A Decidable Clock Language for Synchronous Specifications

Mirabelle Nebut and Sophie Pinchinat *

Submitted to SLAP'02

Abstract. Presence and absence of signals inside a reaction are inherent to the synchronous paradigm, as well as clocks which are sets of instants that indicate when a given condition is fulfilled over a sequence of reactions (e.g. when a signal is present). Clocks are essential to capture the control in data-flow specifications; more generally relations between clocks should be analyzed to detect e.g. inconsistencies and verify some properties. These relations express particular safety properties many of which can be verified without considering the dynamic of systems, by means of a static abstraction. We propose a language $\mathcal{CL}$ to describe such properties and prove it decidable. Finally, model-checking is derived for e.g. SIGNAL programs, on the basis of a translation from the static abstraction of SIGNAL into $\mathcal{CL}$.

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

1 Introduction

Synchronous languages [6, 11, 8, 3] have been proposed to specify reactive systems, which interact continuously with the environment they are connected to. The imperative, state-based and data-flow programming paradigms both involve the notion of logical instants and presence/absence of signals. It is particularly apparent in imperative languages like ESTEREL [3], well adapted to problems for which the control-handling aspects are prevalent. A signal of type pure event is only an "impulse": the information it carries is limited to its *status*, that is its presence or absence (valued signals also exist). Basic instructions specify that the system broadcasts signals (`emit S`) or reacts to the presence or absence of signals (e.g. `present S then` p `else` q). The equational data-flow languages like SIGNAL [11] and LUSTRE [8] are better adapted to problems where data-flow is

*IRISA – Campus universitaire de Beaulieu – 35042 Rennes Cédex – France – ESPRESSO group mnebut@irisa.fr and S4 group pinchina@irisa.fr

prevalent. Specifications are systems of equations which describe the possible values carried by signals along the time. At a given instant, a signal (e.g. of type boolean, integer, etc) can be absent or present, hence carrying a significant value. The set of instants at which it is present is called its *clock*.

In the synchronous paradigm models of executions of the system are sequences of instantaneous reactions along a logical time line, which attach a status to each signal (and possibly a value, accordingly). Such an assignment is a *valuation*. For data-flow programs, these time-indexed sequences of valuations are generally called *flows*, or traces. A program is then associated a particular set of possible flows, called a *process* here.

Contrary to imperative languages, data-flow specifications do not make the control explicit: hence it must be synthesized. The powerful notion of *clocks* is essential for this purpose since clocks are *sets of instants* that characterize the instants *when* particular conditions hold: e.g. when a signal carries a significant value, when the sum of two significant values of signals is positive, or more generally when an input has to be read, when a particular computation has to be performed, etc.

Clocks are handled differently in LUSTRE and SIGNAL. In LUSTRE a basic clock is attached to the program, from which "slower" clocks are *functionally* extracted by means of boolean flows. More generally in SIGNAL equations state *relations* between clocks, among which the extraction principle of LUSTRE. Then in the case of LUSTRE it is quite easy to derive an order for computations which meets dependencies between variables, while the larger SIGNAL expressiveness makes necessary a deduction mechanism and a real synthesis. Nevertheless in both cases and even in the whole synchronous paradigm, the analysis of relations between clocks allows to obtain code of higher quality and e.g. to detect inconsistencies. Therefore powerful techniques to automate deductions and more generally perform verifications need to be developed.

Relations between clocks express *instantaneous* properties, which describe only what happens inside a reaction, with no reference to the preceding or following ones. E.g. "the level will never exceed 10" (for all instant t, *level* at t is lower than 10), or "each time the tank becomes empty, then the faucet is turned on" (for all instant t, if *level* at t is equal to 0 then *faucet* is present) are instantaneous properties, which fall into the class of *safety* properties. On the contrary, and as expected, liveness properties or properties that involve references to more than a single instant such as "the value of x increases from one instant to the next one" are not instantaneous properties.

Standard approaches to establish safety properties rely on operational models which take account of the dynamic of systems. It is also the case for instantaneous properties, e.g. the fact that "the level will never exceed 10" likely strongly depends on the evolution of the signal *level* along the time. Nevertheless many of them can be verified while abstracting from dynamic aspects, but instead focusing on static features (obtained by a *static abstraction* of programs).

