# Preserving Contract Satisfiability Under Non-monotonic Composition

Jonas Westman, Mattias Nyberg

# Preserving Contract Satisfiability under Non-Monotonic Composition

Jonas Westman[1] and Mattias Nyberg[1,2]

[1] Royal Institute of Technology (KTH), Stockholm, Sweden
jowestm@kth.se
[2] Scania, Södertälje, Sweden

**Abstract.** A contracts theory embeds *non-monotonic composition* (with respect to implementation) if the fact that a composition of two components implements a specification $\mathcal{S}$ does not generally follow from one of these components implementing $\mathcal{S}$. In contrast to monotonic composition, non-monotonic composition offers the additional expressiveness of specifying properties that *only* hold locally for a component since non-monotonic composition does not enforce all properties to be preserved when composing. Despite that this additional expressiveness is clearly needed, it implies that cases where monotony is indeed desired needs to be managed explicitly. The present paper elaborates on this topic by introducing a contracts theory embedding non-monotonic composition, and exploring conditions for ensuring monotonic composition in the context of this theory.

**Keywords:** Contracts, Non-monotonic, Composition, Satisfiability

## 1 Introduction

The notion of *contracts* was first introduced in [10] as a pair of pre- and post-conditions [8] to be used in formal specification of software (SW) *components*. More recent, general contracts theories [2,6,9,11–13] consider a wider notion of a contract applicable for components in any domain (SW, hardware, physical, etc.). In such general theories, a contract $(\mathcal{A}, \mathcal{G})$ expresses a commitment of a component to guarantee the *implementation* of a specified property $\mathcal{G}$, given that the component is *composed* with an environment implementing a specified property $\mathcal{A}$.

More concretely, despite using different terminology and notation, such general contracts theories [2,6,9,11–13] are all grounded in the following concepts:

- component $\mathsf{C}$;
- composition $\times$ on pairs of components;
- specification $\mathcal{S}$; and
- implementation relation $\models$ on components and specifications.

Defined from these concepts, a contract is a pair $(\mathcal{A}, \mathcal{G})$ of specifications and is *satisfied* by a component $\mathsf{C}$ if $\mathsf{E} \times \mathsf{C} \models \mathcal{G}$ for each component $\mathsf{E} \models \mathcal{A}$.

Another common characteristic of general contracts theories [2,6,9,11–13] is that they embed *monotonic composition* (with respect to implementation), i.e.

$$\mathsf{C} \models \mathcal{S} => \forall \mathsf{C}'.\ \mathsf{C}' \times \mathsf{C} \models \mathcal{S}\ . \tag{1}$$

An example is the theory in [2] where components and specifications both are characterized by *assertions*, i.e., sets of value sequences, over the same (universal) set of variables. Composition $\times$ and implementation $\models$ are instantiated by $\cap$ and $\subseteq$, respectively, which means that (1) translates to $\mathsf{C} \subseteq \mathcal{S} => \forall \mathsf{C}'.\ \mathsf{C}' \cap \mathsf{C} \subseteq \mathcal{S}$, which generally holds.

Despite previous general contracts theories [2,6,9,11–13] embedding monotonic composition, there is at least one strong reason why a general contracts theory by itself should not enforce (1), i.e., why a contracts theory should embed *non-monotonic composition* instead.

This reason is that theories embedding monotonic composition is inherently limited to specification of properties that are preserved through composition. Formulated differently, a theory with monotonic composition does not support specifying properties that *only* hold locally for a component since all properties are preserved when composing. This means that it is not possible to express, e.g., that a component *shall not* constrain a variable $x$ to any particular value, but that the composition of this component and another *shall* constrain $x$. In an engineering context, this could correspond to a specification expressing that "a subsystem shall not send out a particular signal" – a property that will not be preserved when composed with a component that sends out the signal.

Thus, in contrast to embedding monotonic composition, embedding non-monotonic composition in a contracts theory offers the additional expressiveness of specifying purely local properties. Despite that this additional expressiveness is clearly needed, there are many cases where we do want composition to be monotonic with respect to implementation. Since a theory embedding non-monotonic composition does not ensure this by itself, in such a theory, it is necessary to understand the conditions for when composition is monotonic and when it is not.

More explicitly, in a theory embedding non-monotonic composition, consider that the guarantee of a contract $(\mathcal{A}, \mathcal{G})$ expresses a property intended to be *global*, i.e. always preserved through composition. If a component $\mathsf{C}$ satisfying $(\mathcal{A}, \mathcal{G})$ is first composed with a component $\mathsf{E} \models \mathcal{A}$, it follows that $\mathsf{E} \times \mathsf{C} \models \mathcal{G}$; however, if $\mathsf{E} \times \mathsf{C}$ is later composed with another component $\mathsf{C}'$, it does not generally follow that $\mathsf{C}' \times \mathsf{E} \times \mathsf{C} \models \mathcal{G}$, and thus, this has to be ensured explicitly. More generally, independent of the order in which components are composed, it needs to be ensured that for each $\mathsf{E} \models \mathcal{A}$ and $\mathsf{C}$ satisfying $(\mathcal{A}, \mathcal{G})$, it holds that $\mathsf{C}' \times \mathsf{E} \times \mathsf{C} \models \mathcal{G}$, which is equivalent to

$$\forall \mathsf{C}. (\mathsf{C} \text{ satisfies } (\mathcal{A}, \mathcal{G}) \Rightarrow \mathsf{C}' \times \mathsf{C} \text{ satisfies } (\mathcal{A}, \mathcal{G}))\ . \tag{2}$$

Since (2) is not embedded in a contracts theory with non-monotonic composition, ensuring (2) needs to be done explicitly. The present paper elaborates on this topic by introducing a contracts theory embedding non-monotonic composition, and exploring conditions for ensuring (2) in the context of this theory.

The introduction of this contracts theory constitutes a first contribution. Similar to the previously exemplified theory [2], in the introduced theory, components are characterized by assertions and composition is set intersection; however, in contrast to [2], specifications are sets of assertions instead of just assertions, and implementation is instantiated by $\in$ instead of $\subseteq$. The proposed theory embeds non-monotonic composition since (1) instantiates to $\mathsf{C} \in \mathcal{S} \Rightarrow \forall \mathsf{C}'.\ \mathsf{C}' \cap \mathsf{C} \in \mathcal{S}$, which clearly does not generally hold.

As a second contribution, conditions for ensuring (2) are presented. This includes a sufficient condition, which is that component $\mathsf{C}'$ satisfies contract $(\mathcal{G}|\mathcal{G})$. It is shown that this condition is in fact also necessary if, for each $\mathsf{G} \in \mathcal{G}$, there exists a component that satisfies $(\mathcal{A}, \{\mathsf{G}\})$. Furthermore, it shown that monotonic composition with respect to composition can also be ensured by a condition based on having components $\mathsf{C}$, $\mathsf{C}'$, and each $\mathsf{E} \in \mathcal{A}$ implement special types of specifications, called *interface port specifications*, which ensure that the set of variables constrained by components $\mathsf{C}'$ and $\mathsf{E} \cap \mathsf{C}$ are respectively disjoint, and thus, that the components are isolated from each other.

As previously mentioned, general contracts theories [2, 6, 9, 11–13] all embed monotonic composition, instead of non-monotonic composition as in the contracts theory presented in this paper. However, there are contracts meta theories [1,3] that are not limited to monotonic composition. More specifically, these meta theories support instantiating theories embedding non-monotonic composition, but also contracts theories embedding monotonic composition. Thus, these meta theories are defined at a level agnostic to whether composition is monotonic or non-monotonic, which means that these meta theories naturally do not support reasoning about properties that are specific to contracts theories embedding non-monotonic composition. In contrast, the present paper studies the concept of non-monotonic composition in depth, presenting several theorems and propositions manifesting properties inherent to this concept.

The rest of the paper is organized as follows. Section 2 presents established concepts from previous contracts theories. Section 3 introduces, as a first contribution, a novel contracts theory where composition is non-monotonic with respect to implementation. Section 4 then presents, as a second contribution, conditions to ensure monotony of composition in this contracts theory. Section 5 summarizes the paper and draws conclusions.

