

Proving the Properties of Communicating Imperfectly-Clocked Synchronous Systems

Julien Bertrane

École Normale Supérieure, Paris, France
bertrane@di.ens.fr

Abstract. Our work aims at certifying that all the executions of several collaborating synchronous systems in a realistic environment follow a given specification. In order to analyze the numerous executions that may happen while considering a set of synchronous systems whose clocks are non-perfect and that communicate through non-instantaneous channels, we define two new abstract domains. The *Changes counting domain* and the *Integral bounding domain* gap the imprecisions of the previously defined Constraint domain that occur because of these hardware imprecisions. We define a reduced product between these domains that allows a much more precise though sound analysis than the three analyses that may have been defined in each domain.

1 Introduction

The design of critical embedded command systems often relies on the GALS (globally asynchronous, locally synchronous) paradigm. This means that several synchronous subsystems communicate with each other in order to compute the decisions, each one being timed by its own clock. These clocks may however desynchronize the one compared to the others. The reason for this design method is that the propagation of the information inside large systems is too long for it to be made of a single synchronous system. Furthermore, in order to prevent design and hardware errors, it is common to use several redundant units executing on different systems.

The designers have therefore to take this clock skew problem into consideration. Another consequence of such a design is that it implies communications between the synchronous systems, for example through buses. The communication delays between the subsystems cannot be neglected nor considered constant. The two temporal imprecisions caused by this hardware limitations make their analysis very difficult.

We introduce several abstract domains that interact and allow an abstract interpretation based analysis of the code of embedded systems built according to the previous hypotheses. Their goal is to try to prove that the software part of the system is robust enough to satisfy the specifications despite the hardware imprecisions.

Previous Works

The design of data processing systems for critical software is often based on the *Lustre* and *SCADE* synchronous programming frameworks. Their theoretical

foundations and properties were presented in [9,11] and [3,2,8]. We try to never choose between their syntax, and any example given may be easily written in both of them. The difficulties resulting from asynchrony and non-instantaneous communications for synchronous systems were also studied in [7]. A way to face these difficulties is presented in [1]: a protocol is defined that ensures the robustness of the system to clock desynchronization. Our studies doesn't aim at proposing robustness techniques but at proving them, even if they are hidden inside a huge amount of code. S. Thompson and A. Mycroft, proposed in [12] several abstractions to study asynchronous circuits that share some characteristics with the changes counting domain we introduce here : both describe signals through their value changes. We make an extensive use of the abstract interpretation theory and tools presented in particular of [6,5,10]. We presented a simpler analysis method in [4].

Sect. 2 presents the characteristics and the limits of the synchronous subsystems that we study and of the hardware they will be run on. We then introduce in Sect. 3 the semantics of the executions of these systems. In Sect. 4, we recall briefly the *Constraint domain*. Sect. 5 extends our abstract model by defining a new abstract domain, the *Changes counting domain* and considering a reduced product with the *Constraint domain*. This extension is continued in Sect. 6 with the definition of a third abstract domain, the *Integral bounding domain*, also interacting thanks to a reduced product with the two former ones. An example of the interaction between these domains is presented in Sect. 7. In Sect. 8, we give some details about an implementation of these domains and about future improvements.

2 Communications Between Non-perfect Systems

The systems we analyze are made of several units written in a synchronous language, either based on the *node* notion like **Lustre** or on a graphical representation like **SCADE**. In our examples, we use graphical descriptions, since it is easy to include the numerical bounds of the limitations of the hardware in them. In this work, we only consider boolean values and boolean operators.

First, the synchronous systems execute on non-perfect clocks. In order to model their imprecision, each system is connected to a time interval containing the delay between two consecutive ticks. In our example, systems are boxes containing a small box providing this interval. The two effects of a unit (buffering of inputs and computation of an output emitted at the next tick) may be considered separately, as we did in [4]. The buffering and periodic reading is computed by an operator called **DISCR** (depicted as `|||` in [4]). The computation and the release of its result is made by an operator called **SHIFT**.

These systems are connected through buses. The transfer of an information along a bus is non-instantaneous. Buses are thus timed by an interval specifying the minimum and the maximum duration of this communication. Of course, a value doesn't necessarily arrive in a system exactly at the time it gets used. Therefore, we assume that these systems are plugged on buffers that receive the

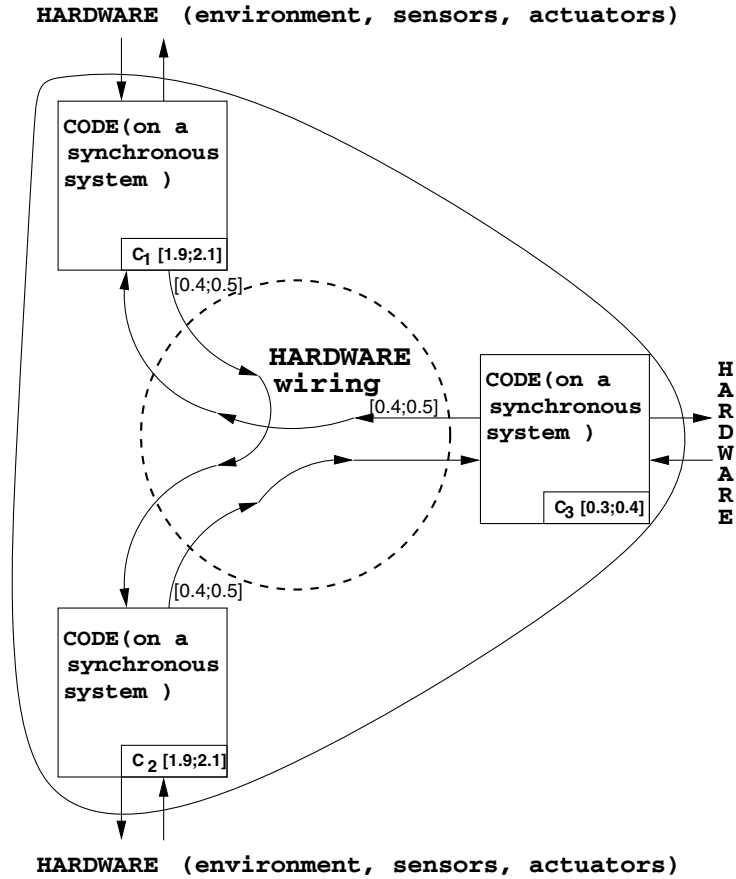


Fig. 1. Three synchronous systems, with bounds on their clock imprecisions and their communication delays

