Synchronous Interfaces and Assume/Guarantee Contracts

Albert Benveniste[1] and Benoît Caillaud[1]

Inria, Campus de Beaulieu, 35042 Rennes cedex, France, Email: firstname.lastname@inria.fr

Abstract. In this short note, we establish a link between the theory of Moore Interfaces proposed in 2002 by Chakraborty et al. as a specification framework for synchronous transition systems, and the Assume/Guarantee contracts as proposed in 2007 by Benveniste et al. as a simple and flexible contract framework. As our main result we show that the operation of *saturation* of A/G contracts (namely the mapping $(A,G) \mapsto (A, G \vee \neg A)$), which was considered a drawback of this theory, is indeed implemented by the *Moore Game* of Chakraborty et al. We further develop this link and come up with some remarks on Moore Interfaces.

Keywords: Assume/Guarantee Contract · Moore Interface · Synchronous Interface · Compositional Design

Hej Kim!

It is both a pleasure and an honor to write a tribute to Kim. Kim was *preincarnated* a "contractor": in his previous life, by inventing modal specifications he contributed to contracts way before the concept ever existed. But there was a long way to the grail: getting to the point where Modal Interfaces have become comprehensive and solid occurred only recently. While joining the aristocracy of formal methods, Modal Interfaces have become terribly sophisticated. Tom (Hallo Tom!) kept telling us: "those asynchronous interfaces are too complex, look for the synchronous ones". We offer this trial to Kim as a gift. Is it really simple? We let you judge.

1 Introduction

Since the early 2000 and the pioneering paper [22], the community of formal verification started to address component based design in a new, game based, way. The idea is to support a process, by which different actors would contribute to developing a system by designing sub-systems independently, for subsequent integration by the system designer. Each sub-system is developed with some abstract specification of what the system should do, as well as its contexts of use. And the goal is, of course, that, after integration, the resulting system shall work as expected.

Specification [2,1,3,6,7,8,9], *Interface* [22,19,25,23,26,27,16,17,15], and *Contract* [10,14,21,24,9,20] theories were proposed with this common objective in mind. The models are numerous and vary in many respects: automata or state machines, transition

systems, dataflow systems are considered as an underlying paradigm; assumptions and guarantees may be explicitly manipulated, or they may be folded into a single entity called the "interface"; in all cases, however, a notion of environment is considered. The area is rich in technicalities. As a result, the reader may get confused when searching for the essence of the subject beyond its general objectives.

For these reasons a group of hard workers has proposed a *meta-theory of contracts* [12] as an attempt to capture the essence of all the different frameworks. This meta-theory supports the cooperative development of systems from sub-systems and/or components, all of them generically referred to as *components* in the meta-theory. Regarding the components, we assume a composition $\times$ for them that is commutative and associative. The meta-theory defines the semantics of a contract as a pair of two sets of components: a set of legal *environments* (or contexts of use), and a set of *implementations*: $\mathbf{Sem}(\mathscr{C}) = (\mathcal{E}, \mathcal{M})$. To rephraze this, a component E is a legal environment for $\mathscr{C}$ (written $E \models^{\text{E}} \mathscr{C}$) if $E \in \mathcal{E}$ and a component M is a legal implementation for $\mathscr{C}$ (written $M \models^{\text{M}} \mathscr{C}$) if $M \in \mathcal{M}$. To account for the fact that some syntax must exist for contracts to be finitely described, not all pairs of sets of components define contracts. We thus assume some underlying abstract class $\mathbb{C}$ of contracts, whose semantics are pairs $(\mathcal{E}, \mathcal{M})$. To capture substitutability, we say that $\mathscr{C}'$ *refines* $\mathscr{C}$, written $\mathscr{C}' \preceq \mathscr{C}$, if $\mathcal{E}' \supseteq \mathcal{E}$ and $\mathcal{M}' \subseteq \mathcal{M}$, which immediately defines the *conjunction* as the Greatest Lower Bound (GLB) $\mathscr{C}_1 \wedge \mathscr{C}_2$. Most interesting is then the definition of the contract composition $\mathscr{C}_1 \otimes \mathscr{C}_2$ in the meta-theory: it is the *min* of the set of all contracts $\mathscr{C}$ such that: (*i*) $M_1 \models^{\text{M}} \mathscr{C}_1$ and $M_2 \models^{\text{M}} \mathscr{C}_2$ imply $M_1 \times M_2 \models^{\text{M}} \mathscr{C}$, and (*ii*) $E \models^{\text{E}} \mathscr{C}$ and $M_2 \models^{\text{M}} \mathscr{C}_2$ imply $E \times M_2 \models^{\text{E}} \mathscr{C}_1$. Parallel composition is shown to be monotonic with respect to refinement. We regard as axioms the existence of the above invoked GLB and min. To summarize, it is shown in [12] that the meta-theory by itself supports substitutability and other properties that are useful for systems design in an OEM/supplier context.

In [12], it was also shown that, by instantiating the framework of components in various ways, the meta-theory instantiates as existing theories of interfaces or contracts, thus capturing the very essence of them. Among them, *Assume/Guarantee contracts* (A/G contracts) are simple and elegant [10,12]. An A/G contract is a pair (A, G) of *assumption* and *guarantee*, consisting of predicates over the sets of behaviors of a tuple of variables. The pairs $(\mathcal{E}, \mathcal{M})$ of the meta-theory follow directly via the association $\mathcal{E} \leftrightarrow A$ (legal environments are those satisfying A) and $\mathcal{M} \leftrightarrow [A \Rightarrow G]$, where $\Rightarrow$ denotes implication (legal implementations are those satisfying the entailment $A \Rightarrow G$). The latter association reflects that implementations must meet the guarantees only if put in a legal context. The need for manipulating the so-called *saturation* operation $(A, G) \mapsto [A \Rightarrow G] = [A \vee \neg G]$, which seemingly requires computing disjunctions and complements, has been considered a drawback of A/G contracts — even if G is a finite state automaton, computing its complement is computationally costly as soon as G is nondeterministic.

