Affine Transformations in SIGNAL and Their Application in the Specification and Validation of Real-Time Systems

Irina Smarandache and Paul Le Guernic

IRISA-INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract. In this paper we present affine transformations as an extension of the SIGNAL language for the specification and validation of real-time systems. To each SIGNAL program is associated a system of equations which specify synchronization constraints on clock variables. The SIGNAL compiler resolves these equations and verifies if the control of a program is functionally safe. By means of the new transformations, affine relations can be defined between clocks and it gets necessary to enhance the compiler with facilities for the resolution of synchronization constraints on these clocks. We propose thus an extension of the compiler based essentially on a canonical form of the affine relations.

1 Introduction

Real-time systems, and more generally reactive systems [1], are in continuous interaction with their environment. Therefore, they must respond *in time* to external stimuli. Moreover, real-time systems must be safe, thus one would wish to prove their correctness. Response time and safety are two important aspects in the design of real-time applications.

In order to deal with these aspects, a new family of languages has emerged – *the synchronous languages* [2]. The hypotheses of synchronous programming are: 1. All actions in a system are instantaneous; 2. Two or more actions can take place at the same instant (they are simultaneous). Thus, from the point of view of the temporal properties of a system, only succession and simultaneity of instants are of interest, their exact time values have no signification.

SIGNAL [3] is a synchronous language developed for the specification, validation and implementation of real-time systems. Due to the synchronous approach, the verification of the logical properties of an application is possible before considering the physical properties of a particular implementation. The notion of *clock* is important in SIGNAL: with each SIGNAL variable is associated a clock that summarizes the temporal properties of the variable. A SIGNAL program is a system of equations which specify synchronization constraints on clocks and dependencies between values of variables (data). The SIGNAL compiler resolves these equations and verifies if the control of a program is functionally safe. Currently, proving the correctness of a SIGNAL program with respect to its clock constraints is based on boolean equation resolution methods which appeared sufficient for the validation of real-time systems where control is an important aspect [4, 5]. Most of the real-time signal processing applications imply important numeric calculi; in order to treat them, we define *affine transformations* on clocks and extend the existing compiler in order to treat synchronizations on clocks defined by affine transformations. The extended compiler has more powerful proof facilities which increase the specification and validation power of the SIGNAL language, either if SIGNAL is employed alone or in conjunction with other languages in a data flow framework.

Large grain data flow programming [6] has been considered a convenient paradigm for describing digital signal processing systems. Nevertheless, when such systems must respect real-time constraints, the runtime overhead induced by dynamic scheduling has to be eliminated. Data flow models and languages have been adapted to tackle this aspect. *Synchronous Data Flow* (SDF) graphs [7, 8] are particular data flow graphs whose properties permit to verify statically the existence of sequential and parallel schedules. Moreover, in [9] the SDF model is enlarged in order to tackle with a larger range of applications by introducing boolean conditions in a generalized SDF model.

Another example of restricted data flow graphs is that of *reduced dependence graphs* which are specifications of systems in terms of periodic acyclic precedence graphs, where only one period is illustrated, and its dependence on previous periods is specified by indexing [10]. Reduced dependence graphs have optimal schedules which can be found at compile time. Their application domain is more restricted than for SDFs: reduced dependence graphs are generally used to describe regular iterative algorithms which can be mapped onto processor arrays.

Based on this approach, the ALPHA language has been developed as a functional language based on the formalism of affine recurrence equations [11]. ALPHA allows the description of regular algorithms, like multiplication of matrices or vectors, and the transformation of programs for optimization. A transformational environment has been developed around ALPHA for the synthesis of parallel and regular architectures [12, 13].

Although based on a control-oriented formalism [5], SIGNAL has been used in the specification of periodic synchronous VLSI systems modeled by SDF graphs [14, 15]. This has pointed out two problems: the need to define a communication interface between SDF nodes and the important time and space resources necessary to verify if an application is functionally correct. Moreover, SIGNAL does not treat adequately computations on regular data like matrices or vectors. Thus, SDF nodes describing treatments on multidimensional data have been expressed using the C language. In order to obviate these problems and due to the complementary properties of the languages SIGNAL and ALPHA, we have projected a hybrid framework [16] in which the two languages are combined for the specification, validation and implementation of applications. ALPHA provides optimal scheduling on hardware or software for regular iterative algorithms, while SIGNAL specifies the general control context in which ALPHA algorithms execute. For the modeling and validation of the SIGNAL-ALPHA cooperation, affine transformations on SIGNAL clock variables are introduced. In order to treat synchronization constraints on clocks obtained by affine transformation, we must extend the current proof system implemented in the SIGNAL compiler.

The paper is organized as follows: in Sect. 2 we introduce the SIGNAL language and its compiler and present the context of our work. In Sect. 3, the new features introduced in SIGNAL are described, namely the affine transformations and a canonical form of the affine relations induced thereby. Finally, in Sect. 4, we conclude with applications and future directions of our work.

2 The SIGNAL Language

SIGNAL as a synchronous and equation-based language, has proved very useful for programming correct real-time systems. The synchronous approach enables the abstraction of the specification from the details of a given implementation, offering the possibility to verify logical properties and to guarantee the functional safety of an application [2]. Moreover, programming in SIGNAL consists simply of writing systems of equations on program variables [3]. The SIGNAL compiler resolves these systems and verifies thus the correctness of the program.

