A logic for complex computing systems: Properties preservation along integration and abstraction

Marc Aiguier, Bilal Kanso

Scientific Annals of Computer Science vol. _, 2014, pp. 1–46

A logic for complex computing systems: Properties preservation along integration and abstraction

Marc Aiguier[1], Bilal Kanso[1]

Abstract

In a previous paper [1], we defined both a unified formal framework based on L.-S. Barbosa's components for modeling complex software systems, and a generic formalization of integration rules to combine their behavior. In the present paper, we propose to continue this work by proposing a variant of first-order fixed point modal logic to express both components and systems requirements. We establish the important property for this logic to be adequate with respect to bisimulation. We then study the conditions to be imposed to our logic (characterization of sub-families of formulæ) to preserve properties along integration operators, and finally show correctness by construction results. The complexity of computing systems results in the definition of formal means to manage their size. To deal with this issue, we propose an abstraction (resp. simulation) of components by components. This enables us to build systems and check their correctness in an incremental way.

Keywords: Component modeling, μ-calculus, coalgebra, correct by construction, refinement/abstraction

1 Introduction

Systems Engineering (SE) is an interdisciplinary branch of engineering which is focused on how large industrial systems (i.e. complex systems) should be

[1]Ecole Centrale de Paris
Laboratoire de Mathématiques Appliquées aux Systèmes (MAS)
Grande Voie des Vignes F-92295 Châtenay-Malabry, France
email: marc.aiguier@ecp.fr, bilal.kanso.ecp@gmail.com

designed, managed and maintained throughout their life cycle. Progressively emerged since the 50's, SE is characterized by a number of concepts, methods and organizational/technical practices that the industry has developed to deal with the complexity of systems design (see [4, 9, 31, 38, 44] for further details). At the heart of SE is the notion of system which is generally described as a set of interconnected components which, in turns, are themselves (recursively) defined as systems, interacting one another to participate permanently to a common goal. In mathematical terms, a system is commonly defined with models coming from:

- control theory and physics, that deal with systems as partial functions (dynamical systems may also be rewritten in this way), called transfer functions, of the form:

  $$\forall t \in T,\ y(t) = F(x, q, t)$$

  where x, q and y are inputs, states and outputs data-flows, and where T stands for time (usually considered in these approaches as continuous - see [4, 12, 43]).

- theoretical computer sciences and software engineering, with systems that can be depicted by models equivalent to different types of state-based machines, evolving on discrete times generally considered as a universal predefined sequence of steps, and whose coalgebras provide a general framework (see [23, 28, 37]).

The formal characterization of SE is a fundamental aspect which is concerned with the formal method integration within the scope of SE, i.e. within the design cycle of a complex system. The formalization of SE entails two basic aspects: the development of modeling languages for rigorously specify a systems design and the development of formal techniques for the analysis of the modeled system.

In a preceding paper [1], we introduced a formal abstract framework for modeling complex computing systems, which is based on Barbosa's coalgebraic definition of components [6, 7, 32]. In that respect ([1]) a *complex computing system* consists of the interconnection of a number of *components*, which are recursively combined by means of two basic operators: the Cartesian product and the feedback operator (i.e. two standard operators in the theory of dynamic and physical systems).

In [1], we restricted our formalisation to discrete (time) systems, i.e. systems for which time is considered as an order-isomorphic copy of natural

numbers. In [2] we extended such a discrete-time modelling approach, by proposing a novel formalism (based on deterministic Mealy automata), that, relying on results of non-standard analysis, allows one to consider homogeneously heterogeneous time scales (i.e. both continuous and discrete timing) for the modelled systems.
By extending Rutten's works [16] to Barbosa's components, in [1, 2] we then showed how causal transfer functions can be associated to system semantics allowing us to link with methods from control theory.

In this paper, we propose to further extend the formalization of SE for (discrete-time) complex computing systems, by considering two additional, fundamental aspects:

1. the possibility to express expected properties of a system, often called *system requirements*, that allow for formally analyzing the modeled system. This will be complementary to the approach followed in [1, 25] where a conformance testing theory had been defined to validate a system design.

2. the possibility to describe system behavior at different abstraction levels. For that, we propose to give a formal meaning to a central concept in SE, i.e. *component abstraction*. Such a concept can be seen as the inverse of refinement commonly used in software modeling [13, 20].

To fulfill the first aspect (system's properties verification), it is necessary to consider a framework that on one hand allows us to formally express meaningful requirements addressing a system's correctness, and on the other allows us to exhaustively check whether the considered system fulfills them. Since our modeling formalism is essentially based on the extension of Mealy automata with a monad T (i.e. thus allowing for capturing the most relevant computation structures including determinism, non-determinism and partiality [35]), it naturally follows that the language for stating system's requirements should allow one to express temporal properties of a system with the ability to express constraints that relate the production of output values from input ones.

Being mainly interested in this paper by theoretical results of behavior and property preservation, we propose to extend a logic that subsumes most of modal and temporal logics: the *μ-calculus* [5, 11, 27]. More precisely, following our work in [3], we propose a variant of first-order fixed point modal logic [26, 45]. This extension to the first-order will allow us to export expected properties from components to systems, and thus allowing to study

their preservation along integration operators.
The logic we introduce herein is then an adaptation to first-order of that presented in [10] to our components. Of course, this logic will probably be restricted to the propositional case when we are interested in future works in its computational aspects such as system synthesis [10] or the definition of model-checking algorithms. Here, being interested in showing how the truth of formulæ is preserved both by bisimulation and along integration and abstraction operators, the variant of first-order fixed point modal logic we propose is quite adequate.

The interest for studying property preservation is twofold: with respect to the integration operators, properties preservation allows for establishing "correct-by-construction" proofs [19] (whatever is proved to hold at components level is guaranteed to hold on the system resulting by composition of components); with respect to the abstraction operator, the interest of property preservation is one of complexity gain: the analysis of a system behavior at a more abstract level of description (hence at a reduced model size) obviously enjoys a reduced complexity.

Such preservation results, as we will show in the remainder, allow us to obtain an incremental design method which can be applied to development and validation of large and complex systems.

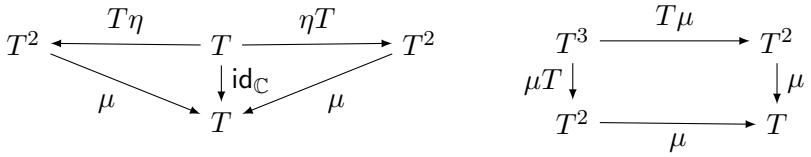
Moreover, they will be established both independently of the type of integration operator and for a large family of formulæ which anyway contains all the interesting properties we can express on systems such as deadlock freedom, reachability, existence of finite and infinite paths, etc.

The paper is structured as follows. Section 2 recalls the basic notions of monads the paper heavily relies upon. Section 3 recalls the formalism defined in [1], including the definition of Barbosa's components and that of integration operators (for components composition) while also introducing the notion of bisimulation with respect to systems. Section 4 introduces the logic we will consider to refer to (so-formalized) systems. i.e. an adaptation of the μ-calculus to the extension of Mealy automata with monads. The results at the core of the paper are illustrated in the last two sections. Section 5 outlines the properties preservation results with respect to the integration operators (i.e. Cartesian product and feedback); Section 6 describes the formalization of the abstraction operator and outlines the corresponding results, i.e. showing that system correctness is preserved along this operator.

2 Preliminaries

This paper relies on many terms and notations from the categorical theory of monads. We briefly introduce them here, but interested readers may refer to textbooks such as [8, 14] for further details.

Monads [30] are a powerful abstraction for adding structure to objects. Given a category $\mathbb{C}$, a **monad** consists of an endofunctor $T : \mathbb{C} \to \mathbb{C}$ equipped with two natural transformations $\eta : \mathsf{id}_{\mathbb{C}} \Rightarrow T$ and $\mu : T^2 \Rightarrow T$ which satisfy the conditions $\mu \circ T\eta = \mu \circ \eta T = \mathsf{id}_{\mathbb{C}}$ and $\mu \circ T\mu = \mu \circ \mu T$:



η is called the *unit* of the monad. Its components map objects in $\mathbb{C}$ to their naturally structured counterpart. μ is the *product* of the monad. Its components map objects with two levels of structure to objects with only one level of structure. The first condition states that a doubly structured object $\eta_{T(X)}(t)$ built by η from a structured object t is flattened by μ to the same structured object as a structured object $T(\eta_X)(x)$ made of structured objects built by η. The second condition states that flattening two levels of structure can be made either by flattening the outer (with $\mu_{T(X)}$) or the inner (with $T(\mu_X)$) structure first.

Let us consider a monad built on the powerset functor $\mathcal{P} : \mathbf{Set} \to \mathbf{Set}$. We use it to model non-deterministic state machines by replacing the target state of a transition by a set of possible states[2]. The component $\eta_S : S \to \mathcal{P}(S)$ of the unit of this monad has to build a set of states from a state. We can choose $\eta_S : s \mapsto \{s\}$. The component $\mu_S : \mathcal{P}(\mathcal{P}(S)) \to \mathcal{P}(S)$ of the product of the monad has to flatten a set of sets of states into a set of states. For a set of sets of states (S_i), $\forall i, S_i \in \mathcal{P}(S)$, we can choose $\mu_S : \{S_1 \ldots S_i \ldots\} \mapsto \cup S_i$.

In computing science, and in particular in the area of functional programming, monads have been used to represent many computation situations such as partiality, side-effects, exceptions, etc. [35]. More recently they have also been employed in complex systems' modeling where they have been used to obtain a more generic representation of components obtained by adding computation structures [6, 7, 32] to them.

[2] **Set** is the category of sets.

3 Components and systems

We recall the basic definitions on components and their composition [1], and introduce both simulation and bisimulation notions.

3.1 Component

Definition 1 *(Computation structure) A* **computation structure for component** *is a monad* $T : \mathbf{Set} \to \mathbf{Set}$ *together with two natural transformations* $\eta' : T \Longrightarrow \mathcal{P}$ *and* $\eta'^{-1} : \mathcal{P} \Longrightarrow T$ *such that* $\eta'^{-1} \circ \eta' = \mathsf{id}_T$.
A **computation morphism** σ *between two computation structures* $(T_1, \eta'_1, \eta'^{-1}_1)$ *and* $(T_2, \eta'_2, \eta'^{-1}_2)$ *is a natural transformation* $\sigma : T_1 \Longrightarrow T_2$ *such that* $\eta'_1 = \eta'_2 \circ \sigma$ *and* $\eta'^{-1}_2 = \sigma \circ \eta'^{-1}_1$.
Obviously, computation structures and computation morphisms form a category.

In the following, we will denote any computation structure (T, η', η'^{-1}) simply by T when this does not generate ambiguities.

Most monads used to represent computation situations satisfy the above condition. For instance, for the monad $T : S \mapsto \mathcal{P}(S)$, both η'_S and η'^{-1}_S are the identity on sets. For the functor $T : S \mapsto S \cup \{\bot\}$, η'_S associates the singleton $\{s\}$ to any $s \in S$ and the empty set to $\bot$, and η'^{-1} associates the state s to the singleton $\{s\}$ and $\bot$ to every other subset of S which is not a singleton.
It is important to note that less conventional monads such as the distribution monad classically defined by $T : S \mapsto \{\mu : S \to \mathbb{R}^{\geq 0} | \sum_{s \in S} \mu(s) = 1\}$ are not directly applicable here. Indeed, the natural transformation η' cannot be defined without losing the probability attached to states. To reacquire such a monad in the framework developed here, the powerset monad $\mathcal{P}$ should be applied to the set $S \times [0, 1]$.
Following the authors in [15], branching systems are often expressed as a function of the form $\alpha : X \to TFX$ where $T : \mathbf{Set} \to \mathbf{Set}$ is a monad (for branching type) and $F : \mathbf{Set} \to \mathbf{Set}$ is a functor (for transition type). Therefore, whereas in [15], the authors encapsulate distributions in branching, we would encapsulate distributions rather in transitions, i.e., $F : S \mapsto S \times [0, 1]$, and set $T = \mathcal{P}$ with conditions that for every $s \in S$, $\sum_{(s', p) \in \alpha(s)} p = 1$ (what is substantially similar to the notion of bag in [7] to introduce a (elementary) form of probabilistic non-determinism). Hence, the monad T being

the powerset monad, both η' and η'^{-1} remain the natural transformation identity.

The interest of computation structures as defined in Definition 1 is they will allow us to associate semantics (based on causal transfer functions) to components (see Definition 4).

Definition 2 *(Components) Let I and O be two sets denoting, respectively, the input and output domains. Let T be a computation structure. A* ***component*** *$\mathcal{C}$ is a coalgebra (S, α) for the signature $H = T(O \times \_)^I : \mathbf{Set} \longrightarrow \mathbf{Set}$ with a distinguished element $init \in S$ denoting the initial state of the component $\mathcal{C}$.*

By using the vocabulary of the theory of coalgebras [23, 37], a morphism of components is then a morphism between coalgebras, i.e. $f : (S, \alpha) \rightarrow (S', \alpha')$ is a **morphism** if $f : S \rightarrow S'$ is a mapping preserving initial states such that the following diagram commutes:

$$\begin{array}{ccc} S & \xrightarrow{\quad f \quad} & S' \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha'} \\ H(S) & \xrightarrow[\quad H(f) \quad]{} & H(S') \end{array}$$

Let us note $\mathbf{Comp}(H)$ the category of components.

Example 1 (Encoder/decoder) *We illustrate the notions previously mentioned with an encoder/decoder system. Many other examples can be found in [1, 24]. An* encoder/decoder *is usually used to guarantee certain characteristics (e.g. error detection) when transmitting data across a link. A simple example of such an encoder/decoder is represented in Figure 1. It consists of two parts:*

- *An* encoder *that takes in an incoming bit sequence and produces an encoded value which is then transmitted on the link. In our framework, this encoder is considered as a component $\mathcal{E} = (\{s_0, s_1\}, s_0, \alpha_1)$ where the transition function $\alpha_1 : \{s_0, s_1\} \longrightarrow (\{0, 1\} \times \{s_0, s_1\})^{\{0,1\}}$ is graphically shown in the left of Figure 1.*

- *A* decoder *that takes the values from the link and produces the original value. In our framework, this decoder is considered as a component $\mathcal{D} = (\{q_0, q_1\}, q_0, \alpha_2)$ where the transition function $\alpha_2 : \{q_0, q_1\} \longrightarrow (\{0, 1\} \times \{q_0, q_1\})^{\{0,1\}}$ is graphically shown in the right of Figure 1.*

As we can observe, both components are deterministic. Hence, they are defined over the signature $\mathsf{Id}(\{0,1\} \times \_)^{\{0,1\}}$ *where* Id *is the computation structure defined by the identity functor* Id *as monad together with* (η', η'^{-1}) *where for every set* S, $\eta'_S : s \mapsto \{s\}$ *and* η'^{-1}_S *is any mapping that associates* $\{s\}$ *to* s*, and every subset of* S *which is not a singleton to a* [3] *given* $s' \in S$*.*

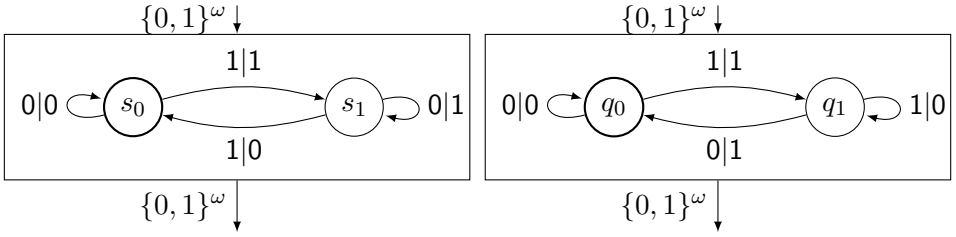


Figure 1: Encoder (on the left) and Decoder (on the right)

Following Rutten's works [16], component semantics can be defined by causal transfer functions.