In a landmark paper [19], *Synchronous Interfaces* with the special case of *Moore Interfaces*, were introduced. Two verbatims from [19] (modulo notations) are reproduced in Table 1. These requirements for an interface theory stated in [19] suggest that synchronous interfaces should obey the meta-theory. While reading the above reference in an attempt to properly discussing it in our paper [12], we observed that the game asso-

In the study of compatibility, game-based approaches quantify inputs existentially, and outputs universally. When two interfaces $\mathscr{C}_1$ and $\mathscr{C}_2$ are composed, their composition may have illegal states, where one component emits outputs that are illegal inputs for the other one. Yet, $\mathscr{C}_1$ and $\mathscr{C}_2$ are considered compatible as long as there is some input behavior that ensures that, for all output behaviors, the illegal states are avoided: in other words, $\mathscr{C}_1$ and $\mathscr{C}_2$ are compatible if there is some environment in which they can be used correctly together. In turn, the input behaviors that ensure compatibility constitute the legal behaviors for the composition $\mathscr{C}_1 \otimes \mathscr{C}_2$: when composing component models, both the possible output behaviors, and the legal input behaviors, are composed.

The game view leads to an *alternating* view of refinement: a more detailed interface $\mathscr{C}_2$ refines an abstract interface $\mathscr{C}_1$ if all legal inputs for $\mathscr{C}_1$ are also legal for $\mathscr{C}_2$, and if, when $\mathscr{C}_1$ and $\mathscr{C}_2$ are subject to the same legal inputs, $\mathscr{C}_2$ generates output behaviors that are a subset of those of $\mathscr{C}_1$. This definition ensures that, whenever $\mathscr{C}_2 \preceq \mathscr{C}_1$, we can substitute $\mathscr{C}_2$ for $\mathscr{C}_1$ in every design without creating any incompatibility: in the game view, substitutivity of refinement holds.

Table 1. Two verbatims from [19]

ciated to the composition of Moore interfaces seemed to solve the *saturation* operation on A/G contracts: $(A, G) \mapsto (A, G \vee \neg A)$, see (6). We thought that this observation was worth further investigations, which lead to this paper in which we show that this guess was indeed correct. The contributions of this paper are the following:

1. We show that the *Moore Game* of [19] yields an effective algorithm for performing the *saturation* operation $(A, G) \mapsto (A, G \vee \neg A)$.
2. We clarify the correspondence between A/G contracts and Moore Interfaces. It turns out to be almost perfect. The only missing feature of the alternating refinement of Moore Interfaces is the proper consideration of legal environments, which has consequences for the parallel composition of Moore Interfaces as well.
3. We propose a slight adjustment of the Moore Interfaces that match A/G contracts (and thus the meta-theory).

2 Background on synchronous Assume/Guarantee contracts

In Assume/Guarantee contracts (A/G contracts), Assumptions characterize the valid environments for the considered component, whereas the Guarantees specify the commitments of the component itself, when put in interaction with a valid environment. We develop here A/G contracts for synchronous frameworks in which behaviors are sequences of successive reactions assigning values to the set of variables of the considered system. To simplify the exposure, we focus on the simplest case of a *fixed* alphabet of variables. The extension to the general case relies on a standard mechanism of alphabet extension, for which the reader is referred to [12].

We consider a finite alphabet V of variables possessing identical domain D. Synchronous assertions, which constitute the basis of synchronous A/G-components and contracts, are introduced next. A *reaction* assigns to each variable of V a value from its domain: $s \in D^V$. By adding a distinguished symbol $\perp \notin D$ to model the *absence* of

an actual variable in the considered reaction, we get the multiple-clocked synchronous model used by synchronous languages [13]. Denote by $\varepsilon = \bot^V$ the *silent reaction*, assigning $\bot$ to every variable. A *synchronous behavior* σ is a finite or infinite sequence of reactions. A *synchronous assertion* P is a set of synchronous behaviors:

$$P \subseteq (V \mapsto (D \cup \{\bot\}))^\omega. \tag{1}$$

Say that P is *stuttering invariant* [11] if: 1) it is closed under the transformations

$$\sigma = s_1, \ldots, s_k, s_{k+1}, \ldots \quad \mapsto \quad stretch_k(\sigma) = s_1, \ldots, s_k, \bot^V, s_{k+1}, \ldots \tag{2}$$

where k is an arbitrary integer — inserting at any time k a silent reaction in a behavior of P still yields a behavior of P —, and 2) P is a closed set when $(V \mapsto (D \cup \{\bot\}))^\omega$ is equipped with the product discrete topology. In particular, if P is stuttering invariant, then by using condition 1) of stuttering invariance, it contains behaviors beginning with the silent behavior ε^k with an arbitrary length k. By condition 2) of stuttering invariance, the behavior ε^ω having only silent reactions, which is the limit with respect to the product topology of a sequence of behaviors beginning by ε^k, also belongs to P. Stuttering invariance is a desirable property for an open system, since it may be subsequently put in an environment that is acting when the considered system is sleeping. From now on and until otherwise mentioned, we omit the term "synchronous". Assertions are equipped with the set algebra $\cap, \cup, \neg$, where $\neg$ denotes set complement.

Definition 1. *A* component *is any stuttering invariant assertion.*

Thus, it is always allowed for a component to do nothing. The class of components is stable under intersection. Two components are always composable and we define component composition by the intersection of their respective assertions:

$$P_1 \times P_2 = P_1 \cap P_2 \tag{3}$$

Formulas (1) and (3) define a framework of synchronous components. It coincides with the framework used in [11].

Definition 2. *A* contract *is a pair* $\mathscr{C} = (A, G)$ *of assertions, called the* assumptions *and the* guarantees. *The set* $\mathcal{E}_\mathscr{C}$ *of the legal environments for* $\mathscr{C}$ *collects all components* E *such that* $E \subseteq A$. *The set* $\mathcal{M}_\mathscr{C}$ *of all components implementing* $\mathscr{C}$ *is defined by* $A \times M \subseteq G$.