The present work proposes a language called $\mathcal{CL}$ (read "clock language") to express instantaneous properties of processes (that hold in all possible flow of the process). Basically, the formulas of the language express inclusions between clocks inside flows, defined in Sect. 2. Provided clocks are defined using a decidable theory, the language $\mathcal{CL}$, presented in Sect. 3, is proved decidable and a decision procedure is given in Sect. 4. Moreover, Sect. 5 shows in the case of SIGNAL that the static abstraction of data-flow specifications translates naturally in $\mathcal{CL}$: model-checking can then be easily derived from the decision algorithm (in this context the $\mathcal{CL}$ language as well as its decision procedure strictly generalize existing works on SIGNAL, where clocks are defined using only a boolean theory, see e.g. [1]). Conclusion and perspectives are given in Sect. 6.

2 Synchronous Data-flow Semantics

We assume given a finite set of variables S of domain D (e.g. boolean, integer, etc). 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*[1] V on A denotes a snapshot of execution: 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 $\mathcal{V}_A$. The valuation which maps every variable of A to $\star$ is the *silent* valuation ${}^{\star}V_A$. A non-silent valuation is said *significant* or *concrete*. The *restriction* of $V \in \mathcal{V}_A$ to $A_1 \subseteq A$ maps any $x \in A_1$ to $V(x)$.

A *trace* T on A is an infinite sequence of valuations: $T : \mathbb{N} \to \mathcal{V}_A$. It defines an execution of the system. For $t \in \mathbb{N}$, a valuation $T(t) = {}^{\star}V_A$ indicates an inactivity of the system; it is said to *stutter*. A *flow* is a particular trace in which stuttering is left implicit: no silent valuation precedes a significant one. 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 sets of instants for which x is present (or *switched on*) in $F(t)$: $\forall t \in \mathbb{N},\ t \in \widehat{x}_F$ iff $F(t)(x) \neq \star$. A natural pre-order $\subseteq_F$ 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 property : $\forall t \in \mathbb{N},\ F(t)(x) \neq \star$ implies $F(t)(y) \neq \star$. It can be verified by examining the set of all valuations that occur at some instant in F, denoted by $[\![F]\!]_{\mathcal{V}_A} \triangleq \{V \in \mathcal{V}_A \mid \exists t \in \mathbb{N},\ V = F(t)\}$.

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 element are $\varpi, \varpi_1, \varpi_2$. 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 $A_1 \cap A_2$. The *static abstraction* ϖ_S of a process $\varpi \in \mathcal{P}_A$ contains any flow composed of valuations that can occur in ϖ: $\varpi_S = \{F \in \mathcal{F}_A \mid F(t) \in \bigcup_{F' \in \varpi} [\![F']\!]_{\mathcal{V}_A}\}$. $\varpi \in \mathcal{P}_A$ is *static* if $\varpi = \varpi_S$.

We refer to [10] for further details.

[1] Valuations are sometimes called *instantaneous events*.

3 A Clock Language

The clock language $\mathcal{CL}$ describes instantaneous properties of processes, or flows of the process. As expected, it expresses inclusions between sets of instants in a flow, defined by clock terms like $\widehat{x}$ seen in Sect. 2, but also by predicates of a first order theory that are relations between variables. The semantics of instantaneous properties (true of all valuations of a flow) enables to interpret $\mathcal{CL}$ formulas over valuations. $\mathcal{CL}$ 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

We focus here on notations needed to describe general predicates that involve symbols of constant, variable and predicate. We assume given a first order language $\mathcal{L}$ with equality whose set of symbols of variable contains S. The set of formulas of $\mathcal{L}$ is denoted by $\mathcal{F}(\mathcal{L})$, whose typical elements are R, R_1, R_2 (e.g. 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 (e.g. D is $\mathbb{Z}$, $+$ is the addition over integers). Given an interpretation function $l : S \to D$ for variables in S, 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[2]:

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

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 $on_V(.)$ 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:*

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

- *not* $on_V(\mathbb{O})$*;*
- *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* $(V(x_1), \ldots, V(x_n)) \models\!\approx R$*;*

$on_V(h_1\ op\ h_2)$ *is defined by:*

- $on_V(h_1 \cap h_2)$ *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 3.1 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 $\langle x > 0 \vee y > 0\rangle$ is switched off, since $\widehat{y}$ is switched off (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:

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

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 simply write $h_1 \equiv h_2$ for $h_1 \subseteq h_2 \wedge h_2 \subseteq h_1$[3], 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 3.2 *Given* $A \subseteq S$*,* $V \in \mathcal{V}_A$*,* $h_1, h_2 \in \mathcal{CT}_A$*,* $x \in S$ *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* $V' \in \mathcal{V}_A$ *s.t.* $V' \models \phi$ *and* $\forall y \in A \setminus \{x\}, V'(y) = V(y)$*;*
- $V \models \neg\phi$ *iff* $V \not\models \phi$*.*

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

[3] The new notation $\equiv$ is introduced to denote equivalence of clocks, since $=$ is kept to denote equality of the first order language $\mathcal{L}$.