Definition 3 *(Transfer function) Let* I *and* O *be two sets denoting the input and output domains, respectively. Let us*[4] *note* I^ω *(resp.* O^ω*) the set of mappings from* ω *to* I *(resp.* O*). A* **transfer function** $\mathcal{F} : I^\omega \longrightarrow O^\omega$ *is a function that is causal, i.e.:*

$$\forall n \in \omega, \forall x, y \in I^\omega, (\forall m, 0 \leq m \leq n, x(m) = y(m)) \Longrightarrow \mathcal{F}(x)(n) = \mathcal{F}(y)(n)$$

In the following, to simplify the notations, we will prefer to note $\eta'_{O\times S}(\alpha(s)(i))_{|_i}$ with $i = 1, 2$ rather than using the more standard notation $\mathcal{P}(\pi_i)(\eta'_{O\times S}(\alpha(s)(i)))$ for the power set image of the projections.

Definition 4 *(Component semantics) Let* $\mathcal{C} = (S, init, \alpha)$ *be a component over* $T(O \times \_)^I$ *and* $s \in S$*. Let us note* $\mathsf{beh}_\mathcal{C}(\mathbf{s})$ *the set of causal transfer functions* $\mathcal{F} : I^\omega \longrightarrow O^\omega$ *that associate to every* $x \in I^\omega$ *the stream* $y \in O^\omega$

[3] As already explained in [1], in computation structure is never required that the couple (η', η'^{-1}) is unique given a monad T. However, for most of monads, this will be the case. When it is not, the choice of η'^{-1} is often irrelevant because all of them do it.

[4] We note ω the least infinite ordinal, identified with the corresponding hereditarily transitive set.

such that there exists an infinite sequence of couples $(o_1, s_1), \ldots, (o_k, s_k), \ldots \in O \times S$ *satisfying:*

$$\forall j \geq 1, (o_j, s_j) \in \eta'_{O\times S}(\alpha(s_{j-1})(x(j-1)))$$

with $s_0 = s$*, and for every* $k \in \omega,\ y(k) = o_{k+1}$.
Hence, $\mathcal{C}$***'s semantics*** *is the set* $\mathsf{beh}_{\mathcal{C}}(init)$.

The interest of both natural transformations η' and η'^{-1} is they allow us to "compute" for an input sequence $(i_0, \ldots, i_{n-1})$ all the outputs o after going through any sequence of states $(s_0, \ldots, s_n)$ such that s_j is obtained from s_{j-1} by i_{j-1}. Without them, we could not characterise s_j with respect to $\alpha(s_{j-1})(i_{j-1})$ because nothing ensures that elements in $\alpha(s_{j-1})(i_{j-1})$ are (output, state) couples. Indeed, the monad T may yield a set with a structure different from $O \times S$. The mapping $\eta'_{O\times S}$ maps back to this structure. $\eta'^{-1}_{O\times S}$ is useful for going back to T.

Example 2 *The behaviour* $\mathsf{beh}_{\mathcal{E}}(s_0)$ *of the encoder component* $\mathcal{E}$ *presented in Example 1 is defined by the unique function* $\mathcal{F} : \{0,1\}^\omega \longrightarrow \{0,1\}^\omega$ *defined for every* $x \in \{0,1\}^\omega$ *by* $y \in \{0,1\}^\omega$ *such that:*

- $y(0) = x(0)$
- $\forall k, 0 < k < \omega$

$$y(k) = \begin{cases} 0 & \textit{if} \begin{cases} (y(k-1) = 0 \ \textit{and}\ x(k) = 0) \ \textit{or} \\ (x(k-1) = 0, y(k-1) = 1, \ \textit{and}\ x(k) = 1) \end{cases} \\ 1 & \textit{if} \begin{cases} (y(k-1) = 1 \ \textit{and}\ x(k) = 0) \ \textit{or} \\ (x(k-1) = y(k-1) = 0 \ \textit{and}\ x(k) = 1) \end{cases} \end{cases}$$

Under some standard conditions on the cardinality of $\mathsf{beh}_{\mathcal{C}}(s)$ for every state s, we showed in [1] the existence of a final component.

Next we define the standard notion of simulation and bisimulation [33, 34] which will play an important role to show the adequacy of the logic (see Section 4). Moreover, abstraction of components will be based on an extension of simulation in order to take into account components defined over different signatures while simulation and bisimulation are defined for components over a same signature.

Definition 5 *(Simulation and bisimulation) Let* $\mathcal{C}_1 = (S_1, init_1, \alpha_1)$ *and* $\mathcal{C}_2 = (S_2, init_2, \alpha_2)$ *be two components over a signature* $H = T(O \times \_)^I$*. A subset* $R \subseteq S_1 \times S_2$ *is a* ***simulation*** *if, and only if for all* $(s_1, s_2) \in S_1 \times S_2$ *and* $i \in I$*:*

$$s_1 \; R \; s_2 \Longrightarrow [\forall (o, s_1') \in \eta'_{O \times S_1}(\alpha_1(s_1)(i)), \exists (o, s_2') \in \eta'_{O \times S_2}(\alpha_2(s_2)(i)), s_1' \; R \; s_2']$$

We call R *a* **bisimulation** *if both* R *and its (relational) inverse* R^{-1} *are simulations.*

Finally, $\mathcal{C}_1$ *is* **similar** *(resp.* **bisimilar***) to* $\mathcal{C}_2$ *if there exists a simulation (resp. a bisimulation)* R *such that* $init_1 \; R \; init_2$*.*

As it is usual in the coalgebras theory, bisimulation can be expressed more concisely by the fact that the projections from R to S_1 and S_2 are morphisms, i.e. the following diagram commutes:

$$\begin{array}{ccccc} S_1 & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & S_2 \\ {\scriptstyle \alpha_1}\downarrow & & \downarrow{\scriptstyle \alpha_R} & & \downarrow{\scriptstyle \alpha_2} \\ H(S_1) & \xleftarrow{H(\pi_1)} & H(R) & \xrightarrow{H(\pi_2)} & H(S_2) \end{array}$$

All the basic facts on bisimulations remain true in our framework. Among others, the greatest bisimulation between $\mathcal{C}_1$ and $\mathcal{C}_2$, noted $\sim_{\mathcal{C}_1,\mathcal{C}_2}$ or simply $\sim$ when the context is clear, exists and is defined as the union of all bisimulations between $\mathcal{C}_1$ and $\mathcal{C}_2$.

Theorem 1 *Let* $\mathcal{C}_1 = (S_1, init_1, \alpha_1)$ *and* $\mathcal{C}_2 = (S_2, init_2, \alpha_2)$ *be two components over a signature* $T(O \times \_)^I$*. We have:*

$$\forall s_1 \in S_1, \forall s_2 \in S_2, s_1 \sim s_2 \Longleftrightarrow \mathsf{beh}_{\mathcal{C}_1}(s_1) = \mathsf{beh}_{\mathcal{C}_2}(s_2)$$

Proof.
($\Longrightarrow$) Let $i, j \in \{1, 2\}$ such that $i \neq j$. Let $\mathcal{F} \in \mathsf{beh}_{\mathcal{C}_i}(s_i)$. Let $x \in I^\omega$. By definition, there exists an infinite sequence of states $s_{i1}, \ldots, s_{ik}, \ldots \in S_i$ with $s_{i_1} = s_i$ such that for every $l \geq 1$, $(\mathcal{F}(x)(l+1), s_{i(l+1)}) \in \eta'_{O_i \times S_i}(\alpha_i(s_{il})(x(l)))$. By the fact that $s_1 \sim s_2$, there also exists an infinite sequence $s_{j1}, \ldots, s_{jk}, \ldots \in S_j$ with $s_{j_1} = s_j$ such that for every $l \geq 1$, $(\mathcal{F}(x)(l+1), s_{j(l+1)}) \in \eta'_{O_j \times S_j}(\alpha_j(s_{jl})(x(l)))$ and $s_{i(l+1)} \sim s_{j(l+1)}$, and then $\mathcal{F} \in \mathsf{beh}_{\mathcal{C}_j}(s_j)$.

($\Longleftarrow$) Let $R \subseteq S_1 \times S_2$ be the binary relation defined by:

$$s_1 \; R \; s_2 \Longleftrightarrow \begin{cases} \exists \mathcal{F} \in \mathsf{beh}_{\mathcal{C}_1}(s_1) \cap \mathsf{beh}_{\mathcal{C}_2}(s_2), \exists x \in I^\omega \\ \exists s_{11}, \ldots, s_{1k}, \ldots \in S_1, \exists s_{21}, \ldots, s_{2k}, \ldots \in S_2, \\ s_{11} = s_1 \wedge s_{21} = s_2 \wedge (\forall j \in \{1,2\}, \forall l \geq 1, \\ (\mathcal{F}(x)(l+1), s_{j(l+1)}) \in \eta'_{O_j \times S_j}(\alpha_j(s_{jl})(x(l))) \end{cases}$$

It is not difficult to show that R is a bisimulation.

□

3.2 Systems

Larger components are built through the composition of two basic integration operators: *cartesian product* and *feedback*.

Cartesian product. The cartesian product is a composition where both components are executed simultaneously when triggered by a pair of input values.

Definition 6 *(Cartesian product)*
Let $\mathcal{C}_1 = (S_1, init_1, \alpha_1)$ and $\mathcal{C}_2 = (S_2, init_2, \alpha_2)$ be two components over $H_1 = T(O_1 \times \_)^{I_1}$ and $H_2 = T(O_2 \times \_)^{I_2}$, respectively. The **cartesian product** *$\otimes(\mathcal{C}_1, \mathcal{C}_2)$ of $\mathcal{C}_1$ and $\mathcal{C}_2$, is the component $(S, (init_1, init_2), \alpha)$ over $H = T((O_1 \times O_2) \times \_)^{(I_1 \times I_2)}$ where:*

- *$S = S_1 \times S_2$ is the set of states,*
- *$init = (init_1, init_2)$ is the initial state,*
- *$\alpha : S \longrightarrow T((O_1 \times O_2) \times S)^{I_1 \times I_2}$ is the unique mapping such that the following diagram commutes*[5]*.*

$$\begin{array}{ccccc} S_1 & \xleftarrow{\pi_1} & S_1 \times S_2 & \xrightarrow{\pi_2} & S_2 \\ {\scriptstyle \alpha_1}\downarrow & & \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \alpha_2} \\ H_1(S_1) & \xleftarrow{T(\pi_1^o, \pi_1)^{\pi_1^i}} & H(S) & \xrightarrow{T(\pi_2^o, \pi_2)^{\pi_2^i}} & H_2(S_2) \end{array}$$

where $\pi_j^o : O_1 \times O_2 \to O_j$ and $\pi_j^i : I_1 \times I_2 \to I_j$ with $j = 1, 2$ are projections.

[5] α exists and is unique due to the universal property of the product in the category **Set**.

Example 3 *The Cartesian product $\otimes(\mathcal{E}, \mathcal{D})$ of the encoder component $\mathcal{E}$ and the decoder component $\mathcal{D}$ over the signature $\Sigma_\otimes = (I_\otimes, O_\otimes)$ with $I_\otimes = O_\otimes = \{0,1\} \times \{0,1\}$ is illustrated in Figure 2.*

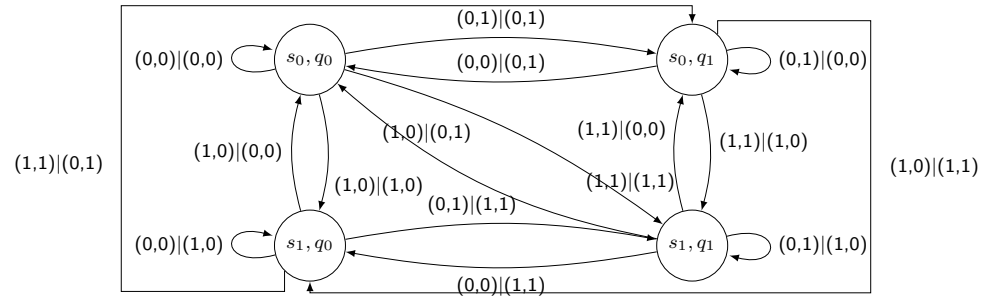


Figure 2: The product $\otimes(\mathcal{E}, \mathcal{D})$ of $\mathcal{E}$ and $\mathcal{D}$

Feedback. A component with *feedback* has directed cycles, where an output from a component is fed back to affect an input of the same component [29] (see Figure 3). That means the output of a component in any feedback composition depends on an input value that in turn depends on its own output value.The feedback operator is then a composition where some outputs of a component are linked to its inputs i.e. some outputs can be fed back as inputs. In order to obtain a model which fits our component definition, we need to take into account the computational effects of the monad T. This monad impacts both the evolution of component states and the observation of its outputs. Therefore, the feedback link between outputs and inputs carries the parts of the structure imposed by T to the inputs. First, we introduce feedback interfaces for defining correspondences between

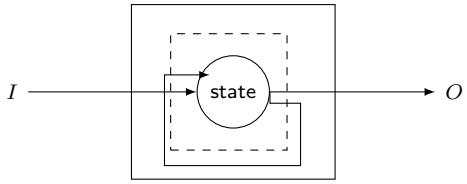


Figure 3: Illustration of a system with feedback

outputs and inputs of components and only keeping both inputs and outputs that are not involved in feedback.

Definition 7 *(Feedback interface) Let $H = T(O \times \_)^I$ be a signature. A* ***feedback interface*** *over H is a triple $\mathcal{I} = (f, \pi_i, \pi_o)$ where $f : I \times O \longrightarrow I$ is a mapping, and $\pi_i : I \longrightarrow I'$ and $\pi_o : O \longrightarrow O'$ are surjective mappings such that $\forall (i,o) \in I \times O, f(f(i,o),o) = f(i,o)$ and $\pi_i(i) = \pi_i(f(i,o))$.*

The mapping f specifies how components are linked and which parts of their interfaces are involved in the composition process. Both mappings π_i and π_o can be thought of as extensions of the hiding connective found in process calculi [21].

As this is usual when dealing with feedback, the existence of an instantaneous fixpoint is required.

Definition 8 *(Well-formed feedback composition) Let $H = T(O \times \_)^I$ be a signature. Let $\mathcal{C}$ be a component over H and $\mathcal{I} = (f, \pi_i, \pi_o)$ be a feedback interface over H. We say that the* ***feedback of $\mathcal{C}$ over $\mathcal{I}$ is well-formed*** *if, and only if for every $(i,s) \in I \times S$:*

1. **Fixpoint property.**
   $\eta'_{O\times S}(\alpha(s)(i)) \neq \emptyset \Longrightarrow \exists (o,s') \in O \times S, (o,s') \in \eta'_{O\times S}(\alpha(s)(f(i,o)))$
2. **Preservation property.**
   $\forall (o,s') \in O \times S, (o,s') \in \eta'_{O\times S}(\alpha(s)(f(i,o))) \Longrightarrow (o,s') \in \eta'_{O\times S}(\alpha(s)(i))$

By the fixpoint property of Definition 8, feedback will be allowed to make a pruning of transitions. Then, the preservation property of Definition 8 which did not occur in [1] will then ensure that there is no transition has been added through feedback. This last property will be useful to obtain our preservation results of Section 5.

