# Quotient for Assume-Guarantee Contracts

Íñigo Íncer Romeo[1], Alberto Sangiovanni-Vincentelli[1], Chung-Wei Lin[2], and Eunsuk Kang[3]
[1] *University of California, Berkeley* [2] *National Taiwan University* [3] *Carnegie Mellon University*
*{inigo, alberto}@eecs.berkeley.edu* *cwlin@csie.ntu.edu.tw* *eskang@cmu.edu*

***Abstract*—We introduce a novel notion of quotient set for a pair of contracts and the operation of quotient for assume-guarantee contracts. The quotient set and its related operation can be used in any compositional methodology where design requirements are mapped into a set of components in a library. In particular, they can be used for the so called *missing component* problem, where the given components are not capable of discharging the obligations of the requirements. In this case, the quotient operation identifies the contract for a component that, if added to the original set, makes the resulting system fulfill the requirements.**

## 1. Introduction

Contracts have been proposed as a formal mechanism to support compositional design first for complex software (e.g., see [1] and references therein) and later for system design (e.g., see [2] and references therein). We embed our contribution on contracts in reuse-based, meet-in-the-middle design methodologies such as Platform-Based Design (e.g., see [3]). In these methodologies that are fairly common in industry, during the top-down phase, the specification for a system is decomposed into a set of refined specifications of sub-components, i.e., the high-level architecture of the design is determined. This step fits in a refinement driven process where higher level specifications are mapped into lower level implementations. This decomposition is "guided" by the existence of a set of predefined components in a library, the bottom-up part of the methodology.

More formally, suppose a designer wishes to implement a system that satisfies a top-level specification $\mathcal{T}$, and will use in this design a set of $n$ components from a library with specifications $\mathcal{F} = (\mathcal{T}_i)_{i=1}^n$. If the composition of these $n$ design elements refines the top level specification $\mathcal{T}$, the design assembled from the components satisfies the specification. However, if this is not the case, the designer must add at least one element to the library. In other words, the designer must identify a specification $\mathcal{T}_M$ such that its composition with the composition of $(\mathcal{T}_i)_{i=1}^n$ refines $\mathcal{T}$.

This problem corresponds to identifying the *missing (unknown) component* in a library (e.g., see [4] and references therein). In the language of contract-based design, we need to compute the *contract quotient* that guarantees that $\mathcal{T}_M$ is as "large" a specification as possible in the refinement order so that whoever is in charge of adding the element to the library has the maximum degree of freedom in its implementation. In industry, this problem is in general tackled heuristically. Our proposed notion of quotient for assume-guarantee contracts aims at finding a rigorous procedure for the determination of the largest specification of the missing component.

*Lin and Kang were at Toyota ITC U.S.A. Inc., when this work was done.*

**Related work.** In [5], Le et al. address the problem of fixing a decomposition so that it refines a specification: given a top-level contract $\mathcal{C}$ and family of contracts $(\mathcal{C}_i)_{i=1}^n$ whose composition may not refine $\mathcal{C}$, find a family $(\mathcal{C}_i')_{i=1}^n$ such that $\mathcal{C}_i' \leq \mathcal{C}_i$ for all $i$ and $\bigotimes_{i=1}^n \mathcal{C}_i' \leq \mathcal{C}$. In this setting, the designer begins with a high-level specification $\mathcal{C}$ and an initial decomposition of the specification into various components, but the specification, $\mathcal{C}_i$, of any component may need to be corrected to $\mathcal{C}_i'$.

Notions similar to quotients have been previously investigated in the context of various behavioral formalisms. Chilton et al. formulate in [6] an assume-guarantee framework for reasoning about components modeled as a variant of interface automata introduced by Chen et al. [7], including quotient operations. Similarly, Bhaduri and Ramesh [8] investigate the problem of synthesizing, given $P$ and $Q$ as interface automata, $R$ such that the composition of $R$ and $P$ refines $Q$; they provide a game-theoretic formulation of the problem as computing winning strategies over a game between $P$ and $Q$. In addition, Raclet [9] introduces the concept of a *residual specification* in the context of modal automata.

Another related line of research is on assumption generation in the context of compositional verification [10]–[12]: Given a component $M$ and a desired property $P$, what is the weakest assumption $A$ that $M$ can make about its environment such that $M \oplus A \models P$? These works typically assume $M$ and $A$ to be labelled transition systems, and exploit their structures as part of a learning algorithm (e.g., L* [13]).

Compared to the contributions mentioned above, our approach differs in that we provide a general form for the contract quotient for assume-guarantee contracts, i.e., a formulation that holds for all variants of assume-guarantee contracts. Furthermore, as far as we know, our approach is the first to introduce the notion of a *quotient set*, which defines a range of contracts that can be composed with $C_1$ to yield the largest contract that refines $C$.

**Specific Contributions.** Benveniste et al. in their comprehensive review of contract based design [2], on page 188, state that "no Least Upper Bound and no quotient are known for A/G contracts." One of our contributions is an explicit form for the quotient operation of A/G contracts (Theorem 3.5 in Section 3). We point out that notions similar to quotients have been proposed for A/G contracts before, but these operations are defined when contracts are expressed in specific formalisms [6], [10], [14]. To the best of our knowledge, the quotient operation for general A/G contracts has not been addressed.

We also introduce the notion of the quotient set (Definition 3.6), a set that contains all contracts $\mathcal{C}'$ such that $\mathcal{C}' \otimes \mathcal{C}_1 = (\mathcal{C}/\mathcal{C}_1) \otimes \mathcal{C}_1$ for given contracts $\mathcal{C}$ and $\mathcal{C}_1$. The quotient set tells us which contracts *extend* $\mathcal{C}_1$ into $\mathcal{C}$ in the largest way possible. We fully characterize the quotient set in Theorem 3.7 (Section 3).

A further contribution of our paper is an alternate definition of the quotient operation for the meta-theory of contracts (Section 4). The new definition makes the quotient operation an obvious dual of the composition operation and makes the derivation of the quotient for A/G contracts almost immediate.

Finally, we show a methodology and examples of the use of the quotient operation in the design of an ALU and of an automotive system (Section 5). We believe the theory in this paper can significantly improve the integration process in practical system design. For example, it can assist automotive Original Equipment Manufacturers (OEMs) in defining the specification of a component to be implemented[1]. We show that with appropriate automation tools, which are left as future work, the process can be formal but still efficient.

## 2. Contract-Based Design

Contract-based design (CBD) [15] has emerged as a formal methodology for system-level design (see [2] for a comprehensive analysis of the state-of-the-art). This methodology supports open systems and the ability to fuse multiple viewpoints in a single component. The former is needed to support independent design, and the later to enable the independent characterization of complementary aspects of the same component (e.g., functionality and timing). At the core of the contract algebra lies the notion of a component. Components are described by their IO connectivity and by their dynamics. A specification, or contract, has semantics given by a pair $(\mathcal{E}, \mathcal{M})$ of sets of components which are, respectively, *environments* for the contract and of components which are *implementations* for the contract. Contracts provide a designer with (i) formal notions of refinement (to enable substitution of components) and parallel composition (to enable merging multiple components into one); (ii) the ability to check compatibility between the objects to be composed; and (iii) a formal means of communication of requirements to suppliers. With a contract, a supplier has all the required information to implement a subsystem in a way which guarantees system integration.

1. In industrial designs, OEMs generally keep most of the components and only change as few components as possible when a new design is developed. The task here is to perform the minimum amount of work needed to identify and implement the new components

To be able to manipulate contracts, we need to assume more structure in them. Thus, there exist two specialized contract theories: interface theories and the assume-guarantee framework, our focus in this paper. In the assume-guarantee theory, components are modeled as *sets of behaviors*, where behaviors are successive assignments of values to the various variables in the modeling framework (which also determines the granularity of this succession: discrete, continuous, asynchronous, etc.). For instance, if a designer is implementing synchronous digital logic, his modeling framework uses a discrete-event model of computation, where behaviors are infinite, discrete sequences of values for all variables under consideration. More concretely, suppose we are dealing with only two boolean variables $a$ and $b$; one possible behavior is $a\,b, a\,\neg b, \neg a\,\neg b, a\,b \ldots$ If the designer is implementing a cyber-physical system, his modeling framework may need to enable analysis of continuous dynamics.

An assume-guarantee contract $\mathcal{C}$ is given by a pair $(A, G)$, where $A$ and $G$ are, respectively, *sets of behaviors for assumptions and for guarantees*. In other words, assumptions and guarantees are properties, sets of traces, and thus can be expressed in any formal language, such as LTL, MTL, and STL (e.g., [16]–[18]). This means that A/G contracts are very flexible. We say a component $M$ satisfies a contract if it provides the contract guarantees subject to the contract assumptions, i.e., if $M \subseteq G \cup \neg A$. With this notion of satisfaction of an A/G contract, many A/G contract pairs $(A, G)$ can have the same semantics. In order to deal with unique objects, we use contracts in *saturated* form. We say contract $\mathcal{C}$ is saturated if $G = G \cup \neg A$. Saturated contracts have a maximal set of guarantees.

The algebra of contracts defines an order relation called refinement and a binary operation called composition. Refinement allows designers to substitute a component with another that has at least the same capabilities. This order is defined as follows: let $\mathcal{C}' = (A', G')$ and $\mathcal{C} = (A, G)$ be saturated contracts. Contract $\mathcal{C}'$ refines $\mathcal{C}$, written $\mathcal{C}' \leq \mathcal{C}$, if $G' \subseteq G$ and $A \subseteq A'$. Refinement amounts to weakening assumptions and strengthening guarantees. The binary operation of composition produces a contract which represents the simultaneous operation of the two given contracts: let $\mathcal{C}'' = (A'', G'')$ be a saturated contract. Given $\mathcal{C}'$ and $\mathcal{C}''$, the binary operation $\otimes$ generates a new contract $\mathcal{C}' \otimes \mathcal{C}'' = (A' \cap A'' \cup \neg G' \cup \neg G'', G' \cap G'')$. This is usually called the *horizontal composition* of the contracts $\mathcal{C}'$ and $\mathcal{C}''$. The composite of two contracts is required to provide the guarantees of both; the composite assumes simultaneously the assumptions of $\mathcal{C}'$ and of $\mathcal{C}''$ but relaxes these assumptions by the guarantees of both $\mathcal{C}'$ and $\mathcal{C}''$. This relaxation occurs because the guarantees of each contract may help the to meet the other's assumptions.