values transmitted by the bus. If a new value arrives in a buffer which has still not been read, then the new value replaces the older one. A bus is graphically represented by a wire and its delay interval.

Therefore, a diagram contains informations not only on the software but also on the hardware, as shown on Fig 1. It presents three synchronous systems connected with three clocks C_1 , C_2 and C_3 . C_1 and C_2 have the same characteristics : for these clocks, the length of a cycle between two ticks belongs to $[1.9; 2.1]$. In the case of C_3 , the cycle length is assumed to belong to $[0.3; 0.4]$. The communication buses are denoted by arrows connecting synchronous systems (inside the hardware wiring label). In a real system, we could have more complex wiring, for example each couple of systems may be connected in both sense (emission and reception). In that case, arrows may split to send the same information to several systems. The characteristics of all the buses in this example are the same : the communication lasts between 0.4 and 0.5 time units.

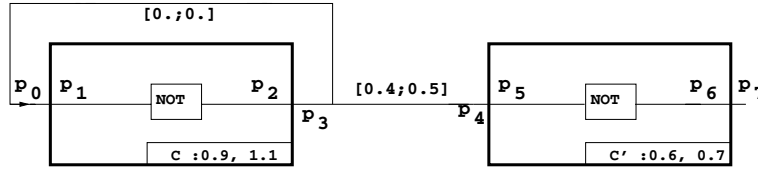


Fig. 2. A couple of communicating imperfectly clocked synchronous systems

3 Concrete Semantics of Communicating Non-perfectly Clocked Synchronous Systems

The concrete semantics of a communicating group of non-perfectly clocked synchronous systems is the precise description of all its possible behaviors. Indeed, because of the hardware imprecision, we cannot define **the** behaviour for a system, since it is not unique, even for a given input : we don't even know the precise time of each clock tick, but only an approximation of this time. We first define an equivalent to the classical notion of variable, since our representation does not provide names for each point of a program :

Definition 1 (Control point). *A control point is any input or output of an operator or of a synchronous system. The set of control points is denoted by P .*

We may now define informally the concrete semantics :

Definition 2 (Concrete semantics). *The concrete semantics of a communicating group of non-perfectly clocked synchronous systems S is the set of all its possible executions. It belongs to $\mathcal{P}(P \rightarrow (\mathbb{R}^+ \rightarrow \mathbb{B}))$, where \mathbb{R}^+ is the time (after the beginning of the execution).*

Since this set is not easy to handle, we immediately abstract it in the canonical way into an element of $P \rightarrow \mathcal{P}(\mathbb{R}^+ \rightarrow \mathbb{B})$. The function $\llbracket S \rrbracket \in P \rightarrow \mathcal{P}(\mathbb{R}^+ \rightarrow \mathbb{B})$ maps each control point p_i to the set $\llbracket S \rrbracket(p_i)$ of behaviors possible at point p_i . Each of these behaviors is described by a **signal** : a function v connecting each time t to the value $v(t) \in \mathbb{B}$ at control point p_i at the time t .

Now, we can incrementally build (see Fig. 3) the semantics (as soon as the parameters C, C' and the communication delays are chosen) of the system presented on Fig. 2. On this figure, t stands for *true*, f for *false*, and we let $C_0 = 0, C_1, C_2, C'_0 = 0, C'_1, C'_2$ as well as vertical dashed lines denote the first clocks ticks. The arrows show which already built part allows to build a new segment of the signals. For example, in the first step of the building of this solution, we use the initialization convention which expresses that before time $t = 0$, the signals are set to false. During the second step, we discover that between $C_0 = 0$ and C_1 , the signal at p_3 is set to false, since p_3 is simply the result of the previous cycle at point p_2 , which was false. The same occurs for p_6 and p_7 . The signal at p_4 is the delayed version of the one at p_3 . And each step extends the previous one.

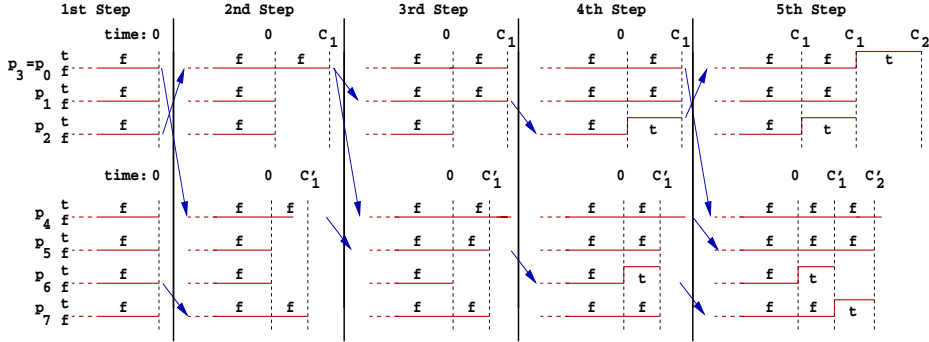


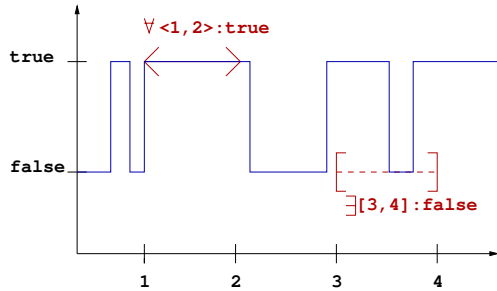
Fig. 3. The semantics of the diagram on Fig. 2

Studying this concrete semantics is very difficult because we cannot even describe **one** behavior at **one** control point, since it lasts an infinite time. *A fortiori*, describing infinitely many of them seems impossible, as well as manipulating them. We therefore make use of the abstract interpretation framework and define several abstract domains that are able of manipulating this “infinite” elements or sets, and prove their properties.

