Testing of component-based systems

Bilal Kanso
École Centrale Paris
Grande Voie des Vignes
F-92295 Châtenay-Malabry
bilal.kanso@ecp.fr

Marc Aiguier
École Centrale Paris
Grande Voie des Vignes
F-92295 Châtenay-Malabry
marc.aiguier@ecp.fr

Fréderic Boulanger
Supelec E3S
3 rue Joliot-Curie
F-91192 Gif-sur-Yvette cedex
frederic.boulanger@supelec.fr

Christophe Gaston
CEA LIST Saclay
F-91191 Gif-sur-Yvette cedex
christophe.gaston@cea.fr

***Abstract*—In this paper, we pursue our works on generic modeling and conformance testing of component-based systems. Here, we extend our theory of conformance testing to the testing of component-based systems. We first show that testing a global system can be done by testing its components thanks to the projection of global behaviors onto local ones. Secondly, based on our projection techniques, we define a framework to build adequate test purposes automatically for testing components in the context of the global system where they are plugged in. The underlying idea is to identify from any trace *tr* of the global system, the trace of any component involved in *tr*. Those projected traces can be then seen as test cases that should be tested on individual components.**

Keywords: Component-based system, Conformance testing, Compositional testing, Testing in context, Projection, Test purpose.

INTRODUCTION

In the last decades, the component-based software approach [1], [2] has emerged due to the great advantages it offers: modularity, re-usability, cost-effective solution. Components are then designed, developed and validated in order to be widely used, while complex software systems are described recursively, at a higher level of abstraction, as interconnections of those components. Hence, each sub-system (or component) can be either a complex system itself or a simple component, elementary enough to be handled without further decomposition. Composition is used for fitting different components together and then defining larger systems. Such a composition is defined by operations which take components as well as the nature of their interactions to provide a description of a new and more complex component or system.

In [3], we proposed a formal framework for modeling basic components viewed as abstract state-based systems. Components were then modeled as coalgebras over **sets**-endofunctor with monads [4], [5] following *Barbosa*'s component definition [6], [7]. Monads enabled us to generically consider a wide range of computation structures such as partiality, non-determinism, etc. [5], and then to define components independently of any computation structure. This definition allowed us to unify in a same framework a large family of state-based formalisms such as Mealy automata [9], [8], Labeled Transition Systems [10], Input-Output Labeled Transition Systems [14], [11], etc. Larger systems are then built by integrating components from integration operators defined by composition of two basic ones: Cartesian product and feedback. In [3], we showed that most standard integration operators such as sequential and concurrent composition or synchronous product are subsumed by our generic definition of integration operators. Based on this framework, a conformance testing theory has been defined in [3].

The "plug and play" nature of component-based system design leads naturally to build always bigger systems whose correctness happens to be more and more difficult to assert. This is of course due to the fact that analyzing big systems generates state and time explosion problems, but it may also be caused by the system architecture (e.g. distributed system) which may complicate the ability to instrument the system in order to observe behaviors to be analyzed. Even more, if a "faulty" behavior is observed in such a system, the size of the system is a problem to identify the cause of the fault at the debugging phase.

All these reasons call to find ways to make system validation modular. Such methods enable to analyze a system, subsystems per subsystems, in a modular way, rather than "as a whole". Analyzed such systems are smaller (less prone to generate explosion problems), more observable and controllable (thus their behaviors are easier to cover), and debugging is greatly facilitated.

Compositional testing [15], [20], [22] is viewed as one of the most promising directions to bridge the gap between the increasing complexity of systems and actual testing method limits due to the reasons discussed above. Similarly to compositinality result in [20] establishing under certain hypothesis that the conformance testing relation $ioco$ is compositional with respect to parallel composition and hiding, we have established a compositinality result in [3]. This result expresses that for the conformance relation $ioco$[1] and n implementations and specifications iut_i and spec_i, $1 \leq i \leq n$, each one modeled by a component as defined in [3], if for each i, $1 \leq i \leq n$, iut_i $ioco$ spec_i, then for any integration operator of arity n (see Definition 1.7), $op(\mathsf{iut}_1, \ldots, \mathsf{iut}_n)$ $ioco$ $op(\mathsf{spec}_1, \ldots, \mathsf{spec}_n)$. The compositinality result obtained in [3] is thus an extension of *Tretmans*'s result [20] since it is established independently of a given integration operator.

This result justifies the approach that consists in testing separately the components of a system in order to build the

[1]Actually, a slight extension of this relation to our components called $cioco$ in [3] (see Definition 2.1 in this paper).

correctness of the global system. However, it turns out that in practice, such an integration theory is not enough. Such a result does not help to choose test purposes that are meaningful. Indeed each $\mathsf{iut}_{i_{(i\leq n)}}$ is tested with respect to its specification $\mathsf{spec}_{i_{(i\leq n)}}$, but since testing means selecting a finite number of executions (test cases) to evaluate the conformance, the question is then how to build a meaningful set of executions? Following approaches in [20] and [3] which are dedicated to model-based testing, we propose to extract test cases from specification. However, spec_i, standing alone, does not contain enough explanation to know how iut_i will be used in the context of the whole system. This usage is in the end the only aspect that matters at test selection phases since all behaviors reflecting a non-conformance between iut_i and spec_i which are never activated in the context of the whole system $op(\mathsf{iut}_1, \ldots, \mathsf{iut}_n)$, will by definition never cause a fault at the system level. For example, if a system uses a calculator component to invoke only addition, then the component may well be "faulty" for multiplication; this will not cause a fault at the system level. Even more, wasting time to test such behaviors reduces the time and resources to test behaviors of the component that will be activated in the frame of the system. This may have dramatically harmful consequences. For example, the disaster of *Ariane* 5 in 1996 is caused by the absence of testing in context of a software component which was only tested for *Ariane* 4. We will give in this paper, a new compositinality result that will take into account the behavior of global system in which components are plugged in. This last result is inspired from the approach proposed in [15], initially developed in the setting of $IOSTS$ (symbolic automaton). In [15], only projection is defined, but no compositinality result is given.

Based on this result, we will then propose a technique that strengthens testing of each component involved in a global system, by choosing suitable test purposes for them. This will be done by defining a projection mechanism that, from global behaviors of a system, will help generating test purposes capturing the behaviors of the sub-systems, that typically occur in the context of the whole system.

The paper is structured as follows: Section I recalls our framework for modeling components and systems. Section II introduces the conformance testing theory and discusses its main limitation for the validation of complex software systems. Section III presents the compositinality result and shows how components can be tested while taking the system to which they belong into account.

I. Components and systems

A. Components

In [3], a component is defined as a generalized Mealy automaton in which the dependence between outputs and both current state and inputs is relaxed from a strict deterministic, to encompass more complex behaviors such as partiality, non determinism, etc. Components are defined using terminology and notations of coalgebras [24] and monads [4]. Hence, a component in [3] is a coalgebra (S, α) over a signature $T(O \times \_)^I : \mathbf{Set} \longrightarrow \mathbf{Set}$ where T is a monad. The monads have been introduced because they allow us to generically consider many computation situations such as determinism, non-determinism, partiality, etc. (see [3], [5] for more explanations).

Here, to make easier the readability of the paper, we restrict ourself to a particular case when T stands for the powerset monad $\mathcal{P}$. The generalization to any monad T does not raise any difficulties.

Definition 1.1 (Component): Let I and O be two sets denoting, respectively, the input and output domains. A **component** $\mathcal{C}$ over (I, O) is a triplet $(S, init, \alpha)$ where:

- S is the set of states of $\mathcal{C}$;
- $init \in S$ is a distinguished element denoting the initial state of $\mathcal{C}$;
- $\alpha : S \times I \longrightarrow \mathcal{P}(O \times S)$ is the transition function.

Example 1.1: To illustrate our approach, we will consider in this paper a simple system $\mathcal{S}$ that computes grade averages presented in Figure 1. This system $\mathcal{S}$ is built from two basic components: a "graphical interface" that helps the user to make various operations on grades and a "calculator" that receives operation commands from the user, performs the requested operation, and reports back to the user.

