A Calculus of Cyber-Physical Systems

Ruggero Lanotte[1] and Massimo Merro[2]

[1] Dipartimento di Scienza e Alta Tecnologia, Università dell'Insubria, Como, Italy
ruggero.lanotte@uninsubria.it
[2] Dipartimento di Informatica, Università degli Studi di Verona, Italy
massimo.merro@univr.it

Abstract. We propose a hybrid process calculus for modelling and reasoning on *cyber-physical systems* (CPSs). The dynamics of the calculus is expressed in terms of a *labelled transition system* in the SOS style of Plotkin. This is used to define a *bisimulation-based* behavioural semantics which support compositional reasonings. Finally, we prove run-time properties and system equalities for a non-trivial case study.

Keywords: Process calculus, cyber-physical system, semantics.

1 Introduction

Cyber-Physical Systems (CPSs) are integrations of networking and distributed computing systems with physical processes, where feedback loops allow physical processes to affect computations and vice versa. For example, in real-time control systems, a hierarchy of *sensors*, *actuators* and *control processing components* are connected to control stations. Different kinds of CPSs include supervisory control and data acquisition (SCADA), programmable logic controllers (PLC) and distributed control systems.

The *physical plant* of a CPS is often represented in the literature by means of a *discrete-time state-space model*[1] consisting of two equations of the form

$$x_{k+1} = Ax_k + Bu_k + w_k$$
$$y_k = Cx_k + e_k$$

where $x_k \in \mathbb{R}^n$ is the current *(physical) state*, $u_k \in \mathbb{R}^m$ is the *input* (i.e., the control actions implemented through actuators) and $y_k \in \mathbb{R}^p$ is the *output* (i.e., the measurements from the sensors). The *uncertainty* $w_k \in \mathbb{R}^n$ and the *measurement error* $e_k \in \mathbb{R}^p$ represent perturbation and sensor noise, respectively, and A, B, and C are matrices modelling the dynamics of the physical system. The *next state* x_{k+1} depends on the current state x_k and the corresponding control actions u_k, at the sampling instant $k \in \mathbb{N}$. Note that, the state x_k cannot be directly observed: only its measurements y_k can be observed.

The physical plant is supported by a communication network through which the sensor measurements and actuator data are exchanged with the *controller(s)*, i.e., the *cyber* component, also called logics, of a CPS.

[1] See [17] for a tassonomy of the time-scale models used to represent CPSs.

The range of CPSs applications is rapidly increasing and already covers several domains: automotive, avionics, energy conservation, environmental monitoring, critical infrastructure control, etc. However, there is still a lack of research on the modelling and validation of CPSs through formal methodologies that might allow to model the interactions among the system components, and to verify the correctness of a CPS, as a whole, before its practical implementation. A straightforward utilisation of these techniques is for *model-checking*, i.e. to statically assess whether the current system deployment behaves as expected. However, they can also be an important aid for system planning, for instance to decide whether different deployments are behavioural equivalent.

In this paper, we propose a contribution in the area of *formal methods* for CPSs, by defining a *hybrid process calculus*, called `CCPS`, with a clearly-defined *behavioural semantics* for specifying and reasoning on CPSs. In `CCPS`, systems are represented as terms of the form $E \bowtie P$, where E denotes the *physical plant* (also called environment) of the system, containing information on state variables, actuators, sensors, evolution law, etc., while P represents the *cyber component of the system*, i.e., the *controller* that governs sensor reading and actuator writing, as well as channel-based communication with other cyber components. Thus, channels are used for logical interactions between cyber components, whereas sensors and actuators make possible the interaction between cyber and physical components. Despite this conceptual similarity, messages transmitted via channels are "consumed" upon reception, whereas actuators' states (think of a valve) remains unchanged until its controller modifies it.

`CCPS` is equipped with a *labelled transition semantics* (LTS) that satisfies some standard time properties such as: *time determinism*, *patience*, *maximal progress*, and *well-timedness*. Based on our LTS, we define a natural notion of *weak bisimilarity*. As a main result, we prove that our bisimilarity is a congruence and it is hence suitable for *compositional reasoning*. We are not aware of similar results in the context of CPSs. Finally, we provide a non-trivial *case study*, taken from an engineering application, and use it to illustrate our definitions and our semantic theory for CPSs. Here, we wish to remark that while we have kept the example simple, it is actually far from trivial and designed to show that various CPSs can be modelled in this style.

In this extended abstract, proofs are omitted; full details can be found in [10].

Outline. In Section 2, we give syntax and operational semantics of `CCPS`. In Section 3, we provide a bisimulation equivalence for `CCPS`, and prove its compositionality. In Sect. 4, we propose a case study, and prove for it run-time properties as well as system equalities. In Section 5, we discuss related and future work.

2 The Calculus

In this section, we introduce our *Calculus of Cyber-Physical Systems* `CCPS`. Let us start with some preliminary notations. We use $x, x_k \in \mathcal{X}$ for *state variables*; $c, d \in \mathcal{C}$ for *communication channels*, $a, a_k \in \mathcal{A}$ for *actuator devices*, and $s, s_k \in \mathcal{S}$ for *sensors devices*. *Actuator names* are metavariables for actuator devices like

valve, *light*, etc. Similarly, *sensor names* are metavariables for sensor devices, e.g., a sensor *thermometer* that measures, with a given precision, a state variable called *temperature*. *Values*, ranged over by $v, v' \in \mathcal{V}$, are built from basic values, such as Booleans, integers and real numbers; they also include names.