4 Constraint Domain

We defined in [4] an abstract domain based on the notion of temporal constraints. It allows the proof of some temporal properties and it is our first abstract domain, with only a few modifications. The two basic elements are :

- A constraint denoted by $\exists[a; b] : x$, meaning that any signal takes the value x at least once during the interval $[a; b]$.
- A constraint denoted by $\forall[a; b] : x$, meaning that any signal takes the value x during the whole interval $[a; b]$.



A signal satisfying two abstract constraints

These constraints and the formula built with them using the conjunction and disjunction operators may express many temporal properties. This abstract domain is denoted by \mathcal{C} . The connection between this abstract domain and the concrete semantics is maintained by a **concretization** function γ that connects any abstract formula to all the signals that satisfy it. In order to perform an analysis, we defined in [4] abstract operators that spread any information obtained in a control point to its neighbors, remaining in this abstract domain. For each

operator OP , we defined in [4] a backward abstract operator $\overleftarrow{\Psi}_{\text{OP}}^{\#}$. They are very intuitive : for example, $\overleftarrow{\Psi}_{\text{AND}}^{\#}(\forall\langle a; b \rangle : \text{True}) = (\forall\langle a; b \rangle : \text{True}, \forall\langle a; b \rangle : \text{True})$. The result is a couple of constraints since the AND operator has two inputs, and we are reasoning backwards. On the other hand, $\overleftarrow{\Psi}_{\text{AND}}^{\#}$ cannot deal precisely with the constraint $\forall\langle a; b \rangle : \text{False}$, which would require a non-countable infinity of disjunctions, and it doesn't gain in that case any information, returning top (\top), i.e. the absence of constraint.

We now also connect any time-independent (i.e. logical) operator with a trivial forward abstract operator $\overrightarrow{\Psi}^{\#}$ that simply mimics the execution of the operator : $\overrightarrow{\Psi}_{\text{AND}}^{\#}(\forall\langle 1; 3 \rangle : \text{True}, \forall\langle 2; 4 \rangle : \text{True}) = \forall\langle 2; 3 \rangle : \text{True}$.

Spreading abstract information from one control point to the other ones, we prove abstract properties of the communicating non-perfectly clocked synchronous systems. The abstract interpretation theory provides a simple way to prove that this propagation of the information is sound. For any gate performing a transformation Ψ , the abstract counterpart $\Psi^{\#}$ must satisfy :

$$\Psi \circ \gamma \subseteq \gamma \circ \Psi^{\#}.$$

All of the abstract operators we defined in [4] satisfy this condition.

Constraints Borders. We define four functions connecting a constraint to a real :

$$\begin{aligned} \text{right_border} &: \begin{cases} \forall\langle a; b \rangle : \text{False} \mapsto b \\ \exists[a; b] : \text{False} \mapsto b \end{cases} \\ \text{left_border} &: \begin{cases} \forall\langle a; b \rangle : \text{False} \mapsto a \\ \exists[a; b] : \text{False} \mapsto a \end{cases} \\ \text{right_influence_border} &: \begin{cases} \forall\langle a; b \rangle : \text{False} \mapsto b \\ \exists[a; b] : \text{False} \mapsto a \end{cases} \\ \text{left_influence_border} &: \begin{cases} \forall\langle a; b \rangle : \text{False} \mapsto a \\ \exists[a; b] : \text{False} \mapsto b \end{cases} \end{aligned}$$

`right_border` and `left_border` simply give the borders of the constraints. `right_influence_border(c)`, where c is a constraint, gives the latest (i.e. the biggest) time after which we have for sure thanks to c an information on any signal satisfying c . In case of a $\forall\langle a; b \rangle : u$ constraint, we know the signal value at any point inside $[a; b]$, so that the result is b . On the other hand, a $\exists[a; b] : u$ constraint only says that after a , we will have one information about the signal. Similarly, `left_influence_border` gives the earliest time before which we have an information on the signal.

In order to prevent the loss of information which leads among others the result of $\overleftarrow{\Psi}_{\text{AND}}^{\#}(\forall\langle a; b \rangle : \text{False})$ to be defined as \top , we introduce a new abstract domain which interacts with the Constraint domain and prevents this imprecision.

5 Changes Counting Domain

Goals of the Changes Counting Domain. Synchronous programs are often used in embedded systems, that they control through *actuators*, after having com-

puted reactualized orders depending on what they received from *sensors*. Most of these sensors, however, work **in real time**, i.e. they deliver a value at any time. For example, the temperature is usually measured by a resistance varying continuously with the temperature. Of course, engineers take this into consideration and discretize the signal, and as a consequence stabilize it. We try to take advantage of this kind of properties by creating a new abstract domain that interacts with our Constraint domain.

Definition of the Changes Counting Domain. In fact we define two abstract domains, but they are close enough to share most of the code of their abstract transfer functions. The **Changes counting** domain, denoted by \mathcal{N} simply describes, at each control point, the maximal number of changes that may happen :

- either during a particular interval $[a; b]$: it is the Local Changes counting domain
- or during any interval of given width (after time $t = 0$, starting point of the execution) : it is the Global Changes counting domain.

