Towards a Distributive Interpreter in Coq for Reactive Systems

Benjamin Lion[1], David Nowak[2], Jean-Pierre Talpin[1]

[1] Inria, Rennes, France
[2] CNRS, Lille, France

Abstract. Principled "separation of concerns" traditionally advocates to orchestrate cyber-physical system (CPS) design under the three distinct concerns of its cybernetic (time), physics (evolution) and system (control program). That way, a CPS is safe once its program is checked functionally deterministic, its timing schedulable, and its physics controllable. And yet, time and physics are the root side effects that could cause the whole CPS not to be (deterministic). We hence advocate to lift the domain of such control program to allow for the representation of such effects. Our goal is to design fully verified (safety-critical) programs, and not just programs that are deterministic assuming a correct scheduler and a correct physical model. To this end, we investigate the use of a comonad combined with a monad to formally represent trustable control programs in the constructive proof assistant Coq, as a mean to support the end-to-end verification of their intended behavior until embedded in a CPS.
Keywords: comonad, monad, monitoring, CPS, Coq

1 Introduction

Most approaches to study cyber-physical systems are based on real arithmetic and abstract the discrete and the approximate nature of program operations. The benefit of using real arithmetics for reasoning about continuous systems is well known, one of which is the decidability of the first order theory. However, in practice, the application has access to only a subset of the model's information: the measures are approximated, the time of the measure may be controlled by the scheduler, etc. In order to reason about runtime cyber-physical effects of a program, a semantic model must capture the streaming nature of program actuation and (approximate) sensor reading, and the effects of running the program in its immediate (cybernetic) and neighbouring (physical) environment.

Recent research on semantics for reactive programs has led to notions of computations worth exploring for capturing cyber-physical effects. The use of monads for modeling effects [8] of programs at a semantic level enables powerful static analysis (using theorem provers) on effectful runtime behaviors. The recent exploration of comonads combined with a monad for dataflow programming [10] showed powerful result to give a compositional semantics of effectful dataflow programs. A categorical semantic for programs with cyber-physical would therefore mix both a comonad for the streaming of values, and a monad for the effect of an execution. We motivate the need for such semantics with the following example.

To ground our thoughts, consider a cyber-physical system consisting of a controller C that monitors a variable x with two actions: $\mathbf{read}(x)$, to acquire a value from sensor x, and $\mathbf{a}(n)$, to trigger actuator n. We assume that the profile of values at sensor x is given by a piece-wise linear and right-continuous function of time $x(t)$.

t	C	P
0.5	$v_x := \mathbf{read}(x)$	$x(t) \in \mathbb{R}$
0.7	$\mathbf{if}\ x > c\ \mathbf{then}\ \mathbf{a}(x)\ \mathbf{else\ skip\ fi}$	$\dot{x}(t^+) = -\dot{x}(t^-)$

Table 1. Interaction between discrete commands and continuous environment.

The C column shows two instructions, run in sequence, from a controller. The column P contains the events that are observed on the plant controlled by C. For instance, the instruction $\mathbf{read}(x)$ has an effect on P, namely, it extracts an approximation of the value $x(t)$, which is stored in v_x. The second instruction in C is a branching condition that, if x is greater than the value c, actuates P with $\mathbf{a}(x)$ and therefore changes the internal dynamic of x (e.g., change the derivative of x). Note that, for each instruction, the column t labels the occurrence with a time value. We discuss two constraints inherent to the description in Table 1 that are essential for monitoring of cyber-physical systems:

- The exact (real) time duration between two actions is unpredictable. The resulting interaction between the controller C and the physical plant P is therefore inherently non-deterministic as the controller cannot accurately control the time of its action. The semantics of a program C with effectful actions such as $\mathbf{read}(x)$ and $\mathbf{a}(x)$ must take the side-effects of such non-determinism into account.
- The semantics of $x > c$ may depend on the current value v_x, but also on values previously read. As a result, the branching condition is ambiguous: is the condition on the value of x sensitive to the time at which the value is read? What happens if the precision in the condition is higher than the precision used to store the value in the variable v_x? Can we infer a larger precision of x with additional readings?

Related Work In [10], the authors use a combination of monads and comonads to give a semantics for Lustre, a dataflow language. In this semantics, monads capture effects (clock, value at a time), and comonad captures dataflow (list of past values).