A variety of tools exist around the SIGNAL language and its compiler; among these, a *Graphical User Interface* for high-level system design [17] and a C code generator for functional simulation. The basic data structure of the SIGNAL environment is the *synchronized data-flow graph* (SDG) which is constructed during the compilation from the clock equations and data dependencies graphs associated with each SIGNAL equation [4]. The construction of the SDG proceeds in parallel with the verification if the program is deterministic, has no cycles and if the coded functional properties are guaranteed.

From all operations performed by the SIGNAL compiler, we will particularly insist on those methods involved in the resolution of the synchronization constraints on clocks, i.e. the *clock calculus*. Therefore, we give a more detailed presentation of SIGNAL and its clock calculus and we discuss an example of a SIGNAL program in detail. We conclude this section with a presentation of the context of our work and an introduction to the affine transformations in SIGNAL.

2.1 Basic Concepts

Before introducing the basic language constructs and the clock equations, we give some definitions. The SIGNAL language operates on variables called *signals* which are infinite sequences of values, each value being indexed by a positive integer temporal *index*. The values of the index domain define the logical instants where the signal is present, i. e. the *clock* of the signal. Two signals have the same clocks if they are present (respectively absent) at the same instants.

The SIGNAL kernel is defined by a small number of elementary processes (also called statements). Each process has a formal semantic that introduces a clock equation and a data dependencies graph on the signals involved [18]. We present briefly the elementary processes and the extensions of SIGNAL in Table 1 (for a more detailed presentation refer to [3]). The clock equations associated with the statements are shown in the right-hand column of Table 1.

Table 1. Basic SIGNAL constructs and extensions.

	Syntax	Clock calculus
Elementary processes		
stepwise extensions: op is an arithmetic, relational or boolean operator	X := A op B	$\hat{X} = \hat{A} = \hat{B}$
delay: ZA is the $\mathtt{N}^{th}$ past value of A	ZA := A \$ N	$\hat{ZA} = \hat{A}$
down-sampling by boolean condition: Y is equal to X when C is present and true	Y := X when C	$\hat{Y} = \hat{X} \wedge [C]$
merge with priority: Z is equal to X if X is present; else Z is equal to Y if Y is present; else Z is absent	Z := X default Y	$\hat{Z} = \hat{X} \vee \hat{Y}$
process composition: The SIGNAL processes P and Q share their identically named signals	(| P | Q |)	
process restriction: The signal a is not visible to the exterior of the process P	P/a	
SIGNAL extensions		
The clock of the true instants of C	X := when C	$\hat{X} = [C]$
The clock of X	Y := ^X	$\hat{Y} = \hat{X}$
Explicit synchronization constraint: the clock of A is equal to the clock of B	A ^= B	$\hat{A} = \hat{B}$

By $\hat{X}$ we denote the clock of the signal X. The clock $[C]$ defines the instants where the boolean-valued signal C is present and has the value *true*. The classical set-operators $(\wedge, \vee, \setminus)$ can be applied to clocks. The inclusion relation between sets defines a partial order on the set of clocks; moreover the clock *poset*[1] is a boolean lattice [19]. The SIGNAL clock calculus uses the properties of the boolean lattice and the relation that exists between $[C]$ and $[\neg C]$, given by the following *partition* equations:

$$[C] \vee [\neg C] = \hat{C} \qquad [C] \wedge [\neg C] = \emptyset \ . \tag{1}$$

where $\emptyset$ is the empty set of instants. These equations state that the pair $([C], [\neg C])$ defines a *partition* of the clock $\hat{C}$.

2.2 The SIGNAL Compiler

The SIGNAL compiler resolves the clock equations and synthesizes the most frequent clock *(master clock)* of a SIGNAL program, if such a clock exists. In this case, the program is said to be *endochronous* and the master clock defines its timing rate. Executable code for functional simulation can be generated for endochronous programs. If the result of the clock calculus is a set of local maxima clocks, the SIGNAL process is *exochronous*. The synchronizations of its variables are not completely specified and executable code cannot be generated.

The resolution method is based on a triangularization strategy and a tree representation [5]. The idea is to derive definitions for all clocks in terms of already defined ones and to verify the equivalence between multiple definitions.

[1] A *poset* is a set with a partial order on it.

The properties of a boolean lattice and the partition equations are rewriting rules used during the compilation. The tree structure speeds up the rewritings and incrementally builds the master clock of the program.

2.3 Example of a SIGNAL Program

In Fig. 1 we give an example of a SIGNAL program and show the associated clock calculus. More details on the clock calculus are provided in Sect. 3.3.

```
(1) process OVERSAMPLE =
(2)  ( ? integer FB;  % FB is an input signal %
(3)    ! integer N )  % N is an output signal %
(4)   (| N:= FB default (ZN - 1)
(5)    | ZN:= N $ 1
(6)    | FB ^= when (ZN <= 1))  |)
(7)  where integer ZN init 1
(8)  end
```

Clock equations

(4) $\hat{\texttt{N}} = \hat{\texttt{FB}} \vee \hat{\texttt{ZN}}$

(5) $\hat{\texttt{ZN}} = \hat{\texttt{N}}$

(6) $\hat{\texttt{FB}} = [\texttt{ZN} \leq 1]$

Fig. 1. A SIGNAL program and the associated clock calculus.

The SIGNAL program in Fig. 1 implements an oversampling: for each input value FB, the OVERSAMPLE process produces FB output values for N (cf. the timing diagram in Fig. 2). ZN is the previous value of N; line (4) of the program states that, for each logical instant, the value of N is equal to that of FB, if the signal FB is present, or to (ZN $-$ 1), if FB is absent ('-' in Fig. 2). The input signal FB is accepted only at the *true* value instants of (ZN $\leq$ 1) (see line (6)).

