FROM A SYNCHRONOUS DECLARATIVE LANGUAGE TO A TEMPORAL LOGIC DEALING WITH MULTIFORM TIME

(Extended Abstract)

Daniel PILAUD, Nicolas HALBWACHS
Laboratoire de Génie Informatique - Institut IMAG
B.P. 53 , 38041 Grenoble Cedex - France

1. Introduction

Temporal logics were first used in computer science to describe the behaviour of concurrent systems [7,11,14]. More recently, some efforts [10,12,16] have been devoted to the design of temporal logics well-suited to the description of the real-time aspects of the behaviour of computer systems. These works are generally based on linear time logics, and consider the time grain of such logics as a physical time unit.

At the same time, several authors [2,4,5,8,9] have argued that *synchronous reactive systems* form a suitable framework for modelling and programming real-time systems. The basic idea is that, from a suitable level of abstraction, the only notion which is necessary for describing a real-time system is the notion of instantaneous reaction to external events. In this framework, physical time is considered as an external event, without any privileged role, and conversely, any external event may be viewed as defining a time-scale. This is the *multiform time* point of view. It encourages modularity, versatility and abstraction, since it allows a real-time process to be defined without explicit reference to physical time, and a given process description to be instantiated with different time scales.

For instance, in an actual railway regulation system, the on-board speed regulation device perceives several events: signals from the central system, beacons along the track, wheel revolutions, and milliseconds from its local clock. Now, one can consider all these events in a symmetric way. So, time can be counted in wheel revolutions as well as in milliseconds, and a "watch-dog" may be set on any event.

In this paper, we propose an adaptation of temporal logic to multiform time. For this purpose, we use temporal logic to express the semantics of the synchronous declarative language LUSTRE [3,5]. The underlying model of LUSTRE is close to that of temporal logic, since it is based on sequences: as in LUCID [1], any variable in LUSTRE is a sequence of values, and programs operate globally over sequences. Moreover, these sequences are synchronously interpreted: any variable has a *clock* associated with it, and takes the n-th value of its sequence at the n-th "tick" of its clock.

After a brief overview of LUSTRE and its formal semantics, we shall see that Pnueli's linear temporal logic is powerful enough to describe it. In this description, we shall derive new modal operators, which express the multiform time point of view. Then, the properties of these operators will be studied.

2. Overview of the language LUSTRE

2.1 Variables, Clocks, Equations, Data operators

As indicated above, any variable or expression in LUSTRE denotes a sequence of values. Moreover, each variable has a clock, and is intended to take the n-th value of its sequence at the n-th tick of its clock. A program has a cyclic behaviour, which defines its *basic clock* : a variable which is on the basic clock takes its n-th value at the n-th cycle of the program. Other, slower, clocks may be defined by means of boolean variables: any boolean variable C may be used as a clock, which is the sequence of cycles when C is true.

For instance, consider a boolean variable C on the basic clock, and a boolean variable C′ on the clock C. The following table shows the different time scales they define:

basic time	0	1	2	3	4	5	6	7
C	true	false	true	true	false	true	false	true
time on C	0		1	2		3		4
C′	false		true	false		true		true
time on C′			0			1		2

Variables are defined by means of equations: if X is a variable and E is an expression, the equation "X=E" defines X to be the sequence $(x_0=e_0, x_1=e_1, ..., x_n=e_n, ...)$ where $(e_0, e_1, ..., e_n, ...)$ is the sequence of values of the expression E. Moreover, the equation states that X has the same clock as E.

Expressions are built up from variables, constants (considered to be infinite constant sequences on the basic clock) and operators. Usual operators on values (arithmetic, boolean, conditional operators) are extended to pointwisely operate over sequences, and are hereafter referred to as *data operators* . For instance, the expression

if X>Y then X-Y else Y-X

denotes the sequence whose n-th term is the absolute difference of the n-th values of X and Y. The operands of a data-operator must be on the same clock, which is also the clock of the result.

2.2 Sequence Operators

In addition to the data operators, LUSTRE contains only four non-standard operators, called sequence operators, which actually manipulate sequences.