The only difference is in fact that the global version adds a universal quantifier to its condition.

Definition 3 (Local Changes Counting Domain). *The set of Local Changes counting is $(\mathbb{N} \cup \{+\infty\}) \times \mathbb{R}^+ \times (\mathbb{R}^+ \cup \{+\infty\})$. The top element, denoted by \top is $(+\infty, 0, +\infty)$. The bottom element, denoted by \perp is $(+\infty, 0, 0)$.*

Definition 4 (Global Changes Counting Domain). *The set of Global Changes counting is $(\mathbb{N} \cup \{+\infty\}) \times (\mathbb{R}^+ \cup \{+\infty\})$. The top element, denoted by \top is $(+\infty, +\infty)$. The bottom element, denoted by \perp is $(+\infty, 0)$.*

In the following, we consider the Global Changes counting domain only. All the statements can be easily adapted to the Local Changes counting domain by considering **one** interval instead of **all** of the ones with a particular width. The meaning of an element $(n, \delta)_{\mathcal{N}} \in \mathcal{N}$ connected to a control point p_1 , where S_1 denotes set of signals in the concrete semantics, is :

$\forall v \in S_1, \forall x \in \mathbb{R}^+, \exists x_0, \dots, x_{n+1} \in [x; x + \delta]$, such that

$$x_0 < x_1 < \dots < x_n < x_{n+1} \\ v(x_0) \neq v(x_1) \wedge v(x_1) \neq v(x_2) \wedge \dots \wedge v(x_n) \neq v(x_{n+1}).$$

This prevents v from changing its value more than n times during $[x; x + \delta]$. This automatically defines a *concretization function* γ as the following set :

$$\gamma(n, \delta) \triangleq \left\{ v : \mathbb{R}^+ \mapsto \mathbb{B}, \left| \begin{array}{l} \forall x \in \mathbb{R}^+, \exists x_0, \dots, x_{n+1} \in [x; x + \delta], \\ x_0 < x_1 < \dots < x_n < x_{n+1} \\ v(x_0) \neq v(x_1) \wedge \dots \wedge v(x_n) \neq v(x_{n+1}) \end{array} \right. \right\}$$

This function relates the concrete and the abstract domain. It automatically defines a pre-order $\leq_{\mathcal{N}}$ on $\mathcal{N} : \forall n_1, n_2 \in \mathcal{N}, n_1 \leq_{\mathcal{N}} n_2 \iff \gamma(n_1) \subseteq \gamma(n_2)$. In fact, we consider the abstract elements modulo $\leq_{\mathcal{N}} \cap \geq_{\mathcal{N}}$ so that $\leq_{\mathcal{N}}$ is an order.

Applications of This Domain. This domain is useful in three types of context :

- It may simply be a direct consequence of the property that we try to prove (or its negation). It is interesting in that case to rewrite it in the Changes counting domain.
- Some operators (like the discretization previously considered) may induce a stability that may be converted into a Changes counting domain element.
- An input may be by physical construction (hardware property) stable and this property may be translated into a Changes counting domain element.

*Transfer Function Inside the **Changes counting** Domain.* If the width (called δ above) of any abstract element of the Changes counting domain was the same for each abstract element of this domain, the transfer functions would be really easy to define. Considering for example the AND gate connecting two input control points p_1 and p_2 to an output control point p_3 , if we know that :

- any signal in the semantics connected to point p_1 changes its value at most n_1 times during any interval of width δ ,
- any signal in the semantics connected to point p_2 changes its value at most n_2 times during any interval of width δ ,

clearly, any signal in the semantics connected to point p_3 changes its value at most $n_1 + n_2$ times during any interval of width δ . In that simplified case, this allows a simple definition of an **abstract forward transfer function** for the AND operator.

But, if we try to compute the result of the abstract forward transfer function $\Psi_{\text{AND}}^{\#}$ on the couple of abstract elements $(5, 2.1)_{\mathcal{N}}$, $(3, 4.7)_{\mathcal{N}}$, one for each input of the AND gate, we cannot use the same argument.

However, as pictured on Fig. 4, if the signal at the second input control point cannot change its value more than three times during 4.7 time units, it also cannot change its value more than three times during 2.1 time units, and therefore satisfies the abstract element $(3, 2.1)_{\mathcal{N}}$. As a consequence :

$$\vec{\Psi}_{\text{AND } \mathcal{N}}^{\#}((5, 2.1), (3, 4.7)) \leq_{\mathcal{N}} \vec{\Psi}_{\text{AND } \mathcal{N}}^{\#}((5, 2.1), (3, 2.1))$$

and this allows the same type of computation as the one above : the output control point satisfies the abstract element $(8, 2.1)_{\mathcal{N}}$. This is done automatically by a *reframing* function :

Definition 5. A *reframing* function takes **two** abstract elements containing a notion of interval, and returns two new abstract elements whose interval widths

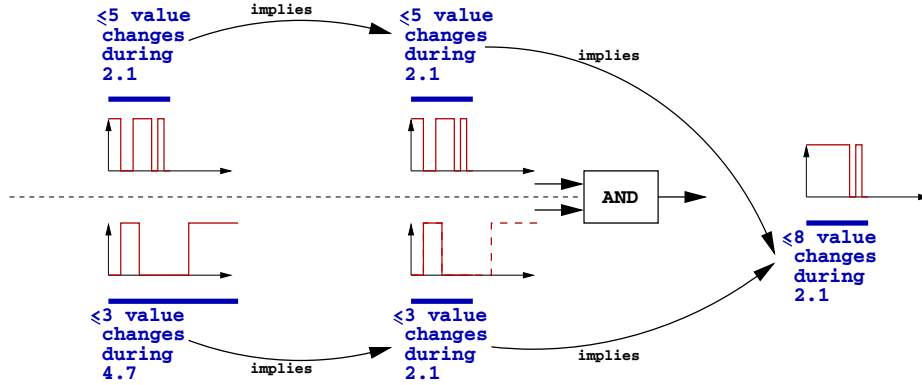


Fig. 4. Computing the abstract operator AND in the changes counting domain

are equal, and that are consequences of the two input elements. We call φ_0 the basic reframing function that returns its input elements, only modified in that the interval width is the shortest of its two input interval widths.