Observe that we are not requiring any particular condition on the sets A and G. In particular, they may not be stuttering invariant — for instance the guarantee G may request that every reaction shall be non-silent, which is a progress condition. A or G may even be empty. For this section, the underlying set $\mathbb{C}$ of contracts is the set of all pairs (A, G) of assumptions and guarantees as defined above. By Definition 1,

$$\begin{aligned}&\text{contract } \mathscr{C} = (A, G) \text{ is } \textit{compatible} \text{ if and only if } \varepsilon^\omega \in A, \text{ and in this case}\\ &E_\mathscr{C} = A \text{ is the maximal (for set inclusion) environment of } \mathscr{C}.\end{aligned} \tag{4}$$

Denoting by $\neg A$ the complement of set A, any component M such that $M \subseteq G \cup \neg A$ is an implementation of $\mathscr{C}$. Thus,

$$\begin{aligned}&\text{contract } \mathscr{C} = (A, G) \text{ is } \textit{consistent} \text{ if and only } \varepsilon^\omega \in G \cup \neg A, \text{ and in this case}\\ &M_\mathscr{C} = G \cup \neg A \text{ is the maximal (for set inclusion) implementation of } \mathscr{C}.\end{aligned} \tag{5}$$

Observe that two contracts $\mathscr{C}$ and $\mathscr{C}'$ with identical alphabets of variables, identical assumptions $A' = A$, and such that $G' \cup \neg A' = G \cup \neg A$, possess identical sets of implementations: $\mathcal{M}_{\mathscr{C}'} = \mathcal{M}_{\mathscr{C}}$. According to our meta-theory, such two contracts are equivalent. Say that contract

$$\mathscr{C} = (A, G) \text{ is } \textit{saturated} \text{ if } G = G \cup \neg A, \text{ or, equivalently, if } G \cup A = \Omega\,, \tag{6}$$

where $\Omega =_{\text{def}} (V \mapsto D)^* \cup (V \mapsto D)^\omega$ is the trivial assertion collecting all behaviors. Contract $\mathscr{C} = (A, G)$ is equivalent to its saturated form $(A, G \cup \neg A)$. Refinement, conjunction, and parallel composition are defined as follows, for A/G contracts in saturated form:

Definition 3. *Let $\mathscr{C}_1$ and $\mathscr{C}_2$ be two saturated contracts with identical alphabets of variables.*

1. *Say that $\mathscr{C}_2$* refines *$\mathscr{C}_1$, written $\mathscr{C}_2 \preceq \mathscr{C}_1$, iff $A_2 \supseteq A_1$ and $G_2 \subseteq G_1$;*
2. *The* conjunction *of $\mathscr{C}_1$ and $\mathscr{C}_2$ is defined as being the corresponding GLB: $\mathscr{C}_1 \wedge \mathscr{C}_2 =_{\text{def}} (A_1 \cup A_2, G_1 \cap G_2)$;*
3. *The* parallel composition *of $\mathscr{C}_1$ and $\mathscr{C}_2$, denoted by $\mathscr{C}_1 \otimes \mathscr{C}_2$, is defined as being the pair (A, G) such that $G = G_1 \cap G_2$ and $A = (A_1 \cap A_2) \cup \neg(G_1 \cap G_2)$.*

Comment 1 (regarding saturated contracts) As the reader has noticed, getting saturated contracts is important in A/G contracts. This seems to require computing unions and complements of assertions. In fact, we only need to be able to compute the operation $(A, G) \mapsto G \cup \neg A$, which we like to interpret as the entailment $A \Rightarrow G$. As we shall see in Section 4, it turns out that the *Moore Interfaces*, the simplest form of *Synchronous Component Interfaces* proposed by Chakrabarti et al. [19], provide a way of computing this entailment, for a restricted class of A/G contracts.

3 An illustration example for Moore Interfaces

To give the intuition behind Moore Interfaces, we reproduce the following example, borrowed verbatim from the thesis of Arindam Chakrabarti [18]. It is shown in Figure 1.

The guarded-command syntax used in this figure is derived from the one of reactive modules [4] and Mocha [5]; input atoms describe the input assumptions, and the output atoms describe the output behavior. When more than one guard is true, the command is selected nondeterministically. Input variables not mentioned by the command are updated nondeterministically.

We illustrate the features of Moore interfaces by modeling a simple example: a ± 1 adder driven by a binary counter. The adder `Adder` has two control inputs q_0 and q_1, data inputs $i_7, \ldots, i_0$, and data outputs $o_7, \ldots, o_0$. When $q_0 = q_1 = 1$, the adder leaves the input unchanged: the next value of $o_7, \ldots, o_0$ is equal to $i_7, \ldots, i_0$. When $q_0 = 0$ and $q_1 = 1$, the next outputs are given by $[o'_7, \ldots, o'_0] = [i_7, \ldots, i_0] + 1 \bmod 2^8$, where primed variables denote the values at the next clock cycle, and $[o'_7, \ldots, o'_0]$ is the integer encoded in binary by $o'_7, \ldots, o'_0$. Similarly, when $q_1 = 0$ and $q_0 = 1$, we have $[o'_7, \ldots, o'_0] = [i_7, \ldots, i_0] - 1 \bmod 2^8$.

```
interface Counter
  output q0, q1: bool;
  input cl: bool;
  input atom
    init
      [] true -> cl :=nondet
    update
      [] true -> cl:=nondet
  endatom
  output atom
    init
      [] true -> q0:=1; q1:=1;
    update
      [] cl -> q1:=1; q0:=1
      [] ~cl & q1 & q0 -> q1:=1; q0:=0
      [] ~cl & q1 & ~q0 -> q1:=0; q0:=1
      [] ~cl & ~q1 & q0 -> q1:=0; q0:=0
      [] ~cl & ~q1 & ~q0 -> q1:=1; q0:=1
  endatom
end interface
```

```
interface Adder
  input q0, q1: bool; di: [0..7];
  output do: [0..7];
  input atom
    init
      [] true -> q0:=1
      [] true -> q1:=1
    update
      [] true -> q0:=1
      [] true -> q1:=1
  endatom
  output atom
    init
      [] true -> do:=nondet
    update
      [] q0 & q1 -> do:=di
      [] ~q0 & q1 -> do:=di+1
      [] q0 & ~q1 -> do:=di-1
  endatom
end interface
```

Fig. 1. A counter (left) and an adder (right) modeled as Moore interfaces.