From the associated clock equations, it results that the clock of the output N is "more frequent" than that of the input FB. Moreover, the clock $\hat{\texttt{N}}$ is the only upper limit for the SIGNAL program; therefore, the program is endochronous.

	t_0	t_1	t_2	t_3	t_4	t_5	$\cdots$
FB :	3	–	–	2	–	5	$\cdots$
ZN :	1	3	2	1	2	1	$\cdots$
N :	3	2	1	2	1	5	$\cdots$

Fig. 2. Timing diagram.

2.4 Introduction to Affine Transformations on Clocks

The facility for expressing oversampling in SIGNAL permits to describe temporal properties of processes that, although specified as instantaneous, exhibit a clear succession of logical instants (e.g. pipelining in algorithms for matrices, vectors and generally regular data). Presently, SIGNAL disposes of limited facilities for programming algorithms on regular data. This limit is removed by the possibility to call *external processes*: an external process can be the specification of an algorithm in other languages than SIGNAL. For example, the SIGNAL *functions* are external processes with synchronous inputs and outputs; because of the synchrony of its entries, a function can be a user-defined C function.

Because of its facilities for expressing computations on regular data with affine recurrence equations, the ALPHA language [13] can be used in cooperation with SIGNAL for the specification, validation and implementation of real-time systems. ALPHA programs are represented as external processes in SIGNAL. Two situations may occur [16]:

1. The external ALPHA process generates a SIGNAL *function* whose execution is considered instantaneous. Figure 3 a) shows the example of the multiplication of two synchronous matrices $\mathbf{A}$ and $\mathbf{B}$ ($\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$).
2. The external ALPHA process generates a SIGNAL *process* with asynchronous inputs and outputs: the specification/architecture transformations existing in the ALPHA environment induce a temporal index which must be considered in SIGNAL. In Fig. 3 b), $\mathbf{C}$ is not synchronous with $\mathbf{A}$ and $\mathbf{B}$.

In the latter case, the regularity of manipulated data is preserved by the relative position of the clocks $\hat{C}$ and $\hat{A}$, respectively $\hat{B}$: the output $\mathbf{C}$ is present at every 4^{th} instant after the presence of the inputs $\mathbf{A}$ and $\mathbf{B}$ (in Fig. 3 b) below $1, 2, 3$ are instants where intermediate computations on $\mathbf{C}$ are performed). This is an example where a clock $\hat{C}$ has been obtained by an *affine transformation* of the clock $\hat{A}$. Clocks $\hat{C}$ and $\hat{A}$ are said to be in *affine relation*.

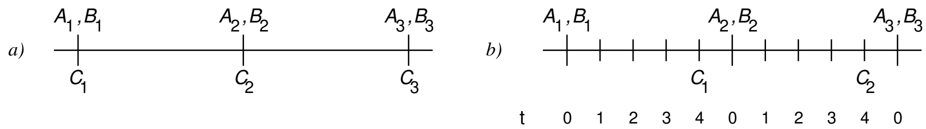

Fig. 3. Affine transformations in SIGNAL: example.

3 Affine Transformations in SIGNAL

Affine transformations on clocks generate affine relations whose properties permit to check synchronization constraints which cannot be proved using the current compiler. In this section we present the operators introduced in SIGNAL to support affine transformations. We then describe the existing clock calculus and its limits and introduce the main principles of the new *affine clock calculus*. We conclude by presenting the equivalence relation between two affine relations and the canonical form of affine relations with respect to their equivalence class.

3.1 Definition

An *affine transformation* of parameters (n, φ, d) applied to the clock H produces a clock K by inserting $(n-1)$ instants between any two successive instants of H and counting each d^{th} instant, from the φ^{th} instant on (see Fig. 4).

The affine transformation (n, φ, d) defines an (n, φ, d)-affine relation between H and K. Moreover, the affine relation is oriented in the sense the affine transformation is performed, i.e. *from* H *to* K; it will be noted $H \stackrel{(n,\varphi,d)}{\rightarrow} K$.

A $(d, -\varphi, n)$-affine relation exists also between K and H : $K \stackrel{(d,-\varphi,n)}{\rightarrow} H$ (i.e. H can be obtained by applying the $(d, -\varphi, n)$ transformation on the clock K).

Figure 4 presents an example of a clock K obtained by an affine transformation applied to the clock H. A clock *more frequent* than H and K is represented: its instants are vertical lines, while those of H and K are presented using circles. The distance between two successive instants has no signification in SIGNAL; only the succession and relative position of instants are important.

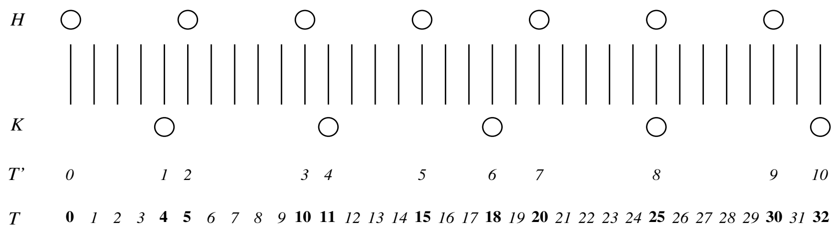

Fig. 4. Example : H and K are in $(5, 4, 7)$-affine relation.

3.2 Basic Affine Constructs

We propose three new operators in order to express affine transformations on SIGNAL variables. The basic process is:

`Y:= sample{`φ`,d}(X )`

where φ and `d` are positive integers (`d` strictly positive). It defines the signal `Y` as a down-sampling with phase φ and period `d` on the signal `X`. An example for $\varphi = 3$ and `d` $= 4$ is presented in Fig. 5.