Given a generic set of names $\mathcal{N}$, we write $\mathbb{R}^{\mathcal{N}}$ to denote the set of functions assigning a real value to each name in $\mathcal{N}$. For $\xi \in \mathbb{R}^{\mathcal{N}}$, $n \in \mathcal{N}$ and $v \in \mathbb{R}$, we write $\xi[n \mapsto v]$ to denote the function $\psi \in \mathbb{R}^{\mathcal{N}}$ such that $\psi(m) = \xi(m)$, for any $m \neq n$, and $\psi(n) = v$. Given $\xi_1 \in \mathbb{R}^{\mathcal{N}_1}$ and $\xi_2 \in \mathbb{R}^{\mathcal{N}_2}$ such that $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$, we denote with $\xi_1 \uplus \xi_2$ the function in $\mathbb{R}^{\mathcal{N}_1 \cup \mathcal{N}_2}$ such that $(\xi_1 \uplus \xi_2)(x) = \xi_1(x)$, if $x \in \mathcal{N}_1$, and $(\xi_1 \uplus \xi_2)(x) = \xi_2(x)$, if $x \in \mathcal{N}_2$. Finally, given $\xi \in \mathbb{R}^{\mathcal{N}}$ and a set of names $\mathcal{M} \subseteq \mathcal{N}$, we write $\xi|_{\mathcal{M}}$ for the restriction of function ξ to the set $\mathcal{M}$.

In CCPS, a *cyber-physical system* consists of two components: a *physical environment* E that encloses all physical aspects of a system (state variables, physical devices, evolution law, etc) and a *cyber component*, represented as a concurrent process P that interacts with the physical devices (sensors and actuators) of the system, and can communicate, via channels, with other processes of the same CPS or with processes of other CPSs.

We write $E \bowtie P$ to denote the resulting CPS, and use M and N to range over CPSs. Let us formally define physical environments.

Definition 1 (Physical Environment). *Let $\hat{\mathcal{X}} \subseteq \mathcal{X}$ be a set of state variables, $\hat{\mathcal{A}} \subseteq \mathcal{A}$ be a set of actuators, and $\hat{\mathcal{S}} \subseteq \mathcal{S}$ be a set of sensors. A* physical environment *E is 7-tuple $\langle \xi_{\mathrm{x}}, \xi_{\mathrm{u}}, \xi_{\mathrm{w}}, evol, \xi_{\mathrm{e}}, meas, inv \rangle$, where:*

- $\xi_{\mathrm{x}} \in \mathbb{R}^{\hat{\mathcal{X}}}$ *is the* state function*,*
- $\xi_{\mathrm{u}} \in \mathbb{R}^{\hat{\mathcal{A}}}$ *is the* actuator function*,*
- $\xi_{\mathrm{w}} \in \mathbb{R}^{\hat{\mathcal{X}}}$ *is the* uncertainty function*,*
- $evol : \mathbb{R}^{\hat{\mathcal{X}}} \times \mathbb{R}^{\hat{\mathcal{A}}} \times \mathbb{R}^{\hat{\mathcal{X}}} \to 2^{\mathbb{R}^{\hat{\mathcal{X}}}}$ *is the* evolution map*,*
- $\xi_{\mathrm{e}} \in \mathbb{R}^{\hat{\mathcal{S}}}$ *is the* sensor-error function*,*
- $meas : \mathbb{R}^{\hat{\mathcal{X}}} \times \mathbb{R}^{\hat{\mathcal{S}}} \to 2^{\mathbb{R}^{\hat{\mathcal{S}}}}$ *is the* measurement map*,*
- $inv : \mathbb{R}^{\hat{\mathcal{X}}} \to \{\mathsf{true}, \mathsf{false}\}$ *is the* invariant function*.*

All the functions defining an environment are total functions*.*

The *state function* ξ_{x} returns the current value (in $\mathbb{R}$) associated to each state variable of the system. The *actuator function* ξ_{u} returns the current value associated to each actuator. The *uncertainty function* ξ_{w} returns the uncertainty associated to each state variable. Thus, given a state variable $x \in \hat{\mathcal{X}}$, $\xi_{\mathrm{w}}(x)$ returns the maximum distance between the real value of x and its representation in the model. Both the state function and the actuator function are supposed to change during the evolution of the system, whereas the uncertainty function is supposed to be constant.

Given a state function, an actuator function, and an uncertainty function, the *evolution map evol* returns the set of next *admissible state functions*. This function models the *evolution law* of the physical system, where changes made on actuators may reflect on state variables. Since we assume an uncertainty in our models, the evolution map does not return a single state function but a set of possible state functions. Note that we admit evolution maps that are not

necessarily linear. Note also that, although the uncertainty function is constant, it can be used in the evolution map in an arbitrary way, with different weights.

The *sensor-error function* ξ_{e} returns the maximum error associated to each sensor. Again due to the presence of the sensor-error function, the *measurement map meas* returns a set of admissible measurement functions rather than a single one.

Finally, the *invariant function inv* represents the conditions that the state variables must satisfy to allow for the evolution of the system. A CPS whose state variables don't satisfy the invariant is in *deadlock*.

Let us now formalise in CCPS the cyber components of CPSs. We extend the *timed process algebra TPL* [8] with two constructs: one to read values detected at sensors, and one to write values on actuators.

Definition 2 (Processes). Processes *are defined by the grammar:*

$$P, Q \ ::= \ \mathsf{nil} \mid \mathsf{idle}.P \mid P \parallel Q \mid \lfloor \pi.P \rfloor Q \mid [b]\{P\},\{Q\} \mid P\backslash c \mid X \mid \mathsf{rec}\, X.P.$$

We write nil for the *terminated process*. The process $\mathsf{idle}.P$ sleeps for one time unit and then continues as P. We write $P \parallel Q$ to denote the *parallel composition* of concurrent processes P and Q. The process $\lfloor \pi.P \rfloor Q$, with $\pi \in \{\mathsf{snd}\, c\langle v\rangle, \mathsf{rcv}\, c(x), \mathsf{read}\, s(x), \mathsf{write}\, a\langle v\rangle\}$, denotes *prefixing with timeout*. Thus, $\lfloor \mathsf{snd}\, c\langle v\rangle.P \rfloor Q$ sends the value v on channel c and, after that, it continues as P; otherwise, if no communication partner is available within one time unit, it evolves into Q. The process $\lfloor \mathsf{rcv}\, c(x).P \rfloor Q$ is the obvious counterpart for channel reception. The process $\lfloor \mathsf{read}\, s(x).P \rfloor Q$ reads the value v detected by the sensor s, whereas $\lfloor \mathsf{write}\, a\langle v\rangle.P \rfloor Q$ writes the value v on the actuator a. The process $P\backslash c$ is the channel restriction operator of CCS. The process $[b]\{P\},\{Q\}$ is the standard conditional, where b is a decidable guard. For simplicity, as in CCS, we identify $[b]\{P\},\{Q\}$ with P, if b evaluates to true, and $[b]\{P\},\{Q\}$ with Q, if b evaluates to false. In processes of the form $\mathsf{idle}.Q$ and $\lfloor \pi.P \rfloor Q$, the occurrence of Q is said to be *time-guarded*. The process $\mathsf{rec}\, X.P$ denotes *time-guarded recursion* as all occurrences of the process variable X may only occur time-guarded in P.

