INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Canonical Form for Affine Relations in SIGNAL

Irina Smarandache, Paul Le Guernic

N ° 3097

February 1997

THÈME 2

apport de recherche

ISSN 0249-6399

INRIA
RENNES

A Canonical Form for Affine Relations in SIGNAL

Irina Smarandache, Paul Le Guernic*

Thème 2 — Génie logiciel
et calcul symbolique
Projet EP-ATR

Rapport de recherche n° 3097 — February 1997 — 19 pages

Abstract: In this paper we present affine transformations as an extension of the SIGNAL language for the specification and validation of real-time applications. A SIGNAL program is a system of equations which specify dependencies between program data and synchronization constraints on clock variables. In order to test if a program is functionally safe, the SIGNAL compiler resolves the clock constraints and verifies that the data dependency graph contains no cycles. By means of the new transformations, *affine relations* can be defined between clock variables and it gets necessary to enhance the compiler with facilities for the resolution of synchronization constraints on these clocks. To tackle these constraints, we propose an extension of the compiler based essentially on a canonical form of the affine relations. This extension removes some of the limits of the existing compiler and enlarges the range of applications that can be validated using SIGNAL.

Key-words: SIGNAL, real-time languages, clock calculus, affine transformations, canonical form

(Résumé : tsvp)

Paper accepted to the Fourth International AMAST Workshop on Real-Time Systems, Concurrent and Distributed Software (ARTS'97).

* Irina.Smarandache@irisa.fr, Paul.Leguernic@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 02 99 84 71 00 – Télécopie : (33) 02 99 84 71 71

Une Forme Canonique pour des Relations Affines en SIGNAL

Résumé : Cet article présente les transformations affines comme une extension de SIGNAL pour la spécification et la validation d'applications temps-réel. Un programme SIGNAL est un système d'équations qui représente des dépendances entre des données et des contraintes de synchronisation entre des variables d'horloges. Afin de tester si un programme est correct, le compilateur de SIGNAL résoud les contraintes de synchronisation et vérifie que le graphe des dépendances de données est acyclique. Les nouvelles transformations permettent de définir des *relations affines* entre des variables d'horloges et il s'impose de fournir les outils nécessaires pour la vérification des contraintes de synchronisation entre ces horloges. Une extension du compilateur basée sur une forme canonique des relations affines a été proposée : elle élargi la capacité de preuve du compilateur actuel et augmente le nombre d'applications qui peuvent être spécifiées et validées avec SIGNAL.

Mots-clé : SIGNAL, langages temps-réel, calcul d'horloges, transformations affines, forme canonique

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 (communications and calculi) in a system are considered to be 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, 4] 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 taking into consideration the physical properties (time, resources) 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 reposes on boolean equation resolution methods which appeared sufficient for the validation of real-time systems where control is an important aspect [5, 6]. Most of the real-time signal processing applications contain important numeric computations; in order to treat such applications, we define *affine transformations* on clock variables and we 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 [7] has been considered an intuitive and convenient paradigm for describing digital signal processing systems, i.e. systems with important numeric treatments. 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. In [8, 9] *Synchronous Data Flow* (SDF) graphs are particular data flow graphs in which the amount of data produced and consumed by a data flow node is specified a priori for each input and output. This property makes SDF graphs good candidates for verifying statically the existence of sequential and parallel schedules: consistency properties of an SDF graph can be checked at compile time. Moreover, in [10] the SDF model is enlarged in order to tackle with a larger range of signal processing 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 [11]. Reduced dependence graphs have optimal schedules which can be found at compile time. However, their application domain is more restricted than for SDFs: reduced dependence graphs are generally used to describe regular iterative algorithms, which can be systematically 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 [12]. 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 [13, 14].

Initially designed for use in the specification and validation of real-time signal processing systems, SIGNAL is based currently on a control-oriented formalism [6]. However, due to its data flow flavor, SIGNAL has been used in the specification of periodic synchronous VLSI systems modeled by SDF graphs [15, 16]. The modeling and simulation of SDF graphs in SIGNAL have 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 by external C processes. In order to obviate these problems and due to the complementary properties of the languages SIGNAL and ALPHA, we have projected a hybrid framework [17] in which SIGNAL and ALPHA 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. We also present the additional proof system and its main properties. 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 [4]. The SIGNAL compiler resolves these systems and produces executable code whenever the program is correct.

A variety of tools exist around the SIGNAL language and its compiler. They constitute the SIGNAL environment. There are a *Graphical User Interface* for high-level system design [18], C and Fortran code generators to produce code for functional simulators, an intermediate format generator for the access to formal verification tools [19] and a VHDL code generator to access hardware synthesis tools [20]. 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 elementary data dependencies graphs associated with each SIGNAL equation [5]. The construction of the SDG proceeds in parallel with the verification if the program is deterministic, has no cycles, if no constraints are imposed on its inputs 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 an integer (positive) temporal index. The values of the index domain define the logical instants where the signal is present. A signal can be present or absent at a given logical instant from the positive integer domain. We define the *clock* of a signal as the set of instants where the signal is present (and has a value). 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 [21, 22]. We present briefly the elementary processes and the extensions of SIGNAL in Table 1 (for a more detailed presentation refer to [4, 19]). The clock equations associated with the statements are shown in the right-hand column of Table 1.

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* (similarly $[\neg C]$ is the clock of the *false* values of C). The classical operations on sets $(\wedge, \vee, \setminus)$ can be performed on clocks. The inclusion relation between sets defines a partial order on the set of clocks; moreover the clock *poset*[1] is a boolean lattice [23]. In order to resolve the clock equations of a SIGNAL program, we use, in addition to the well-known properties of the boolean lattice, the special relation that exists between $[C]$ and $[\neg C]$, given by the following *partition* equations:

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

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

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

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

Table 1: **Basic SIGNAL constructs and extensions** .

2.2 The SIGNAL compiler

The SIGNAL compiler resolves the clock equations using an algorithm that synthesizes the most frequent clock *(master clock)* of a SIGNAL program, if such a clock exists. If not, the result of the clock calculus is a set of clocks which are local maxima for the SIGNAL program. If there is a unique master clock, 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. Otherwise, SIGNAL processes are said to be *exochronous*. They do not dispose of all necessary information for their execution, i.e. the synchronizations of variables are not completely specified.

The resolution method uses a triangularization strategy and a tree representation in order to synthesize the most frequent clock of a program and to verify the synchronization constraints on clocks [6]. The idea is to derive definitions for all clocks in terms of already defined ones and to verify the equivalence between multiple definitions. The properties of a boolean lattice and the partition equations are rewriting rules largely 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) $\hat{\mathtt{N}} = \hat{\mathtt{FB}} \vee \hat{\mathtt{ZN}}$

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

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

Figure 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`; ligne (4) of the SIGNAL program states that, for each logical instant, the value for `N` is equal to `FB`, if the signal `FB` is present, or to $(\mathtt{ZN} - 1)$, if `FB` is absent. The input signal `FB` is accepted only at the *true* value instants of $(\mathtt{ZN} \leq 1)$ (see ligne (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 $\hat{\mathtt{N}}$ 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	$\cdots$

Figure 2: **Timing diagram.** The t_i are logical instants. Shown are the values of the signals FB, ZN and N, where '-' means that a signal is absent at a given instant.

2.4 Introduction to affine transformations on clocks

The facility for expressing oversampling is an important feature of the SIGNAL language. It can be used to describe temporal properties of processes that, although considered as instantaneous, exhibit a clear succession of logical instants (e.g. pipelining in algorithms for matrices, vectors and generally regular data).

