Model checking robustness to desynchronization

Jean-Pierre Talpin
INRIA *project* ESPRESSO - IRISA, *Campus de Beaulieu, 35042 Rennes cedex, France*

Abstract The engineering of an everyday broader spectrum of systems requires reasoning on a combination of synchronous and asynchronous interaction, ranging from co-designed hardware-software architectures, multi-threaded reactive systems to distributed telecommunication applications. Stepping from the synchronous specification of a system to its distributed implementation requires to address the crucial issue of desynchronization: how to preserve the meaning of the synchronous design on a distributed architecture ? We study this issue by considering a simple SCCS-like calculus of synchronous processes. In this context, we formulate the properties of determinism and of robustness to desynchronization. To check a specification robust to desynchronization, we consider a canonical representation of synchronous processes that makes control explicit. We show that the satisfaction of the property of determinism and of robustness to desynchronization amounts to a satisfaction problem which consists of hierarchically checking boolean formula.

1. INTRODUCTION

Synchronous programming [1, 3, 8] has been proposed as an efficient approach for a trusted design of reactive systems. It has been widely publicized, using the idealized model of zero-time computation and instantaneous broadcast communication. Distributed systems do not, however, obey this idealized picture of perfect synchrony: computations and communications take time, interaction topologies evolve during service. Synchrony and asynchrony are fundamentally different concepts in nature. Asynchrony is traditionally relevant for reasoning on distributed algorithms and for modeling non-determinism, failure, mobility. It meets a natural implementation by networked point-to-point communication. Synchrony is specific to the design of reactive systems and digital circuits. In this context, timeless logical concurrency and determinism are suitable hypothesis. A synchronous design hypothesis consists of assuming that communications and computations are instantaneous between successive execution steps of a system. Making this hypothesis allows one to focus on the logics of the system, which is characterized by synchronization and causal

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35599-3_29

B. Kleinjohann et al. (eds.), *Design and Analysis of Distributed Embedded Systems*

relations between events. By contrast, time prevail in an asynchronous design. Communication time is to be taken into account at every level of the system under design. Nonetheless, an everyday broader range of software development areas requires reasoning on a combination of synchronous and asynchronous interaction at the different architectural levels of the system under design. Relevant practical examples are co-designed hardware-software architectures, reconfigurable embedded devices, multi-threaded reactive systems components on real-time virtual machines and operating systems, distributed and reactive telecommunication applications on fault-tolerant middle-ware. In summary, every system whose design requires robustness to latency, to distribution, to threading. In the present article, we formulate the issue of robustness in the algebraic and operational setting of a calculus of synchronous processes. We start by giving a structured operational semantics of synchronous processes. Then, we characterize the synchronous and desynchronized traces of synchronous processes. We probe the minimality and adequacy of this setting by formulating the properties of determinism (defined by the equivalence between the synchronous (internal) and asynchronous (external) observations) and of robustness to desynchronization (defined by the mutual acceptability of desynchronized traces by synchronous processes). To check desynchronization correct, we consider a representation of processes in terms of polynomials dynamical equations over the ring $\mathbb{Z}/3\mathbb{Z}$. We show that the satisfaction of the properties of determinism and of robustness to desynchronization is amenable to model-checking the corresponding invariant expressed as a (vector of) constraint(s) over $\mathbb{Z}/3\mathbb{Z}$.

2. COMMUNICATING SYNCHRONOUS PROCESSES

In the signal calculus, a process p consists of elementary actions await $x(u)$ and emit $x(u)$ combined using synchronous composition $p \times q$ and non-deterministic choice $p+q$. We use an infinite countable set of names to denote processes $f \in \mathcal{F}$, signals $x, y \in \mathcal{X}$ and variables $v, w \in \mathcal{V}$ (we assume $\mathcal{F}$, $\mathcal{X}$ and $\mathcal{V}$ disjoint). Meta-variables are noted $c \in \mathbb{B} = \{tt, ff\}$ for constants, $m, n \in \mathcal{N} = \mathcal{X} + \mathcal{V}$ for names and $u \in \mathcal{U} = \mathcal{V} + \mathbb{B}$ for parameters. We write $\tilde{n}$ for a sequence of names. The actions await $x(u)$ and emit $x(u)$ implement synchronous broadcast communications (a denotes the prefix await or emit of an action). The action emit $x(u)$ immediately broadcasts the value u along the signal x. The action await $x(u)$ instantaneously receives a value along the signal x and expect this value to match u. The silent action nil is defined by rec $f()$.next $f()$. Simultaneous actions p and q are combined using synchronous composition $p \times q$ and non-deterministic choice $p+q$. Restriction $(n)p$ limits the scope of a signal or variable n to a process p. The evolution of a process in time is modeled by recursion and unit delay. A recursive process rec $f(v).p$ consists of a process p, a parameter (v) and a name f. Its recursive call with the

actual parameter u is written $f(u)$ and prefixing it by next delays its execution in time.

$$p, q ::= \mathsf{nil} \mid \mathsf{await}\, x(u) \mid \mathsf{emit}\, x(u) \mid p \times q \mid p + q \mid (n)p \mid (\mathsf{rec}\, f(v).p)(u) \mid f(u) \mid \mathsf{next}\, p$$