To keep track of the value of an expression from one cycle to the next, there is a memory or delay operator called "pre" (previous). If $X=(x_0,x_1,...,x_n,...)$ then

$$pre(X) = (nil,x_0,x_1,...,x_{n-1},...)$$

where nil is an *undefined* value, akin to the value of an uninitialized variable in imperative languages. The clock of pre(X) is the clock of X.

To initialize variables, the -> (followed by) operator is introduced. If $X=(x_0,x_1,...,x_n,...)$ and $Y=(y_0,y_1,...,y_n,...)$ are two variables (or expressions) of the same type and the same clock, then

$$X->Y = (x_0,y_1,y_2,...,y_n,...)$$

i.e. X->Y is equal to Y except at the first instant.

The last two operators are used to define expressions with different clocks:

If E is an expression, B is a boolean expression, and if E and B are on the same clock, then "E when B" is an expression on the clock defined by B, and whose sequence of values is extracted from the sequence of E by taking only those values which occur when B is true.

If E is an expression on a clock CK different from the basic clock, then "current(E)" is an expression on the same clock as CK, whose value at each cycle of this clock is the value taken by E at the last cycle when CK was true.

The following table shows the effect of these operators.

B = (	false	true	false	false	true	true	false	true	...)
X = (	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	...)
Y = X whenB = (		x_1			x_4	x_5		x_7	...)
Z = current(Y) = (	nil	x_1	x_1	x_1	x_4	x_5	x_5	x_7	...)

The rules for statically computing the clock of any expression are presented in [5]. In the following, we shall always assume that the rules of clock consistency are satisfied, and that the clock of any expression is known.

2.3 Formal Semantics [5]

The semantics of a LUSTRE program is a sequence $(\sigma_0, \sigma_1, ..., \sigma_n, ...)$ of memories, where a memory is a function from the set ID of identifiers to the set VAL of values, and where σ_n represents the state of the variables at the n-th cycle of the basic clock of the program. The set VAL contains both *nil* (the undefined value) and $\perp$, which will be used to represent the absence of value: a variable has no value when it must not be computed (its clock is false or doesn't have to be computed). Such a sequence of memories will be called a *history*, and we shall note Hist(VAL) the set of histories on the set of values VAL.

The semantics will be described by means of structural inference rules [13]. Without loss of generality, let us consider the following simplified syntax:

- an equation is assumed to involve at most one operator;
- the "pre" and "current" operators may be given an additional parameter, that records the necessary past value of their operand;
- in the case of "current", the clock of its operand (which is assumed to be known, as a byproduct of the clock analysis) is also assumed to be a parameter.

With this syntax, at a given instant, we only need to know the memory in order to compute the value of an expression . This value is assumed to be $\perp$ if the clock of the expression is not true. An equation "X=E" will be said to be *compatible* with a memory σ if and only if $\sigma(X)$ is the result of the expression E, evaluated in σ.

The rules of Figure 1 define the predicate:

$$[eq] -\sigma\rightarrow [eq']$$

which means:

"The equation eq is compatible with the memory σ and will be later on evaluated as eq′ "

Let us comment some of these rules:

System of equations: A system of equations is compatible with a memory σ iff each of its equations is compatible with σ.

Data operator: From the clock consistency, the operands of a data operator "op" are on the same clock. So, either none or all of them have a value. In the latter case, the resulting value is the result of the corresponding standard operator, noted "op" (any data operator, except the conditional, is strict with respect to "nil").

"Previous" operator: There are four rules for defining the semantics of "pre(Y)". The first one applies when the clock of Y is true for the first time: then the value of pre(Y) is nil, and the value of Y is stored. The second rule applies the subsequent times the clock of Y is true: the value of pre(Y) is the stored value, and the value of Y is stored. The last two rules concern the cases when the clock of Y is not true. The same four cases are considered for the operator "current".

