

Static Analysis by Abstract Interpretation of the Quasi-Synchronous Composition of Synchronous Programs

Julien Bertrane
bertrane@di.ens.fr

Computer Science Department, École normale supérieure, Paris, France

Abstract We present a framework to graphically describe and analyze embedded systems which are built on asynchronously wired synchronous subsystems. Our syntax is close to electronic diagrams. In particular, it uses logic and arithmetic gates, connected by wires, and models synchronous subsystems as boxes containing these gates.

In our approach, we introduce a continuous-time semantics, connecting each point of the diagram to a value, at **any** moment. We then describe an analysis derived from the abstract interpretation framework enabling to statically and automatically prove temporal properties of the diagrams we defined. We can prove, for example, that the output of a diagram cannot be equal to a given value in a given interval of time.

1 Introduction

Embedded systems are often built on synchronous subsystems. Several tools help programmers to such a design, like SCADETM[2]/LUSTRE[5] or SIMULINK. For safety matters, these synchronous subsystems must however be redundant. This is the origin of several issues.

First, in case of disagreement between these redundant synchronous subsystems, the system has to *choose* which subsystem should be trusted. Then, it appears that the former problem will happen very frequently, because of the *de-synchronization* of the clocks of these subsystems: two physical clocks, even started simultaneously, cannot stay synchronized. This phenomenon is unavoidable as soon as we consider real embedded system with long use duration. In this case, the different synchronous subsystems compute in a de-synchronized way on different inputs and consequently always disagree!

This is why, without complementary hypothesis, asynchronous composition won't satisfy many safety properties. We therefore assume that the synchronous subsystems are always quasi-synchronous, which means that the duration of the cycles of these subsystems are very close to each other. Provided with this hypothesis, we may prove safety properties of some fault-tolerant systems.

Synchronous systems may have a discrete semantics. On the other hand, quasi-synchronous systems must be connected to a semantics that considers the time as continuous. Considering the quasi-synchronous hypothesis may indeed

drive us to suppose that one cycle starts for example between time t_1 and time t_2 , which is a continuous-time property.

We present a framework that enables a graphical description of such processes. We then propose a tool to prove temporal safety properties of such complex compositions of programs. This tool is based on several analyses, derived from the abstract interpretation theory.

Previous works. P. Caspi and R. Salem presented in [6] a system *with fault tolerance without synchronization* between synchronous subsystems. They managed to prove by hand the robustness properties of the procedures they wrote. In [1] were introduced the LTTA, enabling, under some hypotheses, synchronous systems with non-perfect clocks to communicate in a secure way. S. Thompson and A. Mycroft, on the other side, proposed in [12] several abstractions to study *asynchronous circuits* that convinced us of the high simplification that could be achieved through abstraction. Their methods, however, could not be directly used to reach our goal, because their abstractions partially discard the time. Last, we widely used the theory and we inspired by the applications developed by P. Cousot, R. Cousot and their team ([4,3,8,7,11,9]).

We introduce in Section 2 a new syntax and a continuous-time semantics. We then present in Section 3 an abstract domain based on constraints. Section 4 defines abstract operators. We then present our first analysis and an example in Section 5. Lastly, we propose in Section 6 another improved analysis.

2 Syntax and semantics

The semantics developed in order to give a meaning to the asynchronous composition of synchronous programs often relies on a translation into a synchronous environment. However, it implies to check all the possible interleavings of the events of the synchronous subsystems. In our case, it appears that the number of interleavings is by far too big to be exhaustingly explored. That's why it seems reasonable to introduce a continuous-time semantics.

2.1 Syntax

We chose an easily representable syntax, inspired by electronic diagrams, with gates (arithmetic, logic,...), wires connecting the gates, and boxes isolating parts of the diagrams. The advantage of this graphic representation is that it reminds us that the elements of the syntax are continuous, like in a real electronic diagram, where each point is connected to a voltage at each time.

2.1.1 Calculus units: gates

The calculus of our programs are described by gates:

- +, -, \times , with two inputs and one output
- OR, XOR, AND, with two inputs and one output

- NOT, with one input and one output
- CONST_α , with no input and one output
- TRASH, with one input and no output
- $\text{DELAY}[\alpha, \beta]$, with one input and one output

The meaning of the three first types of gates is obvious. The gate CONST_α is meant to generate a constant value, TRASH simply absorbs the value it is given, and $\text{DELAY}[\alpha, \beta]$ postpones the input values by a real delay which can vary between α and β .

2.1.2 Replacing variables: internal wires

The previous gates are connected by wires, which simply transmit values arriving at their inputs. We also allow splitting wires, which transmit values identically and simultaneously to several gates.

With the previous objects, we can already build diagrams like the one on Fig. 1.

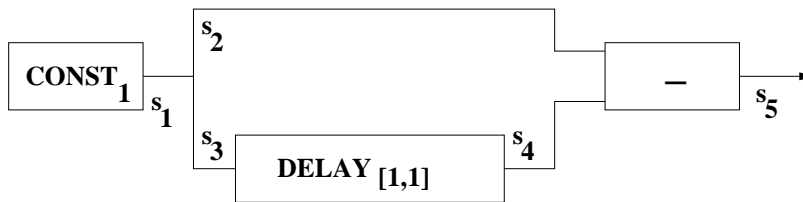


Figure 1. A simple diagram

Definition 1 (control point). : A control point is any input or output of a gate or of a wire.