The current version of SIGNAL has limited facilities for manipulating regular data. It is possible to multiply elementary operations by declaring *arrays of processes* . Thus, matrix or vector operations can be expressed in SIGNAL, however only a restricted range of algorithms can be specified in this way. Nevertheless, SIGNAL offers the alternative 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 predefined or user-defined C function.

Because of its facilities for expressing and manipulating regular data using the formalism of affine recurrence equations, the ALPHA language [12, 14] can be used in cooperation with SIGNAL for the specification, validation and implementation of real-time signal processing systems. ALPHA processes are represented as external processes in SIGNAL. Two situations may occur [17]:

1. The external ALPHA process generates a SIGNAL function (i.e. the inputs and outputs are synchronous). In this case the execution of the algorithm is considered instantaneous. Figure 3 a) shows the example of the multiplication of two synchronous matrices A and B ($C = A \cdot B$);
2. The external ALPHA process generates a SIGNAL process with asynchronous inputs and outputs. In this case, the high-level specification/architecture transformations existing in the ALPHA synthesis environment induce a temporal index which must be taken into consideration in SIGNAL. In Fig. 3 b), the result of the multiplication of A and B is no longer synchronous with A and B ($C = A \cdot B$).

If C is not synchronous with the input matrices A and B, the regularity of manipulated data is preserved by the relative position of the clocks $\hat{C}$ and $\hat{A}$, respectively $\hat{B}$: the output C is present at every 4^{th} instant after the presence of the input signals A and B (in Fig. 3 b) below $1, 2, 3$ are intermediate instants where partial computations on the output C are performed). This is an example where a clock $\hat{C}$ has been obtained by an *affine transformation* on the clock $\hat{A}$ (see Sect. 3.1 for a formal definition of affine transformations). $\hat{C}$ and $\hat{A}$ are said to be in *affine relation*.

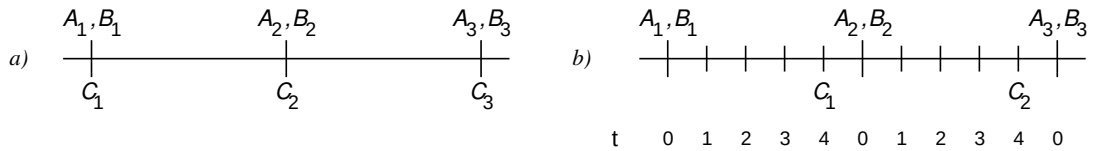

Figure 3: **Affine transformations in SIGNAL: example.**

3 Affine transformations in SIGNAL

Affine transformations on clock variables generate affine relations whose properties are used in order 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 then counting on this fictional set each d^{th} instant, starting with the φ^{th} instant (see Figure 4). The affine transformation of parameters (n, φ, d) defines a (n, φ, d)-affine relation between the clocks H and K (H and K will also be said (n, φ, d)-related). Moreover, the affine relation is oriented in the sense the affine transformation is performed, i.e. *from H to K*. We will note $H \overset{(n,\varphi,d)}{\rightarrow} K$ this affine relation in order to point out the orientation. A $(d, -\varphi, n)$-affine relation exists also between K and H: $K \overset{(d,-\varphi,n)}{\rightarrow} H$ (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 has been represented: its instants are vertical lines, while H and K instants are presented using circles. The distance between two successive instants has no signification in the semantic of SIGNAL; only the succession and relative position of instants are important. Thus, we have introduced the support (reference) clock in order to point out the way the affine transformation is performed; any other clock can be used instead with the condition that the position of instants of H with respect to the instants of K remains unchanged. This relative position defines the $(5, 4, 7)$-affine relation between H and K ($H \overset{(5,4,7)}{\rightarrow} K$).

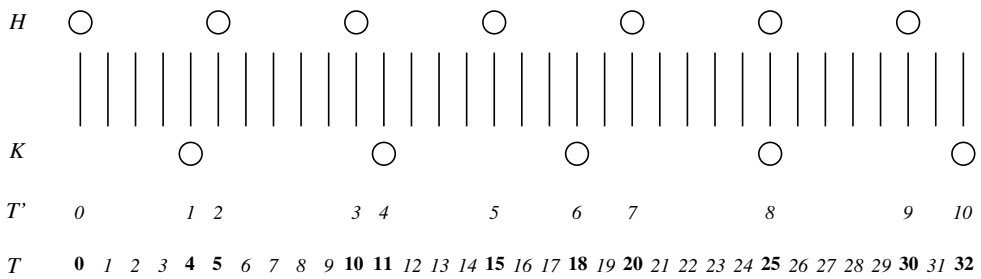

Figure 4: **Example:** H and K are in $(5, 4, 7)$-affine relation.

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 Figure 5.

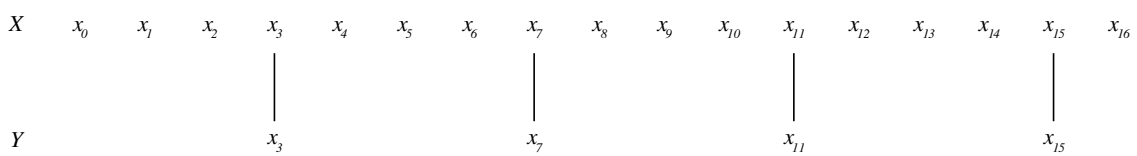

Figure 5: `Y := sample{ 3,4 }( X )`. `Y` is present at every 4^{th} instant, starting with instant 3.

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

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

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

Finally, `Y := unsample{ n,φ }( X, Z)`, with $n > 0$ 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    |)
```

Table 2 summarizes the affine processes and their associated clock equations. For an arbitrary 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. H and K will be said to be (n, φ, d)-related if $K = (H)_{(n,\varphi,d)}$.

Synchronization constraints on clocks obtained by affine transformations cannot be verified with the current compiler. It is thus necessary to extend the clock calculus with new proof facilities based on the properties of affine relations. We present in the sequel *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.

3.3.1 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, ($op \in \{\wedge, \vee, \backslash\}$), a right-to-left orientation is performed in order to consider the formula H_1 *op* H_2 as the definition of the clock variable 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. 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 present the rewriting rules used during the compilation are deduced from the axioms of the boolean lattice and from the properties of clocks defined as down-samplings by boolean conditions, i.e. the partition equations on $[C]$ and $[\neg C]$ (see 2.1).

	Syntax	Description	Clock calculus
SIGNAL constructs			
	`Y := sample{ φ,d }( X )`	down-sampling by the affine function $\lambda t(dt + \varphi)$	$\hat{Y} = [\hat{X}]_{(\varphi,n)}$
SIGNAL extensions			
	`clk_affine{ n,φ,d }( X,Y )`	affine relation between clocks of `X` and `Y`	$\hat{Y} = (\hat{X})_{(n,\varphi,d)}$
	`Y := unsample{ n,φ }( X, Z)`	affine oversampling using `X` and `Z`	$\hat{Y} = (\hat{X})_{(n,\varphi,1)}$ $\hat{Y} = \hat{Z}$

Table 2: **New SIGNAL constructs and extensions** .

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 [5].

3.3.2 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 Figure 1: in the process `OVERREG`, for each input value `FB`, `(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
```

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

Figure 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 Figure 6 (a) the clock equations associated with each line after the instantiation. After the clock calculus the transformed system is presented in Figure 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 $\{\wedge, \vee, \backslash\}$ operators. In order to tackle equations on clocks obtained by affine transformations, we need new rewriting rules based on the properties of affine relations.