We write $p[x/y]$ for the substitution of y by x in p and $\mathrm{dom}\, S$ for the domain of a substitution S. We write $\mathrm{fv}(p) \subset \mathcal{N}$, $\mathrm{dv}(p) \subset \mathcal{X}$ and $\mathrm{rv}(p) \subset \mathcal{V}$ for the set of free names, defined signals and receiving variables of a process. In particular, $\mathrm{rv}(\mathsf{await}\, x(v)) = \{v\}$, $\mathrm{rv}(\mathsf{emit}\, x(u)) = \mathrm{rv}(\mathsf{await}\, x(c)) = \mathrm{rv}(f(u)) = \emptyset$ and $\mathrm{dv}(\mathsf{await}\, x(u)) = \mathrm{dv}(\mathsf{emit}\, x(u)) = \{x\}$. The *operational semantics* of processes p is formally defined by the structural relation $p \equiv q$ and by the transition relation $p \xrightarrow{\boldsymbol{e}} q$. The structural equivalence relation $p \equiv q$ gives a structure of sets to $(\mathcal{P}, +)$ and of monoid to $(\mathcal{P}, \times, \mathsf{nil})$.

$$\begin{array}{lllll} p\times(q\times r)\equiv(p\times q)\times r & p+q\equiv q+p & (m)p\equiv(n)(p[n/m]) & {\scriptstyle n\notin \mathrm{fv}(p)} \\ p+(q+r)\equiv(p+q)+r & p\times q\equiv q\times p & p\times(n)q\equiv(n)(p\times q) & \\ (n)\mathsf{nil}\equiv\mathsf{nil} \quad p\times\mathsf{nil}\equiv p+p\equiv p & (n)(m)p\equiv(m)(n)p & p+(n)q\equiv(n)(p+q) & \end{array}$$

In the present article, we exclusively consider tail-recursive processes, by syntactically limiting the number of guarded recursive calls $\mathsf{next}\, f\tilde{u}$ which may occur within the body p of a recursive process definition $\mathsf{rec}\, f\tilde{v}.p$. The relation $\mathcal{T}_1$ (resp. $\mathcal{T}_0$) accepts exactly the processes p which have at most one (resp. zero) delayed recursive call per choice branch.

$$\frac{}{\mathcal{T}_0[\![\mathsf{await}\, x(u)]\!]} \quad \frac{}{\mathcal{T}_0[\![\mathsf{emit}\, x(u)]\!]} \quad \frac{}{\mathcal{T}_0[\![\mathsf{nil}]\!]} \quad \frac{}{\mathcal{T}_1[\![\mathsf{next}\, f(u)]\!]} \quad \frac{\mathcal{T}_0[\![p]\!] \;\; \mathcal{T}_0[\![q]\!]}{\mathcal{T}_0[\![p\times q]\!]} \quad \frac{\mathcal{T}_0[\![p]\!] \;\; \mathcal{T}_1[\![q]\!]}{\mathcal{T}_1[\![p\times q]\!]} \quad \frac{\mathcal{T}_0[\![p]\!]}{\mathcal{T}_1[\![p]\!]} \quad \frac{\mathcal{T}_i[\![p]\!]}{\mathcal{T}_i[\![(n)p]\!]}$$

$$\frac{\mathcal{T}_i[\![p]\!] \;\; p \equiv q}{\mathcal{T}_i[\![q]\!]} \quad \frac{\mathcal{T}_i[\![p]\!]}{\mathcal{T}_i[\![\mathsf{next}\, p]\!]} \quad \frac{\mathcal{T}_i[\![p]\!]}{\mathcal{T}_i[\![(\mathsf{rec}\, f\tilde{v}.(\mathsf{next}\, f(v)+p))(u)]\!]} \quad \frac{\mathcal{T}_i[\![p]\!] \;\; \mathcal{T}_i[\![q]\!]}{\mathcal{T}_i[\![p+q]\!]} \;\; {\scriptstyle \forall i\in\{0,1\}}$$

The term $\boldsymbol{e}$ in the transition relation $p \xrightarrow{\boldsymbol{e}} q$ is a partial function of $\mathcal{E} = \mathcal{X} \rightharpoonup (\mathbb{B} + \{\bot\})$ which represents the *context or environment* of a process at the instant at which an execution step from p to q takes place. It defines the *events* that occur at that instant. An event $x \mapsto u$ of $\boldsymbol{e}$ (also written $\boldsymbol{e}(x) = u$) denotes the value u of the signal x at the (logical) instant denoted by $\boldsymbol{e}$. The signal x can alternatively be regarded as absent at a given instant. This is denoted by associating x to the mark $\bot$ in $\boldsymbol{e}$ (i.e. $\boldsymbol{e}(x) = \bot$). The rule **(eqv)** embeds the structural equivalence relation in the operational semantics. It allows for the syntactic recombination of processes. The semantics of $(n)p$ depends on whether n is a receiving variable or a defined signal of p. In **rule (sub)**, the variable v is substituted by a value c in p. The meaning of restriction $(x)p$, **rule (let)**, is to limit the scope of the signal x to the expression p. We write $\boldsymbol{e}_x$ for $\boldsymbol{e}$ without x (i.e. $x \notin \mathrm{dom}\, \boldsymbol{e}_x$). **Rule (com)** is the axiom of communication. An atomic action await $x(c)$ or emit $x(c)$ reduces to nil by associating x to c in the environment. **Rule (or)** is the choice rule. It enables a transition from $p+q$ to r with $\boldsymbol{e}$ if a transition from the branch p to r with $\boldsymbol{e}$ is possible. **Rule (and)** implements synchronous composition. It stipulates that the simultaneous transitions from p to p' with $\boldsymbol{e}$ and from q to q' with $\boldsymbol{f}$ are valid if and only if the environments $\boldsymbol{e}$ and $\boldsymbol{f}$ agree on the assignment to all signals shared by