The language Lola [1] has been introduced for monitoring stream based applications. In Lola, streams are synchronized via a global clock. In [2], the authors extend Lola with real time capabilities, namely a variable-rate input stream that labels input events with real time, and a sliding window primitive that aggregates data over a time window.

Several formalisms exist to specify cyber-physical systems, such as Platzer with its formal framework for analysis of cyber-physical systems [9], Naijun Zhang with the formalisation of hybrid systems [7], and Edward Lee with an actor-based implementation of hybrid systems [5]. Moreover, the quantum physicist Nicolas Gisin also witnesses that using computational model for explaining physics is a region rich of theoretical and practical questions, fertile for discoveries [3].

In [4], the authors propose a hybrid monad as a semantic from imperative reactive programs, and especially give a semantics for (possibly diverging) iteration. However, such semantics is not operational, and requires calculating total execution time of a program.

Contributions We give a categorical semantics for a language for controllers that exposes the cyber-physical effects within a comonad and a monad. This new semantic approach for cyber-physical systems enables compositional specifications (discrete controller composed with a context), and provides a formal ground for powerful certification methods in a proof assistant. Moreover, the combination of a comonad and a monad in a semantics reflect runtime concerns, and is therefore suitable for analysis of runtime behaviors.

Outline In Section 2, we introduce preliminaries about biKleisli categories. In Section 3, we present an imperative language for cyber-physical controller, whose semantics is a morphism in a biKleisli category. In Section 4, we describe the implementation in Coq of a distributive law, required for the semantics to be compositional.

2 Preliminaries on the BiKleisli Category

In this section, we assume that the reader is familiar with basic categorical notions such as categories, functors and natural transformations. A *comonad* on a category $\mathbb{C}$ consists of an endofunctor $W : \mathbb{C} \to \mathbb{C}$ with two natural transformations: the *counit* $\varepsilon : W \to 1_{\mathbb{C}}$ (where $1_{\mathbb{C}}$ is the identity functor on $\mathbb{C}$) and the *comultiplication* $\delta : W \to W^2$ (where W^2 is the composite functor $W \circ W$). These are required to make the two diagrams on the right (a.k.a., *coherence conditions*) commute (where the symbol $\cdot$ denotes the horizontal composition of natural transformations). In computer science, comonads are used to model context-dependent computations.

$$\begin{tikzcd}
W \arrow[r, "\delta"] \arrow[d, "\delta"'] \arrow[dr, equal] & W^2 \arrow[d, "\varepsilon \cdot 1_W"] \\
W^2 \arrow[r, "1_W \cdot \varepsilon"'] & W
\end{tikzcd}$$

$$\begin{tikzcd}
W \arrow[r, "\delta"] \arrow[d, "\delta"'] & W^2 \arrow[d, "\delta \cdot 1_W"] \\
W^2 \arrow[r, "1_W \cdot \delta"'] & W^3
\end{tikzcd}$$

In this paper, we take $\mathbb{C}$ to be the category whose objects are Coq types and morphisms are functions. We will consider the comonad that maps a type X into the type of non-empty lists over X. Concretely, we will use those non-empty lists to keep a current value in the head of the list and previous ones in the tail.

A *monad* is the dual of a comonad: roughly speaking, it has the same components (and coherence conditions) except that the arrows are reversed and are called the *unit* and the *multiplication*. It is used in computer science to model effectful computations. We will consider two monads: the maybe monad that maps a type X into the sum type $1 + X$ thus allowing to represent the failure of a computation; and the state monad that maps a type X into the function type $S \to (X \times S)$ thus allowing to represent stateful computation (where S is the type of the state).

Given a comonad (W, ε, δ) and a monad (M, η, μ), a *distributive law* of the comonad over the monad consists of a natural transformation $\xi : WM \to MW$ such that the following diagrams commute:

$$\begin{tikzcd}
 & W \arrow[dl, "1_W \cdot \eta"'] \arrow[dr, "\eta \cdot 1_W"] & \\
WM \arrow[rr, "\xi"'] & & MW
\end{tikzcd}
\qquad
\begin{tikzcd}
WM^2 \arrow[r, "\xi \cdot 1_M"] \arrow[d, "1_W \cdot \mu"'] & MWM \arrow[r, "1_M \cdot \xi"] & M^2W \arrow[d, "\mu \cdot 1_W"] \\
WM \arrow[rr, "\xi"'] & & MW
\end{tikzcd}$$

$$\begin{tikzcd}
 & M & \\
WM \arrow[ur, "\varepsilon \cdot 1_M"] \arrow[rr, "\xi"'] & & MW \arrow[ul, "1_M \cdot \varepsilon"']
\end{tikzcd}
\qquad
\begin{tikzcd}
W^2M \arrow[r, "1_W \cdot \xi"] & WMW \arrow[r, "\xi \cdot 1_W"] & MW^2 \\
WM \arrow[u, "\delta \cdot 1_M"] \arrow[rr, "\xi"'] & & MW \arrow[u, "1_M \cdot \delta"']
\end{tikzcd}$$