3.4.1 Resolution strategy

The resolution algorithm uses two important notions: the equivalence between affine relations (explained in section 3.4.2) and the canonical form of affine relation (detailed in section 3.4.3). The resolution strategy is similar to the triangularization presented above. Let us summarize the main steps in the case of the newly defined clocks in order to point out the similarities and differences with the existing method. Each equation $H = \Psi_H$, with Ψ_H a formula containing at most two 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. For this orientation to be possible, we suppose that the clocks H_1 and H_2 have already been defined (the principle of the triangularization). 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 (n, φ, d)-related 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 $H_1 \overset{(n_1, \varphi_1, d_1)}{\rightarrow} H$ equivalent with $H_1 \overset{(n'_1, \varphi'_1, d'_1)}{\rightarrow} H$ and/or $H_2 \overset{(n_2, \varphi_2, d_2)}{\rightarrow} H$ equivalent with $H_2 \overset{(n'_2, \varphi'_2, d'_2)}{\rightarrow} H$.

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 $H \overset{(n, \varphi, d)}{\rightarrow} H$ and $H \overset{(1, 0, 1)}{\rightarrow} H$ 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 $H_1 \overset{(n, \varphi, d)}{\rightarrow} H$ and $H_1 \overset{(n', \varphi', d')}{\rightarrow} H$ has to be checked.

The affine clock calculus illustrates an important point: a clock variable can be defined by two different affine transformations with no equivalence condition on the corresponding affine relations. Consider H defined as follows:

$$H = (H_1)_{(n_1, \varphi_1, d_1)} \qquad H = (H_2)_{(n_2, \varphi_2, d_2)}$$

If no constraints exist on clocks H_1 and H_2, an orientation can be performed on each of these equations. The formulas $(H_1)_{(n_1, \varphi_1, d_1)}$ and $(H_2)_{(n_2, \varphi_2, d_2)}$ are both *definitions* of H. In fact, these formulas state that H is affinely related with both H_1 and H_2, which is perfectly legal.

Consider now the following definitions for H:

$$H = (H')_{(n_1, \varphi_1, d_1)} \qquad H = (H')_{(n_2, \varphi_2, d_2)}$$

In this case, the right-hand parts of the equations must exhibit the same relation between H and H'. Thus, the relations $H' \overset{(n_1, \varphi_1, d_1)}{\rightarrow} H$ and $H' \overset{(n_2, \varphi_2, d_2)}{\rightarrow} H$ must be equivalent.

In this paper, we outline the theoretical support and proofs concerning the equivalence relation existing on the set of affine relations and the *canonical form* of affine relations, two important properties used as support of the resolution algorithm presented above.

3.4.2 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 strictly monotonic 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 $f_H(\mathbb{N}) \cup f_K(\mathbb{N})$, 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 Figure 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 (the instants of the reference clock are represented by vertical lines; we have highlighted – in boldface – the positions corresponding to H and K on the set T).

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

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

$$\begin{array}{l}\mathcal{A}_K(H) \ : \ \mathbb{N}^* \ \rightarrow \ \mathbb{N}, \\ \mathcal{A}_K(H)(t) = \left\{ \begin{array}{ll} s+1, & \text{if } \exists s \in \mathbb{N} \text{ maximal, } f_H(s) \leq f_K(t-1) \\ 0, & \text{otherwise} \end{array} \right.\end{array}$$

The occurrence function of K with respect to H is similarly defined; it will be noted $\mathcal{A}_H(K)$.

By definition, $\mathcal{A}_K(H)(t)$ denotes the number of occurrences of H before the t^{th} occurrence of K, $\forall\ t \in \mathbb{N}^*$ ($\mathbb{N}^* = \mathbb{N} \backslash \{0\}$).

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

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

Proposition 1 *Let* $H \overset{(n,\varphi,d)}{\rightarrow} K$*. We have the following expressions for* $\mathcal{A}_K(H)$ *and* $\mathcal{A}_H(K)$*:*

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

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

Proof :

$$\begin{array}{rcl} \mathcal{A}_K(H)(t) & = & \left\{ \begin{array}{ll} s+1, & \text{if } \exists s \in \mathbb{N} \text{ maximal, } f_H(s) \leq f_K(t-1) \\ 0, & \text{otherwise} \end{array} \right. \\ & = & \left\{ \begin{array}{ll} s+1, & \text{if } \exists s \in \mathbb{N} \text{ maximal, } ns \leq d(t-1)+\varphi \\ 0, & \text{otherwise} \end{array} \right. \\ & = & \left\{ \begin{array}{ll} s+1, & \text{if } \exists s \in \mathbb{N} \text{ maximal, } s \leq \frac{d(t-1)+\varphi}{n} \\ 0, & \text{otherwise} \end{array} \right. \\ & = & \left\{ \begin{array}{ll} 0, & \text{if } d(t-1) < -\varphi \\ \left[\frac{d(t-1)+\varphi}{n}\right] + 1, & \text{otherwise} \end{array} \right. \end{array}$$

□

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

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

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:*

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

Proof : We just sketch here the idea of the proof; for more details see Appendix A.

The idea is that one occurrence function is sufficient to completely define the relative position of H *and* K*. Therefore,* $\mathcal{A}_K(H)_{(n,\varphi,d)} = \mathcal{A}_K(H)_{(n',\varphi',d')}$ *ensures the same relative position of* H *and* K *for both affine relations* $H \overset{(n,\varphi,d)}{\rightarrow} K$ *and* $H \overset{(n',\varphi',d')}{\rightarrow} K$*. Thus,* $\mathcal{A}_H(K)_{(n,\varphi,d)} = \mathcal{A}_H(K)_{(n',\varphi',d')}$ *also holds.* □

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:*

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

$$b) \quad (\ gcd(n,d) \not\mid \varphi) \ \wedge \ (\ gcd(n',d') \not\mid \varphi') \qquad nd' = n'd \qquad \left\lceil \frac{dt+\varphi}{n} \right\rceil = \left\lceil \frac{d't+\varphi'}{n'} \right\rceil, \forall\ t \in \mathbb{N},\ \ dt+\varphi \geq 0$$

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

1. *From Definition 4 and Propositions 1, 2, a necessary 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:* $\left\lceil \frac{dt+\varphi}{n} \right\rceil = \left\lceil \frac{d't+\varphi'}{n'} \right\rceil, \forall\ t \in \mathbb{N},\ \ dt+\varphi \geq 0$.

   Using $[n] \leq n < [n]+1$ *and the previous relation, it follows:*

$$\begin{cases} \frac{dt+\varphi-n}{d't+\varphi'} < \frac{n}{n'}, & \forall\ t \in \mathbb{N} \\ \frac{d't+\varphi'-n'}{dt+\varphi} < \frac{n'}{n}, & \forall\ t \in \mathbb{N} \end{cases} \Leftrightarrow \begin{cases} \lim_{t\to+\infty} \frac{dt+\varphi-n}{d't+\varphi'} \leq \frac{n}{n'} \\ \lim_{t\to+\infty} \frac{d't+\varphi'-n'}{dt+\varphi} \leq \frac{n'}{n} \end{cases} \Leftrightarrow \begin{cases} \frac{d}{d'} \leq \frac{n}{n'} \\ \frac{d'}{d} \leq \frac{n'}{n} \end{cases} \Leftrightarrow \quad nd' = n'd$$

   Similarly to f_H *and* f_K, *we consider* f'_H, f'_K *strictly monotonic defined as follows:* $f'_H : \mathbb{N} \to \mathbb{N}$, $f'_H(t) = n't + max(0, -\varphi')$ *and* $f'_K : \mathbb{N} \to \mathbb{N}$, $f'_K(t) = d't + max(0, \varphi')$.

