{\ 1\,;\ldots;n\ \}$, the definition of $\lfloor at\,(-i\,\text{s})\ \langle\phi'\rangle\rceil$ is obtained from d_i by replacing $\langle\phi'\rangle$ by $\lfloor at\,(-i\,\text{s})\ \langle\phi'\rangle\rceil$".*

So, from a past toe memory $\mathcal{M}em_{\langle\phi\rangle}$ which verifies the invariant, the definition of the new operator $\langle\phi\rangle$ is easy to calculate.

Example 5.

We suppose that the following definitions are true:

	definition of c	definition of !s
$t=0$	$\{\mathsf{c} == 6\}$	$\{!\text{s} == \mathsf{true}\}$
$t=1$	$\{\mathsf{c} == 3\}$	$\{!\text{s} == \mathsf{true}\}$
$t=2$	$\{\mathsf{c} == 6\}$	$\{!\text{s} == \mathsf{false}\}$
$t=3$	$\{\mathsf{c} == 9\}$	$\{!\text{s} == \mathsf{true}\}$
$t=4$	$\{\mathsf{c} == 8\}$	$\{!\text{s} == \mathsf{true}\}$

Thus, at the instant 5, if $\mathcal{M}em_{\lfloor \mathsf{at}\,(-3\,\text{s})\ \mathsf{c}\rceil}$ verifies the invariant then

$\mathcal{M}em_{\lfloor \mathsf{at}\,(-3\,\text{s})\ \mathsf{c}\rceil} = [\ \{\mathsf{c} == 3\}\ \{\mathsf{c} == 9\}\ \{\mathsf{c} == 8\}\ ]$.

And so, the definition of $\lfloor \mathsf{at}\,(-3\,\text{s})\ \mathsf{c}\rceil = \{\lfloor \mathsf{at}\,(-3\,\text{s})\ \mathsf{c}\rceil == 3\}$ at 5.

$(\mathsf{at}\ 0)\dfrac{}{[\,\emptyset\ \ldots\ \emptyset\ \emptyset\,]}$	if $t = 0$
$(\mathsf{at}\ 1)\dfrac{[\,d_n\ \ldots\ d_2\ d_1\,]}{[\,d_n\ \ldots\ d_2\ d_1\,]}$	if $t > 0$ and $\mathcal{F}_{t-1}\ \vdash !\text{s} == \mathit{false}$
$(\mathsf{at}\ 2)\dfrac{[\,d_n\ \ldots\ d_2\ d_1\,]}{[\,d_{n-1}\ \ldots\ d_1\ D\,]}$	if $t > 0$ and $\mathcal{F}_{t-1}\ \vdash !\text{s} == \mathit{true}$

where D stands for the definition of $\langle\phi'\rangle$ at the previous instant $t-1$ and $[\,d_n\ \ldots\ d_2\,d_1\,]$ is the memory of the new operator $\lfloor at\,(-n\,\text{s})\,\langle\phi'\rangle\rceil$ at $t-1$.

Fig. 2. Past toe memory rules for the operator $\phi = at\,(-n\,\text{s})\,\phi'$

The rules of Figure 2 show how past atom memories are initialized and updated. The rule (at 0) initializes the memory and the others update it.

Example 6.

From the previous example, we suppose at 5 that the definition of c is $\{\mathsf{c} == 6\}$ and the definition of !s is $\{!\text{s} == \mathsf{true}\}$. Thus at 6, according to the rule (at 2), $\mathcal{M}em_{\lfloor \mathsf{at}\,(-3\,\text{s})\ \mathsf{c}\rceil}$ becomes $[\ \{\mathsf{c} == 9\}\ \{\mathsf{c} == 8\}\ \{\mathsf{c} == 6\}\ ]$. And so, at 6, the definition of $\lfloor \mathsf{at}\,(-3\,\text{s})\ \mathsf{c}\rceil$ is $\{\lfloor \mathsf{at}\,(-3\,\text{s})\ \mathsf{c}\rceil == 9\}$.