Definition 9 *(Feedback) Let $\mathcal{I} = (f, \pi_i, \pi_o)$ be a feedback interface over $H = T(O \times \_)^I$. Let $\mathcal{C} = (S, init, \alpha)$ be a component over H whose the feedback over $\mathcal{I}$ is well-formed. The* ***feedback*** $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$ ***of*** $\mathcal{C}$ **over** $\mathcal{I}$*, is the component $\mathcal{C}' = (S, init, \alpha')$ over $H' = T(O' \times \_)^{I'}$ where α' is the mapping defined for every $s \in S'$ and every $i' \in I'$ by $\alpha'(s)(i') = \eta'^{-1}_{O'\times S'}(\Pi)$ where Π is the set:*

$$\{(o', s') \mid \exists (i,o) \in I \times O, (o,s') \in \eta'_{O\times S}(\alpha(s)(f(i,o))), \pi_i(i) = i', \pi_o(o) = o'\}$$

(when $\eta'_{O\times S}(\alpha(s)(i)) \neq \emptyset$ then so is Π because the feedback of $\mathcal{C}$ is well-formed over $\mathcal{I}$)

Here, feedback is defined in terms of its argument as concrete coalgebras. A definition of feedback has been defined in [1] in terms of its behaviors, and then built over the terminal model when it exists [6].

Complex operators and systems

Definition 10 *(Complex operator) The* ***set of complex operators*** *is inductively defined as follows:*

- $\_$ *is a complex operator of arity* 1*;*
- *if* op_1 *and* op_2 *are complex operators of arity* n_1 *and* n_2 *respectively, then* $op_1 \otimes op_2$ *is a complex operator of arity* $n_1 + n_2$*;*
- *if* op *is complex operator of arity* n *and* $\mathcal{I}$ *is a feedback interface, then* $\circlearrowleft_{\mathcal{I}}(op)$ *is a complex operator of arity* n*.*

Complex operators will not be necessarily defined when they are applied to a sequence of components. Indeed, for a complex operator of the form $\circlearrowleft_{\mathcal{I}}(op)$, according to the component $\mathcal{C}$ resulting from the evaluation of op, the interface $\mathcal{I}$ has to be defined over the signature of $\mathcal{C}$ and the feedback over $\mathcal{C}$ has to be well-formed. Hence, a system will be the component resulting from the evaluation of complex operators over a sequence of components, when it is defined.

Definition 11 *(Systems) The* ***set of systems*** *is inductively defined as follows:*

- *for any component* $\mathcal{C}$ *over a signature* H*,* $\_(\mathcal{C}) = \mathcal{C}$ *is a system over* H *and* $\_$ *is* ***defined for*** $\mathcal{C}$*;*
- *if* $op_1 \otimes op_2$ *is a complex operator of arity* $n = n_1 + n_2$ *then for every sequence* $(\mathcal{C}_1, \ldots, \mathcal{C}_{n_1}, \mathcal{C}_{n_1+1}, \ldots, \mathcal{C}_n)$ *of components with each* $\mathcal{C}_i$ *over* $H_i = T(O_i \times \_)^{I_i}$*, if both* op_1 *and* op_2 *are defined for* $\mathcal{C}_1, \ldots, \mathcal{C}_{n_1}$ *and* $\mathcal{C}_{n_1+1}, \ldots, \mathcal{C}_n$ *respectively, then* $op_1 \otimes op_2(\mathcal{C}_1, \ldots, \mathcal{C}_n) = \mathcal{S}_1 \otimes \mathcal{S}_2$ *with* $\mathcal{S}_1 = op_1(\mathcal{C}_1, \ldots, \mathcal{C}_{n_1})$ *and* $\mathcal{S}_2 = op_2(\mathcal{C}_{n_1+1}, \ldots, \mathcal{C}_n)$ *over* $H'_1 = T(O'_1 \times \_)^{I'_1}$ *and* $H'_2 = T(O'_2 \times \_)^{I'_2}$*, is a system over* $T((O'_1 \times O'_2) \times \_)^{I'_1 \times I'_2}$ *and* $op_1 \otimes op_2$ *is* ***defined for*** $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$*, else* $op_1 \otimes op_2$ *is* ***undefined for*** $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$*;*

[6] Indeed, as already explained, this terminal object does not always exit, and depends on constraints on the cardinality of $\mathsf{beh}_{\mathcal{C}}(s)$ for every state s.

- *if* $\circlearrowleft_{\mathcal{I}}(op)$ *is a complex operator of arity* n*, then for every sequence* $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ *of components, if op is defined for* $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ *with* $\mathcal{S} = op(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ *is over* H*,* $\mathcal{I}$ *is a feedback interface over* H *and the feedback of* $\mathcal{S}$ *is well-formed, then* $\circlearrowleft_{\mathcal{I}}(op)(\mathcal{C}_1, \ldots, \mathcal{C}_n) = \circlearrowleft_{\mathcal{I}}(\mathcal{S})$ *is a system over* H' *and*[7] $\circlearrowleft_{\mathcal{I}}(op)$ *is* ***defined for*** $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$*, else* $\circlearrowleft_{\mathcal{I}}(op)$ *is* ***undefined for*** $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$*.*

In [1], we showed that most of standard integration operators such as sequential, concurrent compositions or synchronous product can be obtained by composition of feedback and product. Moreover, both basic and complex operators can be defined on transfer functions (see [1] for their complete definitions). Hence, if for every complex operator *op*, we note $\overline{op}$ its equivalent on transfer functions, we have the following compositionality result:

Theorem 2 *(Compositionality) [1] Let op be a complex operator of arity* n*. Let* $\mathcal{C}_1, \ldots, \mathcal{C}_n$ *be components. If* $\mathcal{C} = op(\mathcal{C}_1, \ldots, \mathcal{C}_n)$*, then*

$$\mathsf{beh}_{\mathcal{C}}(init) = \overline{op}(\mathsf{beh}_{\mathcal{C}_1}(init_1), \ldots, \mathsf{beh}_{\mathcal{C}_n}(init_n))$$

Similar compositionally results have been obtained in [17, 18] but in a more categorical framework. Following notations in [17, 18], from set of complex operators we can easily generate an algebraic signature that can be seen as an FP-theory $\mathbb{L}$ over a basic set of sorts $S \subseteq \mathbf{Set} \times \mathbf{Set}$ where for $(\mathsf{In}, \mathsf{Out}) \in S$, In and Out denote input and output sets, respectively, and operations are complex operators (a monad T is supposed identical for every couple $(\mathsf{In}, \mathsf{Out})$ in the FP-theory $\mathbb{L}$). Outer models can then be defined along the functor $\mathbb{C} : \mathbb{L} \longrightarrow \mathbf{Cat}$ that associates to any couple $(\mathsf{In}, \mathsf{Out})$ the category $\mathbf{Comp}(H)$ with $H = T(\mathsf{Out} \times \_)^{\mathsf{In}}$ and to any operator the partial functor defined in Definition 10. Finally, inner models are defined by the natural transformation $X : \mathbf{1} \Longrightarrow \mathbb{C}$ where $\mathbf{1}$ is the constant functor that associates to any $S \in \mathbb{L}$ the trivial object category $\mathbf{1}$, which to any couple $(\mathsf{In}, \mathsf{Out})$ associates the final object [8] in $\mathbf{Comp}(H)$ and to any complex operator *op*, the mapping on behaviors noted $[[op]]$ in [17, 18] that contains *op* semantics on both components and transfer functions.

The difference between our works and those mentioned above is to have defined integration operations by composition of two elementary operators,

[7] H' is the signature of the feedback.

[8] This then requires constraints on monads to ensure the existence of such a terminal model in $Comp(H)$.

product and feedback and not as a term algebra. The interest was then to demonstrate a set of general properties on these integration operators such as the results of compositionality given in [1] or of correctness-by-construction that will be given in Section 5, by showing that these properties are valid for the product and feedback and are preserved by composition.

Hence, Theorem 2 is similar to Theorem 4.7 in [18] at least in these goals to establish a generic result of compositionality independent of a given integration operator.

Example 4 (Encoder/decoder) *In this example, we show how the encoder/decoder system can be built from both encoder $\mathcal{E}$ and decoder $\mathcal{D}$ components presented in Example 1. As Figure 4 illustrates, the encoder and decoder components are interconnected side-by-side in which the output (i.e.* 0 *or* 1*) of the first is the input of the second. This kind of composition is known as* sequential *(or* cascade*). The reaction of the resulting component consists then of a reaction of both $\mathcal{E}$ and $\mathcal{D}$, where $\mathcal{E}$ reacts first, produces its outputs, and then $\mathcal{D}$ reacts. That is to say, when $\mathcal{E}$ is triggered by an input i from the environment, $\mathcal{E}$ executes i and the produced output is fed to $\mathcal{D}$.*

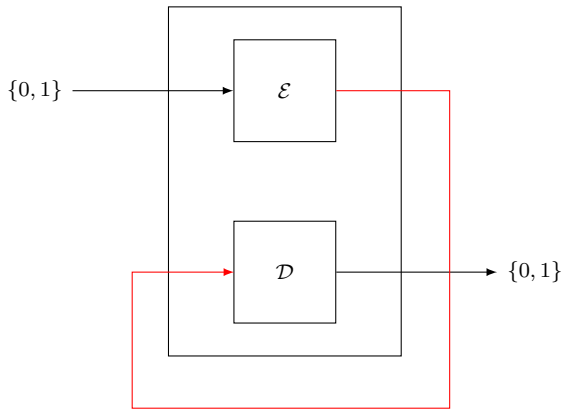


Figure 4: Sequential composition

As already said above, the sequential composition, noted $\triangleright$, can be naturally defined using both the feedback operator $\circlearrowleft_{\mathcal{I}}$ and the cartesian product $\otimes$ by:

$$\triangleright(\mathcal{E}, \mathcal{D}) = \circlearrowleft_{\mathcal{I}}(\mathcal{E} \otimes \mathcal{D}) \tag{1}$$

where $\mathcal{I} = (f, \pi_i, \pi_o)$ is the feedback interface defined for every $(i, i') \in$

$\{0,1\} \times \{0,1\}$ *and* $(o, o') \in \{0,1\} \times \{0,1\}$ *as follows:*

$$f((i,i'),(o,o')) = (i,o), \quad \pi_i((i,i')) = i \quad \textit{and} \quad \pi_o((o,o')) = o'$$

Let us now construct the encoder/decoder as a composition of the encoder and the decoder as illustrated in Equation 1. We first define the Cartesian product $\otimes(\mathcal{E},\mathcal{D})$ *of* $\mathcal{E}$ *and* $\mathcal{D}$ *as illustrated in Example 3 (see Figure 2). It is easy to see that* $\otimes(\mathcal{E},\mathcal{D})$ *is a well-formed feedback composition over* $\mathcal{I}$*. Let us check this for* (s_0, q_0)*:*

- $((0,0),(s_0,q_0)) \in \eta'(\alpha_\otimes((s_0,q_0))(f((0,0),(0,0))))_{|_1}$
- $((1,1),(s_1,q_1)) \in \eta'(\alpha_\otimes((s_0,q_0))(f((1,1),(1,1))))_{|_1}$
- $((0,0),(s_0,q_0)) \in \eta'(\alpha_\otimes((s_0,q_0))(f((0,1),(0,0))))_{|_1}$
- $((1,1),(s_1,q_1)) \in \eta'(\alpha_\otimes((s_0,q_0))(f((1,0),(1,1))))_{|_1}$

and for (s_1, q_0)*:*

- $((1,1),(s_1,q_1)) \in \eta'(\alpha_\otimes((s_1,q_0))(f((0,0),(1,1))))_{|_1}$
- $((0,0),(s_0,q_0)) \in \eta'(\alpha_\otimes((s_1,q_0))(f((1,1),(0,0))))_{|_1}$
- $((0,0),(s_0,q_0)) \in \eta'(\alpha_\otimes((s_1,q_0))(f((1,0),(0,0))))_{|_1}$
- $((1,1),(s_1,q_1)) \in \eta'(\alpha_\otimes((s_1,q_0))(f((0,1),(1,1))))_{|_1}$

Then, we can apply the feedback operator $\circlearrowleft_\mathcal{I}$ *on* $\otimes(\mathcal{E},\mathcal{D})$*. This leads to a new component* $\circlearrowleft_\mathcal{I}(\otimes(\mathcal{E},\mathcal{D}))$ *(see Figure 5) where all outputs of* $\mathcal{E}$ *(i.e* 0 *and* 1*) that are fed back to* $\mathcal{D}$ *are hidden (i.e. synchronized).*

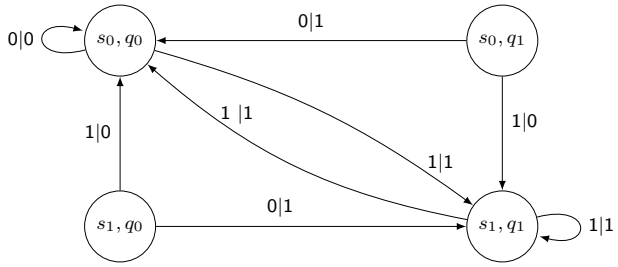


Figure 5: The encoder/decoder system

4 Component logic

We present a logic $\mathcal{L}$ for components and systems and define its semantics. This logic is a slight extension of μ-calculus to input and output values as done in [10], except that we will also quantify over input and output variables. Quantifying over input and output variables will allow us to manipulate formulæ independently from a given component or system (i.e. independently from a given signature). In the next section, we will show that properties involving quantification are the only properties which can be exported from a component to the system to which it belongs, because they are independent of signatures.

4.1 Syntax and satisfaction

In the next definition, we need a set of supplementary variables, called *fixed point* variables, to express formulæ in μ-calculus that denote recursion on states. To differentiate these variables from the input and output variables, we will denote input and output variables by $x, x', x_1, x_2, \ldots, y, y', y_1, y_2, \ldots$ and fixed point variables by $\overline{x}, \overline{x'}, \overline{x_1}, \overline{x_2}, \ldots, \overline{y}, \overline{y'}, \overline{y_1}, \overline{y_2}, \ldots$ [9]

Definition 12 *(Component formulæ) Let $H = T(O \times \_)^I$ be a signature. Let X be a set of fixed point variables. Let $V = V_i \coprod V_o$ be*[10] *a set variables such that variables in V_i (resp. V_o) are called input (resp. output) variables.* ***The set of formulæ $\mathcal{L}$*** *is given by the following grammar:*

$$\varphi \quad := \quad true \mid \quad \overline{x} \mid \quad x_i \downarrow y_o \mid \quad [x_i]\varphi \mid \quad \neg\varphi \mid \quad \varphi_1 \wedge \varphi_2 \mid \quad \forall x.\varphi \mid \quad \nu\overline{x}.\psi$$

where $x_i \in V_i \cup I, y_o \in V_o \cup O, x \in V, \overline{x} \in X$ and ψ is a formula in the logic that may contain occurrences of the variable $\overline{x}$ provided that every free occurrence of $\overline{x}$ in ψ occurs positively, i.e. within the scope of an even number of negations.

A formula φ is **closed** *when every fixed point variable $\overline{x}$ is within the scope of an operator $\nu\overline{x}$, and every input (resp. output) variable x (resp. y) is within the scope of a quantifier $\forall x$ (resp. $\forall y$).*

[9] It is worth to note that x and $\overline{x}$ are independent variables. $\overline{x}$ is not obtained from x by applying any mapping $\bar{\_}$ to input and output variables. The over line over letters is just a notation to differentiate fixed point variables from input and output variables.

[10] $\coprod$ is the disjoint union of sets.