   H *and* K *are in affine relation,* $H \overset{(n,\varphi,d)}{\rightarrow} K$ *and* $H \overset{(n',\varphi',d')}{\rightarrow} K$, *and from the definition of an affine relation (Definition 2) there exist two maps* $g, g' : \mathbb{N} \to \mathbb{N}$ *monotonically increasing on* $\mathbb{N}$ *and strictly monotonic on* $f_H(\mathbb{N}) \cup f_K(\mathbb{N})$, *respectively* $f'_H(\mathbb{N}) \cup f'_K(\mathbb{N})$, *which denote the value sets of* H *and* K, *that is* $ValSet(H) = H_{(g \circ f_H)(\mathbb{N})} = H_{(g' \circ f'_H)(\mathbb{N})}$ *and* $ValSet(K) = K_{(g \circ f_K)(\mathbb{N})} = K_{(g' \circ f'_K)(\mathbb{N})}$.

   Therefore $(g \circ f_H)(\mathbb{N}) = (g' \circ f'_H)(\mathbb{N})$ *and* $(g \circ f_K)(\mathbb{N}) = (g' \circ f'_K)(\mathbb{N})$.

   With the previous equalities, we have the following equivalences :

$$\begin{aligned} gcd(n,d) \mid \varphi \quad &\Leftrightarrow \quad \text{the Diophantine equation } ns = dt + \varphi \text{ has at least one solution } (s,t) \in \mathbb{N}^2 \\ &\Leftrightarrow \quad \exists (s,t) \in \mathbb{N}^2,\ f_H(s) = f_K(t) \\ &\Leftrightarrow \quad \exists (s,t) \in \mathbb{N}^2,\ (g \circ f_H)(s) = (g \circ f_K)(t) \\ &\Leftrightarrow \quad \exists (s,t) \in \mathbb{N}^2,\ (g' \circ f'_H)(s) = (g' \circ f'_K)(t) \\ &\Leftrightarrow \quad \exists (s,t) \in \mathbb{N}^2,\ f'_H(s) = f'_K(t) \\ &\Leftrightarrow \quad \text{the Diophantine equation } n's = d't + \varphi' \text{ has at least one solution } (s,t) \in \mathbb{N}^2 \\ &\Leftrightarrow \quad gcd(n',d') \mid \varphi' \end{aligned}$$

   In case $gcd(n,d) \mid \varphi$, *the Diophantine equations* $ns = dt + \varphi$ *and* $n's = d't + \varphi'$ *have a common solution* $(s_0, t_0) \in \mathbb{N}^2$, *i.e.* $ns_0 = dt_0 + \varphi$ *and* $n's_0 = d't_0 + \varphi'$. *From the first equation,* $s_0 = \frac{dt_0+\varphi}{n}$; *by replacing this expression in the second equation, we obtain :*

$$\frac{dt_0+\varphi}{n} = \frac{d't_0+\varphi'}{n'} \ \Rightarrow\ dn't_0 + \varphi n' = nd't_0 + \varphi' n \ \overset{nd'=dn'}{\Rightarrow}\ n\varphi' = n'\varphi$$

2. *Moreover, the relation* $\left\lceil \frac{dt+\varphi}{n} \right\rceil = \left\lceil \frac{d't+\varphi'}{n'} \right\rceil, \forall\ t \in \mathbb{N},\ \ dt+\varphi \geq 0$ *is a sufficient condition for the affine relations* $H \overset{(n,\varphi,d)}{\rightarrow} K$ *and* $H \overset{(n',\varphi',d')}{\rightarrow} K$ *to be equivalent.*

   $\forall\ t \in \mathbb{N},\ dt+\varphi \geq 0 \ \Rightarrow\ \left\lceil \frac{dt+\varphi}{n} \right\rceil \geq 0 \ \Rightarrow\ \left\lceil \frac{d't+\varphi'}{n'} \right\rceil \geq 0 \ \Rightarrow\ d't+\varphi' \geq 0$ *and reciprocally,* $\forall\ t \in \mathbb{N},\ d't+\varphi' \geq 0 \ \Rightarrow\ dt+\varphi \geq 0$, *i.e. the following equivalence holds :* $\forall t \in \mathbb{N}^*,\ (\ d(t-1)+\varphi \geq 0 \ \Leftrightarrow\ d'(t-1)+\varphi' \geq 0\ )$.

   Therefore, from the previous results and Proposition 1, it gets $\mathcal{A}_K(H)_{(n,\varphi,d)} = \mathcal{A}_K(H)_{(n',\varphi',d')}$, *which implies further, by Proposition 2,* $\mathcal{A}_H(K)_{(n,\varphi,d)} = \mathcal{A}_H(K)_{(n',\varphi',d')}$.

   It results then that the affine relations $H \overset{(n,\varphi,d)}{\rightarrow} K$ *and* $H \overset{(n',\varphi',d')}{\rightarrow} K$ *are equivalent.* □

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).

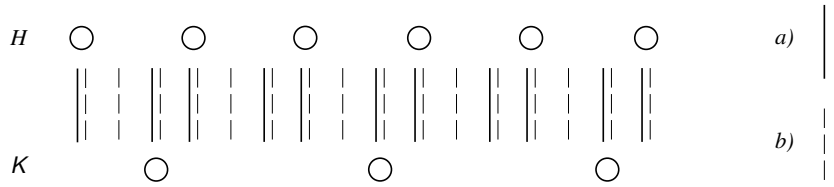

Figure 7: **Examples of equivalent affine relations.** *a)* $K = (H)_{(2,1,4)}$, *b)* $K = (H)_{(3,2,6)}$.

3.4.3 The canonical form of affine relations

As the canonical form concerns the values of n, φ and d, we will 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 will prove that there exists a unique representative of this equivalence class – the smallest affine relation equivalent with the (n, φ, d) relation – we call it the *canonical form* of the affine relation (n, φ, d). The following lemma gives the expression of the canonical form [24] 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:*

$$\begin{array}{lrcl} \texttt{a)} & G \mid \varphi & \Rightarrow & CF_{(n,\varphi,d)} = (\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G}) \\ \texttt{b)} & G \not\mid \varphi & \Rightarrow & CF_{(n,\varphi,d)} = (2\frac{n}{G}, (2\lfloor\frac{\varphi}{G}\rfloor + 1), 2\frac{d}{G}) \end{array}$$

Proof : See Appendix B. □

The identity of the equivalence classes for two equivalent affine relations implies the equality of the corresponding minimal elements, i.e. the canonical forms of the affine relations are identical. The canonical form is very useful for the implementation of the affine transformations in SIGNAL : 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.

3.4.4 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 section 3.3.2) using affine processes. Figure 8 exhibits the transformed program `EXAMPLEAFF`, the associated clock equations and the clock calculus.