We often collapse the two control points of the same wire. We call V the set of control points of a diagram.

2.1.3 Synchronous units: boxes

A box is meant to represent a whole synchronous program, i. e. gates and wires, executed quasi-synchronously. As a consequence, it has to be isolated from the rest of the diagram, which represents other programs and wires between them. A synchronous program is executed in a cyclic way. Each cycle is supposed to last between α and β . We therefore connect each box to an *execution duration interval* $[\alpha, \beta]$. For example, the box in Fig. 2 is assumed to be executed in constant time 1, since its execution duration interval is $[1, 1]$. Lastly, a box has the same inputs and outputs as the ones of the diagram it is built on. For example, the box in Fig. 2 has no input and one output.

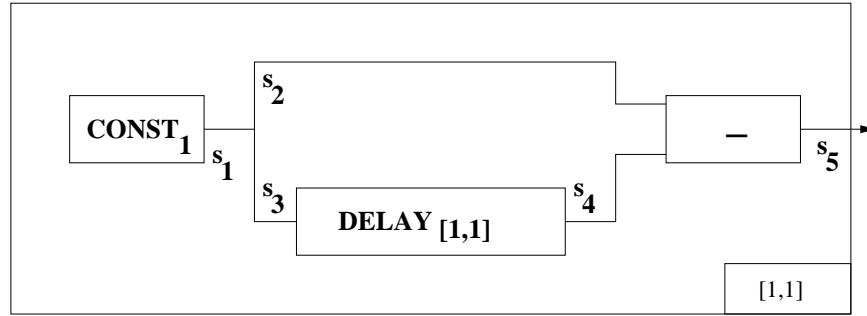


Figure 2. A simple box

2.1.4 Communication units: external wires

External wires connect an output of a box to an input of a box, in an oriented non-instantaneous way. They are tagged with a time interval, and therefore represented on diagrams as a DELAY gate, as depicted on the left diagram of Fig. 3.

2.2 Continuous-time semantics

We write \mathbb{R} for the set of reals, \mathbb{R}_* for the negative reals, and \mathbb{B} for the booleans.

2.2.1 The set of concrete elements

We said above that we want our semantics to be continuous. The easier way to achieve this is to choose *signals* as elements of the concrete domain. This choice is by the way very coherent with the graphical representation of the programs.

Definition 2 (signal). : A signal is a function $\mathbb{R} \mapsto \mathbb{B}$ or \mathbb{R} which is equal to 0 or false on \mathbb{R}_* .

Definition 3 (concrete elements). : The set of concrete elements is $V \rightarrow \mathcal{P}(S)$. This set is ordered by the usual pointwise inclusion of sets.

2.2.2 Two new gates: the sampler and the SHIFT

These gates won't be used when designing diagrams. They aim at translating the discrete properties of synchronous programs into our continuous model. They thus take a set of *clocks* as parameter, and have one input and one output. A *clock* is any strictly increasing function $c : \mathbb{N} \mapsto \mathbb{R}$, such that $c(0) = 0$. At this point, we do not discard Zeno's paradox. However, the semantics of the three time-sensitive gates (sampler, SHIFT, and DELAY $[\mu, \nu]$) prevents it to appear by requiring $\mu > 0$.

The *sampler* is represented on diagrams as $|||[\mu, \nu]$ and the SHIFT as SHIFT $[\mu, \nu]$, where μ and ν are parameters restricting the clocks that may affect the input signal. They must satisfy: $\nu \geq \mu > 0$.

2.2.3 Equations generated by the gates

- The semantics of a **OR** gate with two input control points E_1 and E_2 and an output S_1 is noted $\llbracket (E_1, E_2), \text{OR}, S_1 \rrbracket$ and is the equation:

$$\forall t \in \mathbb{R}, S_1(t) = \text{True} \text{ iff } E_1(t) = \text{True} \text{ or } E_2(t) = \text{True}$$

The other logic and arithmetic gates have a similar semantics.

- $\llbracket E_1, \text{DELAY}[\alpha, \beta], S_1 \rrbracket$ is the equation :

$$\exists \delta : \mathbb{R} \rightarrow \mathbb{R}^+, \text{ increasing, such that}$$

$$\forall t \in \mathbb{R}, \delta(t) - t \in [\alpha, \beta] \text{ and } \forall t \in \mathbb{R}, S_1(\delta(t)) = E_1(t).$$

- The semantics of a sampler $\lll [\mu, \nu]$ is $\llbracket E_1, \lll [\mu, \nu], S_1 \rrbracket$, defined as:

$$\exists c \text{ clock}, \forall n \in \mathbb{N}, c(n+1) - c(n) \in [\mu, \nu], \text{ and}$$

$$\forall t \in [c(n), c(n+1)[, S(t) = S(c(n))$$

- The semantics of a **SHIFT SHIFT** $[\mu, \nu]$ is $\llbracket E_1, \text{SHIFT}[\mu, \nu], S_1 \rrbracket$, defined as¹:

$$\exists c \text{ clock}, \forall n \in \mathbb{N}, c(n+1) - c(n) \in [\mu, \nu], \text{ and}$$

$$\forall t \in [c(n), c(n+1)[, S(t) = S(c(n)^-)$$

- Lastly, the equations generated by wires are simple equalities.