## 2 Preliminaries

This section presents definitions already established in [13], which in turn draws on the contracts theory in [2].

Let $\Xi = \{x_1, \ldots, x_N\}$ denote the set of variables considered, and we call this the *universal set* of variables. Similarly, let $T_{\geq 0} \subseteq \mathbb{R}$ denote the universal set of time points considered. $T_{\geq 0}$ can be defined to be a continuous, e.g. if $T_{\geq 0}$ is defined by $[0, 1000]$, or discrete, e.g. if $T_{\geq 0} = \{0.1 * n | n \in \{0, 1, 2, \ldots, 10000\}\}$.

The time points in $T_{\geq 0}$ do not represent absolute time, but instead, time relative to the start of execution or observation of the system. Furthermore, $T_{\geq 0}$ contains 0 and no negative time points.

**Definition 1 (Run, see [13]).** *A run $\omega = (x_1(t), \ldots, x_N(t))$ is a vector valued function, with the elements being the variables in $\Xi$, and defined on a time subinterval $T^\omega \subseteq T_{\geq 0}$ starting at time 0, i.e. $0 \in T^\omega$.* □

For example, a run can be a *trace* [5,7,14] or an *execution* as presented in [11]. Note that two different runs may be defined on two different time intervals.

Let $\Omega$ denote the set of all possible runs.

**Definition 2 (Behavior, see [13]).** *A behavior $\mathsf{B}$ is a, possibly empty, set of runs.* □

This notion corresponds to an *assertion* as presented in [2,4].

*Example 1.* Consider a universal set of variables $\Xi = \{x, y\}$. Examples of runs are $\omega_1 = (x(t), y(t)) = (t, e^t)$ and $\omega_2 = (t, 2e^t)$ defined on an interval $T^\omega = [0, 10]$. These two runs can be combined to form four different behaviors $\mathsf{B}_1 = \{\}$, $\mathsf{B}_2 = \{(t, e^t)\}$, $\mathsf{B}_3 = \{(t, 2e^t)\}$, and $\mathsf{B}_4 = \{(t, e^t), (t, 2e^t)\}$. □

*Example 2.* Let $\Xi = \{x, y\}$ where $x$ and $y$ are Boolean variables and $T^\omega = \{0\}$ for all runs. Examples of behaviors are $\mathsf{B}_1 = \{(0, 0)\}$ , $\mathsf{B}_2 = \{(0, 1)\}$, $\mathsf{B}_3 = \{(0, 0), (0, 1)\}$, and $\mathsf{B}_4 = \{(0, 0), (1, 1)\}$. □

Let $\omega[X]$ denote the subvector of $\omega$ having the variables $X \subseteq \Xi$. For example, if $\Xi = \{a, b, c, d\}$, then $\omega[\{b, d\}] = (b, d)$. If $X = \emptyset$, then let $\omega[X] = \emptyset$.

**Definition 3 (Projection, see [13]).** *The* projection *of behavior $\mathsf{B}$ onto a set of variables $X \subseteq \Xi$, denoted $\mathrm{proj}_X \mathsf{B}$, is the set of runs:*

$$\{\omega \in \Omega \mid \exists \nu \in \mathsf{B}. \nu[X] = \omega[X]\} . \qquad \Box$$

Note that projection is called *extended projection* in [13] and is defined using slightly different notation. Note also that if $\mathsf{B} \neq \emptyset$, then $\mathrm{proj}_\emptyset \mathsf{B} = \Omega$ and $\mathrm{proj}_\Xi \mathsf{B} = \mathsf{B}$. Furthermore, for any variable set $Y$, it holds $\mathrm{proj}_Y \emptyset = \emptyset$ and $\mathrm{proj}_Y \Omega = \Omega$.

*Example 3.* For the behaviors defined in Example 2, it holds that $\mathrm{proj}_{\{x\}} \mathsf{B}_1 = \mathrm{proj}_{\{x\}} \mathsf{B}_2 = \{(0, 0), (0, 1)\}$ and $\mathrm{proj}_{\{x\}} \mathsf{B}_3 = \mathsf{B}_3$, and $\mathrm{proj}_{\{x\}} \mathsf{B}_4 = \Omega$. □

Let $\{x\}^\mathsf{C}$ denote the complement set of $\{x\}$, i.e. $\{x\}^\mathsf{C} = \Xi \setminus \{x\}$.

**Definition 4 (Behavior Constrains, see [13]).** *A behavior $\mathsf{B}$* constrains *a variable $x$ if*

$$\mathrm{proj}_{\{x\}^\mathsf{C}} \mathsf{B} \neq \mathsf{B} . \qquad \Box$$

Let $X_\mathsf{B}$ denote *the set of variables constrained by* $\mathsf{B}$. Note that $X_\Omega = \emptyset$ and $X_\emptyset = \emptyset$.

*Example 4.* For the behaviors discussed in Example 2 and Example 3, it holds $\{y\}^{\mathsf{C}} = \{x\}$. Furthermore, $\mathrm{proj}_{\{y\}^{\mathsf{C}}} \mathsf{B}_1 = \mathrm{proj}_{\{x\}} \mathsf{B}_1 \neq \mathsf{B}_1$. Therefore $\mathsf{B}_1$ constrains $x$. In fact, $\mathsf{B}_1$ also constrains $y$, and therefore $X_{\mathsf{B}_1} = \{x, y\}$.

Since $\mathrm{proj}_{\{x\}} \mathsf{B}_3 = \mathsf{B}_3$, it holds that $\mathsf{B}_3$ does not constrain $y$. However, behavior $\mathsf{B}_3$ does constrain $x$ since $\mathrm{proj}_{\{y\}} \mathsf{B}_3 = \{(0,0),(0,1),(1,0),(1,1)\} \neq \mathsf{B}_3$. □

## 3 Composition, Specifications, and Contracts

Based upon the definition of behavior in Section 2 and as the first contribution of the paper, this section presents the basis of a contracts theory in which specifications are represented as sets of behaviors. The achievement is a theory where composition is non-monotonous with respect to implementation of specifications.

### 3.1 Composition

We assume that each component is characterized by a behavior. When two components, characterized by behaviors $\mathsf{B}_1$ and $\mathsf{B}_2$ respectively, are put together, we assume that a new behavior, called *the composition of* $\mathsf{B}_1$ *and* $\mathsf{B}_2$, is formed and that it conforms to the following definition.

**Definition 5 (Composition of Behaviors).** *The* composition *of two behaviors* $\mathsf{B}_1$ *and* $\mathsf{B}_2$ *is their intersection* $\mathsf{B}_1 \cap \mathsf{B}_2$. □

### 3.2 Specifications

A *specification* $\mathcal{S}$ expresses an intended property of a component. Commonly such an intended property is called a 'requirement'. If a behavior *implements* a specification, then this really means that the component characterized by the behavior has the intended property expressed by $\mathcal{S}$. These notions are now formalized in the following two definitions.

**Definition 6 (Specification).** *A* specification $\mathcal{S}$ *is a, possibly empty, set of behaviors.* □

**Definition 7 (Behavior Implements Specification).** *A behavior* $\mathsf{B}$ implements *a specification* $\mathcal{S}$ *if* $\mathsf{B} \in \mathcal{S}$. □

Note that the empty set $\emptyset$ can be a specification, and considering that $\emptyset$ can also be a behavior, the set $\{\emptyset\}$ can be a specification.

*Example 5.* Let $\Xi = \{x, y\}$ and $T^\omega = \{0\}$. Then

$$\mathcal{S} = \{\{(0,0),(0,1)\},\{(0,1)\},\{(0,0),(1,1)\}\}$$

is a possible specification. Behavior $\mathsf{B}_3 = \{(0,0),(0,1)\}$ implements $\mathcal{S}$ and also $\mathsf{B}_4 = \{(0,0),(1,1)\}$ implements $\mathcal{S}$. However, the composition $\mathsf{B}_3 \cap \{(0,0),(1,1)\} = \{(0,0)\}$ does not implement $\mathcal{S}$. □

Example 5 highlights the fact that even though the two behaviors $\mathsf{B}_3$ and $\mathsf{B}_3$ individually implement the specification $\mathcal{S}$, their composition does not. That is, composition is non-monotonic with respect to implementation of specification.