The equations $\hat{Y_1} = (F\hat{B}B_1)_{(m+1,0,1)}$ and $\hat{Y_2} = (F\hat{B}B_2)_{(m+1,0,1)}$ have been been replaced by their equivalents $F\hat{B}B_1 = (\hat{Y_1})_{(1,0,m+1)}$ and $F\hat{B}B_2 = (\hat{Y_2})_{(1,0,m+1)}$. We can observe that the equivalence $F\hat{B}B_1 = F\hat{B}B_2$ induced by the synchronization constraints on X can now be verified using the affine clock calculus: $F\hat{B}B_1 = (\hat{Y_1})_{(1,0,m+1)}$ and $F\hat{B}B_2 = (\hat{Y_1})_{(1,0,m+1)}$specify the same affine relation $(1, 0, m+1)$ (in its canonical form) between $F\hat{B}B_1$ and $\hat{Y_1}$, respectively $F\hat{B}B_2$ 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 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

```
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 )
    (| FBB := m when ^FB
     | ZN := N $ 1
     | N := unsample{m+1, 0} (FBB, ZN)
     |)
   where integer ZN init 0
  end
end
```

Clock equations (a)

(1)
$\hat{\mathrm{FBB}}_1 = \hat{\mathrm{X}}$
$\mathrm{Z}\hat{\mathrm{Y}}_1 = \hat{\mathrm{Y}}_1$
$\hat{\mathrm{Y}}_1 = (\hat{\mathrm{FBB}}_1)_{(m+1,0,1)}$
$\hat{\mathrm{Y}}_1 = (\hat{\mathrm{FBB}}_1 \wedge \hat{\mathrm{Y}}_1) \vee \mathrm{Z}\hat{\mathrm{Y}}_1$
$\mathrm{Z}\hat{\mathrm{Y}}_1 = \hat{\mathrm{Y}}_1$

(2)
$\hat{\mathrm{FBB}}_2 = \hat{\mathrm{X}}$
$\mathrm{Z}\hat{\mathrm{Y}}_2 = \hat{\mathrm{Y}}_2$
$\hat{\mathrm{Y}}_2 = (\hat{\mathrm{FBB}}_2)_{(m+1,0,1)}$
$\hat{\mathrm{Y}}_2 = (\hat{\mathrm{FBB}}_2 \wedge \hat{\mathrm{Y}}_2) \vee \mathrm{Z}\hat{\mathrm{Y}}_2$
$\mathrm{Z}\hat{\mathrm{Y}}_2 = \hat{\mathrm{Y}}_2$

(3)
$\hat{\mathrm{Y}} = \hat{\mathrm{Y}}_1 = \hat{\mathrm{Y}}_2$

Clock calculus (b)

$\mathrm{Z}\hat{\mathrm{Y}}_1 = \hat{\mathrm{Y}}_1$
$\hat{\mathrm{Y}}_2 = \hat{\mathrm{Y}}_1$
$\hat{\mathrm{Y}} = \hat{\mathrm{Y}}_1$
$\mathrm{Z}\hat{\mathrm{Y}}_2 = \hat{\mathrm{Y}}_1$
$\hat{\mathrm{FBB}}_1 = (\hat{\mathrm{Y}}_1)_{(1,0,m+1)}$
$\hat{\mathrm{FBB}}_2 = (\hat{\mathrm{Y}}_1)_{(1,0,m+1)}$
$\hat{\mathrm{X}} = \hat{\mathrm{FBB}}_1$
$\hat{\mathrm{X}} = \hat{\mathrm{FBB}}_2$

Figure 8: **The affine clock calculus.**

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 [26, 27]. Since SIGNAL and ALPHA have complementary properties, they can be used in such a hw/sw co-design framework; they would thus respond to some of the problems existing in co-design, like system specification and synthesis[17, 26].

We are currently working on an extension of the proposed methods in order to treat composition of affine transformations. We are also investigating the influence of the affine clock calculus on the implementation of regular data computations in SIGNAL.

References

[1] A. Benveniste, G. Berry. *Special section on another look at real-time programming*, Proc. of the IEEE, 79(9):1268-1336, Sept. 1991.

[2] A. Benveniste, G. Berry. *The Synchronous Approach to Reactive and Real-Time Systems*, Proc. of the IEEE, 79(9), Sept. 1991.

[3] P. Le Guernic, T. Gautier. *Data-Flow to von Neumann: the* SIGNAL *Approach*, Advanced Topics in Data-Flow Computing, ed. Jean-Luc Gaudiot - Lubomir Bic, 413-438, 1991.

[4] P. Le Guernic, M. Le Borgne, T. Gautier, C. La Maire. *Programming real time applications with* SIGNAL, Technical Report no. 1446, IRISA, Rennes, France, June 1991.

[5] L. Besnard. *Compilation de* SIGNAL *: horloges, dépendances, environnement*, PhD thesis, Université de Rennes I, France, Sept. 1992.

[6] T. Amagbegnon, L. Besnard, P. Le Guernic. *Arborescent Canonical Form of Boolean Expressions*, Technical Report no. 2290, INRIA, France, June 1994.

[7] W. B. Ackerman. *Data flow languages*, Selected reprints on Dataflow and Reduction Architectures - The computer society of IEEE, ed. S. S. Thakkar, (4): 179-188, 1987.

[8] E. A. Lee, D. G. Messerschmitt. *Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing*, IEEE Transactions on Computers, C-36: 24-35, Jan. 1987.

[9] E. A. Lee, D. G. Messerschmitt. *Synchronous Data Flow*, Proc. of the IEEE, 75(9): 1235-1245, Sept. 1987.

[10] E. A. Lee. *Consistency in Dataflow Graphs*, IEEE Transactions on Parallel and Distributed Systems, 2(2), April 1991.

[11] R. M. Karp, R. E. Miller, S. Winograd. *The organization of computations for uniform recurrence equations*, Journal of the Association for Computing Machinery, 14(3): 563-590, July 1967.

[12] Ch. Mauras, *Définition de* ALPHA *: un langage pour la programmation systolique*, Technical Report no. 482, INRIA, France, June 1989.

[13] F. Charot, P. Frison, E. Gautrin, D. Lavenier, P. Quinton, F. Raimbault, Ch. Wagner. *From equations to hardware. Towards the systematic mapping of algorithms onto parallel architectures*, International Journal of Pattern Recognition and Artificial Intelligence, Dec. 1992.

[14] D. K. Wilde, O. Sié. *Regular array synthesis using* ALPHA, Technical Report no. 829, IRISA, Rennes, France, May 1994.

[15] A. Kerihuel, R. McConnell, S. Rajopadhye. VSDF*: Synchronous Data Flow for* VLSI, Technical Report no. 843, IRISA, Rennes, France, June 1994.

[16] R. McConnell. *Systèmes* VLSI *synchrones périodiques : modélisation et application*, PhD thesis, Université de Rennes I, France, Oct. 1994.

[17] T. Gautier, P. Le Guernic, P. Quinton, S. Rajopadhye, T. Risset, I. Smarandache. *Le projet* CAIRN *: vers la conception d'architectures à partir de* SIGNAL *et* ALPHA, Actes des Séminaires de l'Action Scientifique - Codesign, Meylan, France, June 1996.

[18] P. Bournai, P. Le Guernic. *Un environnement graphique pour le langage* SIGNAL, Technical Report no. 2040, INRIA, France, Sept. 1993.

[19] T. P. Amagbegnon, P. Le Guernic, H. Marchand, E. Rutten. *The* SIGNAL *dataflow methodology applied to a production cell*, Lecture Notes in Computer Science LNCS,Springer Verlag, Jan. 1995.

[20] M. Belhadj. *Conception d'architectures en utilisant* SIGNAL *et* VHDL, PhD thesis, Université de Rennes I, France, Dec. 1994.

[21] A. Benveniste, P. Le Guernic. *A Denotational Theory of Synchronous Communicating Systems*, Technical Report no. 360, IRISA, Rennes, France, May 1987.

[22] A. Benveniste, B. Le Goff, P. Le Guernic. *Hybrid Dynamical Systems Theory and the Language* SIGNAL, Technical Report no. 838, IRISA, Rennes, France, April 1988.

[23] G. Birkhoff. *Lattice Theory*, AMS colloquium publications, 1973.