System of equations

$$\frac{eq_1 \xrightarrow{\sigma} eq_1' \ , \ eq_2 \xrightarrow{\sigma} eq_2'}{[eq_1;eq_2] \xrightarrow{\sigma} [eq_1'; eq_2']}$$

Simple equation

$$\frac{\sigma(X) = \sigma(Y)}{[X=Y] \xrightarrow{\sigma} [X=Y]}$$

Data operator

$$\frac{\sigma(Y)\neq\perp \ , \ \sigma(Z)\neq\perp \ , \ \sigma(X) = \sigma(Y) \ \underline{op} \ \sigma(Z)}{[X=Y \ op \ Z] \xrightarrow{\sigma} [X=Y \ op \ Z]} \qquad \frac{\sigma(Y)= \sigma(Z)= \sigma(X)= \perp}{[X=Y \ op \ Z] \xrightarrow{\sigma} [X=Y \ op \ Z]}$$

"Previous" operator

$$\frac{\sigma(Y)=k\neq \perp \ , \ \sigma(X)=nil}{[X=pre(Y)] \xrightarrow{\sigma} [X=pre(Y,k)]} \qquad \frac{\sigma(Y)=k'\neq \perp \ , \ \sigma(X)=k}{[X=pre(Y,k)] \xrightarrow{\sigma} [X=pre(Y,k')]}$$

$$\frac{\sigma(Y)= \sigma(X)= \perp}{[X=pre(Y)] \xrightarrow{\sigma} [X=pre(Y)]} \qquad \frac{\sigma(Y)= \sigma(X)= \perp}{[X=pre(Y,k)] \xrightarrow{\sigma} [X=pre(Y,k)]}$$

"Followed-by" operator

$$\frac{\sigma(Y)= \sigma(X)\neq \perp}{[X=Y\text{->}Z] \xrightarrow{\sigma} [X=Z]} \qquad \frac{\sigma(Y)= \sigma(X)= \perp}{[X=Y\text{->}Z] \xrightarrow{\sigma} [X=Y\text{->}Z]}$$

"When" operator

$$\frac{\sigma(C)=true \ , \ \sigma(Y)= \sigma(X)\neq \perp}{[X=(Y \ when \ C)] \xrightarrow{\sigma} [X=(Y \ when \ C)]} \qquad \frac{\sigma(C)=false \ , \ \sigma(Y)\neq \perp \ , \ \sigma(X)= \perp}{[X=(Y \ when \ C)] \xrightarrow{\sigma} [X=(Y \ when \ C)]}$$

$$\frac{\sigma(C)= \sigma(Y)= \sigma(X)= \perp}{[X=(Y \ when \ C)] \xrightarrow{\sigma} [X=(Y \ when \ C)]}$$

"Current" operator , where C is the clock of Y

$$\frac{\sigma(C) = true \ , \ \sigma(Y)= \sigma(X)=k\neq \perp}{[X=current(Y,C)] \xrightarrow{\sigma} [X=current(Y,k,C)]} \qquad \frac{\sigma(C) = true \ , \ \sigma(Y)= \sigma(X)=k'\neq \perp}{[X=current(Y,k,C)] \xrightarrow{\sigma} [X=current(Y,k',C)]}$$

$$\frac{\sigma(C) = false \ , \ \sigma(X)=nil \ , \ \sigma(Y)= \perp}{[X=current(Y,C)] \xrightarrow{\sigma} [X=current(Y,C)]} \qquad \frac{\sigma(C) = false \ , \ \sigma(X)=k \ , \ \sigma(Y)= \perp}{[X=current(Y,k,C)] \xrightarrow{\sigma} [X=current(Y,k,C)]}$$

Figure 1

Now a LUSTRE program P is compatible with an history (σ_0, σ_1 ,..., σ_n ,...) if and only if its system of equations is compatible with σ_0, and must be later on considered as a new program P′, which is compatible with (σ_1 ,..., σ_n ,...) :

$$\frac{[P] -\sigma_0\rightarrow [P'] \ , \ \Sigma \vdash [P']}{\sigma_0.\Sigma \vdash [P]}$$

2.4 Example of LUSTRE program: An axle detector in a railway system

A part of a railway system must perform the following function: A track is divided into districts. At the boundary between two districts, there are two pedals which overlap each other as shown by Figure 2. The device knows the states of the pedals by means of two boolean variables P1 and P2 (Pi is true whenever a wheel is on Pi). It must compute a boolean S which is true when and only when an axle goes from the backward district to the forward district. More precisely, S is set when the axle goes out and reset just after. There can be only one axle in the zone Z. An axle may turn back within the zone Z and even oscillate on or about the zone Z.