As expected, the interpretation of clock formulas can also be lifted to flows, as follows: 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 $\Upsilon(.)$ (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 the 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 $V {\models\!\!\approx} R$ when all variables in $fv(V)$ are present in V and carry values that satisfy R.

Example 3.2 $\Upsilon(x < 10) = \Upsilon(\ \exists z.(z \geq 0 \wedge x < 10 + z)\ ) = \widehat{x}$ and for V as in Ex 3.1 where $V(x) = 3$ and $V(y) = \star$, it is true that $V {\models\!\!\approx} x < 10$, just as $V {\models\!\!\approx} \exists z.(z \geq 0 \wedge x < 10 + z)$. On the contrary, $\Upsilon(x > 0 \vee y > 0) = \widehat{x} \cap \widehat{y}$ and it is not the case that $V {\models\!\!\approx} x > 0 \vee y > 0$. $\diamond$

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

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

Therefore, for any valuation V, $on_V(\langle R \rangle)$ *implies* $on_V(\Upsilon(R))$, namely $\models \langle R \rangle \subseteq \Upsilon(R)$; also it can be proved that $\models \Upsilon(R) \equiv \langle R \rangle \cup \langle \neg R \rangle$, as well as $\models \langle R_1 \rangle \cap \langle R_2 \rangle \equiv \langle R_1 \wedge R_2 \rangle$, etc. Such tautologies are very useful to compute on clocks, and could address axiomatization issues, which is out of the scope of this paper.

Example 3.3 The $\mathcal{CL}$ language formulas can express various fine instantaneous properties. Assume given $x, y \in S$. "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 $\Upsilon(x \geq 0) = \widehat{x}$ hence $\models \langle x \geq 0 \rangle \subseteq \widehat{x}$. "$y$ is always equal to x" can be 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 $\widehat{x} \subseteq \langle \exists y. x = 2y \rangle$, and highlights the need for quantification provided by $\mathcal{L}$. $\diamond$

The $\mathcal{CL}$ language provides the negation of a clock formula (useful in Sect. 5.3), as a meta negation when $\mathcal{CF}$ is interpreted over valuations but not when interpretation is lifted to flows: for $\phi \in \mathcal{CF}$, $F \models \neg\phi$ iff $\forall V \in [\![F]\!]_{\mathcal{V}_S}, V \models \neg\phi$, which differs from $F \not\models \phi$ ($\exists V \in [\![F]\!]_{\mathcal{V}_S}. V \not\models \phi$).

3.4 Boolean Abstraction

Let us give here a few notations for the propositional calculus $\mathbf{B}$. 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 the full propositional calculus in the standard way (e.g. $\delta(G_1 \wedge G_2) = 1$ iff $\delta(G_1) = 1$ and $\delta(G_2) = 1$). Δ denotes the set of distributions on $\mathbf{b}$, and $\mathbf{0}$ denotes the distribution mapping all variables to 0. For $\delta \in \Delta$ and $G \in \mathbf{B}$, we write $\delta \models G$ whenever $\delta(G) = 1$.

$\mathcal{CL}$ can be *abstracted* into $\mathbf{B}$, where $\mathbf{b}$ contains a variable $b_{\widehat{x}}$ (resp. $b_{\langle R \rangle}$) for each $\widehat{x}$ (resp. $\langle R \rangle$) in $\mathcal{BCT}$. The abstraction is a mapping B which maps $h \in \mathcal{CT}$ (resp. $\phi \in \mathcal{CF}$) to h_B (resp. ϕ_B) instead of $B(h)$ (resp. $B(\phi)$) as follows:

Definition 3.3 (Boolean Abstraction) *The mapping B is defined by induction on the structure of elements in $\mathcal{CT}$ and $\mathcal{CF}$ on Fig. 1, where $x \in S$ and $\exists b_{\langle R_x \rangle}.\phi_B$ is written for $\exists b_{\langle R_1 \rangle}.\ldots.b_{\langle R_n \rangle}\phi_B$ in which $\{R_i\}_{i=1..n}$ is the set of all relation-clocks $\langle R \rangle \in BCT(\phi)$ s.t. $x \in fv(R)$.*

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

$\phi \in \mathcal{CF}$	$\phi_B \in \mathbf{B}$
$h_1 \subseteq h_2$	$h_{1_B} \Rightarrow h_{2_B}$
$\neg\phi$	$\neg\ \phi_B$
$\exists x.\phi$	$\exists b_{\widehat{x}}.b_{\langle R_x \rangle}.\phi_B$
$\phi_1 \wedge \phi_2$	$\phi_{1_B} \wedge \phi_{2_B}$

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

We now study properties of models of concrete and abstract quantifier-free formulas. We first define the *abstraction* of a valuation V as the distribution $\delta_V \in \Delta$ which indicates whether variables are required to be present or absent in V, and which formulas $R \in \mathcal{F}(\mathcal{L})$ are required to be satisfied by the present ones, for $\langle R \rangle \in BCT(\phi)$.

Definition 3.4 (Abstraction) *Let $V \in \mathcal{V}_S$. $\delta_V \in \Delta$ is defined by:*

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

Remark 3.1 It can be proved that Def. 3.4 delivers indeed a distribution thanks to the properties of the predicate $on_V(.)$. $\diamond$

Theorem 3.1 *For all $\phi \in \mathcal{CF}^{qf}{}_A$ and $V \in \mathcal{V}_A$, $V \models \phi$ iff $\delta_V \models \phi_B$.*

The proof of Th. 3.1 is by induction over the structure of ϕ, see [10] as well as for other proofs omitted here.

Reciprocally, we define the *concretization* of a distribution as the set of all valuations that assign values in $D \cup \{\star\}$ to variables, *when possible*, in accordance with: on the one hand the requirements of presence/absence specified by the distribution, and on the other hand the satisfaction of some formulas in $\mathcal{F}(\mathcal{L})$ imposed by the values of variables $b_{\langle R \rangle}$. Formally,

Definition 3.5 (Concretization) *Given $\delta \in \Delta$, the concretization $\mathcal{V}_\delta$ of δ is defined by $\mathcal{V}_\delta \subseteq \mathcal{V}_S$:*

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

Example 3.4 If $\mathbf{b}$ is $\{b_{\widehat{x}}, b_{\langle x<0\rangle}, b_{\langle x>3\rangle}\}$ and $\delta(b_{\widehat{x}}) = \delta(b_{\langle x<0\rangle}) = \delta(b_{\langle x>3\rangle}) = 1$ then $\mathcal{V}_\delta = \emptyset$. $\diamond$

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

Theorem 3.2 *Let α and γ be the following mappings:*

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

then the pair (α, γ) is a Galois insertion.

4 A Decision Procedure

In this section we define a decision procedure for the satisfiability of a clock formula $\phi \in \mathcal{CF}$. It answers YES if ϕ is satisfiable: there exists a non-trivial flow F (that contains at least a non-silent valuation) such that $F \models \phi$; and NO otherwise. Since satisfaction for flows relies on satisfaction for all valuations of a flow, there exists a flow which satisfies ϕ iff there exists a valuation which satisfies ϕ. The procedure proceeds as follows: the boolean abstraction ϕ_B of ϕ is computed and leads to a finite number of decision problems in the theory $\mathcal{F}(\mathcal{L})$. It is checked for each model of ϕ_B whether its concretization is not empty, hence contains a model of ϕ. Section 4.1 gives technical materials for the decision procedure which is discussed in Sect. 4.2.

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 have the same models.

4.1 The Satisfiability Problem

We now study the satisfiability problem for completed clock formulas. The key point exploits Th. 3.2, the Galois insertion, which entails the following:

Corollary 4.1 *For all $\phi \in \mathcal{CF}^{qf}{}_A$ and for all $V \in \mathcal{V}_A$, $V \models \phi$ iff $V \in \bigcup_{\delta\models\phi_B} \mathcal{V}_\delta$.*

Corollary 4.1 states that if some $V \models \phi$, then $V \in \mathcal{V}_\delta$ for some $\delta \models \phi_B$, and vice versa. For each $\delta \models \phi_B$, we establish a criterion for the non-emptiness of $\mathcal{V}_\delta$, which relies on the satisfiability of a first order formula R_δ built up from ϕ and δ (see Th. 4.1).
In the rest of this section, we assume given a completed formula $\phi \in \mathcal{CF}^{qf}{}_A$.