[24] J. Hullot. *Compilation de formes canoniques dans des theories equationnelles*, PhD thesis, Paris 11, France, 1980.

[25] U. Banerjee. *Loop Transformations for Restructuring Compilers - The Foundations*, Kluwer Academic Publishers, 1993.

[26] R. K. Gupta, G. de Micheli. *Hardware-Software Cosynthesis for Digital Systems*, IEEE Design & Test of Computers, 29-41, Sept. 1993.

[27] A. Kalavade, E. A. Lee. *A Hardware-Software Codesign Methodology for DSP Applications*, IEEE Design & Test of Computers, 16-28, Sept. 1993.

A Proof of Proposition 2 of section 3.4.2

First of all, we fix some notations to be used in the sequel :

$$f_H, f_K : \mathbb{N} \rightarrow \mathbb{N},\ f_H(t) = nt + max(0, -\varphi),\ f_K(t) = dt + max(0, \varphi)$$
$$f'_H, f'_K : \mathbb{N} \rightarrow \mathbb{N},\ f'_H(t) = n't + max(0, -\varphi'),\ f'_K(t) = d't + max(0, \varphi')$$

$$\forall\ t \in \mathbb{N}^*,\ A_t = \{s \in \mathbb{N} \ /\ f_H(s) \leq f_K(t-1)\}\ A'_t = \{s \in \mathbb{N} \ /\ f'_H(s) \leq f'_K(t-1)\}$$
$$\forall\ t \in \mathbb{N}^*,\ B_t = \{s \in \mathbb{N} \ /\ f_K(s) \leq f_H(t-1)\}\ B'_t = \{s \in \mathbb{N} \ /\ f'_K(s) \leq f'_H(t-1)\}$$

A_t, A'_t, B_t, B'_t have been chosen to verify $\mid A_t \mid$ [4] $= \mathcal{A}_K(H)_{(n,\varphi,d)}$, $\mid A'_t \mid = \mathcal{A}_K(H)_{(n',\varphi',d')}$, $\mid B_t \mid = \mathcal{A}_H(K)_{(n,\varphi,d)}$, $\mid B'_t \mid = \mathcal{A}_H(K)_{(n',\varphi',d')}$.

Using the expressions of f_H, f_K and f'_H, f'_K we can write further :

$$A_t = \{s \in \mathbb{N} \ /\ ns \leq d(t-1) + \varphi\} \qquad A'_t = \{s \in \mathbb{N} \ /\ n's \leq d'(t-1) + \varphi'\}$$
$$B_t = \{s \in \mathbb{N} \ /\ ds + \varphi \leq n(t-1)\} \qquad B'_t = \{s \in \mathbb{N} \ /\ d's + \varphi' \leq n'(t-1)\}$$

Let suppose $\mathcal{A}_K(H)_{(n,\varphi,d)} = \mathcal{A}_K(H)_{(n',\varphi',d')}$ *true and verify* $\mathcal{A}_H(K)_{(n,\varphi,d)} = \mathcal{A}_H(K)_{(n',\varphi',d')}$.

$\mathcal{A}_K(H)_{(n,\varphi,d)} = \mathcal{A}_K(H)_{(n',\varphi',d')}$ *implies :* $\forall t \in \mathbb{N}^*, (\ d(t-1) + \varphi < 0 \Leftrightarrow d'(t-1) + \varphi' < 0\ )$

and $\left\lceil \frac{d(t-1)+\varphi}{n} \right\rceil = \left\lceil \frac{d'(t-1)+\varphi'}{n'} \right\rceil, \forall t \in \mathbb{N}^*, d(t-1) + \varphi \geq 0$

Using these properties, we propose a 3-step proof :

1. *First of all, we prove that $A_t = A'_t, \forall\ t \in \mathbb{N}^*$.*
2. *The previous property will imply $B_t = B'_t, \forall\ t \in \mathbb{N}^*$.*

[4] We note $\mid A \mid$ the cardinal (number of elements) of the set A; by convention $\mid \emptyset \mid = 0$ ($\emptyset$ is the empty set).

3. *Finally,* $B_t = B'_t, \forall\ t \in I\!N^*\ \Rightarrow\ \mathcal{A}_{H(K)}{}_{(n,\varphi,d)}\ = \mathcal{A}_{H(K)}{}_{(n',\varphi',d')}$.

1. $A_t = A'_t, \forall\ t \in I\!N^*$.

   Proof :

   Let consider $t \in I\!N^*$ *an integer strictly positive.*

   - *In case* $d(t-1)+\varphi < 0$ *we obtain :*

$$\left.\begin{array}{l} d'(t-1)+\varphi' < 0 \quad \Rightarrow \quad A'_t = \emptyset \\ \textit{and} \\ A_t = \emptyset \end{array}\right\} \Rightarrow A_t = A'_t$$

   - *In case* $d(t-1)+\varphi \geq 0$ *(which is equivalent with* $d'(t-1)+\varphi' \geq 0$*), it can be proved that* $A_t \subseteq A'_t$ *:*

$$\begin{array}{rcl} \forall\ s \in A_t & \Rightarrow & ns \leq d(t-1)+\varphi \\ & \Rightarrow & s \leq \dfrac{d(t-1)+\varphi}{n} \\ & \overset{s\in I\!N}{\Rightarrow} & 0 \leq s \leq \left[\frac{d(t-1)+\varphi}{n}\right] \\ & \Rightarrow & (s \in I\!N) \bigwedge (0 \leq s \leq \left[\frac{d'(t-1)+\varphi'}{n'}\right] \leq \dfrac{d'(t-1)+\varphi'}{n'}) \\ & \Rightarrow & (0 \leq s \leq \dfrac{d'(t-1)+\varphi'}{n'}) \bigwedge (s \in I\!N) \\ & \Rightarrow & s \in A'_t \end{array}$$

   Similarly can be proved that $A'_t \subseteq A_t$*. Therefore,* $A_t = A'_t, \forall\ t \in I\!N^*$.

   (proof step 1.) □

2. $B_t = B'_t, \forall\ t \in I\!N^*$.

   Proof :

   We prove first the following equivalence : $\forall t \in I\!N^*,\ (\ n(t-1)-\varphi < 0\ \Leftrightarrow\ n'(t-1)-\varphi' < 0\ )$.

   Let suppose it exists $s \in I\!N^*$ *verifying* $n(s-1)-\varphi < 0$ *and* $n'(s-1)-\varphi' \geq 0$.

$$\begin{array}{rcl} n(s-1)-\varphi < 0 & \Rightarrow & \forall\ t \in I\!N^*,\ n(s-1)-\varphi < d(t-1) \\ & \Rightarrow & \forall\ t \in I\!N^*,\ (s-1) \in A_t \\ & \overset{A_t = A'_t}{\Rightarrow} & \forall\ t \in I\!N^*,\ (s-1) \in A'_t \\ & \Rightarrow & \forall\ t \in I\!N^*,\ n'(s-1) \leq d'(t-1)+\varphi' \\ & \Rightarrow & \forall\ t \in I\!N^*,\ n'(s-1)-\varphi' \leq d'(t-1) \\ & \Rightarrow & n'(s-1)-\varphi' \leq 0 \end{array}$$

   $s \in I\!N^*$ *satisfies* $n'(s-1)-\varphi' \leq 0$ *and* $n'(s-1)-\varphi' \geq 0$*. Therefore* $n'(s-1)-\varphi' = 0$.