Intuitively, a formula of the form $[x_i]\varphi$ stands for a state formula, and states that after performing the input x_i, all immediately reachable states satisfy φ. A formula of the form $x_i \downarrow y_o$ stands for an output formula, and states that it is possible to produce the output y_o after performing the input x_i. In practice, in such formulæ, x_i and y_o will be often elements of I and O respectively, and not variables in V_i and V_o. Finally, a formula of the form $\nu\overline{x}.\psi$ stands for a formula that expresses a recursion on states and is defined semantically as a function with fixed points. Indeed, each formula φ, free fixed point variables of which are among $\{\overline{x}_1, \ldots, \overline{x}_n\}$, can be semantically defined as a function $f_\varphi : \mathcal{P}(S)^n \to \mathcal{P}(S)$ that given n subsets of states in S yields the set of states that satisfy φ. Therefore, a formula φ of the form $\nu\overline{x}.\psi$ that can be seen as a "looping", denotes the greatest fixed point of the function $f_\varphi : S' \mapsto f_\psi(\ldots, S', \ldots)$ where $f_\psi : \mathcal{P}(S)^n \to \mathcal{P}(S)$ and $\overline{x}_i = \overline{x}$ (i.e. we force the free variable $\overline{x}_i$ to be interpreted by S' in ψ). It is well-known that such a fixed point exists when f_φ is monotonic on $\mathcal{P}(S)$. The condition that every free occurrence of $\overline{x}$ in ψ occurs positively, ensures monotonicity [11]. In this paper, we will not interpret each μ-formula φ as a function f_φ but will prefer to follow a more classical definition of satisfaction, i.e. defining a binary relation $\models$ between components and μ-formulæ.

Example 5 *We give here some formulæ that are about the encoder and decoder components $\mathcal{E}$ and $\mathcal{D}$ presented in Example 1.*

- A state having an outgoing transition labeled by $0|0$ can be reached:

$$\mu\overline{x}.\exists x.\langle x\rangle\overline{x} \ \vee 0 \downarrow 0$$

- There exists an infinite path of $\mathcal{E}$ that passes infinitely often by states having an outgoing transition labeled with $0|0$*:*

$$\nu\overline{x}.(\mu\overline{y}.(0 \downarrow 0 \vee \exists x\langle x\rangle\overline{y}) \wedge \exists x.\langle x\rangle\overline{x}$$

In the following, we will be sometimes led up to use some derived operators: $\langle x_i\rangle\varphi \iff \neg[x_i]\neg\varphi$, $false \iff \neg true$, $\exists x.\varphi \iff \neg\forall x.\neg\varphi$ and $\neg\nu\overline{x}.\psi \iff \mu\overline{x}.\neg\psi'$ where ψ' is the formula obtained from ψ by substituting $\neg\overline{x}$ for $\overline{x}$ in all free occurrences of $\overline{x}$ in ψ

Definition 13 *(Satisfaction) Let $\mathcal{C} = (S, init, \alpha)$ be a component over $T(O \times \_)^I$. Let φ be a formula in $\mathcal{L}$. For every fixed point variable interpretation $\lambda : X \to \mathcal{P}(S)$, every input and output variable interpretation*

$\iota : V \to I \cup O$ *such that for every* $x \in V_i$ *(resp.* $x \in V_o$*),* $\iota(x) \in I$ *(resp.* $\iota(x) \in O$*) and for every state* $s \in S$*,* $\mathcal{C}$ **satisfies** φ *for* s, ι *and* λ*, noted* $\mathcal{C} \models_{s,\iota,\lambda} \varphi$*, if, and only if:*

- $\mathcal{C} \models_{s,\iota,\lambda} true$
- $\mathcal{C} \models_{s,\iota,\lambda} \overline{x}$ *iff* $s \in \lambda(\overline{x})$
- [11] $\mathcal{C} \models_{s,\iota,\lambda} x_i \downarrow y_o$ *iff* $\iota(y_o) \in \eta'_{O\times S}(\alpha(s)(\iota(x_i)))_{|_1}$
- $\mathcal{C} \models_{s,\iota,\lambda} [x_i]\varphi'$ *iff* $\forall s' \in \eta'_{O\times S}(\alpha(s)(\iota(x_i)))_{|_2}, \mathcal{C} \models_{s',\iota,\lambda} \varphi'$
- $\mathcal{C} \models_{s,\iota,\lambda} \nu\overline{x}.\psi$ *iff* $\exists S' \subseteq S$ *such that* $s \in S'$ *and* $\forall s' \in S', \mathcal{C} \models_{s',\iota,\lambda[S'/\overline{x}]} \psi$
  Here, $\lambda[S'/\overline{x}]$ *is the interpretation such that,* $\lambda[S'/\overline{x}](\overline{x}) = S'$ *and* $\lambda[S'/\overline{x}](\overline{x'}) = \lambda(\overline{x'})$ *for every* $\overline{x'} \neq \overline{x}$*.*
- *Propositional connectives and quantifier are handled as usual.*

$\mathcal{C}$ **satisfies** *a formula* φ*, noted* $\mathcal{C} \models \varphi$*, if and only if for every valuation* λ *and every valuation* $\iota : V \to I \cup O$*,* $\mathcal{C} \models_{init,\iota,\lambda} \varphi$*.*

From Definition 13, it is obvious to show that for every closed formula φ and every state $s \in S$:

$$\forall \lambda : X \to \mathcal{P}(S), \forall \iota : V \to I \cup O, \mathcal{C} \models_{s,\iota,\lambda} \varphi \Leftrightarrow \mathcal{C} \models_{s,\emptyset} \varphi$$

where $\emptyset : X \to \mathcal{P}(S)$ is the fixed point variable interpretation that associates the emptyset $\emptyset$ to every $\overline{x} \in X$, and $\mathcal{C} \models_{s,\emptyset} \varphi$ means that for every input and output variable interpretation ι, $\mathcal{C} \models_{s,\iota,\emptyset} \varphi$. Hence, for closed formulæ, both input and output variable interpretation and fixed point variable interpretation are irrelevant, the latter being calculated for the satisfaction of the closed formula.

The derived operators are then interpreted as follows:

- $\mathcal{C} \models_{s,\iota,\lambda} \langle x_i \rangle \psi$ iff $\exists s' \in \eta'_{O\times S}(\alpha(s)(\iota(x_i)))_{|_2}, \mathcal{C} \models_{s',\iota,\lambda} \psi$
- $\mathcal{C} \models_{s,\iota,\lambda} \mu\overline{x}.\psi$ iff $\forall S' \subseteq S, (\{s' \in S \mid \mathcal{S} \models_{s',\iota,\lambda[S'/\overline{x}]} \psi\} \subseteq S' \Rightarrow s \in S')$

[11] By convention, $\iota(x_i) = x_i$ (resp. $\iota(y_o) = y_o$) when $x_i \in I$ (resp. $y_o \in O$).

We would be able to resort explicitly to the μ extension of logics induced by the functor H in the spirit of the standard coalgebraic logic such as defined in [36]. Indeed, following [36], the modality $[\_]$ can also be defined through natural transformations $\mu(i) : H \to \mathcal{P}$ where $i \in I$ such that for every set $S \in \mathbf{Set}$, $\mu(i)_S : f \mapsto \eta'_{O\times S}(f(i))_{|_2}$. The modality $[x_i]$ then becomes by taking the notations in [36], the modality $\Box_{\mu(x_i)}$, and has as semantics:

$$\mathcal{C} \models_{s,\iota,\lambda} \Box_{\mu(x_i)}\psi \Longleftrightarrow \forall s' \in \mu(\iota(x_i))_S(\alpha(s)), \mathcal{C} \models_{s',\iota,\lambda} \psi$$

In the same way, atoms of the form $i \downarrow o$ can be induced by natural transformations $\widehat{i \downarrow o} : H \to 2$ where $2 = \{true, false\}$ defined by:

$$\widehat{i \downarrow o}_S : f \mapsto \exists s' \in S, (o, s') \in \eta'_{O\times S}(f(i))$$

This leads to the following satisfaction definition:

$$\mathcal{C} \models_{s,\iota,\lambda} x_i \downarrow y_o \Longleftrightarrow (\widehat{\iota(x_i) \downarrow \iota(y_o)})_S \circ \alpha)(s) = true$$

This, it is on, will give a more categorical definition of the logic but perhaps less practical in its use. Our goal here is to give a formal framework for system engineering. That is why we prefer to follow the approach developed in [10].

4.2 Adequacy and characterization

The following theorem shows that $\mathcal{L}$ is expressive enough to characterize bisimilarity.

Theorem 3 *(Adequacy) Let $\mathcal{C}_1 = (S_1, init_1, \alpha_1)$ and $\mathcal{C}_2 = (S_2, init_2, \alpha_2)$ be two components over $T(O \times \_)^I$ that are finite image i.e. $\forall j = 1, 2, \forall (i, s) \in I \times S_j, |\eta'_{O\times S_j}(\alpha_j(s)(i))| < \infty$. Then, we have:*

$$(\forall \varphi, \mathcal{C}_1 \models \varphi \Leftrightarrow \mathcal{C}_2 \models \varphi) \Longleftrightarrow init_1 \sim init_2$$

Proof. To prove the *only if* implication, let us suppose that $init_1 \sim init_2$. Let $\lambda_2 : X \to \mathcal{P}(S_2)$. Let us define $\lambda_1 : X \to \mathcal{P}(S_1)$ by:

$$\lambda_1(\overline{x}) = \{s_1 \mid \exists s_2 \in \lambda_2(\overline{x}), s_1 \sim s_2\}$$

It is quite obvious to show by structural induction on formulæ that for every φ:

$$\mathcal{C}_1 \models_{init_1,\iota,\lambda_1} \varphi \Longleftrightarrow \mathcal{C}_2 \models_{init_2,\iota,\lambda_2} \varphi$$

We can apply the same reasoning from any valuation $\lambda_1 : X \to \mathcal{P}(S_1)$.

For the converse (the *if* part), let us define the relation $\equiv \subseteq S_1 \times S_2$ as follows: $s \equiv s'$ iff for every $\lambda : X \to \mathcal{P}(S_1)$, and every $\iota : V \to I \cup O$,

$$\forall \varphi, \mathcal{C}_1 \models_{s,\iota,\lambda} \varphi \Leftrightarrow \mathcal{C}_2 \models_{s',\iota,\lambda'} \varphi$$

where $\lambda' : X \to \mathcal{P}(S_2)$ is the mapping that associates the set $\{s' | \exists s \in \lambda(\overline{x}), s \equiv s'\}$ to each $\overline{x} \in X$. Let us show that $\equiv \subseteq \sim$. Let us suppose that $s \equiv s'$. By definition, this means for every $\lambda : X \to \mathcal{P}(S_1)$, every $i \in I$ and every $o \in \eta'_{O \times S_1}(\alpha_1(s)(i))_{|_1}$ that $\mathcal{C}_1 \models_{s,\lambda} i \downarrow o$, and then by hypothesis, $\mathcal{C}_2 \models_{s',\lambda'} i \downarrow o$, i.e. $o \in \eta'_{O \times S_2}(\alpha_2(s')(i))_{|_1}$. It remains to prove that for every $\overline{s} \in \eta'_{O \times S_1}(\alpha_1(s)(i))_{|_2}$, there exists $\overline{s}' \in \eta'_{O \times S_2}(\alpha_2(s')(i))_{|_2}$ such that $\overline{s} \equiv \overline{s}'$. For a given $\overline{s} \in \eta'_{O \times S_1}(\alpha_1(s)(i))_{|_2}$, let us suppose the opposite, i.e. there does not exist such a $\overline{s}'$. By hypothesis we have for every mapping λ that $\mathcal{C}_1 \models_{s,\lambda} \langle i \rangle \ true$ and then $\mathcal{C}_2 \models_{s',\lambda'} \langle i \rangle \ true$. Hence, the set $\eta'_{O \times S_2}(\alpha_2(s)(i))_{|_2}$ is not empty. Now, to have supposed the contrary, for every $\overline{s}' \in \eta'_{O \times S_2}(\alpha_2(s)(i))_{|_2}$, there exists a formula $\psi_{\overline{s}'}$ such that $\mathcal{C}_1 \models_{\overline{s},\iota,\lambda} \psi_{\overline{s}'}$ and $\mathcal{C}_2 \not\models_{\overline{s}',\iota,\lambda'} \psi_{\overline{s}'}$. By hypothesis, the cardinality of $\eta'_{O \times S_2}(\alpha_2(s)(i))_{|_2}$ is finite. Therefore, we have $\mathcal{C}_1 \models_{s,\iota,\lambda} \langle i \rangle \bigwedge_{\overline{s}' \in \eta'_{O \times S_2}(\alpha_2(s)(i))_{|_2}} \psi_{\overline{s}'}$ and $\mathcal{C}_2 \not\models_{s',\iota,\lambda'} \langle i \rangle \bigwedge_{\overline{s}' \in \eta'_{O \times S_2}(\alpha_2(s)(i))_{|_2}} \psi_{\overline{s}'}$, what is not possible as $s \equiv s'$. □

When bisimulations rest on the same component, we have further the following result:

Theorem 4 *(Characterization) Let $\mathcal{C} = (S, init, \alpha)$ be a component with finite image over a signature $H = T(O \times \_)^I$ such that I is finite. Then there exists for any $s \in S$, a closed formula φ_s such that:*

$$\forall s' \in S, s \sim s' \Leftrightarrow \mathcal{C} \models_{s',\emptyset} \varphi_s$$

Proof. *Let us associate to any state $s \in S$, the variable $x_s \in X$, and let us define the formula $\overline{\varphi}_s = \nu x_s.\psi_s$ where $\psi_s = \bigwedge_{i \in I, (o,s') \in \eta'_{O \times S}(\alpha(s)(i))} \langle i \rangle \ x_{s'} \wedge i \downarrow o$. Then, let us define φ_s as the formula obtained from $\overline{\varphi}_s$ recursively as follows:*

- $\Gamma_0 = \{x_s\}$ *and* $\varphi_s^0 = \overline{\varphi}_s$*;*

- φ_s^i *is the formula obtained from* φ_s^{i-1} *by replacing every variable* $x_{s'} \notin \Gamma_{i-1}$ *by* $\overline{\varphi}_{s'}$, *and*

$$\Gamma_i = \Gamma_{i-1} \cup \{x_{s'} \mid x_{s'} \text{ has been replaced by } \overline{\varphi}_{s'} \text{ in } \varphi_s^{i-1}\}$$

Then, let us set $\varphi_s = \varphi_s^\omega$. S *being finite, this process is terminating. Hence, every fixed point variable in* φ_s *is within the scope of fixed point operator* ν.