Let us write the program which computes S. We introduce two boolean variables "last_in_b" and "out_f": the former is true whenever the last axle which entered the zone Z came from the backward district, and the latter is true whenever an axle leaves the zone Z to the forward district.

Now, we have

```
S = false -> out_f and lastin_b;
```

An axle enters the zone Z from the backward (resp. forward) district whenever the input P1 (resp. P2) has a rising edge when P2 (resp. P1) is false:

```
last_in _b= if P1 and not pre(P1) and not P2 then true
            else if P2 and not pre(P2) and not P1 then false
            else pre(last_in_b) ;
```

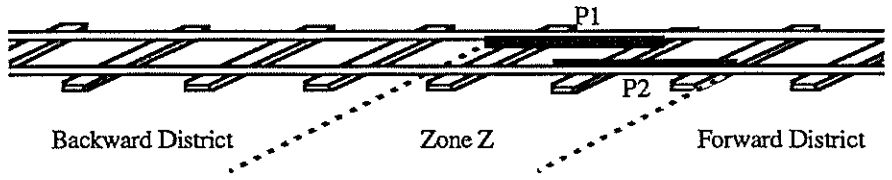

Figure 2

An axle leaves the zone Z to the forward district whenever the input P2 has a falling edge when P1 is false:

```
out_f = not P2 and pre(P2) and not P1;
```

So the whole program is as follows:

```
        S =   false -> out_f and lastin_b;
last_in_b =   if P1 and not pre(P1) and not P2 then true
              else if P2 and not pre(P2) and not P1 then false
              else pre(last_in_b) ;
    out_f =   not P2 and pre(P2) and not P1;
```

3. Describing LUSTRE in Temporal Logic

3.1 Linear temporal logic

We use the logic of [7]: A formula is built from proposition symbols, usual boolean operators ($\neg,\wedge,\vee,\supset,\equiv$), $\Box$(*always* , unary), $\Diamond$ (*eventually* , unary), O (*next* , unary) and U (*until* , binary). A model is a sequence $S=(s_0,s_1, \ldots, s_n, \ldots)$ of states, each state giving values to the proposition symbols. The semantics of a formula p is a function |p| from the set of models to {true, false}. The semantics of modal operators are as follows:

- $|\Box p|(S)$=true iff $\forall n\geq 0$, $|p|(S^{+n})$ = true , where S^{+n} denotes the n-th suffix $(s_n,s_{n+1}, \ldots)$ of S;
- $|\Diamond p|(S)$ = true iff $\exists n\geq 0$ such that $|p|(S^{+n})$ = true ;
- $|Op|(S)$ = true iff $|p|(S^{+1})$ = true ;
- $|pUq|(S)$ = true iff $\exists n\geq 0$ such that $|q|(S^{+n})$ = true, and $\forall m$, $0\leq m<n$, $|p|(S^{+m})$ = true

This logic is axiomatized by means of 8 axioms:

A0. $\vdash \Diamond p \equiv \neg\Box(\neg p)$ A1. $\vdash \Box(p\supset q)\supset(\Box p\supset \Box q)$
A2. $\vdash O(\neg p) \equiv \neg Op$ A3. $\vdash O(p\supset q)\equiv(Op\supset Oq)$
A4. $\vdash \Box p \supset(p\wedge O\Box p)$ A5. $\vdash \Box(p\supset Op)\supset(p\supset\Box p)$
A6. $\vdash pUq \supset\Diamond q$ A7. $\vdash pUq \equiv q\vee(p\wedge O(pUq))$

and 3 rules of inference

R1. If f is a tautology, then $\vdash f$.
R2. (Modus Ponens) If $\vdash f_1 \supset f_2$ and $\vdash f_1$ then $\vdash f_2$.
R3. (Generalization) If $\vdash f$ then $\vdash \Box f$.