p and q (they behave like p and q for all other signals). This is specified by the side-condition to the construction of the composition $e \uplus f$ of the partial functions e and f, which requires $e(x)$ and $f(x)$ to be equal for all x shared by $\mathrm{dom}\, e$ and $\mathrm{dom}\, f$, and by further requires e and f to be defined at least on all defined signals of p and q. **Rule (fix)** handles the call $(\mathrm{rec}\, f(v).p)(c)$ to a recursive process f. As in SCCS, it requires a transition of p where (v) is substituted by (c) and f is unfolded by $\mathrm{rec}\, f(v).p$. **Rule (nxt)** is the axiom for unit delay $\mathrm{next}\, p$. Its sole purpose is to put off the execution of p until the next step. It requires an empty environment.

$$\text{(eqv)}\ \frac{p \equiv p'\ \ p' \xrightarrow{e} q'\ \ q' \equiv q}{p \xrightarrow{e} q} \qquad \text{(sub)}\ \frac{p[c/v] \xrightarrow{e} q}{(v)p \xrightarrow{e} q} \qquad \text{(let)}\ \frac{p \xrightarrow{e} q}{(x)p \xrightarrow{e_x} (x)q}$$

$$\text{(or)}\ \frac{p \xrightarrow{e} r}{p+q \xrightarrow{e} r} \qquad \text{(and)}\ \frac{p \xrightarrow{e} p'\ \ q \xrightarrow{f} q'}{p \times q \xrightarrow{e \uplus f} p' \times q'}\ {\scriptstyle \forall x \in \mathrm{dv}(p) \cap \mathrm{dv}(q)} \begin{pmatrix} {\scriptstyle x \in \mathrm{dom}\, e \cap \mathrm{dom}\, f} \\ {\scriptstyle e(x) = f(x)} \end{pmatrix}$$

$$\text{(fix)}\ \frac{(p[c/v])[\mathrm{rec}\, f(v).p/f] \xrightarrow{e} q}{(\mathrm{rec}\, f(v).p)(c) \xrightarrow{e} q} \qquad \text{(com)}\ \mathrm{await}\, x(c) \xrightarrow{x \mapsto c} \mathrm{nil} \qquad \mathrm{emit}\, x(c) \xrightarrow{x \mapsto c} \mathrm{nil}$$

$$\text{(nxt)}\ \mathrm{next}\, p \xrightarrow{\emptyset} p \qquad \text{(nil)}\mathrm{nil} \xrightarrow{e} \mathrm{nil}\ {\scriptstyle (\forall x \in \mathrm{dom}\, e, e(x) = \bot)}$$

Example 1 *In order to understand the semantics of the signal calculus, let us for instance consider a simple counting process* even, *which samples every even occurrence of the signal x by emitting the signal y. The state of the counter is stored in place of the actual parameter of the process. The process* even *makes use of the syntax $[u = v]$ for guards. A transition across a guard $[u = v]$ is allowed iff the names u and v match. A guard can be defined using a receiving name w and a pair of intermediate ports x and y: $[u = v] \equiv (w, x, y)(\mathrm{emit}\, x(u) \times \mathrm{emit}\, y(v) \times \mathrm{await}\, x(w) \times \mathrm{await}\, y(w))$. If the signal x is present (e.g. $e = x()$) then the process* even *reacts to it by unfolding its definition (by substituting the name* even *by the term f_{even}) and by substituting its formal parameter u by its actual parameter ff. Then, the only possible choice of the process* even *is to match e_1 with the action of awaiting x and to cross the process p_1 guarded by the tautology $[ff = ff]$, yielding $\mathrm{nil} \times (\mathrm{nil} \times \mathrm{next}\, f_{\mathsf{even}}(tt)) \equiv \mathrm{next}\, f_{\mathsf{even}}(tt)$. Finally, the process becomes $f_{\mathsf{even}}(tt)$ after stepping the* next *statement, changing the value of its actual parameter to tt, in order to emit y next time the signal x occurs, by forçing the selection of the branch p_2.*

$$\overbrace{\mathrm{rec}\, \mathsf{even}(u).\left(\mathrm{next}\, \mathsf{even}(u) + \left(\mathrm{await}\, x() \times \left(\begin{matrix} ([u = ff] \times \mathrm{next}\, \mathsf{even}(tt)) \\ +([u = tt] \times \mathrm{next}\, \mathsf{even}(ff) \times \mathrm{emit}\, y())\end{matrix}\right)\right)\right)(ff)}^{f_{\mathsf{even}}}$$

$$\mathrm{next}\, f_{\mathsf{even}}(ff) + (\mathrm{await}\, x() \times (\overbrace{([ff = ff] \times \mathrm{next}\, f_{\mathsf{even}}(tt))}^{p_1} + \overbrace{([ff = tt] \times \mathrm{next}\, f_{\mathsf{even}}(ff) \times \mathrm{emit}\, y())}^{p_2}))$$

3. DETERMINISM AND ROBUSTNESS

In order to determine under which properties the synchronous design of a system can safely be distributed on an asynchronous network, we establish formal connections between synchrony and asynchrony. We start by giving a formal definition of the properties under consideration. A *synchronous trace* consists of an infinite countable series of events that correspond to the successive transitions of a process in time.