Let us suppose that $s \sim s'$. *Then, we can easily show by induction on the number of nested occurrences of* ν*-formulæ in* φ_s *that* $\mathcal{C} \models_{s,\emptyset} \varphi_s$. *Let us suppose that this number is one. This means that there exists* $i \in I$ *and* $o \in O$ *such that* $(o, s) \in \eta'_{O\times S}(\alpha(s)(i))$ *and then* φ_s *is of the form* $\nu x_s.\langle i\rangle\ x_s \wedge i \downarrow o$. *It is obvious that in this case* $\mathcal{C} \models_{s,\emptyset} \varphi_s$. *It is sufficient to choose* $S' = \{s\}$. *Let us suppose that the number of nested occurrences of* ν*-formulæ in* φ_s *is greater that one. Then, this means that* φ_s *is of the form* $\nu x_s. \bigwedge_{i\in I,(o,s')\in\eta'_{O\times S}(\alpha(s)(i))} \langle i\rangle\ \varphi_{s'} \wedge i \downarrow o$ *where* $\varphi_{s'}$ *is a closed formula except maybe for the variable* x_s. *By definition, we know that* $(o, s') \in \eta'_{O\times S}(\alpha(s)(i))$. *By induction hypothesis, we have that* $\mathcal{C} \models_{s',\emptyset} \varphi_{s'}$, *and by hypothesis* $\mathcal{C} \models_{s,\emptyset} i \downarrow o$. $\varphi_{s'}$ *is closed except for* x_s. *Therefore* $\mathcal{C} \models_{s,[x_s/\{s\}]} \langle i\rangle\varphi_{s'}$. *We can then conclude that* $\mathcal{C} \models_{s,\emptyset} \varphi_s$. *By Theorem 3, since* $s \sim s'$, *we also have* $\mathcal{C} \models_{s',\emptyset} \varphi_s$.
Conversely, let us define the binary relation $\equiv$ *on* S *by:*

$$s \equiv s' \Leftrightarrow \mathcal{C} \models_{s',\emptyset} \varphi_s$$

Let us show that $\equiv$ *is a bisimulation over* S. *Let* $i \in I$ *and* $(o, \overline{s}) \in \eta'_{O\times S}(\alpha(s)(i))$. *By definition,* $\mathcal{C} \models_{s',\emptyset} i \downarrow o$. *It remains to prove there exists* $\overline{s'}$ *such that* $(o, \overline{s'}) \in \eta'_{O\times S}(\alpha(s')(i))$ *and* $\overline{s} \equiv \overline{s'}$. *Let us suppose the contrary, i.e.* $\mathcal{C} \not\models_{\overline{s}',\emptyset} \varphi_{\overline{s}}$. *We then have that* $\mathcal{C} \not\models_{s',\emptyset} \varphi_s[x_{\overline{s}}/\varphi_{\overline{s}}]$. *As* φ_s *is closed, we also have that* $\mathcal{C} \not\models_{s',\emptyset} \varphi_s$ *which is impossible since* $s \equiv s'$. *The same reasoning can be carried out for* $\equiv^{-1}$. □

5 Correctness-by-construction

Here, we are interested in building correct systems from correct components, i.e. we are going to give correctness-by-construction results. These correctness-by-construction results rest on component properties that can be exported to systems. These exported properties have then to be able to

be expressed independently from any component and system. Indeed, the correctness-by-construction results can only concern formulæ that do not contain concrete inputs and outputs (i.e. some $i \in I$ and $o \in O$) so that they can be interpreted by both the component and the system where it is plugged on. Therefore, they relate all the formulæ in $\mathcal{L}$ containing no input and output values, i.e. all the formulæ defined by the following grammar:

$$\varphi \quad := \quad true \mid \quad \overline{x} \mid \quad [x]\varphi \mid \quad \neg\varphi \mid \quad \varphi \wedge \varphi \mid \quad \forall x.\varphi \mid \quad \nu\overline{x}.\varphi \qquad (2)$$

where $\overline{x} \in X$ and $x \in V_i$ (the set of input variables). Note that whatever the signature H considered, the set of formulæ defined by Grammar (2) is always the same. This will be also the case for other logics defined in this section.

This grammar is sufficient to express most of interesting general properties on both systems and components such as the fact they are deadlock freedom:

$$\nu\overline{x}.(\exists x.\langle x\rangle true \wedge \forall y.[y]\overline{x})$$

or the fact any path is finite:

$$\mu\overline{x}.\forall x.[x]\overline{x}$$

or conversely, there exists an infinite path:

$$\nu\overline{x}.\exists x.\langle x\rangle\overline{x}$$

It would be easy to put a set of propositional variables P in signatures, and then to add a mapping $\delta : S \to 2^P$ to components. In this case, we would be able to express supplementary properties such as it is possible to reach a state satisfying a propositional variable p:

$$\mu\overline{x}.p \vee \exists x.\langle x\rangle\overline{x}$$

Such formulæ are completely preserved along Cartesian product, that is to say any system of the form $\otimes(\mathcal{C}_1, \mathcal{C}_2)$ satisfies all the properties of its components $\mathcal{C}_i$ (for $i = 1, 2$) and nothing more.

Proposition 1 *(Preservation by product) Let $\mathcal{C}_1 = (S_1, init_1, \alpha_1)$ and $\mathcal{C}_2 = (S_2, init_2, \alpha_2)$ be two components over H_1 and H_2, respectively. Then, for every formula φ defined by Grammar (2), we have:*

$$(\forall i = 1, 2,\ \mathcal{C}_i \models \varphi) \Longleftrightarrow \mathcal{C} \models \varphi$$

where $\mathcal{C} = (S, init, \alpha)$ *is the Cartesian product* $\mathcal{C} = \otimes(\mathcal{C}_1, \mathcal{C}_2)$ *of* $\mathcal{C}_1$ *and* $\mathcal{C}_2$.

Proof. By structural induction over φ, we first prove the following property: $\forall (s_1, s_2) \in S, \forall \lambda : X \to \mathcal{P}(S), \forall \iota : V \to I \cup O$,

$$\mathcal{C}_i \models_{s_i, \iota_{|_i}, \lambda_{|_i}} \varphi, i = 1, 2 \Longleftrightarrow \mathcal{C} \models_{(s_1,s_2),\iota,\lambda} \varphi$$

where $\lambda_{|_i} : X \to \mathcal{P}(S_i)$ is defined by:

$$\overline{x} \mapsto \{s'_i \mid \exists s'_j \in S_j, j \neq i, (s'_1, s'_2) \in \lambda(\overline{x})\}$$

$\iota_{|_i} : V \to I_i \cup O_i$ is defined by: $x \mapsto \iota(x)_{|_i}$.

Therefore, let $(s_1, s_2) \in S$. Let $\lambda : X \to \mathcal{P}(S)$ be a valuation. Let $\iota : V \to I \cup O$ be a variable interpretation (recall that $I = I_1 \times I_2$ and $O = O_1 \times O_2$).

Basic case: this is obvious for *true*. For $\varphi = \overline{x}$, the equivalence rests on the following equivalence, that is true by definition of $\lambda_{|_i}$ for every $i = 1, 2$:

$$(s_1, s_2) \in \lambda(\overline{x}) \Longleftrightarrow s_i \in \lambda_{|_i}(\overline{x}), i = 1, 2$$

General case: many cases have to be considered:

- $\varphi = [x]\psi$. Let $(s'_1, s'_2) \in \eta'_{O \times S}(\alpha((s_1, s_2))(\iota(x)))_{|_2}$. By induction hypothesis, we have:

  $$\mathcal{C}_i \models_{s'_i, \iota_{|_i}, \lambda_{|_i}} \psi, i = 1, 2 \Longleftrightarrow \mathcal{C} \models_{(s'_1,s'_2),\iota,\lambda} \psi$$

  By definition, if $(s'_1, s'_2) \in \eta'_{O \times S}(\alpha((s_1, s_2))(\iota(x))_{|_2}$, then for every $i = 1, 2$, $s'_i \in \eta'_{O_i \times S_i}(\alpha_i(s_i)(\iota_{|_i}(x)))_{|_2}$. If we suppose that $\mathcal{C}_i \models_{s_i, \iota_{|_i}, \lambda_{|_i}} [x]\psi$ for every $i = 1, 2$, then $\mathcal{C}_i \models_{s'_i, \iota_{|_i}, \lambda_{|_i}} \psi$, whence we can conclude that $\mathcal{C} \models_{(s_1,s_2),\iota,\lambda} [x]\psi$.

  Let us suppose that $\mathcal{C} \models_{(s_1,s_2),\iota,\lambda} [x]\psi$, and let $s'_i \in \eta'_{O_i \times S_i}(\alpha_i(s_i)(\iota_{|_i}(x))_{|_2}$ for each $i = 1, 2$. Therefore, we have that $\mathcal{C} \models_{(s'_1,s'_2),\iota,\lambda} \psi$, whence for every $i = 1, 2$, we can conclude that $\mathcal{C}_i \models_{s_i, \iota_{|_i}, \lambda_{|_i}} [x]\psi$.

- $\varphi = \nu\overline{x}.\psi$. Let us prove the *only if* part. Let us suppose $S' \subseteq S$ such that $(s_1, s_2) \in S'$ and for every $(s'_1, s'_2) \in S'$, $\mathcal{C} \models_{(s'_1,s'_2),\iota,\lambda[S'/\overline{x}]} \psi$. By induction hypothesis, we have that $\mathcal{C}_i \models_{s'_i, \iota_{|_i}, \lambda[S'/\overline{x}]_{|_i}} \psi$ that is equivalent

to $\mathcal{C}_i \models_{s'_i,\iota_{|_i},\lambda_{|_i}[S'_{|_i}/\overline{x}]} \psi$ where $S'_{|_i} = \{s'_i \mid \exists s'_j \in S_j, j \neq i, (s'_1, s'_2) \in S'\}$, whence we can conclude that $\mathcal{C}_i \models_{s_i,\iota_{|_i},\lambda_{|_i}} \nu\overline{x}.\psi$.

Conversely, let us suppose for every $i = 1, 2$ there exists $S'_i \subseteq S_i$ such that $s_i \in S'_i$ and for every $s'_i \in S'_i$, $\mathcal{C}_i \models_{s'_i,\iota_{|_i},\lambda_{|_i}[S'_i/\overline{x}]} \psi$. Let us set $S' = S'_1 \times S'_2$. Obviously, we have $(s_1, s_2) \in S'$. By the induction hypothesis, we can write $\mathcal{C} \models_{(s'_1,s'_2),\iota,\lambda[S'/\overline{x}]} \psi$ for every $(s'_1, s'_2) \in S'$ whence we can conclude that $\mathcal{C} \models_{(s_1,s_2),\iota,\lambda} \nu\overline{x}.\psi$.

- The cases for the propositional connectives $\wedge, \neg$ and the quantifier $\forall$ are obvious.

Therefore, let us suppose that for every formula φ and every $i = 1, 2$, $\mathcal{C}_i \models \varphi$. Let $\lambda : X \to \mathcal{P}(S)$ and $\iota : V \to I \cup O$. By hypothesis, we have that for every $i = 1, 2$ that $\mathcal{C}_i \models_{init_i,\iota_{|_i},\lambda_{|_i}} \varphi$ and then $\mathcal{C} \models_{init,\iota,\lambda} \varphi$.

Inversely, let us suppose that for every formula $\mathcal{C} \models \varphi$. Let $i \in \{1, 2\}$, $\lambda_i : X \to \mathcal{P}(S_i)$ and $\iota_i : V_i \to I_i \cup O_i$. By definition, there exists $\lambda : X \to \mathcal{P}(S)$ and $\iota : V \to I \cup O$ such that $\lambda_{|_i} = \lambda_i$ and $\iota_{|_i} = \iota_i$. By hypothesis, we have that $\mathcal{C} \models_{init,\iota,\lambda} \varphi$, and then $\mathcal{C}_i \models_{init_i,\iota_i,\lambda_i} \varphi$. □

On the contrary, with feedback, as we can see in Example 4, when applying the feedback to the Cartesian product of encoder $\mathcal{E}$ and decoder $\mathcal{C}$, we prune transitions. Hence, we cannot ensure property preservation from $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$ to its component $\mathcal{C}$. Actually, the problem comes from formulæ of the form $[x]\psi$ which are here of the type of emergent properties for composability, that is, properties that call into question components behaviour (here $\mathcal{C}$'s behaviours) when components are integrated into systems (here through feedback). Indeed, emergence being the result of transition pruning, it may be that in $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$ all the transitions that invalidate φ have been removed from $\mathcal{C}$.
Dually, formulæ of the form $\langle x\rangle\psi$ cannot be preserved anymore from $\mathcal{C}$ to $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$. As we are interested by a correctness-by-construction result, to preserve properties along feedback, we need to restrict the expressive power of the logic and then preventing formulæ of the form $\langle x\rangle\psi$. Hence, we obtain a first correctness-by-construction result for feedback by restricting formulæ defined by Grammar (2) to the following grammar:

$$\varphi \;:=\; true \mid \; false \mid \; \overline{x} \mid \; [x]\varphi \mid \; \varphi \; C \; \varphi \mid \; Qx.\varphi \mid \; @\overline{x}.\varphi' \qquad (3)$$

where $C \in \{\wedge, \vee, \Rightarrow\}$, $Q \in \{\forall, \exists\}$, $@ \in \{\mu, \nu\}$, and φ' is a formula built according to the rules of Grammar (3) in which $\overline{x}$ occurs positively. Positiveness of $\overline{x}$ in a formula φ where $\overline{x}$ is free, is defined as follows:

- if $\varphi = true$ or $\varphi = false$, then $\overline{x}$ is positive in φ;
- if $\varphi = \overline{y}$, then $\overline{x}$ is positive in φ iff $\overline{y} = \overline{x}$;
- if $\varphi = [x]\varphi'$, then $\overline{x}$ is positive in φ iff $\overline{x}$ is positive in φ';
- if $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi = \varphi_1 \vee \varphi_2$, then $\overline{x}$ is positive in φ iff $\overline{x}$ is positive in φ_1 and φ_2;
- if $\varphi = \varphi_1 \Rightarrow \varphi_2$, then $\overline{x}$ is positive in φ iff $\overline{x}$ is not positive in φ_1 or $\overline{x}$ is positive in φ_2;
- if $\varphi = @\overline{y}.\varphi'$ with $@ \in \{\mu, \nu\}$ (necessarily, we have $\overline{y} \neq \overline{x}$), then $\overline{x}$ is positive in φ iff $\overline{x}$ is positive in φ'.

The fact that $\overline{x}$ occurs positively in φ', also ensures that $f_{\varphi'}$ is monotone.

Proposition 2 *(Preservation for feedback 1) Let $\mathcal{C} = (S, \alpha, init)$ be a component over H, and let $\mathcal{I} = (f, \pi_i, \pi_o)$ be a feedback interface such that $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) = (S, \alpha', init')$ is defined. For every φ defined by Grammar (3), we have:*

$$\mathcal{C} \models \varphi \Longrightarrow \circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models \varphi$$

Proof. By structural induction over φ, we first prove the following property: $\forall s \in S, \forall \lambda : X \to \mathcal{P}(S), \forall \iota : V \to I \cup O$

$$\mathcal{C} \models_{s,\lambda,\iota} \varphi \Longrightarrow \circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s,\lambda,\iota'} \varphi$$

where $\iota' : V \to \pi_i(I) \cup \pi_o(O)$ is defined by:

$$\iota'(x) = \begin{cases} \pi_i(\iota(x)) & \text{if } x \in V_i \\ \pi_o(\iota(x)) & \text{otherwise (i.e. } x \in V_o) \end{cases}$$

The basic cases defined by the formulæ $true$, $false$ and $\overline{x}$ are obvious. For such formulæ φ, we can even prove for every $s \in S$, every $\lambda : X \to \mathcal{P}(S)$ and every $\iota : V \to I \cup O$ that

$$\mathcal{C} \models_{s,\lambda,\iota} \varphi \Longleftrightarrow \circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s,\lambda,\iota'} \varphi$$

For the general case, many cases have to be considered:

- $\varphi = [x]\psi$. Let $s' \in \eta'_{O' \times S}(\alpha'(s)(\iota'(x)))_{|_2}$. By definition, this means there are $i \in I$ and $o \in O$ such that $(s', o) \in \eta'_{O \times S}(\alpha(s)(f(i, o)))$ and $\iota'(x) = \pi_i(i) = \pi_i(f(i, o))$. Hence, by the second property of Definition 8, $(s', o) \in \eta'_{O \times S}(\alpha(s)(i))$. Therefore, by hypothesis, we have that $\mathcal{C} \models_{s',\lambda,\iota} \psi$. Hence, by the induction hypothesis, we have $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s',\lambda,\iota'} \psi$, whence we can conclude that $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s,\lambda,\iota'} \varphi$.

- $\varphi = \nu\overline{x}.\psi$. By hypothesis, we know there exists $S' \subseteq S$ such that $s \in S'$ and for every $s' \in S'$, $\mathcal{C} \models_{s',\lambda[S'/\overline{x}],\iota} \psi$. By the induction hypothesis, we then have for every $s' \in S'$ that $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s',\lambda[S'/\overline{x}],\iota'} \psi$. Therefore, we can conclude that $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s,\lambda,\iota'} \varphi$.

- $\varphi = \mu\overline{x}.\psi$. Let $S' \subseteq S$ such that $\{s' \mid \circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s',\lambda[S'/\overline{x}],\iota'} \psi\} \subseteq S'$. By the induction hypothesis, we have $\{s' \mid \mathcal{C} \models_{s',\lambda[S'/\overline{x}],\iota} \psi\} \subseteq \{s' \mid \circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s',\lambda[S'/\overline{x}],\iota'} \psi\}$, and then $s \in S'$. Therefore, we can conclude that $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s,\lambda,\iota'} \varphi$.