## 3.3 Contracts

In alignment with the notion of *contract* and *satisfy* as introduced in Section 1, this section presents the instantiation resulting from the definitions of specification and implementation presented in Section 3.2.

A contract is a pair $(\mathcal{A}, \mathcal{G})$ of specifications expressing an intended property of a composition of two behaviors $\mathsf{E}$ and $\mathsf{B}$ where $\mathsf{E} \in \mathcal{A}$.

**Definition 8 (Contract).** *A* contract *is a pair $(\mathcal{A}, \mathcal{G})$ of specifications, where:*

*(i) $\mathcal{G}$ is non-empty and called* guarantee*; and*
*(ii) $\mathcal{A}$ is possibly empty and called* assumption*.* □

**Definition 9 (Behavior Satisfies Contract).** *A behavior* $\mathsf{B}$ satisfies *a contract $(\mathcal{A}, \mathcal{G})$ if, for each $\mathsf{E}$ implementing $\mathcal{A}$, it holds that the composition $\mathsf{E} \cap \mathsf{B}$ implements $\mathcal{G}$.* □

If the instantiations of *specification* and *implements* are written out, we obtain that a behavior $\mathsf{B}$ satisfies a contract $(\mathcal{A}, \mathcal{G})$ if, and only if,

$$\forall \mathsf{A} \in \mathcal{A}.\ \mathsf{A} \cap \mathsf{B} \in \mathcal{G} \tag{3}$$

Note that a contract $(\mathcal{A}, \mathcal{G})$ is trivially satisfied if $\mathcal{A} = \emptyset$.

## 3.4 Specification is Agnostic to Set of Variables

As discussed above, a consequence of Definitions 6 and 7 of *specification* and *implements* is that composition is not inherently monotonic with respect to implementation. However, in many cases monotony is indeed desirable. One such case is when a specification represents an intended relation on a proper subset of the universal set $\Xi$. An example is when $\mathcal{S}_1$ expresses an intended property $x = 0$ and where $x \subset \Xi$. Consider a first component that itself implements $\mathcal{S}_1$ by setting $x = 0$ and not constraining any other variables. If this first component is composed with a second component that does not in any way constrain $x$, then we should expect that their composition also implements $\mathcal{S}_1$. As a fundamental concept to support such reasoning, we introduce the notion of *agnostic specification*:

**Definition 10 (Specification is Agnostic to Set of Variables).** *A specification $\mathcal{S}$ is* agnostic to a set of variables $X$ *if*

$$\forall \mathsf{B} \in \mathcal{S}.\ (\forall \mathsf{B}'.\ \mathrm{proj}_{X^{\mathrm{c}}} \mathsf{B}' = \mathrm{proj}_{X^{\mathrm{c}}} \mathsf{B} \Rightarrow \mathsf{B}' \in \mathcal{S})\ . \qquad \square$$

A specification is agnostic to a set of variables $X$ if the fact whether or not a behavior implements the specification is independent on whether or not the behavior constrains variables in $X$.

*Example 6.* Let $\Xi = \{x, y\}$ where $x$ and $y$ are Boolean variables and $T^\omega = \{0\}$. Two examples of specifications that are agnostic to $\{y\}$ are:

– $\{\{(0,0)\},\{(0,1)\},\{(0,0),(0,1)\}\}$
– $\{\{(0,0)\},\{(0,1)\},\{(0,0),(0,1)\},\{(1,1)\},\{(1,0)\},\{(1,0),(1,1)\}\}$.

Any proper subset of these behavior sets are *not* agnostic to $\{y\}$. Three such examples are:

– $\{\{(0,0)\}\}$
– $\{\{(0,0)\},\{(0,1)\}\}$
– $\{\{(0,0),(0,1),(1,0),(1,1)\}\}$. □

Now, to illustrate how the concept of agnostic specification can be used to ensure monotonic composition with respect to implementation of specification, consider again the previously introduced specification $\mathcal{S}_1$. Assume that $\mathcal{S}_1$, expressed by $x=0$, is agnostic to $\{y\}$. Since the first component implements $\mathcal{S}_1$, it holds $\mathsf{B}_1 \in \mathcal{S}_1$. Also, since the first component does not constrain any other variables than $x$, it holds that $\mathsf{B}_1$ does not constrain $y$. For any behavior $\mathsf{B}_2$ constraining *only* $y$, it holds $\mathrm{proj}_{\{x\}}\,\mathsf{B}_2 = \Omega$ (see Proposition 2 in Appendix A). Next, we will use the following proposition with proof given in Appendix A:

**Proposition 1.** *Let $\mathsf{B}_1$ and $\mathsf{B}_2$ be two behaviors such that $X_{\mathsf{B}_1} \cap X_{\mathsf{B}_2} = \emptyset$. Then, for an arbitrary set of variables $Y$, it holds*

$$\mathrm{proj}_Y(\mathsf{B}_1 \cap \mathsf{B}_2) = \mathrm{proj}_Y\,\mathsf{B}_1 \cap \mathrm{proj}_Y\,\mathsf{B}_2\ . \qquad \square$$

By using Proposition 1, we derive that

$$\mathrm{proj}_{\{x\}}(\mathsf{B}_1 \cap \mathsf{B}_2) = \mathrm{proj}_{\{x\}}\,\mathsf{B}_1 \cap \mathrm{proj}_{\{x\}}\,\mathsf{B}_2 = \mathrm{proj}_{\{x\}}\,\mathsf{B}_1 \cap \Omega = \mathrm{proj}_{\{x\}}\,\mathsf{B}_1 \quad (4)$$

Specification $\mathcal{S}_1$ being agnostic to $\{y\}$ means according to Definition 10 that

$$\forall \mathsf{B}'.\ \mathrm{proj}_{\{x\}}\,\mathsf{B}' = \mathrm{proj}_{\{x\}}\,\mathsf{B}_1 \Rightarrow \mathsf{B}' \in \mathcal{S}_1 \quad (5)$$

Combining (4) and (5), implies that the composition $\mathsf{B}' = \mathsf{B}_1 \cap \mathsf{B}_2 \in \mathcal{S}_1$.

In summary, the fact is that since $\mathcal{S}_1$, expressing $x=0$, is agnostic to $y$, the composition of any behavior in $\mathcal{S}_1$ with any other behavior that only constrains $y$ will result in a behavior also in $\mathcal{S}_1$. This fact will be generalized in Section 4.3 presenting a similar condition, but in terms of satisfying a contract instead of implementing a specification.

## 4 Preserving Contract Satisfiability Under non-Monotonic Composition

Section 3 presented a contracts theory where components are characterized by behaviors, composition is set intersection, specifications are behavior sets, and implementation is defined by $\in$. As previously mentioned in Section 1, the proposed theory embeds non-monotonic composition since (1) instantiates to $\mathsf{B} \in \mathcal{S} => \forall \mathsf{B}'.\ \mathsf{B}' \cap \mathsf{B} \in \mathcal{S}$, which clearly does not generally hold. As also stated in Section 1, this means that it follows that (2) also does not hold, i.e., it does not hold that

$$\forall \mathsf{B}\ .\ \mathsf{B} \text{ satisfies } (\mathcal{A}, \mathcal{G}) \Rightarrow \mathsf{B}' \cap \mathsf{B} \text{ satisfies } (\mathcal{A}, \mathcal{G})\ . \tag{6}$$

However, ensuring (6) is desirable for any use-case where $\mathcal{G}$ expresses a property that is intended to be implemented by a composition of $\mathsf{B}'$ and the composition $\mathsf{B} \cap \mathsf{E}$ of a component $\mathsf{B}$ satisfying $(\mathcal{A}, \mathcal{G})$ and a component $\mathsf{E} \in \mathcal{A}$.