   H and K are in affine relation, $H \overset{(n,\varphi,d)}{\rightarrow} K$ *and* $H \overset{(n',\varphi',d')}{\rightarrow} K$*, and from the definition of an affine relation (Definition 2) there exist two maps* $g, g' : I\!N \rightarrow I\!N$ *monotonically increasing on* $I\!N$ *and strictly monotonic on* $f_H(I\!N) \cup f_K(I\!N)$*, respectively* $f'_H(I\!N) \cup f'_K(I\!N)$*, which denote the value sets of H and K, that is* $ValSet(H) = H_{(g \circ f_H)(I\!N)}\ = H_{(g' \circ f'_H)(I\!N)}$ *and* $ValSet(K) = K_{(g \circ f_K)(I\!N)}\ = K_{(g' \circ f'_K)(I\!N)}$.

   Therefore $(g \circ f_H)(I\!N) = (g' \circ f'_H)(I\!N)$ *and* $(g \circ f_K)(I\!N) = (g' \circ f'_K)(I\!N)$.

   The identities above imply :

$$\begin{array}{rcl} (\forall)t, s \in I\!N,\ nt = ds+\varphi & \Leftrightarrow & f_H(t) = f_K(s) \\ & \Leftrightarrow & (g \circ f_H)(t) = (g \circ f_K)(s) \\ & \Leftrightarrow & (g' \circ f'_H)(t) = (g' \circ f'_K)(s) \\ & \Leftrightarrow & f'_H(t) = f'_K(s) \\ & \Leftrightarrow & (\forall)t, s \in I\!N,\ n't = d's+\varphi' \end{array}$$

As we have obtained $\exists s \in \mathbb{N}^*, n'(s-1)-\varphi'=0$, *it results* $n(s-1)-\varphi=0$ *which is in contradiction with the initial supposition* $n(s-1)-\varphi<0$.

Therefore, the supposition that $\exists s \in \mathbb{N}^*$, $n(s-1)-\varphi<0$ *and* $n'(s-1)-\varphi'=0$ *is false and we can conclude that* $\forall t \in \mathbb{N}^*$, $(\ n(t-1)-\varphi<0 \Leftrightarrow n'(t-1)-\varphi'<0\ )$.

Let consider $t \in \mathbb{N}^*$ *an integer strictly positive.*

- *In case* $n(t-1)-\varphi<0$ *we obtain :*

$$\left.\begin{array}{l} n'(t-1)-\varphi'<0 \quad \Rightarrow \quad B'_t=\emptyset \\ \quad et \\ B_t=\emptyset \end{array}\right\} \Rightarrow B_t=B'_t$$

- *In case* $n(t-1)-\varphi\geq 0$ *(which is equivalent with* $n'(t-1)-\varphi'\geq 0$*), it can be proved that* $B_t \subseteq B'_t$ *:*

$$\begin{array}{rcll} \textit{Let consider } s\in B_t & \Rightarrow & ds\leq n(t-1)-\varphi & (1)\\ \textit{If we suppose } s\notin B'_t & \Rightarrow & d's+\varphi'>n'(t-1) & \\ & \overset{t\in\mathbb{N}^*}{\Rightarrow} & d's+\varphi'>0 \ \ and \ \ (t-1)\in A'_{s+1}\neq\emptyset & \\ & \overset{A'_{s+1}=A_{s+1}}{\Rightarrow} & (t-1)\in A_{s+1} & \\ & \Rightarrow & n(t-1)\leq ds+\varphi & \\ & \overset{n'(t-1)\neq d's+\varphi'}{\Rightarrow} & nt<ds+\varphi & \end{array}$$

which is in contradiction with hypothesis (1).

We obtained that $\forall\ s\in\mathbb{N},\ s\in B_t \ \Rightarrow\ s\in B'_t$, *i.e.* $B_t\subseteq B'_t$.
Similarly we have $B'_t\subseteq B_t$, *and it results* $B'_t=B_t, \forall t\in\mathbb{N}^*$. *(proof step 2.)* □

3. $B'_t=B_t, \forall t\in\mathbb{N}^* \ \Rightarrow\ |\ B'_t\ |\ =|\ B_t\ |\ , \forall t\in\mathbb{N}^* \ \Rightarrow\ \mathcal{A}_{H(K)}{}_{(n,\varphi,d)}\ =\mathcal{A}_{H(K)}{}_{(n',\varphi',d')}$

Therefore $\mathcal{A}_{K(H)}{}_{(n,\varphi,d)}\ =\mathcal{A}_{K(H)}{}_{(n',\varphi',d')}\ \Rightarrow \mathcal{A}_{H(K)}{}_{(n,\varphi,d)}\ =\mathcal{A}_{H(K)}{}_{(n',\varphi',d')}$ *and the inverse implication can be proved similarly.* *(proof Proposition 2)* □

B Proof of Lemma 2 of section 3.4.3

We will prove that $CF_{(n,\varphi,d)}$ *is the smallest relation – with respect to the* $\leq$ *order defined on 3-tuples – equivalent to* (n,φ,d).

a) a.1 *(n,φ,d) and $(\frac{n}{G},\frac{\varphi}{G},\frac{d}{G})$ are equivalent (they satisfy the a) conditions of Lemma 1).*

a.2 *Let (n',φ',d') be equivalent with (n,φ,d) and $(n',\varphi',d')\neq(\frac{n}{G},\frac{\varphi}{G},\frac{d}{G})$. It results (n',φ',d') equivalent with $(\frac{n}{G},\frac{\varphi}{G},\frac{d}{G})$ and they satisfy the a) conditions of Lemma 1, i.e. $n'/d'=(\frac{n}{G})/(\frac{d}{G})$ and $d'/\varphi'=(\frac{d}{G})/(\frac{\varphi}{G})$.*
$\exists\ n_m,d_m,\varphi_m,\ n=Gn_m, d=Gd_m, \varphi=G\varphi_m$ and moreover n_m,d_m,φ_m are relatively prime numbers. Therefore $\frac{n'}{d'}=\frac{n_m}{d_m}$ and $\frac{d'}{\varphi'}=\frac{d_m}{\varphi_m}$.
Because $(n',\varphi',d')\neq(\frac{n}{G},\frac{\varphi}{G},\frac{d}{G})=(n_m,\varphi_m,d_m)$ and n_m,d_m,φ_m are relatively prime numbers, the above results imply $n'>\frac{n}{G},\varphi'>\frac{\varphi}{G}$ and $d'>\frac{d}{G}$, i.e. $(n',\varphi',d')>(\frac{n}{G},\frac{\varphi}{G},\frac{d}{G})$.
In conclusion, $(\frac{n}{G},\frac{\varphi}{G},\frac{d}{G})$ is the smallest element of the equivalence class of (n,φ,d), i.e. $(\frac{n}{G},\frac{\varphi}{G},\frac{d}{G})$ is the canonical form of (n,φ,d).

b) *Suppose that* $\varphi>0$ *(the case* $\varphi\leq 0$ *can be treated similarly using the relation* $\left[\frac{nt-\varphi}{d}\right]=\left[\frac{n't-\varphi'}{d'}\right], \forall\ t\in\mathbb{N}$*). In this case* $dt+\varphi>0, \forall\ t\in\mathbb{N}$. *Thus, the* b) *conditions of lemma 1 are equivalent with:*

$$\begin{array}{l} (\ gcd(n,d)\ \not|\ \varphi)\wedge(\ gcd(n',d')\ \not|\ \varphi') \\ nd'=dn' \\ \left[\frac{dt+\varphi}{n}\right]=\left[\frac{d't+\varphi'}{n'}\right], \forall\ t\in\mathbb{N} \end{array} \quad\Leftrightarrow\quad \begin{array}{l} (\ gcd(n,d)\ \not|\ \varphi)\wedge(\ gcd(n',d')\ \not|\ \varphi') \\ nd'=dn' \\ \left[\frac{dt+\varphi}{n}\right]=\left[\frac{d't+\varphi'}{n'}\right], \forall\ t\in\{0,1,2,...,(n/G)-1\} \end{array}$$

From the last relation in the right-hand group above we can see that it is sufficient to take into consideration only a finite set of values for t.