3.2 Multiform time operators

The intuitive real-time interpretation of linear temporal logic consists of assimilating logical instants with physical instants, belonging to some metric time-scale: for instance, the formula $\bigcirc p$ would be considered as meaning "p will be true at the next second". In order to handle multiform time, we must be able to define several independent time scales, so as to express in the same way, for instance, the property "p will be true at the next wheel revolution". These time scales will always be sequences of instants which are subsequences of the logical time. Intuitively, an event such as "second" or "wheel revolution" will be represented by a formula which is true whenever the event occurs. Now, with each time-scale defined in this way, we need to associate specific modal operators.

For any formula f, we define five abbreviations $\Box_f$ (*always when f*), $\Diamond_f$ (*eventually when f*), F_f (*at first f*), $\bigcirc_f$ (*at next f*), U_f (*whenever f until*),as follows:

$\Box_f p \equiv \Box(f \supset p)$

$\Diamond_f p \equiv \Diamond(f \wedge p)$

$\Diamond f \supset (F_f p \equiv (\neg f \, U \, (p \wedge f)))$

$\bigcirc_f p \equiv \bigcirc F_f p$

$p \, U_f \, q \equiv (f \supset p) \, U \, (f \wedge q)$

Remarks:

- When the formula $\Box f$ holds, the operators $\Box_f$, $\Diamond_f$, $\bigcirc_f$, U_f respectively coincide with their homologues $\Box, \Diamond, \bigcirc, U$. In this case F_f is the identity operator.

- The operator F_f is not completely defined; its value is only known on models satisfying $\Diamond f$. This fact is not surprising since, if f is never true, there is no information about the value of p when f is true. The same remark applies to the operator $\bigcirc_f$.

3.3 Describing boolean LUSTRE

Here, we consider programs which only contain boolean variables and expressions. The semantics of such a program P is then a set $\Sigma(P)$ of histories belonging to Hist({true, false, nil, $\perp$}). From the semantics of LUSTRE , we have

$$\frac{[P] \xrightarrow{-\sigma_0\rightarrow} [P'] \, , \, \sigma \in \Sigma(P')}{\sigma_0.\sigma \in \Sigma(P)}$$

Only histories belonging to Hist({true,false}) can straightforwardly be considered as models for our temporal logic. So, from the point of view of logical properties, we shall consider both undefined values nil (unassigned variable) and $\perp$ (uncomputed variable) as undetermined values, which only satisfy the predicate "true". So, with each history $\Sigma=\{\sigma_0,\sigma_1,\ldots,\sigma_n,\ldots\}$, we associate a subset $S(\Sigma)$ of Hist({true,false}) defined as follows:

$S(\Sigma) = \{\Sigma'=\{\sigma'_0,\ldots,\sigma'_n,\ldots\} \mid \forall n\geq 0, \forall x\in ID, \sigma_n(x)\neq nil \ \& \ \sigma_n(x)\neq\perp \Rightarrow \sigma_n(x)=\sigma'_n(x)\}$

Now, we want to associate with each program P, a formula $f(P)$ such that

$|f(P)|(\Sigma') = true$ iff there exists $\Sigma\in\Sigma(P))$ such that $\Sigma'\in S(\Sigma)$

This association is inductively defined by the following rules, where c always stands for the clock of X:

$f(X=Y) = \Box_c (X \equiv Y)$

$f(X= Y \text{ op } Z) = \Box_c (X \equiv Y \text{ op } Z)$, for any boolean operator op

$f(X=pre(Y)) = \Box_c (O_c (X) \equiv Y)$

$f(X=Y\text{->}Z) = F_c (X \equiv Y \wedge O_c \Box_c (X \equiv Z))$

$f(X=Y \text{ when } Z) = \Box_Z (X \equiv Y)$

$f(X=current(Y)) = \Box_c (c'\supset(X\equiv Y) \wedge O_c (\neg c')\supset(O_c (X)\equiv X))$, where c' is the clock of Y

The formula associated with a system of equations is obviously the conjunction of the formulas associated with its components:

$f(eq_1;eq_2) = f(eq_1) \wedge f(eq_2)$

3.4. Correctness

Let us sketch the proof of correctness of the formula associated with the "pre" operator: The set of histories compatible with a LUSTRE program has been defined by means of a sequence of program states. Let us extend our notion of model as follows: an extended history is an infinite sequence of pairs, each pair consisting of a program and a memory. When the program is reduced to a single equation, such an extended history is a sequence

$$\Xi=(eq_0,\sigma_0).(eq_1,\sigma_1). \ldots .(eq_n,\sigma_n). \ldots)$$

satisfying $[eq_n]-\sigma_n\rightarrow[eq_{n+1}]$, for any n. Let us introduce the following basic predicates on extended histories:

- [eq] is true for Ξ, iff $\Xi = (eq,\sigma).\Xi'$, for some σ and some Ξ'

- Let p be a basic predicate on memories; then {p} is true for Ξ, iff $\Xi = (eq,\sigma).\Xi'$, for some eq and some Ξ', and for some σ such that $|p|(\sigma)$ holds. This notation is extended to full memories: $\{\sigma\}$ is true for Ξ iff σ is the memory part of the first pair of Ξ.

So, a rewriting predicate $[eq]-\sigma\rightarrow[eq']$ may be expressed by

$$[eq] \wedge \{\sigma\} \supset O[eq']$$

and from the special form of our inference rules, a rule

$$\frac{p_1, p_2, \cdots, p_k}{q}$$

may be translated into

$$\vdash \Box(p_1 \wedge p_2 \wedge \ldots \wedge p_k \supset q)$$

Now, from the rules defining the "pre" operator, we get

$$\Box((\{Y \neq \bot\} \wedge [X = pre(Y,k)]) \supset \{X = k\})$$

and

$$\Box((\{Y = \bot\} \wedge [X = pre(Y,k)]) \supset (\{X = \bot\} \wedge \bigcirc[X = pre(Y,k)])$$

So,

$$\Box(\Diamond\{Y \neq \bot\} \supset ([X = pre(Y,k)] \supset \{X = \bot\} U \{X = k\})$$

On the other hand, the rules provide

$$\Box((\{Y = k'\} \wedge [X = pre(Y)]) \supset \bigcirc[X = pre(Y,k')])$$

$$\Box(\{Y = k'\} \wedge [X = pre(Y,k)]) \supset \bigcirc[X = pre(Y,k')])$$

It follows that

$$([X = pre(Y)] \wedge \Box\Diamond\{Y \neq \bot\}) \supset \Box(\{Y = k\} \supset \bigcirc\{X = \bot\} U \{X = k\}) \qquad (1)$$

Now, the following lemma results from the clock correctness:

$[X = pre(Y)] \supset (\{X \neq \bot\} \equiv \{Y \neq \bot\} \equiv c)$, where c is the common clock of X and Y

So, the implication (1) becomes

$$([X = pre(Y)] \wedge \Box\Diamond c) \supset \Box(\{Y = k\} \supset \bigcirc(\neg c U (c \wedge \{X = k\})))$$

or

$$([X = pre(Y)] \wedge \Box\Diamond c) \supset \Box_{\{Y=k\}} \bigcirc_c \{X = k\}$$

Separately considering the cases where k is true or false, we get

$$([X = pre(Y)] \wedge \Box\Diamond c) \supset \Box_c (\bigcirc_c X) \equiv Y$$

which is the desired result.

3.5. Application

Let us come back to the program of the axle detector. An expanded form of the program is:

```
            S  = false -> out_f and lastin_b;
   last_in_b  = (P1 and not pre_P1 and not P2) or
                        (not(P2 and not pre_P2 and not P1 ) and pre_last_in_b );
        out_f = not P2 and pre_P2 and not P1;
       pre_P1 = pre(P1);
       pre_P2 = pre(P2);
pre_last_in_b = pre(last_in_b);
```

From this program and the rules of §3.3, we get the following formula, where b stands for the basic clock of the program:

$$F_b(\neg S \wedge O_b \Box_b (S \equiv (out\_f \wedge last\_in\_b) \wedge$$
$$\Box_b\ last\_in\_b \equiv ((P1 \wedge \neg pre\_P1 \wedge \neg P2)\ \vee (\neg(P2 \wedge \neg pre\_P2 \wedge \neg P1) \wedge pre\_last\_in\_b)) \wedge$$
$$\Box_b\ out\_f \equiv (\neg P2 \wedge pre\_P2 \wedge \neg P1) \wedge$$
$$\Box_b\ (O_b\ (pre\_P1) \equiv P1\ \wedge\ O_b\ (pre\_P2) \equiv P2\ \wedge\ O_b\ (pre\_last\_in\_b) \equiv last\_in\_b)$$