In the two constructs $\lfloor \mathsf{rcv}\, c(x).P \rfloor Q$ and $\lfloor \mathsf{read}\, s(x).P \rfloor Q$, the variable x is said to be *bound*. Similarly, the process variable X is bound in $\mathsf{rec}\, X.P$. This gives rise to the standard notions of *free/bound (process) variables* and *α-conversion*. We identify processes up to α-conversion (similarly, we identify CPSs up to renaming of state variables, sensor names, and actuator names). A term is *closed* if it does not contain free (process) variables, and we assume to always work with closed processes: the absence of free variables is preserved at run-time. As further notation, we write $T\{^{v}/_{x}\}$ for the substitution of the variable x with the value v in any expression T of our language. Similarly, $T\{^{P}/_{X}\}$ is the substitution of the process variable X with the process P in T.

The syntax of our CPSs is slightly too permissive as a process might use sensors and/or actuators which are not defined in the physical environment.

Definition 3 (Well-formedness). *Given a process P and an environment $E = \langle \xi_{\mathrm{x}}, \xi_{\mathrm{u}}, \xi_{\mathrm{w}}, evol, \xi_{\mathrm{e}}, meas, inv\rangle$, the CPS $E \bowtie P$ is* well-formed *if: (i) for any sensor s mentioned in P, the function ξ_{e} is defined in s; (ii) for any actuator a mentioned in P, the function ξ_{u} is defined in a.*

Hereafter, we will always work with well-formed networks.

$$\text{(Outp)}\ \frac{-}{\lfloor \mathsf{snd}\, c\langle v\rangle.P\rfloor Q \xrightarrow{\overline{c}v} P} \qquad \text{(Inpp)}\ \frac{-}{\lfloor \mathsf{rcv}\, c(x).P\rfloor Q \xrightarrow{cv} P\{^{v}/_{x}\}}$$

$$\text{(Write)}\ \frac{-}{\lfloor \mathsf{write}\, a\langle v\rangle.P\rfloor Q \xrightarrow{a!v} P} \qquad \text{(Read)}\ \frac{-}{\lfloor \mathsf{read}\, s(x).P\rfloor Q \xrightarrow{s?v} P\{^{v}/_{x}\}}$$

$$\text{(Com)}\ \frac{P \xrightarrow{\overline{c}v} P' \quad Q \xrightarrow{cv} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \qquad \text{(Par)}\ \frac{P \xrightarrow{\lambda} P' \quad \lambda \neq \mathsf{idle}}{P \parallel Q \xrightarrow{\lambda} P' \parallel Q}$$

$$\text{(ChnRes)}\ \frac{P \xrightarrow{\lambda} P' \quad \lambda \notin \{cv, \overline{c}v\}}{P\backslash c \xrightarrow{\lambda} P'\backslash c} \qquad \text{(Rec)}\ \frac{P\{^{\mathsf{rec}\, X.P}/_{X}\} \xrightarrow{\lambda} Q}{\mathsf{rec}\, X.P \xrightarrow{\lambda} Q}$$

$$\text{(TimeNil)}\ \frac{-}{\mathsf{nil} \xrightarrow{\mathsf{idle}} \mathsf{nil}} \qquad \text{(Delay)}\ \frac{-}{\mathsf{idle}.P \xrightarrow{\mathsf{idle}} P}$$

$$\text{(Timeout)}\ \frac{-}{\lfloor \pi.P\rfloor Q \xrightarrow{\mathsf{idle}} Q} \qquad \text{(TimePar)}\ \frac{P \xrightarrow{\mathsf{idle}} P' \quad Q \xrightarrow{\mathsf{idle}} Q' \quad P \parallel Q \xrightarrow{\tau}\!\!\!\!\!/}{P \parallel Q \xrightarrow{\mathsf{idle}} P' \parallel Q'}$$

Table 1. LTS for processes

Finally, we assume a number of *notational conventions*. We write $\pi.P$ instead of $\mathsf{rec}\, X.\lfloor \pi.P\rfloor X$, when X does not occur in P. We write $\mathsf{snd}\, c$ (resp. $\mathsf{rcv}\, c$) when channel c is used for pure synchronisation. For $k \geq 0$, we write $\mathsf{idle}^k.P$ as a shorthand for $\mathsf{idle}.\mathsf{idle}.\ldots.\mathsf{idle}.P$, where the prefix idle appears k consecutive times. Given $M = E \bowtie P$, we write $M \parallel Q$ for $E \bowtie (P \parallel Q)$, and $M\backslash c$ for $E \bowtie P\backslash c$.

2.1 Labelled Transition Semantics

In this section, we provide the dynamics of CCPS in terms of a *labelled transition system (LTS)* in the SOS style of Plotkin. In Definition 4, for convenience, we define some auxiliary operators on environments.

