Electronic Notes in Theoretical Computer Science 65 No. 5 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume65.html 15 pages

A Decidable Clock Language for Synchronous Specifications

Mirabelle Nebut [1] and Sophie Pinchinat [2]

IRISA – *Campus universitaire de Beaulieu – 35042 Rennes Cédex – France*

Abstract

Presence and absence of signals inside a reaction are inherent to the synchronous paradigm. Clocks are sets of instants (indicating for example when a signal is present) mainly used to describe the control part of data-flow specifications. The language $\mathcal{CL}$ we define here expresses relations between clocks. Such relations can describe the combinational part of specifications, as well as particular instantaneous safety properties. We give a decision procedure for $\mathcal{CL}$ and apply it to the model-checking of SIGNAL programs abstracted from their state handling part. Thanks to the use of clocks, absence is not explicitly encoded by a special value.

Key words: synchronous data-flow languages, clock, absence, safety property, decision procedure, abstraction, model-checking

1 Introduction

Synchronous languages [4] have been proposed to specify reactive systems, which interact continuously with the environment they are connected to. All synchronous programming paradigms involve the notion of logical instants yielding to the notion of presence/absence of signals. It is particularly apparent in imperative languages like ESTEREL (see [4]), especially dedicated to control-handling aspects. The system can among other things broadcast signals (`emit S`) or react to their *status* of presence or absence (e.g. `present S then ... else ...`). In this paper we focus on equational data-flow languages like SIGNAL [10] and LUSTRE [5], particularly adapted to data-flow aspects. Data-flow specifications are systems of equations which describe infinite sequences of values carried by signals along the time. At a given instant, a signal (e.g. of type boolean, integer, etc) can be absent (represented by a special "absent" value) or present, hence carrying a significant value.

[1] ESPRESSO group, Email: mnebut@irisa.fr
[2] S4 group, Email: pinchina@irisa.fr

An execution of the system is represented by an infinite sequence of instantaneous reactions along a logical time line. A reaction attaches a status and, when relevant, a value to each signal. Such an assignment is called a *valuation*. We call *flow* a time-indexed sequence of valuations. A system is implicitly composed of a *combinational* or *static* part (that is without state), connected to a state handling part. The combinational part describes the admissible valuations for a reaction, by means of a relation between the status and values of signals. It involves potentially absent signals. The state part memorizes from one instant to the next one significant values, to which are set some signals in the reaction.

Reasoning about systems thus involves heterogeneous values for signals: relevant values plus the "absent" value. The data-flow paradigm then offers the notion of *clock*, which refers to a set of instants (e.g. the instants "when the sum of two signals is positive"). In particular the *clock of a signal* x, denoted by $\widehat{x}$, indicates its instants of presence: handling values of x only at instants of $\widehat{x}$ *avoids to use the "absent" value*. Signal widely relies on clocks, used both for programming and analysis. Especially they are essential to describe the control of systems (e.g. by indicating when a computation must be performed). Relations between clocks (stating inclusions, and called *synchronizations*) are analyzed to synthesize clever control structures and detect inconsistencies.

Since clocks are related to signals and time index, they are formally defined w.r.t. flows. Nevertheless, relations between clocks describe a statement that holds in any reaction. For example "$\widehat{x}$ is included into $\widehat{y}$" means that *at any instant* t, the presence of x implies the presence of y. Such a statement is purely combinational, thus describes essentially a set of valuations (which can be rearranged as a set of flows). In other words, a relation between clocks states a particular *instantaneous safety* property of a system, describing what happens at any instant, independently of the preceding or following ones (e.g. "x is always positive"). The verification of a safety property requires in general (even for instantaneous ones) the exploration of the state space of the system. Nevertheless, many of them (in particular those needed to analyze synchronizations) can be verified by considering only the combinational part of the system, abstracting from its states by means of a *static abstraction*.

The present work proposes a "Clock Language" called $\mathcal{CL}$ to express instantaneous data-flow statements or properties (a data-flow semantics is recalled in Sect. 2). Basically, the formulas of the language, presented in Sect. 3, express inclusions between clocks, and can represent any combinational relation between the status and values of signals. We give in Sect. 4 a decision procedure for $\mathcal{CL}$, which as announced does *without the "absent" value*. We illustrate the approach in Sect. 5: combinational Signal statements are translated into $\mathcal{CL}$, then model-checking of the static abstraction of a Signal program is easily derived from the decision algorithm. Comparison with related work on Lustre and Signal (in particular concerning the handling of absence) is given in Sect. 6. Section 7 concludes. We also refer to [9] for further details.

2 Synchronous Data-flow Semantics

We assume given a finite set of variables S of domain D (for example boolean, integer, etc), with typical elements x, x_i, y, z, c. A data-flow specification describes a system by a set of equations over $A \subseteq S$. At any instant, a variable $x \in A$ can be absent (denoted by the special value $\star \notin D$) or present: it carries a significant value $v \in D$. A *valuation* V on A denotes a reaction: it is a function $V : A \to D \cup \{\star\}$ which assigns a value to each variable. The set of valuations on A is written $\mathcal{V}_A$. It contains the *silent* valuation ${}^{\star}V_A : A \to \star$, and non-silent valuations said *significant*. The *restriction* of $V \in \mathcal{V}_A$ to $A_1 \subseteq A$ maps any $x \in A_1$ to $V(x)$.

A *flow* F on A is an infinite denumerable sequence of valuations: $F : \mathbb{N} \to \mathcal{V}_A \setminus \{{}^{\star}V_A\}$. It defines an execution of the system. The set of all valuations that occur at some instant in F is $[\![F]\!]_{\mathcal{V}_A} \triangleq \{V \in \mathcal{V}_A \mid \exists t \in \mathbb{N},\ V = F(t)\}$. The set of flows on A is denoted by $\mathcal{F}_A$. The *clock* of a variable $x \in A$ w.r.t. a flow $F \in \mathcal{F}_A$ is denoted by $\widehat{x}_F$ and defined as the set of instants for which x is present in $F(t)$: $\forall t \in \mathbb{N},\ t \in \widehat{x}_F$ iff $F(t)(x) \neq \star$. A natural preorder $\subseteq_F$ between clocks (w.r.t. to the flow F) is derived accordingly: $\widehat{x} \subseteq_F \widehat{y}$ iff $\widehat{x}_F \subseteq \widehat{y}_F$. If $\widehat{x} =_F \widehat{y}$ then x and y are *synchronous* in F (both absent or present at the same time). $\widehat{x} \subseteq_F \widehat{y}$ expresses an instantaneous relation: $\forall t \in \mathbb{N},\ F(t)(x) \neq \star$ implies $F(t)(y) \neq \star$.