- The cases for the propositional connectives $\wedge, \vee, \Rightarrow$ and the quantifiers $\exists, \forall$ are not difficult to treat.

Hence, let $\lambda : X \to \mathcal{P}(S)$ be a valuation and let $\iota' : V \to I' \cup O'$ be a variable interpretation. By definition, there exists $\iota : V \to I \cup O$ such that for every $x \in V$, $\iota'(x) = \begin{cases} \pi_i(\iota(x)) & \text{if } x \in V_i \\ \pi_o(\iota(x)) & \text{otherwise (i.e. } x \in V_o) \end{cases}$
By hypothesis, we have $\mathcal{C} \models_{init,\lambda,\iota} \varphi$, and then by the property above, we also have $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{init,\lambda,\iota'} \varphi$, whence we can conclude $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models \varphi$. □

The problem is that Grammar (3) is too restrictive and many examples of formulæ given previously are not taken into account by such a grammar. When we look more closely at this kind of formulæ, they are closed and their semantics is expressed by the membership of an outgoing state of a transition labeled by x or y to a set of states. Such formulæ being closed, their semantics then consists in checking that the state as argument of validation belongs to the smallest or the greatest fixpoint according to the way fixed point variables are quantified [12]. Thus, the formulæ that we will take into account are all the formulæ defined with the following supplementary restrictions:

[12] Here, we are only interested by the membership of states into a set of states and not non-membership, because all fixed point variables are in the scope of an even number of negations. So at the end, if one pushes the negation to be adjacent to atoms, negations cancel.

- Negation is removed, and
- For every sub-formula of the form $\langle x\rangle\psi$ and $[x]\psi$, the variable x is in the direct scope of a quantifier $\forall$ or $\exists$, respectively, and ψ is a **positive propositional formula**, i.e. a formula defined by the following grammar:

$$\psi \quad := \quad true \mid \quad \overline{x} \mid \quad \psi \wedge \psi \mid \quad \psi \vee \psi \mid \quad \psi \Rightarrow \psi$$

Therefore, the formulæ which will be considered here are generated by the following grammar:

$$\varphi \quad := \quad \theta \mid \quad @\overline{x}.\varphi' \mid \quad \varphi \wedge \varphi \mid \quad \varphi \vee \varphi \mid \quad \varphi \Rightarrow \varphi \tag{4}$$

where $@ \in \{\mu, \nu\}$, φ' is a formula built according to the rules of Grammar (4) in which $\overline{x}$ occurs positively, and θ is a **state formula** defined as follows:

$$\theta \quad := \quad \psi \mid \quad \forall x.[x]\psi \mid \quad \exists x.\langle x\rangle\psi \mid \quad \theta \wedge \theta \mid \quad \theta \vee \theta \mid \quad \theta \Rightarrow \theta$$

where ψ is a positive propositional formula.

The expressive power of such formulæ is now sufficient to describe all the examples of general properties given at the beginning of this section.

Here, when formulæ are closed, the obtained preservation result is both sufficient and necessary.

Proposition 3 *(Preservation for feedback 2) Let $\mathcal{C} = (S, \alpha, init)$ be a component over H, and let $\mathcal{I} = (f, \pi_i, \pi_o)$ be a feedback interface such that $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) = (S, \alpha', init)$ is defined. For every closed formula φ defined by Grammar (4), we have:*

$$\mathcal{C} \models \varphi \Longleftrightarrow \circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models \varphi$$

Proof. Let φ be a closed formula. Let $\overline{x}_1, \ldots, \overline{x}_n$ be its fixed point variables such that:

1. for every i, $1 \leq i \leq n$, there exists a unique sub-formula of the form $@_i\overline{x}_i.\varphi_i$ in φ with $@_i \in \{\mu, \nu\}$,

2. and for every i, j, $1 \leq i, j \leq n$, if $pos_\varphi(@_i\overline{x}_i.\varphi_i) \prec pos_\varphi(@_j\overline{x}_j.\varphi_j)$ then $i < j$. [13]

By the structure of φ, for every i, $1 \leq i \leq n$, if we note S_i the least (resp. the greatest) fixpoint (in that $@_i$ is μ or ν) for the mapping $f : S' \mapsto \{s' \in S \mid \mathcal{C} \models_{s',\lambda[S'/\overline{x}_i]} @_i\overline{x}_i.\varphi_i\}$ where $S' \subseteq S$ and $\lambda : \overline{x}_j \mapsto \begin{cases} S_j & \text{if } j < i \\ \emptyset & \text{otherwise} \end{cases}$, and S'_i is the least (resp. the greatest) fixpoint (in that $@_i$ is μ or ν) for the mapping $f' : S' \mapsto \{s' \in S \mid \circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s',\lambda'[S'/\overline{x}_i]} @_i\overline{x}_i.\varphi_i\}$ where $S' \subseteq S$ and $\lambda' : \overline{x}_j \mapsto \begin{cases} S'_j & \text{if } j < i \\ \emptyset & \text{otherwise} \end{cases}$, then the proof of Proposition 3 amounts to show the following equivalence: $\forall i, 0 \leq i \leq n, \forall s \in S$,

$$\mathcal{C} \models_{s,\emptyset[S_1/\overline{x}_1,\ldots,S_n/\overline{x}_n]} \varphi'_i \iff \circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s,\emptyset[S'_1/\overline{x}_1,\ldots,S'_n/\overline{x}_n]} \varphi'_i \qquad (5)$$

where φ'_0 is obtained from φ by replacing recursively every sub-formula $@_i\overline{x}_i.\varphi_i$ by φ'_i (i.e. all fixpoint operators have been removed).

The proof of (5) is done by structural induction over φ'_i. Among the basic cases, the only two cases a little complicated are $\forall x.[x]\psi$ and $\exists x.\langle x\rangle\psi$.

- $\varphi'_{\mathbf{i}} = \forall \mathbf{x}.[\mathbf{x}]\psi$:

  ($\Rightarrow$) Let $\iota' : V \to I' \cup O'$ and let $s' \in \eta'_{O'\times S}(\alpha'(s)(\iota'(x)))_{|_2}$. By definition, this means there exists $(i, o) \in I \times O$ such that $(o, s') \in \eta'_{O\times S}(\alpha(s)(f(i, o)))$ and $\pi_i(i) = \iota'(x)$. By the preservation property of Definition 8, we have that $(o, s') \in \eta'_{O\times S}(\alpha(s)(i))$. Therefore, by hypothesis, we have $\mathcal{C} \models_{s',\emptyset[S_1/\overline{x}_1,\ldots,S_n/\overline{x}_n]} \psi$. It is not difficult to show by structural induction on ψ and by the fact that for every i, $1 \leq i \leq n$, $S'_i \subseteq S_i$ (a simple consequence of the way the least and the greatest fixpoints are calculated), that in this case $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s',\emptyset[S'_1/\overline{x}_1,\ldots,S'_n/\overline{x}_n]} \psi$. We then conclude that $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s,\emptyset[S'_1/\overline{x}_1,\ldots,S'_n/\overline{x}_n]} \varphi'_i$.
  ($\Leftarrow$) Let $\iota : V \to I \cup O$ and let $s' \in \eta'_{S\times O}(\alpha(s)(\iota(x)))_{|_2}$. By the fixed-point property, this means there exists $(o, s'') \in \eta'_{O\times S}(\alpha(s)(f(\iota(x), o)))$.

[13] Formulæ can be standardly represented by trees. Using a standard numbering of tree nodes by natural number strings, we can refer to positions in a formula tree. Thus, given a formula tree φ, a position of φ is a string $\omega \in \mathbb{N}^*$ which represents the path from the root of φ to the sub-formula φ' whose the root occurs at that position. We note $pos_\varphi(\varphi')$ this position, and $\prec$ is the lexicographic order over positions.

Moreover, we have $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s'',\emptyset[S'_1/\overline{x}_1,\ldots,S'_n/\overline{x}_n]} \psi$. By definition, ψ is either logically equivalent to *true* or expresses some membership properties on $\overline{x}_j$. Hence, for each of these $\overline{x}_j$, $S'_j \neq \emptyset$, and then by the way the least and the greatest fixpoints are calculated, $s' \in S_j$, whence we can conclude $\mathcal{C} \models_{s,\emptyset[S_1/\overline{x}_1,\ldots,S_n/\overline{x}_n]} \varphi'_i$.

- $\varphi'_{\mathbf{i}} = \exists \mathbf{x}.\langle \mathbf{x} \rangle \psi$:

  ($\Rightarrow$) Let $\iota : V \rightarrow I \cup O$ such that there exists $s' \in \eta'_{O\times S}(\alpha(s)(\iota(x)))_{|_2}$ satisfying $\mathcal{C} \models_{s',\emptyset[S_1/\overline{x}_1,\ldots,S_n/\overline{x}_n]} \psi$. By the fixpoint property of Definition 8, this means there exists $o \in O$ such that $\eta'_{O\times S}(\alpha(s)(f(\iota(x),o))) \neq \emptyset$, and then $\eta'_{O'\times S}(\alpha'(s)(\pi_i(\iota(x)))) \neq \emptyset$. ψ defining membership properties on some $\overline{x}_j$, for such $\overline{x}_j$, by the way both least and greatest fixpoints are calculated, we have that $S'_j \cap \eta'_{O'\times S}(\alpha'(s)(\pi_i(\iota(x))))_{|_2} \neq \emptyset$, whence we can conclude $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s,\emptyset[S'_1/\overline{x}_1,\ldots,S'_n/\overline{x}_n]} \varphi'_i$.
  ($\Leftarrow$) Let $\iota' : V \rightarrow I' \cup O'$ such that there exists $s' \in \eta'_{O'\times S}(\alpha'(s)(\iota'(x)))_{|_2}$ satisfying $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) \models_{s',\emptyset[S'_1/\overline{x}_1,\ldots,S'_n/\overline{x}_n]} \psi$. By definition, $\exists (i,o) \in I \times O$ such that $(o,s') \in \eta'_{O\times S}(\alpha(s)(f(i,o)))$ and $\pi_i(i) = \iota'(x)$. By the preservation property of Definition 8, we have that $(o,s') \in \eta'_{O\times S}(\alpha(s)(i))$. It is not difficult to show by structural induction on ψ and by the fact that for every i, $1 \leq i \leq n$, $S'_i \subseteq S_i$ (a simple consequence of the way the least and the greatest fixpoints are calculated), that in this case $\mathcal{C} \models_{s',\emptyset[S_1/\overline{x}_1,\ldots,S_n/\overline{x}_n]} \psi$. We can conclude $\mathcal{C} \models_{s,\emptyset[S_1/\overline{x}_1,\ldots,S_n/\overline{x}_n]} \varphi'_i$.

□

Theorem 5 *(Correct-by-construction) Let $op(\mathcal{C}_1,\ldots,\mathcal{C}_n)$ be a system over a signature $H = T(O \times \_)^I$ where each $\mathcal{C}_i$ is over $H_i = T(O_i \times \_)^{I_i}$ for every i, $1 \leq i \leq n$. Let φ be a formula satisfying the same conditions as in Proposition 2 (resp. in Proposition 3).Then:*

$$\forall i, 1 \leq i \leq n, \mathcal{C}_i \models \varphi \Longrightarrow \textit{(resp. } \Longleftrightarrow\textit{)}\ op(\mathcal{C}_1,\ldots,\mathcal{C}_n) \models \varphi$$

Proof. By induction on the structure of the complex operator *op* by applying Propositions 1 and 2 (resp. 3) . □

Hence, the result we have established here is a result of correctness-by-construction by composability ([41]). This result is sufficiently general to be applied to both most of integration operators and a large family of formulæ (at least, most of formulæ expected on system behaviors).

6 Abstraction/Refinement

6.1 Definition

Abstraction allows us to consider the right systemic level for describing systems, according to modeling needs. It is thus a fundamental tool to deal with the growing complexity of systems by hiding unnecessary low-level details related to system behavior. It helps people to better understand a system and makes easier the formal analysis by working on abstraction of systems.

By Definition 11, systems being defined finally as components, abstraction of systems will be based on the abstraction of components.

Abstraction can be seen as the inverse of refinement. Then, as this is usual when dealing with the formalization of systems by state-based machines, component abstraction will be naturally defined from the concept of simulation to consider that transitions of the abstract component are preserved in the concrete one [32]. However, the concept of simulation as defined in Definition 5 needs to be revisited in order to take into account the fact that the two systems in play in the abstraction can be defined over different signatures. The main idea is abstraction/simulation can be understood as a zoom from the point of view of overall behavior, i.e. a transition in the abstract system can be "zoomed" into a succession of transitions in the concrete system in such a way all the intermediate observations are only inputs and outputs that are not contained in the abstract signature.

Definition 14 *(Simulation revisited) Let $H = T(O \times \_)^I$ and $H' = T(O' \times \_)^{I'}$ be two signatures such that $I' \subseteq I$ and $O' \subseteq O$. Let $\mathcal{C} = (S, init, \alpha)$ and $\mathcal{C}' = (S', init', \alpha')$ be two components over H and H', respectively. A binary relation $R \subseteq S' \times S$ is a* **simulation** *if, and only if s' R s implies for every $i' \in I'$, and every $(o', \overline{s}') \in \eta'_{O' \times S'}(\alpha'(s')(i'))$, there exists $i_1, \ldots, i_n \in I$, $s_0 \in S$ and $(o_1, s_1), \ldots, (o_n, s_n) \in O \times S$ such that:*

- $s = s_0$;
- $i_1 = i'$ *and* $o_n = o'$;
- $\forall j, 1 \leq j \leq n, (o_j, s_j) \in \eta'_{O \times S}(\alpha(s_{j-1})(i_j))$;
- $\forall j, 1 < j \leq n, i_j \in I \setminus I'$;

- $\forall j, 1 \leq j < n, o_j \in O \setminus O'$;
- $\overline{s}' \ R \ s_n$.

R is a **bisimulation** *if, and only if R is a simulation and $s' \ R \ s$ further implies for every $i_1, \ldots, i_n \in I$ and every $(o_1, s_1), \ldots, (o_n, s_n) \in O \times S$ such that:*

- $\forall j, 1 \leq j \leq n, (o_j, s_j) \in \eta'_{O \times S}(\alpha(s_{j-1})(i_j))$ *with* $s_0 = s$;
- $i_1 \in I'$ *and* $o_n \in O'$;
- $\forall j, 1 < j \leq n, i_j \in I \setminus I'$;
- $\forall j, 1 \leq j < n, o_j \in O \setminus O'$

there exists $i' \in I'$, $\overline{s}' \in S'$ such that $(o, \overline{s}') \in \eta'_{O' \times S'}(\alpha'(s')(i_1))$ and $\overline{s}' \ R \ s_n$.

If R is a simulation (resp. a bisimulation) and $s' \ R \ s$, then s' is said **similar** *(resp.* **bisimilar***) to s.*

$\mathcal{C}'$ is **similar** *(resp.* **bisimilar***) to $\mathcal{C}$ if there exists a simulation (resp. bisimulation) R such that $init' \ R \ init$.*

It is straightforward to see from definitions that when $\mathcal{C}$ and $\mathcal{C}'$ are over the same signature H, simulation (resp. bisimulation) in Definition 14 is equivalent to the notion of simulation (resp. bisimulation) given in Definition 5.