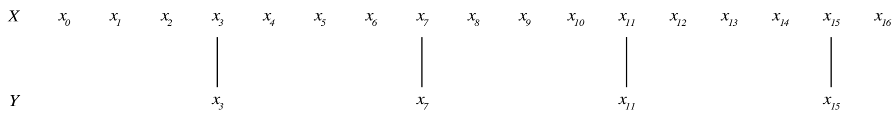

Fig. 5. `Y:= sample{3,4}(X )`. `Y` is present at every 4^{th} instant, starting with instant 3.

We now use `sample` to define two other affine processes as SIGNAL extensions. For $\mathtt{n}, \mathtt{d} > 0$ and $\varphi \in \mathbb{Z}$, `clk_affine{n,`φ`,d}(X,Y )` defines an affine relation $(\mathtt{n}, \varphi, \mathtt{d})$ between the clocks of `X` and `Y`. Its definition in SIGNAL is[2]:

```
                                 (| H:= sample{ max(0, -φ), n }(I)
clk_affine{n,φ,d}(X,Y ) ⇔        | K:= sample{ max(0, φ), d }(I)
  (n, d > 0 and φ ∈ ℤ)           | H ^= X | K ^= Y |) / H, K, I
```

Finally, `Y:= unsample{n,`φ`}(X, Z)`, with $\mathtt{n} > 0$ and $\varphi \in \mathbb{Z}$, defines `Y` as an affine oversampling using `X` and `Z`:

```
                                 (| clk_affine{ n, φ, 1 }(X, Y)
Y:= unsample{n,φ}(X, Z) ⇔        | Y:= ( X when ^Y ) default Z
   (n > 0 and φ ∈ ℤ)             | Y ^= Z |)
```

For a clock H, we note $[H]_{(\varphi,n)}$ the clock obtained by down-sampling with phase φ and period n on H and $(H)_{(n,\varphi,d)}$ the result of an affine transformation of parameters (n, φ, d) applied to the clock H.

Synchronization constraints on clocks obtained by affine transformations can not be verified with the current compiler. It is thus necessary to provide new proof techniques based on the properties of affine relations. In the sequel we present the *affine clock calculus* as an extension of the *clock calculus* existing in the compiler.

3.3 The Clock Calculus

In order to test the correctness of a SIGNAL program with respect to its clock constraints and to generate executable code for functional simulation, it is necessary to resolve the clock equations and to determine the most frequent clock of a SIGNAL program, if such a clock exists.

Resolution Strategy. The resolution algorithm existing in the compiler is based on a *triangularization* scheme: a triangular system of equations is progressively constructed from the original system (represented in Table 1). For each equation $H = H_1 \; op \; H_2$, $(op \in \{\wedge, \vee, \setminus\})$, a right-to-left orientation is performed in order to consider the formula $H_1 \; op \; H_2$ as the definition of H. During this process, some orientations are not possible. There are two reasons for that:

1. The equation under consideration is of the form $H = H_1 \; op \; H_2$, but there is already a definition for H. In this case a rewriting can be performed in order to verify the equivalence of $H_1 \; op \; H_2$ with the existing definition.

[2] We use the following identity: $\varphi \; = max(0, \varphi) - max(0, -\varphi)$.

2. An orientation will induce a cycle in the clock-to-clock dependency graph obtained by triangularization. In this case, a rewriting can be performed to break the cycle.

At the end of the compilation process, the program is said to be temporally incorrect if there are non-proved equalities or equations whose orientation induces a cycle. A detailed description of this strategy can be found in [4].

Example and Limits. We illustrate in this section one of the limits of the clock calculus. Consider the SIGNAL program given below. It contains a modification of the process `OVERSAMPLE` presented in Fig. 1: in the process `OVERREG`, for each input value `FB`, $(\mathtt{m}+1)$ values are produced for the output `N`.

```
     process EXAMPLE = {integer m}
         (? integer X;! integer Y)
(1)        (| Y1:= OVERREG{m}(X)
(2)         | Y2:= OVERREG{m}(X)
(3)         | Y:= Y1 * Y2  |)
     where integer Y1, Y2
           process OVERREG = {integer m}
             (? integer FB;! integer N)
            (| N:= m when ^FB default (ZN-1)
             | ZN:= N $ 1
             | FB ^= when (ZN = 0) |)
            where integer ZN init 0
          end
     end
```

a) Clock eqs.	b) Clock calc.
(1) $\hat{Y}_1 = \hat{X} \vee Z\hat{Y}_1$	$Z\hat{Y}_1 = \hat{Y}_1$
$Z\hat{Y}_1 = \hat{Y}_1$	$Z\hat{Y}_2 = \hat{Y}_1$
$\hat{X} = [ZY_1 = 0]$	$\hat{Y}_2 = \hat{Y}_1$
(2) $\hat{Y}_2 = \hat{X} \vee Z\hat{Y}_2$	$\hat{Y} = \hat{Y}_1$
$Z\hat{Y}_2 = \hat{Y}_2$	$\hat{X} = [ZY_1 = 0]$
$\hat{X} = [ZY_2 = 0]$	$\hat{X} = [ZY_2 = 0]$
(3) $\hat{Y} = \hat{Y}_1 = \hat{Y}_2$	

Fig. 6. A more complex SIGNAL example.