Refinement and composition allows us to discuss the binary operation of *quotient*, a central topic of this paper. The quotient between contracts $\mathcal{C}$ and $\mathcal{C}_1$, expressed $\mathcal{C}/\mathcal{C}_1$, is defined as

$$\mathcal{C}' \leq \mathcal{C}/\mathcal{C}_1 \iff \mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}. \qquad (1)$$

The quotient provides the largest contract in the refinement order that, when composed with $\mathcal{C}_1$, refines $\mathcal{C}$. The explicit formulation of the quotient is not known for A/G contracts [2]. One of our key contributions is the explicit expression for this quotient.

## 3. Quotient of Assume-Guarantee Contracts

Assume that $\mathcal{C} = (A, G)$ is the top-level specification we wish to implement and that $\mathcal{C}_1 = (A_1, G_1)$ is the specification of a component that will be used in the design. We are interested in the part of the specification that is not discharged by $\mathcal{C}_1$, and in the maximal specification we can form by composing $\mathcal{C}_1$ with a missing specification with the result refining $\mathcal{C}$. We introduce the quotient operation of assume-guarantee contracts in Section 3.1. We introduce in section 3.2 the *quotient set* (Definition 3.6), a notion that expresses what we need to do to complement $\mathcal{C}_1$ to satisfy $\mathcal{C}$ in a maximal way in the refinement order. Theorems 3.5 and 3.7 are our main results in this section. Theorem 3.5 provides an explicit formula for the quotient operation for assume-guarantee contracts, and Theorem 3.7 provides a complete characterization of the quotient set as an interval of contracts. Throughout this section, we assume all A/G contracts are given in saturated form. We observe that this is not restrictive since a contract can always be saturated through the mapping $(A, G) \longmapsto (A, G \cup \neg A)$. A contract is semantically equivalent to its saturated form. In our manipulations, we evaluate set-theoretic operators in the order $\neg$, $\cap$, $\cup$. Missing proofs are given in the appendix.

### 3.1. Quotient Operation of A/G Contracts

In this section we introduce the quotient operation of A/G contracts. Given contracts $\mathcal{C}$ and $\mathcal{C}_1$, we recall from (1) that the quotient operation $\mathcal{C}/\mathcal{C}_1$ results in a contract with the following defining property: for any saturated contract $\mathcal{C}'$, we have $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$ if and only if $\mathcal{C}' \leq \mathcal{C}/\mathcal{C}_1$. To derive the quotient, we first define contract $R$. Lemma 3.4 shows that $R$ has properties akin to those of the quotient operation, and Theorem 3.5 uses this result to provide the quotient operation.

***Definition 3.1.*** Let $\mathcal{C} = (A, G)$ be an A/G contract. We use the notation $a(\mathcal{C})$ and $g(\mathcal{C})$ to refer to the sets of assumptions and guarantees of $\mathcal{C}$, respectively, i.e., $(A, G) \xmapsto{a} A$ and $(A, G) \xmapsto{g} G$ .

We provide various bounds on a saturated contract $\mathcal{C}'$ that satisfies $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$.

***Lemma 3.2.*** Let $\mathcal{C} = (A, G)$, $\mathcal{C}_1 = (A_1, G_1)$, and $\mathcal{C}' = (A', G')$ be saturated A/G contracts such that $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$. Then

$$g(\mathcal{C}') \subseteq \neg G_1 \cup G, \tag{2}$$
$$g(\mathcal{C}' \otimes \mathcal{C}_1) \subseteq G_1 \cap G, \tag{3}$$
$$A \cup \neg G_1 \subseteq a(\mathcal{C}' \otimes \mathcal{C}_1), \quad \text{and} \tag{4}$$
$$A \cap G_1 \subseteq a(\mathcal{C}'). \tag{5}$$

We proceed to define contract $R$, which is computed from $\mathcal{C}$ and $\mathcal{C}'$. We show in Lemma 3.4 that contract $R$ has properties that resemble those of the quotient operation, and in Theorem 3.5 we use this result to provide the quotient operation.

***Definition 3.3.*** Let $\mathcal{C} = (A, G)$ and $\mathcal{C}_1 = (A_1, G_1)$ be saturated contracts. We define the contract

$$R(\mathcal{C}, \mathcal{C}_1) := (A \cap G_1, G \cup \neg G_1) .$$

When the context of $\mathcal{C}$ and $\mathcal{C}_1$ is clear, we may use the notation $R = R(\mathcal{C}, \mathcal{C}_1)$.

***Lemma 3.4.*** Let $\mathcal{C}$, $\mathcal{C}_1$, and $\mathcal{C}'$ be saturated contracts. The following statements are equivalent:

i. $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$ and
ii. $\mathcal{C}' \leq R$ and $A \cap G' \subseteq A_1$.

*Proof:* (i. $\Rightarrow$ ii.). Let $\mathcal{C}'$ be a saturated contract such that $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$. From Lemma 3.2, $g(\mathcal{C}') \subseteq G \cup \neg G_1 = g(R)$ and $a(\mathcal{C}') \supseteq A \cap G_1 = a(R)$, i.e., $\mathcal{C}' \leq R$. Moreover, from i., $A \subseteq A_1 \cap A' \cup \neg G_1 \cup \neg G' \subseteq A_1 \cup \neg G'$. Intersecting both sides with $G'$, we obtain $A \cap G' \subseteq A_1$.

(ii. $\Rightarrow$ i.). Assume $\mathcal{C}' \leq R$ and $A \cap G' \subseteq A_1$. Then $g(\mathcal{C}' \otimes \mathcal{C}_1) = g(\mathcal{C}') \cap G_1 \subseteq g(R) \cap G_1 \subseteq g(\mathcal{C})$. From ii., we have $A \cap G' \subseteq A_1$ and $A \cap G_1 = a(R) \subseteq a(\mathcal{C}') = A'$; thus, $A_1 \cap A' \supseteq A \cap G' \cap G_1$. Therefore, we can write $A_1 \cap A' = A_1 \cap A' \cup A \cap G' \cap G_1$ (since we are just adding a subset). With this identity, we have $a(\mathcal{C}' \otimes \mathcal{C}_1) = A_1 \cap A' \cup \neg G' \cup \neg G_1 = A_1 \cap A' \cup \neg G' \cup \neg G_1 \cup A \cap G' \cap G_1 \supseteq A = a(\mathcal{C})$. We conclude that $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$. □

***Theorem 3.5 (Quotient of A/G contracts).*** Given saturated contracts $\mathcal{C}$ and $\mathcal{C}_1$, the operation defined as

$$\mathcal{C}/\mathcal{C}_1 := (A \cap G_1, A_1 \cap G \cup \neg(A \cap G_1)) \tag{6}$$

is the quotient in the formalism of assume-guarantee contracts.

*Proof:* We observe that the operation just defined produces a contract in saturated form. We wish to show that this operation satisfies (1).

($\Leftarrow$) in (1). Suppose $\mathcal{C}'$ is a saturated contract such that $\mathcal{C}_1 \otimes \mathcal{C}' \leq \mathcal{C}$. By Lemma 3.4, $A \cap g(\mathcal{C}') \subseteq A_1 \Rightarrow g(\mathcal{C}') \subseteq A_1 \cup \neg A$ and $\mathcal{C}' \leq R \Rightarrow g(\mathcal{C}') \subseteq G \cup \neg G_1$. Thus, $g(\mathcal{C}') \subseteq (A_1 \cup \neg A) \cap (G \cup \neg G_1) = g(\mathcal{C}/\mathcal{C}_1)$. Moreover, since $\mathcal{C}' \leq R$, it follows that $a(\mathcal{C}') \supseteq A \cap G_1 = a(\mathcal{C}/\mathcal{C}_1)$. We thus conclude that $\mathcal{C}' \leq \mathcal{C}/\mathcal{C}_1$.

($\Rightarrow$) in (1). Suppose $\mathcal{C}'$ is a saturated contract such that $\mathcal{C}' \leq \mathcal{C}/\mathcal{C}_1$. We have $g\left((\mathcal{C}/\mathcal{C}_1) \otimes \mathcal{C}_1\right) = G_1 \cap A_1 \cap G \cup G_1 \cap \neg A \subseteq g(\mathcal{C})$. We also have $a\left((\mathcal{C}/\mathcal{C}_1) \otimes \mathcal{C}_1\right) = A_1 \cap A \cap G_1 \cup \neg G_1 \cup A \cap G_1 \cap (\neg A_1 \cup \neg G) \supseteq A = a(\mathcal{C})$. We conclude that $\mathcal{C} \geq (\mathcal{C}/\mathcal{C}_1) \otimes \mathcal{C}_1 \geq \mathcal{C}' \otimes \mathcal{C}_1$. □

We can develop intuition about the expression we derived. A component which is designed to the specifications of the quotient can use the assumptions of $\mathcal{C}$ and can assume that $\mathcal{C}_1$ will meet its guarantees; thus the assumptions of the quotient are $A \cap G_1$. On the side of guarantees, the component built to the quotient specifications must provide the guarantees of $\mathcal{C}$ and the assumptions of $\mathcal{C}_1$ (to make sure $\mathcal{C}_1$ can meet its guarantees); however, the component can relax its guarantees by the assumptions of $\mathcal{C}$ and by the guarantees of $\mathcal{C}_1$. This results in the guarantees we derived for the quotient.