Due to lack of space, we omit the inference rules for past atoms. The computation of the new operator definition generated from past atoms is performed under the following requirement:

Requirement 1 *Every past atom of the form* $\Box \mid_1 n_1 \, \mathrm{S}_1 .. n_2 \, \mathrm{S}_2 \mid_2 B'$ *or* $\Diamond \mid_1 n_1 \, \mathrm{S}_1 .. n_2 \, \mathrm{S}_2 \mid_2 B'$ *is closed*[1] *and is such that* B' *can be decided in a sensible computational duration at each instant.*

For a new operator $\lfloor \Box \mid_1 n_1 \, \mathrm{S}_1 .. n_2 \, \mathrm{S}_2 \mid_2 B' \rceil$ or $\lfloor \Diamond \mid_1 n_1 \, \mathrm{S}_1 .. n_2 \, \mathrm{S}_2 \mid_2 B' \rceil$, the past atom memory requires $\mathcal{O}(n_1 + n_2)$ space. At the end of the second stage of the operational semantics, $\mathcal{P}^\beta$ is $\mathcal{P}^\alpha$ to which the set In of the current definitions of the incoming signals and the definitions of all the past atoms have been added.

The third stage of the store computation: The third stage does not change. It is the same as the one in the first semantics and computes a store noted $\mathcal{P}^\gamma$ from $\mathcal{P}^\beta$ by eliminating the tails. However, we assume in the sequel that the programs we consider are stratified so that the evaluation of tails is decidable and tractable, avoiding the problem of causality loop.

Requirement 2 *The tails of each timed conditional equation of a given stratum are expressed through the timed conditional equations of the lower strata.*

Stratification of programs has already been considered in logic programming languages [12].

The fourth stage of the store computation: The aim of fourth stage is to prevent from overflows due to unbound number of conditional equations of the stores during the execution. It consists in computing for all operators ω all the current values v of ω in the store $\mathcal{P}^\gamma$ and replacing the definition of ω by atemporal equations $\omega == v$. However, ω may be a function with parameters. So, we define a class of operators for which the transformation is relevant: ω is an *array operator* if its domain is finite and if the evaluation of ω for each possible value of its parameters is tractable. In this way, the number of the possible values of the parameters is bounded. Thus, the number of combinations for the conditional equations defining ω is bounded too. We write $\mathcal{P}^\delta$ the result of the fourth stage.

For the timed operators which are not array operators, the transformation cannot be applied because it would be too long or it would not terminate. The following restriction ensures that the definition of such operators will not get bigger and bigger during the execution.

Requirement 3 *A timed operator* ω *which is not an array operator cannot be defined recursively from the past.*

So, at the end of the fourth stage, $\mathcal{P}^\delta$ contains only definitions whose size is bounded.

Example 7.
Consider a 3-bit register 3-Bit-Register of the Example 1. If, at 1, a tick of S occurs and S is *true*, the definition of 3-Bit-Register in $\mathcal{P}^\gamma$ is:

[1] i.e. it contains no free variable

3-Bit-Register(0) == true
3-Bit-Register(1) == false
3-Bit-Register(2) == false

The operational semantics: To recapitulate the operational semantics, the history has been replaced by a set of memories whose role is to enable the calculus of the new operator definitions in the current store. In a formal way, a *state* has to be computed at each instant where a state is a pair $<\mathcal{M}em\,,\,\mathcal{F}>$, $\mathcal{M}em$ is a set of memories and $\mathcal{F}$ is a store. The memories are initialized and updated by the system of inference rules. Since memory calculus does not depend on the current value of the timed operators, for an implementation, the memory initialization can be performed statically and the memory updating can be performed before the inputs occur. The store generation consists in calculating $\mathcal{P}^{\delta}$ for the current instant.

Definition 4. *(Soundness w.r.t. the behavior of the transition system)*
We say that the operational semantics is sound w.r.t. the behavior of the transition system *if, at each instant t, for every timed operator ω, $\lfloor\omega\rceil^t$ defined by the labeled transition system and ω defined by the operational semantics stand for the same mathematical function.*

Theorem 1. *For the programs which verify the requirements, the operational semantics is sound w.r.t. the behavior of the transition system.*

5 Conclusion and Related work

We gave the broad outlines of a new kernel framework allowing conservative extensions of declarative programs with time. We illustrated our approach over functional-logic programs. However, our approach can be generalized to other declarative programs such as logic programming by defining a notion of timed predicate expressions as in e.g. $at\,(-n\,\mathrm{s}\,)$ *Pred*. We proposed an efficient operational semantics so that we can tackle real-world applications. Notice that, most real-time synchronous languages transform the programs into finite-state automata. This 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` [14] 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 process algebra is orthogonal to the store and the store is atemporal and does not contain any past reference. So, the information transmission from an instant to another is performed explicitly by the combinator next. In our case, the mechanisms of transmission is hidden and thus more declarative. Furthermore, we have shown that these mechanisms could be rather complicated in the case of past atoms and they could manipulate definitions of operators which is not possible in `tcc`. Another problem of `tcc` inherited from `ccp` is the negative information detection. Each instant, `tcc` may only detect an absence of information at the end of the store modification computation and so the reaction is put back one instant later. A solution was presented in [15]. The new paradigm is called `Timed Default cc` and enables to detect an absence of information and to react instantaneously. In compensation, the programs need to be tested deterministic statically. This way was dismissed in [14] ("paradox-free" property). Our store is well-defined stratified functional logic program 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`.