During the compilation, lines (1) and (2) are replaced with the instances of `OVERREG` corresponding to each set of effective parameters. We present in Fig. 6 a) the clock equations associated with each line after the instantiation. After the clock calculus the transformed system is presented in Fig. 6 b). Clocks $Z\hat{Y}_1$, $Z\hat{Y}_2$, $\hat{Y}$ and $\hat{Y}_2$ have been defined as equal to $\hat{Y}_1$. The last two equations contain two definitions for $\hat{X}$: $[ZY_1 = 0]$ and $[ZY_2 = 0]$. Therefore, these formulas have to be proved equivalent (as one can write $[ZY_1 = 0] = [ZY_1 = 0] \wedge Z\hat{Y}_1$ and $[ZY_2 = 0] = [ZY_2 = 0] \wedge Z\hat{Y}_2$, the conditions of case 1 of the triangularization strategy are satisfied). The program `EXAMPLE` will be rejected as incorrect because the equivalence of $[ZY_1 = 0]$ and $[ZY_2 = 0]$ can not be proved, although $\mathtt{ZY}_1$ and $\mathtt{ZY}_2$ have 0 value at the same instants (no calculus can be performed on values of signals). We will see in the sequel how this limit can be removed using the affine clock calculus.

3.4 The Affine Clock Calculus

At present, the compiler manipulates systems of equations on clocks that can be free variables in the boolean lattice, down-samplings by boolean conditions, or expressions constructed from the other clocks using the operators $\wedge, \vee$ and $\setminus$. In order to tackle equations on clocks obtained by affine transformations, we need new rewriting rules based on the properties of affine relations.

Resolution Strategy. The resolution algorithm is similar to the triangularization presented above. Each equation $H = \Psi_H$, with Ψ_H a formula containing at most two already defined clocks H_1 and H_2, is oriented to the left so that Ψ_H will be considered as the definition of the clock H. As before, the orientation process is not solvable if:

1. The equation under consideration is of the form $H = H_1\ op\ H_2$ with H_1 and H_2 in (n, φ, d)-affine relation and if there is already a definition of the form $H = (H_1)_{(n'_1, \varphi'_1, d'_1)}$ or $H = (H_2)_{(n'_2, \varphi'_2, d'_2)}$.
   In case there exist (n_1, φ_1, d_1), (n_2, φ_2, d_2) induced by (n, φ, d) such that $H = (H_1)_{(n_1, \varphi_1, d_1)}$ and/ or $H = (H_2)_{(n_2, \varphi_2, d_2)}$, the following equivalences should be proved: the relation $H_1 \stackrel{(n_1, \varphi_1, d_1)}{\rightarrow} H$ equivalent with $H_1 \stackrel{(n'_1, \varphi'_1, d'_1)}{\rightarrow} H$ and/or $H_2 \stackrel{(n_2, \varphi_2, d_2)}{\rightarrow} H$ equivalent with $H_2 \stackrel{(n'_2, \varphi'_2, d'_2)}{\rightarrow} H$.
2. The current equation is of the form $H = H\ op\ H'$, with H and H' in affine relation. The orientation of the equation will induce a cycle in the clock-to-clock dependency graph. In this case, the (n, φ, d)-relation between $H\ op\ H'$ and H should be calculated and the equivalence between $H \stackrel{(n, \varphi, d)}{\rightarrow} H$ and $H \stackrel{(1,0,1)}{\rightarrow} H$ demonstrated (as it will be demonstrated in the sequel, the canonical affine relation between H and H is $(1, 0, 1)$).
3. The equation is of the form $H = (H_1)_{(n, \varphi, d)}$ and there exists another definition of H of the form $H = (H_1)_{(n', \varphi', d')}$. In this case, the equivalence between $H_1 \stackrel{(n, \varphi, d)}{\rightarrow} H$ and $H_1 \stackrel{(n', \varphi', d')}{\rightarrow} H$ has to be checked.

In this paper, we outline the theoretical support concerning the equivalence relation existing on the set of affine relations and the *canonical form* of affine relations, two important properties used in the resolution algorithm presented above. Because of space constraints, we give only the statements of the main propositions and lemmas. Detailed proofs of all results can be found in [20].

Equivalence of Affine Relations. The main result of this section is represented by Lemma 1. which summarizes the conditions for the equivalence of two affine relations. First of all, let us give some basic definitions.

Definition 1. For a clock H, *ValSet*(H) is the set of the values of H indexed by a temporal index, i.e. *ValSet*$(H) = \{H_{i_0}, H_{i_1}, ..., H_{i_t}, ...\}$, with $i_t \in \mathbb{N}, \forall\ t \in \mathbb{N}$ and $i_0 < i_1 < ... < i_t < ...$.

If I is the set of the temporal indices i_t, we will note $ValSet(H) = H_I$.

In the sequel we consider f_H, f_K defined as follows: $f_H : \mathbb{N} \rightarrow \mathbb{N}$, $f_H(t) = nt + max(0, -\varphi)$ and $f_K : \mathbb{N} \rightarrow \mathbb{N}$, $f_K(t) = dt + max(0, \varphi)$.

Definition 2 (Affine relation). $H \overset{(n,\varphi,d)}{\rightarrow} K$ iff there exists a map $g : \mathbb{N} \rightarrow \mathbb{N}$ monotonically increasing on $\mathbb{N}$ and strictly monotonic on $f_K(\mathbb{N}) \cup f_H(\mathbb{N})$, which denotes the value sets of H and K, that is $ValSet(H) = H_{(g \circ f_H)(\mathbb{N})}$ and $ValSet(K) = K_{(g \circ f_K)(\mathbb{N})}$.