## 3.2. Quotient Set of A/G Contracts

We introduce the concept of quotient set to answer two questions:

- When designing a system to meet a specification $\mathcal{C}$, what is the *biggest* specification achievable if a component with specification $\mathcal{C}_1$ must be used in the design? That is, we seek to characterize the contracts whose compositions with $\mathcal{C}_1$ result in the biggest possible contract that refines $\mathcal{C}$.
- Given a contract $\mathcal{C}_1$ and a contract $\mathcal{C}$ that is the result of the composition of $\mathcal{C}_1$ with another contract, what are all contracts $\mathcal{C}'$ that satisfy $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}'$? In this case, we solve the inverse problem to composition.

We call the set of such contracts the *quotient set*. We begin our quest for the quotient set with an observation: suppose that $\mathcal{C}'$ is a saturated contract that satisfies $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$. Then $\mathcal{C}' \leq \mathcal{C}/\mathcal{C}_1$. This last statement implies that $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}/\mathcal{C}_1 \otimes \mathcal{C}_1$. Thus, the greatest resulting contract achievable by composing $\mathcal{C}_1$ with a contract with the result refining $\mathcal{C}$ is $\mathcal{C}_1 \otimes \mathcal{C}/\mathcal{C}_1$. A quick computation reveals that

$$\mathcal{C}_1 \otimes \mathcal{C}/\mathcal{C}_1 = (A \cup \neg G_1, G_1 \cap (A_1 \cap G \cup \neg A)).$$

Thus, we define a set $Q$ whose elements are saturated contracts whose composition with $\mathcal{C}_1$ is equal to the composition we just derived (Definition 3.6). We then show in Theorem 3.7 that the quotient set is completely characterized as an interval of contracts. After providing this result, we discuss a special and important case of contract decomposition which allows us to simplify many expressions.

***Definition 3.6.*** Let $\mathcal{C} = (A, G)$ and $\mathcal{C}_1 = (A_1, G_1)$ be saturated contracts. We define the quotient set $Q(\mathcal{C}, \mathcal{C}_1)$ as

$$\begin{aligned} Q(\mathcal{C}, \mathcal{C}_1) := \{\mathcal{C}' \,|\, & \mathcal{C}' \text{ is saturated}, \\ & g(\mathcal{C}' \otimes \mathcal{C}_1) = G_1 \cap (A_1 \cap G \cup \neg A), \\ & a(\mathcal{C}' \otimes \mathcal{C}_1) = A \cup \neg G_1\}. \end{aligned}$$

We also define the lower quotient operation $(\mathcal{C}/\mathcal{C}_1)_-$ as

$$(\mathcal{C}/\mathcal{C}_1)_- := (A \cup \neg G_1 \cup \neg A_1, G_1 \cap (A_1 \cap G \cup \neg A)).$$

***Theorem 3.7.*** Let $\mathcal{C} = (A, G)$ and $\mathcal{C}_1 = (A_1, G_1)$ be saturated contracts. Then $Q(\mathcal{C}, \mathcal{C}_1) = [(\mathcal{C}/\mathcal{C}_1)_-, \mathcal{C}/\mathcal{C}_1]$.

*Proof:* ($\Leftarrow$) Let $\mathcal{C}'$ be a saturated contract with $(\mathcal{C}/\mathcal{C}_1)_- \leq \mathcal{C}' \leq \mathcal{C}/\mathcal{C}_1$. We wish to show that $\mathcal{C}' \in Q$. A quick calculation shows that $\mathcal{C}/\mathcal{C}_1, (\mathcal{C}/\mathcal{C}_1)_- \in Q$, which means that $\mathcal{C}_1 \otimes (\mathcal{C}/\mathcal{C}_1)_- = \mathcal{C}_1 \otimes \mathcal{C}/\mathcal{C}_1 = \mathcal{C}_1 \otimes \mathcal{C}''$ for any $\mathcal{C}'' \in Q$. Composing our assumption with $\mathcal{C}_1$ results in $\mathcal{C}_1 \otimes (\mathcal{C}/\mathcal{C}_1)_- \leq \mathcal{C}_1 \otimes \mathcal{C}' \leq \mathcal{C}_1 \otimes \mathcal{C}/\mathcal{C}_1$; therefore, $\mathcal{C}_1 \otimes \mathcal{C}' = \mathcal{C}_1 \otimes \mathcal{C}''$, which implies that $\mathcal{C}' \in Q$.

($\Rightarrow$) Let $\mathcal{C}' = (A', G') \in Q$. From the definition of $Q$, it follows immediately that $\mathcal{C}_1 \otimes \mathcal{C}' \leq \mathcal{C}$. Thus, $\mathcal{C}' \leq \mathcal{C}/\mathcal{C}_1$. We need to show that $(\mathcal{C}/\mathcal{C}_1)_- \leq \mathcal{C}'$. Expanding $\mathcal{C}_1 \otimes \mathcal{C}'$ and using the fact that $\mathcal{C}' \in Q$ gives

$$A \cup \neg G_1 = a(\mathcal{C}_1 \otimes \mathcal{C}') = A_1 \cap A' \cup \neg G_1 \cup \neg G' \text{ and} \quad (7)$$
$$G_1 \cap (A_1 \cap G \cup \neg A) = g(\mathcal{C}_1 \otimes \mathcal{C}') = G_1 \cap G'. \quad (8)$$

Equation (8) gives the following bounds:

$$G_1 \cap (A_1 \cap G \cup \neg A) \subseteq G' \subseteq A_1 \cap G \cup \neg A \cup \neg G_1.$$

Plugging *any* of these bounds in (7) results in $A \cup \neg G_1 = A_1 \cap A' \cup \neg G_1 \cup \neg G \cup A \cap \neg A_1$. From this expression, we get the bound $A' \subseteq A \cup \neg G_1 \cup \neg A_1 = a((\mathcal{C}/\mathcal{C}_1)_-)$. Moreover, we note that the leftmost expression of (8) is $g((\mathcal{C}/\mathcal{C}_1)_-)$; thus, (8) gives us $g((\mathcal{C}/\mathcal{C}_1)_-) \subseteq G'$. We conclude that $(\mathcal{C}/\mathcal{C}_1)_- \leq \mathcal{C}'$. □

It should be emphasized that Theorem 3.7 gives a full characterization of the quotient set, which tells the contracts extending $\mathcal{C}_1$ into $\mathcal{C}$ in the largest way possible. The bounds we obtained are the quotient operation and the lower quotient. We understand what the quotient operation gives us (the part of the top-level spec $\mathcal{C}$ that $\mathcal{C}_1$ is missing). But what is the lower quotient? We start by recalling that $Q$ contains the contracts whose composition with $\mathcal{C}_1$ gives the biggest possible contract that refines $\mathcal{C}$; the result of this composition is $\mathcal{C}_1 \otimes \mathcal{C}/\mathcal{C}_1$ since the quotient gives the largest extension of $\mathcal{C}_1$ into $\mathcal{C}$. We showed that $(\mathcal{C}/\mathcal{C}_1)_- \in Q$, so $\mathcal{C}_1 \otimes (\mathcal{C}/\mathcal{C}_1)_-$ is the biggest contract achievable by using $\mathcal{C}_1$ while refining $\mathcal{C}$. We observe that the assumptions of the lower quotient are $A \cup \neg G_1 \cup \neg A_1$, i.e., the lower quotient must fulfill its guarantees when the assumptions of $\mathcal{C}$ are met, when the assumptions of $\mathcal{C}_1$ are not met, or when the guarantees of $\mathcal{C}_1$ are not met; this means that if $\mathcal{C}_1$ fails to behave as it promised, $(\mathcal{C}/\mathcal{C}_1)_-$'s guarantees are in force. And what are these guarantees? If $A$ holds, these guarantees are $G_1 \cap A_1 \cap G$, i.e., the contract meets the guarantees of $\mathcal{C}$ and meets the assumptions and guarantees of $\mathcal{C}_1$; if $A$ does not hold, these guarantees are $G_1$. Thus, we interpret $(\mathcal{C}/\mathcal{C}_1)_-$ as the contract whose implementations are maximally redundant with respect to contract $\mathcal{C}_1$ while completing $\mathcal{C}_1$ in the largest possible way. In contrast, $\mathcal{C}/\mathcal{C}_1$ relies completely on the fact that $\mathcal{C}_1$ will behave as it promises.