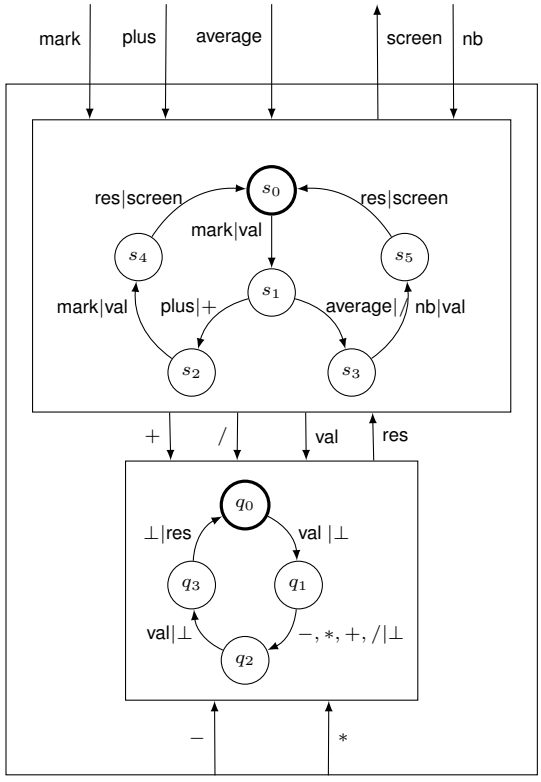


Fig. 1: Grade averages system as a composition of the graphical interface and the calculator

In our framework, the graphical interface is modeled as the component $\mathcal{G} = (\{s_0, s_1, s_2, s_3, s_4, s_5\}, s_0, \alpha_1)$ over the

signature

$$\Sigma_1 = (\{\text{mark}, \text{plus}, \text{average}, \text{nb}, \text{res}\}, \{+, /, \text{screen}, \text{val}\})$$

and the calculator as the component $\mathcal{C} = (\{q_0, q_1, q_2, q_3\}, q_0, \alpha_2)$ over the signature

$$\Sigma_2 = (\{+, *, -, /\}, \{\text{res}, \perp\})$$

α_1 (resp. α_2) is depicted in the box at the top side (resp. bottom side) of Figure 1.

The semantics of a component is characterized by the set of finite sequences of couples (input|output), that is illustrated by the following definition:

Definition 1.2 (Component finite traces): The **finite trace** from a state s of a component $\mathcal{C}$, noted $\mathbf{Trace}_{\mathcal{C}}(\mathbf{s})$, is the whole set of the finite input-output sequences $\langle i_0|o_0, \ldots, i_n|o_n\rangle$ such that there exists a finite sequence $(s_0, \ldots, s_{n+1}) \in S^*$ of states where for every $j, 0 \leq j \leq n, (o_j, s_{j+1}) \in \alpha(s_j)(i_j)$ with $s_0 = s$.

Hence, **the set of traces of** $\mathcal{C}$, noted $Trace(\mathcal{C})$, is the set $Trace_{\mathcal{C}}(init)$.

In the following, we note $\alpha(s)(i)_{|_1}$ (*resp.* $\alpha(s)(i)_{|_2}$) the set composed of all first arguments (*resp.* second arguments) of couples in $\alpha(s)(i)$.

B. Systems

Larger systems are built by composition from two basic 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 1.3 (Cartesian product $\otimes$): Let $\mathcal{C}_1 = (S_1, init_1, \alpha_1)$ and $\mathcal{C}_2 = (S_2, init_2, \alpha_2)$ be two components over (I_1, O_1) and (I_2, O_2) respectively. $\mathcal{C}_1 \otimes \mathcal{C}_2$, the **cartesian product** of $\mathcal{C}_1$ and $\mathcal{C}_2$, is the component $(S_1 \times S_2, (init_1, init_2), \alpha)$ over $(I_1 \times I_2) \times (O_1 \times O_2)$ where α is the mapping defined for every $(i_1, i_2) \in I_1 \times I_2$ and every $(s_1, s_2) \in S$ by:
$\alpha((s_1, s_2))((i_1, i_2)) = \big\{((o_1, o_2), (s'_1, s'_2))|(o_k, s'_k) \in \alpha(s_k)(i_k)$ for $k = 1, 2\big\}$

Feedback: The concept of feedback composition is intrinsic in dynamic system modeling in control theory [16], [17]. Here, we fit it to discrete systems. A component with *feedback* has directed cycles, where an output from a component is fed back to affect an input of the same component. 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.

First, we introduce feedback interfaces for defining correspondences between outputs and inputs of components and only keeping both inputs and the outputs that are not involved in feedback.

Definition 1.4 (Feedback interface): A **feedback interface** over an interface signature (I, O) is a triplet $\mathcal{I} = (f, \pi_i, \pi_o)$ 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. It finds the new value of the input that it is both a valid input and a valid output of the component, given its current state. Both mappings π_i and π_o can be thought as extensions of the hiding connective found in process calculi [19].

The feedback operator[2] we consider here is *synchronous*. That means the reaction of a system takes no observable time [18] and its outputs are produced synchronously with its inputs. More precisely, at some reaction r, the output of component $\mathcal{C}$ in r must be available to its inputs in the same reaction r. The synchronous feedback requires then the existence of an instantaneous fix-point (i.e. defined at the same time and not deferred of one unit). This gives rise to the notion of *well-formed feedback interface*.

Definition 1.5 (Well-formed feedback interface): Let $\mathcal{C}$ be a component over $\Sigma = (I, O)$ and $\mathcal{I} = (f, \pi_i, \pi_o)$ be a feedback interface over Σ. We say that $\mathcal{I}$ is **well-formed w.r.t** $\mathcal{C}$ if, and only if for every state $s \in S$ and every sequence of inputs $x_1, \ldots, x_n$, there exists a sequence of outputs $y_1, \ldots, y_n$ such that for every $j, 1 \leq j < n,\ y_j \in \alpha(s)(f(x_j, y_j))_{|_1}$.

We want to build a component that hides the feedback of a component $\mathcal{C}$. As one can see in Figure 2, the feedback component $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$ is defined over the signature (I', O'). The

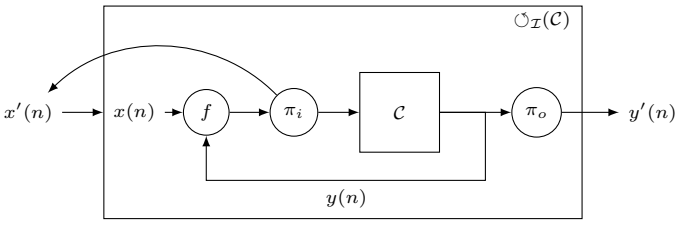


Fig. 2: Feedback composite: $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$

outputs are then hidden from any state s that are fed back as inputs to s. The result is a component with input and output sets I' and O' respectively. This is done by means of the feedback interface $\mathcal{I} = (f, \pi_i, \pi_o)$. Let us suppose that the current state of $\mathcal{C}$ at the n^{th} reaction is $s_n \in S$ and the current external input is $x(n) \in I$, then let us compute both new input $x'(n) \in I'$ and output $y'(n) \in O'$ when $\mathcal{C}$ is triggered by $x(n)$. First, by f, we compute the input $\bar{x}(n) = f(x(n), y(n))$. Then, $\bar{x}(n)$ becomes the new input of $\mathcal{C}$. Indeed, component $\mathcal{C}$ reacts by updating its state to s_{n+1} and producing an output $y(n)$ (such a $y(n)$ exists since $\mathcal{I}$ is well-formed w.r.t $\mathcal{C}$). Second, by means of π_i and π_o, we hide both input and output involved in the feedback, and then produce the input $x'(n) = \pi_i(x(n))$ and the output $y'(n) = \pi_o(y(n))$ of the feedback component $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$.

[2]There is another kind of feedback called relaxed feedback. Interested readers may refer to [3].

Definition 1.6 (Synchronous feedback $\circlearrowleft$): Let $\mathcal{I} = (f, \pi_i, \pi_o)$ be a feedback interface over $\Sigma = (I, O)$. Let $\mathcal{C} = (S, init, \alpha)$ be a component over Σ such that $\mathcal{I}$ is well-formed w.r.t $\mathcal{C}$. $\circlearrowleft_{\mathcal{I}}(\mathcal{C})$, the **synchronous feedback over** $\mathcal{I}$, is the component $\mathcal{C}' = (S, init, \alpha')$ over $\Sigma' = (I', O')$ where α' the mapping defined for every $s \in S$ and every $i' \in I'$ by: $\alpha'(s)(i') = \big\{(o', s') | \exists (i, o) \in (I \times O), (o, s') \in \alpha(s)(f(i, o)), \pi_i(i) = i' \ and \ \pi_o(o) = o'\big\}$

Complex operators and systems:
As previously explained, from Cartesian product and feedback operators, we can build more complex ones by composition.

Definition 1.7 (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.