In Fig. 4, T' denotes the set of temporal indices for H and K and T the total set of temporal indices. In this example, $g = id_{\mathbb{N}}$ and[3] $f_H(t) = 5t$, $f_K(t) = 7t + 4$ give the position of H and K on the reference clock (we have highlighted (in boldface) the positions corresponding to H and K on the set T).

Definition 3 (Occurrence function). If $H \overset{(n,\varphi,d)}{\rightarrow} K$, we define *the occurrence function* of K with respect to H: [4]

$$\mathcal{A}_H(K) \; : \; \mathbb{N}^* \rightarrow \mathbb{N},$$
$$\mathcal{A}_H(K)(t) = \begin{cases} s+1, & \text{if } \exists s \in \mathbb{N} \text{ maximal, } f_K(s) \leq f_H(t-1) \\ 0, & \text{otherwise} \end{cases}$$

The occurrence function of H with respect to K is similarly defined; it will be noted $\mathcal{A}_K(H)$. By definition, $\mathcal{A}_H(K)(t)$ denotes the number of occurrences of K before the t^{th} occurrence of H, $\forall\, t \in \mathbb{N}^*$.

For two clocks H and K in (n, φ, d)-affine relation, it is possible to give an expression for $\mathcal{A}_H(K)$ (and $\mathcal{A}_K(H)$). This expression will be useful in the sequel.

In the following, we use the symbol $[n]$ for the integer part of n and $a \mid b$ to specify that the positive integer a divides the integer b ($a \not| \, b$ if not).

Proposition 1. *Let $H \overset{(n,\varphi,d)}{\rightarrow} K$. The following equalities hold :*

$$\mathcal{A}_K(H)(t) = \begin{cases} 0, & \text{if } d(t-1) < -\varphi \\ \left[\frac{d(t-1)+\varphi}{n}\right] + 1, & \text{otherwise} \end{cases}$$
$$\mathcal{A}_H(K)(t) = \begin{cases} 0, & \text{if } n(t-1) < \varphi \\ \left[\frac{n(t-1)-\varphi}{d}\right] + 1, & \text{otherwise} \end{cases}$$

In the sequel, the notations $\mathcal{A}_K(H)_{(n,\varphi,d)}$ and $\mathcal{A}_H(K)_{(n,\varphi,d)}$ will be used to emphasize that $\mathcal{A}_K(H)$ and $\mathcal{A}_H(K)$ depend on (n, φ, d).

Definition 4 (Equivalence of affine relations). *The relations $H \overset{(n,\varphi,d)}{\rightarrow} K$ and $H \overset{(n',\varphi',d')}{\rightarrow} K$ are said to be* equivalent *if the following equalities are verified:*

$$\begin{aligned} \mathcal{A}_K(H)_{(n,\varphi,d)} &= \mathcal{A}_K(H)_{(n',\varphi',d')} \\ \mathcal{A}_H(K)_{(n,\varphi,d)} &= \mathcal{A}_H(K)_{(n',\varphi',d')} \end{aligned}$$

[3] $id_{\mathbb{N}}$ is the identity function on $\mathbb{N}$.

[4] $\mathbb{N}^* = \mathbb{N} \backslash \{0\}$.

The following proposition states that any one of the above two equalities is sufficient for the equivalence of two affine relations.

Proposition 2. *For $H \overset{(n,\varphi,d)}{\rightarrow} K$ and $H \overset{(n',\varphi',d')}{\rightarrow} K$ two affine relations between H and K, the following equivalence holds:*

$$\mathcal{A}_K(H)_{(n,\varphi,d)} = \mathcal{A}_K(H)_{(n',\varphi',d')} \Leftrightarrow \mathcal{A}_H(K)_{(n,\varphi,d)} = \mathcal{A}_H(K)_{(n',\varphi',d')}$$

Proof. The idea is that one occurrence function is sufficient to completely define the relative position of H and K; for more details refer to [20]. □

It is necessary to determine the constraints on (n, φ, d) and (n', φ', d') for the implied affine relations to be equivalent. The following lemma gives these conditions. It is a fundamental lemma for the affine clock calculus.

Lemma 1. *Two affine relations $H \overset{(n,\varphi,d)}{\rightarrow} K$ and $H \overset{(n',\varphi',d')}{\rightarrow} K$ are equivalent iff one of the groups of constraints* a)-b) *is verified:*

a) $gcd(n,d) \mid \varphi \;\wedge\; gcd(n',d') \mid \varphi'$
$nd' = n'd$
$n\varphi' = n'\varphi$

b) $gcd(n,d) \not\mid \varphi \;\wedge\; gcd(n',d') \not\mid \varphi'$
$nd' = n'd$
$\left[\frac{dt+\varphi}{n}\right] = \left[\frac{d't+\varphi'}{n'}\right], \forall t \in I\!N, dt + \varphi \geq 0$

with $gcd(n,d)$ the greatest common divisor of n and d.

Proof. From Definition 4 and Propositions 1. and 2., a necessary and sufficient condition for $H \overset{(n,\varphi,d)}{\rightarrow} K$ and $H \overset{(n',\varphi',d')}{\rightarrow} K$ to be equivalent is that the following relation holds: $\left[\frac{dt+\varphi}{n}\right] = \left[\frac{d't+\varphi'}{n'}\right], \forall\, t \in I\!N, dt + \varphi \geq 0$. This relation and the properties of the integer part and greatest common divisor will imply the previous conditions a)-b). For a complete proof, see [20]. □

Figure 7 represents an example of two affine relations which satisfy constraints b) of Lemma 1. (the relative position of H and K is the same for both affine relations).