### 3.2.1. A simplification of the quotient.

Now we consider a simplification that occurs to the quotient operation when $A \cap G \subseteq A_1$. This condition holds when, for instance, all inputs of $\mathcal{C}_1$ can be mapped directly to either compatible inputs or compatible outputs of $\mathcal{C}$; note that in this case we are interpreting the assumptions and guarantees of $\mathcal{C}_1$ in terms of IO behavior. The following corollary shows how the quotient set and its bound simplify when $A \cap G \subseteq G_1$:

***Corollary 3.8.*** Let $\mathcal{C} = (A, G)$ and $\mathcal{C}_1 = (A_1, G_1)$ be saturated contracts. If $A \cap G \subseteq A_1$, the quotient set simplifies to

$$\begin{aligned} Q(\mathcal{C}, \mathcal{C}_1) := \{\mathcal{C}' \,|\, & \mathcal{C}' \text{ is saturated}, \\ & g(\mathcal{C}' \otimes \mathcal{C}_1) = G_1 \cap G, \\ & a(\mathcal{C}' \otimes \mathcal{C}_1) = A \cup \neg G_1\}, \end{aligned}$$

the quotient operation becomes $\mathcal{C}/\mathcal{C}_1 = R(\mathcal{C},\mathcal{C}_1)$, and the lower quotient becomes $(\mathcal{C}/\mathcal{C}_1)_- = L(\mathcal{C},\mathcal{C}_1)$, where

$$L(\mathcal{C},\mathcal{C}_1) := (A \cup \neg(G_1 \cap A_1), G \cap G_1)\,.$$

Corollary 3.8 tells us that $R$ is equal to the quotient operation and $L$ is equal to the lower quotient when the condition $A \cap G \subseteq A_1$ holds. Observe the form of $R$: $R = (A \cap G_1, G \cup \neg G_1)$. It assumes the assumptions of $\mathcal{C}$ and the guarantees of $\mathcal{C}_1$, and it guarantees $g(\mathcal{C})$ relaxed by whatever $\mathcal{C}_1$ guarantees. This is a very intuitive notion of the quotient. And what is $L$? Note that $L = (A \cup \neg(G_1 \cap A_1), G \cap G_1)$. This contract is responsible for its guarantees when either the assumptions of $\mathcal{C}$ are in force or when either the assumptions or guarantees of $\mathcal{C}_1$ are not met (i.e., when $\mathcal{C}_1$ fails). And the guarantees of $L$ are $G \cap G_1$, i.e., $L$ provides the guarantees of both $\mathcal{C}_1$ and $\mathcal{C}$. Thus, $L$ is correlated with a maximally redundant design.

We now consider a simple example that demonstrates the use of these concepts.

### 3.3. An Illustrative Example

Suppose we are designing a system with a Boolean input $r$ and Boolean outputs $s$ and $p$. Upon the assertion of $r$, the purpose of this system is to eventually assert $s$ and to eventually assert $p$ as long as the environment respects reasonable physical constraints. That is, the system must guarantee $\mathbf{G}(r \longrightarrow \mathbf{F}s \wedge \mathbf{F}p)$, subject to the assumptions $e \in E$ (i.e., a continuous environment variable is within some acceptable set). The top level contract is

$$\mathcal{C} = (A, G) = (e \in E, \mathbf{G}(r \longrightarrow \mathbf{F}s \wedge \mathbf{F}p) \vee e \notin E)\,.$$

Suppose a component to be used in the design guarantees the assertion of $s$ one time event after the assertion of $r$. Moreover, suppose that this component has laxer requirements on the environment than the top-level spec. It follows this component obeys the contract

$$\mathcal{C}_1 = (A_1, G_1) = (e \in E_1, \mathbf{G}(r \longrightarrow \mathbf{X}s) \vee e \notin E_1)\,,$$

where $E_1 \supseteq E$.

If a component with contract $\mathcal{C}_1$ is used to build a design that meets the spec $\mathcal{C}$, we compute the quotient to determine how much of the top-level spec $\mathcal{C}_1$ is missing. Intuitively, what do we expect the quotient to give us? We see that $\mathcal{C}_1$ can satisfy the $\mathbf{F}s$ part of $\mathcal{C}$. Thus, we expect the quotient to only have to guarantee $\mathbf{G}(r \longrightarrow \mathbf{F}p)$. We now carry out the computation.

Since $A \subseteq A_1$, it follows that $R$ is the contract quotient $\mathcal{C}/\mathcal{C}_1$ (Corollary 3.8). We compute $R = (A \cap G_1, G \cup \neg G_1)$:

$$\begin{aligned} a(R) =& e \in E \wedge \mathbf{G}(r \longrightarrow \mathbf{X}s) \vee e \in E \wedge e \notin E_1 \\ g(R) =& \mathbf{G}(r \longrightarrow \mathbf{F}s \wedge \mathbf{F}p) \\ & \quad \vee e \notin E \vee \neg \mathbf{G}(r \longrightarrow \mathbf{X}s) \wedge e \in E_1. \end{aligned}$$

Since $E \subseteq E_1$, we can simplify the assumptions to $a(R) = e \in E \wedge \mathbf{G}(r \longrightarrow \mathbf{X}s)$. The guarantees become

$$\begin{aligned} g(R) &= \mathbf{G}(r \longrightarrow \mathbf{F}s \wedge \mathbf{F}p) \vee e \notin E \vee \neg \mathbf{G}(r \longrightarrow \mathbf{X}s) \\ &= \mathbf{G}\,(r \longrightarrow \mathbf{F}s \wedge \mathbf{F}p) \wedge \mathbf{G}\,(r \longrightarrow \mathbf{X}s) \\ &\quad \vee e \notin E \vee \neg \mathbf{G}(r \longrightarrow \mathbf{X}s) \\ &= \mathbf{G}\,((r \longrightarrow \mathbf{F}s \wedge \mathbf{F}p) \wedge (r \longrightarrow \mathbf{X}s)) \\ &\quad \vee e \notin E \vee \neg \mathbf{G}(r \longrightarrow \mathbf{X}s) \\ &= \mathbf{G}(r \longrightarrow \mathbf{X}s \wedge \mathbf{F}p) \\ &\quad \vee e \notin E \vee \neg \mathbf{G}(r \longrightarrow \mathbf{X}s) \\ &= \mathbf{G}(r \longrightarrow \mathbf{F}p) \vee e \notin E \vee \neg \mathbf{G}(r \longrightarrow \mathbf{X}s). \end{aligned}$$

$R$ is (up to saturation) what we posited was missing: $\mathbf{G}(r \longrightarrow \mathbf{F}p)$.

Computing $L = (A \cup \neg(A_1 \cap G_1), G \cap G_1)$ results in $a(L) = e \in E \vee e \notin E_1 \vee \neg \mathbf{G}(r \longrightarrow \mathbf{X}s)$ and $g(L) = \mathbf{G}(r \longrightarrow \mathbf{X}s \wedge \mathbf{F}p) \vee \mathbf{G}(r \longrightarrow \mathbf{X}s) \wedge e \notin E \vee e \notin E_1$. As we discussed before, $L$ has the characteristic of providing the guarantees of $\mathcal{C}$ and of $\mathcal{C}_1$, i.e., $L$ provides a specification with redundancy.

## 4. Quotient in the Meta-Theory of Contracts

In this section, we consider contracts at their most general level, i.e., we stop considering the assume-guarantee framework and operate on our contracts as defined in the meta-theory of contracts. Our contributions are a new definition for the quotient operation and the introduction of the quotient set in the meta-theory. Our purpose in introducing a new (but equivalent) definition of the quotient operation is that the new formulation is an obvious adjoint of the composition operation, making the definition of quotient symmetrical to the definition of composition. From our definition, the derivation of the quotient operation in the assume-guarantee framework comes readily. Hence, at the end of the section we provide a second derivation of the A/G quotient.

### 4.1. What is the Meta-Theory of Contracts?

Before discussing various contract theories, Benveniste et al. [2] introduce a meta-theory of contracts. This meta-theory defines contracts on a set of primitives and allows for a birds-eye view of the subject, focusing on semantic concepts. Several key facts can be proved at this level. We can interpret other contract theories, like assume-guarantee and interface theories, as specializations of the meta-theory.

In the meta-theory, the most primitive concept is the component. Composition, $\times$, is a partial binary operation on components. We say components $M_1$ and $M_2$ are *composable* if $M_1 \times M_2$ is well-defined. We say a component $E$ is an environment for component $M$ if $M \times E$ is well-defined. A contract $\mathcal{C}$ has semantics given by $(\mathcal{E}, \mathcal{M})$, where $\mathcal{E}$ and $\mathcal{M}$ are sets of components which are valid environments and implementations, respectively, of the contract $\mathcal{C}$.

A contract $\mathcal{C}$ is called *consistent* if $\mathcal{M} \neq \emptyset$ and *compatible* if $\mathcal{E} \neq \emptyset$. We say that a component $M$ is an implementation of contract $\mathcal{C}$ ($M \models^M \mathcal{C}$) iff $M \in \mathcal{M}$; we say that a component $E$ is an environment of contract $\mathcal{C}$ ($E \models^E \mathcal{C}$) iff $E \in \mathcal{E}$.

Refinement in the meta-theory is defined as follows: we say contract $\mathcal{C}' = (\mathcal{E}', \mathcal{M}')$ is a refinement of contract $\mathcal{C} = (\mathcal{E}, \mathcal{M})$ if $\mathcal{E}' \supseteq \mathcal{E}$ and $\mathcal{M}' \subseteq \mathcal{M}$. For contracts $\mathcal{C}$ and $\mathcal{C}'$, $\mathcal{C} \wedge \mathcal{C}'$ and $\mathcal{C} \vee \mathcal{C}'$ are the Greatest Lower Bound (GLB) and Least Upper Bound (LSB), respectively, in the refinement order. In the meta-theory, we make the following assumption:

**Assumption 1.** *Both the GLB and LUB are well-defined.*

Now we get to composition. The composition of contracts $\mathcal{C}_1 = (\mathcal{E}_1, \mathcal{M}_1)$ and $\mathcal{C}' = (\mathcal{E}', \mathcal{M}')$ is defined as follows: $\mathcal{C}_1 \otimes \mathcal{C}'$ is well-defined if $M_1$ and $M'$ are composable for all $M_1 \in \mathcal{M}_1$ and $M' \in \mathcal{M}'$. If it is well-defined, this composition is $\mathcal{C}_1 \otimes \mathcal{C}' = \wedge \mathbf{C}_{\mathcal{C}_1, \mathcal{C}'}$, where

$$\mathbf{C}_{\mathcal{C}_1,\mathcal{C}'} = \left\{ \mathcal{C} \;\middle|\; \begin{array}{l} M_1 \times E \models^E \mathcal{C}', \quad M' \times E \models^E \mathcal{C}_1, \\ \text{and } M' \times M_1 \models^M \mathcal{C} \text{ for all} \\ E, M_1, M' \text{ such that } E \models^E \mathcal{C}, \\ M_1 \models^M \mathcal{C}_1, \text{ and } M' \models^M \mathcal{C}' \end{array} \right\}.$$

The quotient operation is defined as $\mathcal{C}/\mathcal{C}_1 = \vee \{\mathcal{C}' \mid \mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}\}$, i.e., the greatest $\mathcal{C}'$ such that $\mathcal{C}' \otimes \mathcal{C}_1$ refines $\mathcal{C}$. Any contract that refines the quotient also refines $\mathcal{C}$ when composed with $\mathcal{C}_1$ and vice versa, as expressed in (1).