Definition 1 (trace) *If* $p_0 = p$ *and* $p_i \xrightarrow{e_i} p_{i+1}$ *for all* $i > 0$ *then the series* $t = (e_i)_{i \geq 0}$ *is a* synchronous trace *of* p. *We say that* p *and* q *are* synchronously equivalent, *written* $p \simeq q$, *iff* p *and* q *accept the same synchronous traces.*

An *desynchronized trace* $T \in \hat{\mathcal{E}}^*$ is obtained from a synchronous trace t by removing absence: $E, F \in \hat{\mathcal{E}} = \mathcal{X} \rightharpoonup \mathbb{B}$. Formally, $\hat{\emptyset} = \emptyset$, $\widehat{e, x \mapsto \bot} = \hat{e}$, $\widehat{e, x \mapsto c} = (\hat{e}, x \mapsto c)$. Hence $\hat{t} = (\hat{e_i})_{i \geq 0}$ is the asynchronous abstraction of a trace $t = (e_i)_{i \geq 0}$. Intuitively, a desynchronized trace respects the ordering of events along signals in time, but discards reference to absence. We relate desynchronized traces T with the partial order relation $\leq$. The ordering of traces under the relation $\leq$ renders the loss of a global reference of time incurred by the removal of absence yet preserves the causal ordering between successive events. For all $T = (E_i)_{i \geq 0}$, we write $\mathrm{dom}\, T = \cup_{i \geq 0} \mathrm{dom}\, (E_i)$.

Definition 2 (desynchronization) T' *is a desynchronization of* T, *written* $T \leq T'$, *iff* $T \leq^x T'$ *holds for all* $x \in \mathrm{dom}\, T \cap \mathrm{dom}\, T'$. *We write* $ET \leq^x FT'$ *iff, either* $x \notin \mathrm{dom}\, F$ *and* $ET \leq^x T'$, *or* $E(x) = F(x)$ *and* $T \leq^x T'$.

It is useful to relate synchronous traces under the equivalence relation of stuttering. A transition $p \xrightarrow{e} q$ is stuttering iff it is silent i.e. $\mathrm{im}\, e = \{\bot\}$. We write $t \lessapprox t'$ iff t and t' only differ by silent transitions e. In the remainder, we write $\mathcal{S}[\![p]\!]$ for the set of synchronous traces of a process p, $\hat{\mathcal{S}}[\![p]\!] = \{\hat{t} \mid t \in \mathcal{S}[\![p]\!]\}$ for its asynchronous abstraction and $\mathcal{S}_{\lessapprox}[\![p]\!]$ for its equivalence classes for the relation of stuttering $\lessapprox$. We write $\mathcal{A}[\![p]\!] = \{T \mid t \in \mathcal{S}[\![p]\!] \wedge \hat{t} \leq T\}$ for the set of desynchronized (or admissible) traces of a process p.

Definition 3 *Traces* et *and* $e't'$ *are stuttering-equivalent, written* $et \lessapprox e't'$ *iff* $\mathrm{im}\, e = \{\bot\}$ *and* $t \lessapprox e't'$; *or* $\mathrm{im}\, e' = \{\bot\}$ *and* $et \lessapprox t'$; *or* $e = e'$ *and* $t \lessapprox t'$.

The property of determinism is defined by the equivalence between the internal (synchronous) and external (asynchronous) observations of a process. A process is said deterministic iff, given an asynchronous trace T of p, it is possible to reconstruct a synchronous trace t that is unique modulo stuttering and such that $\hat{t} \leq T$ (i.e. the trace t of p accepts T). This means that every asynchronous trace of the process corresponds to a synchronous trace in which the successive instants of the execution have been reconstructed from the values of signals present in the asynchronous trace. Concretely, a deterministic process forms a unit of compilation: interaction with a deterministic process does not require any knowledge on its internal clock.

Definition 4 p *is deterministic iff* $\forall T \in \mathcal{A}[\![p]\!], \exists! t \in \mathcal{S}_{\lessapprox}[\![p]\!], \hat{t} \leq T$.

The property of robustness to desynchronization is defined by the considering the desynchronized traces of p, q and $p \times q$. The desynchronized composition

of p and q can be modeled by $\mathcal{A}[\![p]\!] \cap \mathcal{A}[\![q]\!]$ because the intersection of $\mathcal{A}[\![p]\!]$ and $\mathcal{A}[\![q]\!]$ corresponds to the traces T which are both desynchronizations of a synchronous trace t of p (i.e. $t \in \mathcal{S}[\![p]\!]$ and $\hat{t} \leq T$ hence $T \in \mathcal{A}[\![p]\!]$) and of a synchronous trace of trace t' of q (i.e. $t' \in \mathcal{S}[\![p]\!]$ and $\hat{t'} \leq T$ hence $T \in \mathcal{A}[\![q]\!]$). If $\mathcal{A}[\![p \times q]\!] = \mathcal{A}[\![p]\!] \cap \mathcal{A}[\![q]\!]$ holds, this means that all desynchronized traces T of $p \times q$ uniquely correspond via the relations $\hat{t} \leq T$ and $\hat{t'} \leq T$ to a pair of synchronous traces of p and q. Hence, the desynchronized interaction between p and q is deterministic, just as it is for synchronous composition, and no global synchronization is not required to achieve it. The property of robustness ensures that the synchronous design of a system supports the distribution of its components over an asynchronous network without loss of semantics.