In order to be able to support such use-cases, this section now proceeds to present conditions for ensuring (6). This includes a sufficient and necessary condition of (6); however, to gradually introduce this condition to the reader, a condition is first presented that is only sufficient and not necessary, but that is less technical and directly follows from the necessary and sufficient condition.

## 4.1 Sufficient Condition for Preserving Satisfiability

Consider constructing a contract $(\mathcal{A}', \mathcal{G}')$ for component $\mathsf{B}'$ such that (6) is ensured if $\mathsf{B}'$ satisfies the contract. Since $\mathcal{G}'$ is intended to express a property of the composition of $\mathsf{B}'$ and a behavior $\mathsf{A} \in \mathcal{A}'$, and $\mathcal{G}$ is to be implemented by the composition, it makes sense that $\mathcal{G}' = \mathcal{G}$. Regarding the assumption $\mathcal{A}'$, to enforce that the composition of $\mathsf{B}'$ and any behavior implementing $\mathcal{G}$ also implements $\mathcal{G}$, each behavior implementing $\mathcal{G}$ should be in $\mathcal{A}'$. Since $\mathcal{G}$ contains each behavior implementing it, it also makes sense that $\mathcal{A}' = \mathcal{G}$. Indeed, as shown in the following corollary, if $\mathsf{B}'$ satisfies $(\mathcal{G}, \mathcal{G})$, then (6) holds.

**Corollary 1.** *Given a contract* $(\mathcal{A}, \mathcal{G})$*, if a behavior* $\mathsf{B}'$ *satisfies* $(\mathcal{G}, \mathcal{G})$*, then*

$$\forall \mathsf{B}\ .\ \mathsf{B} \textit{ satisfies } (\mathcal{A}, \mathcal{G}) \Rightarrow \mathsf{B}' \cap \mathsf{B} \textit{ satisfies } (\mathcal{A}, \mathcal{G})\ .$$

*Proof.* This corollary follows from Theorem 1 in Section 4.2 since Definition 11 implies that $\mathcal{R}_{\mathcal{G}}(\mathcal{A}, \mathcal{G}) \subseteq \mathcal{G}$, which means, in accordance with (3), that: $\mathsf{B}'$ satisfies $(\mathcal{G}, \mathcal{G}) \Rightarrow \mathsf{B}'$ satisfies $(\mathcal{R}_{\mathcal{G}}(\mathcal{A}, \mathcal{G}), \mathcal{G})$. □

*Example 7.* Let $\Xi = \{x, y\}$ where $x$ and $y$ are Boolean variables and $T^{\omega} = \{0\}$. Consider a contract $(\mathcal{A}, \mathcal{G})$ where $\mathcal{A} = \{\mathsf{A}_1, \mathsf{A}_2\} = \{\{(0,0)\}, \{(0,0), (0,1)\}\}$ and $\mathcal{G} = \{\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3\} = \{\{(0,0)\}, \{(1,0)\}, \{(0,0), (1,0)\}\}$.

Behavior $\mathsf{B} = \{(0,0), (1,1)\}$ satisfies the contract $(\mathcal{A}, \mathcal{G})$ since

$$\begin{aligned}
\mathsf{A}_1 \cap \mathsf{B} &= \{(0,0)\} \cap \{(0,0), (1,1)\} = \{(0,0)\} = \mathsf{G}_1 \in \mathcal{G} \\
\mathsf{A}_2 \cap \mathsf{B} &= \{(0,0), (0,1)\} \cap \{(0,0), (1,1)\} = \{(0,0)\} = \mathsf{G}_1 \in \mathcal{G}\ .
\end{aligned}$$

Behavior $\mathsf{B}'_1 = \{(0,0), (1,0), (1,1)\}$ satisfies the contract $(\mathcal{G}, \mathcal{G})$ since

$$\begin{aligned}
\mathsf{G}_1 \cap \mathsf{B}'_1 &= \{(0,0)\} \cap \{(0,0), (1,0), (1,1)\} = \{(0,0)\} = \mathsf{G}_1 \in \mathcal{G} \\
\mathsf{G}_2 \cap \mathsf{B}'_! &= \{(1,0)\} \cap \{(0,0), (1,0), (1,1)\} = \{(1,0)\} = \mathsf{G}_2 \in \mathcal{G} \\
\mathsf{G}_3 \cap \mathsf{B}'_1 &= \{(0,0), (1,0)\} \cap \{(0,0), (1,0), (1,1)\} = \{(0,0), (1,0)\} = \mathsf{G}_3 \in \mathcal{G}\ .
\end{aligned}$$

Then, in agreement with Corollary 1, composition $\mathsf{B}'_1 \cap \mathsf{B}$ satisfies $(\mathcal{A}, \mathcal{G})$ since

$$\mathsf{A}_1 \cap \mathsf{B}'_1 \cap \mathsf{B} = \{(0,0)\} \cap \{(0,0),(1,0),(1,1)\} \cap \{(0,0),(1,1)\} = \{(0,0)\} = \mathsf{G}_1 \in \mathcal{G}$$
$$\mathsf{A}_2 \cap \mathsf{B}'_1 \cap \mathsf{B} = \{(0,0),(0,1)\} \cap \{(0,0),(1,0),(1,1)\} \cap \{(0,0),(1,1)\} = \{(0,0)\} = \mathsf{G}_1 \in \mathcal{G} \ .$$

On the other hand, the behavior $\mathsf{B}'_2 = \{(0,1),(1,0),(1,1)\}$ does not satisfy the contract $(\mathcal{G}, \mathcal{G})$ since

$$\mathsf{G}_1 \cap \mathsf{B}'_2 = \{(0,0)\} \cap \{(0,1),(1,0),(1,1)\} = \emptyset \notin \mathcal{G} \ .$$

Also so we see that the composition $\mathsf{B}'_2 \cap \mathsf{B}$ does not satisfy $(\mathcal{A}, \mathcal{G})$ since

$$\mathsf{A}_1 \cap \mathsf{B}'_2 \cap \mathsf{B} = \{(0,0)\} \cap \{(0,1),(1,0),(1,1)\} \cap \{(0,0),(1,1)\} = \emptyset \notin \mathcal{G} \ . \quad \square$$

## 4.2 Necessary and Sufficient Condition for Preserving Satisfiability

Section 4.1 presented Corollary 1 expressing a condition for ensuring (6). As previously mentioned, this condition is a sufficient, but not necessary condition of (6).

This is due to the fact that the assumption in the contract $(\mathcal{G}, \mathcal{G})$ may include behaviors that cannot be a composition of a behavior satisfying $(\mathcal{A}, \mathcal{G})$ and a behavior $\mathsf{A} \in \mathcal{A}$. The subset of $\mathcal{G}$ obtained by removing such behaviors from $\mathcal{G}$ is here called the *realizable guarantee of* $(\mathcal{A}, \mathcal{G})$.

**Definition 11 (Realizable Guarantee of Contract).** *The* realizable guarantee of a contract $(\mathcal{A}, \mathcal{G})$*, denoted* $\mathcal{R}_\mathcal{G}(\mathcal{A}, \mathcal{G})$*, is the specification* $\{\mathsf{A} \cap \mathsf{B} \mid \mathsf{A} \in \mathcal{A} \wedge \mathsf{B}$ *satisfies* $(\mathcal{A}, \mathcal{G})\}$. $\square$

As expressed in the following proposition, contracts $(\mathcal{A}, \mathcal{G})$ and $(\mathcal{A}, \mathcal{R}_\mathcal{G}(\mathcal{A}, \mathcal{G}))$ are satisfied by the exact same behaviors.

**Proposition 2.** *A behavior satisfies a contract* $(\mathcal{A}, \mathcal{G})$ *if and only if the behavior satisfies* $(\mathcal{A}, \mathcal{R}_\mathcal{G}(\mathcal{A}, \mathcal{G}))$.