Definition 4. *Let* $E = \langle \xi_{\mathrm{x}}, \xi_{\mathrm{u}}, \xi_{\mathrm{w}}, evol, \xi_{\mathrm{e}}, meas, inv\rangle$ *be a physical environment.*
- $read\_sensor(E, s) = \{\xi(s) : \xi \in meas(\xi_{\mathrm{x}}, \xi_{\mathrm{e}})\}$
- $update\_act(E, a, v) = \langle \xi_{\mathrm{x}}, \xi_{\mathrm{u}}[a \mapsto v], \xi_{\mathrm{w}}, evol, \xi_{\mathrm{e}}, meas, inv\rangle$
- $next(E) = \bigcup_{\xi \in evol(\xi_{\mathrm{x}}, \xi_{\mathrm{u}}, \xi_{\mathrm{w}})} \{\langle \xi, \xi_{\mathrm{u}}, \xi_{\mathrm{w}}, evol, \xi_{\mathrm{e}}, meas, inv\rangle\}$
- $inv(E) = inv(\xi_{\mathrm{x}})$.

The operator $read\_sensor(E, s)$ returns the set of possible measurements detected by sensor s in the environment E; it returns a set of possible values rather than a single value due to the error $\xi_{\mathrm{e}}(s)$ of sensor s. $update\_act(E, a, v)$ returns the new environment in which the actuator function is updated in such a manner to associate the actuator a with the value v. $next(E)$ returns the set of the next admissible environments reachable from E, by an application of the evolution map. $inv(E)$ checks whether the state variables satisfy the invariant (here, with an abuse of notation, we overload the meaning of the function inv).

In Table 1, we provide standard transition rules for processes. Here, the meta-variable λ ranges over labels in the set $\{\mathsf{idle}, \tau, \overline{c}v, cv, a!v, s?v\}$. The symmetric counterparts of rules (Com) and (Par) are omitted.

$$\text{(Out)}\ \frac{P \xrightarrow{\overline{c}v} P' \quad inv(E)}{E \bowtie P \xrightarrow{\overline{c}v} E \bowtie P'} \qquad \text{(Inp)}\ \frac{P \xrightarrow{cv} P' \quad inv(E)}{E \bowtie P \xrightarrow{cv} E \bowtie P'}$$

$$\text{(SensRead)}\ \frac{P \xrightarrow{s?v} P' \quad inv(E) \quad v \in read\_sensor(E,s)}{E \bowtie P \xrightarrow{\tau} E \bowtie P'}$$

$$\text{(ActWrite)}\ \frac{P \xrightarrow{a!v} P' \quad inv(E) \quad E' = update\_act(E,a,v)}{E \bowtie P \xrightarrow{\tau} E' \bowtie P'}$$

$$\text{(Tau)}\ \frac{P \xrightarrow{\tau} P' \quad inv(E)}{E \bowtie P \xrightarrow{\tau} E \bowtie P'} \qquad \text{(Time)}\ \frac{P \xrightarrow{\mathsf{idle}} P' \quad E \bowtie P \xrightarrow{\overline{\tau}} \quad inv(E) \quad E' \in next(E)}{E \bowtie P \xrightarrow{\mathsf{idle}} E' \bowtie P'}$$

Table 2. LTS for CPSs

In Table 2, we lift the transition rules from processes to systems. All rules have a common premise $inv(E)$: a CPS can evolve only if the invariant is satisfied, otherwise it is deadlocked. Here, actions, ranged over by α, are in the set $\{\tau, \overline{c}v, cv, \mathsf{idle}\}$. These actions denote: non-observable activities (τ); observable logical activities, *i.e.*, channel transmission ($\overline{c}v$ and cv); the passage of time (idle). Rules (Out) and (Inp) model transmission and reception, with an external system, on a channel c. Rule (SensRead) models the reading of the current data detected at sensor s. Rule (ActWrite) models the writing of a value v on an actuator a. Rule (Tau) lifts non-observable actions from processes to systems. A similar lifting occurs in rule (Time) for timed actions, where $next(E)$ returns the set of possible environments for the next time slot. Thus, by an application of rule (Time) a CPS moves to the next physical state, in the next time slot.

Below, we report a few desirable time properties which hold in our calculus: (a) *time determinism*, (b) *maximal progress*, (c) *patience*, and (d) *well-timedness* (symbol $\equiv$ denotes standard *structural congruence* for timed processes [8, 13]).

Theorem 5 (Time Properties). *Let $M = E \bowtie P$ an arbitrary CPS.*

(a) If $M \xrightarrow{\mathsf{idle}} \hat{E} \bowtie Q$ and $M \xrightarrow{\mathsf{idle}} \tilde{E} \bowtie R$, then $\{\hat{E}, \tilde{E}\} \subseteq next(E)$ and $Q \equiv R$.
(b) If $M \xrightarrow{\tau} M'$ then there is no M'' such that $M \xrightarrow{\mathsf{idle}} M''$.
(c) If $M \xrightarrow{\mathsf{idle}} M'$ for no M' then either $next(E) = \emptyset$ or $inv(M) = \mathsf{false}$ or there is N such that $M \xrightarrow{\tau} N$.
(d) There is k such that whenever $M \xrightarrow{\alpha_1} .. \xrightarrow{\alpha_n} N$, with $\alpha_i \neq \mathsf{idle}$, then $n \leq k$.

Well-timedness [13, 4] ensures the absence of infinite instantaneous traces which would prevent the passage of time, and hence the physical evolution of a CPS. The proof of this property relies on time-guardedness of recursive processes.

3 Bisimulation

Once defined the labelled transition semantics, we are ready to define our bisimulation-based behavioural equality for CPSs. We recall that the only *observable activities* in CCPS are: time passing and channel communication. As a

consequence, the capability to observe physical events depends on the capability of the cyber components to recognise those events by acting on sensors and actuators, and then signalling them using (unrestricted) channels.

We adopt a standard notation for weak transitions: we write $\Rightarrow$ for the reflexive and transitive closure of τ-actions, namely $(\xrightarrow{\tau})^*$, whereas $\stackrel{\alpha}{\Longrightarrow}$ means $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$, and finally $\stackrel{\hat{\alpha}}{\Longrightarrow}$ denotes $\Rightarrow$ if $\alpha = \tau$ and $\stackrel{\alpha}{\Longrightarrow}$ otherwise.