The adder is designed with the assumption that q_0 and q_1 are not both 0: hence, the input transition relation of `Adder` states that $q_0'q_1' \neq 00$. In order to cycle between adding $0, +1, -1$, the control inputs q_0 and q_1 are connected to the outputs q_1 and q_0 of a two-bit count-to-zero counter `Counter`. The counter has only one input, cl: when $cl = 0$, then $q_0'q_1' = 11$; otherwise, $[q_1'q_0'] = [q_1q_0] - 1 \bmod 4$.

When the counter is connected to the adder, the joint system can take a transition to a state where $q_1q_0 = 00$, violating the adder's input assumptions. In spite of this, the counter and the adder are compatible, since there is a way to use them together: to avoid the incompatible transition, it suffices to assert $cl = 0$ early enough in the count-to-zero cycle of the counter. To reflect this, when we compose `Counter` and `Adder`, we synthesize for their composition `Counter` $\times$ `Adder` a new input assumption, that ensures that the input assumptions of both `Counter` and `Adder` are satisfied.

To determine the new input assumption, we solve a game between Input, which chooses the next values of cl and $i_7, \ldots, i_0$, and Output, which chooses the next values of q_0, q_1, and $o_7, \ldots, o_0$. The goal of Input is to avoid a transition to $q_1q_0 = 00$. At the states where $q_1q_0 = 01$, Input can win if $cl = 0$, since at the next clock cycle we will have $q_0'q_1' = 11$; but Input cannot win if $cl = 1$. By choosing $cl' = 0$, Input can also win from the states where $q_1q_0 = 10$. Finally, Input can always win from the states where $q_1q_0 = 11$, for all cl'. Thus, we associate with `Counter` $\times$ `Adder` a new input assumption encoded by the transition relation requiring that whenever $q_1q_0 = 10$, then $cl' = 0$. The input requirement $q_1q_0 = 00$ of the adder gives rise, in the composite system, to the requirement that the reset-to-1 occurs early in the count-to-zero cycle of the counter.

So far this was verbatim quote from [18]. This text illustrates the intuition for how composition works for Moore Interfaces. Can we relate this to the composition of A/G contracts?

Item 3 of Definition 3 states that, in the composition of A/G contracts, the overall assumption A is discharged from what is already mutually guaranteed by the two contracts — this corresponds to the term $\cup\neg(G_1 \cap G_2)$. To parallel this with the discussion

of the game associated with Moore Interfaces, the Input only checks what, in the raw product of the two machines, may lead to violating input assumptions of one interface. This expresses that the job of the game is to complement what is already natively offered by each interface.

Considering again the composition of A/G contracts, the remaining duty of the overall assumption A is to ensure that input assumptions of both interfaces remain satisfied in the composition — referring to Item 3 of Definition 3, this corresponds to the term $A_1 \cap A_2$. But this is exactly what the game associated with Moore Interfaces finds, namely: "whenever $q_1 q_0 = 10$, then $cl' = 0$" is the missing global property that inputs must satisfy in the composition of the two Moore interfaces.

This parallel suggests that there should be a tight relation between Moore Interfaces and A/G contracts. Formalizing this relation is the subject of this paper.

4 Implementing contract saturation using Moore Interfaces [19]

In this section we develop the results announced in Comment 1 regarding contract saturation. We specialize our previous trace- or behavior-based framework of A/G contracts to a sub-case where the saturation operation can be made effective by using the Moore Interfaces.

4.1 Moore Interfaces and associated A/G contracts

We now assume that assertions A and G are defined via transition relations having a specific structure. We are given a disjoint copy V' of the set V of variables and call it the set of *next variables*. For $x \in V$, its counterpart in V' is x'. For P a predicate on V, we denote by P' the predicate obtained by replacing in P every $x \in V$ by $x' \in V'$. We next assume that each variable from V has finite domain $D \cup \{\perp\}$ and a decomposition of V is given into *input* and *output* variables: $V = V^{\text{in}} \uplus V^{\text{out}}$. We finally assume

$$\begin{array}{l} \text{a predicate } I_A \text{ on } V^{\text{in}} \text{ and a predicate } T_A \text{ on } V \cup (V^{\text{in}})' \ ; \\ \text{a predicate } I_G \text{ on } V^{\text{out}} \text{ and a predicate } T_G \text{ on } V \cup (V^{\text{out}})' \ . \end{array} \tag{7}$$

Thus, predicates I_A and T_A control input variables, whereas predicates I_G and T_G control output variables.[1] Call *Moore Interface* [19] the tuple

$$\mathscr{C} = (V, I_A, I_G, T_A, T_G) \, .$$

Each Moore Interface defines an A/G contract (A, G) where the two synchronous assertions A (assumption) and G (guarantee) are given by

$$\begin{array}{l} A = \{\, \sigma \mid \sigma(0) \models I_A \ \text{ and } \ \forall k \, . \, (\sigma(k), \sigma(k{+}1)) \models T_A \,\} \\ G = \{\, \sigma \mid \sigma(0) \models I_G \ \text{ and } \ \forall k \, . \, (\sigma(k), \sigma(k{+}1)) \models T_G \,\} \end{array} \tag{8}$$

where, as usual, symbol $\models$ means "satisfies". We now need to define what the components are, for this contract framework.

[1] In addition, [19] assumes some kind of satisfiability condition for these four predicates. We do not consider this assumption in our development.