Definition 4.1 *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 4.1 *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

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

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

Theorem 4.1 gives a criterion for the non-emptiness of $\mathcal{V}_\delta$. Finally its corollary (Cor. 4.2) states that the satisfiability of ϕ is fully determined by the pairs (δ, R_δ), where $\delta \models \phi_B$.

Theorem 4.1 *Let $\delta \models \phi_B$. Then, $\mathcal{V}_\delta \neq \emptyset$ iff "$\mathcal{R}_\delta = \emptyset$ or else R_δ is satisfiable". Moreover $\forall V \in \mathcal{V}_\delta$, $V \models\approx R_\delta$.*

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. 4.1, then $on_V(\langle R\rangle)$. Therefore $V \models\approx R$ 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:

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

Notice that W is well defined since by Lemma 4.1, the three cases of Eq. (3) are disjoint. We prove now that $W \in \mathcal{V}_\delta$, that is, for all $h \in BCT(\phi)$,

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

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

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

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. 4.1 and Lem. 4.1. 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. 4.1: 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 4.2 *ϕ 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. 4.2 induces a decision procedure for $\mathcal{CL}$. The algorithm of Fig. 2 takes $\phi \in \mathcal{CF}^{qf}$ as input, and returns YES if ϕ is satisfied by a valuation $V \neq {}^\star V_A$. Otherwise it returns NO. Note that a quantified formula $\exists x.\phi$ is satisfiable iff ϕ is satisfiable.

```
Let DIST = {δ ∈ Δ | δ ⊨ φ_B and δ ≠ 0} in
For all δ ∈ DIST do
  if 𝓡_δ = ∅ then return YES else
     if R_δ is satisfiable then return YES;
return NO.
```

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

Because the set $DIST$ is finite, the algorithm terminates. It is correct by Cor. 4.2. The decision procedure above can be improved in order to effectively exhibit a model of ϕ, provided a constructive decision procedure exists for $\mathcal{L}$: the definition of the valuation W in the proof of Th. 4.1 returns a model of ϕ.

Complexity issues can be briefly answered as follows. Define $|\phi|$ as the number of all symbols occuring in ϕ, included the symbols that appear 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 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

In this section we derive from the decision procedure a model-checking algorithm for 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. Then model-checking is discussed in Sect. 5.3.

5.1 The Synchronous Data-flow Language Signal

A SIGNAL [11, 2] program on a set of variables A is written $\mathtt{P} \in \mathcal{P}g_A$; it denotes a process $[\![\mathtt{P}]\!] \in \mathcal{P}_A$. The operator of parallel composition `|` corresponds to the parallel composition of processes. The kernel language contains four elementary equation patterns whose semantics is given on Fig. 3 as follows: for $exp(F,t)$ an expression built on F and t, $[\![\mathtt{P}]\!] = \{F \in \mathcal{F}_A \mid \forall t \in \mathbb{N},\ exp(F,t)\}$ is written $\mathtt{P} \leadsto exp(F,t)$ and $F(a)(t)$ is written a_t.

P		$[\![\mathtt{P}]\!]$
(a) `y := g(x1, ... ,xn)`	$\leadsto$	$y_t = \star \Rightarrow x_{1_t} = \cdots = x_{n_t} = \star$ $\bigwedge y_t \neq \star \Rightarrow\ \forall i, x_{i_t} \neq \star \wedge y_t = g(x_{1_t}, \ldots, x_{n_t})$
(b) `y := x $1 init v0`	$\leadsto$	$y_t = \star \Leftrightarrow x_t = \star$ $\bigwedge (x_0 \neq \star \Rightarrow y_0 = v_0)\ \bigwedge\ (t \geq 1 \Rightarrow y_t = x_{t-1})$
(c) `y := x when c`	$\leadsto$	$c_t \neq \mathit{true} \Rightarrow y_t = \star$ $\bigwedge c_t = \mathit{true} \Rightarrow y_t = x_t$
(d) `y := u default v`	$\leadsto$	$u_t \neq \star \Rightarrow y_t = u_t$ $\bigwedge u_t = \star \Rightarrow y_t = v_t$