*Proof.* Follows directly from (3) and Definition 11. $\square$

Having $\mathsf{B}'$ satisfy $(\mathcal{R}_\mathcal{G}(\mathcal{A}, \mathcal{G}), \mathcal{G})$ instead of $(\mathcal{G}, \mathcal{G})$ is indeed sufficient for ensuring (6). The behaviors in $\mathcal{G} \setminus \mathcal{R}_\mathcal{G}(\mathcal{A}, \mathcal{G})$ are not compositions of a behavior $\mathsf{B}$ satisfying $(\mathcal{A}, \mathcal{G})$ and a behavior $\mathsf{A} \in \mathcal{A}$; therefore, these behaviors can be excluded from the assumption of $(\mathcal{G}, \mathcal{G})$ since the implication in (6) is trivially true for any behavior $\mathsf{B}$ that does not satisfy $(\mathcal{A}, \mathcal{G})$. The following theorem not only shows that it is indeed the case that $\mathsf{B}'$ satisfying $(\mathcal{R}_\mathcal{G}(\mathcal{A}, \mathcal{G}), \mathcal{G})$ is sufficient for ensuring (6), but that this is in fact also a necessary condition.

**Theorem 1.** *Given a contract* $(\mathcal{A}, \mathcal{G})$*, a behavior* $\mathsf{B}'$ *satisfies* $(\mathcal{R}_\mathcal{G}(\mathcal{A}, \mathcal{G}), \mathcal{G})$ *if and only if:*

$$\forall \mathsf{B} \ . \ \mathsf{B} \textit{ satisfies } (\mathcal{A}, \mathcal{G}) \Rightarrow \mathsf{B}' \cap \mathsf{B} \textit{ satisfies } (\mathcal{A}, \mathcal{G}) \ . \tag{7}$$

The proof of Theorem 1 follows after the following lemma.

**Lemma 1.** *A behavior satisfies a contract* $(\mathcal{A}_1 \cup \ldots \cup \mathcal{A}_N, \mathcal{G})$ *if the behavior satisfies each contract* $(\mathcal{A}_i, \mathcal{G})$.

*Proof of Lemma 1.* In accordance with (3), consider as given a behavior $\mathsf{B}$ such that $\forall \mathsf{A}_i \in \mathcal{A}_i . \mathsf{A}_i \cap \mathsf{B} \in \mathcal{G}$. It follows directly that $\forall \mathsf{A} \in (\mathcal{A}_1 \cup \ldots \cup \mathcal{A}_N) . \mathsf{A} \cap \mathsf{B} \in \mathcal{G}$, which means that $\mathsf{B}$ satisfies $(\mathcal{A}_1 \cup \ldots \cup \mathcal{A}_N, \mathcal{G})$ in accordance with (3). □

*Proof of Theorem 1.* For the if-only case, in accordance with (3), consider as given

$$\forall \mathsf{A} \in \mathcal{R}_{\mathcal{G}}(\mathcal{A}, \mathcal{G}) . \mathsf{A} \cap \mathsf{B}' \in \mathcal{G} \ . \tag{8}$$

Consider an arbitrary behavior $\mathsf{B}$ that satisfies $(\mathcal{A}, \mathcal{G})$, which means that $\forall \mathsf{A} \in \mathcal{A} . \mathsf{A} \cap \mathsf{B} \in \mathcal{R}_{\mathcal{G}}(\mathcal{A}, \mathcal{G})$ in accordance with Proposition 2 and (3). This and (8) imply $\forall \mathsf{A} \in \mathcal{A} . \mathsf{A} \cap \mathsf{B} \cap \mathsf{B}' \in \mathcal{G}$, which means that $\mathsf{B}' \cap \mathsf{B}$ satisfies $(\mathcal{A}, \mathcal{G})$ in accordance with (3). This also holds for each $\mathsf{B}$ satisfying $(\mathcal{A}, \mathcal{G})$ since $\mathsf{B}$ characterizes an arbitrary behavior, and (7) hence holds.

For the if case, consider that (7) holds. It follows in accordance with (3) that

$$\forall \mathsf{B} . \, (\mathsf{B} \text{ satisfies } (\mathcal{A}, \mathcal{G}) \Rightarrow (\forall \mathsf{A}' \in \mathcal{A} . \mathsf{A}' \cap \mathsf{B}' \cap \mathsf{B} \in \mathcal{G})) \ .$$

Notably, $\forall \mathsf{A}' \in \mathcal{A} . \mathsf{A}' \cap \mathsf{B}' \cap \mathsf{B} \in \mathcal{G}$ can be rewritten as $\forall \mathsf{A} \in \{\mathsf{A}' \cap \mathsf{B} \mid \mathsf{A}' \in \mathcal{A}\} . \mathsf{A} \cap \mathsf{B}' \in \mathcal{G}$, which in accordance with (3) means that

$$\forall \mathsf{B} . \mathsf{B} \text{ satisfies } (\mathcal{A}, \mathcal{G}) \Rightarrow \mathsf{B}' \text{ satisfies } (\{\mathsf{A} \cap \mathsf{B} \mid \mathsf{A} \in \mathcal{A}\}, \mathcal{G}) \ .$$

This and Lemma 1 imply that $\mathsf{B}'$ satisfies

$$\left( \bigcup_{\mathsf{B}, \mathsf{B} \text{ satisfies } (\mathcal{A}, \mathcal{G})} \{\mathsf{A} \cap \mathsf{B} \mid \mathsf{A} \in \mathcal{A}\} , \mathcal{G} \right) ,$$

which is equivalent to $\mathsf{B}'$ satisfying $(\{\mathsf{A} \cap \mathsf{B} \mid \mathsf{A} \in \mathcal{A}, \mathsf{B} \text{ satisfies } (\mathcal{A}, \mathcal{G})\}, \mathcal{G})$. In accordance with Definition 11, this means that $\mathsf{B}'$ satisfies $(\mathcal{R}_{\mathcal{G}}(\mathcal{A}, \mathcal{G}), \mathcal{G})$, which ends the proof. □

### 4.3 Preserving Satisfiability through Interface Port Specifications

Similar to Sections 4.1 and 4.2, this section presents a condition for preserving contract satisfiability; however, in contrast to the conditions presented in Sections 4.1 and 4.2, rather than ensuring (6), this condition is sufficient for

$$\forall \mathsf{B} \in \mathcal{I}_X \ . \ \mathsf{B} \text{ satisfies } (\mathcal{A}, \mathcal{G}) \Rightarrow \mathsf{B}' \cap \mathsf{B} \text{ satisfies } (\mathcal{A}, \mathcal{G}) \ , \tag{9}$$

where $\mathcal{I}_X = \{\mathsf{B} \mid X_{\mathsf{B}} \subseteq X\}$. In contrast to (6), which quantifies over each behavior, formula (9) quantifies over each behavior $\mathsf{B}$ that implements a special type of specification, here denoted $\mathcal{I}_X$ and called *interface port specification*, which expresses only a restriction on the set of variables that $\mathsf{B}$ can constrain.

**Definition 12 (Interface Port Specification).** *An* interface port specification for $X$ *is a specification* $\{\mathsf{B} \mid X_\mathsf{B} \subseteq X\}$. $\square$

Prior to presenting a theorem ensuring (9), an example will be studied.

*Example 8.* Consider $\Xi = \{x, y, u, v\}$ where $x$, $y$, $u$, and $v$ are Boolean variables and $T^w = \{0\}$. Let $\mathsf{B}'$ be a behavior and $(\mathcal{A}, \mathcal{G})$ a contract where:

a) $\mathsf{B}' = \{(x, y, u, v) \in \Omega \mid u = v\}$;
b) $\mathcal{G} = \{\mathsf{B} \neq \emptyset \mid \forall(x, y, u, v) \in \mathsf{B}.\ x = 1\}$; and
c) $\mathcal{A} = \{\mathsf{B} | \ \forall(x, y, u, v) \in \mathsf{B}.\ y = 1\} \cap \mathcal{I}_{\{x,y\}}$.