A *process* is a set of flows; it describes the possible behaviors of the system. The set of processes on A is denoted by $\mathcal{P}_A$ whose typical elements are $\varpi, \varpi_1, \varpi_2$. Classically, the *restriction* of $\varpi \in \mathcal{P}_A$ to $A_1 \subseteq A$ is inherited from the restriction of valuations. The *parallel composition* of $\varpi_1 \in \mathcal{P}_{A_1}$ and $\varpi_2 \in \mathcal{P}_{A_2}$, denoted by $\varpi_1 \parallel \varpi_2 \in \mathcal{P}_{A_1 \cup A_2}$, contains behaviors of both ϖ_1 and ϖ_2, provided they agree on their shared variables in $A_1 \cap A_2$.

3 The Clock Language $\mathcal{CL}$

$\mathcal{CL}$ describes instantaneous statements: what happens inside any reaction. Thus according to previous notations its semantics is given by means of valuations, hence it can be lifted to flows (that is full executions) by arbitrary concatenation of valuations. $\mathcal{CL}$ formulas express inclusions between sets of instants, represented by clock terms like $\widehat{x}$ (see Sect. 2), but also based on predicates of a first order theory which describe relations between values of variables. $\mathcal{CL}$ is presented in Sect. 3.2 and 3.3, and a boolean abstraction dedicated to the decision procedure is presented in Sect. 3.4.

3.1 Notations for predicates

We assume given a first order language $\mathcal{L}$ with equality whose set of variables X (with typical element v) contains S. The set of formulas of $\mathcal{L}$ is denoted by $\mathcal{F}(\mathcal{L})$, with typical element R (for example R is $x_1 < x_2 + 10$, where $x_1, x_2 \in S$). The set of free variables of $R \in \mathcal{F}(\mathcal{L})$ is denoted by $fv(R)$ and we

note $R(x_1, \ldots, x_n)$ to emphasize that $fv(R) = \{x_1, \ldots, x_n\}$. A structure (or a realization) $\mathcal{M}$ for $\mathcal{L}$ is obtained by choosing a domain D and an interpretation for symbols of predicate and function according to D (for example D is $\mathbb{Z}$, $+$ is the addition over integers). Given an interpretation function $l : X \to D$ for variables in X, we write $\mathcal{M}, l \models\!\!\!\approx R$ or simply $l \models\!\!\!\approx R$ (when $\mathcal{M}$ is clear) if R is satisfied by l, or $(d_1, \ldots, d_n) \models\!\!\!\approx R(x_1, \ldots, x_n)$ for $d_i \in D, i = 1 \ldots n$. In the following we assume chosen a fixed structure for $\mathcal{L}$.

3.2 Clock Terms

Clock terms can be the clock of a variable, the empty clock, or a composition of terms by set-like operators, but also "embedded" predicates of $\mathcal{F}(\mathcal{L})$, called *relation-clocks*. The set of clock terms $\mathcal{CT}$ is described by[3]:

$$\mathcal{CT}\ (\ni h, h_1, h_2) ::= \mathbb{O} \mid \widehat{x} \mid \langle R(x_1, \ldots, x_n) \rangle \mid h_1 \cap h_2 \mid h_1 \cup h_2 \mid h_1 \setminus h_2$$

where $x \in S$ and $R \in \mathcal{F}(\mathcal{L})$. The empty clock $\mathbb{O}$, clock variables $\widehat{x}$ and relation-clocks $\langle R \rangle$ are *basic* clock terms, whose set is denoted by $\mathcal{BCT}$, or $\mathcal{BCT}_A$ when variables can only belong to A. An *inductive* clock term has the form h_1 *op* h_2 where *op* $\in \{\cup, \cap, \setminus\}$. The set of clock terms on A is denoted by $\mathcal{CT}_A$, whose typical elements are h, h_1, h_2, h^1, etc.

The interpretations of clock terms are valuations: for $x \in A$, if $V \in \mathcal{V}_A$ is such that $V(x) \neq \star$ then V is said to *switch on* the clock of x, which is written $on_V(\widehat{x})$. The unary predicate $on_V(\cdot)$ generalizes to any clock term:

Definition 3.1 Given $A \subseteq S$, $V \in \mathcal{V}_A$ and $h, h_1, h_2 \in \mathcal{CT}_A$, $on_V(\cdot)$ is defined by induction over clock terms:

- $on_V(\mathbb{O})$ never;
- for $x \in A$, $on_V(\widehat{x})$ iff $V(x) \neq \star$;
- for $\{x_1, \ldots, x_n\} \subseteq A$, $on_V(\ \langle R(x_1, \ldots, x_n) \rangle\ )$ iff for $i = 1 \ldots n$, $on_V(\widehat{x_i})$ and $(V(x_1), \ldots, V(x_n)) \models\!\!\!\approx R$;
- $on_V(h_1 \cap h_2)$ iff $on_V(h_1)$ and $on_V(h_2)$;
- $on_V(h_1 \cup h_2)$ iff $on_V(h_1)$ or $on_V(h_2)$;
- $on_V(h_1 \setminus h_2)$ iff $on_V(h_1)$ and not $on_V(h_2)$.

Example 3.2 Assume given $A = \{x, y\}$ and let V be defined by $V(x) = 3$ and $V(y) = \star$. Then, $on_V(\widehat{y})$ is false, $on_V(\widehat{x} \setminus \widehat{y})$ is true, $\langle x > 0 \rangle$ is switched on in V but $\widehat{y}$ hence $\langle x > 0 \vee y > 0 \rangle$ are switched off, since y is absent in V. $\diamond$

3.3 Clock Formulas

Clock formulas describe predicates over clock terms, which basically rely on inclusions. The set $\mathcal{CF}$ of clock formulas is described by:

[3] We use BNF in which the symbol | denotes choice in the grammar and has nothing to do with the syntactical operator of parallel composition |.

$$\mathcal{CF}\ (\ni \phi, \phi_1, \phi_2) ::= h_1 \subseteq h_2 \ \mid\ \phi_1 \wedge \phi_2 \ \mid\ \exists x.\phi \ \mid\ \neg\phi$$

where $h_1, h_2 \in \mathcal{CT}$ and $x \in S$. For $A \subseteq S$, the set of clock formulas on A is denoted by $\mathcal{CF}_A$. The *clock term universe* associated to $\phi \in \mathcal{CF}_A$ and denoted by $CT(\phi) \subseteq \mathcal{CT}$ is defined inductively: $CT(h_1 \subseteq h_2)$ is the set of clock sub-terms that compose h_1 and h_2; $CT(\exists x.\phi) = CT(\neg\phi) = CT(\phi)$ and $CT(\phi_1 \wedge \phi_2) = CT(\phi_1) \cup CT(\phi_2)$. We denote by $BCT(\phi)$ the set $\mathcal{BCT} \cap CT(\phi)$, the set of basic clock terms of ϕ and by $CF(\phi) \subseteq \mathcal{CF}$ the set of sub-formulas of ϕ. The set of quantifier-free formulas (that contain no sub-formulas of the kind $\exists x.\phi$) in $\mathcal{CF}$ (resp. $\mathcal{CF}_A$) is denoted by $\mathcal{CF}^{qf}$ (resp. $\mathcal{CF}^{qf}{}_A$).