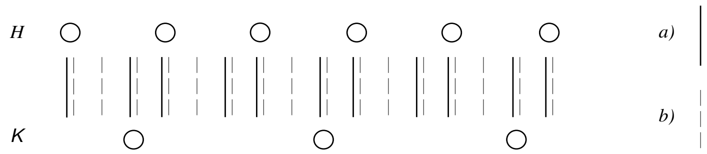

Fig. 7. Examples of equivalent affine relations : *a)* $K = (H)_{(2,1,4)}$, *b)* $K = (H)_{(3,2,6)}$.

The Canonical Form of Affine Relations. As the canonical form concerns the values of n, φ, d, we note the relation $H \overset{(n,\varphi,d)}{\rightarrow} K$ by the 3-tuple (n, φ, d).

We define on the set of 3-tuples a partial order as follows: $(n, \varphi, d) \leq (n', \varphi', d')$ `iff` $n \leq n', \varphi \leq \varphi'$ and $d \leq d'$. The set of all relations (n', φ', d') that satisfy constraints a) or b) of Lemma 1. is the equivalence class of (n, φ, d). We proved (Lemma 2.) that there exists a unique representative of this equivalence class – the smallest affine relation equivalent with (n, φ, d) – we call it the *canonical form* of the affine relation (n, φ, d). The following lemma gives the expression of the canonical form [21] of the affine relation (n, φ, d).

Lemma 2. *For the affine relation* (n, φ, d) *and* $G = gcd(n, d)$*, there exists a canonical form CF defined as follows:*

$$\texttt{a)}\quad G \mid \varphi \Rightarrow CF_{(n,\varphi,d)} = (\tfrac{n}{G}, \tfrac{\varphi}{G}, \tfrac{d}{G})$$

$$\texttt{b)}\quad G \not\mid \varphi \Rightarrow CF_{(n,\varphi,d)} = (2\tfrac{n}{G}, (2\left[\tfrac{\varphi}{G}\right] + 1), 2\tfrac{d}{G})$$

The identity of the equivalence classes for two equivalent affine relations implies the equality of the corresponding canonical forms. The canonical form is very useful for the implementation of the affine clock calculus: in order to check the equivalence of two affine relations, it is not necessary to test the conditions of Lemma 1.. It is sufficient to determine the canonical form of each affine relation and to verify the equality of the corresponding canonical forms.

The Signal **Example in Affine Context.** In this section, we illustrate the affine calculus and the equivalence relation by the specification of the program `EXAMPLE` (from Sect. 3.3) with affine processes. Figure 8 exhibits the transformed program `EXAMPLEAFF`, the associated clock equations and the clock calculus.

The equations $\hat{Y_1} = (\hat{F_1})_{(m+1,0,1)}$ and $\hat{Y_2} = (\hat{F_2})_{(m+1,0,1)}$ have been replaced by their equivalents $\hat{F_1} = (\hat{Y_1})_{(1,0,m+1)}$ and $\hat{F_2} = (\hat{Y_2})_{(1,0,m+1)}$. We can observe that the equivalence $\hat{F_1} = \hat{F_2}$ induced by the synchronization constraints on X can now be verified using the affine clock calculus: $\hat{F_1} = (\hat{Y_1})_{(1,0,m+1)}$ and $\hat{F_2} = (\hat{Y_1})_{(1,0,m+1)}$ specify the same affine relation $(1, 0, m + 1)$ (in its canonical form) between $\hat{F_1}$ and $\hat{Y_1}$, respectively $\hat{F_2}$ and $\hat{Y_1}$.

4 Conclusion

In this paper we presented the ideas for augmenting the Signal compiler facilities in order to improve the specification and validation power of the Signal language. With each Signal program can be associated a system of equations which represent synchronization constraints on clock variables. The Signal compiler resolves these equations and verifies if the control of a program is functionally safe. Control-based real-time applications can thus be specified and validated in Signal, but the modeling and validation of more general signal processing

```
process EXAMPLEAFF={integer m}
   (? integer X;! integer Y)
(1) (| Y1:= OVERREGAFF{m}(X)
(2)  | Y2:= OVERREGAFF{m}(X)
(3)  | Y:= Y1 * Y2  |)
 where
  integer Y1, Y2
  process OVERREGAFF={integer m}
     (? integer FB;! integer N)
    (| F:= m when ^FB
     | ZN:= N $ 1
     | N:= unsample{m+1,0}(F, ZN)
     |)
   where integer ZN init 0
  end
end
```

	a) Clock eqs.	b) Clock calc.
(1)	$\hat{F_1} = \hat{X}$	$\hat{ZY_1} = \hat{Y_1}$
	$\hat{ZY_1} = \hat{Y_1}$	$\hat{Y_2} = \hat{Y_1}$
	$\hat{Y_1} = (\hat{F_1})_{(m+1,0,1)}$	$\hat{Y} = \hat{Y_1}$
	$\hat{Y_1} = (\hat{F_1} \wedge \hat{Y_1}) \vee \hat{ZY_1}$	$\hat{ZY_2} = \hat{Y_1}$
	$\hat{ZY_1} = \hat{Y_1}$	$\hat{F_1} = (\hat{Y_1})_{(1,0,m+1)}$
		$\hat{F_2} = (\hat{Y_1})_{(1,0,m+1)}$
(2)	$\hat{F_2} = \hat{X}$	$\hat{X} = \hat{F_1}$
	$\hat{ZY_2} = \hat{Y_2}$	$\hat{X} = \hat{F_2}$
	$\hat{Y_2} = (\hat{F_2})_{(m+1,0,1)}$	
	$\hat{Y_2} = (\hat{F_2} \wedge \hat{Y_2}) \vee \hat{ZY_2}$	
	$\hat{ZY_2} = \hat{Y_2}$	
(3)	$\hat{Y} = \hat{Y_1} = \hat{Y_2}$	