Fig. 3: Equations of the kernel of SIGNAL

Equation (a) defines the value of y as the result of applying the mapping g to the values of variables x_i, and constrains all arguments to be synchronous. E.g if g is addition $\star + 2$ is irrelevant. Functions allow the construction of monochronous predicates (e.g. `c := x<y` builds the relation $x < y$, and c, x and y are synchronous). Equation (b) defines y as the *delay* of x and induces dynamic features: y and x have to be synchronous and when present y takes the last non-$\star$ value of x, initially v_0. Equation (c) performs a filtering: if c is present and *true* then y takes the value of x, otherwise y is absent. Equation (d) performs a merge with priority given to the first argument u: if u is present then y takes the value of u, otherwise y is v. Restriction is provided: `(P / x)` means that `x` is hidden in `P`, which corresponds to the restriction of $[\![\mathtt{P}]\!]$ to variables in $A \setminus \{x\}$. The language also provides derived operators (e.g. the operator `^=` constrains its arguments to be synchronous) and powerful constructions for modular programming we will not deal with here.

Example 5.1 Figure 4 shows on the left SIGNAL equations defined on the set of integer variables $A = \{a, y, p, n\}$, such that the value of a is the "absolute value" of the value of y. Equations on the right defined on $A = \{a, ma, N\}$ describe a "counter": a decrements and reinitializes itself after it has reached 0, with the positive input variable N. Since the variable ma carries the previous value of a (initially 0), N is synchronized with instants where $ma = 0$. Note that the absolute value denotes a static process, contrary to the counter. $\diamond$

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

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

Fig. 4: SIGNAL examples

The *static abstraction* of a program P, denoted by $\mathtt{P}_{\mathcal{S}}$, corresponds to the "static closure" of $[\![\mathtt{P}]\!]$. $\mathtt{P}_{\mathcal{S}}$ is easily obtained: in the case of a non-delay equation, $\mathtt{P}_{\mathcal{S}}$ is just P. `y := x $1 init v0` is abstracted into `y ^= x`, which only states that y and x are synchronous. The abstraction is modular: $(\mathtt{P}_1 \mid \mathtt{P}_2)_{\mathcal{S}}$ is $\mathtt{P}_{1\mathcal{S}} \mid \mathtt{P}_{2\mathcal{S}}$. The kernel of *static* SIGNAL (written $\mathcal{S}$SIGNAL) contains Eq. (a), (c) and (d), without the delay.

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. 5.1. Note that synchronizations translate directly into clock formulas: e.g. `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 := u default v`	$\widehat{y} \equiv \widehat{u} \cup \widehat{v} \ \wedge\ \widehat{u} \subseteq \langle y = u\rangle \ \wedge\ \widehat{v} \setminus \widehat{u} \subseteq \langle y = v\rangle$
`( 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 5.1 *For all $\mathcal{S}$SIGNAL program P and $V \in \mathcal{V}_S$,*

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

Note that not all properties expressed by $\mathcal{CL}$ can be expressed in SIGNAL, which does not provide negation. As a consequence SIGNAL can express neither that a clock is not empty, nor that a clock inclusion is strict.

Example 5.2 We give on Fig. 6 the two clock formulas ϕ_{abs} and ϕ_{ct} that represent the two SIGNAL specifications of Ex. 5.1. $\diamond$

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

(a) ϕ_{abs}

$$\begin{array}{l} \widehat{a} \equiv \widehat{ma} \equiv \langle a = N \rangle \cup \widehat{a} \\ \wedge\ \langle ma = 0 \rangle \equiv \langle a = N \rangle \equiv \widehat{N} \equiv \langle N \geq 0 \rangle \\ \wedge\ \widehat{ma} \setminus \widehat{N} \subseteq \langle a = ma - 1 \rangle \end{array}$$

(b) ϕ_{ct}

Fig. 6: Clock Formulas for Ex. 5.1

5.3 Model-checking

We show here how to check whether a SIGNAL program P satisfies an instantaneous property $\phi \in \mathcal{CF}^{qf}$, thanks to the decision procedure. We aim at answering the problem "does any flow $F \in [\![\mathtt{P}]\!]$ satisfies ϕ ?", which reduces to "does any valuation $V \in [\![F]\!]$ for all $F \in [\![\mathtt{P}]\!]$ satisfies ϕ ?", and in turn, by using the translation from $\mathcal{S}$SIGNAL to $\mathcal{CL}$, to "$\models \phi_{\mathtt{P}_\mathcal{S}} \Rightarrow \phi$?". The decision procedure for $\mathcal{CL}$ is then used to answer the satisfiability of the formula $\phi_{\mathtt{P}_\mathcal{S}} \wedge \neg\phi$ (in which quantifiers are pushed out, e.g. by renaming techniques). As expected, since an abstraction of P is used, if the answer is "NO" then P satisfies ϕ, otherwise nothing can be inferred.