In the following, we write $h_1 \equiv h_2$ for $h_1 \subseteq h_2 \wedge h_2 \subseteq h_1$ (the symbol $=$ is kept to denote equality of $\mathcal{L}$) and $h_1 \equiv h_2 \equiv h_3$ means $h_1 \equiv h_2 \ \wedge\ h_2 \equiv h_3$.

Semantics of clock formulas is as announced based on valuations.

Definition 3.3 Given $A \subseteq S$, $V \in \mathcal{V}_A$, $h_1, h_2 \in \mathcal{CT}_A$, $x \in S \setminus A$ and $\phi, \phi_1, \phi_2 \in \mathcal{CF}_A$:

- $V \models h_1 \subseteq h_2$ iff $on_V(h_1)$ implies $on_V(h_2)$;
- $V \models \phi_1 \wedge \phi_2$ iff $V \models \phi_1$ and $V \models \phi_2$;
- $V \models \exists x.\phi$ iff there exists $V' \in \mathcal{V}_{A \cup \{x\}}$ s.t. $V' \models \phi$ and $\forall y \in A, V'(y) = V(y)$;
- $V \models \neg\phi$ iff $V \not\models \phi$.

A formula $\phi \in \mathcal{CF}_A$ is *valid*, written $\models \phi$, whenever $V \models \phi$ for all $V \in \mathcal{V}_A$.

This definition is easily lifted to flows: given a formula $\phi \in \mathcal{CF}_A$, we say that a flow $F \in \mathcal{F}_A$ satisfies ϕ, written again $F \models \phi$, whenever ϕ holds for all valuations $V \in [\![F]\!]_{\mathcal{V}_A}$.

To clarify notations we define a function $\Upsilon(\cdot)$ (read "clock of") which associates a clock term to a formula $R \in \mathcal{F}(\mathcal{L})$:

$$\Upsilon(R) \text{ is the clock term } \bigcap\{\widehat{x} \mid x \in fv(R)\}.$$

Intuitively, "the clock of R", when interpreted in a flow, is the greatest set of instants (for set inclusion) where all variables whose value is required to strictly evaluate R are present. Note that $\Upsilon(\neg R)$ is equal to $\Upsilon(R)$ (since formulas R and $\neg R$ have same free variables) and that by definition $on_V(\Upsilon(R))$ iff $on_V(\widehat{x})$, for all $x \in fv(R)$. By abuse of notations we write $V \!\upharpoonright\!\approx R$ when all variables in $fv(V)$ are present in V and carry values that satisfy R.

Example 3.4 $\Upsilon(x < 10) = \Upsilon(\ \exists z.(z \geq 0 \wedge x < 10 + z)\ ) = \widehat{x}$ and for V as in Ex 3.2 where $V(x) = 3$ and $V(y) = \star$, it is true that $V \!\upharpoonright\!\approx x < 10$, just as $V \!\upharpoonright\!\approx \exists z.(z \geq 0 \wedge x < 10 + z)$. On the contrary, because $\Upsilon(x > 0 \vee y > 0) = \widehat{x} \cap \widehat{y}$, $V \not\upharpoonright\!\approx x > 0 \vee y > 0$. $\diamond$

The semantics of relation-clocks can be rewritten more concisely using Υ:

$$on_V(\langle R \rangle) \text{ iff } on_V(\Upsilon(R)) \text{ and } V \!\upharpoonright\!\approx R \qquad (1)$$

Therefore, for any valuation V, $on_V(\langle R\rangle)$ *implies* $on_V(\Upsilon(R))$, namely $\models \langle R\rangle \subseteq \Upsilon(R)$. It can also be proved that $\models \Upsilon(R) \equiv \langle R\rangle \cup \langle \neg R\rangle$.

Example 3.5 $\mathcal{CL}$ can express various fine instantaneous statement. Assume given $x, y \in S$ and a variable $v \in X$. "x is always positive" is expressed by the formula $\widehat{x} \equiv \langle x \geq 0\rangle$, or equivalently $\widehat{x} \subseteq \langle x \geq 0\rangle$ (since $\models \langle x \geq 0\rangle \subseteq \widehat{x}$). "$y$ is always equal to x" is expressed by $\widehat{y} \equiv \langle y = x\rangle$ or equivalently $\widehat{y} \subseteq \langle y = x\rangle$, as y can be absent while x is present. "x is always even" is expressed by $\widehat{x} \subseteq \langle \exists v.x = 2v\rangle$, and highlights the need for quantification provided by $\mathcal{L}$. $\diamond$

3.4 Boolean Abstraction

We define the abstraction of the quantifier-free fragment of $\mathcal{CL}$ into the propositional calculus $\mathbf{B}$, useful for the decision procedure of Sect. 4, with the following notations. Given $\mathbf{b}$ a set of propositional variables, $\delta : \mathbf{b} \to \{0, 1\}$ is a *distribution* of values on $\mathbf{b}$ which naturally extends to formulas of $\mathbf{B}$ in the standard way (for example $\delta(G_1 \wedge G_2) = 1$ iff $\delta(G_1) = 1$ and $\delta(G_2) = 1$). The set of distributions on $\mathbf{b}$ is Δ. The distribution $\mathbf{0}$ maps all variables to 0. For $\delta \in \Delta$ and $G \in \mathbf{B}$, we write $\delta \models G$ whenever $\delta(G) = 1$.

Now, assume $\mathbf{b}$ contains a variable $b_{\widehat{x}}$ (resp. $b_{\langle R\rangle}$) for each $\widehat{x}$ (resp. $\langle R\rangle$) in $\mathcal{BCT}$. The mapping B of Def. 3.6 abstracts $h \in \mathcal{CT}$ into $B(h)$, simply written h_B, and $\phi \in \mathcal{CF}^{qf}$ into $B(\phi)$, denoted by ϕ_B. Note that semantics of operators is similar in the concrete and abstract worlds.