In Example 1.2, as an example of a complex operator, we show how the sequential operator can be defined in our framework.

Example 1.2: The *sequential composition* $\triangleright$ of two components $\mathcal{C}_1$ and $\mathcal{C}_2$ corresponds to a composition where both components $\mathcal{C}_1$ and $\mathcal{C}_2$ are interconnected side-by-side and the output of one is the input of the other. This kind of composition can be naturally defined in our framework as follows:

$$\triangleright((\mathcal{C}_1, \mathcal{C}_2)) = \circlearrowleft_{\mathcal{I}}((\mathcal{C}_1 \otimes \mathcal{C}_2))$$

where $\mathcal{I} = (f, \pi_i, \pi_o)$ is the feedback interface defined $\forall (i, i') \in I_1 \times I_2, \forall (o, o') \in O_1 \times O_2$ by:

$$f((i, i'), (o, o')) = (i, o), \pi_i((i, i')) = i \ and \ \pi_o((o, o')) = o'$$

Other standard operators have been also defined similarly in [3]

Complex operators will not be necessarily defined when 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, from a sequence of components, when it is defined.

Definition 1.8 (Systems): Let $\mathbb{C}$ be a set of components. The **set of systems over** $\mathbb{C}$ is inductively defined as follows:

- for any $\mathcal{C} \in \mathbb{C}$, a component over a signature Σ, $\_(\mathcal{C}) = \mathcal{C}$ is a system over the signature Σ 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 $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{n_1}, \mathcal{C}_{n_1+1}, \ldots, \mathcal{C}_n)$ of components in $\mathbb{C}$ with each $\mathcal{C}_i$ over $\Sigma_i = (I_i, O_i)$, if both op_1 and op_2 are defined for $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{n_1}$ and $\mathcal{C}_{n_1+1}, \ldots, \mathcal{C}_n$ respectively, then $op_1 \otimes op_2(\mathcal{C}_1, \ldots, \mathcal{C}_n) = op_1(\mathcal{C}_1, \ldots, \mathcal{C}_{n_1}) \otimes op_2(\mathcal{C}_{n_1+1}, \ldots, \mathcal{C}_n)$ is a system over $\Sigma = (\prod_{i=1}^{n} I_i, \prod_{i=1}^{n} O_i)$ 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** $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$;
- 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 in $\mathbb{C}$, 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 Σ, $\mathcal{I}$ is a feedback interface over Σ and $\mathcal{I}$ is well-formed w.r.t $\mathcal{S}$, then $\circlearrowleft_{\mathcal{I}}(op)(\mathcal{C}_1, \ldots, \mathcal{C}_n) = \circlearrowleft_{\mathcal{I}}(\mathcal{S})$ is a system over Σ' and[3] $\circlearrowleft_{\mathcal{I}}(op)$ is **defined for** $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$, else $\circlearrowleft_{\mathcal{I}}(op)$ is **undefined for** $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$.

We introduce the definition of a sub-system involved in a given system. This intuitively allows us to characterize the set of all basic sub-systems from which the global system can be built.

Definition 1.9 (Sub-systems): Let $\mathcal{S} = op(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ be a system over a signature Σ. The **set of sub-systems of** $\mathcal{S}$, noted $\mathbb{S}ub(\mathcal{S})$, is inductively defined on the structure of *op* as follows:

- if $op = \_$, then $\mathbb{S}ub(\mathcal{S}) = \{\mathcal{S}\}$;
- if $op = op_1 \otimes op_2$ with op_1 and op_2 of arity n_1 and n_2 respectively (i.e. $n = n_1 + n_2$), then $\mathbb{S}ub(\mathcal{S}) = \{\mathcal{S}\} \cup \mathbb{S}ub(op_1(\mathcal{C}_1, \ldots, \mathcal{C}_{n_1})) \cup \mathbb{S}ub(op_2(\mathcal{C}_{n_1+1}, \ldots, \mathcal{C}_n))$;
- if $op = \circlearrowleft_{\mathcal{I}}(op')$, then $\mathbb{S}ub(\mathcal{S}) = \{\mathcal{S}\} \cup \mathbb{S}ub(op'(\mathcal{C}_1, \ldots, \mathcal{C}_n))$.

Example 1.3: The system $\mathcal{S}$ to compute grade averages is obtained as a composition of $\mathcal{G}$ and $\mathcal{C}$ using our basic integration operators. Hence to define the system $\mathcal{S}$, we first apply the Cartesian product $\otimes((\mathcal{G}, \mathcal{C}))$ to $\mathcal{G}$ and $\mathcal{C}$ over the signature $\Sigma_{\otimes} = (I_{\otimes}, O_{\otimes})$ with: $I_{\otimes} = (\{\text{mark}, \text{plus}, \text{average}, \text{nb}\} \times \{\text{val}, +, /\})$ and $O_{\otimes} = (\{\text{val}, \text{screen}, \perp\} \times \{\perp, \text{res}\})$. We can then see that:

- both outputs + and / of $\mathcal{G}$ are returned as inputs of $\mathcal{C}$;
- the output "res" of $\mathcal{C}$ is returned as input of $\mathcal{G}$.

Then, we apply the synchronous feedback to $\otimes((\mathcal{G}, \mathcal{C}))$. This leads to the operator $\circlearrowleft_{\mathcal{I}}$ over the interface signature $\mathcal{I} = (f, \pi_i, \pi_o)$ as follows:

$$f: \ I_{\otimes} \times O_{\otimes} \longrightarrow I_{\otimes}$$
$$((i, i'), (o, o')) \mapsto \begin{cases} (i, o) & if \ i' = o \\ (i, i') & otherwise \end{cases}$$

$$\pi_i: \ I_{\otimes} \longrightarrow I_{\mathcal{G}} \cup I_{\mathcal{C}}$$
$$(i, i') \mapsto \begin{cases} i & if \ i' \in O_{\mathcal{C}} \\ i' & otherwise \end{cases}$$

$$\pi_o: \ O_{\otimes} \longrightarrow O_{\mathcal{G}} \cup O_{\mathcal{C}}$$
$$(o, o') \mapsto \begin{cases} o' & if \ o \in I_{\mathcal{G}} \\ o & otherwise \end{cases}$$

[3] Σ' is the signature of the synchronous feedback.

Applying $\circlearrowleft_{\mathcal{I}}$ to $\otimes((\mathcal{G},\mathcal{C}))$ leads to a new component $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G},\mathcal{C}))$ (see Figure 3) where all outputs of $\mathcal{G}$ (i.e $+, /$ and val) that are fed back to $\mathcal{C}$ and the output "res" of $\mathcal{G}$ that is fed back to $\mathcal{G}$ are hidden (i.e. synchronized).

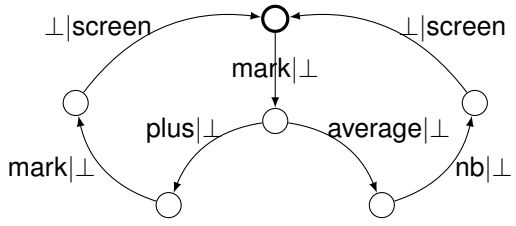


Fig. 3: Component $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G},\mathcal{C}))$

II. Conformance testing

Conformance testing theory is usually based on the comparison between the behavior of a specification and an implementation using a conformance relation. The goal of this relation is to specify what the conformance of an implementation is with respect to its specification. It has been shown that the input-output conformance relation *cioco* is the most suitable for testing our components [3]. This relation distinguishes input and outputs actions, and requires that the implementation behaves according to a specification, but also allows behaviors on which the specification puts no constraint.

The specification spec of a component is the formal description of its behavior given by a component over a signature (I, O). On the contrary, its implementation iut is an executable component, which is considered as a black box [25], [26]. We interact with the implementation through its interface, by providing inputs to stimulate it and observing its behavior through its outputs. Hence, to be able to treat the implementation iut, we make the following two assumptions about it:

- The implementation iut can be modeled as a component $(S, init, \alpha)$ over the signature (I', O') with $I \subseteq I'$ to allow the implementation to accept all the inputs of the[4] specification and $O' \subseteq O$ to allow the specification to accept all the responses of the implementation.
- iut is **input-enabled**, i.e. at any state, it must produce answers for all inputs provided by the environment: $\forall(s,i) \in S \times I, \exists(o,s') \in O \times S$ such that $(o,s') \in \alpha(s)(i)$

The conformance relation that we will call here *cioco*[5] is a slight adaptation of the standard relation $ioco$ [11].