You can notice that all the gates are considered as instantaneous, except $\text{DELAY}[\alpha, \beta]$, $\text{SHIFT}[\mu, \nu]$, and $\lll [\mu, \nu]$. The definition of $\text{DELAY}[\alpha, \beta]$ implies that it keeps the order between the values on the signal. It thus represents a *serial* transmission.

2.2.4 The concrete semantics

A preliminary transformation: We would like to consider the *function* connecting the control points to a set of signals which, at these points, satisfy the equations generated by a diagram as the semantics of the diagram. But this definition faces the existence of boxes. That's why we define a function Φ which connects a diagram S to another one $\Phi(S)$ such that, at each box B with an execution duration interval $[\alpha_B, \beta_B]$:

- we add at each input of B a gate $\lll [\alpha_B, \beta_B]$.
- we add at each output of B a gate:
 - $\text{DELAY}[\alpha_B, \beta_B]$ if the box contains no **DELAY** gate.
 - $\text{SHIFT}[\alpha_B, \beta_B]$ if the box contains at least one **DELAY** gate.
- we remove the box!

¹ We define $f(x^-) \triangleq \lim_{\substack{y \rightarrow x \\ y < x}} f(y)$

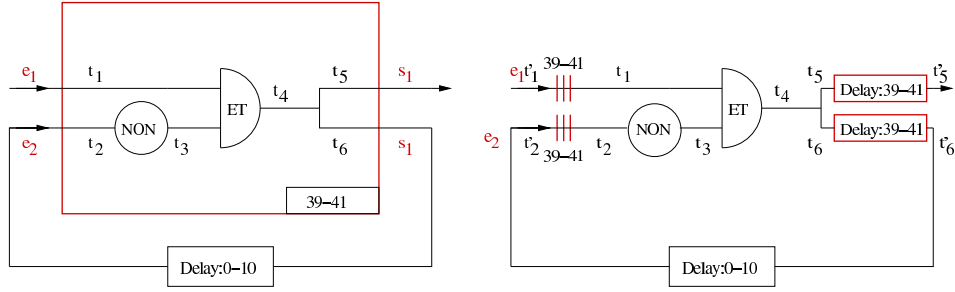


Figure 3. Transforming a simple diagram

Semantics.

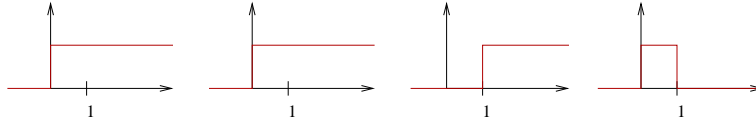
Definition 4 (Semantics). *The equational semantics of a diagram is the function $f : V \rightarrow \mathcal{P}(\mathcal{S})$, such that for each $v \in V$, $f(v)$ is the set of signals u such that for each $w \in V \setminus \{v\} = \{w_1, \dots, w_{\#V-1}\}$ exists a signal s_w , such that $(u, s_{w_1}, \dots, s_{w_{\#V-1}})$ satisfies all the equations in the semantics of the elements of the diagram.*

We note $\llbracket D \rrbracket$ for the semantics of the diagram D .

An example. let us consider the semantics² of the diagram on Fig. 2:

$$(\cdot, \text{CONST}_1, s_1) \quad (s_1, \text{DUPL}, (s_2, s_3)) \quad (s_3, \text{DELAY}_{[1,1]}, s_4) \quad ((s_2, s_4), -, s_5)$$

$$\begin{array}{llll} s_1 \mapsto & s_2, s_3 \mapsto & s_4 \mapsto & s_5 \mapsto \\ \{\lambda t. \text{if } t < 0 & \{\lambda t. \text{if } t < 0 & \{\lambda t. \text{if } t < 1 & \{\lambda t. \text{if } 0 \leq t < 1 \\ \text{then } 0 \text{ else } 1\} & \text{then } 0 \text{ else } 1\} & \text{then } 0 \text{ else } 1\} & \text{then } 1 \text{ else } 0\} \end{array}$$



2.3 Concrete operators

The equational semantics is unfortunately quite difficult to handle. If, for example, we have an information on one control point of a diagram, it has consequences on the control points on the other side of any gate connected to this

² We here use Church's lambda calculus notation: $\lambda x.e$ is the function mapping x to the expression e .

control point. But we still cannot easily control the propagation of the information. We choose to *orientate* this propagation *backwards*. To propagate the information backwards without losing too much precision, we just define operators that compute the weakest precondition implying the postcondition we already have as hypothesis.

2.3.1 Operator DELAY $[\alpha, \beta]$

We define $\Psi_{\text{DELAY}[\alpha, \beta]}(A)$ as :

$$\left\{ a \text{ such that } \exists \delta \left| \begin{array}{l} \delta : \mathbb{R} \rightarrow \mathbb{R} \\ \forall t, \delta(t) - t \in [\alpha, \beta] \\ \lambda t. a(\delta(t)) \in A \end{array} \right. \right\}.$$

If we assume that (E_1, S_1) satisfies $\llbracket e_1, \text{DELAY}[\alpha, \beta], s_1 \rrbracket$, then $E_1 \subseteq \Psi_{\text{DELAY}[\alpha, \beta]}(S_1)$.

2.3.2 Operator DUPL

$$\Psi_{\text{DUPL}}(A, B) \triangleq A \cap B$$

Again, if $(E_1, (S_1, S_2))$ satisfies $\llbracket e_1, \text{DUPL}, (s_1, s_2) \rrbracket$, then $E_1 \subseteq \Psi_{\text{DUPL}}(S_1)$.