Definition 5 *$p \times q$ is robust to desynchronization iff $\mathcal{A}[\![p \times q]\!] = \mathcal{A}[\![p]\!] \cap \mathcal{A}[\![q]\!]$*

In order to verify that a synchronous process is deterministic, or that a pair of synchronous processes is robust to desynchronization, we make use of the tool SIGALI [9]. SIGALI is a model-checker that implements resolution techniques on systems of equations expressed in the $\mathbb{Z}/3\mathbb{Z}$ ring. In SIGALI, a system equations characterizes a set of solutions for the states and events of a process. The resolution technique consists of manipulating the equation system, instead of the solution sets, in order to avoid the enumeration of state-spaces. In order to model the behavior of a synchronous process, we encode each of its actions by an equation $\phi = \psi$ over $\mathbb{Z}/3\mathbb{Z}$. In either arms of this equation, a signal or variable n is encoded by its possible states: 0 if it is absent, 1 if it is present and true, -1 if it is present and false. Formula ϕ and ψ are composed by multiplication $\phi.\psi$ (ϕ and ψ), addition $\phi + \psi$ (ϕ or ψ) and subtraction $\phi - \psi$ (ϕ and not ψ). We write $x^2 = x.x$ for the clock of the signal x (it tells whether x is present or absent). A primed variable v' denotes the next value of v. A scored name v_0 denotes the initial value of v. Composition $\Phi \wedge \Psi$ and scope-restriction $\exists n.\Phi$ define the translation of a synchronous process into a system of equations in $\mathbb{Z}/3\mathbb{Z}$ that describes the evolution of the defined signals $\tilde{x} = \mathrm{dv}(p)$ and receiving variables $\tilde{v} = \mathrm{rv}(p)$ of a process p in time. In this section, we identify the term $\tilde{x}$ (resp. $\tilde{v}$) to an event represented by a vector of $\mathbb{Z}/3\mathbb{Z}^{|\mathrm{dv}(p)|}$ (resp. a state of $\mathbb{Z}/3\mathbb{Z}^{|\mathrm{rv}(p)|}$). We write $X \subset \mathbb{Z}/3\mathbb{Z}^{|\mathrm{dv}(p)|}$ (resp. $V \subset \mathbb{Z}/3\mathbb{Z}^{|\mathrm{rv}(p)|}$) for a set of events and states. The function $G_p(\tilde{x})$ defines the guard or clock of the process p. The process p is active iff its clock $G_p(\tilde{x})$ equals 1. The function $I_p(\tilde{v})$ defines an initial assignment of values to variables $\tilde{v}$ which must equal 0. The constraint $C_p(\tilde{x}, \tilde{v})$ defines an assignment of values to signals $\tilde{x}$ and variables $\tilde{v}$ which must always equal 0. The transition function $T_p(\tilde{x}, \tilde{v})$ defines the next values $\tilde{v}'$ of the variables $\tilde{v}$ in p as a function of the current values of signals $\tilde{x}$ and variables $\tilde{v}$.

$$\begin{array}{rl} \phi, \psi ::= 0 \mid 1 \mid -1 \mid n \mid v' \mid v_0 \mid \phi.\psi \mid \phi + \psi \mid \phi - \psi & \text{(formula)} \\ \Phi, \Psi ::= (\phi = \psi) \mid \Phi \wedge \Psi \mid \exists n.\Phi & \text{(equation)} \\ (I_p(\tilde{v}) = 0,\ T_p(\tilde{x}, \tilde{v}) = \tilde{v}',\ G_p(\tilde{x}) = 1,\ C_p(\tilde{x}, \tilde{v}) = 0) & \text{(constrained transition system)} \end{array}$$

SIGALI [9] uses the theory of algebraic geometry and, in particular, operations on varieties, ideals and morphisms, in order to define and prove prop-