Definition 3.6 B is defined on Fig. 1 by induction on the structure of terms and formulas, where $x \in S$.

$h \in \mathcal{CT}$	$h_B \in \mathbf{B}$
$\widehat{x}$	$b_{\widehat{x}}$
$\langle R\rangle$	$b_{\langle R\rangle}$
$\mathbb{O}$	$false$
$h_1 \cap h_2$	$h_{1_B} \wedge h_{2_B}$
$h_1 \cup h_2$	$h_{1_B} \vee h_{2_B}$
$h_1 \setminus h_2$	$h_{1_B} \wedge \neg h_{2_B}$

$\phi \in \mathcal{CF}^{qf}$	$\phi_B \in \mathbf{B}$
$h_1 \subseteq h_2$	$h_{1_B} \Rightarrow h_{2_B}$
$\neg\phi$	$\neg\ \phi_B$
$\phi_1 \wedge \phi_2$	$\phi_{1_B} \wedge \phi_{2_B}$

Fig. 1. Abstraction of $\mathcal{CL}$

Abstraction of valuations as well as concretization of distributions are defined in accordance with the boolean abstraction of terms. The *abstraction* of a valuation V is the distribution $\delta_V \in \Delta$ which indicates for each basic clock term h whether it is switched on by V or not.

Definition 3.7 Let $V \in \mathcal{V}_S$. $\delta_V \in \Delta$ is defined by:

$$\text{for all } h \in \mathcal{BCT},\ \delta_V(h_B) = 1 \text{ iff } on_V(h).$$

Reciprocally, the *concretization* of a distribution δ is the (possibly empty) set $\mathcal{V}_\delta$ of all valuations that assign values in $D \cup \{\star\}$ to variables, in accordance with the requirements specified by δ on presence/absence of variables, and satisfaction of formulas $R \in \mathcal{F}(\mathcal{L})$ imposed by variables $b_{\langle R \rangle}$. Formally:

Definition 3.8 Given $\delta \in \Delta$, $\mathcal{V}_\delta \subseteq \mathcal{V}_S$ is defined by:

$$\mathcal{V}_\delta = \{V \in \mathcal{V}_S \mid \forall h \in \mathcal{BCT}, on_V(h) \text{ iff } \delta(h_B) = 1\}.$$

It can be proved that Def. 3.7 and Def. 3.8 generalize to any term $h \in \mathcal{CT}$ and deliver indeed a distribution and a set of valuations respectively.

Example 3.9 If $\mathbf{b}$ is $\{b_{\widehat{y}}, b_{\widehat{x}}, b_{\langle x<0\rangle}, b_{\langle x>2\rangle}\}$ and $V \in \mathcal{V}_S$ is s.t. $V(x) = 3$ and $V(y) = \star$, then $\delta_V(b_{\widehat{x}}) = \delta_V(b_{\langle x>2\rangle}) = 1$ and $\delta_V(b_{\widehat{y}}) = \delta_V(b_{\langle x<0\rangle}) = 0$. $\mathcal{V}_{\delta_V}$ contains any valuation V' s.t. $V'(y) = \star$ and $V'(x) > 2$, in particular V. For δ equal to δ_V excepted that $\delta(b_{\langle x<0\rangle}) = 1$, we have $\mathcal{V}_\delta = \emptyset$. $\diamond$

As expected, these abstraction and concretization define a Galois insertion.

Theorem 3.10 *Let α and γ be the following mappings:*

$$\begin{array}{lll} \alpha : \mathcal{P}(\mathcal{V}_S) \to \mathcal{P}(\Delta) & \text{and} & \gamma : \mathcal{P}(\Delta) \to \mathcal{P}(\mathcal{V}_S) \\ \quad \{V_i\}_{i\in I} \mapsto \{\delta_{V_i}\}_{i\in I} & & \quad \Delta' \mapsto \bigcup_{\delta\in\Delta'} \mathcal{V}_\delta \end{array}$$

then $\gamma \circ \alpha \supseteq Id_{\mathcal{P}(\mathcal{V}_S)}$ and $\alpha \circ \gamma = Id_{\mathcal{P}(\Delta)}$.

4 A Decision Procedure

We define in Sect. 4.2 a decision procedure for the satisfiability of a clock formula $\phi \in \mathcal{CF}$. It answers Yes iff there exists a significant valuation V (hence a flow) such that $V \models \phi$. It relies on properties of the boolean abstraction discussed in Sect. 4.1: the computation of the boolean abstraction ϕ_B of a quantifier-free formula ϕ stands out a finite number of decision problems in the theory $\mathcal{F}(\mathcal{L})$. Then we only need to address these problems.

4.1 The Satisfiability Problem

In this section we focus on the quantifier-free fragment of $\mathcal{CL}$. Abstraction and concretization defined in Sect. 3.4 have strong properties when applied to models of $\phi \in \mathcal{CF}^{qf}$, since $\gamma \circ \alpha(\{V \in \mathcal{V}_S \mid V \models \phi\}) = \{V \in \mathcal{V}_S \mid V \models \phi\}$. Indeed, though the precise values carried by V are lost, δ_V and ϕ_B encode whether $V \models \phi$ or not:

Theorem 4.1 *For any $\phi \in \mathcal{CF}^{qf}{}_A$ and $V \in \mathcal{V}_A$, $V \models \phi$ iff $\delta_V \models \phi_B$.*

The proof of Th. 4.1 is by induction over the structure of ϕ, see [9].

By Th. 3.10 and 4.1, if some $V \models \phi$, then $V \in \mathcal{V}_\delta$ for some $\delta \models \phi_B$, and vice versa:

Corollary 4.2 *For any $\phi \in \mathcal{CF}^{qf}{}_A$ and $V \in \mathcal{V}_A$, $V \models \phi$ iff $V \in \bigcup_{\delta \models \phi_B} \mathcal{V}_\delta$.*

Consequently, we establish in the following a criterion for the non-emptiness of $\mathcal{V}_\delta$, for any $\delta \models \phi_B$. This criterion relies on the satisfiability of a first order formula R_δ built up from ϕ and δ (see Th. 4.5).

For technical reasons, we only consider *completed* formulas ϕ of the kind $\phi' \wedge \bigwedge\{\Upsilon(R) \equiv \langle R\rangle \cup \langle \neg R\rangle \mid \langle R\rangle \in BCT(\phi')\}$; ϕ is then the *completion of* ϕ'. Notice that restricting to completed formulas is no loss of generality since any formula and its completion are equivalent. In the rest of this section, we assume given a completed formula $\phi \in \mathcal{CF}^{qf}{}_A$.