2.3.3 Logic operators

Let us define:

$$\Psi_{\text{AND}}(A) \triangleq \{(x, y) \mid \exists a \in A, \forall t, x(t) \wedge y(t) = a(t)\}$$

Again, if $((E_1, E_2), S_1)$ satisfies $\llbracket (e_1, e_2) \text{AND} s_1 \rrbracket$, then $(E_1, E_2) \subseteq^2 \Psi_{\text{AND}}(S_1)$. We define the same way $\Psi_{\text{OP}}(A)$ for the other logic gates, as well as for the arithmetic gates.

2.3.4 Operator $\lll[\mu, \nu]$

$$\Psi_{\lll[\mu, \nu]}(A) \triangleq \left\{ x \left| \begin{array}{l} \exists a \in A \\ \exists C \in \mathbb{N} \rightarrow \mathbb{R} \text{ str. } \nearrow \end{array} \right| \begin{array}{l} \forall n, C(n+1) - C(n) \in [\mu, \nu] \\ \forall t \in [C(n), C(n+1)[\\ a(t) = x(C(n)) \end{array} \right\}$$

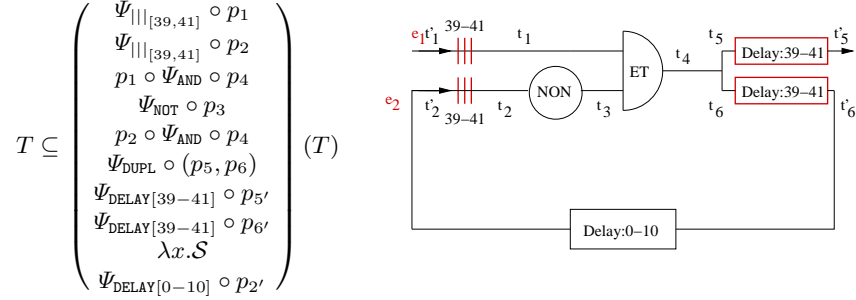
If (E_1, S_1) satisfies $\llbracket e_1, \lll[\mu, \nu], s_1 \rrbracket$, then $E_1 \subseteq \Psi_{\lll[\mu, \nu]}(S_1)$

2.3.5 Operator SHIFT

In the following, we won't consider any longer the **SHIFT** operator. It can, however be connected to concrete operator and, later, to an abstract one. $\text{SHIFT}_{[\mu, \nu]}$ is indeed equivalent to a $\text{DELAY}_{[\varepsilon, \varepsilon]}$ followed by a $\lll[\mu, \nu]$, where ε is very small compared to all the delays present in the box it comes from. Once divided into two gates, it may be transformed into two operators, either concrete or abstract ones.

2.3.6 Coding the diagram into operators

We are now able to connect any diagram S to an operator Ψ_S coding its wiring. For example, Ψ connects the diagram on the right to the table on the left. We let T denote the semantics of the diagram on the right, and it thus is a function. We write $\Psi_{\text{OP}} \circ p_i(T)$ for $\Psi_{\text{OP}} \circ T(i)$, and $p_i(T) \circ \Psi_{\text{OP}}$ for the i -th coordinate of the result of Ψ_{OP} . As T is the semantics of the diagram, for any control point \mathfrak{t} , $T(\mathfrak{t}) \subseteq p_{\mathfrak{t}} \circ \Psi_{\text{OP}_{\mathfrak{t}}} \circ p'_{\mathfrak{t}}(T)$, where $\text{OP}_{\mathfrak{t}}$ is the gate after \mathfrak{t} , $p_{\mathfrak{t}}$ and $p'_{\mathfrak{t}}$ represent the wiring with that gate.



The advantage of such a notation is that it is very easy to be translated into any abstract domain: we replace any concrete operator Ψ_{OP} by an abstract $\Psi_{\text{OP}}^\#$. These new abstract operators must however be *linked* to the concrete ones, so that the abstract properties possibly proved thanks to these abstract operators can be translated back into the concrete domain.

2.3.7 Expressing concrete properties with concrete operators and fixpoints

We said above that we have an operator Ψ_S built on all the operators Ψ_{OP} , satisfying the property: if T is the semantics of a diagram S , then $T \subseteq \Psi_S(T)$.

Let us try to express this property in a fixpoint form.

Let suppose we want to prove a property P on the signals in the semantics of a diagram S . Let $Z_{-P} : V \rightarrow \mathcal{P}(\mathcal{S})$ such that if $s_{v_1}, \dots, s_{v_n} \in Z_{-P}(v_1), \dots, Z_{-P}(v_n)$ then s_{v_1}, \dots, s_{v_n} doesn't satisfy P . We thus would like to prove the assertion $A \triangleq \llbracket S \rrbracket \cap Z_{-P} = \emptyset$. Now, since $A \subseteq \llbracket S \rrbracket$, $A \subseteq \Psi(A)$. Then $A = (\Psi \cap Id)(A)$, so that, by Tarski's fixpoint theorem,

$$A \subseteq \mathbf{gfp}_{Z_{-P}}(\Psi \cap Id)$$

Let us now try to prove that this fixpoint is empty. Introducing the operators, we solved the difficulty of controlling the propagation of the information. We now would like to take advantage of a better way to handle this information contained in the sets of signals. In particular, it should be easily handled by a computer program. We therefore design a new abstract domain.