Definition 2.1: (cioco) Let $\mathsf{spec}, \mathsf{iut}$ be two components over (I,O) and (I',O') respectively such that $I \subseteq I', O' \subseteq O$ and iut is input-enabled. iut **is in conformance with** spec, noted iut *cioco* spec, if and only if

$$\forall tr \in Trace(\mathsf{spec}), \forall i \in I,$$
$$Out(\mathsf{iut} \text{ after } (tr,i)) \subseteq Out(\mathsf{spec} \text{ after } (tr,i))$$

[4] I and O are the input and output sets of the specification respectively.
[5] c for component

where for any component $\mathcal{C}$, any finite trace tr, and any input i of $\mathcal{C}$, $Out(\mathcal{C} \text{ after } (tr,i))$ is the set

$$\{o \mid tr.\langle i|o\rangle \in Trace(\mathcal{C})\}$$

When the $Out(\mathsf{spec} \text{ after } (tr,i))$ is empty, that ensures the quiescence notion introduced by *Tretmans* in [13].

Similarly to [20], we studied in [3] compositionality properties for *cioco* over integration operators defined in Section I-B. We then proved the following theorem:

Theorem 2.1 (Compositionality [3]): Let op be a complex operator of arity n. Let $\mathsf{iut}_1, \ldots, \mathsf{iut}_n, \mathsf{spec}_1, \ldots, \mathsf{spec}_n$ be input-enabled components such that $\forall i, 1 \leq i \leq n$, iut_i *cioco* spec_i, then one has $op(\mathsf{iut}_1, \ldots, \mathsf{iut}_n)$ *cioco* $op(\mathsf{spec}_1, \ldots, \mathsf{spec}_n)$.

That means if single components of a system conform to their specifications, the whole system built over our integration operators is in accordance with its specification, unless the specification model is input-enabled. Such a testing compositionality result theory provides a way to test the integrated system only by testing its sub-systems i.e. there is no need to re-test its conformance correction. Hence, once this property is verified, the correctness of the integrated system is obtained from the correctness of the individual components. To test the integrated system, it is not necessary to consider it as a whole, but it is enough to consider its sub-systems and test them separately. Indeed, the contraposition of this property is the following:

$$\neg\Big(op(\mathsf{iut}_1, \ldots, \mathsf{iut}_n) \textit{ cioco } op(\mathsf{spec}_1, \ldots, \mathsf{spec}_n)\Big) \Longrightarrow$$
$$\exists i, 1 \leq i \leq n, \neg(\mathsf{iut}_i \textit{ cioco } \mathsf{spec}_i)$$

Thus, by looking at this new property, we can easily see that non-correctness of the integrated system under test $op(\mathsf{iut}_1, \ldots, \mathsf{iut}_n)$ implies that at least one of its components $\mathsf{iut}_1, \ldots, \mathsf{iut}_n$ is incorrect. In other words, that means to test $op(\mathsf{iut}_1, \ldots, \mathsf{iut}_n)$, it suffices to test $\mathsf{iut}_1, \ldots, \mathsf{iut}_n$ in isolation.

In the sequel, we will show how to improve significantly the result obtained in Theorem 2.1 by taking into account the global system in which components are plug in. This will be achieved by using projection mechanisms.

III. Projection and test purposes

A. Projection and compositionality

Projection techniques [15] are defined by pruning from any global behavior p, all that does not concern the sub-system that we want to test. This will allows us to generate more relevant unit test cases to test individual components. As an illustration, let us again consider the system that computes grade averages (see Example 1.3). According to the result obtained in Theorem 2.1, to test the grade average system, it suffices to test separately the calculator $\mathcal{C}$ and the controller $\mathcal{G}$. Now, testing the calculator $\mathcal{C}$ separately may lead to the consideration of test cases involving arithmetic operations which are irrelevant to computing student grade averages such

as subtraction or multiplication. This may cause test cases of interest to the system to be missed, *i.e.* test cases only bringing into play addition and division for grades ranging from 0 to 20. In the approach we propose in the following, we intend to generate a test purpose that guides the test derivation process of $\mathcal{C}$ by only testing operations needed to compute grade averages. We do this by making a projection of this behavior on calculator component $\mathcal{C}$.

Definition 3.1 (Projection): Let $\mathcal{S} = op(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ be a system over (I, O). Let $sub \in \mathbb{S}ub(\mathcal{S})$ be a sub-system of $\mathcal{S}$ over (I', O'). Let $tr = \langle i_1|o_1, \ldots, i_m|o_m\rangle \in Trace(\mathcal{S})$. The **projection of** tr **on** sub, denoted by $tr_{\downarrow_{sub}}$, is the subset of $Trace(sub)$ inductively defined as follows:

- if $op = \_$, then $tr_{\downarrow_{sub}} = \{tr\}$;
- if $op = op_1 \otimes op_2$ with op_1 and op_2 of arity n_1 and n_2 respectively (i.e. $n = n_1 + n_2$), then[6]:

$$tr_{\downarrow_{sub}} = \begin{cases} is\ the\ projection\ of\ \langle i_{1_{|_1}}|o_{1_{|_1}}, \ldots, i_{m_{|_1}}|o_{m_{|_1}}\rangle \\ on\ sub\ \ if\ sub \in \mathbb{S}ub(op_1(\mathcal{C}_1, \ldots, \mathcal{C}_{n_1})) \\ \\ is\ the\ projection\ of\ \langle i_{1_{|_2}}|o_{1_{|_2}}, \ldots, i_{m_{|_2}}|o_{m_{|_2}}\rangle \\ on\ sub\ otherwise \end{cases}$$

- if $op = \circlearrowleft_{\mathcal{I}}(op')$ with $\mathcal{I} = (f, \pi_i, \pi_o)$, then $tr_{\downarrow_{sub}} = \bigcup_{tr' \in tr_{\downarrow_{\mathcal{S}'}}} tr'_{\downarrow_{sub}}$ where
  - $\mathcal{S}' = op'(\mathcal{C}_1, \ldots, \mathcal{C}_n)$
  - and $tr_{\downarrow_{\mathcal{S}'}} = \big\{\langle i'_1|o'_1, \ldots, i'_m|o'_m\rangle \mid \forall j, 1 \leq j \leq m, \exists s_j \in S',$ $o'_j \in \alpha_{\mathcal{S}'}(s_j)(f(i'_j, o'_j))_{|_1}, i_j = \pi_i(i'_j)\ and\ o_j = \pi_o(o'_j)\big\}$

We then introduce the projection of a system on a one of its sub-systems.

Definition 3.2 (Component in context): Let $\mathcal{S}$ be a system over (I, O) and $sub \in \mathbb{S}ub(\mathcal{S})$ be a subsystem of $\mathcal{S}$ over (I', O'). The **component obtained by projecting** $\mathcal{S}$ **on** sub, noted $\mathcal{S}_{\downarrow_{sub}}$ is the triplet (S, s^0, α) defined by:

- $s^0 = \langle\rangle$
- S is the whole set of finite traces defined as follows:
  - $s^0 = \{\langle\rangle\}$
  - $\forall j, 1 \leq j \leq n, s^j = \{tr'.\langle i|o\rangle \mid \exists tr' \in s^{j-1}, \exists i \in I', \exists o \in O', \exists tr \in Trace(\mathcal{S}) such\ that\ tr'.\langle i|o\rangle \in tr_{\downarrow_{sub}}$

  Hence, $S = \bigcup_{0 \leq j \leq \omega} s^i$
- $\alpha : S \times I' \longrightarrow \mathcal{P}(O' \times S)$ is the mapping which for every $\langle i_0|o_0, \ldots, i_m|o_m\rangle \in S$ and every input $i \in I'$ associates the set:

$$\Pi = \{(o, \langle i_0|o_0, \ldots, i_m|o_m, i|o\rangle) \mid \exists o \in O', \exists tr \in Trace(\mathcal{S})\ such\ that\ \langle i_0|o_0, \ldots, i_m|o_m, i|o\rangle\ \in tr_{\downarrow_{sub}}\}$$

[6] $a_{|_i}$ is the projection of the n-tuple a on i^{th} argument.

It is easy to see that the traces of the component $\mathcal{S}_{\downarrow_{sub}}$ obtained by projection is a subset of the traces of the component sub itself.