Definition 15 *(Component abstraction) Let $H = T(O \times \_)^I$ and $H' = T(O' \times \_)^{I'}$ be two signatures such that $I' \subseteq I$ and $O' \subseteq O$. Let $\mathcal{C} = (S, init, \alpha)$ and $\mathcal{C}' = (S', init', \alpha')$ be two components over H and H', respectively. Then, $\mathcal{C}'$ is an* **abstraction** *of $\mathcal{C}$, noted $\mathcal{C} \rightsquigarrow \mathcal{C}'$ if, and only if $\mathcal{C}'$ is similar (according to Definition 14) to $\mathcal{C}$.*
Abstraction is further **complete***, noted $\mathcal{C} \bowtie \mathcal{C}'$, when $\mathcal{C}'$ and $\mathcal{C}$ are bisimilar (according to Definition 14).*

The concepts introduced in Definition 15 are similar to the notions of interface refinement (but restricted to inclusions), replaceability and behavior refinement in [32]. Indeed, abstraction reflects that the behavior observed from $\mathcal{C}'$ are structural restriction of $\mathcal{C}$ with respect to the behavioral model captured by T. More precisely, following the works of Hughes and Jacobs

in [22], Meng and Barbosa in [32] abstractly define behavior refinement through the notion of simulation based on a refinement preorder. Here this refinement preorder $\sqsubseteq$ is the binary relation over $T(O \times S)^I$ defined by:

$$f \sqsubseteq g \iff (\forall i \in I, \eta'_{O\times S}(f(i)) \subseteq \eta'_{O\times S}(g(i)))$$

In [32], simulations are restricted to morphisms, called forward morphisms, and then are defined for components over a same signature H. Hence, following the notations given just above, $\mathcal{C} \rightsquigarrow \mathcal{C}'$ if, and only if there exists a morphism $h : S' \to S$ such that for every $s' \in S'$, $Th(\alpha'(s')) \sqsubseteq \alpha(h(s'))$.

Example 6 (Coffee machine) *Figure 6 shows a simple example of a coffee machine $\mathcal{S}_r$ over the signature $\mathcal{P}_{\text{fin}}(O \times \_)^I$ where $I = \{\text{coin}, \text{coffee}, \text{enough}, \text{not\_enough}\}$ and $O = \{\text{refund}, \text{abs}, \text{served}, \text{verify}\}$. Figure 7 shows an abstraction of $\mathcal{S}_r$ defined by the component $\mathcal{S}_a$ over the signature $\mathcal{P}_{\text{fin}}(O' \times \_)^{I'}$ where $I' = \{\text{coin}, \text{coffee}\}$ and $O' = \{\text{refund}, \text{abs}, \text{served}\}$. $\mathcal{S}_r$ works similarly to $\mathcal{S}_a$ except $\mathcal{S}_r$ behavior is refined by adding a verification step. Indeed, when the user asks for a coffee, the coffee machine interface does a verification step which consists in checking whether the introduced coin is enough or not for buying a coffee.*

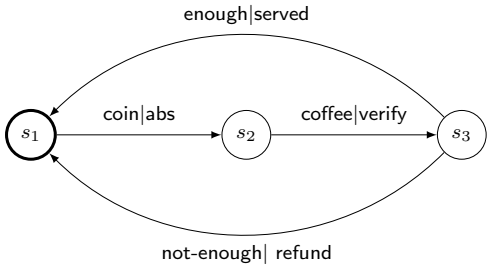


Figure 6: Concrete coffee machine

It is easy to see that $\mathcal{S}_a$ is an abstraction of $\mathcal{S}_r$ which is further complete. Indeed, it is sufficient to consider the binary relation $R = \{(s'_1, s_1), (s'_2, s_2)\}$.

An important question we must address concerns consistency of our definition of system abstraction: is the behavior of the abstraction of a system the abstraction of the behavior of this system? To answer this question, we have first to define what is the abstraction of system behaviors.

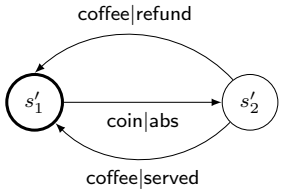


Figure 7: Abstract coffee machine

Definition 16 *(Transfer function abstraction) Let I, I', O and O' be sets of input and output values, respectively, such that $I' \subseteq I$ and $O' \subseteq O$. Let $\mathcal{F} : I^\omega \to O^\omega$ and $\mathcal{F}' : I'^\omega \to O'^\omega$ be two transfer functions. $\mathcal{F}'$ is an* **abstraction** *of $\mathcal{F}$ if, and only if for every $x' \in I'^\omega$, there exists $x \in I^\omega$ such that:*

- *for $j = 0$, there exists $k_0 \in \mathbb{N}$ such that:*
  - $x'(0) = x(0)$ *and* $\mathcal{F}(x)(k_0) = \mathcal{F}'(x')(0)$;
  - $\forall l, 1 \leq l \leq k_0, x(l) \in I \setminus I'$;
  - $\forall l, 0 \leq l < k_0, \mathcal{F}(x)(l) \in O \setminus O'$
- *for $j = n$, there exists $k \in \mathbb{N}$ such that:*
  - $k_n = k_{n-1} + k$;
  - $x'(n) = x(k_{n-1} + 1)$ *and* $\mathcal{F}(x)(k_n) = \mathcal{F}'(x')(n)$;
  - $\forall l, 2 \leq l \leq k, x(k_{n-1} + l) \in I \setminus I'$;
  - $\forall l, 1 \leq l < k, \mathcal{F}(x)(k_{n-1} + l) \in O \setminus O'$

Theorem 6 *(Consistency of abstraction) The behaviour of the abstraction of a system is the abstraction of the behaviour of this system, i.e. when $\mathcal{C} \rightsquigarrow \mathcal{C}'$, then for every $\mathcal{F}' \in beh_{\mathcal{C}'}(init')$ there exists $\mathcal{F} \in beh_{\mathcal{C}}(init)$ such that $\mathcal{F}'$ is an abstraction of $\mathcal{F}$. If $\mathcal{C} \bowtie \mathcal{C}'$, then we have the reverse correspondence.*

Proof. The proof of this Theorem is straightforward regarding the definition of the system abstraction, which is defined as abstracting the behaviour of the initial system. □

Now, the question is: what are properties preserved along abstraction operator? Formally, if $\mathcal{C} \rightsquigarrow \mathcal{C}'$, then for every formula φ over H' such that

$\mathcal{C}' \models \varphi$, is $\mathcal{C} \models \varphi$? The problem is φ has to be transformed to take into account the fact that transitions in $\mathcal{C}'$ may have been expanded into paths in $\mathcal{C}$. This then leads to the following result:

Theorem 7 *Let $\mathcal{C}$ and $\mathcal{C}'$ be two components over H and H', respectively, such that $\mathcal{C} \rightsquigarrow \mathcal{C}'$. Then, for every closed formula φ over H' (i.e. φ is a closed formula defined according to the grammar given in Definition 12), and for every $s \in S$ and $s' \in S'$ such that s' is similar to s, we have:*

$$\mathcal{C}' \models_{s',\emptyset} \varphi \Longrightarrow (\forall \varphi' \in \overline{\varphi}_s, \mathcal{C} \models_{s,\emptyset} \varphi')$$

where $\overline{\varphi}_s$ is the set of formulæ over H defined by structural induction over φ as follows:

- *if φ is true or $\overline{x}$, then $\overline{\varphi}_s = \{\varphi\}$;*
- *if $\varphi = i' \downarrow o'$, then by hypothesis there exists in $\mathcal{C}$ a finite path, $s \xrightarrow{i' \downarrow o_1} s'_1 \xrightarrow{i_2 \downarrow o_2} \ldots \xrightarrow{i_n \downarrow o'} s_n$ such that:*
  - $\forall j, 1 < j \leq n, i_j \in I \setminus I'$;
  - $\forall j, 1 \leq j < n, o_j \in O \setminus O'$.

  We then set

  $$\overline{\varphi}_s = \{i' \downarrow o_1 \wedge <i'> i_2 \downarrow o_2 \wedge \ldots \wedge <i'><i_2> \ldots <i_{n-1}> i_n \downarrow o_n\}$$

- *if φ is $[i]\psi$, then $\overline{\varphi}_s = \{[i]\psi' | \psi' \in \bigcup_{\overline{s} \in \eta'_{O \times S}(\alpha(s)(i))_{|_2}} \overline{\psi}_{\overline{s}}\}$.*
- *if φ is $\forall x.\psi[x]$ with $x \in V_i$ (resp. $x \in V_o$), then $\overline{\varphi}_s = \bigcup_{i' \in I'} \overline{\psi[x/i']}_s$ (resp. $\overline{\varphi}_s = \bigcup_{o' \in O'} \overline{\psi[x/o']}_s$).*
- *if φ is $\neg\psi$, $\varphi_1 \wedge \varphi_2$, $\nu\overline{x}.\psi$, then $\overline{\varphi}_s$ is*
  - $\{\neg\psi' | \psi' \in \overline{\psi}_s\}$
  - $\{\varphi'_1 \wedge \varphi'_2 | \varphi'_j \in \overline{\varphi_j}_s, j = 1, 2\}$

– $\{\nu\overline{x}.\psi' | \psi' \in \overline{\psi}_s\}$

(Let us remark when components are image finite and both I' and O' are finite sets, $\overline{\varphi}_s$ can be generated effectively.)

Proof. The proof is quite simple and is done by structural induction over φ. □

The equivalence holds when dealing with complete abstraction.

This result reflects the fact that all the properties studied at the abstract level are preserved at the more concrete one modulo the fact that input and output variables have been replaced by values in I' and O', respectively. Thus, at the more concrete level we can only focus on the properties which were not included in the abstract behaviour. For instance, a property of the form $\forall x.[x]\varphi$ which has been checked to be valid at the abstract level, should be checked at the more concrete level only with values in $I \setminus I'$.

6.2 Abstraction along integration

Large systems usually may require many abstraction steps. This leads to the notion of sequential composition of abstraction steps. Usually, composition of abstraction is mainly divided into two concepts:

1. *horizontal composition* that deals with abstraction of subparts of complex systems when they are structured into "blocks". In our framework, blocks are components as defined in Definition 2;

2. *vertical composition* that deals with many abstraction steps.

Horizontal composition. An important result in the systemic approach is to preserve abstraction through integration. Hence, given a complex operator *op* with arity n, a sequence of components $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ and an abstraction $\mathcal{C}_i \rightsquigarrow \mathcal{C}'_i$, does $op(\mathcal{C}_1, \ldots, \mathcal{C}_i, \ldots, \mathcal{C}_n) \rightsquigarrow op(\mathcal{C}_1, \ldots, \mathcal{C}'_i, \ldots, \mathcal{C}_n)$ hold? This of course has also to be proven for complete abstraction. First, the inclusion conditions on input and output sets should be satisfied, i.e. if $op(\mathcal{C}_1, \ldots, \mathcal{C}_i, \ldots, \mathcal{C}_n)$ is over $H = T(O \times \_)^I$ and $op(\mathcal{C}_1, \ldots, \mathcal{C}'_i, \ldots, \mathcal{C}_n)$ is over $H' = T(O' \times \_)^{I'}$, then the inclusions between input and output values have to be preserved, i.e. $I' \subseteq I$ and $O' \subseteq O$. Obviously, this will depend on the structure of the complex operator *op*. Actually, because of feedback, *op* will be also prone to be modified into a complex operator $\overline{op}$. Indeed, the reason

is because of feedback interface $\mathcal{I}$. Let $H = T(O \times \_)^I$ and $H' = T'(O' \times \_)^{I'}$ be two signatures such that $I' \subseteq I$ and $O' \subseteq O$. Let $\mathcal{I} = (f, \pi_i, \pi_o)$ with $\pi_i : I \to \overline{I}$ and $\pi_o : O \to \overline{O}$, be a feedback interface over H. $\mathcal{I}$ has to be able to be extended into a feedback interface $\mathcal{I}' = (f', \pi_i', \pi_o')$ over $T'(O' \times \_)^{I'}$, to deal with inputs and outputs in I' and O', respectively. The question is how to extend $\mathcal{I}$ into $\mathcal{I}'$?
We could set: $f' = f_{|_{I' \times O'}}$. The problem is, given $(i', o') \in I' \times O'$, $f(i', o')$ does not necessarily belong to I'. When this holds, it is easy to define $\overline{I}'$ and $\overline{O}'$, and then π_i' and π_o':

- $\overline{I}' = \pi_i(I')$ and $\overline{O}' = \pi_i(O')$;
- $\pi_i' = \pi_{i|_{I'}}$ and $\pi_o' = \pi_{o|_{O'}}$.

We will then say that a feedback interface $\mathcal{I} = (f, \pi_i, \pi_o)$ over H is **compatible** with a signature $H' = T(O' \times \_)^{I'}$ such that $I' \subseteq I$ and $O' \subseteq O$ if: $\forall (i', o') \in I' \times O', f(i', o') \in I'$. In the following, we will always suppose this property.

To preserve abstraction along integration, we need to impose a condition on some transitions. Again, this is due to the feedback operator. Indeed, let us suppose $\mathcal{C} \rightsquigarrow \mathcal{C}'$ where $\mathcal{C} = (S, init, \alpha)$ and $\mathcal{C}' = (S', init', \alpha')$. Let us suppose $\circlearrowleft_{\mathcal{I}}(\mathcal{C}) = (S, init, \overline{\alpha})$ and $\circlearrowleft_{\mathcal{I}'}(\mathcal{C}') = (S', init', \overline{\alpha}')$ where $\mathcal{I}'$ has been defined as previously. As $\mathcal{C} \rightsquigarrow \mathcal{C}'$, there exists a simulation $R \subseteq S' \times S$. Is this simulation preserved after feedback? Actually, without a supplementary condition on transitions, the answer is not. Indeed, let $s' \ R \ s$, and let $i' \in \overline{I}'$ and $(o', \overline{s}') \in \eta'_{\overline{O}', S'}(\overline{\alpha}'(s')(i'))$. By definition of feedback, there exists $i \in I'$ and $o \in O'$ such that $(o, \overline{s}') \in \eta'_{O' \times S'}(\alpha'(s')(f'(i, o)))$. By definition of simulation, there exists a path in $\mathcal{C}$

$$s \xrightarrow{f'(i,o)|o_1} s_1 \xrightarrow{i_2|o_2} \dots \xrightarrow{i_n|o} \overline{s}$$

such that $\overline{s}' \ R \ \overline{s}$. By definition of feedback, the transition $s \xrightarrow{i'|\pi_o'(o_1)} s_1$ occurs in $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$. On the contrary, there is no guarantee that the other transitions are preserved in $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$ except if the following condition holds:

$$\forall j, 2 \leq j \leq n, \exists i \in I, f(i, o_j) = i_j$$

In this case, we ensure that the transition $s_{j-1} \xrightarrow{\pi(i_j)|\pi_o(o_j)} s_j$ exists in $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$.

We will then say that $\mathcal{I}$ **preserves** the simulation R if for every $s'\ R\ s$ and every transition $s' \xrightarrow{i'|o'} \overline{s}'$ in $\mathcal{C}'$ such that $i' = \pi'_i(f'(i,o))$ and $o' = \pi'_o(o)$, there exists a path $s \xrightarrow{f'(i,o)|o_1} s_1 \xrightarrow{i_2|o_2} \ldots \xrightarrow{i_n|o} \overline{s}$ in $\mathcal{C}$ satisfying all the conditions of Definition 14 and the supplementary condition:

$$\forall j, 2 \leq j \leq n, \exists i \in I, f(i, o_j) = i_j$$

In the following, we will assume that, given $\mathcal{C} \rightsquigarrow \mathcal{C}'$ and a feedback interface $\mathcal{I}$ such that $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$ is defined, there always exists a simulation R preserved by $\mathcal{I}$.