3 A non-relational abstract domain: the constraints

3.1 The constraints

We define three types of constraints:

- A constraint denoted by $[a; b] : x$, meaning that any signal takes the value x at least once during the interval $[a; b]$.
- A constraint denoted by $\langle a; b \rangle : x$, meaning that any signal takes the value x during the whole interval $[a; b]$.
- A constraint `Abs_contr`, which denotes the absence of constraint.

We let \mathcal{Z} denote the set of all constraints.

3.2 Abstract domain

The abstract domain is $V \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{Z}))$. Each element is a disjunction of conjunctions of constraints, described in a disjunctive normal form. They represent the properties of the signals. At this point, we allow infinite conjunctions and disjunctions. However, we will see in Sec. 6 that we can avoid them, so that a computer is able to handle such elements.

3.3 Concretization map

The set of signals that satisfy an abstract property $P^\#$ is given by $\gamma(P^\#)$, where γ is the concretization map, defined as follows.

Definition 5 (Concretization map).

$$\left(\begin{array}{l} V \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{Z})) \quad \rightarrow \\ \lambda v. \{f_{i,v}, i \in I\} \quad \mapsto \quad \lambda v. \bigcap_{i \in I} \left(\bigcup_{([a,b]:y) \in f_{i,v}} \{x, \exists t \in [a,b], x(t) = y\} \right) \cup \left(\bigcup_{(\langle a; b \rangle : y) \in f_{i,v}} \{x, \forall t \in [a,b], x(t) = y\} \right) \end{array} \right)$$

Thus, for example :

- $\gamma(\{\emptyset\}) = \mathcal{S}$
- $\gamma(\{([0, 1] : 1) \wedge ([1, 2] : 0) \wedge \langle 2; 3 \rangle : 1\} \vee \{([0, 1] : 0) \wedge ([1, 2] : 1) \wedge \langle 2; 3 \rangle : 1\})$
 $\subseteq \{f \mid f \text{ switches from 0 to 1 or from 1 to 0 between 0 and 2, but is equal to 1 between 2 and 3}\}$

3.4 Pre-order, union, intersection

The abstract domain is pre-ordered by $\subseteq^\#$ defined by :

$$f^\# \subseteq^\# g^\# \Leftrightarrow \gamma(f^\#) \subseteq \gamma(g^\#)$$

We use a pre-order because a set of signals may be described by several unrelated abstract elements. This change doesn't affect our analysis, and will be discussed later. Indeed, some of the equivalent notations representing the same abstract element are easier to manipulate and we will try to favour them.

We also define $\cap^\#$ and $\cup^\#$, as usually on disjunctive normal forms.

4 Abstract operators

The abstract domain defined above isn't much simpler than the concrete one until we define abstract operators, enabling the translation of the fixpoint-based concrete properties into fixpoint-based abstract properties. We can now define abstract operators, either constraint by constraint, or directly. We thus define, for any operator OP, except $\text{|||}_{[\mu, \nu]}$ and DUPL:

$$\Psi_{\text{OP}}^{\#}(A^{\#}) = \left\{ \left\{ \dot{\Psi}_{\text{OP}}^{\#}(C) \mid C \in A_i^{\#} \right\} \mid A_i^{\#} \in A^{\#} \right\}$$

with

$$\begin{aligned} \Psi_{\text{DUPL}}^{\#}(A^{\#}, B^{\#}) &= A^{\#} \cap^{\#} B^{\#} \\ \dot{\Psi}_{\text{DELAY}[\alpha, \beta]}^{\#}([a, b] : x) &= ([a - \beta, b - \alpha] : x) \\ \dot{\Psi}_{\text{DELAY}[\alpha, \beta]}^{\#}(\langle a, b \rangle : x) &= (\langle a - \alpha, b - \beta \rangle : x) \\ \dot{\Psi}_{\text{ET}}^{\#}(I : \text{True}) &= (I : \text{True}, I : \text{True}) \\ \dot{\Psi}_{\text{OU}}^{\#}(I : \text{False}) &= (I : \text{False}, I : \text{False}) \\ \dot{\Psi}_{\text{NON}}^{\#}(I : x) &= (I : \neg x) \end{aligned}$$

Now, we recursively define:

$$\dot{\Psi}_{\text{|||}_{[\mu, \nu]}}^{\#}([a, b] : x) = ([a - \nu, b] : x)$$

and

$$\begin{aligned} &\Psi_{\text{|||}_{[\mu, \nu]}}^{\#}(A_1, \dots, A_{k-1}, \{C_1, \dots, C_{k-1}, (\langle a, b \rangle : x), C_k, \dots, C_n\}, A_{k+1}, \dots, A_n) \\ &\triangleq \Psi_{\text{|||}_{[\mu, \nu]}}^{\#}(A_1, \dots, A_{k-1}, \{C_i, i \in [1, n]\} \cup \{([t, t] : x), t \in [a, b]\}, A_{k+1}, \dots, A_n) \end{aligned}$$

5 Link between concrete and abstract properties

5.1 From concrete properties to abstract properties

We use the next theorem to link concrete properties and abstract properties:

Theorem 1. *For any Ψ defined above, $\Psi \circ \gamma \subseteq \gamma \circ \Psi^{\#}$*

It is easy to prove, considering each possible Ψ . We then are able to use the next theorem:

Theorem 2. ³ *If :*

- F and $F^{\#}$ are continuous.
- $F \circ \gamma \subseteq \gamma \circ F^{\#}$

³ variant of Proposition 25, [9]

– $A^\#$ is an abstract element such that $F(\gamma(A^\#)) \subseteq \gamma(A^\#)$ and $F^\#(A^\#) \subseteq A^\#$

then :

$$\mathbf{gfp}_{\gamma(A^\#)} F \subseteq \gamma(\mathbf{gfp}_{A^\#} F^\#)$$

Now, if, on the other hand, $Z_{\neg P} = \gamma(A^\#)$ and $\gamma(\mathbf{gfp}_{A^\#}(\Psi_S \cap Id)^\#) = \emptyset$ then it entails that $\mathbf{gfp}_{Z_{\neg P}}(\Psi_S \cap Id) = \emptyset$ so that the wanted property P is true.

5.2 Example

Let us consider the diagram on Fig. 4, and try to prove $P = \exists \delta \in \mathbb{R}, \forall t \in [\delta, \delta + 100], t'_5(t)$ is true. As we said above, we thus consider $\neg P$, and therefore try to propagate an hypothesis constraint $A^\# = \langle \delta, \delta + 100 \rangle : True$ in this diagram, for any δ .

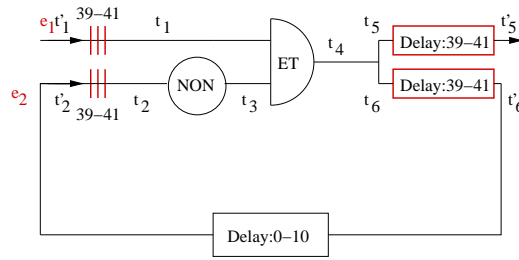


Figure 4.

| Step | Constraint function (control point \rightarrow constraints) |
|------|--|
| 1 | $t'_5 \rightarrow \langle \delta, \delta + 100 \rangle : True$ |
| 2 | $t_5 \rightarrow \langle \delta - 39, \delta + 59 \rangle : True$ |
| 3 | $t_4 \rightarrow \langle \delta - 39, \delta + 59 \rangle : True$ |
| 4 | $t_1 \rightarrow \langle \delta - 39, \delta + 59 \rangle : True$ $t_3 \rightarrow \langle \delta - 39, \delta + 59 \rangle : True$ |
| 5 | $t'_1 \rightarrow \bigwedge_{t \in [\delta - 39, \delta + 59]} ([t - 41, t] : True)$ $t_2 \rightarrow \langle \delta - 39, \delta + 59 \rangle : False$ |
| 6 | $t'_2 \rightarrow \bigwedge_{t \in [\delta - 39, \delta + 59]} ([t - 41, t] : False)$ |
| 7 | $t'_6 \rightarrow \bigwedge_{t \in [\delta - 39, \delta + 59]} ([t - 51, t] : False)$ |
| 8 | $t_6 \rightarrow \bigwedge_{t \in [\delta - 39, \delta + 59]} ([t - 82, t - 39] : False)$ |
| 9 | $t_4 \rightarrow \bigwedge_{t \in [\delta - 39, \delta + 59]} ([t - 82, t - 39] : False)$ |

The result is that the control point t_4 must satisfy two contradictory constraints, $\langle \delta - 39, \delta + 59 \rangle : True$ and $\bigwedge_{t \in [\delta - 39, \delta + 59]} ([t - 82, t - 39] : False)$. Indeed, $\mathbf{gfp}_{Z_{-P}}(\Psi \cap Id)^\# \subseteq^\# (\Psi)^\#{}^3(Z_{-P}) \cap^\# (\Psi)^\#{}^9(Z_{-P}) \subseteq^\# \emptyset^\#$. As a consequence, $\mathbf{gfp}_{Z_{-P}}(\Psi \cap Id) = \emptyset$. We then proved that $P = \exists \delta, \forall t \in [\delta, \delta + 100], t'_5(t)$ is true.

Here, the contradiction appeared after only nine steps, but we can imagine a much later convergence, or even a convergence after an infinity of steps. In that case, we would have to use a *widening* operator [10].

6 Improvements to the analysis

6.1 A transformation of diagrams

The main loss of information in our previous analysis is due to the imprecision of our abstract operators. The only way to solve this is to provide the operator representing a gate with more information about the gates around it. We now apply a transformation to diagrams in order to face this issue.

First, we label each gate P with $\sigma(P)$ representing its synchronicity group. Two gates are said synchronous if they have the same synchronicity group. This synchronicity group must satisfy:

- If P and P' are any gates (except DELAY and |||) only wired to inputs of the same box, they are synchronous.
- If P and P' are any gates (except DELAY and |||) only wired to synchronous inputs, they are synchronous.

Once this is done, let us define the *minimal stability*, denoted τ of control points. This *minimal stability* represents the minimum delay between two changes of the value of any signal at this control point:

- Any input of the diagram is connected to $\tau = 0$.
- Any output S of a sampler $||[\mu, \nu]$, of input E , is connected to $\tau(S) = \mu$.
- Any output S of a gate DELAY $[\alpha, \beta]$, of input E , is connected to $\tau(S) = \tau(E) - (\beta - \alpha)$.
- Any output S of a gate DUPL or NON, of input E , is connected to $\tau(S) = \tau(E)$.
- Any output S of a gate ET, OR or XOR, of **synchronous** inputs E , is connected to $\tau(S) = \tau(E_1) = \tau(E_2)$.
- Any output S of a gate ET, OR or XOR, of **non synchronous** inputs E , is connected to $\tau(S) = 0$.
- Any output S of a gate CONST $_\alpha$, of input E , is connected to $\tau(S) = \infty$.