For instance, $\varphi_0((5, 2.1), (3, 4.7)) \triangleq ((5, 2.1), (3, 2.1))$. Trickier reframing functions may easily be defined for the Changes counting domain elements.

Transfer Functions of Time-Independent Operators. All the time-independent operators OP are connected to an abstract forward transfer function $\Psi_{OP\#}$ defined by :

$$\Psi_{OP\#, \varphi}((n_1, \delta_1), (n_2, \delta_2)) \triangleq (\tilde{n}_1 + \tilde{n}_2, \tilde{\delta}_1)$$

where φ is a reframing function and $\varphi((n_1, \delta_1), (n_2, \delta_2)) = ((\tilde{n}_1, \tilde{\delta}_1), (\tilde{n}_2, \tilde{\delta}_1))$. This abstract function is clearly sound, since the result of a time-independent operator may change only if one of its inputs changes.

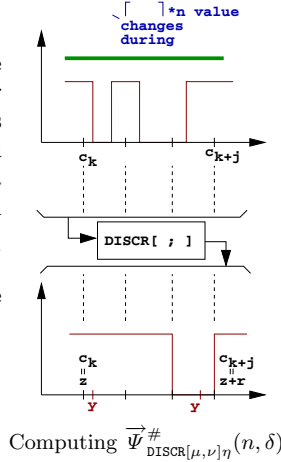
Transfer Function of Time-Dependent Operators. This case is much more difficult. We consider for instance the operator $DISCR_{[\mu, \nu]}$ which models the periodic reading of an input buffer (according to an imprecise clock whose cycles last between μ and ν time units; $\mu, \nu \in \mathbb{R}_*^+$). First, we may deduce stability information without even looking at the input, since this operator performs a discretization. We know that the output signal may not change its value more than once during any interval $[x; x + \mu], x \in \mathbb{R}^+$, which generates the abstract element $(1, \mu)_{\mathcal{N}}$.

But this may not be precise enough. Another argument can be considered. We assume that we already know that during any interval $[x; x + \delta]$, any input signal may change its value at most n times $:(n, \delta)_{\mathcal{N}}$. Let us consider any interval $[y; y + \eta]$ at the control point after the discretization. It is included in at most

$j = \lceil \frac{\eta}{\mu} \rceil + 1$ consecutive cycles¹ of the discretizer, which cover an interval $[z, z + hws\rho]$, with $\rho \leq \lceil \frac{\eta}{\mu} \rceil * \nu + \nu$.

Now, $[z, z + \rho]$ is of width ρ , and during these j cycles, only $\lceil \frac{\rho}{\delta} \rceil * n$ value changes may occur at the input control point. Since the operator is only discretizing the input signal, it doesn't add any value change, and therefore at most $\lceil \frac{\rho}{\delta} \rceil * n$ value changes may occur during the j cycles, and a fortiori during $[y; y + \eta]$. Since $\rho \leq \lceil \frac{\eta}{\mu} \rceil * \nu + \nu$, we know that $\lceil \frac{\rho}{\delta} \rceil * n \leq \lceil \frac{\lceil \frac{\eta}{\mu} \rceil * \nu + \nu}{\delta} \rceil * n$, and we may soundly define :

$$\vec{\Psi}_{DISCR[\mu, \nu]\eta}^\#(n, \delta) \triangleq \left(\left\lceil \frac{\lceil \frac{\eta}{\mu} \rceil * \nu + \nu}{\delta} \right\rceil * n, \eta \right)$$



We introduced a η parameter in the computation of the abstract operator that will necessarily also be a parameter of our analysis. The operator and the analysis are sound for any η . When an analysis is not successful, modifying this parameter is a way to refine it.

Reduced Product Between the Constraints Domain and the Changes Counting Domain. It is clear that any signal that may not change its value more than once in 2 time units, and that also satisfies the constrains $\forall \langle 0; 0 \rangle : \text{True}$ and $\forall \langle 2; 2 \rangle : \text{True}$ is always equal to **True** between 0 and 2. Indeed, if it was equal to *False* at some point t between 0 and 2, then it would change its value at least twice during the $[0; 2]$ interval (once between 0 and t , and once between t and 2).

We now generalize informally to a more realistic yet simple example the reduced product : we study how the conjunction $(c_1 \wedge c_2)_C$ of two constraints and the abstract element $(1, stab)_N$ may be reduced.

We define a function ρ connecting any pair (c_1, c_2) of constraints and any real *stab* that satisfies the following conditions :

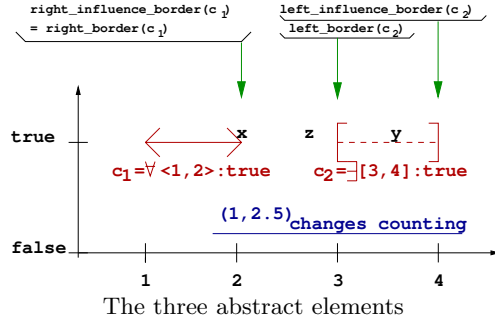
- the intervals of the two constraints are disjoint
- the values of the constraints are equal. We call b this value.
- in case c_1 is before c_2 (which makes sense when the first condition is satisfied), $\text{left_influence_border}(c_2) - \text{right_influence_border}(c_1) < \text{stab}$, and conversely in the other case.

to a new constraint : $\forall \langle \text{right_border}(c_1); \text{left_border}(c_2) \rangle : b$.