Definition 4.3 Let $\mathcal{R}_\delta$ be the set $\{R \mid \langle R\rangle \in BCT(\phi)$ and $\delta(b_{\langle R\rangle}) = 1\}$. When $\mathcal{R}_\delta$ is not empty, R_δ denotes the conjunction of all formulas in $\mathcal{R}_\delta$.

The following lemma ensures that distributions we consider are well-formed, that is the values of $b_{\widehat{x}}$ for $x \in fv(R)$ are coherent with the values of $b_{\langle R\rangle}$.

Lemma 4.4 *Let $\delta \models \phi_B$. For all $R \in \mathcal{R}_\delta$ and for all $x \in fv(R)$, $\delta(b_{\widehat{x}}) \neq 0$.*

Proof. Since ϕ is completed, for each $\langle R\rangle \in BCT(\phi)$ the sub-formula $\Upsilon(R) \equiv \langle R\rangle \cup \langle \neg R\rangle$, say ψ in the following, occurs positively in ϕ. Hence $\delta \models \phi_B$ implies $\delta \models \psi_B$, where ψ_B is

$$\underbrace{\Big(\bigwedge_{x \in fv(R)} b_{\widehat{x}}\Big)}_{\Upsilon(R)_B} \Leftrightarrow b_{\langle R\rangle} \vee b_{\langle \neg R\rangle}. \tag{2}$$

If $R \in \mathcal{R}_\delta$ then $\delta(b_{\langle R\rangle}) = 1$ and by Eq. 2 we conclude. □

Theorem 4.5 gives a criterion for the non-emptiness of $\mathcal{V}_\delta$. Finally its corollary (Cor. 4.6) states that the satisfiability of ϕ is fully determined by the pairs (δ, R_δ), where $\delta \models \phi_B$.

Theorem 4.5 *Let $\delta \models \phi_B$. Then, $\mathcal{V}_\delta \neq \emptyset$ iff "$\mathcal{R}_\delta = \emptyset$ or else R_δ is satisfiable". Moreover $\forall V \in \mathcal{V}_\delta$, $V \not\approx R_\delta$.*

Proof. When $\mathcal{R}_\delta$ is an empty set the theorem simply states that $\mathcal{V}_\delta$ is not empty. Indeed, one can choose an arbitrary value $v \in D$ and define a valuation $W(x) = v$ if $\delta(b_{\widehat{x}}) = 1$, and $\star$ otherwise, which belongs to $\mathcal{V}_\delta$ by construction. We can assume now that $\mathcal{R}_\delta$ is not empty.
$\Rightarrow$) Assume $\mathcal{V}_\delta \neq \emptyset$ and let $V \in \mathcal{V}_\delta$. For all $R \in \mathcal{R}_\delta$, $\delta(b_{\langle R\rangle}) = 1$ by Def. 4.3, then $on_V(\langle R\rangle)$. Therefore $V \not\approx R$ by Eq. (1) and consequently R_δ is satisfiable.
$\Leftarrow$) If R_δ is satisfiable then there exist values v_x for each variable $x \in fv(R_\delta)$ which satisfy R_δ. On their basis we define $W \in \mathcal{V}_A$ (with v an arbitrary value)

by:

$$W(x) = \begin{cases} \star & \text{if } \delta(b_{\widehat{x}}) = 0 \\ v_x & \text{if } x \in fv(R_\delta) \\ v & \text{for the remaining cases.} \end{cases} \quad (3)$$

Notice that W is well defined since by Lemma 4.4, the three cases of Eq. (3) are disjoint. We prove now that $W \in \mathcal{V}_\delta$, that is, for all $h \in BCT(\phi)$,

$$on_W(h) \text{ iff } \delta(h_B) = 1 \quad (4)$$

By construction of W, (4) holds for clock terms $\mathbb{0}$ and $\widehat{x}$ ($x \in A$). Therefore

$$on_W(\Upsilon(R)) \text{ iff } \delta(\Upsilon(R)_B) = 1, \text{ for all } \langle R\rangle \in BCT(\phi) \quad (5)$$

We now show (4) for all $\langle R\rangle \in BCT(\phi)$. Let $\langle R\rangle \in BCT(\phi)$. If $\delta(b_{\langle R\rangle}) = 1$, then $R \in \mathcal{R}_\delta$, hence $\delta(\Upsilon(R)_B) = 1$ by Def. 4.3 and Lem. 4.4. As moreover $W \models\!\!\approx R$ by construction, by Eq. (1) and (5) we have $on_W(\langle R\rangle)$. Otherwise, $\delta(b_{\langle R\rangle}) = 0$. Two cases can be distinguished: if $\delta(\Upsilon(R)_B) = 0$, by Eq. (5) *not* $on_W(\Upsilon(R))$ necessary holds, which in turn entails *not* $on_W(\langle R\rangle)$, by Eq. (1). Otherwise, $\delta(\Upsilon(R)_B) = 1$. By Eq. (2), we necessarily have $\delta(b_{\langle \neg R\rangle}) = 1$, therefore $\neg R \in \mathcal{R}_\delta$. By the first case of the proof, we have $on_W(\langle \neg R\rangle)$ which implies by Eq. (1) that $W \models\!\!\approx \neg R$. Hence $W \not\models\!\!\approx R$ and therefore *not* $on_W(\langle R\rangle)$.□

Notice that assuming that ϕ is a completed formula is necessary to get Th. 4.5: consider the non-completed formula $\langle x > 0\rangle \subseteq \widehat{x} \ \wedge \ \langle \neg(x > 0)\rangle \subseteq \widehat{x}$. Define the distribution $\delta(b_{\widehat{x}}) = 1$, and 0 otherwise, in particular $\delta(b_{\langle x>0\rangle}) = 0$ and $\delta(b_{\langle \neg(x>0)\rangle}) = 0$. Although δ is a model of ϕ_B, it cannot be concretized into a valuation, entailing emptiness for $\mathcal{V}_\delta$. And yet $\mathcal{R}_\delta$ is also empty.

Corollary 4.6 *$\phi \in \mathcal{CF}^{qf}{}_A$ is satisfiable iff there exists $\delta \in \Delta$ s.t. $\delta \models \phi_B$ and R_δ is satisfiable.*

4.2 The Decision Procedure

Provided the first order language $\mathcal{L}$ is decidable, Cor. 4.6 induces a decision procedure for the full language $\mathcal{CL}$, as a quantified formula $\exists x.\phi$ is satisfiable iff ϕ is satisfiable. The algorithm of Fig. 2 takes $\phi \in \mathcal{CF}^{qf}{}_A$ as input, and returns Yes iff ϕ is satisfied by a valuation $V \neq {}^\star V_A$. It clearly terminates and is correct by Cor. 4.6. It could be improved to effectively exhibit a model of ϕ (if any) by exploiting the valuation W in the proof of Th. 4.5, provided a constructive decision procedure exists for $\mathcal{L}$.