For example, this new transformation, applied to Fig. 4, gives the result shown on figure Fig. 5. In this diagram, the gates NOT, DUPL et AND are synchronous, and labeled “A”. The control points are followed by their minimal stability.

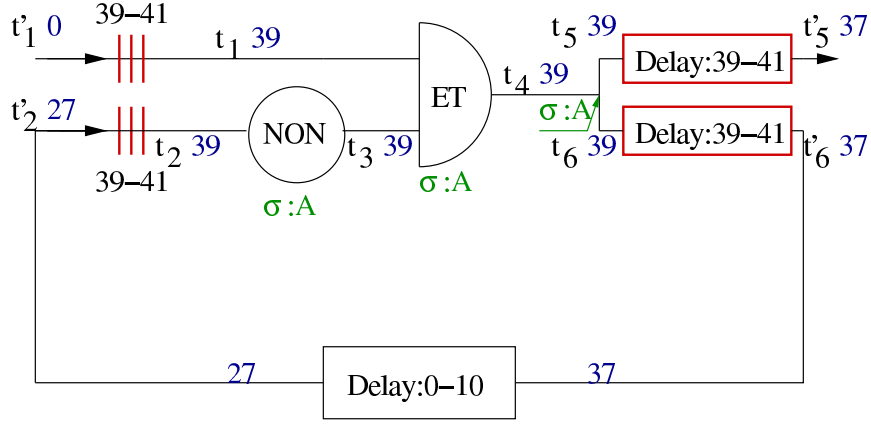


Figure 5. Fig. 4, modified

6.2 Modification of the operators $\Psi^\#$

The operators, as formerly defined, cannot be easily programmed. Indeed, the operator $\Psi^\#_{|||[\mu, \nu]}$, applied to $\langle a, b \rangle : x$ generates $\bigwedge_{t \in [a, b]} \dot{\Psi}^\#_{|||[\mu, \nu]}(\langle [t, t] : x \rangle)$, which cannot be easily handled by a computer.

$\Psi^\#_{|||[\mu, \nu]}$, instead of being computed on:

$$\Gamma_1 \vee \dots \vee (C_1 \wedge \dots \wedge C_i \wedge (\langle a; b \rangle : x) \wedge \dots \wedge C_k) \vee \dots \vee \Gamma_n$$

should now compute $\Psi^\#_{|||[\mu, \nu]}$ of

$$\Gamma_1 \vee \dots \vee \left(C_1 \wedge \dots \wedge C_i \wedge ([a, a] : x) \wedge \dots \wedge \left(\left[a + \left\lfloor \frac{(b-a)}{\eta} \right\rfloor \eta, a + \left\lfloor \frac{(b-a)}{\eta} \right\rfloor \eta \right] : x \right) \wedge \dots \wedge C_k \right) \vee \dots \vee \Gamma_n$$

This enables for example the equality:

$$\Psi^\#_{|||[\mu, \nu]}(\{\langle a, b \rangle : x\}) = \bigcap_{j \in [0; \lfloor \frac{(b-a)}{\eta} \rfloor]} \Psi^\#_{|||[\mu, \nu]}(\{\langle [a + j\eta, a + j\eta] : x \rangle\}) \text{ where}$$

$\eta = \frac{\mu}{k}$, is a parameter, with k an integer. This change is correct with respect to the theorem 1 of the Sec. 5.1, since it simply forgets some constraints. On the other hand, it appears that there won't be too much loss of information, since the former and the new constraint are equivalent (for $\subseteq^\#$), as far as the signals are restricted to those with a minimal stability superior or equal to μ .

6.3 Modification of the abstract domain

The abstract domain of Sec. 3 was non-relational. This means that the constraints on the different control points are independant. This is an issue if, for example one try to find which constraints on the inputs of a gate AND implies that its output satisfies $\langle a; b \rangle : false$. A *relational* abstract domain should therefore be defined.

Definition: The relational abstract domain is the disjunctive completion of the domain of Sec. 3, that is the set $\mathcal{Z}^{\vee \wedge} = \mathcal{P}(\mathcal{P}(V \rightarrow \mathcal{Z}))$, representing the disjunctions of the conjunctions of *functions* connecting control points to *one* constraint. We link the both abstract domains with two functions, that were implemented in `ocaml`.

Link with the non-relational domain

Let us consider: `List_tab` : $U \mapsto (W \text{ list}) \rightarrow (U \mapsto W) \text{ list}$, returning the list of the functions such that the n-th element of this list connects any element u of U to the n-th element of the list connected to u . In the following example, U is the set of integers and a function from U is represented as an array ($1 \mapsto x, 2 \mapsto y = [[x; y]]$):

$$\text{List_tab}([[1; 3; 5]; [2; 4; 6]]) = [[[1; 2]]; [[3; 4]]; [[5; 6]]]$$

Let us consider `Tab_list` : $(U \mapsto W) \text{ list} \rightarrow U \mapsto (W \text{ list})$, returning a function connecting each u of U to to all the elements connected to u by a function of the list in argument. For example, with the same conventions as before:

$$\text{Tab_list}([[[1; 2]]; [[3; 4]]; [[5; 6]]) = [[1; 3; 5]; [2; 4; 6]]$$

Let `map f l` be the function returning the list of the results of `f` applied to each element of `l`. Then, let

$$\alpha = \text{Tab_list} \circ (\text{map Tab_list})$$

and

$$\gamma = (\text{map List_tab}) \circ \text{List_tab}$$

(α, γ) is a Galois connection. We can thus define for any operator `OP`.

$$\Psi_{\text{OP}}^{\text{relational}} = \gamma \circ \Psi_{\text{OP}}^{\text{non-relational}} \circ \alpha$$

These new operator are thus very close to the former ones, and won't improve precision. We therefore now refine some of them.