¹ $\lceil v \rceil$ denotes the smallest integer greater or equal to v .

For example, we consider on the figure on the right that the following abstract elements are satisfied :

- $c_1 = \forall\langle 1, 2 \rangle : true$
- $c_2 = \exists[3, 4] : true$
- $(1, 2.5)_{\mathcal{N}}$, i.e. one value change at most during any interval of width 2.5



This transformation is sound for the same reason as the one proving the $\forall\langle 0; 0 \rangle : True \wedge \forall\langle 2; 2 \rangle : True \wedge (1, 2.)_{\mathcal{N}}$ case. For instance, if we assume that a signal s_p at a control point p satisfies the constraints c_1 and c_2 , the changes counting condition $(1, stab)$, that the three previous conditions are satisfied and that c_1 is before c_2 , then we know that :

- at a point x after `right_influence_border`(c_1) and before `right_border`(c_1), the signal takes the value b
- at a point y after `left_border`(c_2) and before `left_influence_border`(c_2), the signal takes the value b

If at some point z between x and y , the signal takes the value $-b$, then $(1, stab)_{\mathcal{N}}$ is not satisfied, since the width of $[x; y]$ is smaller than the width of

$$[\text{right_influence_border}(c_1); \text{left_influence_border}(c_2)]$$

which is smaller than $stab$: the signals changes its value once between x and z and once between z and y . As a consequence, the signal must take the value b at point z , and therefore at any point between x and y and a fortiori during the whole interval $[\text{right_border}(c_1); \text{left_border}(c_2)]$.

This argument is generalized to any finite set of constraints. The ρ function is applied to all the pairs present in the constraint set, and the result is soundly added to this set of abstract elements. It may also be generalized to the case where the changes counting abstract element is $(n, stab)_{\mathcal{N}}$ instead of $(1, stab)_{\mathcal{N}}$.

6 Integral Bounding

Origin and Goals of This Domain. We assume that a system contains two control points p_1 and p_2 clocked by different desynchronized clocks C_1 and C_2 . Comparing the values $v_1(t)$ and $v_2(t)$ of the semantics restricted to these control points at a time t is needed in order to set interesting properties and to prove them. However, which time t should we consider ? It would be necessary to consider any n -th tick $C_1(n)$ and $C_2(n)$. But we often don't even know their interleaving. However, one thing that vary continuously with a slight move of a clock tick is the integral of the value. We consider the boolean signals are converted into 0

or 1 integer signals. Since the semantics is defined step by step, following the clocks, the signals may vary only finitely many times during an interval of finite width, and there is no risk of undefined integral.

Definition of the Integral Bounding Domain. As for the changes counting domain, this domain may be split into a **local** and a **global** version. We present the more complex global one, from which it is easy to extract the local version. The basic element of the **Integral bounding** domain, denoted by \mathcal{I} , is the *Integral bound*.

Definition 6 (Integral bound). *The set of Integral bounds is $\mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+$. The top element, denoted by \top is $(+\infty, 0, +\infty)$. The bottom element, denoted by \perp is $(1, 1, 0)$.*

The meaning of an element (δ, α, β) at a control point p_1 , where S_1 denotes the set of signals in the concrete semantics, is :

$$\forall v \in S_1, \forall x \in \mathbb{R}^+, \alpha \leq \int_x^{x+\delta} \text{int_of_bool}(v(t))dt \leq \beta,$$

where $\text{int_of_bool} : \begin{cases} True \mapsto 1 \\ False \mapsto 0 \end{cases}$. We may therefore define a *concretization function* γ as the following set :

$$\gamma(\delta, \alpha, \beta) \triangleq \left\{ v : \mathbb{R}^+ \mapsto \mathbb{B}, \forall x \in \mathbb{R}^+, \alpha \leq \int_x^{x+\delta} \text{int_of_bool}(v(t))dt \leq \beta \right\}.$$

It automatically defines a pre-order on $\mathcal{I} : \forall i_1, i_2 \in \mathcal{I}, i_1 \leq_{\mathcal{I}} i_2 \iff \gamma(i_1) \subseteq \gamma(i_2)$. In fact, we consider the equivalence classes of elements of this abstract domain according the equivalence relation $\leq_{\mathcal{I}} \cap \geq_{\mathcal{I}}$. On these classes, $\leq_{\mathcal{I}}$ is an order.

The Integral bounding domain transfer functions that propagate the information available at one control point to the other ones according to the syntax are all constant equal to \top , which is clearly sound. A reason for this deliberate loss of precision is that the Integral bounding domain is always used in reduced products, that will be described in the next sections.

Reduced Product Between the Integral Bounding Domain and the Changes Counting Domain. In the ideal case, we are in presence of two already proven properties of any signal v in the semantics for a particular control point p_1 and for any $x \in \mathbb{R}^+$:

- v satisfies a changes counting condition : the value $v(t)$ at time t doesn't change more than n times during any interval $[x; x + \delta]$
- v satisfies an integral bounding condition : $\alpha \leq \int_x^{x+\delta} v(t)dt \leq \beta$.