Complexity issues can be briefly answered as follows. Define $|\phi|$ as the number of all symbols occuring in ϕ, included symbols appearing in formulas $R \in \mathcal{F}(\mathcal{L})$ for $\langle R\rangle \in BCT(\phi)$. First, solutions of ϕ_B are enumerated. This runs in exponential time on $|\phi_B|$ ($\leq |\phi|$), since checking that a distribution

Let $DIST = \{\delta \in \Delta \mid \delta \models \phi_B \text{ and } \delta \neq \mathbf{0}\}$ in
For all $\delta \in DIST$ do
  if $\mathcal{R}_\delta = \emptyset$ then return Yes else
    if R_δ is satisfiable then return Yes;
return No.

Fig. 2. A decision procedure for the satisfiability of ϕ

satisfies ϕ_B is linear time. Hence, this enumeration is in $O(2^{|\phi|})$. Then, for each distribution δ, some decision procedure of $\mathcal{L}$ is performed for R_δ; notice that $|R_\delta| \leq |\phi|$. By calling $\mathbf{C}_\mathcal{L}(|R|)$ the complexity for deciding $R \in \mathcal{F}(\mathcal{L})$, we obtain an upper bound complexity for the decision algorithm of $\mathcal{CL}$ in $O(2^{|\phi|} * \mathbf{C}_\mathcal{L}(|\phi|))$. For the lower bound, the complexity is the maximum of (1) the lower bound for enumerating all solutions of a propositional formula (at least NP-hard because of SAT problem) and (2) the complexity of the decision for $\mathcal{L}$.

5 Model-checking for Data-flow Specifications

Deciding whether a $\mathcal{CL}$ formula is satisfiable or not can be of significant help for Signal analyses (see Sect. 6): we illustrate it in Sect. 5.3 by addressing the model-checking of Signal programs. The Signal language is presented in Sect. 5.1 then we show in Sect. 5.2 how the static abstraction of programs translates into clock formulas.

5.1 The Synchronous Data-flow Language Signal

A Signal [10] program on a set of variables A, written $\mathtt{P} \in \mathcal{P}g_A$, is a set of equations; it denotes a process $[\![\mathtt{P}]\!] \in \mathcal{P}_A$. The operator of parallel composition `|` corresponds to the parallel composition of processes. An equation `y := E` involves a signal `y` and an expression `E` composed of signals, constants and operators; it specifies that when present `y` is always equal to `E`. The kernel language offers four operators, single-clocked (that is which constrains its arguments to be synchronous) or multi-clocked. The dynamic *delay* operator `$` involves a state: `y := x $1 init v0` specifies that `y` and `x` are synchronous and when present `y` takes the last non-$\star$ value of `x`, initially the constant `v0`. The semantics of the three combinational operators is given on Fig. 3 as follows. We write $\mathtt{P} \rightsquigarrow exp(F,t)$ to express $[\![\mathtt{P}]\!] = \{F \in \mathcal{F}_A \mid \forall t \in \mathbb{N},\ exp(F,t)\}$, where $exp(F,t)$ is an expression built upon F and t. We write y_t for $F(y)(t)$, corresponding to the program variable `y`.

In (a) the generic operator `g` is any usual arithmetic or boolean function, point-wisely applied to the values of present variables $\mathtt{x_i}$, all arguments being constrained to be synchronous (for example if `g` is addition $\star + 2$ is irrelevan-

P		$[\![\mathtt{P}]\!]$
(a) `y := g(x1, ... ,xn)`	$\rightsquigarrow$	$y_t = \star \Rightarrow x_{1_t} = \cdots = x_{n_t} = \star$ $\bigwedge\ y_t \neq \star \Rightarrow\ \forall i, x_{i_t} \neq \star \wedge y_t = g(x_{1_t}, \ldots, x_{n_t})$
(b) `y := x when c`	$\rightsquigarrow$	$c_t \neq true \Rightarrow y_t = \star$ $\bigwedge\ c_t = true \Rightarrow y_t = x_t$
(c) `y := x default z`	$\rightsquigarrow$	$x_t \neq \star \Rightarrow y_t = x_t$ $\bigwedge\ x_t = \star \Rightarrow y_t = z_t$

Fig. 3. Equations of the kernel of Signal

t). Such an instantaneous function allows the construction of single-clocked predicates (for example `c := x<y` builds the relation $\mathtt{x} < \mathtt{y}$, and `c`, `x` and `y` are synchronous). The other operators are multi-clocked: the `when` operator (b) performs a filtering, or a sampling according to the true values of `c`, and the `default` operator (c) performs a merge with priority given to its first argument `x`. Restriction is provided: `(P/x)` means that `x` is hidden in `P`, which corresponds to the restriction of $[\![\mathtt{P}]\!]$ to variables in $A \setminus \{x\}$. Derived operators also exist (like the operator `^=` which constrains its arguments to be synchronous), as well as powerful constructions for high-level programming.

The *static abstraction* of a program `P`, denoted by $\mathtt{P}_S$, is easily obtained by just keeping its combinational part. Equations of Fig. 3 form the kernel of *static* Signal (written $\mathcal{S}$Signal): the abstraction keeps them as they are. `y := x $1 init v0` is abstracted into `y ^= x`, which only states that `y` and `x` are synchronous. The abstraction is modular: $(\mathtt{P}_1 \mid \mathtt{P}_2)_S$ is $\mathtt{P}_{1S} \mid \mathtt{P}_{2S}$.

Example 5.1 Figure 4 shows Signal equations we consider as the programs $\mathtt{P}_{abs}$ and $\mathtt{P}_{ct}$, defined on the set of integer variables $A = \{a, y, p, n\}$ and $\{a, pa, N\}$ respectively. In $\mathtt{P}_{abs}$ `a` is the absolute value of `y`. In $\mathtt{P}_{ct}$, `a` is a "counter": it decrements and reinitializes itself after it has reached 0, with the positive input variable `N`. Since the variable `pa` carries the previous value of `a` (initially 0), `N` is synchronized with instants where `pa` = `0`. Note that the absolute value denotes a static process, contrary to the counter. ◇

```
(| p : = y when y>=0
 | n := -y when y<0
 | a :=  p default n |)
```

```
(| pa := a $1 init 0
 | a := N default pa-1
 | N ^= when N>=0
 | N ^= when pa=0 |)
```

Fig. 4. Signal programs $\mathtt{P}_{abs}$ (left) and $\mathtt{P}_{ct}$ (right)

5.2 From Static Signal *to* $\mathcal{CL}$

We give in Fig. 5 a mapping which associates to a $\mathcal{S}$Signal program `P` a formula $\phi_{\mathtt{P}} \in \mathcal{CF}$. This translation is correct thanks to Th. 5.2. Note that

synchronizations translate directly into clock formulas: for example `y ^= x` translates into $\widehat{y} \equiv \widehat{x}$.