Example 5.3 Let us consider the absolute value given on Ex. 5.1 and prove that "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 $\phi_{abs} \wedge \neg\phi$. Three distributions δ_1, δ_2, δ_3 satisfy ϕ'_B. They differ depending on whether they "switch on" $\widehat{p}$ (resp. $\widehat{n}$) or not. We show that none of them corresponds to a concrete valuation. Say δ_1 switches on clocks $\widehat{p}$ and $\widehat{n}$, then also $\langle y \geq 0 \rangle$ and $\langle y < 0 \rangle$. Therefore the exclusive conditions $y \geq 0$ and $y < 0$ belong to $\mathcal{R}_{\delta_1}$, then R_{δ_1} is not satisfiable. Hence $\mathcal{V}_{\delta_1} = \emptyset$. δ_2 switches off $\widehat{n}$ and switches on $\widehat{p}$, hence also $\langle y \geq 0 \rangle$, $\langle p = y \rangle$, $\langle a = p \rangle$ (see ϕ_{abs}) and $\langle \neg y < 0 \rangle$ and $\langle \neg a \geq 0 \rangle$ (due to completion); therefore R_{δ_2} is not satisfiable, hence $\mathcal{V}_{\delta_2} = \emptyset$. The reasoning is symmetrical for δ_3 which switches on $\widehat{n}$ and switches off $\widehat{p}$. Thus the property is proved. $\diamond$

Example 5.4 Let us consider the counter given on Ex. 5.1. It is easy to prove like in Ex. 5.3 that "N is always positive". Similarly we prove the following inductive property: it is always true that $0 \leq ma \leq N$. Initially, it is verified since $ma = 0$. We then prove the induction step, that is if $0 \leq ma \leq N$ then $0 \leq a \leq N$ (recall that ma takes the past value of a). It is expressed by the clock formula ϕ: $\langle 0 \leq ma \leq N \rangle \subseteq \langle 0 \leq a \leq N \rangle$. Let us denote by ϕ' the completion of $\phi_{ct} \wedge \neg\phi$. δ_1, δ_2, δ_3 satisfy ϕ'_B. Both switch on $\widehat{a}$, $\widehat{ma}$, $\langle 0 \leq ma \leq N \rangle$ and $\langle \neg 0 \leq a \leq N \rangle$ (due to completion). δ_1 and δ_2 also switch on $\langle ma = 0 \rangle$ and $\langle a = N \rangle$ (reinitialization of the counter) therefore R_{δ_1} and R_{δ_2} are not satisfiable. δ_3 corresponds to decrementation of the counter and switches on $\langle a = ma - 1 \rangle$ and $\langle \neg ma = 0 \rangle$ (due to completion), therefore R_{δ_3} entails the formula $\neg 0 \leq a \leq N \wedge 0 \leq a < N$ thus is not satisfiable. Thus the property is proved. $\diamond$

Most of the reactive systems specified in Signal use multiple theories: in this case $\mathcal{L}$ is multi-sorted and the formula R_δ must be split into sub-formulas to perform separate satisfiability tests. On the other hand a program does not use sophisticated theories for non-boolean variables, but e.g. the Presburger algebra or the additive theory of reals.

6 Conclusion and Perspectives

The present work defines a language $\mathcal{CL}$ based on clock relations, which sentences (or formulas), are naturally suitable for the synchronous paradigm. The $\mathcal{CL}$ language enables to express some safety properties, which describe something happening inside a single reaction. In particular, as expected, the static abstraction of synchronous data-flow specifications can be translated into $\mathcal{CL}$.