Consider a behavior $\mathsf{B} = \{(x, y, u, v) \in \Omega \mid x = y\}$, and an arbitrary behavior $\mathsf{A} \in \mathcal{A}$. Their intersection $\mathsf{A} \cap \mathsf{B}$ is behavior $\{(x, y, u, v) \in \Omega \mid x = y = 1\}$, which implements $\mathcal{G}$. This means that in order for (9) to hold, behavior $\mathsf{B}' \cap \mathsf{A} \cap \mathsf{B}$ must also implement $\mathcal{G}$. This is indeed the case since $\mathsf{B}' \cap \mathsf{A} \cap \mathsf{B} = \{(x, y, u, v) \in \Omega \mid x = y = 1 \text{ and } u = v\}$, contained in $\mathcal{G}$. $\square$

In accordance with Definitions 10 and 12, conditions (a)-(c) in Example 8 respectively imply:

i) $\mathsf{B}' \in \mathcal{I}_{\{u,v\}}$;
ii) $\mathcal{G}$ is agnostic to $\{u, v, y\} \supseteq \{u, v\}$ and $\emptyset \notin \mathcal{G}$; and
iii) $\mathcal{A} \subseteq \mathcal{I}_{\{u,v\}^\mathsf{C}}$.

As will be shown in the following theorem, conditions (i)-(iii) ensure that if a behavior $\mathsf{B}$ satisfies $(\mathcal{A}, \mathcal{G})$ and implements an interface specification for a set disjoint to $\{u, v\}$, e.g. $\{x, y\}$ as in Example 8, then it holds that $\mathsf{B}' \cap \mathsf{B}$ satisfies $(\mathcal{A}, \mathcal{G})$.

**Theorem 2.** *Given a contract $(\mathcal{A}, \mathcal{G})$ and two disjoint sets of variables $X$ and $Y$, if*

*i)* $\mathsf{B}' \in \mathcal{I}_Y$
*ii)* $\mathcal{G}$ *is agnostic to a set* $Z \supseteq Y$ *and* $\emptyset \notin \mathcal{G}$*; and*
*iii)* $\mathcal{A} \subseteq \mathcal{I}_{Y^\mathsf{C}}$*,*

*then it holds:* $\quad \forall \mathsf{B} \in \mathcal{I}_X\ .\ \mathsf{B}$ *satisfies* $(\mathcal{A}, \mathcal{G}) \Rightarrow \mathsf{B}' \cap \mathsf{B}$ *satisfies* $(\mathcal{A}, \mathcal{G})$.

The lemmas used in the following proof can be found in Appendix A.

*Proof.* Assume (i)-(iii), and an arbitrary behavior $\mathsf{B} \in \mathcal{I}_X$ satisfying $(\mathcal{A}, \mathcal{G})$, i.e., $\forall \mathsf{A} \in \mathcal{A}. \mathsf{A} \cap \mathsf{B} \in \mathcal{G}$ in accordance with (3). From (i), in accordance with Definition 12, it follows that $X_{\mathsf{B}'} \subseteq Y$. In accordance with Lemma 4, this and (ii) imply that $\mathcal{G}$ is agnostic to $X_{\mathsf{B}'}$. For an arbitrary $\mathsf{A} \in \mathcal{A}$, in accordance with Definition 10 and since $\mathsf{A} \cap \mathsf{B} \in \mathcal{G}$, it follows that

$$\left(\mathrm{proj}_{X^\mathsf{C}_{\mathsf{B}'}}(\mathsf{B}' \cap (\mathsf{A} \cap \mathsf{B})) = \mathrm{proj}_{X^\mathsf{C}_{\mathsf{B}'}}(\mathsf{A} \cap \mathsf{B})\right) \Rightarrow \mathsf{B}' \cap (\mathsf{A} \cap \mathsf{B}) \in \mathcal{G}\ . \qquad (10)$$

In accordance with Definition 12, $\mathsf{B} \in \mathcal{I}_X$ and (iii) imply $X_{\mathsf{B}} \subseteq X$ and $X_{\mathsf{A}} \subseteq Y^{\mathsf{C}}$, respectively. Since $X \cap Y = \emptyset$ was given, it follows that $X_{\mathsf{B}} \subseteq Y^{\mathsf{C}}$, and further that $X_{\mathsf{A}} \cup X_{\mathsf{B}} \subseteq Y^{\mathsf{C}}$. Then in accordance with Lemma 5 and since $\mathsf{A} \cap \mathsf{B} \neq \emptyset$ due to $\emptyset \notin \mathcal{G}$ in (ii), it holds that $X_{\mathsf{A}\cap\mathsf{B}} \subseteq Y^{\mathsf{C}}$, which means that $X_{\mathsf{A}\cap\mathsf{B}} \cap X_{\mathsf{B}'} = \emptyset$ since $X_{\mathsf{B}'} \subseteq Y$. In accordance with Proposition 1, this means that (10) is equivalent to

$$\Big(\mathrm{proj}_{X^{\mathsf{C}}_{\mathsf{B}'}}(\mathsf{B}') \cap \mathrm{proj}_{X^{\mathsf{C}}_{\mathsf{B}'}}(\mathsf{A} \cap \mathsf{B}) = \mathrm{proj}_{X^{\mathsf{C}}_{\mathsf{B}'}}(\mathsf{A} \cap \mathsf{B})\Big) \Rightarrow \mathsf{B}' \cap (\mathsf{A} \cap \mathsf{B}) \in \mathcal{G} \ . \quad (11)$$

In accordance with Lemma 2, it holds that $\mathrm{proj}_{X^{\mathsf{C}}_{\mathsf{B}'}}(\mathsf{B}') = \Omega$, which means that the left side of the implication in (11) is equivalent to $\Omega \cap \mathrm{proj}_{X^{\mathsf{C}}_{\mathsf{B}'}}(\mathsf{A} \cap \mathsf{B}) = \mathrm{proj}_{X^{\mathsf{C}}_{\mathsf{B}'}}(\mathsf{A} \cap \mathsf{B})$. Since $\Omega$ includes all runs, it follows that $\mathrm{proj}_{X^{\mathsf{C}}_{\mathsf{B}'}}(\mathsf{A} \cap \mathsf{B}) = \mathrm{proj}_{X^{\mathsf{C}}_{\mathsf{B}'}}(\mathsf{A} \cap \mathsf{B})$, which means that it follows that the right side of the implication in (11) holds, i.e., $\mathsf{B}' \cap (\mathsf{A} \cap \mathsf{B}) \in \mathcal{G}$. Since $\mathsf{A}$ characterizes an arbitrary behavior in $\mathcal{A}$, in accordance with (3), $\mathsf{B}' \cap \mathsf{B}$ satisfies $(\mathcal{A}, \mathcal{G})$. Since $\mathsf{B}$ characterizes an arbitrary behavior $\mathsf{B} \in \mathcal{I}_X$ satisfying $(\mathcal{A}, \mathcal{G})$, it follows that $\forall \mathsf{B} \in \mathcal{I}_X$ . $\mathsf{B}$ satisfies $(\mathcal{A}, \mathcal{G}) \Rightarrow \mathsf{B}' \cap \mathsf{B}$ satisfies $(\mathcal{A}, \mathcal{G})$, which ends the proof. □

*Example 9.* Consider contract $(\mathcal{A}, \mathcal{G})$ and behavior $\mathsf{B}'$ from Example 8. As previously mentioned, it holds that: i) $\mathsf{B}' \in \mathcal{I}_{\{u,v\}}$; ii) $\mathcal{G}$ is agnostic to $\{u, v, y\} \supseteq \{u, v\}$ and $\emptyset \notin \mathcal{G}$; and iii) $\mathcal{A} \subseteq \mathcal{I}_{\{u,v\}^{\mathsf{C}}}$. In accordance with Theorem 2, to establish that the composition of $\mathsf{B}'$ and a behavior $\mathsf{B}$ satisfies contract $(\mathcal{A}, \mathcal{G})$, it suffices to show that $\mathsf{B}$ satisfies $(\mathcal{A}, \mathcal{G})$ and implements an interface specification for a subset of $\{x, y\}$. An example of such a behavior is $\mathsf{B} = \{(x, y, u, v) \in \Omega \mid x = y\}$, presented in Example 8. □

## 5 Conclusion

As argued for, and exemplified in Section 1, to support properties that may be invalidated through composition, a contracts theory should not have composition inherently monotonic with respect to implementation. Therefore, the paper has presented, as a first contribution, the basis for a novel contracts theory where composition is non-monotonic with respect to implementation. The key solution is that specifications are represented as sets of sets of runs instead of only sets of runs as in previous theories .