4.2 Components for Moore Interfaces

Throughout this section we use the concepts introduced in Section 4.1 and develop what the right notion of component is, for A/G contracts defined by Moore Interfaces. Since assumptions A and guarantees G are both specified as transition systems, it is natural to require that the underlying class $\mathbb{M}$ of components consists of all transitions systems on V of the form

$$M = (V_M^{\text{in}}, V_M^{\text{out}}, I_M, T_M),$$

where $V = V_M^{\text{in}} \uplus V_M^{\text{out}}$ is a decomposition of V into *input* and *output* variables, the *initial condition* I_M is a predicate over V^{out}, and the *transition relation* T_M is a predicate over $V \cup (V^{\text{out}})'$. We assume the following conditions on predicates I_M and T_M, where $[V/\bot]$ denotes the assignment of the value $\bot$ to every variable belonging to V and similarly for $[V'^{\text{out}}/\bot]$:

$$\begin{array}{ll} [V/\bot] \text{ satisfies } I_M\text{; and} & (a) \\ \forall V. T_M[V'^{\text{out}}/\bot] \text{ holds,} & (b) \end{array} \tag{9}$$

which means that M is stuttering invariant. Note that, for an arbitrary pair (I_M, T_M), the transformation

$$(I_M, T_M) \;\mapsto\; \Big(I_M \vee [\forall v{\in}V : v{=}\bot]\,,\; T_M \vee [\forall v'{\in}V'^{\text{out}} : v'{=}\bot] \Big) \tag{10}$$

returns a pair satisfying (9). It is, however, a weakening of the original pair.

Two components M_1 and M_2 are *composable* if $V_{M_1}^{\text{out}} \cap V_{M_2}^{\text{out}} = \emptyset$. The composition $M = M_1 \times M_2$ is given by

- $V_M^{\text{out}} = V_{M_1}^{\text{out}} \cup V_{M_2}^{\text{out}}$, $V_M^{\text{in}} = V \setminus V_M^{\text{out}}$,
- $I_M = I_{M_1} \wedge I_{M_2}$, and $T_M = T_{M_1} \wedge T_{M_2}$.

Observe that the so defined pair (I_M, T_M) satisfies (9). The composition $\times$ is associative and commutative.

4.3 Computing the maximal environment and the maximal implementation

The authors of [19] associate, to a pair of Moore Interfaces, a certain two-player game and use it to define the parallel composition and compatibility condition. In our development, we reuse a variation of this game to compute the most liberal environment and the most liberal implementation.

More precisely, to $\mathscr{C}$ a Moore Interface as above, we associate the two-player "Moore game" $\Gamma_{\mathscr{C}}$ introduced next. Playing $\Gamma_{\mathscr{C}}$ results in the construction of a certain behavior σ through its successive reactions. Each round of the game extends the current behavior by one more reaction. We borrow the description of the game $\Gamma_{\mathscr{C}}$ from [19], while exchanging the roles of players *in* and *out*:

Definition 4 (Moore game $\Gamma_{\mathscr{C}}$ [19]).

- *At each round of the game, player* in *chooses new values for the input variables* V^{in} *according to* I_A *at the first round, and then according to* T_A*;*

– *Simultaneously and independently, player* out *chooses unconstrained new values for the output variables* V^{out}*;*
– *Player* out *wins if the resulting behavior* σ *belongs to* G *defined in* (8).

The Moore game $\Gamma_{\mathscr{C}}$ is an adaptation of the game introduced in [19] — the original game will be reintroduced in our context in Section 4.4, when discussing the compatibility between Moore Interfaces and their parallel composition. We closely adapt from [19] an iterative algorithm for computing, if it exists, the most liberal winning strategy for player *out*. This algorithm approximates iteratively

– the predicate C characterizing the set of states from which the player *out* can win the game, and
– the most liberal winning transition relation.

Set $C_0 = \text{T}$ and, for $k \geq 0$:

$$\begin{aligned} T_{k+1} &= \forall (V^{\text{in}})'. \left[T_A \Rightarrow (T_G \wedge C'_k) \right] \\ C_{k+1} &= C_k \wedge \exists (V^{\text{out}})'. T_{k+1} \end{aligned} \tag{11}$$

Note that T_{k+1} is a predicate on $V \cup (V^{\text{out}})'$ and C_{k+1} is a predicate on V. The sequences of predicates C_k and T_k are non-increasing. Since all variables possess a finite domain, the convergence of C_k and T_k to their limits C_∞ and T_∞ arises in finitely many steps and we have

$$\begin{aligned} C_\infty &= \exists (V^{\text{out}})'. \forall (V^{\text{in}})'. \left[T_A \Rightarrow (T_G \wedge C'_\infty) \right] \\ T_\infty &= \qquad\qquad \forall (V^{\text{in}})'. \left[T_A \Rightarrow (T_G \wedge C'_\infty) \right] \end{aligned} \tag{12}$$

which expresses that C_∞ represents the set of states from which player *out* can win the game when setting the initial condition of G to true. Hence,

– $I_\star =_{\text{def}} [I_A \Rightarrow I_G] \wedge C_\infty$ is the weakest initial condition that player *out* must select;
– $T_\star =_{\text{def}} [C_\infty \Rightarrow T_\infty]$ is the most liberal transition relation for *out* to win the game.

The following result is immediate:

Lemma 1. *If* T_G *satisfies condition* (9-*b*)*, then the pair* (C_∞, T_∞) *satisfies* (9)*. If, in addition,* I_G *satisfies condition* (9-*a*)*, then the pair* $(I_\star, T_\star)$ *also satisfies* (9)*.*

Reference [19] contains detailed implementation considerations regarding algorithm (11). If (I_A, T_A) satisfies (9), then $\mathscr{C}$ is compatible and we can consider the component $E_{\mathscr{C}} =_{\text{def}} (V^{\text{out}}, V^{\text{in}}, I_A, T_A)$. If player *out* can win, i.e., $I_\star$ is satisfiable, and if $(I_\star, T_\star)$ satisfies (9), then $\mathscr{C}$ is consistent and we can consider the component $M_{\mathscr{C}} =_{\text{def}} (V^{\text{in}}, V^{\text{out}}, I_\star, T_\star)$.