Another class of programming languages is the temporal logic programming languages based on temporal logics. Interesting overviews are presented in [13]. The meaning of "temporal" for languages like `Templog` [1] and `Temporal Prolog` of Gabbay [7] is not the same as one in the synchronous world. The time is indeed represented in a constraint system and does not constraint temporally the execution of the program. The causality of the condition in timed conditional equations is not preserved in these extensions of the logic programming, whereas it is in our paradigm. Our temporal operators always $\Box$ and eventually $\Diamond$ have parameters to specify which interval in the past is concerned. The calculus of the past term is performed by sub-intervals which is advocate by Brzoska in [2] to reduce its duration. Furthermore, in our case, this latter is performed on-line. `Chronolog` is a data-flow temporal logic language [16] designed to do efficient computation. This language only manages one clock. Since the need for a multi-granular time is very important in temporal programming, `Chronolog(MC)` [11] have been developed. However this language is not synchronous at all.

The synchronous languages are modular deterministic languages specially designed for reactive programming [9]. They manipulate scalar values sampled over time. These values constitute a stream called signal whose elements are associated implicitly with instants with a null duration. Our notion of signal subsumes the previous one thanks to the possibility of expressing functions with possibly infinite definition domain and with a changing definition. The most representative declarative synchronous languages are `Lustre` [5] and `Signal` [8]. The primitives of the latter languages may be grouped into three categories:

- *The classical arithmetic or logical operators* are static monochronous operators extended to sequences. These operators cannot be defined in the language. Thus, they are either predefined or defined in a host language. This is not practical neither from the point of view of the conception, nor to ensure properties such as determinism or ending in a sensible duration. In our paradigm, data type as well as functions are user-defined.
- *The delay operators* (pre in `Lustre` and \$ in `Signal`) are dynamical monochronous operators whose role is to access during the execution to past values of signals. The operator at subsumes the previous operators.
- *The polychronous operators* act on the ticks of the signals. They perform, for instance, intersections on them or unions with a deterministic merge of the

corresponding values. In our formalim, any operation over signals is possible since it can be defined by the user.

References

1. M. Abadi and Z. Manna. Temporal logic programming. *J. of Symbolic Computation*, 8(3):277–295, September 1989.
2. C. Brzoska. Programming in metric temporal logic. *Theoretical Computer Science*, 202:55–125, 1998.
3. Curry: An integrated functional logic language. available at `http://www.informatik.uni-kiel.de/~mh/curry/report.html`.
4. N. Dershowitz and D. Plaisted. Logic programming *cum* applicative programming. In *IEEE Symp. on Logic Programming*, pages 54–66, Boston, MA, 1985.
5. C. Dumas-Canovas and P. Caspi. A pvs proof obligation generator for lustre programs. In *2nd Int. Conf. on Logic for Programming and Reasonning*, 2000. to appear in LNAI.
6. R. Echahed and W. Serwe. Combining mobile processes and declarative programming. In *Proc. of the First Int. Conf. on Computational Logic*, volume 1861 of *LNAI*, July 2000.
7. D. Gabbay. *Modal and Temporal Logic Programming*, chapter 6, pages 197–237. Academic Press, 1987.
8. 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.
9. N. Halbwachs. Synchronous programming of reactive systems. In *Computer Aided Verification (CAV'98)*, volume 1427 of *LNAI*, pages 1–16, 1998.
10. M. Hanus. Distributed programming in a multi-paradigm declarative language. In *Proc. of PPDP'99*, volume 1702 of *LNCS*, pages 188–205, 1999.
11. 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.
12. J. W. Lloyd. *Foundations of Logic Programming*. Symbolic Computation Artificial Intelligence series. Springer-Verlag, 1987.
13. 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*, volume 827 of *LNAI*, pages 445–479, July 1994.
14. 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.
15. V. A. Saraswat, R. Jagadeesan, and V. Gupta. Timed default concurrent constraint programming. *Journal of Symbolic Computation*, 22(5 – 6):475 – 520, November – December 1996.
16. 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.