Definition 6 (Bisimulation). *A binary symmetric relation $\mathcal{R}$ over CPSs is a bisimulation if $M\ \mathcal{R}\ N$ and $M \xrightarrow{\alpha} M'$ implies that there exists N' such that $N \stackrel{\hat{\alpha}}{\Longrightarrow} N'$ and $M'\ \mathcal{R}\ N'$. We say that M and N are bisimilar, written $M \approx N$, if $M\ \mathcal{R}\ N$ for some bisimulation $\mathcal{R}$.*

A main result of the paper is that our bisimilarity can be used to compare CPSs in a compositional manner. In particular, our bisimilarity is preserved by parallel composition of (non-interfering) CPSs, by parallel composition of (non-interfering) processes, and by channel restriction.

Two CPSs do not interfere with each other if they have a disjoint physical plant. Thus, let $E^i = \langle \xi^i_{\mathrm{x}}, \xi^i_{\mathrm{u}}, \xi^i_{\mathrm{w}}, \mathit{evol}^i, \xi^i_{\mathrm{e}}, \mathit{meas}^i, \mathit{inv}^i \rangle$ with sensors in $\hat{\mathcal{S}}_i$, actuators in $\hat{\mathcal{A}}_i$, and state variables in $\hat{\mathcal{X}}_i$, for $i \in \{1,2\}$. If $\hat{\mathcal{S}}_1 \cap \hat{\mathcal{S}}_2 = \emptyset$ and $\hat{\mathcal{A}}_1 \cap \hat{\mathcal{A}}_2 = \emptyset$ and $\hat{\mathcal{X}}_1 \cap \hat{\mathcal{X}}_2 = \emptyset$, then we define the *disjoint union* of the environments E_1 and E_2, written $E_1 \uplus E_2$, to be the environment $\langle \xi_{\mathrm{x}}, \xi_{\mathrm{u}}, \xi_{\mathrm{w}}, \mathit{evol}, \xi_{\mathrm{e}}, \mathit{meas}, \mathit{inv} \rangle$ such that: $\xi_{\mathrm{x}} = \xi^1_{\mathrm{x}} \uplus \xi^2_{\mathrm{x}}$, $\xi_{\mathrm{u}} = \xi^1_{\mathrm{u}} \uplus \xi^2_{\mathrm{u}}$, $\xi_{\mathrm{w}} = \xi^1_{\mathrm{w}} \uplus \xi^2_{\mathrm{w}}$, $\xi_{\mathrm{e}} = \xi^1_{\mathrm{e}} \uplus \xi^2_{\mathrm{e}}$, and

$$\begin{aligned} \mathit{evol}(\xi,\psi,\phi) &= \{\xi' = \xi_1 \uplus \xi_2 \,:\, \xi_i \in \mathit{evol}^i(\xi|_{\hat{\mathcal{X}}_i}, \psi|_{\hat{\mathcal{A}}_i}, \phi|_{\hat{\mathcal{X}}_i}),\ \text{for } i \in \{1,2\}\} \\ \mathit{meas}(\xi,\psi) &= \{\xi' = \xi_1 \uplus \xi_2 \,:\, \xi_i \in \mathit{meas}^i(\xi|_{\hat{\mathcal{X}}_i}, \psi|_{\hat{\mathcal{S}}_i}),\ \text{for } i \in \{1,2\}\} \\ \mathit{inv}(\xi) &= \mathit{inv}^1(\xi|_{\hat{\mathcal{X}}_1}) \wedge \mathit{inv}^2(\xi|_{\hat{\mathcal{X}}_2}). \end{aligned}$$

Definition 7. *Let $M_i = E_i \bowtie P_i$, for $i \in \{1,2\}$. We say that M_1 and M_2 do not interfere with each other if E_1 and E_2 have disjoint sets of state variables, sensors and actuators. In this case, we write $M_1 \uplus M_2$ to denote the CPS defined as $(E_1 \uplus E_2) \bowtie (P_1 \parallel P_2)$.*

A similar but simpler definition can be given for processes. Let $M = E \bowtie P$, a non-interfering process Q is a process which does not interfere with the plant E as it never accesses its sensors and/or actuators. Thus, in the system $M \parallel Q$ the process Q cannot interfere with the physical evolution of M. However, process Q can definitely affect the observable behaviour of the whole system by communicating on channels. Notice that, as we only consider well-formed CPSs (Definition 3), a non-interfering processes is basically a (pure) TPL process [8].

Definition 8. *A* non-interfering *process never acts on sensors and/or actuators.*

Now, everything is in place to prove the compositionality of our bisimilarity.

Theorem 9 (Congruence). *Let M and N be two CPSs.*

1. *$M \approx N$ implies $M \uplus O \approx N \uplus O$, for any non-interfering CPS O;*
2. *$M \approx N$ implies $M \parallel P \approx N \parallel P$, for any non-interfering process P;*
3. *$M \approx N$ implies $M \backslash c \approx M \backslash c$, for any channel c.*

As we will see in the next section, these compositional properties will be very useful when reasoning about complex systems.

4 Case Study

In this section, we model in CCPS an engine, called *Eng*, whose temperature is maintained within a specific range by means of a cooling system. The physical environment *Env* of the engine is constituted by: (i) a state variable *temp* containing the current temperature of the engine; (ii) an actuator *cool* to turn on/off the cooling system; (iii) a sensor $s_{\rm t}$ (such as a thermometer or a thermocouple) measuring the temperature of the engine; (iv) an uncertainty $\delta = 0.4$ associated to the only variable *temp*; (v) a simple evolution law that increases (resp., decreases) the value of *temp* of one degree per time unit if the cooling system is inactive (resp., active) — the evolution law is obviously affected by the uncertainty δ; (vi) an error $\epsilon = 0.1$ associated to the only sensor $s_{\rm t}$; (vii) a measurement map to get the values detected by sensor $s_{\rm t}$, up to its error ϵ; (viii) an invariant function saying that the system gets faulty when the temperature gets out of the range $[0, 30]$.