This is optimal in that the width of both types of the intervals (δ) is the same. If this isn't the case, we may reuse a *reframe* function defined before. Now, if we consider $[x; x + \delta[$ as the union $\bigcup_j [x + \frac{j}{k} \times \delta; x + \frac{j+1}{k} \times \delta[$, and we try to dispatch the n value changes inside these k intervals, we discover that there

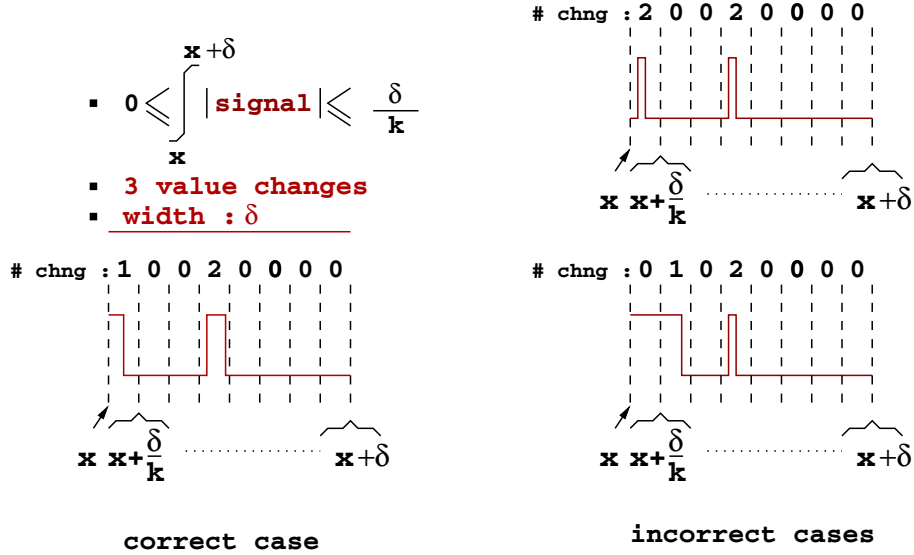


Fig. 5. Three cases that may happen while dispatching the n value changes in the intervals, the two on the right are discarded, because they do not satisfy the conditions, respectively the changes counting condition and the integral bounding one

is a finite disjunction of cases, and that we have some information about the constraints on each of these cases. Some must be discarded, since in any case, they won't satisfy the Integral bounding condition. Three cases are presented on Fig. 5.

*Reduced Product Between the **Integral Bounding Domain** and the **Constraint Domain**.* We consider here a control point where the following local integral bounding condition is satisfied :

$$0 \leq \int_x^{x+4} v(t)dt \leq 3.$$

We also assume that at this control point, the constraints $\forall \langle x; x + 1 \rangle : \text{True}$ and $\forall \langle x + 3; x + 4 \rangle : \text{True}$ hold. Since

$$\int_x^{x+4} v(t)dt = \int_x^{x+1} v(t)dt + \int_{x+1}^{x+3} v(t)dt + \int_{x+3}^{x+4} v(t)dt = 2 + \int_{x+1}^{x+3} v(t)dt,$$

it is clear that we may safely replace the integral bounding condition by a new more precise one :

$$0 \leq \int_{x+1}^{x+3} v(t)dt \leq 1.$$

Having, as in this example, the interval of a Constraint included in the interval of a local integral bounding is exceptional. But in the case of a global

integral bounding condition, we may “choose” any starting point : the condition is valid for each of them. An interesting choice made by the analyzer for this starting or ending point would be the borders of the Constraint elements already accumulated at the considered control point.

*The Reduced Product Between the **Integral Bounding**, the **Constraint** and the **Changes Counting Domain**.* A reduced product is obtained from the three basic domains by considering all the optimizations described earlier and combining them. We propose an example in the next section.

7 Analyzing Imperfectly-Clocked Synchronous Systems

We present here the simulation of an analysis build on top of the three presented domains of an academic example. We analyze a simple system which could easily be build for example in SCADE. A graphical representation of this system is presented on Fig. 6. The classical PRE operator is initiated with the `false` value, as a `p4=false->PRE p2` in Lustre. We assume the entrance signal is quite stable : it may not change its value more than once in any interval of width 100 time units.

The stability at point p_1 is translated into the abstract element $(1, 100)_{\mathcal{N}}$. Following the abstract operator for the changes counting domain defined in Sect. 5, we get a condition for the signals at points p_2 and p_3 :

$$\Psi_{\text{DISCR}[39,41]116}^{\#}(1, 100)_{\mathcal{N}} = \left(\left(\left\lceil \frac{\lceil \frac{116}{39} \rceil * 41 + 41}{100} \right\rceil * n, 116 \right) \right) = (2, 116)_{\mathcal{N}}.$$

As a consequence, at points p_2 , p_3 , and p_4 the signals satisfy the condition $(2, 116)_{\mathcal{N}}$ and a fortiori $(2, 100)_{\mathcal{N}}$. At point p_5 , the abstract operator for the logical gate XOR computes the abstract element $(4, 116)_{\mathcal{N}}$. Imagine we try to prove that for a signal v at point p_5 , $\int_x^{x+100} v(t)dt < 95$. We propose a proof by contradiction. We therefore assume $(100, 95, 100)_{\mathcal{I}}$ is satisfied at point p_5 and try to prove that we converge into \perp .

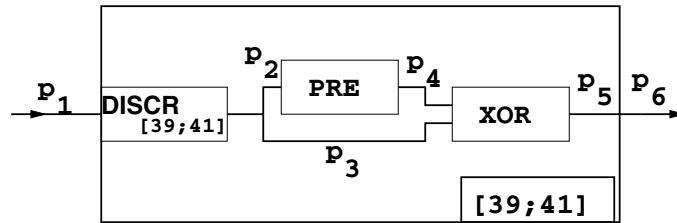


Fig. 6. A example of system where the cooperation between the three abstract domains is needed