erties of systems such as liveness and invariance. Let $X \subset \mathbb{Z}/3\mathbb{Z}^{|\mathrm{dv}(p)|}$ a set of events, $V \subset \mathbb{Z}/3\mathbb{Z}^{|\mathrm{rv}(p)|}$ a set of states and consider the quotient ring of polynomial functions $\mathcal{A}[XV] = \mathbb{Z}/3\mathbb{Z}[XV]/(X^3 - X, V^3 - V)$. An ideal of the sets of events X and states V in $\mathcal{A}$ is defined by $\mathcal{I}(XV) = \{\phi \in \mathcal{A}[XV] \,|\, \forall(\tilde{x}, \tilde{v}) \in X \times V, \phi(\tilde{x}, \tilde{v}) = 0\}$. It represents the set of equations which have the solution XV. Reciprocally, to any system of equations Φ, the variety $\mathcal{V}(\Phi) = \{(\tilde{x}, \tilde{v}) \in X \times V \,|\, \forall \phi \in \Phi, \phi(\tilde{x}, \tilde{v}) = 0\}$ represents the set of states and events acceptable by Φ. There exists a direct correspondence between ideals and variety: $\mathcal{V}(\mathcal{I}(XV)) = XV$ and $\mathcal{I}(\mathcal{V}(\langle\Phi\rangle)) = \langle\Phi\rangle$ where $\langle\Phi\rangle$ is the set of linear combinations of the system of equations Φ. As a consequence, an ideal can be represented by a single equation, its principal generator. To capture the evolution of a system, one regards the transition function T as a morphism of $(\mathbb{Z}/3\mathbb{Z})^{|\mathrm{fv}(p)|} \to (\mathbb{Z}/3\mathbb{Z})^{|\mathrm{rv}(p)|}$ that defines post-conditions. The associated comorphism $T^* \in (\mathbb{Z}/3\mathbb{Z})^{\mathrm{rv}(p)|} \to (\mathbb{Z}/3\mathbb{Z})^{|\mathrm{fv}(p)|}$ defines the pre-conditions by $T^*(\phi(\tilde{v})) = \phi(T_i(\tilde{x}, \tilde{v}))_{1 \leq i \leq ||\mathrm{rv}(p)|}$ for any $\phi \in (\mathbb{Z}/3\mathbb{Z})^{|\mathrm{rv}(p)|}$ (The series $(T_i)_{0<i\leq|\mathrm{rv}(p)|}$ are the components of T). Many properties of systems of equations can be verified using operations of varieties, ideals, morphisms and comorphisms. One important is liveness. We say that a system is alive iff it cannot reach a state from which no transition can be taken (a deadlock). We henceforth restrict the study of further properties to such systems, in which all trajectories are infinite. It is proved in [9] that the liveness of a system under the set of constraints C can be stated as $T^*(\langle C\rangle \cap \mathbb{Z}/3\mathbb{Z}[V]) \subseteq \langle C\rangle$. It is implemented by a fixed-point iteration algorithm.

Definition 6 (liveness) *Let p be a process characterized by (I, T, G, C). A state $\tilde{v} \in (\mathbb{Z}/3\mathbb{Z})^{|\mathrm{rv}(p)|}$ is alive if there exists an event $\tilde{x} \in (\mathbb{Z}/3\mathbb{Z})^{|\mathrm{dv}(p)|}$* s.t. *$C(\tilde{x}, \tilde{v}) = 0$. A set of states W is alive iff every state $\tilde{v} \in W$ is alive. A system is alive iff, for all $(\tilde{x}, \tilde{v})$* s.t. *$C(\tilde{x}, \tilde{v}) = 0$, $T(\tilde{x}, \tilde{v})$ is alive.*

The second property of interest in the present article is invariance. The tool SIGALI allows to verify other properties, such as invariance under control, reachability and attractivity [9]. It is proved in [9] that the invariance of a property represented by a set of states W can be stated as $T^*(\mathcal{I}(W)) \subseteq \langle C\rangle + \mathcal{I}(W)Z/3\mathbb{Z}[XV]$.

Definition 7 (invariance) *Let p be a process characterized by (I, T, G, C). A subset of state $W \subset (\mathbb{Z}/3\mathbb{Z})^{|\mathrm{rv}(p)|}$ is invariant iff for every $\tilde{v} \in W$ and every $\tilde{x} \in (\mathbb{Z}/3\mathbb{Z})^{|\mathrm{dv}(p)|}$* s.t. *$C(\tilde{x}, \tilde{v}) = 0$, the state $T(\tilde{x}, \tilde{v})$ is in W.*

The recursive function $\Phi[\![p]\!]$ translates a tail-recursive process p into the system of equations (I, T, G, C) that characterizes its control. It is defined by induction on the structure of p. For a recursive definition $\mathrm{rec}\, f(v).p$, we associate the formal parameter v to f by scoring v with f. The system of equations $\Phi[\![p]\!]$ is a point-wise translation of the meaning of the process p, as

specified by the relation $p \rightarrow^{e} q$. In the rule for synchronous product $p \times q$, we write $T_{p\restriction\tilde{x}}$ for the completion of T_p to the signals $\tilde{x}$ which are present in q and absent from p. Since T_p is, by construction, of the form $\wedge_i(\phi_i.T_p^i)$, the completion of T_p by the signals $\tilde{x}$ is defined by the completion of all T_p^i with $y = 0$, for all $y \in \tilde{x}$ and not referenced in T_p^i: $(\wedge_i \phi_i.(T_p^i))_{\restriction\tilde{x}} = \wedge_i \phi_i.(T_p^i \wedge (\bigwedge_{y \in \tilde{x} \setminus \mathrm{fv}(T_p^i)}(y = 0)))$. In the rule for non-deterministic choice $p+q$, we identify $\phi.(\psi = \psi')$ to $\phi.\psi = \phi.\psi'$ for ϕ a clock and $v = \phi \wedge v = \psi$ to $v = \phi.\psi$. The requirement of determinism between the branches p or q of the choice is rendered by imposing the constraint $\phi_p.\phi_q = 0$. We write $[\![u]\!]$ for the encoding of a parameter u in $\mathbb{Z}/3\mathbb{Z}$: $[\![v]\!] = v$, $[\![tt]\!] = 1$, $[\![ff]\!] = -1$.