Formally, $Env = \langle \xi_{\rm x}, \xi_{\rm u}, \xi_{\rm w}, evol, \xi_{\rm e}, meas, inv \rangle$ with:

- $\xi_{\rm x} \in \mathbb{R}^{\{temp\}}$ and $\xi_{\rm x}(temp) = 0$;
- $\xi_{\rm u} \in \mathbb{R}^{\{cool\}}$ and $\xi_{\rm u}(cool) = \mathsf{off}$; for the sake of simplicity, we can assume $\xi_{\rm u}$ to be a mapping $\{cool\} \to \{\mathsf{on}, \mathsf{off}\}$ such that $\xi_{\rm u}(cool) = \mathsf{off}$ if $\xi_{\rm u}(cool) \geq 0$, and $\xi_{\rm u}(cool) = \mathsf{on}$ if $\xi_{\rm u}(cool) < 0$;
- $\xi_{\rm w} \in \mathbb{R}^{\{temp\}}$ and $\xi_{\rm w}(temp) = 0.4 = \delta$;
- $evol(\xi^i_{\rm x}, \xi^i_{\rm u}, \xi_{\rm w}) = \big\{ \xi \,:\, \xi(temp) = \xi^i_{\rm x}(temp) + heat(\xi^i_{\rm u}, cool) + \gamma \quad \wedge \quad \gamma \in [-\delta, +\delta] \big\}$, where $heat(\xi^i_{\rm u}, cool) = -1$ if $\xi^i_{\rm u}(cool) = \mathsf{on}$ (active cooling), and $heat(\xi^i_{\rm u}, cool) = +1$ if $\xi^i_{\rm u}(cool) = \mathsf{off}$ (inactive cooling);
- $\xi_{\rm e} \in \mathbb{R}^{\{s_{\rm t}\}}$ and $\xi_{\rm e}(s_{\rm t}) = 0.1 = \epsilon$;
- $meas(\xi^i_{\rm x}, \xi_{\rm e}) = \big\{\xi : \xi(s_{\rm t}) \in [\xi^i_{\rm x}(temp) - \epsilon \,,\, \xi^i_{\rm x}(temp) + \epsilon]\big\}$;
- $inv(\xi_{\rm x}) = \mathsf{true}$ if $0 \leq \xi_{\rm x}(temp) \leq 30$; $inv(\xi_{\rm x}) = \mathsf{false}$, otherwise.

The cyber component of *Eng* consists of a process *Ctrl* which models the controller activity. Intuitively, process *Ctrl* senses the temperature of the engine at each time interval. When the sensed temperature is above 10, the controller activates the coolant. The cooling activity is maintained for 5 consecutive time units. After that time, if the temperature does not drop below 10 then the controller transmits its *ID* on a specific channel for signalling a *warning*, it keeps cooling for another 5 time units, and then checks again the sensed temperature; otherwise, if the sensed temperature is not above the threshold 10, the controller turns off the cooling and moves to the next time interval. Formally,[2]

$$\begin{array}{rcl} Ctrl & = & \mathsf{rec}\, X.\mathsf{read}\, s_t(x).[x > 10]\{Cooling\}, \{\mathsf{idle}.X\} \\ Cooling & = & \mathsf{write}\, cool\langle \mathsf{on}\rangle.\mathsf{rec}\, Y.\mathsf{idle}^5.\mathsf{read}\, s_t(x). \\ & & [x > 10]\{\mathsf{snd}\, warning\langle \mathrm{ID}\rangle.Y\}, \{\mathsf{write}\, cool\langle \mathsf{off}\rangle.\mathsf{idle}.X\} \ . \end{array}$$

The whole engine is defined as: $Eng = Env \bowtie Ctrl$, where *Env* is the physical environment defined before.

Our operational semantics allows us to formally prove a number of *run-time properties* of our engine. For instance, the following proposition says that our engine never reaches a warning state and never deadlocks.

[2] We recall that $\pi.P$ is a shorthand for $\mathsf{rec}\, X.\lfloor \pi.P \rfloor X$, when X does not occur in P.

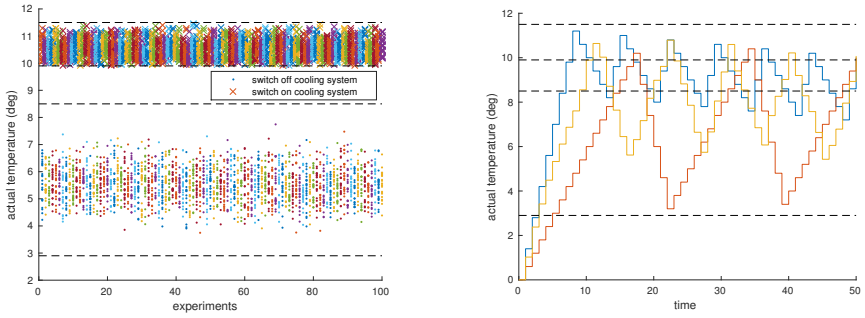

Fig. 1. Simulations in MATLAB of the engine *Eng*

Proposition 10. *Let Eng be the CPS defined before. If* $Eng \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Eng'$, *for some* Eng', *then* $\alpha_i \in \{\tau, \mathsf{idle}\}$, *for* $1 \leq i \leq n$, *and there is* Eng'' *such that* $Eng' \xrightarrow{\alpha} Eng''$, *for some* $\alpha_i \in \{\tau, \mathsf{idle}\}$.

Actually, we can be quite precise on the temperature reached by the engine before and after the cooling activity: in each of the 5 time slots of cooling, the temperature will drop of a value laying in the interval $[1-\delta, 1+\delta]$, where δ is the uncertainty of the model. Formally,

Proposition 11. *For any execution of Eng, we have:*

- *when Eng* turns on *the cooling, the value of the state variable temp ranges over* $(10 - \epsilon\,,\ 11 + \epsilon + \delta]$*;*
- *when Eng* turns off *the cooling, the value of the variable temp ranges over* $(10 - \epsilon - 5{*}(1{+}\delta)\,,\ 11 + \epsilon + \delta - 5{*}(1{-}\delta)]$.