$\mathcal{S}$SIGNAL program P	$\phi_{\mathtt{P}} \in \mathcal{CF}$
`y := g(x1, ... ,xn)`	$\widehat{y} \equiv \widehat{x_1} \equiv \ldots \equiv \widehat{x_n} \ \wedge\ \widehat{y} \subseteq \langle y = g(x_1, \ldots, x_n) \rangle$
`y := x when c`	$\widehat{y} \equiv \widehat{x} \cap \langle c \rangle \ \wedge\ \widehat{y} \subseteq \langle y = x \rangle$
`y := x default z`	$\widehat{y} \equiv \widehat{x} \cup \widehat{z} \ \wedge\ \widehat{x} \subseteq \langle y = x \rangle \ \wedge\ \widehat{z} \setminus \widehat{x} \subseteq \langle y = z \rangle$
`( P )/x`	$\exists x.\phi_{\mathtt{P}}$
`P1 | P2`	$\phi_{\mathtt{P}_1} \wedge \phi_{\mathtt{P}_2}$

Fig. 5. $\mathcal{CL}$ semantics for $\mathcal{S}$SIGNAL

Theorem 5.2 *For all $\mathcal{S}$SIGNAL program P and $V \in \mathcal{V}_S$,*

$$V \models \phi_{\mathtt{P}} \text{ iff } V \in [\![F]\!]_{\mathcal{V}_S} \text{ for some } F \in [\![\mathtt{P}]\!].$$

Example 5.3 We give on Fig. 6 the two clock formulas ϕ_{abs} and ϕ_{ct} that represent the two SIGNAL programs of Ex. 5.1. ◇

$$\begin{array}{l} \widehat{p} \equiv \langle y \geq 0 \rangle \equiv \langle p = y \rangle \equiv \langle a = p \rangle \\ \wedge\ \widehat{n} \equiv \langle y < 0 \rangle \equiv \langle n = -y \rangle \\ \wedge\ \widehat{n} \setminus \widehat{p} \subseteq \langle a = n \rangle \\ \wedge\ \widehat{a} \equiv \widehat{p} \cup \widehat{n} \end{array} \qquad \begin{array}{l} \widehat{a} \equiv \widehat{pa} \equiv \langle a = N \rangle \cup \widehat{a} \\ \wedge\ \langle pa = 0 \rangle \equiv \langle a = N \rangle \equiv \widehat{N} \equiv \langle N \geq 0 \rangle \\ \wedge\ \widehat{pa} \setminus \widehat{N} \subseteq \langle a = pa - 1 \rangle \end{array}$$

Fig. 6. Clock Formulas ϕ_{abs} (left) and ϕ_{ct} (right) for Ex. 5.1

5.3 Model-checking

Using the static abstraction and the translation from $\mathcal{S}$SIGNAL to $\mathcal{CL}$ given in Sect. 5.1 and 5.2, deciding whether a SIGNAL program P satisfies a property $\phi \in \mathcal{CF}^{qf}$ can be reduced to the validity problem $\phi_{\mathtt{P}_S} \Rightarrow \phi$. This problem reduces in turn to the satisfiability of the formula $\phi_{\mathtt{P}_S} \wedge \neg\phi$, by using renaming to push out quantifiers. As expected, since an abstraction of P is used, if the answer is "NO" then P satisfies ϕ, otherwise nothing can be inferred.

Example 5.4 Let us consider the program $\mathtt{P}_{abs}$ of Ex. 5.1 and address whether "a is always positive". This instantaneous property can be expressed by ϕ: $\widehat{a} \subseteq \langle a \geq 0 \rangle$. Let us denote by ϕ' the completion of $\phi_{abs} \wedge \neg\phi$, which contains additional clock terms like $\langle \neg a \geq 0 \rangle$. Apart from distributions s.t. $\delta(b_{\langle R \rangle}) = \delta(b_{\langle \neg R \rangle}) = 1$ that trivially cannot be concretized, three distributions δ_i satisfy ϕ'_B. According to the table below, none of the formulas R_{δ_i} is satisfiable, so the sets $\mathcal{V}_{\delta_i}$ are empty, hence ϕ indeed holds for $\mathtt{P}_{abs}$.

δ_i	some clocks switched on by δ_i	unsat. sub-formula of R_{δ_i}
δ_1	$\widehat{p}$, $\widehat{n}$, $\widehat{a}$, $\langle y \geq 0\rangle$, $\langle y < 0\rangle$, $\langle \neg a \geq 0\rangle$	$y \geq 0 \wedge y < 0$
δ_2	$\widehat{p}$, $\widehat{a}$, $\langle y \geq 0\rangle$, $\langle p = y\rangle$, $\langle a = p\rangle$, $\langle \neg a \geq 0\rangle$	$y \geq 0 \wedge p = y \wedge a = p \wedge \neg a \geq 0$
δ_3	$\widehat{n}$, $\widehat{a}$, $\langle y < 0\rangle$, $\langle n = -y\rangle$, $\langle a = n\rangle$, $\langle \neg a \geq 0\rangle$	$y < 0 \wedge n = -y \wedge a = n \wedge \neg a \geq 0$

$\diamond$

Though the programs we consider are abstracted from their state handling part, we can prove some property depending of state. We can indeed use induction techniques provided we restrict to properties simple enough not to require invariant strengthening (see [3] for the general case in Lustre), as sketched below.