$$\Phi[\![\mathsf{await}\, x(c)]\!] = \big((),\ x = [\![c]\!],\ 1 - (x - [\![c]\!])^2 = 1,\ ()\big)$$
$$\Phi[\![\mathsf{await}\, x(v)]\!] = \big((),\ x = v,\ x^2 = 1,\ ()\big)$$
$$\Phi[\![\mathsf{emit}\, x(u)]\!] = \big((),\ x = [\![u]\!],\ x^2 = 1,\ ()\big)$$
$$\Phi[\![p \times \mathsf{next}\, f(u)]\!] = \Phi[\![p]\!] \wedge \big((),\ v^{f\prime} = [\![u]\!],\ (),\ ()\big)$$
$$\Phi[\![((\mathsf{rec}\, f(v^f).(\mathsf{next}\, f(v^f) + p))(u)]\!] = \Phi[\![p]\!] \wedge \big(v_0^f = [\![u]\!],\ (),\ (),\ ()\big)$$

$$\Phi[\![p \times q]\!] = \exists \tilde{m}\tilde{n}. \begin{pmatrix} I_p \wedge I_q \\ T_{p\restriction\tilde{x}} \wedge T_{q\restriction\tilde{x}} \\ \phi_p = 1 \\ C_p \wedge C_q \wedge (\phi_p - \phi_q = 0) \end{pmatrix} \qquad \Phi[\![(n)p]\!] = \exists n.(\Phi[\![p]\!])$$

$$\Phi[\![p+q]\!] = \exists \tilde{m}\tilde{n}. \begin{pmatrix} \phi_p.I_p \wedge \phi_q.I_q \\ \phi_p.T_p \wedge \phi_q.T_q \\ \phi_p + \phi_q = 1 \\ C_p \wedge C_q \wedge (\phi_p.\phi_q = 0) \end{pmatrix}$$

where $\Phi[\![p]\!] = \exists \tilde{m}.\ (I_p,\ T_p,\ \phi_p = 1,\ C_p)$
$\Phi[\![q]\!] = \exists \tilde{n}.\ (I_q,\ T_q,\ \phi_q = 1,\ C_q)$
$\tilde{x} = \mathrm{dv}(p) \cap \mathrm{dv}(q)$,
$\tilde{m} \cap \tilde{n} = \emptyset$

The function $\Phi[\![p]\!]$ is sound because any synchronous trace t of p satisfies Φ_p. We write $\vec{e}$ the representation of e as a vector of $\mathbb{Z}/3\mathbb{Z}^{|\mathrm{dv}(p)|}$ of indexes in $\mathrm{dv}(p)$, so that for all $x \in \mathrm{dv}(p)$, $\vec{e}_x = [\![e(x)]\!]$.

Theorem 1 (soundness) *If $t \in \mathcal{S}[\![p]\!]$ then, for all $i \geq 0$ and transition $p_i \xrightarrow{t_i} p_{i+1}$, the state $T_{p_i}(\vec{t_i}, \tilde{v})$ is alive for any $\tilde{v}$* s.t. *$I_{p_i}(\tilde{v}) = 0$.*

Checking determinism of a live process p and the robustness of a pair of deterministic processes p and q to desynchronization reduces to checking the invariants $\mathcal{D}_p$ and $\mathcal{R}_{p \times q}$. The criterion $\mathcal{D}_p$ for checking that a process p is deterministic consists of ensuring that the clock of every signal y in p is computable starting from the clock of the main signal x (i.e. $y^2 \leq x^2$). This allows for a unique flow of control to be iteratively reconstructed from the value of the boolean signals present at a given instant. The criterion $\mathcal{R}_{p \times q}$ for checking robustness to desynchronization consists of ensuring that, whenever a shared signal x is present in p (resp. q), then there is enough control expressed by the master clock ϕ_q (resp. ϕ_q) of q to ensure that it cannot be absent from q (resp. p), hence the assertion $(\phi_p.\phi_p^x).(\phi_q.(1 - \phi_q^x)) = 0$. We write ϕ_p^x for the clock of x in p (i.e. either ϕ s.t. C_p implies $x^2 = \phi$ or else x^2).

Theorem 2 (determinism and robustness) *A live process p is deterministic if there exists $x \in \mathrm{dv}(p)$* s.t. *C_p implies $x^2 = \phi_p$and* s.t. *T_p satisfies the invariant $(\mathcal{D}_p)$. The product of deterministic processes $p \times q$ is robust to desynchronization if $T_{p\times q}$ satisfies the invariant $(\mathcal{R}_{p\times q})$.*

$$(\mathcal{D}_p) : C_p \wedge \left(\bigwedge_{y \in \mathrm{dv}(p)} (y^2 \leq x^2) \right) \quad (\mathcal{R}_{p\times q}) : \bigwedge_{x \in \mathrm{dv}(p) \cap \mathrm{dv}(q)} \begin{pmatrix} ((\phi_p.\phi_p^x).(\phi_q.(1-\phi_q^x)) = 0) \\ \wedge((\phi_q.\phi_q^x).(\phi_p.(1-\phi_p^x)) = 0) \end{pmatrix}$$