In Figure 1, the left graphic collects a campaign of 100 simulations, lasting 250 time units each, showing that the value of the state variable *temp* when the cooling system is turned on (resp., off) lays in the interval $(9.9, 11.5]$ (resp., $(2.9, 8.5]$); these bounds are represented by the dashed horizontal lines. Since $\delta = 0.4$, these results are in line with those of Proposition 11. The right graphic shows three examples of possible evolutions of the state variable *temp*.

Now, the reader may wonder whether it is possible to design a variant of our engine which meets the same specifications with better performances. For instance, an engine consuming less coolant. Let us consider the variant: of the engine described before:

$$\overline{Eng} \;=\; \overline{Env} \bowtie Ctrl$$

where $\overline{Env}$ is the same as *Env* except for the evolution map, as we set $heat(\xi^i_{\mathrm{u}}, cool) = -0.8$ if $\xi^i_{\mathrm{u}}(cool) = \mathsf{on}$. This means that in $\overline{Eng}$ we reduce the power of the cooling system by 20%. In Figure 2, we report the results of our simulations over 10000 runs lasting 10000 time units each. From this graph, $\overline{Eng}$ saves in average more than 10% of coolant with respect to *Eng*. So, the new question is: are these two engines behavioural equivalent? Do they meet the same specifications?

Our bisimilarity provides us with a precise answer to these questions.

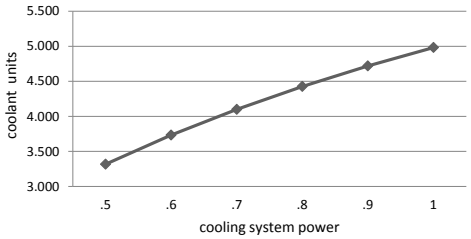

Fig. 2. Simulations in MATLAB of coolant consumption

Proposition 12. *The two variants of the engine are bisimilar: $Eng \approx \overline{Eng}$.*

At this point, one may wonder whether it is possible to improve the performances of our engine even further. For instance, by reducing the power of the cooling system by a further 10%, by setting $heat(\xi^i_{\mathrm{u}}, cool) = -0.7$ if $\xi^i_{\mathrm{u}}(cool) = \mathsf{on}$. We can formally prove that this is not the case.

Proposition 13. *Let $\widehat{Eng}$ be the same as Eng, except for the evolution map, in which $heat(\xi^i_{\mathrm{u}}, cool) = -0.7$ if $\xi^i_{\mathrm{u}}(cool) = \mathsf{on}$. Then, $Eng \not\approx \widehat{Eng}$.*

This is because the CPS $\widehat{Eng}$ may experience a warning, while *Eng* may not.

Finally, we show how we can use the compositionality of our behavioural semantics (Theorem 9) to deal with bigger CPSs. Suppose that *Eng* denotes the modelisation of an airplane engine. We could define in `CCPS` a very simple *airplane control system* that checks whether the left engine (Eng_{L}) and the right engine (Eng_{R}) are signalling warnings. The whole CPS is defined as follows:

$$Airplane \;=\; \big((Eng_{\mathrm{L}} \uplus Eng_{\mathrm{R}}) \parallel Check\big)\backslash warning$$

where $Eng_{\mathrm{L}} = Eng\{^{\mathrm{L}}/_{\mathrm{ID}}\}\{^{temp\_l}/_{temp}\}\{^{cool\_l}/_{cool}\}\{^{s_{\mathrm{t}}\_l}/_{s_{\mathrm{t}}}\}$, and, similarly, $Eng_{\mathrm{R}} = Eng\{^{\mathrm{R}}/_{\mathrm{ID}}\}\{^{temp\_r}/_{temp}\}\{^{cool\_r}/_{cool}\}\{^{s_{\mathrm{t}}\_r}/_{s_{\mathrm{t}}}\}$, and process *Check* is defined as:

$$\begin{aligned}
Check &= \mathsf{rec}\, X.\lfloor \mathsf{rcv}\; warning(x).[x = \mathrm{L}]\{Check^{\mathrm{L}}_1\}, \{Check^{\mathrm{R}}_1\}\rfloor X\\
Check^{id}_i &= \lfloor \mathsf{rcv}\; warning(y).[y \neq id]\{\mathsf{snd}\; alarm.\mathsf{idle}.X\}, \{\mathsf{idle}.Check^{id}_{i+1}\}\rfloor Check^{id}_{i+1}\\
Check^{id}_5 &= \lfloor \mathsf{rcv}\; warning(z).[z \neq id]\{\mathsf{snd}\; alarm.\mathsf{idle}.X\}, \{\mathsf{snd}\; failure\langle id\rangle.\mathsf{idle}.X\}\rfloor\\
&\quad\; \mathsf{snd}\; failure\langle id\rangle.X
\end{aligned}$$

for $1 \leq i \leq 5$. Intuitively, if one of the two engines is in a warning state then the process $Check^{id}_i$, for $id \in \{\mathrm{L}, \mathrm{R}\}$, checks whether also the second engine moves into a warning state, in the following 5 time intervals (i.e. during the cooling cycle). If both engines gets in a warning state then an *alarm* is sent, otherwise, if only one engine is facing a warning then the airplane control system yields a *failure* signalling which engine is not working properly.

So, since we know that $Eng \approx \overline{Eng}$, the final question becomes the following: can we safely equip our airplane with the more performant engines, $\overline{Eng_{\mathrm{L}}}$ and $\overline{Eng_{\mathrm{R}}}$, in which $heat(\xi^i_{\mathrm{u}}, cool) = -0.8$ if $\xi^i_{\mathrm{u}}(cool) = \mathsf{on}$, without affecting the whole observable behaviour of the airplane? The answer is "yes", and this result can be formally proved by applying Proposition 12 and Theorem 9.