Theorem 1.

1. *When seeing* $\mathscr{C}$ *as an A/G contract,* $E_{\mathscr{C}}$ *is the maximal environment for* $\mathscr{C}$*, and* $M_{\mathscr{C}}$ *is the maximal implementation of* $\mathscr{C}$*, see* (5)*.*
2. *The map* $(T_A, T_A \Rightarrow T_G) \mapsto M_{\mathscr{C}}$ *is nondecreasing, when predicates are equipped with the order inherited from* $\text{F} \leq \text{T}$ *and components are ordered by inclusion.*

Proof. Statement 1 holds by the very definition of the Moore game. We thus focus on Statement 2. To prove it, it is enough to prove by induction that

$$\text{the map } (T_A, T_A \Rightarrow T_G) \;\; \mapsto \;\; (C_k, T_{k+1}, T_A \Rightarrow C'_k) \text{ is nondecreasing.} \tag{13}$$

Property (13) holds for $k = 0$ by construction, since $C_0 = \top$ and $T_1 = \forall (V^{\text{in}})'.\, [T_A \Rightarrow T_G]$. Assume that (13) holds until $k-1$ and consider two pairs (T_{A_1}, T_{G_1}) and (T_{A_2}, T_{G_2}) s.t.

$$T_{A_1} \leq T_{A_2} \;\text{ and }\; [T_{A_1} \Rightarrow T_{G_1}] \leq [T_{A_2} \Rightarrow T_{G_2}]$$

By the induction assumption we have

$$C^1_{k-1} \leq C^2_{k-1} \text{ and } T^1_k \leq T^2_k \text{ and } [T_{A_1} \Rightarrow C^{1\prime}_k] \leq [T_{A_2} \Rightarrow C^{2\prime}_k]$$

Using (11) we get, on the one hand,

$$C^1_k = C^1_{k-1} \wedge \exists (V^{\text{out}})'.T^1_k \leq C^2_{k-1} \wedge \exists (V^{\text{out}})'.T^2_k = C^2_k$$

which implies, since $T_{A_1} \leq T_{A_2}$

$$[T^1_A \Rightarrow C^{1\prime}_k] \leq [T^2_A \Rightarrow C^{2\prime}_k]$$

On the other hand, we have:

$$\begin{aligned} T^1_{k+1} &= \forall (V^{\text{in}})'.\left[T^1_A \Rightarrow (T^1_G \wedge C^{1\prime}_k)\right] \\ &= \forall (V^{\text{in}})'.\left[(T^1_A \Rightarrow T^1_G) \wedge (T^1_A \Rightarrow C^{1\prime}_k)\right] \\ &\leq \forall (V^{\text{in}})'.\left[(T^2_A \Rightarrow T^2_G) \wedge (T^2_A \Rightarrow C^{2\prime}_k)\right] \\ &\leq T^2_{k+1} \end{aligned}$$

which finishes the proof of Statement 2. □

4.4 Moore Interfaces, seen as A/G contracts

The parallel composition: We continue our development of the link between Moore Interfaces and A/G contracts by considering the parallel composition. The parallel composition and associated compatibility property were the motivation for the authors of [19] to introduce Moore Interfaces and their associated game. Two Moore Interfaces $\mathscr{C}_1$ and $\mathscr{C}_2$ are *composable* if $V^{\text{out}}_1 \cap V^{\text{out}}_2 = \emptyset$ and their parallel composition should then coincide with the composition $\mathscr{C}_1 \otimes \mathscr{C}_2$ where $\mathscr{C}_1$ and $\mathscr{C}_2$ are seen as A/G contracts.

Returning to A/G contracts, if $\mathscr{C}_1$ and $\mathscr{C}_2$ are two A/G contracts in saturated form, then we have seen that their parallel composition is given by the assume/guarantee pair

$$\mathscr{C}_1 \otimes \mathscr{C}_2 = \left([A_1 \wedge A_2] \vee \neg [G_1 \wedge G_2] \,,\, G_1 \wedge G_2 \right). \tag{14}$$

We immediately see that the computation of this parallel composition can be performed as follows:

1. Introduce the dual contract $\widetilde{\mathscr{C}} = (G_1 \wedge G_2 \,,\, A_1 \wedge A_2)$;
2. Compute its saturated form $\left(G_1 \wedge G_2 \,,\, [A_1 \wedge A_2] \vee \neg [G_1 \wedge G_2] \right)$; (15)
3. Take the dual of the result.

The key point is that step 2 of (15) can be performed by computing the winning strategy of the game associated to $\widetilde{\mathscr{C}}$, seen as a Moore Interface. This indeed yields the algorithm originally presented in equation (1) of [19] for checking compatibility:

$$\begin{aligned} T_{k+1} &= \forall (V^{\text{out}})'. \left[(T_{G_1} \wedge T_{G_2}) \Rightarrow (T_{A_1} \wedge T_{A_2} \wedge C'_k) \right] \\ C_{k+1} &= C_k \wedge \exists (V^{\text{in}})'. T_{k+1} \end{aligned} \tag{16}$$

This is summarized in the following result:

Theorem 2. *Computing the parallel composition of two saturated contracts $\mathscr{C}_1 \otimes \mathscr{C}_2$, as defined in* (14)*, is achieved by computing the fixpoint of the algorithm originally presented in equation* (1) *of* [19] *for checking compatibility.*

Refinement: We now compare the refinement relation $\mathscr{C}_2 \preceq \mathscr{C}_1$ stated in Definition 3 for saturated contracts, with the *alternating simulation* of the game $\Gamma_{\mathscr{C}_2}$ by the game $\Gamma_{\mathscr{C}_1}$, as proposed in [19]. The phrazing from [19] reproduced in Table 1 suggests that this alternating refinement should coincide with the refinement for A/G contracts. We now investigate this question.