Given a category $\mathbb{C}$, a comonad W and a monad M on $\mathbb{C}$, and a distributivity law of W over M, one can define the *biKleisli category* $\mathbb{C}_{W,M}$ whose objects are those of $\mathbb{C}$ and sets of morphisms $\mathbb{C}_{W,M}(X, Y)$ are $\mathbb{C}(WX, MY)$. In a context-dependent and effectful language, a program that inputs a value of type X and outputs a value of type Y is modeled as a function $p : WX \to MY$. For example, in [10], a distributive law of the non-empty list comonad over the maybe monad was used to give a categorical semantics to a clocked dataflow language: the non-empty list comonad allows to model the access to previous values of a flow, and the maybe monad allows to model the absence or presence of a value in a flow. In this paper we investigate the use of a distributive law of the non-empty list comonad over a monad that capture the effect of the program, and show how it can be used to deal with the monitoring of cyber-physical systems.

3 Monitoring of Cyber-Physical systems

The necessity for monitoring cyber-physical systems emerges from the streaming nature of cyber-physical interaction: the effect of a controller cannot be anticipated as it depends on the (real) time of the action, and the (real) state of the physics, which cannot be predicted by a discrete controller.

Each letter C, T, P, and A, in Table 2 corresponds to a (cyber-physical) process that transforms an input stream to an output stream. We give the signature for each process in Section 3.1 The table gives a symbolic example of what each

	C	T	P	A
$(\lfloor x_1 \rfloor, \lfloor t_1 \rfloor)$	$\mathbf{read}(x)$	$(\mathbf{read}(x), t_1 \in \mathbb{R})$	$(x_1 \in \mathbb{R}, t_1)$	$(\lfloor x_1 \rfloor, \lfloor t_1 \rfloor)$
$(\lfloor x_2 \rfloor, \lfloor t_2 \rfloor)$	$\mathbf{read}(x)$	$(\mathbf{read}(x), t_2 \in \mathbb{R})$	$(x_2 \in \mathbb{R}, t_2)$	$(\lfloor x_2 \rfloor, \lfloor t_2 \rfloor)$
$\lfloor t_3 \rfloor$	$\mathbf{a}(n)$	$(\mathbf{a}(n), t_3 \in \mathbb{R})$	t_3	$\lfloor t_3 \rfloor$
...	...	...	...	...

Table 2. Interaction between discrete commands and continuous environment.

process does, and how processes compose. For each process, the left part of the line is its input stream, and the right part is its output. As expected, composition unifies the output of one process with the input of another process. For instance, the controller C takes some approximate reading as inputs, and sends commands to a timer T, that outputs the command labeled with a real time value. The plan P receives the command with a real time, and interprets such command by, for instance, valuating the sensor value x. Finally, the real value are approximated by the process A to fit within the precision of the inputs of the controller. Assuming that $\gg$ stands for the composition, the overall system behavior is the set of streams x such that:

$$(A \gg P \gg T \gg C)\ x = x$$

Properties on the robustness of a controller C can therefore be studied by altering the specification of A or T and analyse invariant on the system behavior. The categorical framework introduced in Section 2 makes possible to define functionally the streaming nature and the effect of individual process, and gives conditions for having an operator of composition between processes (as composition of morphisms in the biKleisli category).

3.1 Categorical semantics

In fact, processes from Table 2 are morphisms of the form $WX \to MY$, where W is the non-empty list comonad, and M is the monad that captures the effect of the program.

Example 1 (Controller). The effects of a controller are captured by morphisms of the form $LV\ I \to LV\ O$), where LVI is the set of input sequences (i.e., sensor values), and LVO is the set of output sequences (i.e., sensor readings, and actuations). A stateful and operational description of a controller is a morphism between the non-empty list comonad LV and the state monad $St\ O$ such that $LV\ I \to St\ O$. We give in Section 3.2 a language for which the semantics of each instruction is a morphism of such form.