Fig. 8. The affine clock calculus.

systems necessit to extend the SIGNAL language and the proof facilities of its compiler. Therefore, we have defined in SIGNAL affine transformations on clock variables and the affine relations induced thereby and we have proposed an extension of the SIGNAL compiler in order to verify constraints on clocks obtained by affine transformations. The extended compiler uses two important notions: the equivalence between affine relations and the *canonical form* of an affine relation. In particular, the canonical form represents an important concept which optimizes the implementation of the extended verification system.

With respect to the potential applications of our work, two domains are of particular interest: first, the specification and validation of a larger range of real-time systems using SIGNAL and second, the integration of SIGNAL and other languages like ALPHA in a *co-design* framework. The term co-design has been used to denote the specification, validation and implementation of systems consisting of both hardware and software components [22, 23]. Since SIGNAL and ALPHA have complementary properties, they can be used in a hw/sw co-design framework in which the ALPHA language would specify and synthesize the hardware parts of a system and SIGNAL will describe the software. The SIGNAL-ALPHA cooperation will thus respond to some of the problems existing in co-design, like system specification and synthesis[16, 22].

References

1. Benveniste, A., Berry, G.: Special section on another look at real-time programming. Proc. of the IEEE **79(9)** (September 1991) 1268–1336
2. Benveniste, A., Berry, G.: The Synchronous Approach to Reactive and Real-Time Systems. Proc. of the IEEE **79(9)** (September 1991)

3. Le Guernic, P., Le Borgne, M., Gautier, T., Le Maire, C.: Programming real-time applications with SIGNAL. Technical Report **1446**, INRIA, France (June 1991)
4. Besnard, L.: Compilation de SIGNAL : horloges, dpendances, environnement. PhD thesis, Universit de Rennes I, France (Septembre 1992)
5. Amagbegnon, T., Besnard, L., Le Guernic, P.: Arborescent Canonical Form of Boolean Expressions. Technical Report **2290**, INRIA, France (June 1994)
6. Ackerman, W. B.: Data flow languages. Selected reprints on Dataflow and Reduction Architectures - The Comp. Soc. of IEEE, **(4)** (1987) 179–188
7. Lee, E. A., Messerschmitt, D. G.: Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. IEEE Trans. Computers **C-36** (January 1987) 24–35
8. Lee, E. A., Messerschmitt, D. G.: Synchronous Data Flow. Proc. of the IEEE **75(9)** (September 1987) 1235–1245
9. Lee, E. A.: Consistency in Dataflow Graphs. IEEE Trans. Parall. and Distr. Syst. **2(2)** (April 1991)
10. Karp, R. M., Miller, R. E., Winograd, S.: The organization of computations for uniform recurrence equations. Jrnl. of Assoc. for Computing Machinery **14(3)** (July 1967) 563–590
11. Mauras, Ch.: Dfinition de ALPHA : un langage pour la programmation systolique. Technical Report **482**, INRIA, France **482** (June 1989)
12. Charot, F., Frison, P., Gautrin, E., Lavenier, D., Quinton, P., Raimbault, F., Wagner, Ch.: From equations to hardware. Towards the systematic mapping of algorithms onto parallel architectures. International Journal of Pattern Recognition and Artificial Intelligence (December 1992)
13. Wilde, D. K., Si, O.: Regular array synthesis using ALPHA. Technical Report **829**, IRISA– Rennes, France (May 1994)
14. Kerihuel, A., McConnell, R., Rajopadhye, S.: VSDF : Synchronous Data Flow for VLSI. Technical Report **843**, IRISA–Rennes, France (June 1994)
15. McConnell, R.: Systmes VLSI synchrones priodiques : modlisation et application. PhD thesis, Universit de Rennes I, France (Octobre 1994)
16. Gautier, T., Le Guernic, P., Quinton, P., Rajopadhye, S., Risset, T., Smarandache, I.: Le projet CAIRN : vers la conception d'architectures partir de SIGNAL et ALPHA. Actes Sm. de l'Action Scientifique–Codesign, Meylan, France (June 1996)
17. Bournai, P., Le Guernic, P.: Un environnement graphique pour le langage SIGNAL. Technical Report **2040**, INRIA, France (September 1993)
18. Benveniste, A., Le Guernic, P.: A Denotational Theory of Synchronous Communicating Systems. Technical Report **360**, IRISA–Rennes, France (May 1987)
19. Birkhoff, G.: Lattice Theory. AMS colloquium publications (1973)
20. Smarandache, I., Le Guernic, P.: Affine Transformations in SIGNAL and Their Application in the Specification and Validation of Real-Time Systems. Technical Report INRIA, France (preliminary version)
21. Hullot, J.: Compilation de formes canoniques dans des theories equationnelles. PhD thesis, Paris 11, France (1980)
22. Gupta, R. K., de Micheli, G.: Hardware-Software Cosynthesis for Digital Systems. IEEE Design & Test of Computers (September 1993) 29–41
23. A. Kalavade, A., Lee, E. A.: A Hardware-Software Codesign Methodology for DSP Applications. IEEE Design & Test of Computers (September 1993) 16–28

This article was processed using the LaTeX macro package with LLNCS style