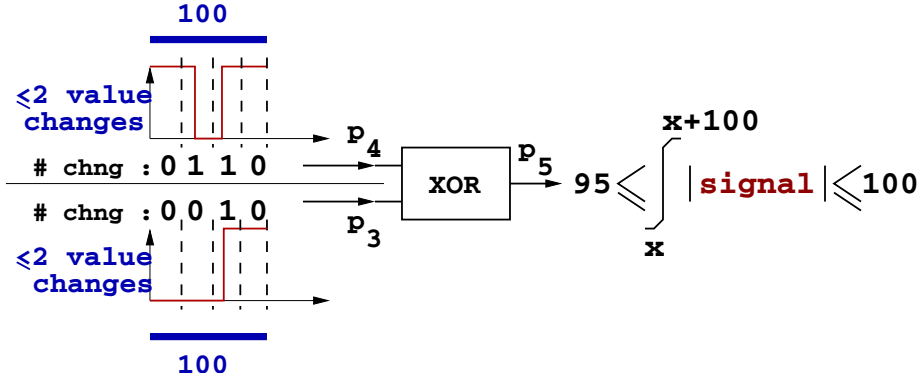


Fig. 7. One possibility of repartition of the value changes

The analyzer will distribute, as in the Sect. presenting the reduced product between the integral bounding domain and the changes counting domain, the possible changes (at most four) into subintervals of $[x; x + 100]$. We consider a case with three value changes. Let us assume that $[x; x + 100]$ is divided into four subintervals (i.e. $k = 4$). The analyzer will then distribute these changes to the inputs, since a change in the result of a XOR operator necessarily comes from a change in one of its inputs. One of the possibilities is presented on Fig 7. However, it is soon discarded. Indeed, it implies (among others) the abstract constraint $\forall \langle 75; 100 \rangle : \text{true}$ for points p_4 and p_2 . Hence, $\overline{\Psi}_{\text{XOR}}^\# (\forall \langle 75; 100 \rangle : \text{True}, \forall \langle 75; 100 \rangle : \text{True}) = \forall \langle 75; 100 \rangle : \text{False}$ for the point p_5 , which is detected as incompatible with the $95 \leq \int_x^{x+100} \text{signal}(t)dt \leq 100$ condition by the reduced product between the Integral bounding domain and the Constraint domain. The interaction of the three abstract domains gaps thus their local weaknesses.

This is a much simplified view of what our analyzer does, even in that simple case. But step by step, our analyzer discards the impossible cases. If it fails, it will perform a refinement, either by dividing each interval again, or by changing the parameters of the abstract operators. If all the cases get discarded, this means that a sound abstract overapproximation of the concrete semantics is empty, and thus that this concrete semantics is empty, which puts a end to this proof by contradiction.

8 Implementation and Future Improvements

This domains were implemented and combined into a prototype of analyzer written in `ocaml`. The principle is the same as in the above example. We consider the intersection A of an overapproximation of the semantics and of the negation of the wanted property. We iterate the abstract operators in order to get a precise abstract representation $A^\#$ of A . If the analyzer proves that $A^\# = \perp$,

the property is proved, since its negation does not intersect with the semantics of the system. Otherwise, either the analyzer is not precise enough, or the property is false and a counter-example may be found in the concretization of $A^\#$.

Instead of reals, which are omnipresent in our domains, we used the `float` type. This may cause rounding errors during the analysis that we would not detect. We would like to take them into consideration, as explained in [5]. We would also like to prove the robustness of some systems to errors in the communication or in the execution. Communication errors could of course be simulated by removing the bus link and setting at its arrival the value to \top , but there are few chances that a system would still satisfy any interesting specification in that case. A more interesting approach would introduce probabilities for an information to be lost in the communication, and it would require a probability for the system to recover from this error. A modified version of the integral bounding domain could undertake this feature.

9 Conclusion

The computerized commands of an embedded system are usually made of several communicating synchronous systems, each one with its own clock. During a real execution, the clocks tick imperfectly and the communication delays between the systems are non-constant. We proposed a realistic model that allows the specification of the allowed clock skew between several clocks as well as the variable communication delays. In this model, there may be a non-countable infinity of different executions of a the synchronous subsystems. We introduced two new abstract domains that can express and handle temporal properties despite of these hardware imperfections. The result of the reduced product of them with a previously defined abstract domain is able of both expressing and proving many of the interesting temporal properties in such an environment. It is the basis of a working prototype of static analyzer.

References

1. A. Benveniste, P. Caspi, P. Le Guernic, H. Marchand, J.-P. Talpin, and S. Tripakis. A protocol for loosely time-triggered architectures. *LNCS, Proceedings of the Second International Conference on Embedded Software*, p. : 252 - 265, 2002.
2. G. Berry. *The Constructive Semantics of Pure Esterel*. 1999.
3. G. Berry. *Proof, language, and interaction: essays in honour of Robin Milner, Pages: 425 - 454: The foundations of Esterel*. MIT Press, 2000.
4. J. Bertrane. Static analysis by abstract interpretation of the quasi-synchronous composition of synchronous programs. *VMCAI*, 2005.
5. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones, LNCS 2566, 85-108*. Springer, 2002.

6. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. *Proc. ACM SIGPLAN 2003 Conf. PLDI, 196-207, San Diego, CA, USA, . ACM Press*, 7-14 juin 2003.
7. P. Caspi. Embedded control: From asynchrony to synchrony and back. *1st International Workshop on Embedded Software, EMSOFT2001, Lake Tahoe, Volume 2211 in LNCS*, October 2001.
8. P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert. From simulink to scade/lustre to tta: a layered approach for distributed embedded applications. *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, 2003.
9. P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice. Lustre: A declarative language for programming synchronous systems. *Proceedings of the 14th ACM symposium on Principles of programming languages, POPL'87*, 1987.
10. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2-3):103—179, 1992.
11. N. Halbwachs. *Synchronous programming of reactive systems*. Dordrecht Boston , Kluwer Academic Publishers, 1993.
12. S. Thompson and A. Mycroft. Abstract interpretation of asynchronous circuits. *SAS, Verona, Italy*, August 2004.