## 4.2. The Quotients in the Meta-Theory

We define the following sets of contracts:

***Definition 4.1.*** Given contracts $\mathcal{C}$ and $\mathcal{C}_1$, let

- $S := \{\mathcal{C}' \mid \mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}\}$. Contracts whose composition with $\mathcal{C}_1$ refines $\mathcal{C}$.
- $U := \{\mathcal{C}' \otimes \mathcal{C}_1 \mid \mathcal{C}' \in S\}$. The compositions of $\mathcal{C}_1$ that refine $\mathcal{C}$.
- $Q := \{\mathcal{C}' \mid \mathcal{C}' \otimes \mathcal{C}_1 = \vee U\}$. Quotient set: contracts whose composition with $\mathcal{C}_1$ is largest while refining $\mathcal{C}$.

The definition of quotient in the meta-theory is given by $\mathcal{C}/\mathcal{C}_1 = \vee S$. From the definitions we introduced, it is clear that $\mathcal{C}/\mathcal{C}_1 = \vee Q$. We now introduce another set of contracts which we will show is equivalent to $S$ (Lemma 4.3).

***Definition 4.2.*** Given contracts $\mathcal{C}$ and $\mathcal{C}_1$, we define the following set of contracts:

$$\mathbf{C}_{\mathcal{C}/\mathcal{C}_1} := \left\{ \mathcal{C}' \;\middle|\; \begin{array}{l} M_1 \times E \models^E \mathcal{C}', \quad M' \times E \models^E \mathcal{C}_1, \\ \text{and } M' \times M_1 \models^M \mathcal{C} \text{ for all} \\ E, M_1, M' \text{ such that } E \models^E \mathcal{C}, \\ M_1 \models^M \mathcal{C}_1, \text{ and } M' \models^M \mathcal{C}' \end{array} \right\}.$$

***Lemma 4.3.*** $\mathbf{C}_{\mathcal{C}/\mathcal{C}_1} = S$.

It follows that we can define the contract in the meta-theory as $\mathcal{C}/\mathcal{C}_1 = \vee \mathbf{C}_{\mathcal{C}/\mathcal{C}_1}$. This definition is analogous to the definition of composition ($\mathcal{C}_1 \otimes \mathcal{C}' = \wedge \mathbf{C}_{\mathcal{C}_1,\mathcal{C}'}$). Finally, we can define the lower quotient operation for the meta-theory as $(\mathcal{C}/\mathcal{C}_1)_- = \wedge Q$. We now show that this definition of quotient in the meta-theory readily leads to a derivation of the quotient operation for assume-guarantee contracts.

## 4.3. A second derivation of the quotient operation of assume-guarantee contracts

We use the new definition of the contract quotient to derive the quotient operation of A/G contracts. The key is expressing $\mathbf{C}_{\mathcal{C}/\mathcal{C}_1}$ in the A/G framework. We do this in the following lemma:

***Lemma 4.4.*** Let $\mathcal{C}$, $\mathcal{C}_1$, and $\mathcal{C}'$ be saturated contracts. $\mathbf{C}_{\mathcal{C}/\mathcal{C}_1}$ in the A/G framework is given as follows:

$$\mathbf{C}_{\mathcal{C}/\mathcal{C}_1} = \left\{ (A', G') \;\middle|\; \begin{array}{l} (A', G') \text{ is saturated,} \\ A \cap G' \subseteq A_1, \\ A \cap G_1 \subseteq A', \text{ and} \\ G' \cap G_1 \subseteq G \end{array} \right\}. \quad (9)$$

We said we can define the quotient operation as $\mathcal{C}/\mathcal{C}_1 = \vee \mathbf{C}_{\mathcal{C}/\mathcal{C}_1}$. Using Lemma 4.4 we obtain another derivation of the A/G quotient operation. First we write the quotient in the A/G framework:

$$\mathcal{C}/\mathcal{C}_1 = \max \left\{ (A', G') \;\middle|\; \begin{array}{l} (A', G') \text{ is saturated,} \\ A \cap G' \subseteq A_1, \\ A \cap G_1 \subseteq A', \text{ and} \\ G' \cap G_1 \subseteq G \end{array} \right\}.$$

From this expression, we obtain the assumptions and guarantees:

$$\begin{aligned} a(\mathcal{C}/\mathcal{C}_1) &= A \cap G_1 \quad \text{and} \\ g(\mathcal{C}/\mathcal{C}_1) &= \max \left\{ G' \;\middle|\; \begin{array}{ll} G' \cap A \subseteq A_1 & \text{and} \\ G' \cap G_1 \subseteq G & \end{array} \right\} \\ &= (A_1 \cup \neg A) \cap (G \cup \neg G_1) \\ &= A_1 \cap G \cup \neg(A \cap G_1). \end{aligned}$$

This derivation of the quotient operation was possible due to the new definition of the contract operation in the meta-theory of contracts.

## 5. Examples

Theorem 3.5 can be used as the basis of a decomposition methodology with contracts: suppose we have a top-level contract $\mathcal{C} = (A, G)$, and one component with contract $\mathcal{C}_1 = (A_1, G_1)$ will be used in the design. Then the set of refinements of $\mathcal{C}/\mathcal{C}_1$ is the same as the set of contracts that, when composed with $\mathcal{C}_1$, refine $\mathcal{C}$. Consequently, in order to synthesize a new contract $\mathcal{C}'$ with the property $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$, we can compute $\mathcal{C}/\mathcal{C}_1$, and we know that any refinement of $\mathcal{C}/\mathcal{C}_1$ will have the property we seek. Moreover, if we wish to synthesize $\mathcal{C}'$, we can seek a refinement of the quotient which is expressible with few parameters in order to make synthesis efficient.

We provide two examples of contract decomposition using the quotient. The first is a logic design example dealing with an Arithmetic Logic Unit (ALU), and the second is an automotive application dealing with Cooperative Adaptive Cruise Control (CACC). Our intention is to show that the

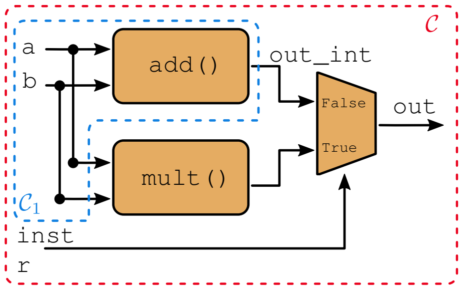

Figure 1. ALU diagram of example 5.1. The ALU is required to output either the sum or the product of the inputs `a` and `b`, according to the value of the input `inst`: if `inst` is deasserted, the output is the sum; otherwise, it is the product. Suppose there is a component available that implements the addition part of the specification. We expect the quotient to indicate that the multiplication remains to be implemented. We use monospace font to refer to constants, variables, and functions.

operations we derived do indeed provide what one would intuit are the missing specifications in a design, assuming that a given component will be used in the final system. To simplify our presentation, both of our examples meet the requirement that $A \cap G \subseteq A_1$, so according to Corollary 3.8, the quotient operation (6) becomes equal to $R(\mathcal{C}, \mathcal{C}_1)$ (see Definition 3.3); thus, we will repeatedly refer to $R$ as the quotient in these examples. Finally, both examples are given in LTL, but we show the propositions explicitly in order to carry out simplifications.

### 5.1. Implementing an ALU

Consider an ALU with the functionality shown in Figure 1. The design receives as input numbers `a` and `b`, a trigger signal `r`, and an instruction `inst`. When `r` is asserted, within $n$ time units the output `out` is equal to either the sum or the multiplication of `a` and `b`, according to the value of `inst`. For this ALU, we can write the top-level contract $\mathcal{C} = (A, G)$, with

$$G = \bigvee_{i=1}^{n} \Big\{ \big((\mathtt{r} \wedge \neg \mathtt{inst}) \rightarrow \mathbf{X}^i \mathtt{out} = \mathtt{add(a,b)}\big) \wedge \big((\mathtt{r} \wedge \mathtt{inst}) \rightarrow \mathbf{X}^i \mathtt{out} = \mathtt{mult(a,b)}\big) \Big\} \vee \neg A$$

$$\text{and } A = \mathtt{r} \rightarrow \left( \bigwedge_{i=1}^{n} (\mathbf{X}^i \neg \mathtt{r}) \wedge (\mathbf{X}^i \mathtt{a} = \mathtt{a}) \wedge (\mathbf{X}^i \mathtt{b} = \mathtt{b}) \wedge (\mathbf{X}^i \mathtt{inst} = \mathtt{inst}) \right),$$

i.e., we assume that once `r` asserts, `r` will remain deasserted during the next $n$ time ticks, and all other input signals will not change value during this time; the contract guarantees that at some point during the next $n$ clock ticks, the output will be equal to the addition or multiplication of `a` and `b`, according to the value of `inst`.