The general support from non-monotony implies that cases where monotony is indeed desired needs to be managed explicitly. To support this, the paper has, as a second contribution, presented Corollary 1 and Theorem 1 and 2, which each provides conditions to ensure monotony of composition with respect to implementation when indeed desired.

## References

1. Bauer, S., David, A., Hennicker, R., Guldstrand Larsen, K., Legay, A., Nyman, U., WÄ...sowski, A.: Moving from Specifications to Contracts in Component-Based

Design. In: Lara, J., Zisman, A. (eds.) Fundamental Approaches to Software Eng., Lec. Notes in Computer Science, vol. 7212, pp. 43–58. Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-28872-2$_$3

2. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple Viewpoint Contract-Based Specification and Design. In: Boer, F.S., Bonsangue, M.M., Graf, S., Roever, W.P. (eds.) Formal Methods for Components and Object, pp. 200–225. Springer-Verlag, Berlin, Heidelberg (2008), http://dx.doi.org/1multiviewPoint0.1007/978-3-540-92188-2_9
3. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J.B., Reinkemeier, P., Sangiovanni-Vincentelli, A., Damm, W., Henzinger, T., Larsen, K.G.: Contracts for System Design. Rapport de recherche RR-8147, INRIA (Nov 2012), http://hal.inria.fr/hal-00757488
4. Benveniste, A., Caillaud, B., Passerone, R.: Multi-Viewpoint State Machines for Rich Component Models. In: Nicolescu, G., Mosterman, P. (eds.) Model-Based Design for Embedded Systems, pp. 487–518. Taylor & Francis (2009), http://www.google.se/books?id=8Cjg2mM-m1MC
5. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A Theory of Communicating Sequential Processes. J. ACM 31(3), 560–599 (Jun 1984), http://doi.acm.org/10.1145/828.833
6. Cimatti, A., Tonetta, S.: A property-based proof system for contract-based design. In: 2012 38th Euromicro Conference on Software Engineering and Advanced Applications. pp. 21–28 (Sept 2012)
7. Dill, D.L.: Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits. In: Proceedings of the fifth MIT conference on Advanced research in VLSI. pp. 51–65. MIT Press, Cambridge, MA, USA (1988), http://dl.acm.org/citation.cfm?id=88056.88061
8. Hoare, C.A.R.: An Axiomatic Basis for Computer Programming. Commun. ACM 12(10), 576–580 (Oct 1969), http://doi.acm.org/10.1145/363235.363259
9. Maier, P.: A Set-Theoretic Framework for Assume-Guarantee Reasoning. In: Orejas, F., Spirakis, P., van Leeuwen, J. (eds.) Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 2076, pp. 821–834. Springer Berlin Heidelberg (2001), http://dx.doi.org/10.1007/3-540-48224-5_67
10. Meyer, B.: Applying "Design by Contract". IEEE Computer 25, 40–51 (1992)
11. Negulescu, R.: Process Spaces. In: Proceedings of the 11th Int. Conf. on Concurrency Theory. pp. 199–213. CONCUR '00, Springer-Verlag, London, UK, UK (2000), http://dl.acm.org/citation.cfm?id=646735.701627
12. Quinton, S., Graf, S.: Contract-based verification of hierarchical systems of components. In: Software Engineering and Formal Methods, 2008. SEFM '08. Sixth IEEE International Conference on. pp. 377 –381 (nov 2008)
13. Westman, J., Nyberg, M.: Conditions of contracts for separating responsibilities in heterogeneous systems. Formal Methods in System Design (Sep 2017), https://doi.org/10.1007/s10703-017-0294-7
14. Wolf, E.S.: Hierarchical Models of Synchronous Circuits for Formal Verification and Substitution. Ph.D. thesis, Stanford University, Stanford, CA, USA (1996), uMI Order No. GAX96-12052

## A Propositions and Lemmas

The following two lemmas are used to prove Proposition 1 and 2, which follow after the lemmas.

**Lemma 2.** *For any two sets of variables $X$ and $Y$, it holds that*

$$\mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B}) = \mathrm{proj}_{X\cap Y} \mathsf{B} \ .$$

*Proof.* Consider an arbitrary $\omega \in \mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B})$. In accordance with Definition 3, this means that $\exists\mu\,.\,\omega[Y] = \mu[Y] \wedge \mu \in \{\omega' \in \Omega | \omega'[X] = \nu[X], \nu \in \mathsf{B}\}$. This can be rewritten as $\exists\nu\exists\mu\,.\,\omega[Y] = \mu[Y] \wedge \mu[X] = \nu[X] \wedge \nu \in \mathsf{B}$. Since $X \cap Y \subseteq X$ and $X \cap Y \subseteq Y$, and $\mu[X] = \nu[X]$ and $\omega[Y] = \mu[Y]$, it holds that $\mu[X \cap Y] = \nu[X \cap Y]$ and $\omega[X \cap Y] = \mu[X \cap Y]$, which also means that $\omega[X \cap Y] = \nu[X \cap Y]$. Thus, it holds $\exists\nu\,.\,\omega[X \cap Y] = \nu[X \cap Y] \wedge \nu \in \mathsf{B}$. In accordance with Definition 3, this means that $\omega \in \mathrm{proj}_{X\cap Y} \mathsf{B}$. Since $\omega$ represents an arbitrary run in $\mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B})$, it must hold that each run in $\mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B})$ is also in $\mathrm{proj}_{X\cap Y} \mathsf{B}$, which means that $\mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B}) \subseteq \mathrm{proj}_{X\cap Y} \mathsf{B}$.

Next consider an arbitrary $\omega \in \mathrm{proj}_{X\cap Y} \mathsf{B}$. In accordance with Definition 3, this means that $\exists\nu\,.\,\omega[X \cap Y] = \nu[X \cap Y] \wedge \nu \in \mathsf{B}$. Create a run $\mu$ such that $\mu[X] = \nu[X]$ and $\mu[X^{\mathsf{C}}] = \omega[X^{\mathsf{C}}]$. Since $\mu[X^{\mathsf{C}}] = \omega[X^{\mathsf{C}}]$, $\omega[X \cap Y] = \nu[X \cap Y]$, and $Y \subseteq X^{\mathsf{C}} \cup (X \cap Y)$, it follows that $\omega[Y] = \mu[Y]$. Thus, we have shown that $\exists\nu\exists\mu\,.\,\omega[Y] = \mu[Y] \wedge \mu[X] = \nu[X] \wedge \nu \in \mathsf{B}$, which can, as previously mentioned, be rewritten as $\exists\mu\,.\,\omega[Y] = \mu[Y] \wedge \mu \in \{\omega' \in \Omega | \omega'[X] = \nu[X], \nu \in \mathsf{B}\}$. In accordance with Definition 3, this means that $\omega \in \mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B})$. Since $\omega$ represents an arbitrary run in $\mathrm{proj}_{X\cap Y} \mathsf{B}$, it must hold that each run in $\mathrm{proj}_{X\cap Y} \mathsf{B}$ is also in $\mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B})$, which means that $\mathrm{proj}_{X\cap Y} \mathsf{B} \subseteq \mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B})$. Since it was previously also shown that $\mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B}) \subseteq \mathrm{proj}_{X\cap Y} \mathsf{B}$, it follows that $\mathrm{proj}_Y(\mathrm{proj}_X \mathsf{B}) = \mathrm{proj}_{X\cap Y} \mathsf{B}$, which ends the proof. □