Let $\mathscr{C}_i = (V_i^{\text{in}} \uplus V_i^{\text{out}}, I_{A_i}, I_{G_i}, T_{A_i}, T_{G_i}), i = 1, 2$, be two Moore Interfaces and denote by (A_i, G_i) their associated A/G contracts. Following Definition 3 of Section 2, we have

$$(A_2, G_2) \preceq (A_1, G_1) \text{ iff } \begin{cases} E_{\mathscr{C}_2} \supseteq E_{\mathscr{C}_1} & (a) \\ M_{\mathscr{C}_2} \subseteq M_{\mathscr{C}_1} & (b) \end{cases} \tag{17}$$

By Statement 2 of Theorem 1, a sufficient condition for the right hand side of (17) to hold is

$$\begin{cases} I_{A_1} \Rightarrow I_{A_2} \text{ and } T_{A_1} \Rightarrow T_{A_2} & (a) \\ [I_{A_2} \Rightarrow I_{G_2}] \Rightarrow [I_{A_1} \Rightarrow I_{G_1}] \text{ and } [T_{A_2} \Rightarrow T_{G_2}] \Rightarrow [T_{A_1} \Rightarrow T_{G_1}] & (b) \end{cases} \tag{18}$$

Following Definition 5 of [19] with appropriate change of notations and taking into account the fact that the alphabet of actions V is fixed, we have $\mathscr{C}_2 \preceq \mathscr{C}_1$ iff $V_2^{\text{in}} = V_1^{\text{in}}$ and the following formulas are valid:

$$[I_{A_1} \wedge I_{G_2} \Rightarrow I_{A_2} \wedge I_{G_1}] \text{ and } [T_{A_1} \wedge T_{G_2} \Rightarrow T_{A_2} \wedge T_{G_1}] \tag{19}$$

Setting $Q = I_A$ or T_A and $P = I_G$ or T_G, we wish to check the following:

$$[Q_2 \Rightarrow P_2] \Rightarrow [Q_1 \Rightarrow P_1] \stackrel{?}{=} [Q_1 \wedge P_2 \Rightarrow Q_2 \wedge P_1]$$

On the one hand we have:

$$\begin{aligned} [Q_1 \wedge P_2 \Rightarrow Q_2 \wedge P_1] &= [Q_2 \wedge P_1] \vee \neg [Q_1 \wedge P_2] \\ &= [Q_2 \wedge P_1] \vee \neg Q_1 \vee \neg P_2 \\ &= [Q_2 \vee \neg Q_1 \vee \neg P_2] \wedge [P_1 \vee \neg Q_1 \vee \neg P_2] \end{aligned}$$

On the other hand, we have:

$$\begin{aligned}[Q_2 \Rightarrow P_2] \Rightarrow [Q_1 \Rightarrow P_1] &= [P_2 \vee \neg Q_2] \Rightarrow [P_1 \vee \neg Q_1] \\ &= P_1 \vee \neg Q_1 \vee [\neg P_2 \wedge Q_2] \\ &= [P_1 \vee \neg Q_1 \vee \neg P_2] \wedge [P_1 \vee \neg Q_1 \vee Q_2] \\ &= [Q_2 \vee \neg Q_1 \vee P_1] \wedge [P_1 \vee \neg Q_1 \vee \neg P_2]\end{aligned}$$

The two expressions differ by the two terms in red. Now, taking (18-*a*) into account, i.e., $Q_1 \Rightarrow Q_2$, the substitution $P_1 \leftrightarrow \neg P_2$ is absorbed by the tautology $Q_2 \vee \neg Q_1$. Thus,

assuming condition (18-*a*), conditions (18-*b*) and (19) become equivalent. (20)

Hence, we can state:

Theorem 3. *Augmenting the alternating refinement* (19) *with condition* (18-*a*) *makes it stronger than A/G contract refinement.*

The possible gap between alternating refinement and A/G contract refinement lies in the fact that (18) is only sufficient for A/G contract refinement. Having (18) *restricted to the set of reachable states* is necessary and sufficient.

The bottom line is that the refinement developed in [19] seems to ignore the condition regarding assumptions. Interestingly enough, the authors were able to relate refinement to parallel composition as expected: parallel composition is monotonic w.r.t. refinement, thus supporting independent development. The following question arises then:

> Is there really any added value in paying attention to both implementations *and environments* as we did in A/G contracts (and in the meta-theory)?

So, what are we missing for sure if we do not handle environments as first class citizens? The answer lies in the meta-theory. One property is lost by Moore Interfaces à la Chakrabarti, namely:

> If E is a legal environment for the composition $\mathscr{C}_1 \otimes \mathscr{C}_2$, and M_1 is an implementation of $\mathscr{C}_1$, then $E \times M_1$ is a legal environment for $\mathscr{C}_2$.

This is a missing property in Moore Interfaces — even in the mind of the authors, see the quotes from [19] reproduced in Table 1 — and we believe its lack weakens somehow Moore Interfaces as a support for independent development.

5 Conclusion

One can say that our contribution in this paper is to mildly modify the Moore Interfaces to make them equivalent to A/G contracts and thus meta-theory compliant, with the advantage of being computationally effective.

We think that the term "interface" used by the authors of [19] is in disagreement with our terminology — we nevertheless kept this term for our exposure. Indeed the

"synchronous interfaces" are not an interface model, in which environments and implementations are folded into a single entity: the "interface". In Moore Interfaces, we rather have two entities T_A and T_G, although both act on the same underlying set of variables. The tight link between Moore Interfaces and A/G contracts — they are nearly identical — that we have just established, further justifies this standpoint. We believe that this link is beneficial both for the A/G contracts and the Moore Interfaces. For A/G contracts, it provides a solution to the embarrassing issue of contract saturation. For Moore Interfaces it points out a (seemingly) missing condition in the alternating refinement.