Example 2 (determinism and robustness) *Let us reconsider the process* even *and its synchronous composition with the process* if *below. Using the criterion $\mathcal{D}$, one can checks that each process is deterministic: the signals y and z satisfy $y^2 \leq x^2$ and $z^2 \leq x^2$ (*i.e. *x is the main clock of* even *and* if*); the choice branches of* even *and* if *have exclusive clocks (*i.e. *$y^2.(1-(v+1)^2) = 0$ or $(1-(v-1)^2).(1-(v+1)^2) = 0$ for* even *and $y^2.z^2 = 0$ or $(1-(w-1)^2).(1-(w+1)^2) = 0$ for* if*). To check that the composition of* even *and* if *is robust to desynchronization, it is additionally necessary to check that the criterion $\mathcal{R}$ is meet. This amounts to showing that y is present in* even *iff it is present in* if, i.e. *that $(x^2.\phi_p^y).(x^2.(1-\phi_q^y)) = 0$ and that $(x^2.\phi_q^y).(x^2.(1-\phi_p^y)) = 0$ (*i.e. *that $(x^2).(1-(v-1)^2).(w-1)^2 = 0$ and $(x^2).(1-(w-1)^2).(v-1)^2 = 0$) is an invariant of $p \times q$. This requires fixed-point iteration using the method proposed in the previous section.*

$$\mathsf{rec\ even}(v).\mathsf{next\ even}(v) + \left(\mathsf{await}\ x() \times \begin{pmatrix} [v = ff] \times \mathsf{next\ even}(tt) \\ +[v = tt] \times \mathsf{next\ even}(ff) \times \mathsf{emit}\ y() \end{pmatrix}\right)(ff)$$

$$\mathsf{rec\ if}(v).\mathsf{next\ if}(w) + \left(\mathsf{await}\ x() \times \begin{pmatrix} [w = tt] \times \mathsf{next\ if}(ff) \times \mathsf{emit}\ y() \\ +[w = ff] \times \mathsf{next\ if}(tt) \times \mathsf{emit}\ z() \end{pmatrix}\right)(ff)$$

4. RELATED WORK

The first formal address of *desynchronization* can be found in [6], where precise relations between well-clocked synchronous functional programs and the subset of Kahn-networks are established, and shown to be amenable to bufferless evaluation. In [7], the author considers the distribution of synchronous automata on asynchronous networks using FIFO-buffered broadcast communications. In [4], a model for the distribution of synchronous programs on distributed architectures is introduced which uses low-level non-blocking one-place buffers. In [2], Ban extensive analysis of the links between synchrony and asynchrony is presented in the context of synchronous transition systems (STS) and the general notion of isochrony is introduced. In [12], an implementation of communicating reactive systems with multiple clocks using ESTEREL is presented. In [5], the theory of latency-insensitive designs is resented as a foundation of a new methodology to design very large digital systems by assembling blocks of existing *intellectual property* (IP).

5. CONCLUSION

We have formulated the problem of checking the robustness of synchronous processes to desynchronization in the algebraic and operational setting of a calculus of communicating synchronous processes: the signal calculus. We have shown that this problem reduces to model-checking invariants expressed as constraints in the algebraic framework of $\mathbb{Z}/3\mathbb{Z}$. Relevant applications for this method are found in the synchronous engineering of systems whose design requires attention on robustness to latency, to distribution, to threading: co-designed hardware-software architectures, (reconfigurable) embedded devices, multi-threaded reactive systems components on real-time virtual machines, distributed and reactive telecommunication applications.

References

[1] A. BENVENISTE, P. LE GUERNIC, C. JACQUEMOT. Synchronous programming with events and relations: the SIGNAL language and its semantics. In *Science of Computer Programming*, v. 16, 1991.

[2] A. BENVENISTE, B. CAILLAUD, P. LE GUERNIC. From synchrony to asynchrony. *International Conference on Concurrency Theory*. Lectures Notes in Computer Science. Springer, 1999.

[3] G. BERRY, G. GONTHIER. The ESTEREL synchronous programming language: design, semantics, implementation. In *Science of Computer Programming*, v. 19, 1992.

[4] G. BERRY, E. SENTOVICH. An Implementation of Constructive Synchronous Constructive Programs in Polis. *Formal Methods in Systems Design 17(2)*. Kluwer Academic Publisher, 2000.

[5] L.P. CARLONI, K.L. MCMILLAN, A.L. SANGIOVANNI-VINCENTELLI. Latency Insensitive Protocols. In *International conference on computer-aided verification*. Lectures Notes in Computer Science n. 1633. Springer, 1999.

[6] P. CASPI. Clocks in dataflow languages. In *Theoretical Computer Science*, v. 94. Elsevier, 1992.

[7] P. CASPI, A. GIRAULT, C. JARD. Distributed reactive systems. In *International Conference on Parallel and Distributed Computing Systems*. ISCA, 1994.

[8] N. HALBWACHS, P. CASPI, P. RAYMOND, D. PILAUD. The synchronous data-flow programming language LUSTRE. In *Proceedings of the* IEEE, v. 79(9). IEEE, 1991.

[9] H. MARCHAND, E. RUTTEN, M. LE BORGNE, M. SAMAAN. Formal Verification of SIGNAL programs: Application to a Power Transformer Station Controller. *Science of Computer Programming*, v. 41(1), pp. 85–104, 2001.

[10] R. MILNER. Calculi for synchrony and asynchrony. In *Theoretical Computer Science*. Elsevier, 1983.

[11] PNUELI, A., SHANKAR, N., SINGERMAN, E. Fair transition systems and their liveness proofs. In *International Symposium on Formal Techniques in Real-time and Fault-tolerant Systems*. Lecture Notes in Computer Science. Springer, 1998.

[12] RAJAN, B., SHYAMASUNDAR, R., K. Modeling distributed embedded systems in multiclock ESTEREL. In *Conference on Formal Description Techniques for Distributed Systems and Communication Protocols*. Kluwer, October 2000.