Example 2 (Environment). Environment E with actionable and sensing A are evaluated as morphisms of the form $LV\ A \to St\ I$ with I the valuation of sensing variables. In other words, an environment generates inputs I given actions and a state S.

Composition of morphisms is defined as arrows in a biKleisli category, where the distributive law threads the execution over the list of state monads.

Composition The definition of a composition beteeen P and its environment E would be $P \gg E$, and requires a distributive law between the non-empty list comonad and the state monad. As explain in Section 4, this question is left as future work.

3.2 Interpreter for CPS controller

In this subsection, we use the categorical framework to give a semantics for a language with cyber-physical effects. We first introduce the term language and its semantics, and then show how it composes with physical processes (Timer, Plant, and Approximate).
Terms:

$$t := f(t_1, ..., t_n) \mid v_s \mid x$$

with $v_s \in \mathcal{S}$ a sensor variable, disjoint from state variables $x \in \mathcal{V}$.
Guards:

$$b := t_1 = t_2 \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b \mid \textbf{false} \mid t_1 < t_2$$

Programs:

$$\begin{aligned} S \;:=\; & \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \mid S_1; S_2 \mid \textbf{while } b \textbf{ do } S \textbf{ od} \mid \\ & \textbf{a}(n) \mid v_s := \textbf{read}(s) \mid x := t \mid \textbf{skip} \end{aligned}$$

with $n \in \mathcal{A}$ an actuator name.

Let Act be the set of actions. We fix $\textbf{check}(b)$, $\textbf{skip}$, $\textbf{read}(s)$, $\textbf{a}(n) \in Act$ for all guards b, sensors s, and actuators n. We fix $I : \mathcal{S} \to D_{\mathcal{S}}$ to be the set of valuations of sensor variables, and $St : \mathcal{V} \to D_{St}$ to be the set of valuations of state variables.

The state monad $State\ A = St \to (A \times St)$ and the comonad $LV\ A$ of non-empty lists are respectively the co-domain and the domain of the process morphisms. Hence, we give a semantics for a program S as a morphism $[\![S]\!] : LV\ I \to (State\ Act)$. We define $[\![\cdot]\!]$ inductively on the structure of the program S, for which we give some rules:

$$\begin{aligned} [\![v_s := \textbf{read}(s)]\!](l, q) &= (\textbf{read}(s), q[v_s] \mapsto val(v_s, l)) \\ [\![\textbf{a}(n)]\!](l, q) &= (\textbf{a}(s), q) \\ [\![x := t]\!](l, q) &= (\textbf{skip}, q[x] \mapsto val(t, l, q)) \\ [\![\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}]\!](l, q) &= [\![S_1]\!](l, q) \ \textit{if } val(b, l, q) = \textbf{true} \\ [\![\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}]\!](l, q) &= [\![S_2]\!](l, q) \ \textit{if } val(b, l, q) = \textbf{false} \\ [\![S_1; S_2]\!](l, q) &= [\![S_2]\!](l, q') \ \textit{with } [\![S_1]\!](l, q) = (o, q') \\ [\![\textbf{while } b \textbf{ do } S \textbf{ od}]\!](l, q) &= [\![\textbf{if } b \textbf{ then } S; \textbf{while } b \textbf{ do } S \textbf{ od else skip fi}]\!](l, q) \end{aligned}$$

where *val* is a context dependent valuation function. For instance, a context dependent evaluation of $x < c$ would use the past read values of x and an inference mechanism to decide if, at the expected present time, x is lower than c. The dataflow description of C in table 2 is given by relating input lists in $LV\ I$ to output actions in $LV\ O$ from denotation $[\![C]\!]$.

4 Implementation

The categorical semantics gives a functional yet effectful and context-dependent description of programs with cyber-physical effects. Once the semantics is formalized in a proof assistant, it becomes possible to formally prove invariant on reactive behaviors. For instance, the prove may ensure that the compiled binary, running on the targeted architecture, has safe cyber-physical effects. The existence of toolchains using CompCert [6] demonstrates the feasibility of such approach.

Distributive law with option monad A general implementation of monads and comonads is in Coq [3], with the axioms that the distributive law must verify (as detailed in Section 2). The distributive law is instantiated for the case of the option monad `Maybe` and the non-empty list comonad `LV`. As a result, processes of the form `f: LV A` $\rightarrow$ `Maybe B` and `g: LV B` $\rightarrow$ `Maybe C` compose to form a process `f >> g : LV A` $\rightarrow$ `Maybe C`. The composition is intuitively collecting, in order, all outputs from `f` for any prefix of the input list in `LV A` and filters absent values from the list to return elements of `LV B`, or propagates the absence of value if the resulting list is empty. There is one special case, however, that if the head of the input list for `f` if the head of the list is an absent value, the output list is therefore also absent.