We fix here some notations to be used in the sequel. We note $N^k_{n/G} = \{0,1,2,...,k(n/G)-1\}$ *with* $k \in \mathbb{N}, k \geq 1$. $\forall\ t \in N^1_{n/G}$, *we note* $a_t = \frac{n}{G}\left[\frac{dt+\varphi}{n}\right] - \frac{d}{G}t$ *and* $A^k_t = \{ka_t + i, \forall\ i \in N^k_{n/G}\}$.

b.1 *In the equivalence class of* (n,φ,d), *we define the minimal relation* (n',φ',d') *as follows:*

$$\exists\ m \in \mathbb{N}, m \geq 2,\ n' = m(n/G)\ \text{and}\ d' = m(d/G); \tag{2}$$

$$m\ \text{is minimal}\ ; \tag{3}$$

$$\varphi' \in \bigcap_{t \in N^1_{n/G}} A^m_t\ \text{and}\ m \not| \varphi'; \tag{4}$$

$$\varphi'\ \text{is minimal.} \tag{5}$$

b.2 *There exists at least one pair* (m,φ') *that satisfies (2) and (4):* (n,φ,d) *belongs to its equivalence class,* $G \not| \varphi$ *and from* $\left[\frac{dt+\varphi}{n}\right] = \left[\frac{dt+\varphi}{n}\right], \forall\ t \in N^1_{n/G}$ *it results* $\varphi \in \bigcap_{t \in N^1_{n/G}} A^G_t$. *Thus* $m = G$ *and* $\varphi' = \varphi$ *satisfy the previous conditions.*

b.3 *For* $m = 2$, *there exists a unique* φ' *which satisfies* $\varphi' \in \bigcap_{t \in N^1_{n/G}} A^2_t$ *and* $2 \not| \varphi'$.

Proof :

– *From* b.2, $\varphi \in \bigcap_{t \in N^1_{n/G}} A^G_t$ *and* $G \not| \varphi$, *i.e.* $\bigcap_{t \in N^1_{n/G}} \{x^t_i | x^t_i = Ga_t + i, \forall\ i \in N^G_{n/G}\ \text{and}\ G \not| x^t_i\} \neq \emptyset$. *We have thus:*

$$\exists\ i_0,...,i_{n/G-1} \in N^G_{n/G},\quad G \not| i_k \quad \text{and}\ Ga_0 + i_0 = ... = Ga_{n/G-1} + i_{n/G-1}$$

equivalent with

$$\exists\ i_0, i_1,...,i_{n/G-1} \in N^G_{n/G},\ \ G \not| i_k,\ \text{satisfying the Diophantine system}$$

$$(S_1)\quad \begin{cases} i_0 - i_1 = G(a_1 - a_0) \\ i_0 - i_2 = G(a_2 - a_0) \\ \cdots \\ i_0 - i_{n/G-1} = G(a_{n/G-1} - a_0) \end{cases}$$

The solutions of the Diophantine system (S_1) *are of the form (see [25]):*

$$(S'_1)\quad \begin{cases} i_0 = s \\ i_1 = s - G(a_1 - a_0) \\ i_2 = s - G(a_2 - a_0) \\ \cdots \\ i_{n/G-1} = s - G(a_{n/G-1} - a_0) \end{cases}$$

with $(s - G(a_k - a_0)) \in N^G_{n/G}$ *and* $G \not| (s - G(a_k - a_0)), \forall\ k \in N^1_{n/G}$.
As (S_1) *has at least one solution,* $\exists\ s = Gr + l$, *with* $r \in N^1_{n/G}$ *and* $l \in \{1,2,...,G-1\}$, *so that* s *satisfies the above system. Moreover,* $s - G(a_k - a_0) = G(r - a_k + a_0) + l$ *and* $s - G(a_k - a_0)$ *must satisfy* $(s - G(a_k - a_0)) \in N^G_{n/G}$ *and* $G \not| (s - G(a_k - a_0)), \forall\ k \in N^1_{n/G}$. *These results imply that the Diophantine system* (S_2) *has at least one solution.*

$$(S_2)\quad \begin{cases} r - p_k = a_k - a_0 \\ r, p_k \in N^1_{n/G},\ \forall\ k \in \{1,2,...,n/G-1\} \end{cases}$$

– *It can be proved by induction that there is a* unique *vector* $(r, p_1, p_2, ..., p_{n/G-1})$ *that satisfies* (S_2).
Moreover, the solutions of (S'_1) *are of the form* $s_l = Gr + l, l \in \{1,2,...,G-1\}$. *As* $\varphi \in \bigcap_{t \in N^1_{n/G}} A^G_t, \exists\ l \in \{1,2,...,G-1\}, \varphi = Ga_0 + s_l$. *We obtain* $\varphi = G(a_0 + r) + l$, *i.e.* $r = \left[\frac{\varphi}{G}\right] - a_0$.

– *As previously for* $\bigcap_{t \in N^1_{n/G}} A^G_t$, *we have:*

$$\exists\ \varphi' \in \bigcap_{t \in N^1_{n/G}} A^2_t \neq \emptyset,\ 2 \not| \varphi' \quad \Leftrightarrow \quad \text{The following system has at least a solution}$$

$$(S_3)\quad \begin{cases} i_0 = s' \\ i_1 = s' - 2(a_1 - a_0) \\ i_2 = s' - 2(a_2 - a_0) \\ \cdots \\ i_{n/G-1} = s' - 2(a_{n/G-1} - a_0) \\ \\ s' - 2(a_k - a_0)) \in N^2_{n/G}, \quad \forall\ k \in N^1_{n/G} \\ 2 \not| (s' - 2(a_k - a_0)), \qquad \forall\ k \in N^1_{n/G} \end{cases}$$

A potential solution of (S_3) *will be of the form* $s' = 2r' + 1$, *with* r' *the first element of the solution vector of* (S_2) . *It follows* $r' = r$ *and* (S_3) *has the unique solution* $s' = 2r + 1$.

– *Therefore, there exists a unique* $\varphi' \in \bigcap_{t \in N^1_{n/G}} A^2_t, \varphi' = 2a_0 + s' = 2(a_0 + r) + 1 = 2\left[\frac{\varphi}{G}\right] + 1$.

(proof b.3) □

b.4 $(n' = 2\frac{n}{G}, \varphi' = 2\left[\frac{\varphi}{G}\right] + 1, d' = 2\frac{n}{G})$ *is the minimal relation equivalent with* (n, φ, d).

$\forall$ (n'', φ'', d'') *equivalent with* (n, φ, d), $\exists\ p \in \mathbb{N}, p \geq 2$, $n'' = p(n/G), d'' = p(d/G)$. *Thus,* $n'' \geq n', d'' \geq n'$. *Moreover, the above proof implies* $\varphi'' = p(a_0 + r) + l_p$, *with* $l_p \in \{1, 2, ..., p-1\}$. *Therefore,* $\varphi'' \geq \varphi'$. *In conclusion,* $(2\frac{n}{G}, 2\left[\frac{\varphi}{G}\right] + 1, 2\frac{n}{G})$ *is the canonical form of* (n, φ, d).

(proof Lemma 2) □

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399