The decidability of $\mathcal{CL}$, i.e. the existence of an algorithm deciding whether a formula is satisfied by a non-trivial valuation, is proved, provided $\mathcal{CL}$ constructs (clocks) rely a decidable theory. The decision procedure relies on a safe boolean abstraction of $\mathcal{CL}$ formulas into propositional calculus formulas: it first determines whether a boolean distribution is a model of the abstract formula, and next if it can be made a concrete valuation from this distribution.

Finally, model-checking is derived for Signal programs and illustrated by two toy examples.

Related Work We only consider synchronous data-flow languages here, first Signal then Lustre.

The Signal clock calculus is an analysis performed on the static abstraction of programs, that synthesizes their control by analyzing relations between clocks. Nevertheless an *abstraction by synchronizations* is also performed (see [10]): basic clock terms are only clocks of variables and relation-clocks of the kind $\langle c \rangle$ where c is a boolean variable, thus this restricted $\mathcal{CL}$ is isomorphic to the propositional calculus (in the spirit of Sect. 3.4) and the decision procedure reduces to a boolean problem. Our $\mathcal{CL}$ induces a strict increase of expressiveness since relations between non-boolean variables are also taken into account. As shown in Sect. 5.3, their consideration is necessary to prove some properties.

[4] presents a reachability analysis of systems that contain only linear relations between non-boolean variables. A stage of the analysis called normalization handles the static part $\mathtt{P}_{\mathcal{S}}$ of programs: it can be seen as a synthesis of instantaneous properties. Nevertheless clocks are not used as objects of the analysis: in practice $\mathtt{P}_{\mathcal{S}}$ is transformed into a set of pairs composed of a list of absent variables and a polyhedron stating a relation between the present ones. There is a clear correspondence between absence of variables indicated by lists and distributions, as well as between a polyhedron and our relation R_δ. Our work therefore can be seen as a clean formalization of the normalization stage.

Numerous analyses and verification tools are dedicated to LUSTRE programs (e.g. Lesar [9] for model-checking, Lurette for test, and recently proofs using PVS). However models used are quite far from clocks, e.g. analyses apply to a control automaton obtained by partial evaluation of the boolean variables of the program [7]. In particular no significant analysis of the static part of programs is performed during the *clock verification* [5], the analogous of the clock calculus for SIGNAL. As shown by this document, the application of works on the $\mathcal{CL}$ language to LUSTRE is nevertheless theoretically possible.

Perspectives The decision procedure we give addresses first boolean aspects and next numerical ones. It proves that a mixing between "boolean" clocks and values is possible, but does not provide a real interaction between the boolean resolution and the numerical one. This interaction is needed in practical algorithms to simulate the interconnection between control and data, as much as possible. It is e.g. needless to perform a boolean reasoning that involves distributions which clearly do not correspond to a concrete model, as well as to perform too early expensive tests for the satisfiability of numerical formulas if no boolean model exists anyway. We currently investigate algorithmic techniques based on the construction of a graph whose nodes are clocks and edges denote the specified inclusion between clocks. Subsets of $\mathcal{F}(\mathcal{L})$ are associated to nodes, in such a way that if R is associated to h then $h \subseteq \langle R \rangle$. Therefore we can e.g. safely deduce from the non-satisfiability of R the emptiness of h.

Once such a structure is built in a bottom-up approach (with an accuracy to be determined), it should be exploited to solve various problems that concern the static part of processes. An immediate application would be the improvement of the clock calculus, leading e.g. to higher quality executable code. As a second application, it would worth modifying the normalization algorithm of [4] (admitted to be naive) to introduce clocks as in our approach, and look if it yields to better results.

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] Albert Benveniste, Paul Le Guernic, and Christian Jacquemot. Synchronous programming with events and relations: the SIGNAL language and its semantics. *Science of Computer Programming*, 16(2):103–149, September 1991.

[3] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, November 1992.

[4] 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.

[5] P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *Proc. of 14th ACM Conf. on Principles of Programming Languages*, pages 178–188. ACM Press, 1987.

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

[7] N. Halbwachs. About synchronous programming and abstract interpretation. In B. LeCharlier, editor, *International Symposium on Static Analysis, SAS'94*, Namur (belgium), September 1994. LNCS 864, Springer Verlag.

[8] 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.

[9] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow programming language lustre. *IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems*, September 1992.

[10] Mirabelle Nebut, Sophie Pinchinat, and Paul Le Guernic. A framework to analyse synchronous data-flow specifications. Technical Report 1402, Irisa, November 2001.

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