Reference [19] also generalizes the Moore Interfaces to *Bidirectional Interfaces.* Bidirectional Interfaces offer a dynamic definition of the i/o profile and initial and transition predicates, in that the decomposition $V{=}V^{\mathrm{in}}(q){\uplus}V^{\mathrm{out}}(q)$ and predicates $I_A(q)$, $I_G(q)$ and $T_A(q)$, $T_G(q)$ depend on some *location* $q \in Q$, where the location q evolves according to a deterministic transition system whose transitions are guarded by predicates over the variables of V. This additional flexibility preserves the possibility of considering the game $\Gamma_{\mathscr{C}}$. The follow-up paper [23] studies the conjunction of such interfaces, under the term of *shared refinement.*

As a final observation, Moore Interfaces require finite domains for their variables. Clearly, contract frameworks allowing for any type of data are needed. By only manipulating abstract assertions (sets of behaviors), A/G contracts offer this possibility [12]. In this case, of course, the contract algebra is no longer effective, hence, in [12] we proposed semi-decision procedures based either on *observers* (a kind of test) or on *abstractions.* It may be worth exploring how to extend the Moore Interfaces to this situation. Can Moore Games still be defined? Can we propose semi-decision procedures based on Moore Games? Is this any superior to the existing approaches?

Hej Kim, what do you think?

References

1. Martín Abadi and Leslie Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, January 1993.
2. Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *ICALP*, volume 372 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1989.
3. Yael Abarbanel, Ilan Beer, Leonid Gluhovsky, Sharon Keidar, and Yaron Wolfsthal. FoCs - Automatic Generation of Simulation Checkers from Formal Specifications. In E. Emerson and A. Sistla, editors, *Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 538–542. Springer Berlin / Heidelberg, 2000.
4. Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
5. Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. MOCHA: modularity in model checking. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525. Springer, 1998.

6. Adam Antonik, Michael Huth, Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Complexity of Decision Problems for Mixed and Modal Specifications. In *FoSSaCS*, pages 112–126, 2008.
7. Felice Balarin and Roberto Passerone. Functional verification methodology based on formal interface specification and transactor generation. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE06)*, pages 1013–1018, Munich, Germany, March 6–10, 2006. European Design and Automation Association, 3001 Leuven, Belgium.
8. Felice Balarin and Roberto Passerone. Specification, synthesis and simulation of transactor processes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(10):1749–1762, October 2007.
9. Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Moving from specifications to contracts in component-based design. In Juan de Lara and Andrea Zisman, editors, *FASE*, volume 7212 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2012.
10. Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In *Proceedings of the Software Technology Concertation on Formal Methods for Components and Objects, FMCO'07*, volume 5382 of *Lecture Notes in Computer Science*, pages 200–225. Springer, October 2008.
11. Albert Benveniste, Benoît Caillaud, and Paul Le Guernic. Compositionality in dataflow synchronous languages: Specification and distributed code generation. *Inf. Comput.*, 163(1):125–171, 2000.
12. Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Tom Henzinger, and Kim Larsen. Contracts for System Design: Theory. submitted.
13. Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone. The Synchronous Languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.
14. Luca Benvenuti, Alberto Ferrari, Leonardo Mangeruca, Emanuele Mazzi, Roberto Passerone, and Christos Sofronis. A contract-based formalism for the specification of heterogeneous systems. In *Proceedings of the Forum on Specification, Verification and Design Languages (FDL08)*, pages 142–147, Stuttgart, Germany, September 23–25, 2008.
15. Ferenc Bujtor, Sascha Fendrich, Gerald Lüttgen, and Walter Vogler. Nondeterministic modal interfaces. *Theor. Comput. Sci.*, 642:24–53, 2016.
16. Ferenc Bujtor and Walter Vogler. Error-pruning in interface automata. In *40th International Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM 2014, pages 162–173, Nový Smokovec, Slovakia, January 26-29, 2014.
17. Ferenc Bujtor and Walter Vogler. Error-pruning in interface automata. *Theor. Comput. Sci.*, 597:18–39, 2015.
18. Arindam Chakrabarti. *A Framework for Compositional Design and Analysis of Systems*. PhD thesis, Electrical Engineering and Computer Sciences University of California at Berkeley, available from http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-174.html, December 2007.
19. Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. Synchronous and Bidirectional Component Interfaces. In *Proc. of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 414–427. Springer, 2002.
20. Chris Chilton, Bengt Jonsson, and Marta Z. Kwiatkowska. Compositional assume-guarantee reasoning for input/output component theories. *Sci. Comput. Program.*, 91:115–137, 2014.

21. W. Damm, E. Thaden, I. Stierand, T. Peikenkamp, and H. Hungar. Using Contract-Based Component Specifications for Virtual Integration and Architecture Design. In *Proceedings of the 2011 Design, Automation and Test in Europe (DATE'11)*, March 2011. To appear.
22. Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *Proc. of the 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'01)*, pages 109–120. ACM Press, 2001.
23. Laurent Doyen, Thomas A. Henzinger, Barbara Jobstmann, and Tatjana Petrov. Interface theories with component reuse. In *Proceedings of the 8th ACM & IEEE International conference on Embedded software, EMSOFT'08*, pages 79–88, 2008.
24. Susanne Graf, Roberto Passerone, and Sophie Quinton. Contract-based reasoning for component systems with rich interactions. In Alberto L. Sangiovanni-Vincentelli, Haibo Zeng, Marco Di Natale, and Peter Marwedel, editors, *Embedded Systems Development: From Functional Models to Implementations*, volume 20 of *Embedded Systems*, chapter 8, pages 139–154. Springer New York, 2014.
25. Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Modal I/O Automata for Interface and Product Line Theories. In *Programming Languages and Systems, 16th European Symposium on Programming, ESOP'07*, volume 4421 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2007.
26. Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. Modal interfaces: Unifying interface automata and modal specifications. In *Proceedings of the Ninth International Conference on Embedded Software (EMSOFT09)*, pages 87–96, Grenoble, France, October 12–16, 2009.
27. Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. A modal interface theory for component-based design. *Fundamenta Informaticae*, 108(1-2):119–149, 2011.