Now suppose we have a component which outputs the addition of `a` and `b` at the $n$-th time tick after the assertion of `r`. If we use this component in our design, we expect only the multiplication remains to be implemented. The component we are using in the design obeys the contract $\mathcal{C}_1 = (A_1, G_1)$ defined as follows:

$$G' = (\mathtt{r} \rightarrow \mathbf{X}^n \mathtt{out\_int} = \mathtt{add(a,b)}) \vee \neg A_1 \quad \text{and}$$
$$A' = \mathtt{r} \rightarrow \left( \bigwedge_{i=1}^{n} (\mathbf{X}^i \neg \mathtt{r}) \wedge (\mathbf{X}^i \mathtt{a} = \mathtt{a}) \wedge (\mathbf{X}^i \mathtt{b} = \mathtt{b}) \right).$$

$\mathcal{C}_1$ guarantees that the output `out_int` will be equal to the addition of the inputs `a` and `b` $n$ ticks after the assertion of `r`. Since $\mathcal{C}_1$ guarantees the addition part of the guarantees of $\mathcal{C}$, we expect the quotient operation to allow us to find a contract that only guarantees the multiplication of the inputs since this is what $\mathcal{C}_1$ is missing from $\mathcal{C}$. Let $\alpha = r \rightarrow \bigwedge_{i=1}^{n} (\mathbf{X}^i \mathtt{inst} = \mathtt{inst})$. We observe that $A = \alpha \wedge A_1$, so $A \subseteq A_1 \subseteq A_1 \cup \neg G_1$, meaning that $R(\mathcal{C}, \mathcal{C}_1) = (A_2, G_2)$ is the quotient operation (Corollary 3.8). Define $\beta = (\mathtt{r} \rightarrow \mathbf{X}^n \mathtt{out\_int} = \mathtt{add(a,b)})$. We compute

$$A_1 = A' \wedge \alpha \wedge \beta \quad \text{and}$$
$$G_1 = \bigvee_{i=1}^{n} \Big\{ \big((\mathtt{r} \wedge \neg \mathtt{inst}) \rightarrow \mathbf{X}^i \mathtt{out} = \mathtt{add(a,b)}\big) \wedge \big((\mathtt{r} \wedge \mathtt{inst}) \rightarrow \mathbf{X}^i \mathtt{out} = \mathtt{mult(a,b)}\big) \Big\} \vee \neg A_1.$$

Now we rewrite $G_2$ as

$$G_1 = \bigvee_{i=1}^{n-1} \Big\{ \big((\mathtt{r} \wedge \neg \mathtt{inst}) \rightarrow \mathbf{X}^i \mathtt{out} = \mathtt{add(a,b)}\big) \wedge \big((\mathtt{r} \wedge \mathtt{inst}) \rightarrow \mathbf{X}^i \mathtt{out} = \mathtt{mult(a,b)}\big) \Big\} \vee \Big\{ \big((\mathtt{r} \wedge \neg \mathtt{inst}) \rightarrow \mathbf{X}^n \mathtt{out} = \mathbf{X}^n \mathtt{out\_int}\big) \wedge \big((\mathtt{r} \wedge \mathtt{inst}) \rightarrow \mathbf{X}^n \mathtt{out} = \mathtt{mult(a,b)}\big) \Big\} \vee \neg A_1.$$

Since $A \subseteq A_1 \cup \neg G$, any refinement of $R$ refines $\mathcal{C}$ when composed with $\mathcal{C}_1$. We propose the refinement $\mathcal{C}' = (A', G')$ with

$$G_2 = \big((\mathtt{r} \wedge \neg \mathtt{inst}) \rightarrow \mathbf{X}^n \mathtt{out} = \mathbf{X}^n \mathtt{out\_int}\big) \wedge \big((\mathtt{r} \wedge \mathtt{inst}) \rightarrow \mathbf{X}^n \mathtt{out} = \mathtt{mult(a,b)}\big) \vee \neg A_2 \text{ and}$$
$$A_2 = A_1 \wedge \alpha.$$

Contract $\mathcal{C}'$ is saturated and satisfies $\mathcal{C}' \leq R$. Since we also have $A \subseteq A_1 \cup \neg G$, Corollary 3.8 guarantees that $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$. We observe that $\mathcal{C}'$ only guarantees the multiplication of `a` and `b`, as we intuited, and that it does not make any assumption about `out_int`; in particular, while $R$ keeps a complete description of the arithmetic relationship between `out_int` and the inputs `a` and `b`, this information is hidden from $\mathcal{C}'$.

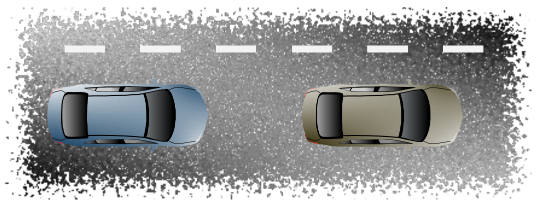

Figure 2. In Cooperative Adaptive Cruise Control, the vehicle on the left, $\mathcal{V}_a$, attempts to keep a fixed distance from the other vehicle, $\mathcal{V}_b$. To run its control algorithm, $\mathcal{V}_a$ must use estimates of its own state and of the state of vehicle $\mathcal{V}_b$: the more faithful the estimates, the more reliable the functionality. Suppose that $\mathcal{V}_b$ can estimate its own state much better than $\mathcal{V}_a$ can estimate it. $\mathcal{V}_b$ can share wirelessly the measurements of its state with $\mathcal{V}_a$ for the latter to make better control decisions.

## 5.2. Connected Vehicles

Connectivity with other vehicles or with the infrastructure can be used to extend the capabilities of a vehicle by increasing the number, types, and quality of sensors that it can access, or by providing more computational capabilities that the vehicle can use to tackle a task.

Consider the application of connectivity to Cooperative Adaptive Cruise Control (CACC). We consider the scenario of Figure 2: a vehicle $\mathcal{V}_a$, moving with velocity $v_a$, attempts to drive on the highway at a certain distance from vehicle $\mathcal{V}_b$. As the top-level spec $\mathcal{C} = (A, G)$, we ask that the distance $d_r$ between the two vehicles be contained in intervals that depend on the speed at which $\mathcal{V}_a$ moves. Thus, we have the guarantees

$$G = \bigwedge_{i \in \mathcal{I}} v_a \in V_i \to d_r \in [L_i^-, L_i^+] \vee \neg A,$$

where $V_i$ stand for ranges of velocities indexed over a set $\mathcal{I}$; $L_i^+$ and $L_i^-$ are real numbers which serve as the limits of $d_r$ for various values of the speed $v_a$.

We observe that vehicle $\mathcal{V}_a$, in order to follow vehicle $\mathcal{V}_b$, must know something about its own state and the state of $\mathcal{V}_b$. Let $x_i$ be the state of vehicle $\mathcal{V}_i$. Assume that $\hat{x}_i$ are the observations which vehicle $\mathcal{V}_a$ makes of the states of each car and which it uses to make control decisions. Note that the $x_i$ are the actual state variables that correspond to reality, but the $\hat{x}_i$ are the state observations that $\mathcal{V}_a$ can access either by making the measurement using its sensors or by receiving data from external sources ($\mathcal{V}_b$ in this case). These observations are affected by the intrinsic accuracy of the sensors, and by the time delays which exist from the analog-to-digital converters that capture physical data to the processors that implement the CACC. These time delays are crucial since, for instance, even if $\mathcal{V}_a$ has access to an extremely accurate velocity measurement of $\mathcal{V}_b$ that was captured 10 minutes ago, the measurement is useless for the purposes of CACC, which operates in real-time.

In order to be able to guarantee its behavior $G$, $\mathcal{V}_a$ must assume that the states of the vehicles are constrained to reasonable values (e.g., that the speeds of the vehicles are bounded) and that the state observations made by $\mathcal{V}_a$ are faithful to the reality up to a known tolerance. Then, we can write the assumptions for the contract (we assume that operations are evaluated in the order $\neg$, $\wedge$, $\to$, $\vee$):

$$\begin{aligned} A =& e \in E \wedge x_a \in \mathcal{X}_a \wedge x_b \in \mathcal{X}_b \wedge \\ & \bigwedge_{i \in \mathcal{I}} v_a \in V_i \to D \in [L_i^-, L_i^+]. \end{aligned}$$

In these expressions, $e$ represents the current state of the environment; $E$ is a set of restrictions of possible behaviors of the environment; the $\mathcal{X}_j$ are restrictions on possible behaviors of the states of the cars; and $D$ is a setting configured by the user and which determines how closely she wishes car $\mathcal{V}_a$ to follow $\mathcal{V}_b$. Note that the contract assumes that the user sets $D$ to a value bounded by the limits $L_i^-$ and $L_i^+$.

In summary, contract $\mathcal{C}$ assumes that the environment and vehicles behave within certain limits (given by the sets), and guarantees that the relative distance between the cars will be bounded by a certain interval, according to the velocity of the first car. We observe that $d_r$ and $v_i$ can be derived from the state variables $x_i$.