Improving the operators let $\check{\Psi}_{(\sigma_1, \sigma_2, \text{ET}, \sigma_3)}^{\#}$ connects:

$$\begin{pmatrix} s_1 \rightarrow & d_1 \\ \vdots & \\ \sigma_3 \rightarrow [a, b] : false \\ \vdots & \\ s_p \rightarrow & d_p \end{pmatrix}$$

to

$$\left\{ \left(\begin{array}{c} s_1 \rightarrow d_1 \\ \vdots \\ \sigma_2 \rightarrow [a, b] : \\ \text{false} \\ \vdots \\ \sigma_1 \rightarrow [a, b] : \\ \text{false} \\ \vdots \\ s_p \rightarrow d_p \end{array} \right), \left(\begin{array}{c} s_1 \rightarrow d_1 \\ \vdots \\ \sigma_2 \rightarrow [a, b] : \\ \text{true} \\ \vdots \\ \sigma_1 \rightarrow [a, b] : \\ \text{false} \\ \vdots \\ s_p \rightarrow d_p \end{array} \right), \left(\begin{array}{c} s_1 \rightarrow d_1 \\ \vdots \\ \sigma_2 \rightarrow [a, b] : \\ \text{false} \\ \vdots \\ \sigma_1 \rightarrow [a, b] : \\ \text{true} \\ \vdots \\ s_p \rightarrow d_p \end{array} \right) \right\}$$

As for the universal constraint $\langle a; b \rangle : \text{false}$, it is transformed into a set of existential constraints $\{[a + k\eta; a + k\eta] : x, k \in [0; \lfloor \frac{(b-a)}{\eta} \rfloor]\}$. We already did such an operation in section 6.2. We then define $\dot{\Psi}_{(\sigma_1, \sigma_2, \text{ET}, \sigma_3)}^\#$ such that:

$$\dot{\Psi}_{(\sigma_1, \sigma_2, \text{ET}, \sigma_3)}^\#(\Gamma_1, \dots, \Gamma_k) = \left\{ a \wedge b, \begin{array}{l} a \in \dot{\Psi}_{(\sigma_1, \sigma_2, \text{ET}, \sigma_3)}^\#(\Gamma_1) \\ b \in \dot{\Psi}_{(\sigma_1, \sigma_2, \text{ET}, \sigma_3)}^\#(\Gamma_2, \dots, \Gamma_n) \end{array} \right\}$$

Lastly,

$$\Psi_{(\sigma_1, \sigma_2, \text{ET}, \sigma_3)}^\#(\Gamma_1, \dots, \Gamma_n) = (\dot{\Psi}_{(\sigma_1, \sigma_2, \text{ET}, \sigma_3)}^\#(\Gamma_1), \dots, \dot{\Psi}_{(\sigma_1, \sigma_2, \text{ET}, \sigma_3)}^\#(\Gamma_n))$$

A similar transformation is of course also applied to $\Psi_{\text{OR}}^\#$ and $\Psi_{\text{XOR}}^\#$. These new operators face the former problem of the loss of all information on some constraints by these operators.

7 Conclusion

The analyses we presented were easily implemented in the `ocaml` language, enabling to prove, the same way we did it for the example presented section 5.2, temporal properties of several diagrams, containing more than 20 gates. The continuous-time semantics we used solved many difficulties often met when analyzing these systems, in particular the exhaustive exploration of all the interleavings. We choosed, in order to prove these properties, a very simple abstract domain, which enabled all the same to prove the properties we submitted to our analyzer.

References

1. Albert Benveniste, Paul Caspi, Paul Le Guernic, Hervé Marchand, Jean-Pierre Talpin, and Stavros 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. 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.
4. 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.
5. Paul Caspi, Daniel Pilaud, Nicolas Halbwachs, and John Plaice. Lustre: A declarative language for programming synchronous systems. *Proceedings of the 14th ACM symposium on Principles of programming languages, POPL'87*, 1987.
6. Paul Caspi and Rym Salem. Threshold and bounded-delay voting in critical control systems. *Vol. 1926 of Lecture Notes in Computer Science*, September 2000.
7. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 238—252, Los Angeles, California, 1977*.
8. P. Cousot and R. Cousot. Constructive versions of tarski's fixed point theorems. *Pacific Journal of Mathematics, Vol. 82, No. 1, pp. 43—57*, 1979.
9. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming, 13(2-3):103—179*, 1992.
10. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. pages 269–295, 1992.
11. Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation, 2(4):511—547*, August 1992.
12. S. Thompson and A. Mycroft. Abstract interpretation of asynchronous circuits. *SAS, Verona, Italy, August 2004*.