**Lemma 3.** *It holds that* $\mathrm{proj}_{X_{\mathsf{B}}} \mathsf{B} = \mathsf{B}$.

*Proof.* In accordance with Definition 4, it holds that $\mathrm{proj}_{\{x\}^{\mathsf{C}}} \mathsf{B} = \mathsf{B}$ for each variable in $X_{\mathsf{B}}^{\mathsf{C}} = \{x_1, \ldots, x_N\}$. It follows that

$$\mathrm{proj}_{\{x_1\}^{\mathsf{C}}}(\mathrm{proj}_{\{x_2\}^{\mathsf{C}}}(\ldots \mathrm{proj}_{\{x_N\}^{\mathsf{C}}}(\mathsf{B}) \ldots)) = \mathsf{B} \ .$$

It follows in accordance with Lemma 2 that $\mathrm{proj}_{\{x_1\}^{\mathsf{C}} \cap \ldots \cap \{x_N\}^{\mathsf{C}}} \mathsf{B} = \mathsf{B}$ and since $X_{\mathsf{B}} = \{x_1\}^{\mathsf{C}} \cap \ldots \cap \{x_N\}^{\mathsf{C}}$, it also holds that $\mathrm{proj}_{X_{\mathsf{B}}} \mathsf{B} = \mathsf{B}$, which ends the proof. □

**Proposition 1.** *Let $\mathsf{B}_1$ and $\mathsf{B}_2$ be two behaviors such that $X_{\mathsf{B}_1} \cap X_{\mathsf{B}_2} = \emptyset$. Then, for an arbitrary set of variables $Y$, it holds*

$$\mathrm{proj}_Y(\mathsf{B}_1 \cap \mathsf{B}_2) = \mathrm{proj}_Y \mathsf{B}_1 \cap \mathrm{proj}_Y \mathsf{B}_2 \ .$$

*Proof.* To prove $\mathrm{proj}_Y(\mathsf{B}_1 \cap \mathsf{B}_2) \subseteq \mathrm{proj}_Y \mathsf{B}_1 \cap \mathrm{proj}_Y \mathsf{B}_2$, assume $\omega \in \mathrm{proj}_Y(\mathsf{B}_1 \cap \mathsf{B}_2)$. According to Definition 3, this means that there exist a $\nu \in (\mathsf{B}_1 \cap \mathsf{B}_2)$ such that $\nu[Y] = \omega[Y]$. Thus for $i = 1, 2$, it holds that $\nu \in \mathsf{B}_i$ and $\nu[Y] = \omega[Y]$, and therefore that $\omega \in \mathrm{proj}_Y \mathsf{B}_i$, which ends the proof.

To prove also that $\mathrm{proj}_Y \mathsf{B}_1 \cap \mathrm{proj}_Y \mathsf{B}_2 \subseteq \mathrm{proj}_Y(\mathsf{B}_1 \cap \mathsf{B}_2)$, consider an arbitrary run $\omega \in \mathrm{proj}_Y \mathsf{B}_1 \cap \mathrm{proj}_Y \mathsf{B}_2$. According to Definition 3 this means that:

$$\exists \nu_1 \in \mathsf{B}_1 . \nu_1[Y] = \omega[Y]$$
$$\exists \nu_2 \in \mathsf{B}_2 . \nu_2[Y] = \omega[Y]$$

Now consider the vector $\nu = (\nu_1[X_{\mathsf{B}_1}], \nu_2[X_{\mathsf{B}_1}^C])$. Clearly it holds that $\omega[Y] = \nu[Y]$.

Since $\nu_1 \in \mathsf{B}_1$, and $\nu[X_{\mathsf{B}_1}] = \nu_1[X_{\mathsf{B}_1}]$, it holds according to Definition 3 that $\nu \in \mathrm{proj}_{X_{\mathsf{B}_1}} \mathsf{B}_1$. Then, according to Lemma 3, it also holds that $\nu_1 \in \mathsf{B}_1$. Similarly, since $\nu_2 \in \mathsf{B}_2$, and $\nu[X_{\mathsf{B}_2}] = \nu_2[X_{\mathsf{B}_2}]$, it must hold that $\nu \in \mathsf{B}_2$.

Thus, we have shown that $\nu \in \mathsf{B}_1 \cap \mathsf{B}_2$ and that $\omega[Y] = \nu[Y]$. Therefore, according to Definition 3, it holds that $\omega \in \mathrm{proj}_Y(\mathsf{B}_1 \cap \mathsf{B}_2)$, which ends the proof. □

**Proposition 2.** *It holds that* $\mathrm{proj}_Y \mathsf{B} = \Omega$ *if* $Y \cap X_\mathsf{B} \neq \emptyset$.

*Proof.* In accordance with Lemma 3, it holds that $\mathrm{proj}_Y \mathsf{B} = \mathrm{proj}_Y(\mathrm{proj}_{X_\mathsf{B}} \mathsf{B})$, and further that $\mathrm{proj}_Y \mathsf{B} = \mathrm{proj}_{Y \cap X_\mathsf{B}} \mathsf{B}$ in accordance with Lemma 2. If $Y \cap X_\mathsf{B} \neq \emptyset$, it follows that $\mathrm{proj}_Y \mathsf{B} = \mathrm{proj}_\emptyset \mathsf{B} = \Omega$. □

The following two lemmas are used to prove Theorem 2.

**Lemma 4.** *If a behavior set $\mathcal{Q}$ is agnostic to a variable set $X$, then it is also agnostic to any variable set $Y \subseteq X$.*

*Proof.* Consider a $\mathsf{B} \in \mathcal{Q}$ and a $\mathsf{B}'$ such that $\mathrm{proj}_{Y^C} \mathsf{B}' = \mathrm{proj}_{Y^C} \mathsf{B}$. Since $X^C \subseteq Y^C$, it holds according to Lemma 2, that

$$\mathrm{proj}_{X^C} \mathsf{B}' = \mathrm{proj}_{X^C} \mathrm{proj}_{Y^C} \mathsf{B}' = \mathrm{proj}_{X^C} \mathrm{proj}_{Y^C} \mathsf{B} = \mathrm{proj}_{X^C} \mathsf{B}$$

From this equality and since $\mathcal{Q}$ is agnostic to $X$, it follows that $\mathsf{B}' \in \mathcal{Q}$. Thus, we have shown that $\mathcal{Q}$ is agnostic to also $Y$. □

**Lemma 5.** *Given two behaviors* $\mathsf{B}$ *and* $\mathsf{B}'$ *where* $\mathsf{B} \cap \mathsf{B}' \neq \emptyset$, *it holds that* $X_{\mathsf{B}\cap\mathsf{B}'} \subseteq X_\mathsf{B} \cup X_{\mathsf{B}'}$.

*Proof.* Assume $X_{\mathsf{B}\cap\mathsf{B}'} \nsubseteq X_\mathsf{B} \cup X_{\mathsf{B}'}$, which will be shown to lead to a contradiction. This means that there exists an $x$ such that $x \notin X_\mathsf{B}$, $x \notin X_{\mathsf{B}'}$, and $x \in X_{\mathsf{B}\cap\mathsf{B}'}$. In accordance with Definition 4, this respectively means that

$$\mathsf{B} = \mathrm{proj}_{\{x\}^C} \mathsf{B} \ , \tag{12}$$
$$\mathsf{B}' = \mathrm{proj}_{\{x\}^C} \mathsf{B}' \ , \text{ and} \tag{13}$$
$$\mathsf{B} \cap \mathsf{B}' \neq \mathrm{proj}_{\{x\}^C}(\mathsf{B} \cap \mathsf{B}') \ . \tag{14}$$

Proposition 1 and (14) imply $\mathsf{B} \cap \mathsf{B}' \neq \mathrm{proj}_{\{x\}^C} \mathsf{B} \cap \mathrm{proj}_{\{x\}^C} \mathsf{B}'$, which in combination with (12) and (13) imply that $\mathsf{B} \cap \mathsf{B}' \neq \mathsf{B} \cap \mathsf{B}'$. Since this is a contradiction, assumption $X_{\mathsf{B}\cap\mathsf{B}'} \nsubseteq X_\mathsf{B} \cup X_{\mathsf{B}'}$ must be false, and it must rather hold that $X_{\mathsf{B}\cap\mathsf{B}'} \subseteq X_\mathsf{B} \cup X_{\mathsf{B}'}$, which completes the proof. □