The distributive law has the signature `LV (Maybe A)` $\rightarrow$ `Maybe (LV A)` and the proof for the laws have been mechanically verified in Coq. The resulting biKleisli category and morphism composition follows from the categorical framework, as shown in 2.

5 Conclusions and future work

While the result of distributive law of the non-empty list comonad over the option monad are encouraging, a reactive language for cyber-physical systems has a stateful execution as shown in Section 3.2. The distributive law for this case would enable composition of morphisms of the form `f: LV A` $\rightarrow$ `State B` and `g: LV B` $\rightarrow$ `State C` to `f >> g: LV A` $\rightarrow$ `State C`. One way to do so is to compute the list of state monads obtained from `f` with each prefix of the input list, thread the state over the list, and collect the outputs in a list for `g`. Proving that such composition satisfies the axioms of a distributive law (or something similar) is left as future work.

[3] The Coq code is reachable here.

References

1. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: runtime monitoring of synchronous systems. In: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA. pp. 166–174. IEEE Computer Society (2005). `https://doi.org/10.1109/TIME.2005.26`, `https://doi.org/10.1109/TIME.2005.26`
2. Faymonville, P., Finkbeiner, B., Schledjewski, M., Schwenger, M., Stenger, M., Tentrup, L., Torfah, H.: Streamlab: Stream-based monitoring of cyber-physical systems. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 421–431. Springer (2019). `https://doi.org/10.1007/978-3-030-25540-4\_24`, `https://doi.org/10.1007/978-3-030-25540-4_24`
3. Gisin, N.: Indeterminism in physics, classical chaos and bohmian mechanics: Are real numbers really real? Erkenntnis **86**(6), 1469–1481 (oct 2019). `https://doi.org/10.1007/s10670-019-00165-8`, `https://doi.org/10.1007%2Fs10670-019-00165-8`
4. Goncharov, S., Neves, R., Proença, J.: Implementing hybrid semantics: From functional to imperative. In: Pun, V.K.I., Stolz, V., Simão, A. (eds.) Theoretical Aspects of Computing - ICTAC 2020 - 17th International Colloquium, Macau, China, November 30 - December 4, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12545, pp. 262–282. Springer (2020). `https://doi.org/10.1007/978-3-030-64276-1\_14`, `https://doi.org/10.1007/978-3-030-64276-1_14`
5. Lee, E.A.: Cyber physical systems: Design challenges. In: 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2008), 5-7 May 2008, Orlando, Florida, USA. pp. 363–369. IEEE Computer Society (2008). `https://doi.org/10.1109/ISORC.2008.25`, `https://doi.org/10.1109/ISORC.2008.25`
6. Leroy, X.: Formal verification of an optimizing compiler. In: 5th ACM & IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2007), May 30 - June 1st, Nice, France. p. 25. IEEE Computer Society (2007). `https://doi.org/10.1109/MEMCOD.2007.371254`, `https://doi.org/10.1109/MEMCOD.2007.371254`
7. Liu, J., Lv, J., Quan, Z., Zhan, N., Zhao, H., Zhou, C., Zou, L.: A calculus for hybrid CSP. In: Ueda, K. (ed.) Programming Languages and Systems - 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6461, pp. 1–15. Springer (2010). `https://doi.org/10.1007/978-3-642-17164-2\_1`, `https://doi.org/10.1007/978-3-642-17164-2_1`
8. Moggi, E.: Notions of computation and monads. Information and Computation **93**(1), 55–92 (1991). `https://doi.org/https://doi.org/10.1016/0890-5401(91)90052-4`, `https://www.sciencedirect.com/science/article/pii/0890540191900524`, selections from 1989 IEEE Symposium on Logic in Computer Science
9. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer (2018). `https://doi.org/10.1007/978-3-319-63588-0`, `https://doi.org/10.1007/978-3-319-63588-0`

10. Uustalu, T., Vene, V.: The essence of dataflow programming. In: Yi, K. (ed.) Programming Languages and Systems, Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3780, pp. 2–18. Springer (2005). `https://doi.org/10.1007/11575467\_2`, `https://doi.org/10.1007/11575467_2`