Let us call this formula *axles* . The program is intended to work under the assumption of synchrony: it runs at a suitable rate so as to perceive any change in the state of its inputs. Let us call *chg* the formula which is true whenever the state of a pedal changes:

$$\neg(P1 \equiv OP1) \vee \neg(P2 \equiv OP2)$$

The assumption of synchrony may be written as the following formula, *sync* :

$$\Box(chg\ \supset b)$$

Moreover, it is assumed that the state of only one pedal, as perceived by the program, may change at a given instant. This assumption is expressed by the following formula, called *excl* :

$$\Box_b\ (\neg(P1 \equiv O_b\ P1) \supset (P2 \equiv O_b\ P2))$$

The proof of an expected property P of the program consists of proving the theorem

$$\vdash (axles\ \wedge sync\ \wedge excl) \supset P$$

For instance, P may express that whenever a train straightly crosses the zone Z from the backward district to the forward one, the output S is true. Such a property is simpler to write on the clock *chg* . The straight crossing of a train from the backward district to the forward one, is expressed by the formula *b-f-cross* :

$$\neg P1 \wedge \neg P2 \wedge O_{chg}P1 \wedge O_{chg}O_{chg}P2 \wedge O_{chg}O_{chg}O_{chg}\neg P1 \wedge O_{chg}O_{chg}O_{chg}O_{chg}\neg P2$$

and the theorem to be proven is

$$\vdash (axles\ \wedge sync\ \wedge excl\ ) \supset \Box_b\ (b\text{-}f\text{-}cross \supset O_{chg}O_{chg}O_{chg}O_{chg}\ S)$$

4. A logic to deal with multiform time

Our multiform-time operators have been shown to be quite useful. Let us now examine their specific properties.

First, we can adapt the axioms of standard temporal operators to our operators. The following properties have been shown, for any formulas f,p,q:

B0. $\vdash \Diamond_f p \equiv \neg\Box_f(\neg p)$
B1. $\vdash \Box_f(p \supset q) \supset (\Box_f p \supset \Box_f q)$
B2. $\vdash \circ\Diamond f \supset (O_f(\neg p) \equiv \neg O_f p)$
B3. $\vdash \circ\Diamond f \supset (O_f(p \supset q) \supset (O_f p \supset O_f q))$
B4. $\vdash \Box\Diamond f \supset (\Box_f p \supset (F_f p \wedge O_f \Box_f p))$
B5. $\vdash \Box_f(p \supset O_f p) \supset (F_f p \supset \Box_f p)$
B6. $\vdash p\ U_f q \supset \Diamond_f q$
B7. $\vdash \Box\Diamond f \supset (p\ U_f q \equiv F_f q \vee (F_f p \wedge O_f(p\ U_f q)))$
B8. $\vdash \Diamond f \supset (F_f(\neg p) \equiv \neg F_f p)$
B9. $\vdash \Diamond f \supset (F_f(p \supset q) \equiv F_f p \supset F_f q)$

A comparison of properties B0-B7 and axioms A0-A7 shows that they only differ in that properties B2, B3, B4 and B7 need an additional premise which states that the formula f must be true infinitely often. So, a good property for an event defining a time-scale is to occur infinitely often (*time never stops*). When time-scales satisfy this property, one can change the time-scale of a system, and adapt all the formulas proven for the initial system only by changing the time-scale (subscript) of the modal operators.

Now, the following properties allow a formula to be projected onto a finer time-scale:

B10. $\vdash \Box(f' \supset f) \supset (\Box_{f'}(p) \equiv \Box_f(f' \supset p))$
B11. $\vdash \Box(f' \supset f) \supset (\Diamond_{f'}(p) \equiv \Diamond_f(f' \wedge p))$
B12. $\vdash (\Box(f' \supset f) \wedge \Diamond f') \supset (F_{f'} p \equiv (\neg f'\ U_f(p \wedge f')))$
B13. $\vdash (\Box(f' \supset f) \wedge \Diamond f') \supset (O_{f'} p \equiv O_f F_{f'} p)$
B14. $\vdash \Box(f' \supset f) \supset (p\ U_{f'} q \equiv (f' \supset p)\ U_f(f' \wedge q))$

These properties express that, if f is more frequently true than f′ (that is, if the formula $\Box(f' \supset f)$ holds) then the modal operators on f′ may be defined in terms of the modal operators on f, in exactly the same way as they were defined in terms of the usual operators (on *true*).