Example 3.1: Consider again the grade average system $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G}, \mathcal{C}))$ given in Figure 3. The projection $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G}, \mathcal{C}))_{\downarrow_{\mathcal{C}}}$ of $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G}, \mathcal{C}))$ on the calculator $\mathcal{C}$ is given in Figure 4. By applying Definition 3.2, we only retain the $\mathcal{C}$'s behaviors that are involved in the final behavior of $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G}, \mathcal{C}))$. Only the addition and the division operations are specified in $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G}, \mathcal{C}))_{\downarrow_{\mathcal{C}}}$, the specifications of both subtraction and multiplication operations are omitted due to their absence in the global system $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G}, \mathcal{C}))$.

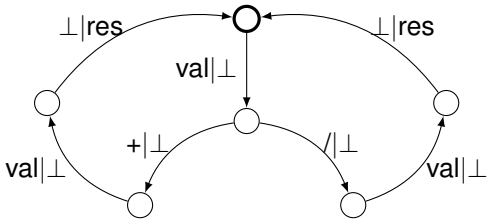


Fig. 4: The projection $\circlearrowleft_{\mathcal{I}}((\mathcal{G}, \mathcal{C}))_{\downarrow_{\mathcal{C}}}$ of $\circlearrowleft_{\mathcal{I}}((\mathcal{G}, \mathcal{C}))$ on the calculator $\mathcal{C}$

Such projected traces will be the cornerstone to improve the compositionality result presented in Theorem 2.1 and to define test purposes dedicated to test components separately while taking into account the behavior of the global system.

Theorem 3.1 (Compositionality with projection): Let op be a complex operator of arity n. Let $\mathsf{iut}_1, \ldots, \mathsf{iut}_n$ be input-enabled implementations and $\mathsf{spec}_1, \ldots, \mathsf{spec}_n$ their specifications respectively. Then, one has $\forall i, 1 \leq i \leq n$

$$(\mathsf{iut}_1\ cioco\ op(\mathsf{spec}_1, \ldots, \mathsf{spec}_n)_{\downarrow_{\mathsf{spec}_1}}),\ \ldots,\ (\mathsf{iut}_n\ cioco\ op(\mathsf{spec}_1, \ldots, \mathsf{spec}_n)_{\downarrow_{\mathsf{spec}_n}}) \Longrightarrow op(\mathsf{iut}_1, \ldots, \mathsf{iut}_n)\ cioco\ op(\mathsf{spec}_1, \ldots, \mathsf{spec}_n)$$

Proof: Sketch of the proof
This is proven by structural induction on the integration operator *op*. The main difficulty is to prove the property preservation over both Cartesian product and feedback operator. Then, we need the following two theorems:

Theorem 3.2 (Compositionality for Cartesian product): Let $\mathcal{C}_1$ and $\mathcal{C}'_1$ be two components over (I_1, O_1), and $\mathcal{C}_2$ and $\mathcal{C}'_2$ be two components over (I_2, O_2). Then, we have:

$$\left.\begin{array}{l} \mathcal{C}_1\ cioco\ \otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_1}} \\ \mathcal{C}_2\ cioco\ \otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_2}} \end{array}\right\} \Longrightarrow \otimes((\mathcal{C}_1, \mathcal{C}_2))\ cioco \otimes((\mathcal{C}'_1, \mathcal{C}'_2))$$

Theorem 3.3 (Compositionality for feedback operator): Let $\Sigma = (I, O)$ be a signature and $\mathcal{I} = (f, \pi_i, \pi_o)$ be a feedback interface. Let $\mathcal{C}_1 = (S_1, \alpha_1)$ and $\mathcal{C}_2 = (S_2, \alpha_2)$ be two components over Σ. Then, we have:

$$\mathcal{C}_1\ cioco\ \circlearrowleft_{\mathcal{I}}(\mathcal{C}_2)_{\downarrow_{\mathcal{C}_2}} \Longrightarrow \circlearrowleft_{\mathcal{I}}(\mathcal{C}_1)\ cioco\ \circlearrowleft_{\mathcal{I}}(\mathcal{C}_2)$$

The proof of both theorems 3.2 and Theorem 3.3 is given in Appendix. ■

Theorem 3.1 then provides a way to test the integrated system only by testing the projection of that system on its sub-systems. As a consequence, to test the integrated system, it is not necessary to consider it as a whole, but it is enough to consider the projection of that system on its sub-systems (which may be done at different development steps and eventually developed by different teams) and test them separately.

Comparing this result with our previous result presented in [3] or *Tretmans*'s result [20], the new result does not require that the specifications are input-enabled. This last property is often hard to get in practice due to the fact that system input domains are usually too large.

B. Test purpose

A specification model usually contains a growth of exponential states which makes the testing process difficult even impossible to be implemented. To cope with this problem, test purposes can be used. A test purpose is a description of the part of the specification that we want to test and for which test cases are later generated. In [14], they are described independently of the model of the specification. In [23], they are deduced from the specification by construction. In order to guide the test derivation process in our approach, we have preferred, as in [23], to describe test purposes by selecting the part of the specification that we want to explore. We therefore consider a test purpose as a tagged finite computation (FCT) tree of the specification. The leaves of the FCT which correspond to paths that we want to test are tagged accept. All internal nodes on such paths are tagged skip, and all other nodes are tagged $\odot$. Formally, FCT is defined as follows:

Definition 3.3 (Finite computation tree of component): Let (S, s_0, α) be a component over (I, O). The **finite computation tree** of depth n of $\mathcal{C}$, noted $FCT(\mathcal{C}, n)$, is the triplet $(S_{FCT}, s^0_{FCT}, \alpha_{FCT})$ defined by:

- S_{FCT} is the whole set of $\mathcal{C}-$paths. A $\mathcal{C}-$path is defined by two finite sequences of states and inputs $(s_0, \ldots, s_n)$ and $(i_0, \ldots, i_{n-1})$ such that:
$$\forall j, 1 \leq j \leq n, s_j \in \alpha(s_{j-1})(i_{j-1})_{|_2}$$
- s^0_{FCT} is the initial $\mathcal{C}-$path $\langle s_0, () \rangle$
- α_{FCT} is the mapping which for every $\mathcal{C}-$path $\langle (s_0, \ldots, s_n), (i_0, \ldots, i_{n-1}) \rangle$ and every input $i \in I$ associates the set:
$\Gamma = \{(o, \langle (s_0, \ldots, s_n, s'), (i_0, \ldots, i_{n-1}, i) \rangle) \mid (o, s') \in \alpha(s_n)(i)\}$

In this definition, S_{FCT} is the set of the nodes of the tree and s^0_{FCT} its root. Each node is represented by the unique $\mathcal{C}$-path $\langle (s_0, \ldots, s_n), (i_0, \ldots, i_{n-1}) \rangle$ which leads to it from the root. α_{FCT} gives, for each node p and for each input i, the set of nodes Γ that can be reached from p when the input i is submitted to $\mathcal{C}$.

We intend in the following to extend the notion of test purpose proposed in [3] to test purpose in context. This latter allows us to test, from a global behavior of a system, the behavior of its involved sub-systems and then guide the component testing intelligently by taking into account the way components are used in systems. Thus, taking a behavior p of a system $\mathcal{S}$, we intend to define test purposes that are able to test the behavior p_i of each sub-system $\mathcal{S}_i \in \mathbb{S}ub(\mathcal{S})$. We identify therefore for each sub-system all its finite paths that are involved in constructing the whole behavior of $\mathcal{S}$.

Definition 3.4 (Test purpose in context): Let $\mathcal{S}$ be a system over (I, O). Let $sub \in \mathbb{S}ub(\mathcal{S})$ be a sub-system of $\mathcal{S}$ and $sub' = \mathcal{S}_{\downarrow_{sub}}$ the projection of $\mathcal{S}$ on sub. Let $FCT(sub, n) = (S, s_0, \alpha)$ be the finite computation tree of sub. A **test purpose in context** TP for sub is a mapping $TP : S_{FCT} \longrightarrow \{\mathsf{accept}, \mathsf{skip}, \odot\}$ such that:

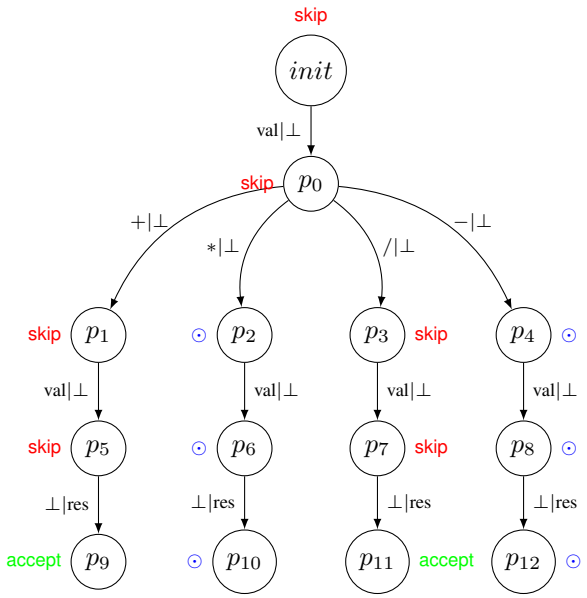
- for every node $p = \langle i_0 | o_0, \ldots, i_m | o_m \rangle \in Trace(sub')$, $TP(p) = \mathsf{accept}$;
- if $TP(\langle i_0 | o_0, \ldots, i_m | o_m) \rangle = \mathsf{accept}$, then:
$\forall j, 0 \leq j \leq m, TP(\langle i_0 | o_0, \ldots, i_{j-1} | o_{j-1}) \rangle) = \mathsf{skip}$
- $TP(\langle \rangle) = \mathsf{skip}$
- if $TP(\langle i_0 | o_0, \ldots, i_k | o_k) \rangle) = \odot$, then:
$TP(i_0 | o_0, \ldots, i_k | o_k, i'_{k+1} | o'_{k+1}, \ldots, i'_{k'} | o'_{k'}) \rangle) = \odot$ for all $k < k' \leq n$ and for all $(i'_l)_{k \leq l < n} \in I'$ and $(o'_l)_{k \leq l < n} \in O'$.

In order to build a test purpose for a subsystem sub, we identify all finite paths of its finite computation tree FCT whose traces embody traces in $Trace(sub')$ and we tag them with accept. We then tag every node which represents a prefix of an accepted behavior with skip. The other nodes, which lead to behaviors that we do not want to test, are tagged with $\odot$.

Example 3.2: In this example, we intend to build a test purpose dedicated to test the behavior of the calculator component $\mathcal{C}$ in the context of the system computing grade averages. To do so, we first build the finite computation tree $FCT(\mathcal{C}, 4)$ of $\mathcal{C}$ that we present in Figure 5. Second, each state of $FCT(\mathcal{C}, 4)$ reachable after each trace tr of the projection $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G}, \mathcal{C}))_{\downarrow_{\mathcal{C}}}$ of $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G}, \mathcal{C}))$ on $\mathcal{C}$ (see Figure 3) is tagged with accept. Then, p_9 and p_{11} are only tagged with accept. All nodes leading from the root $init$ to p_9 or p_{11} are tagged with skip (i.e p_1, p_3, p_5 and p_7). Finally, all other states are tagged with $\odot$.

Thus, testing of $\mathcal{C}$ is re-enforced as far as student grade averages computing is concerned: only behaviors related to grade average computing are chosen and then the behaviors of $\mathcal{C}$ that are not activated in the global system $\circlearrowleft_{\mathcal{I}}(\otimes(\mathcal{G}, \mathcal{C}))$ are not tested. This allows us to restrict the test domain to the one under consideration.

Finally, we use the algorithm developed in Algorithm 1 to generate correct and sound test cases. Given an implementation iut of a subsystem sub of a system $\mathcal{S}$ and the test purpose



$init = \langle q0, ()\rangle, p_0 = \langle (q_0, q_1), (\text{val})\rangle, p_1 = \langle (q_0, q_1, q_2), (+)\rangle,$
$p_2 = \langle (q_0, q_1, q_2), (*)\rangle, p_3 = \langle (q_0, q_1, q_2), (/)\rangle$
$p_4 = \langle (q_0, q_1, q_2), (-)\rangle, p_5 = \langle (q_0, q_1, q_2, q_3), (+, \text{val})\rangle$
$p_6 = \langle (q_0, q_1, q_2, q_3), (*, \text{val})\rangle, p_7 = \langle (q_0, q_1, q_2, q_3), (/, \text{val})\rangle$
$p_8 = \langle (q_0, q_1, q_2, q_3), (-, \text{val})\rangle,$
$p_9 = \langle (q_0, q_1, q_2, q_3, q_4), (+, \text{val}, \bot)\rangle$
$p_{10} = \langle (q_0, q_1, q_2, q_3, q_4), (*, \text{val}, \bot)\rangle$
$p_{11} = \langle (q_0, q_1, q_2, q_3, q_4), (/, \text{val}, \bot)\rangle$
$p_{12} = \langle (q_0, q_1, q_2, q_3, q_4), (-, \text{val}, \bot)\rangle$

Fig. 5: Test purpose of the calculator component

TP for sub generated from $\mathcal{S}$, we want to test the conformance of the iut to the test purpose TP. We start from the root of TP, we choose a possible input i and submit it to the iut. We observe the outputs o and compare them with the possible outputs in TP. If the outputs do not match the ones specified in TP, the verdict of the test is FAIL. Otherwise, if at least one of the nodes which can be reached with $i|o$ is tagged skip in TP, the test goes on. If the nodes are tagged $\odot$, further behavior is not of interest, so the test is inconclusive (INCONC verdict). If one of the nodes is tagged accept, the test succeeds (PASS verdict). It may happen, due to the non-determinism of the specification, that the implementation behaved correctly, but we cannot determine if we reached an accept state or an $\odot$ state. This leads to a WeakPASS verdict.

IV. Conclusion

This paper extends our previous work [3] which defines a generic testing conformance theory. We have proposed an approach to test components that are typically involved in the whole system by defining test purposes from the global behaviour of the whole system. Such test purposes are given in a accurate way by defining a projection mechanism taking a global behaviour p of the whole system and keeping only the part of p being activated in the sub-system that we want to

input : a test purpose
$TP : FCT = (S, s_0, \alpha) \longrightarrow \{\text{accept}, \text{skip}, \odot\}$
and an implementation iut
output: a test case $[i_0|o_0, i_1|o_1, \ldots, i_n|o_n, \text{verdict}]$

Preliminaries;
$Next(CS, i|o)$ returns the set of directly reachable states from the current set of states CS after executing $i|o$;
$NextSkip(CS, i|o)$ returns the set of states in $Next(CS, i|o)$ which are labeled by skip;
$NextPass(CS, i|o)$ returns the set of states in $Next(CS, i|o)$ which are labeled by accept;
initialization ;
$i \leftarrow \texttt{ChooseInputFrom}(\{i \mid \alpha(s_0)(i) \text{ is defined}\})$;
$o \leftarrow \texttt{ReactionOf}(\text{iut}, i)$;
$CS \leftarrow \{s\}$ *// set of explored states*;
$TC \leftarrow []$ // initialization of the test case;
//sending stimuli to iut *and waiting for its output as long as a verdict is not reached*
while
$NextSkip(CS, i|o) \neq \emptyset$ *and* $NextPass(CS, i|o) = \emptyset$ **do**
  $TC \leftarrow \texttt{Concatenate}(TC, i|o)$;
  $CS \leftarrow Next(CS, i|o)$
  $i \leftarrow \texttt{ChooseInputFrom}(\{i \mid i \in \bigcup_{s \in CS} \{i \mid \alpha(s)(i) \text{ is defined}\}\})$;
  $o \leftarrow \texttt{ReactionOf}(\text{iut}, i)$;
end
$TC \leftarrow \texttt{Concatenate}(TC, i|o)$;
// the emission from the iut is not expected with regards to the specification
if $Next(CS, i|o) = \emptyset$ **then**
  $TC \leftarrow \texttt{Concatenate}(TC, \text{FAIL})$;
end
// the emission from the iut is specified, but not compatible with the test purpose
if $Next(CS, i|o) \neq \emptyset$ *and* $NextSkip(CS, i|o) = NextPass(CS, i|o) = \emptyset$ **then**
  $TC \leftarrow \texttt{Concatenate}(TC, \text{INCONC})$;
end
// all next states directly reachable from the set of current set are accept ones
if $Next(CS, i|o) = NextPass(CS, i|o)$ *and* $Next(CS, i|o) \neq \emptyset$ **then**
  $TC \leftarrow \texttt{Concatenate}(TC, \text{PASS})$;
end
// some of the next states are labeled by accept, but not all of them
if $NextPass(CS, i|o) \subset Next(CS, i|o)$ *and* $NextPass(CS, i|o) \neq \emptyset$ **then**
  $TC \leftarrow \texttt{Concatenate}(TC, \text{WeakPASS})$;
end
return TC;

Algorithm 1: Test generation algorithm

test. Thus, our method for generating test purposes from the global system specification helps to generate relevant unit test cases to test individual components.