Suppose that when the driver sets the value of $D$, the CACC controller of $\mathcal{V}_a$ is capable of making $\mathcal{V}_a$ stay at a distance $D$ from vehicle $\mathcal{V}_b$ up to a tolerance $f_\varepsilon$, i.e., the CACC guarantees $|d_r - D| < f_\varepsilon$. This tolerance $f_\varepsilon$ reflects the capability of the CACC control system and depends on the states of both vehicles and on how well the sensors of $\mathcal{V}_a$ match reality. The dependence of $f_\varepsilon$ on the states is obvious since it should be harder to keep a fixed distance to a moving target when that target is accelerating, for instance. The dependence of $f_\varepsilon$ on the state observations reflects the fact that the more faithfully the estimates match reality, the more reliably the control system behaves. Therefore, $f_\varepsilon$ is a function of many arguments: $f_\varepsilon = f_\varepsilon(\|x_a - \hat{x}_a\|, \|x_b - \hat{x}_b\|, x_a, x_b, e)$, where $\|\cdot\|$ is a semi-norm used to tell how well the observations $\hat{x}_i$ match reality $x_i$. To shorten our expressions, we will simply use $f_\varepsilon$ for $f_\varepsilon(\|x_a - \hat{x}_a\|, \|x_b - \hat{x}_b\|, x_a, x_b, e)$. We can write a contract $\mathcal{C}_1 = (A_1, G_1)$ for $\mathcal{V}_a$:

$$\begin{aligned} A_1 =& A, \quad \text{and} \\ G_1 =& |d_r - D| < f_\varepsilon \vee \neg A, \end{aligned}$$

Since $A \subseteq A_1 \cup \neg G$, the refinements of $R = (A_R, G_R)$ also refine $\mathcal{C}$ in composition with $\mathcal{C}_1$. We compute

$$\begin{aligned} A_R =& A \wedge |d_r - D| < f_\varepsilon \quad \text{and} \\ G_R =& \bigwedge_{i \in \mathcal{I}} v_a \in V_i \to d_r \in [L_i^-, L_i^+] \wedge |d_r - D| < f_\varepsilon \\ & \vee \neg A_R. \end{aligned}$$

Now that we have computed the quotient, we know that $R$ contains the part of $\mathcal{C}$ not met by $\mathcal{C}_1$. Observe that up to this point all manipulations have been mechanical: we have simply computed the quotient starting from the specifications. We are interested, however, in a refinement of the quotient that we can implement, and to find this good refinement we need knowledge from the designer. As designers, we introduce a clause in the guarantees to bound the value of the tolerance. Define the proposition $c$ as follows:

$$c = f_\varepsilon < \min(L_i^+ - D, D - L_i^-).$$

We can now write a contract $\mathcal{C}' = (A', G')$ that refines $R$ using this clause:

$$\begin{aligned} A' =& A_R \quad \text{and} \\ G' =& \bigwedge_{i \in \mathcal{I}} v_a \in V_i \to d_r \in [L_i^-, L_i^+] \wedge |d_r - D| < f_\varepsilon \wedge c \\ & \vee \neg A', \end{aligned}$$

We observe that $A \subseteq A'$; thus, $A'$ contains the clause $v_a \in V_i \to D \in [L_i^-, L_i^+]$ for all $i \in \mathcal{I}$. We can use this clause to rewrite $G'$ as follows:

$$\begin{aligned} G' = \bigwedge_{i \in \mathcal{I}} v_a \in V_i \to & \begin{pmatrix} d_r \in [L_i^-, L_i^+] \wedge \\ |d_r - D| < f_\varepsilon \wedge \\ f_\varepsilon < \min(L_i^+ - D, D - L_i^-) \wedge \\ D \in [L_i^-, L_i^+] \end{pmatrix} \\ \vee \neg A'. \end{aligned}$$

The clause $d_r \in [L_i^-, L_i^+]$ is redundant and can thus be eliminated, resulting in

$$\begin{aligned} G' = \bigwedge_{i \in \mathcal{I}} v_a \in V_i \to & \begin{pmatrix} |d_r - D| < f_\varepsilon \wedge \\ f_\varepsilon < \min(L_i^+ - D, D - L_i^-) \wedge \\ D \in [L_i^-, L_i^+] \end{pmatrix} \\ \vee \neg A'. \end{aligned}$$

Finally, since $A'$ contains the clauses $v_a \in V_i \to D \in [L_i^-, L_i^+]$ and $|d_r - D| < f_\varepsilon$, we can write $G'$ as

$$G' = \bigwedge_{i \in \mathcal{I}} v_a \in V_i \to f_\varepsilon < \min(L_i^+ - D, D - L_i^-) \vee \neg A'.$$

We observe that the guarantees of $G'$ do not directly refer to $d_r$; thus, we can further refine $\mathcal{C}'$ with a contract $\mathcal{C}'' = (A'', G'')$ as follows:

$$\begin{aligned} A'' =& A \quad \text{and} \\ G'' =& \bigwedge_{i \in \mathcal{I}} v_a \in V_i \to f_\varepsilon < \min(L_i^+ - D, D - L_i^-) \vee \neg A''. \end{aligned}$$

Since $\mathcal{C}''$ is a refinement of the quotient, we know it can be used to complete the design. But now we ask, how can we implement contract $\mathcal{C}''$? We started from a high level requirement on the relative distance between the two vehicles and through the quotient obtained a restriction on $f_\varepsilon = f_\varepsilon(\|x_a - \hat{x}_a\|, \|x_b - \hat{x}_b\|, x_a, x_b, e)$, the bounds on $\mathcal{V}_a$'s ability to stay close to $\mathcal{V}_b$. From a design standpoint, a design external to $\mathcal{V}_a$ has no control over the states of the vehicles $x_i$ or over the environment $e$; however, an external component can impact $\|x_b - \hat{x}_b\|$, i.e., how closely vehicle $\mathcal{V}_a$ can track the state of $\mathcal{V}_b$. While a component of the observation error of $x_b$ by $\mathcal{V}_a$ is comprised of the sensors used, contract $\mathcal{C}''$ can potentially be implemented by guaranteeing that network latency stays within acceptable bounds; that is, the network guarantees that vehicle $\mathcal{V}_a$ always maintains updated information about the state of $\mathcal{V}_b$, enabling the control system of $\mathcal{V}_a$ to make better control decisions.

## 5.3. Observations

We now discuss some commonalities that emerged in the manipulation of contracts in our examples.

1) Once the quotient $\mathcal{C}_q = (A_q, G_q)$ is computed, the next step in a decomposition methodology is to compute a refinement $\mathcal{C}'$ of $\mathcal{C}_q$ (recall that in composition with $\mathcal{C}_1$ (the given component contract), this contract is guaranteed to refine $\mathcal{C}$). To do this, we manipulate the contract $\mathcal{C}_q$ by adding and removing clauses. First, we observe that $A_q$ can often be expressed as a conjunction of clauses, i.e., $A_q = \bigwedge_i a_i$, and in our ALU and automotive examples $G_q$ is of the form $G_q = \bigwedge_j (p_j \to s_j) \vee \neg A_q$. Observe that $p \to s \vee \neg a = p \to s \wedge a \vee \neg a$. This equivalence was used to insert an assumption clause into the guarantees in the examples in order to carry out simplifications in $s \wedge a$. Further, conjoining clauses to $s$ results in a subset of $G_q$, providing a refinement of $\mathcal{C}_q$.

2) The saturation of the quotient implies that $G_q$ contains all the information needed to carry out simplifications, i.e., no other clauses are needed.

3) Refining the contract quotient involves a notion of what constitutes a good refinement, and set-theoretic operations enriched with simplifications allowed in the formalism in which assumptions and requirements are expressed. The former sets the goal of the simplification process (and thus needs guidance from the designer), while the latter is a systematic mechanism which can be automated. The mechanization of this task is correlated with the identification of the platforms that a given industrial vertical segment will adopt to specify contracts. Here we anticipate that the availability of tools supporting design for a given formalism would stimulate industry adoption. At the same time, a determination of the platforms to be used in an industrial vertical domain would encourage academic research to develop the "right" formalism needed to manipulate contracts for that platform. The question now is how to spark this win-win interaction.

## 6. Conclusions

We have introduced the notion of quotient set for the theory of contracts, and an explicit formula to compute the quotient operation between two assume-guarantee contracts. We illustrated a decomposition methodology of high-level specifications based on the A/G quotient operation. Some lines of future research are the use of the lower quotient operation to devise specifications with redundancy (to ensure reliability) and the creation of software tools that exploit these concepts to decompose specifications.

## Acknowledgments

We gratefully acknowledge helpful conversations with Shinichi Shiraishi, Eric Kim, and Kosuke Watanabe during the preparation of this manuscript. We thank Professor Albert Benveniste and our anonymous reviewers for their valuable suggestions for improving this work.

This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA, by NSF Contract CPS Medium 1739816, and by Toyota InfoTechnology Center, U.S.A., Inc.

## Appendix

**Proof of Lemma 3.2.** Suppose $G' \not\subseteq G \cup \neg G_1$, i.e., $\emptyset \neq G' \cap \neg(G \cup \neg G_1) = G' \cap \neg G \cap G_1$. Then $G' \cap G_1 \not\subseteq G$, which contradicts $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$. Thus, $G' \subseteq G \cup \neg G_1$, proving (2), and $G' \cap G_1 \subseteq (G \cup \neg G_1) \cap G_1 = G \cap G_1$, and (3) holds. Now we prove (4). Since $\mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}$, we have

$$A \subseteq (A_1 \cap A') \cup \neg G_1 \cup \neg G' \qquad (10)$$

For (10) to hold, we must have

$$\begin{aligned} \neg G' &\supseteq A \cap \neg\left((A_1 \cap A') \cup \neg G_1\right) \\ &= A \cap G_1 \cap (\neg A_1 \cup \neg A'). \end{aligned}$$

Equation (10) becomes

$$\begin{aligned} &(A_1 \cap A') \cup \neg G_1 \cup \neg G' \\ &\quad \supseteq (A_1 \cap A') \cup \neg G_1 \cup (A \cap G_1 \cap \neg(A_1 \cap A')) \\ &\quad = (A_1 \cap A') \cup \neg G_1 \cup (A \cap G_1 \cap \neg(A_1 \cap A')) \\ &\quad\quad \cup (A \cap G_1) \cup A = A \cup \neg G_1 \cup (A_1 \cap A') \supseteq A \cup \neg G_1, \end{aligned}$$

proving (4). Now, from (10), $A \cap G' \cap G_1 \subseteq A_1 \cap A' \Rightarrow A \cap G' \cap G_1 \subseteq A'$. Due to the saturation of $\mathcal{C}'$, we also have $A' \supseteq \neg G'$; then $A' \supseteq A \cap G' \cap G_1 \cup \neg G' = A \cap G_1 \cup \neg G' \Rightarrow A' \supseteq A \cap G_1$, proving (5).