Theorem 8 *(Horizontal composition) Let $op(\mathcal{C}_1, \ldots, \mathcal{C}_i, \ldots, \mathcal{C}_n)$ be a system. Let $\mathcal{C}_i \rightsquigarrow \mathcal{C}'_i$ (resp. $\mathcal{C}_i \bowtie \mathcal{C}'_i$). Then,*

$$op(\mathcal{C}_1, \ldots, \mathcal{C}_i, \ldots, \mathcal{C}_n) \rightsquigarrow \overline{op}(\mathcal{C}_1, \ldots, \mathcal{C}'_i, \ldots, \mathcal{C}_n)$$

(resp. $op(\mathcal{C}_1, \ldots, \mathcal{C}_i, \ldots, \mathcal{C}_n) \bowtie \overline{op}(\mathcal{C}_1, \ldots, \mathcal{C}'_i, \ldots, \mathcal{C}_n)$) where $\overline{op}$ is defined by structural induction on the complex operator op as follows:

- *if $op = \_$, then $\overline{op} = \_$;*
- *if $op = op_1 \otimes op_2$, then by definition op_1 and op_2 are respectively of arity $n_1 < n$ and $n_2 < n$. Let us suppose that $i \leq n_1$ (the case where $n_1 \leq i \leq n$ is handled similarly). Then, $\overline{op} = \overline{op}_1 \otimes op_2$;*
- *if $op = \circlearrowleft_{\mathcal{I}}(op')$, then $\overline{op} = \circlearrowleft_{\mathcal{I}'}(\overline{op}')$ where $\mathcal{I}' = (f', \pi'_i : I' \to \overline{I}', \pi_o : O' \to \overline{O}')$ is defined by:*
  - *$f' = f_{|_{I' \times O'}}$;*
  - *$\overline{I}' = \pi_i(I')$ and $\overline{O}' = \pi_i(O')$;*
  - *$\pi'_i = \pi_{i|_{I'}}$ and $\pi'_o = \pi_{o|_{O'}}$.*

Proof. This is proven by structural induction on the complex operator *op*. The basic case is obvious. The induction step is composed of two cases:

1. *op* is of the form $op_1 \otimes op_2$. By definition, op_1 and op_2 are respectively of arity $n_1 < n$ and $n_2 < n$. Let us suppose that $i \leq n_1$. By induction hypothesis we have $op_1(C_1, \ldots, C_i, \ldots, C_{n_1}) \rightsquigarrow \overline{op}_1(C_1, \ldots, C'_i, \ldots, C_{n_1})$ (resp. $op_1(C_1, \ldots, C_i, \ldots, C_{n_1}) \bowtie \overline{op}_1(C_1, \ldots, C'_i, \ldots, C_{n_1})$). This means

by definition there exists a simulation (resp. a bisimulation) $\overline{R}_1$ between $\overline{op}_1(C_1,\ldots,C'_i,\ldots,C_{n_1})$ and $op_1(C_1,\ldots,C_i,\ldots,C_{n_1})$. Let us set $R = \overline{R} \times Id_{S_2}$ where S_2 is the set of states of $op_2(C_{n_1+1},\ldots,C_n)$. It is obvious to show that Cartesian product is stable for simulation and bisimulation (according to Definition 14).

2. op is of the form $\circlearrowleft_{\mathcal{I}}(\overline{op}')$. By induction hypothesis, we have

$$op'(C_1,\ldots,C_i,\ldots,C_n) \rightsquigarrow \overline{op}'(C_1,\ldots,C'_i,\ldots,C_n)$$

(resp. $op(C_1,\ldots,C_i,\ldots,C_n) \bowtie \overline{op}'(C_1,\ldots,C'_i,\ldots,C_{n_1})$). This means by definition there exists a simulation (resp. a bisimulation) $\overline{R}$ preserved by $\mathcal{I}$ between $\overline{op}'(C_1,\ldots,C'_i,\ldots,C_{n_1})$ and $op'(C_1,\ldots,C_i,\ldots,C_{n_1})$. Then let us show that $\overline{R}$ remains a simulation (resp. a bisimulation) between $\circlearrowleft_{\mathcal{I}'}(\overline{op}')(C_1,\ldots,C'_i,\ldots,C_n)$ and $\circlearrowleft_{\mathcal{I}}(op')(C_1,\ldots,C_i,\ldots,C_n)$. Let us assume that $op'(C_1,\ldots,C_i,\ldots,C_{n_1})$ and $\overline{op}'(C_1,\ldots,C'_i,\ldots,C_{n_1})$ are over $H = T(O \times \_)^I$ and $H' = T(O' \times \_)^{I'}$, $\mathcal{I} = (f : I \times O \to I, \pi_i : I \to \overline{I}, \pi_o : O \to \overline{O})$, and then $\mathcal{I}' = (f' : I' \times O' \to I', \pi'_i : I' \to \overline{I}', \pi'_o : O' \to \overline{O}')$. Moreover, let us assume that $op'(C_1,\ldots,C_i,\ldots,C_{n_1}) = (S, init, \alpha)$ and $\overline{op}'(C_1,\ldots,C'_i,\ldots,C_{n_1}) = (S', init', \alpha')$. By definition, $\circlearrowleft_{\mathcal{I}'}(\overline{op}')(C_1,\ldots,C'_i,\ldots,C_n) = (S', init', \overline{\alpha}')$ and $\circlearrowleft_{\mathcal{I}}(op')(C_1,\ldots,C_i,\ldots,C_n) = (S, init, \overline{\alpha})$ where $\overline{\alpha}'$ and $\overline{\alpha}$ are defined following Definition 9. Let us suppose $s' \in S'$ and $s \in S$ such that $s' \ \overline{R} \ s$. Let $i' \in \overline{I}'$ and let $(o', \overline{s}') \in \eta'_{\overline{O}' \times S'}(\overline{\alpha}'(s')(i'))$. By definition of feedback, there exists $i \in I'$ and $o \in O'$ such that $(o, \overline{s}') \in \eta'_{O' \times S'}(\alpha'(s')(f'(i,o)))$,$\pi'_i(i) = i'$ and $\pi'_o(o) = o'$. By definition of simulation, there exists an execution in $\overline{op}'(C_1,\ldots,C'_i,\ldots,C_n)$ of the form:

$$s \xrightarrow{f'(i,o)|o_1} s_1 \xrightarrow{i_2|o_2} \ldots \xrightarrow{i_n|o} \overline{s}$$

with $\overline{s}' \ \overline{R} \ \overline{s}$. By the condition that $\overline{R}$ is preserved by $\mathcal{I}$, we have in $\circlearrowleft_{\mathcal{I}}(op')(C_1,\ldots,C_i,\ldots,C_n)$ the execution:

$$s \xrightarrow{i'|\pi_o(o_1)} s_1 \xrightarrow{\pi_i(i_2)|\pi_o(o_2)} \ldots \xrightarrow{\pi_i(i_n)|\pi_o(o)} \overline{s}$$

□

Vertical composition. Vertical composition is just a consequence of the following simple result.

Theorem 9 *Both* $\rightsquigarrow$ *and* $\bowtie$ *are transitive, i.e.* $\rightsquigarrow \cdot \rightsquigarrow \subseteq \rightsquigarrow$ *and* $\bowtie \cdot \bowtie \subseteq \bowtie$.

Proof. Both $\rightsquigarrow$ and $\bowtie$ are defined w.r.t. revisited similarity and bisimilarity which it is not difficult to show they are transitive relations. □

Horizontal and vertical composition can be easily composed to obtain a bidimensional compositionality.

7 Conclusion

This paper introduced a logic defined as a variant of first-order fixed-point modal logic to express component and system requirements and an abstraction operator to build systems and check their correctness incrementally. For this logic, we proposed conditions to preserve properties expressed in this logic along integration and abstraction, and then showed a means to establish correct-by-construction proofs. The interest of our results is they are completely independent of integration operators. Furthermore, they have been shown to a large family of properties containing at least all the common properties that can be expressed on state-based components such as deadlock free, reachability, etc.

7.1 Perspectives

Both logic and associated results that have been presented here are devoted to discrete/computing complex systems. We are currently working to extend this work to heterogeneous complex systems (i.e. where components can be defined over discrete or continuous time scales). To do so, first we propose to introduce the notion of monad to components in [2] to take into account different computation situations, and then to study the results of properties preservation for the logic defined in [3]. Thus, the defined formalism would be allowed to be used as a formal semantics for the system modelling language SysML.
Moreover, the logic presented in the paper can be related to the language for the power set functor based on regular expressions defined in [42], though the approach followed in [42] differs from ours at least in the following two points:

1. Authors in [42] are interested in defining a "process algebra" like language instead of a modal logic to specify system behaviors;

2. They are also interested in giving a sound and complete axiomatization thereof whereas here, we are mainly interested in giving results of properties preservation along both integration and abstraction operators to get correctness-by-construction results.

However, in future work, when we will look at the definition of an inference system for our logic to conduct correctness proofs and check their feasibility, some connections will be to make with the work set out in [42]. Finally, following the works in [10], we also propose to study computational aspects of our formalism such as synthesis of components to transform requirements into components that satisfy them and the definition of model-checking algorithms. Of course, as already said in the introduction, the logic will be allowed to be restricted to the propositional case. Within the formalism in [2, 3], particular attention should be given to time scale mainly when dealing with continuous times. Indeed, although continuous time scales in [2, 3] are discretely defined and then (non-standard) induction works, their cardinality is not denumerable which is not to allow their computability. In a series of papers, Y. Sergueyev has recently defined a positional numeral system that may allow us to carry out effective computation with infinitesimal and infinitely large numbers [39, 40]. We then propose to study how to introduce the ideas developed in [39, 40] within the formalism developed in [2, 3], with defining algorithms issues in mind both for the synthesis and properties satisfaction in the presence of complex heterogeneous (discrete and continuous) systems.

References

[1] M. Aiguier, F. Boulanger, and B. Kanso. A formal abstract framework for modelling and testing complex software systems. *Theoretical Computer Science*, 455:66–97, 2012.

[2] M. Aiguier, B. Golden, and D. Krob. Modeling of complex systems II: A minimalist and unified semantics for heterogeneous integrated systems. *Applied Mathematics and Computation*, 218(16):8039–8055, 2012.

[3] M. Aiguier, B. Golden, and D. Krob. An adequate logic for heterogeneous systems. In *18th International Conference on Engineering of*

Complex Computer (ICECCS13), pages 65–74. IEEE Computer Society, 2013.

[4] E. Alesken and R. Belcher. *Systems Engineering.* Prentice Hall, 1992.

[5] A. Arnold and D. Niwinski. *The μ-calculus over power set algebras*, chapter Rudiments of μ-calculus, pages 141–153. Elsevier, 2001.

[6] L.S Barbosa. Components as processes: An exercise in coalgebraic modeling. In S. F. Smith and C. L. Talcott, editors, *Fourth International Conference on Formal Methods for Open Object-Oriented Distributed Systems (FMOODS00)*, volume 177 of *IFIP Conference Proceedings*, pages 397–417. Kluwer, 2000.

[7] L.S Barbosa. Towards a calculus of state-based software components. *Journal of Universal Computer Science*, 9(8):891–909, August 2003.

[8] M. Barr and C. Wells, editors. *Category theory for computing science, 2nd ed.* Prentice Hall International Ltd., Hertfordshire, UK, 1995.

[9] B.-S. Blanchard and W.-J. Fabrycky. *Systems engineering and analysis.* Prentice Hall, 1998.

[10] M. Bonsangue, J. M. M. Rutten, and A. Silva. Coalgebraic logic and synthesis of mealy machines. In R.M. Amadio, editor, *11th International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*, volume 4962 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 2008.

[11] J. Bradfield and C. Stirling. Modal mu-calculi. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, pages 721–756. Elsevier, 2007.

[12] D. Cha, J. Rosenberg, and C. Dym. *Fundmantals of Modeling and Analysing Engineering Systems.* Cambridge University Press, 2000.

[13] H. Ehrig and H. Kreowski. *Algebraic Foundations of Systems Specification, IFIP State-of-the-Art Reports*, chapter Refinement and implementation, pages 201–243. Springer-Verlag, 1999.

[14] J. L. Fiadeiro. *Categories for Software Engineering.* SpringerVerlag, 2004.

[15] Ichiro H., B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 4:11, 2007.

[16] H. H. Hansen, D. Costa, and J. J. M. M. Rutten. Synthesis of mealy machines using derivatives. *Electr. Notes Theor. Comput. Sci. (ENTCS)*, 164(1):27–45, 2006.

[17] I. Hasuo, C. Heunen, B.Jacobs, and A. Sokolova. Coalgebraic components in a many-sorted microcosm. In Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors, *Third International Conference on Algebra and Coalgebra in Computer Science (CALCO09)*, volume 5728 of *Lecture Notes in Computer Science*, pages 64–80. Springer, 2009.

[18] I. Hasuo, B. Jacobs, and A. Sokolova. The microcosm principle and concurrency in coalgebra. In Roberto M. Amadio, editor, *11th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS08)*, volume 4962 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2008.

[19] T.A. Henzinger and J.Sifakis. The discipline of embedded systems design. *IEEE Computer Society*, 40(10):32–40, 2007.

[20] C.-A.-R. Hoare. Proof of correctness of data representations. *Acta Informaticae*, 1:271–281, 1972.

[21] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1985.

[22] J. Hughes and B. Jacobs. Simulations in coalgebra. *Theoretical Computer Science*, 327(1-2):71–108, 2004.

[23] B. Jacobs and J. Rutten. A tutorial on coalgebras and coinduction. *EATCS Bulletin*, 62:222-259, 1997.

[24] B. Kanso. *Modeling and testing of component-based systems.* PhD thesis, École Centrale Paris, 2011.

[25] B. Kanso, M. Aiguier, F. Boulanger, and C. Gaston. Testing of component-based systems. In *19th Asia-Pacific Software Engineering Conference (APSEC 2012)*. IEEE Computer Society, 2012.

[26] R. Kashima and K. Okamoto. General models and completeness of first-order modal μ-calculus. *Journal of Logic and Computation*, 18(4):497–507, 2008.

[27] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[28] A. Kurz. Coalgebras and modal logics. *Notes of lectures given at ESSLLI'01*, October 2001.

[29] E.A. Lee and S.A. Seshia. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. Lee and Seshia, 2010.

[30] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, New York, Heidelberg, Berlin, 1971.

[31] M.-W. Maier and E. Rechtin. *The art of system architecturing*. CRC Press, 2002.

[32] S. Meng and L.S. Barbosa. Components as coalgebras: the refinement dimension. *Theoretical Computer Science*, 351(2):276–294, 2006.

[33] R. Milner. A calculus of communicating systems. *Springer-Verlag, New York, USA*, 1982.

[34] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[35] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[36] D. Pattinson. Semantical Principles in the Modal Logic of Coalgebras. In H. Reichel and A. Ferreira, editors, *Proc. 18th Symposium on Theoretical Aspects of Computer Science (STACS 2001)*, volume 2010 of *Lecture Notes in Computer Science*, pages 514–526, 2001.

[37] J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, October 2000.

[38] A.-P. Sage and J.-E. Amstrong. *Introduction to system engineering*. John Wiley, 2000.

[39] Ya.-D. Sergeyev. A new applied approach for executing computations with infinite and infinitesimal quantities. *Informatica*, 19(4):567–594, 2008.

[40] Ya.-D. Sergeyev. Numerical computations and mathematical modelling with infinite and infinitesimal numbers. *Applied Mathematics and Computing*, 29:177–195, 2009.

[41] J. Sifakis. A framework for component-based construction. In *3rd IEEE International Conference on Software Engineering and Formal Methods (SEFM05)*, pages 293–300. IEEE Computer Society, 2005.

[42] Alexandra Silva, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Non-deterministic kleene coalgebras. *Logical Methods in Computer Science*, 6(3), 2010.

[43] E. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, volume 6 of *Textbooks in Applied Mathematics*. Springer-Verlag, 1998.

[44] W.-C. Turner, J.-H. Mize, K.-E. Case, and J.-W. Nazemeth. *Introduction to industrial and systems engineering*. Prentice Hall, 1993.

[45] M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In Ph. Gardner and N. Yoshida, editors, *15th International Conference on Concurrency Theory (CONCUR04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 512–528. Springer-Verlag, 2004.