June 11, 2012

REFERENCES

[1] D'Souza, D.F. and Wills, A.C., *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach* Addison-Wesley Professional,octobre 1998.

[2] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, ACM Press and Addison-Wesley, New York, NY, 1998.

[3] Marc Aiguier, Frédéric Boulanger and Bilal Kanso, *A formal abstract framework for modeling and testing complex software systems*, Theoretical Computer Science (TCS), Elsevier, December 2011, to appear.

[4] S. MacLane, *Categories for the Working Mathematician*, Springer Verlag, Graduate Texts in Mathematics, New York, Heidelberg, Berlin, 1971.

[5] E. Moggi, *Notions of computation and monads*, Information and Computation journal, 93, 55-92, 1991.

[6] L.S Barbosa, *Towards a Calculus of State-based Software Components*, Journal of Universal Computer Science, 9(8):891-909, August 2003.

[7] Meng, S. and Barbosa, L.S., *Components as coalgebras: the refinement dimension*, Theoretical Computer Science (TCS), 351(2):276-294, Elsevier Science Publishers Ltd, Essex,UK, 2006.

[8] G. H. Mealy, *A method for synthesizing sequentiel circuits*, Bell Systems Techn. Jour. journal, 0167-6423, 1955.

[9] S. Eilenberg, *Automata, Languages and Machines*, Academic Press, New York, 1978.

[10] R. Milner, *A Calculus of Communicating Systems*, Springer-Verlag, New York, Inc, secaucus, NG, USA, 1982.

[11] J. Tretmans, *Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation*, Computer networkss and ISDN systems, 29(1):49-79, 1996.

[12] A. Petrenko and N. Yevtushenko, *Testing from partial deterministic fsm specifications*, IEEE Trans. Comput., 54:11541165, September 2005.

[13] J. Tretmans, *Test Generation with Inputs, Outputs and Repetitive Quiescence*, Software-Concepts and Tools, 17(3):103-120, 1996.

[14] C. Jard, T. Jéron, *TGV: theory, principles and algorithms*, International Journal on Software Tools for Technology Transfer,7(4):297–315, August 2005.

[15] A. Faivre, C. Gaston and P. Le Gall, *Symbolic Model Based Testing for Component Oriented Systems*, , TestCom/FATES, 90–106, 2007.

[16] Edward A. Lee and Pravin Varaiya, *Structure and interpretation of signals and systems*, Addison-Wesley,I-XXI, 1-647, 2003.

[17] Edward A. Lee and Sanjit A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*, Lee and Seshia, 978-0-557-70857-4, 2010.

[18] A. Benveniste and G. Berry, *The synchronous approach to reactive and real-time systems*, Proceedings of the IEEE, 1270–1282, 1991.

[19] C. A. R. Hoare, *Communicating Sequential Processes*, journal of Communications of the ACM, 21: 666-677 1985.

[20] H.M. van der Bijl and A. Rensink and J. Tretmans, *Compositional Testing with ioco*, FATES, A. Petrenko and A. Ulrich, LNCS, 2931:86–100, Berlin, 2004.

[21] S. Brookes and A. W. Roscoe, *An Improved Failures Model for Communicating Processes*, journal of NSF-SERC Seminar on Concurrency, Pittsburgh, Springer Lecture Notes in Computer Science(LNCS) 197:281-305. July 1984.

[22] A. Sampaio, S. Nogueira, A. Mota, *Compositional Verification of Input-Output Conformance via CSP Refinement Checking*, In ICFEM, pages 20–48, Springer-Verlag, Rio de Janeiro, Brazil, 2009.

[23] C. Gaston, P. Le Gall, N. Rapin and A. Touil, *Symbolic Execution Techniques for Test Purpose Definition*, TestCom, 1-18, 2006.

[24] J. Rutten, *Universal coalgebra: a theory of systems*, Theoretical Computer Science (TCS), 249(1), pages = 3-80, 2000.

[25] G. Bernot, *Testing Against Formal Specifications: A Theoretical View*, TAPSOFT'91: Proc. of the Intl. Joint Conference on Theory and Practice of Software Development, 2: 99-119, Springer-Verlag, London, UK, 1991.

[26] J. Tretmans, *A Formal Approach to Conformance Testing*, Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI, 257–276, Amsterdam, The Netherlands, 1994.

V. APPENDIX

Proof: **Compositionality for synchronous feedback (Theorem 3.3)**

We first need to prove the following lemma:

Lemma 5.1: Consider two components $\mathcal{C}_1$ and $\mathcal{C}_2$, then we have: $\mathcal{C}_1 \ cioco \ \circlearrowleft_{\mathcal{I}}(\mathcal{C}_2)_{\downarrow_{\mathcal{C}_2}} \Longrightarrow \forall tr \in Trace(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_1)) \cap Trace(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2)), tr_{\downarrow_{\mathcal{C}_2}} \subseteq tr_{\downarrow_{\mathcal{C}_1}}$

Proof: Let us prove this point by induction on the structure of a trace tr in $Trace(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_1)) \cap Trace(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2))$. Let $tr = \langle i_1|o_1, i_2|o_2, \ldots, i_n|o_n\rangle$.

- **Basic Step:** $tr = \langle\rangle$ is empty trace.
  $tr_{\downarrow_{\mathcal{C}_2}} = \{\langle\rangle\} \subseteq tr_{\downarrow_{\mathcal{C}_1}} = \{\langle\rangle\}$ trivially holds.
- **Induction Step:** Let us write tr as concatenation of two finite traces: $tr = \langle i_1|o_1, i_2|o_2, \ldots, i_{n-1}|o_{n-1}\rangle \cdot \langle i_n|o_n\rangle$. Let $\sigma = \langle i'_1|o'_1, \ldots, i'_{n-1}|o_{n-1}, i'_n|o'_n\rangle \in tr_{\downarrow_{\mathcal{C}_2}}$ and let us prove that $\sigma \in tr_{\downarrow_{\mathcal{C}_1}}$. $\sigma \in tr_{\downarrow_{\mathcal{C}_2}}$, then according to the definition of $tr_{\downarrow_{\mathcal{C}_2}}$, there exists a finite sequence of states $s_0, \ldots, s_n$ of S_2 such that $\forall j, 1 \leq j \leq n$:
  - $(o'_j, s_j) \in \alpha_2(s_{j-1})(f(i'_j, o'_j))$ and
  - and $\pi_i(i'_j) = i_j$ and $\pi_o(o'_j) = o_j$

  Now, by induction hypothesis, we have $\sigma = \langle i'_1|o'_1, \ldots, i'_{n-1}|o_{n-1}\rangle \in tr_{\downarrow_{\mathcal{C}_1}}$, then according to the definition of $tr_{\downarrow_{\mathcal{C}_1}}$, there exists a finite sequence of states $s'_0, \ldots, s'_{n-1}$ of S_1 such that $\forall j, 1 \leq j \leq n-1$:
  - $(o'_j, s'_j) \in \alpha_1(s'_{j-1})(f(i'_j, o'_j))$ and
  - and $\pi_i(i'_j) = i_j$ and $\pi_o(o'_j) = o_j$

  One has that $\sigma = \langle i'_1|o'_1, \ldots, i'_{n-1}|o_{n-1}, i'_n|o'_n\rangle \in Trace(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2)_{\downarrow_{\mathcal{C}_2}})$ since $\sigma \in tr_{\downarrow_{\mathcal{C}_2}}$ and $tr_{\downarrow_{\mathcal{C}_2}} \subseteq Trace(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2)_{\downarrow_{\mathcal{C}_2}})$ (see Definition 3.2). That means that

  $$o'_n \in Out(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2)_{\downarrow_{\mathcal{C}_2}} \text{ after } (\langle i'_1|o'_1, \ldots, i'_{n-1}|o'_{n-1}\rangle, i'_n))$$

  But we know that $\mathcal{C}_1$ is input-enabled, then i'_n is inevitably an input of the state s'_{n-1}. Hence, one has

  $$o'_n \in Out(\mathcal{C}_1 \text{ after } (\langle i'_1|o'_1, \ldots, i'_{n-1}|o'_{n-1}\rangle, i'_n))$$

  because of $(\mathcal{C}_1 \ cioco \ \circlearrowleft_{\mathcal{I}}(\mathcal{C}_2)_{\downarrow_{\mathcal{C}_2}})$. That means there exists $s'_n \in S_1$ such that $(o'_n, s'_n) \in \alpha_1(s'_{n-1})(f(i'_n, o'_n))$ since $\mathcal{C}_1$ is well-formed for $\mathcal{I}$. We know also $\pi_i(i'_j) = i_j$ and $\pi_o(o'_j) = o_j$, thus $\langle i'_1|o'_1, \ldots, i'_{n-1}|o'_{n-1}, i'_n|o'_n\rangle \in Tr_{\downarrow_{\mathcal{C}_1}}$. Consequently, $Tr_{\downarrow_{\mathcal{C}_2}} \subseteq Tr_{\downarrow_{\mathcal{C}_1}}$.