Conclusion

In this paper we have shown that the synchronous, data-flow language LUSTRE may be viewed as a subset of a linear temporal logic. This fact is an argument showing the cleanness of the semantics of LUSTRE. Moreover, it suggests that standard decision procedures can be used for proving boolean LUSTRE programs.

On the other hand, translating LUSTRE in temporal logic led us to introduce some abbreviations, which are used as new temporal operators, dealing with multiform time. The usefulness of these operators for expressing properties of real-time systems has been illustrated.

This work must be continued in several directions:

- Of course, non boolean computation must be introduced in the logic. This does not raise serious problem, but involves the loss of decision procedures.
- The same construction must be done in a branching time logic, thus allowing non-deterministic inputs to be taken into account.
- Ultimately, this work should lead to an environment for specifying and verifying LUSTRE programs, similar to some tools [6, 15] developed for imperative languages.

References

1. Ashcroft E.A., Wadge W.W.: *LUCID, the data-flow programming language* . Academic Press, 1985.
2. Austry D., Boudol G.: *Algèbre de processus et synchronisation* . TCS 30, april 84.
3. Bergerand J-L., Caspi P., Halbwachs N., Pilaud D., Pilaud E.: *Outline of a real-time data-flow language* . 1985 Real-Time Symp., San Diego, dec. 85.
4. Berry G., Cosserat L.: *The ESTEREL programming language and its mathematical semantics* . RR nr. 327, INRIA, 1984. To appear in Science of Computer Programming.
5. Caspi P., Pilaud D., Halbwachs N., Plaice J.: *LUSTRE: a declarative language for programming synchronous systems* . 14th ACM Symp. on Principles of Programming Languages, Munich, january 87.
6. Clarke E.M., Emerson E.A., Sistla A.P.: *Automatic verification of finite-state concurrent systems using temporal logic specifications* . ACM TOPLAS, 8(2), 1986.
7. Gabbay D., Pnueli A., Shelah S., Stavi J.: *On the temporal analysis of fairness* . 7th ACM Symp. on Principles of Programming Languages, Las Vegas, january 80.
8. Harel D.: *Statecharts: A visual approach to complex systems* . Advanced NATO Institute on Logics and Models for Verification and Specification of Concurrent Systems, La Colle-sur-Loup, 1984.
9. Le Guernic P., Benveniste A., Bournai P., Gautier T.: *SIGNAL: a data-flow oriented language for signal processing* . RR nr. 378, INRIA, 1985.
10. Koymans R., DeRoever W.P.: *Examples of a real-time temporal logic specification* . In The analysis of Concurrent Systems, LNCS nr.207, august 83.

11. Manna Z., Pnueli A.: *Verification of concurrent programs: the temporal framework* . In The Correctness Problem in Computer Science (R.S.Boyer and J.S.Moore, eds), International Lecture Series in Computer Science, Academic Press, London, 1982.
12. Moszkowski B.C.: *Reasoning about digital circuits* . PhD Thesis, Report STAN-CS-83-970, Dept. of Computer Science, Stanford University, july 83.
13. Plotkin G.D.: *A structural approach to operational semantics* , Lecture Notes, Aarhus University, 1981.
14. Pnueli A.: *The temporal logic of concurrent programs* . 12th ICALP, LNCS 194, 1977.
15. Queille J-P., Sifakis J.: *Specification and verification of concurrent systems in CESAR* . 5th Int. Symp. on Programming, Springer Verlag 1981.
16. Schwartz R.L., Melliar-Smith P.M., Vogt F.H.: *An interval logic for higher-level temporal reasonning: language definition and examples* . Technical Rep. CSL-138, Computer Science Lab., SRI International, february 83.