Proposition 14. *Let* $\overline{Airplane} = \big((\overline{Eng_{\mathrm{L}}} \uplus \overline{Eng_{\mathrm{R}}}) \parallel Check\big)\backslash warning$. *Then,* $Airplane \approx \overline{Airplane}$.

5 Related and Future Work

A number of approaches have been proposed for modelling CPSs using formal methods. For instance, *hybrid automata* [1] combine finite state transition systems with discrete variables (whose values capture the state of the modelled discrete or cyber components) and continuous variables (whose values capture the state of the modelled continuous or physical components).

Hybrid process algebras [5, 2, 15, 7] are a powerful tool for reasoning about physical systems, and provide techniques for analysing and verifying protocols for hybrid automata. `CCPS` shares some similarities with the ϕ-calculus [15], a hybrid extension of the π-calculus. In the ϕ-calculus, a hybrid system is represented as a pair (E, P), where E is the environment and P is the process interacting with the environment. Unlike `CCPS`, in ϕ-calculus, given a system (E, P) the process P can dynamically change both the evolution law and the invariant of the system. However, the ϕ-calculus does not have a representation of physical devices and measurement law. Concerning behavioural semantics, the ϕ-calculus is equipped with a weak bisimilarity between systems that is not compositional.

In the HYPE process algebra [7], the continuous part of the system is represented by appropriate variables whose changes are determined by active influences (i.e., commands on actuators). The authors defines a *strong* bisimulation that extends the *ic-bisimulation* of [2]. Unlike ic-bisimulation, the bisimulation in HYPE is preserved by a notion of parallel composition that is slightly more permissive than ours. However, bisimilar systems in HYPE must always have the same influence. Thus, in HYPE we cannot compare CPSs sending different commands on actuators, as we do in Proposition 12.

Vigo et al. [16] proposed a calculus for wireless-based cyber-physical systems endowed with a theory to study cryptographic primitives, together with explicit notions of communication failure and unwanted communication. The calculus does not provide any notion of behavioural equivalence.

Lanese et al. [9] proposed an untimed calculus of IoT devices. The calculus does not contain any representation of the physical environment, and the bisimilarity is not preserved by parallel composition (compositionality is recovered by significantly strengthening the discriminating power of the bisimilarity).

Lanotte and Merro [11] extended and generalised the work of [9] in a timed setting by providing a bisimulation-based semantic theory that is suitable for compositional reasoning. As in [9], the physical environment is not represented.

Bodei et al. [3] proposed an untimed process calculus for IoT systems supporting a control flow analysis to track how data spread from sensors to the logics of the network, and how physical data are manipulated. Sensors and actuators are modelled as value-passing CCS channels. No behavioural equivalence is defined.

As regards future works, we believe that our paper can lay and streamline *theoretical foundations* for the development of formal and automated tools to verify CPSs before their practical implementation. To that end, we will consider

applying, possibly after proper enhancements, existing tools and frameworks for automated verification, such as Maude [14] and SMC UPPAAL [6]. Finally, in [12], we developed an extended version of `CCPS` to provide a formal study of a variety of *cyber-physical attacks* targeting physical devices. Again, the final goal is to develop formal and automated tools to analyse security properties of CPSs.

Acknowledgements

We thank Riccardo Muradore for providing us with simulations in MATLAB of our case study. We thank the anonymous reviewers for valuable comments.

References

1. Alur, R., Courcoubetis, C., Henzinger, T., Ho, P.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Hybrid Systems. LNCS, vol. 736, pp. 209–229. Springer (1992)
2. Bergstra, J.A., Middleburg, C.A.: Process algebra for hybrid systems. TCS 335(2-3), 215–280 (2005)
3. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: Where do your iot ingredients come from? In: Lluch-Lafuente, A., Proença, J. (eds.) COORDINATION 2016. LNCS, vol. 9686, pp. 35–50. Springer (2016)
4. Cerone, A., Hennessy, M., Merro, M.: Modelling mac-layer communications in wireless systems. Logical Methods in Computer Science 11(1:18) (2015)
5. Cuijpers, P., Reniers, M.: Hybrid process algebra. JLAP 62(2), 191–245 (2005)
6. David, D., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer (2011)
7. Galpin, V., Bortolussi, L., Hillston, J.: HYPE: Hybrid modelling by composition of flows. Formal Asp. Comput. 25(4), 503–541 (2013)
8. Hennessy, M., Regan, T.: A process algebra for timed systems. I&C 117(2), 221–239 (1995)
9. Lanese, I., Bedogni, L., Di Felice, M.: Internet of things: a process calculus approach. In: Shin, S., Maldonado, J. (eds.) ACM SAC 2013. pp. 1339–1346. ACM (2013)
10. Lanotte, R., Merro, M.: A calculus of cyber-physical systems. CoRR abs/1612.00484 (2016)
11. Lanotte, R., Merro, M.: A semantic theory of the internet of things. In: Lluch-Lafuente, A., Proença, J. (eds.) COORDINATION 2016. LNCS, vol. 9686, pp. 157–174. Springer (2016)
12. Lanotte, R., Merro, M., Muradore, R., Viganò, L.: A formal approach to cyber-physical attacks. CoRR abs/1611.01377 (2016)
13. Merro, M., Ballardin, F., Sibilio, E.: A timed calculus for wireless systems. TCS 412(47), 6585–6611 (2011)
14. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of real-time maude. Higher-Order and Symbolic Computation 20(1-2), 161–196 (2007)
15. Rounds, W.C., Song, H.: The ϕ-calculus: A language for distributed control of reconfigurable embedded systems. In: HSCC 2003. pp. 435–449. Springer (2003)
16. Vigo, R., Nielson, F., Riis Nielson, H.: Broadcast, denial-of-service, and secure communication. In: Johnsen, E.B., Petre, L. (eds.) IFM 213. LNCS, vol. 7940, pp. 412–427. Springer (2013)
17. Zacchia Lun, Y., D'Innocenzo, A., Malavolta, I., Di Benedetto, M.D.: Cyber-physical systems security: a systematic mapping Study. CoRR abs/1605.09641 (2016)