■

Let us now prove Theorem 3.3. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two components over (I, O), and $\circlearrowleft_{\mathcal{I}}(\mathcal{C}_1)$ and $\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2)$ over (I', O').

Let $tr = \langle i_1|o_1, \ldots, i_n|o_n\rangle \in Trace(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_1)) \cap Trace(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2))$ and $(i_{n+1}, o_{n+1}) \in I' \times O'$ such that:

$$o_{n+1} \in \mathsf{Out}(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_1) \text{ after } (tr, i_{n+1}))$$

Then, let us prove that

$$o_{n+1} \in \mathsf{Out}(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2) \text{ after } (tr, i_{n+1}))$$

Let us define the set X by

$\{i'_{n+1} \mid \langle i'_1|o'_1, \ldots, i'_n|o'_n, i'_{n+1}|o'_{n+1}\rangle \in tr. < i_{n+1}|o_{n+1} >_{\downarrow_{\mathcal{C}_1}}\}$

of all inputs enabling in $\mathcal{C}_1$ after projecting the trace $tr.\langle i_{n+1}|o_{n+1}\rangle$ on $\mathcal{C}_1$. Since, $tr_{\downarrow_{\mathcal{C}_2}} \subseteq tr_{\downarrow_{\mathcal{C}_1}}$ (By Lemma 5.1), we can extract from X the set:
$Y = \{i'_{n+1} \mid \langle i'_1|o'_1, \ldots, i'_n|o'_n, i'_{n+1}|o'_{n+1}\rangle \in tr_{\downarrow_{\mathcal{C}_1}} \; and \; \langle i'_1|o'_1, \ldots, i'_n|o'_n\rangle \in tr_{\downarrow_{\mathcal{C}_2}}\}$

of all inputs enabling in $\mathcal{C}_1$ after the traces obtained by projecting tr on $\mathcal{C}_2$.

In the same manner, let us define the set
$Z = \{i'_{n+1} \mid \langle i'_1|o'_1, \ldots, i'_n|o'_n, i'_{n+1}|o'\rangle \in tr. < i_{n+1}|o >_{\downarrow_{\mathcal{C}_2}} and \; o \in Out(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2) \text{ after } (tr, i_{n+1}))\}$

of all inputs enabling in $\mathcal{C}_2$ after projecting the trace tr on $\mathcal{C}_2$.

By construction of Y and Z, we have that $Z \subseteq Y$. Since $\mathcal{C}_1 \; cioco \; \mathcal{C}_{2_{\downarrow_{\mathcal{C}_2}}}$, then for every $\sigma \in tr_{\downarrow_{\mathcal{C}_2}}$ and for every $i \in Z$,

$$Out(\mathcal{C}_1 \text{ after } (\sigma, i)) \subseteq Out(\mathcal{C}_{2_{\downarrow_{\mathcal{C}_2}}} \text{ after } (\sigma, i)) \quad (1)$$

Let $\Phi = \{\sigma.\langle i'_{n+1}|o'_{n+1}\rangle \mid \sigma \in tr_{\downarrow_{\mathcal{C}_2}}, o'_{n+1} \in Out(\mathcal{C}_1 \text{ after } (\sigma, i'_{n+1})), and \; i'_{n+1} \in Z\}$

Since $\sigma.\langle i'_{n+1}|o'_{n+1}\rangle \in tr.\langle i_{n+1}|o_{n+1}\rangle_{\downarrow_{\mathcal{C}_1}}$ then by the projection definition, one has

$$\pi_i(i'_{n+1}) = i_{n+1} \; and \; \pi_i(o'_{n+1}) = o_{n+1} \quad (2)$$

By (1), (2) and the definition of $tr.\langle i_{n+1}|o_{n+1}\rangle_{\downarrow_{\mathcal{C}_2}}$, we can conclude that $\Phi = tr.\langle i_{n+1}|o_{n+1}\rangle_{\downarrow_{\mathcal{C}_2}}$. Thus $tr.\langle i_{n+1}|o_{n+1}\rangle \in Trace(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2))$. consequently, $o_{n+1} \in Out(\circlearrowleft_{\mathcal{I}}(\mathcal{C}_2) \text{ after } (tr, i_{n+1}))$. ■

Proof: **Compositionality for Cartesian product (Theorem 3.2)**
Let us assume that

$$\mathcal{C}_1 \; cioco \; \otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_1}} \text{ and } \mathcal{C}_2 \; cioco \; \otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_2}}$$

and then prove that $\otimes((\mathcal{C}_1, \mathcal{C}_2))$ cioco $\otimes((\mathcal{C}'_1, \mathcal{C}'_2))$.
Let us use the contradiction principle. For this, let us assume that

$$\neg(\otimes((\mathcal{C}_1, \mathcal{C}_2)) \text{ cioco } \otimes((\mathcal{C}'_1, \mathcal{C}'_2)))$$

i.e that there exists a finite trace $tr = \langle (i_1, i'_1)|(o_1, o'_1), \ldots, (i_n, i'_n)|(o_n, o'_n)\rangle \in Trace(\otimes((\mathcal{C}'_1, \mathcal{C}'_2)))$ and $(i, i') \in I_1 \times I_2$ such that there exists an output $(o, o') \in O_1 \times O_2$ among the outputs obtained after executing $(tr, (i, i'))$ on $\otimes((\mathcal{C}_1, \mathcal{C}_2))$ not belonging to the ones obtained after executing $(tr, (i, i'))$ on $\otimes((\mathcal{C}'_1, \mathcal{C}'_2))$.

Now, we have

$$tr = \langle (i_1, i'_1)|(o_1, o'_1), \ldots, (i_n, i'_n)|(o_n, o'_n)\rangle \in Trace(\otimes((\mathcal{C}_1, \mathcal{C}_2)))$$

According to the definition of the cartesian product, it is easy to show that the two traces:

$$tr_1 = \langle i_1|o_1, \ldots, i_n|o_n\rangle \in Trace(\mathcal{C}_1)$$

and

$$tr_2 = \langle i'_1|o'_1, \ldots, i'_n|o'_n\rangle \in Trace(\mathcal{C}_2)$$

are respectively the traces involved in $\mathcal{C}_1$ and $\mathcal{C}_2$ to obtain tr.

We also know by the projection definition (see Definition 3.2) that $tr_1 \in Trace(\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_1}})$ and $tr_2 \in Trace(\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_2}})$.

Since $(o, o') \in Out(\otimes((\mathcal{C}_1, \mathcal{C}_2)) \text{ after } (tr, (i, i')))$ and tr is composed of tr_1 and tr_2, then $o \in Out(\mathcal{C}_1 \text{ after } (tr_1, i))$ and $o' \in Out(\mathcal{C}_2 \text{ after } (tr_2, i'))$. Similarly, $o \notin Out(\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_1}} \text{ after } (tr_1, i))$ and $o' \notin Out(\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_2}} \text{ after } (tr_2, i'))$ because $(o, o') \notin Out(\otimes((\mathcal{C}'_1, \mathcal{C}'_2)) \text{ after } (tr, (i, i')))$ and tr_1 and tr_2 are involved to obtain tr. Hence, there exists a trace $tr_1 \in Trace(\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_1}})$, an input i of $\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_1}}$ and an output $o \in O_1$ such that $o \in Out(\mathcal{C}_1 \text{ after } (tr_1, i))$ and $o \notin Out(\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_1}} \text{ after } (tr_1, i))$.

In the same manner, there exists a trace $tr_2 \in Trace(\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_2}})$ an input i' of $\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_2}}$ and an output $o' \in O_2$ such that $o' \in Out(\mathcal{C}_2 \text{ after } (tr_2, i'))$ and $o' \notin Out(\otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_1}} \text{ after } (tr_2, i'))$. Indeed, this means that $\neg(\mathcal{C}_1 \; cioco \; \otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_1}})$ and $\neg(\mathcal{C}_2 \; cioco \; \otimes((\mathcal{C}'_1, \mathcal{C}'_2))_{\downarrow_{\mathcal{C}'_2}})$. Hence, we have a contradiction with our hypothesis. ■