**Proof of Corollary 3.8.** Let $A \cap G \subset A_1$. Then $G_1 \cap (A_1 \cap G \cup \neg A) = G_1 \cap (A_1 \cap G \cup \neg A \cup G \cap \neg A_1) = G_1 \cap G$.

Therefore, we have

$$\begin{aligned} Q(\mathcal{C}, \mathcal{C}_1) &= \left\{ \mathcal{C}' \,\middle|\, \begin{array}{l} \mathcal{C}' \text{ is saturated,} \\ g(\mathcal{C}' \otimes \mathcal{C}_1) = G_1 \cap (A_1 \cap G \cup \neg A), \\ a(\mathcal{C}' \otimes \mathcal{C}_1) = A \cup \neg G_1 \end{array} \right\} \\ &= \left\{ \mathcal{C}' \,\middle|\, \begin{array}{l} \mathcal{C}' \text{ is saturated,} \\ g(\mathcal{C}' \otimes \mathcal{C}_1) = G_1 \cap G, \\ a(\mathcal{C}' \otimes \mathcal{C}_1) = A \cup \neg G_1 \end{array} \right\}. \end{aligned}$$

Moreover, $(\mathcal{C}/\mathcal{C}_1)_- = (A \cup \neg G_1 \cup \neg A_1, G_1 \cap (A_1 \cap G \cup \neg A)) = (A \cup \neg G_1 \cup \neg A_1, G_1 \cap G) = L$.

Finally, we observe that $A_1 \cap G \cup \neg(A \cap G_1) = A_1 \cap G \cup \neg A \cup \neg G_1 = A_1 \cap G \cup \neg A \cup \neg G_1 \cup G \cap \neg A_1 = G \cup \neg G_1$. We thus have $\mathcal{C}/\mathcal{C}_1 = (A \cap G_1, A_1 \cap G \cup \neg(A \cap G_1)) = (A \cap G_1, G \cup \neg G_1) = R(\mathcal{C}, \mathcal{C}_1)$.

**Proof of Lemma 4.3.** We have $S = \{\mathcal{C}' \mid \mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}\}$. We observe we can write $S$ as

$$S = \left\{ \mathcal{C}' \,\middle|\, \begin{array}{l} \mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}, \\ M_1 \times E \models^E \mathcal{C}', \quad M' \times E \models^E \mathcal{C}_1, \\ \text{and } M' \times M_1 \models^M \mathcal{C}_1 \otimes \mathcal{C}' \text{ for all} \\ E, M_1, M' \text{ such that } E \models^E \mathcal{C}_1 \otimes \mathcal{C}', \\ M_1 \models^M \mathcal{C}_1, \text{ and } M' \models^M \mathcal{C}' \end{array} \right\}$$

since we have just conjoined a true statement to the conditions that define the set. Now, since $\mathcal{C}_1 \otimes \mathcal{C}' \leq \mathcal{C}$ within the conditions of $S$, $E \models^E \mathcal{C} \;\Rightarrow\; E \models^E \mathcal{C}_1 \otimes \mathcal{C}'$. Therefore, for any $M_1 \models^M \mathcal{C}_1$, $M' \models^M \mathcal{C}'$, and $E \models^E \mathcal{C}$, we have $M_1 \times E \models^E \mathcal{C}'$, $M' \times E \models^E \mathcal{C}_1$, and $M' \times M_1 \models^M \mathcal{C}_1 \otimes \mathcal{C}'$. Using again the fact that $\mathcal{C}_1 \otimes \mathcal{C}' \leq \mathcal{C}$ gives $M' \times M_1 \models^M \mathcal{C}$. Therefore, $S$ becomes

$$S = \left\{ \mathcal{C}' \,\middle|\, \begin{array}{l} \mathcal{C}' \otimes \mathcal{C}_1 \leq \mathcal{C}, \\ M_1 \times E \models^E \mathcal{C}', \quad M' \times E \models^E \mathcal{C}_1, \\ \text{and } M' \times M_1 \models^M \mathcal{C} \text{ for all} \\ E, M_1, M' \text{ such that } E \models^E \mathcal{C}, \\ M_1 \models^M \mathcal{C}_1, \text{ and } M' \models^M \mathcal{C}' \end{array} \right\}.$$

We observe that the second condition means that $\mathcal{C} \in \mathbf{C}_{\mathcal{C}_1,\mathcal{C}'}$. This implies that $\mathcal{C}_1 \otimes \mathcal{C}' = \wedge \mathbf{C}_{\mathcal{C}_1,\mathcal{C}'} \leq \mathcal{C}$, which means that the first condition is redundant. We can thus write $S$ as

$$S = \left\{ \mathcal{C}' \,\middle|\, \begin{array}{l} M_1 \times E \models^E \mathcal{C}', \quad M' \times E \models^E \mathcal{C}_1, \\ \text{and } M' \times M_1 \models^M \mathcal{C} \text{ for all} \\ E, M_1, M' \text{ such that } E \models^E \mathcal{C}, \\ M_1 \models^M \mathcal{C}_1, \text{ and } M' \models^M \mathcal{C}' \end{array} \right\},$$

that is, $S = \mathbf{C}_{\mathcal{C}/\mathcal{C}_1}$.

**Proof of Lemma 4.4.** (This proof follows closely that of Lemma 4 in [2].)

($\subseteq$). Let $\mathcal{C}' = (A', G')$ be a saturated contract in $\mathbf{C}_{\mathcal{C}/\mathcal{C}_1}$. Let $E = A$, $M_1 = G_1$, and $M' = G'$. Since $\mathcal{C}' \in \mathbf{C}_{\mathcal{C}/\mathcal{C}_1}$, we have $M_1 \times E \models^E \mathcal{C}'$, $M' \times E \models^E \mathcal{C}_1$, and $M' \times M_1 \models^M \mathcal{C}$. These three expressions are equivalent to $G_1 \cap A \subseteq A'$, $G' \cap A \subseteq A_1$, and $G' \cap G_1 \subseteq G$, respectively. Hence, $\mathcal{C}'$ belongs to the set on the right hand side of 9.

($\supseteq$). Let $\mathcal{C}' = (A', G')$ belong to the set on the right hand side of 9. Let $E \subseteq A$, $M_1 \subseteq G'$, and $M_1 \subseteq G_1$. We have

$$\begin{aligned} &E \times M_1 = E \cap M_1 \subseteq A \cap G_1 \subseteq A' \Rightarrow E \times M_1 \models^E \mathcal{C}' \\ &E \times M' = E \cap M' \subseteq A \cap G' \subseteq A_1 \Rightarrow E \times M' \models^E \mathcal{C}_1 \\ &M_1 \times M' = M_1 \cap M' \subseteq G_1 \cap G' \subseteq G \\ &\quad \Rightarrow M_1 \times M' \models^M \mathcal{C}. \end{aligned}$$

Thus, $\mathcal{C}' \in \mathbf{C}_{\mathcal{C}/\mathcal{C}_1}$, proving the left set inclusion.

## References

[1] B. Meyer, *Touch of Class: Learning to Program Well Using Object Technology and Design by Contracts.* Springer, Software Engineering, 2009.

[2] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, "Contracts for system design," *Foundations and Trends® in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.

[3] F. Balarin, M. D'Angelo, A. Davare, D. Densmore, T. Meyerowitz, R. Passerone, A. Pinto, A. Sangiovanni-Vincentelli, A. Simalatsar, Y. Watanabe, G. Yang, and Q. Zhu, *Chapter 10. Platform-Based Design and Frameworks*. Cham: CRC Press, 2009, pp. 259–322.

[4] T. Villa, N. Yevtushenko, R. Brayton, A. Mishchenko, A. Petrenko, and A. Sangiovanni-Vincentelli, *The Unknown Component Problem: Theory and Applications*. Springer, 2012.

[5] T. T. H. Le, R. Passerone, U. Fahrenberg, and A. Legay, "Contract-based requirement modularization via synthesis of correct decompositions," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 2, pp. 33:1–33:26, Feb. 2016.

[6] C. Chilton, B. Jonsson, and M. Kwiatkowska, "Compositional assume-guarantee reasoning for input/output component theories," *Science of Computer Programming*, vol. 91, pp. 115–137, 2014, special Issue on Formal Aspects of Component Software (Selected Papers from FACS'12).

[7] T. Chen, C. Chilton, B. Jonsson, and M. Kwiatkowska, *A Compositional Specification Theory for Component Behaviours*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 148–168.

[8] P. Bhaduri and S. Ramesh, "Interface synthesis and protocol conversion," *Formal Asp. Comput.*, vol. 20, no. 2, pp. 205–224, 2008.

[9] J. Raclet, "Residual for component specifications," *Electr. Notes Theor. Comput. Sci.*, vol. 215, pp. 93–110, 2008.

[10] J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu, "Learning assumptions for compositional verification," in *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, 2003, pp. 331–346.

[11] R. Alur, P. Madhusudan, and W. Nam, "Symbolic compositional verification by learning assumptions," in *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, 2005, pp. 548–562.

[12] M. G. Bobaru, C. S. Pasareanu, and D. Giannakopoulou, "Automated assume-guarantee reasoning by abstraction refinement," in *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, 2008, pp. 135–148.

[13] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.

[14] D. Giannakopoulou, C. S. Pasareanu, and H. Barringer, "Assumption generation for software component verification," in *17th IEEE International Conference on Automated Software Engineering (ASE 2002), 23-27 September 2002, Edinburgh, Scotland, UK*, 2002, pp. 3–12.

[15] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, *Multiple Viewpoint Contract-Based Specification and Design*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 200–225.

[16] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)(FOCS)*, Sept 1977, pp. 46–57.

[17] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, Nov 1990.

[18] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Y. Lakhnech and S. Yovine, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 152–166.