Example 5.5 Let us consider the program $\mathtt{P}_{ct}$ of Ex. 5.1. It implicitly involves a memory, say ξ_a, which memorizes the value of a from one instant to the next one. Recall that pa and a are synchronous. At any instant where pa (then a) is present, pa takes the current value of ξ_a (initially set to 0), then ξ_a is updated by the computed value of a; otherwise ξ_a remains unchanged. Proving "a is always positive" amounts to check that $\xi_a \geq 0$. Initially it is verified since $\xi_a = pa = 0$. Then we must show that, whenever a and pa are present, if ξ_a satisfies $\xi_a \geq 0$ (equivalently $pa \geq 0$) then so does its updated value (equivalently $a \geq 0$). Therefore we must check the instantaneous property expressed by $\widehat{a} \cap \widehat{pa} \subseteq \langle pa \geq 0 \Rightarrow a \geq 0\rangle$, which is achieved analogously to Ex 5.4. $\diamond$

6 Related work

As far as Lustre is concerned, *synchronous observers* [6] were proposed to verify a safety property `Q` (written in Lustre and characterised by a boolean signal `B`) on a program `P`: it is checked that no state of `P|Q` makes `B` false. The model-checking of Sect. 5.3 follows this approach. Nevertheless the problem of absence handling is most often avoided in Lustre. Most verification methods apply to an interpreted automaton generated by the compiler [7] possibly tuned to check a specific property. Anyway due to its functional aspect, the Lustre semantics can be given in terms of streams of values without "absent" value (see for example the analysis of [3] based on proofs by induction).

On the contrary, all analyses of Signal programs cope with absence. Two main approaches are used: use of clocks vs explicit encoding of absence. Clocks are used in the *clock calculus* performed by the compiler [1]. It runs on an *abstraction by synchronizations*, which encodes the boolean combinational part of a program, plus synchronization relations (see [9]). It involves clocks of signals plus boolean conditions of filterings. For example the abstraction of the program $\mathtt{P}_{abs}$ in Ex. 5.1 could be represented by the $\mathcal{CL}$ formula:

$$\widehat{p} \equiv \langle c_1\rangle \wedge \widehat{n} \equiv \langle c_2\rangle \wedge \widehat{a} \equiv \widehat{p} \cup \widehat{n} \wedge$$
$$\langle c_1\rangle \cap \langle \neg c_1\rangle \equiv \mathbb{O} \wedge \langle c_1\rangle \cup \langle \neg c_1\rangle \equiv \widehat{y} \wedge \langle c_2\rangle \cap \langle \neg c_2\rangle \equiv \mathbb{O} \wedge \langle c_2\rangle \cup \langle \neg c_2\rangle \equiv \widehat{y}$$

where c_1 and c_2 are free additional boolean variables. This restricted $\mathcal{CL}$ is isomorphic to the propositional calculus (in the spirit of Sect. 3.4) and the decision procedure corresponds to a boolean problem. Consequently it cannot be proved for this example that "a is always positive", nor for example that n and p are exclusive (which is a property of interest for the synthesis of control). Our $\mathcal{CL}$ is strictly more expressive since it can encode the whole combinational part of programs, even non-boolean.

Works that encode absence by a special value address both the combinational and state handling part of programs. Model-checking of boolean programs is presented in [8]. It encodes values true, false and absent by 1, -1 and 0 respectively, and symbolically encodes a program by a polynomial on the field $\mathbb{Z}/3\mathbb{Z}$. Computations are efficiently performed on BDD-like structures which represent the three valued logic. Properties involving non-boolean values cannot be proved. [2] addresses systems that contain only linear relations between non-boolean variables. All properties provable using $\mathcal{CL}$ can be proved. A normalization stage "solves" the combinational part of programs: it computes the valuations that satisfy it. More precisely, maximal sets of "synchronous" valuations (i.e. which agree on the status of each variable) are symbolically represented by a list of absent signals plus a polyhedron representing a relation over (present) values. There is a clear correspondence between absences indicated by lists and our distributions, as well as between a polyhedron and our relation R_δ. Our work provides another clean formalization of the implemented normalization, admitted to be naive in [2].

7 Conclusion

In the synchronous data-flow paradigm, clocks are sets of instants mainly used to indicate when signals are present in specifications, that is when a value can be used or must be computed when describing the control of systems. The language $\mathcal{CL}$ presented in this work extends the usual definition of clocks so that they can describe a relation between values of present signals. Hence relations between clocks can describe the whole combinational part (i.e. without state) of a system. They can also describe combinational safety properties.

A great advantage of clocks is that handling values of signals only at instants of their clock avoid to handle explicitly a special "absent" value. We illustrate it by our decision procedure and the related model-checking: they use plain boolean reasoning and satisfiability of some first order formulas. Indeed, the boolean abstraction of $\mathcal{CL}$, like the abstraction by synchronizations, allows us to encode clocks of signals into the propositional calculus. We extend the approach so that once present signals are determined in a reaction, the relation satisfied by their values can be retrieved.

So far, our contribution is rather theoretical because of the prohibitive complexity of the decision procedure. Further work can be done to practical applications. The main idea is to define a representation for the combinational

part of a program which allows to reason at various abstract levels, in the philosophy of the Signal environment (see [9]), both for the control and the data part of programs. Such a representation, structured like [1] by clock inclusions, could be used for compilation and verification purposes (in the spirit of the decision procedure for the interaction between boolean clocks and relations between present values, to be compared with [2]).

References

[1] Pascalin Amagbégnon, Loïc Besnard, and Paul Le Guernic. Implementation of the data-flow synchronous language SIGNAL. In *Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation (PLDI)*, pages 163–173, La Jolla, California, 18–21 June 1995.

[2] F. Besson, T. Jensen, and J.P. Talpin. Polyhedral analysis for synchronous languages. In A. Cortesi and G. Filé, editors, *Static Analysis*, volume 1694 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 1999.

[3] C. Dumas and P. Caspi. A PVS proof obligation generator for Lustre programs. In *7th International Conference on Logic for Programming and Automated Reasoning*, volume 1955 of *Lecture Notes in Artificial Intelligence*, 2000.

[4] N. Halbwachs. *Synchronous Programming of Reactive Systems.* Kluwer Academic Publishers, 1993.

[5] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[6] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*, Twente, June 1993. Workshops in Computing, Springer Verlag.

[7] N. Halbwachs and P. Raymond. Validation of synchronous reactive systems: from formal verification to automatic testing. In *ASIAN'99, Asian Computing Science Conference*, Phuket (Thailand), December 1999. LNCS 1742, Springer Verlag.

[8] M. Le Borgne, H. Marchand, E. Rutten, and M. Samaan. Formal verification of signal programs: Application to a power transformer station controller. In *Proceedings of the Fifth International Conference on Algebraic Methodology and Software Technology AMAST'96*, pages 271–285, Munich, Germany, July 1996. Springer-Verlag, LNCS 1101.

[9] Mirabelle Nebut and Sophie Pinchinat. A framework to analyse synchronous data-flow specifications. Technical Report 1402, Irisa, nov 2001.

[10] P. LeGuernic, T. Gautier, M. LeBorgne, and C. LeMaire. Programming real-time applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1321–1336, 1991.