The Algebra of Assume-Guarantee Reasoning[*]

Inigo Incer
Caltech & UC Berkeley
inigo@caltech.edu

Albert Benveniste
INRIA/IRISA Rennes
albert.benveniste@inria.fr

Alberto Sangiovanni-Vincentelli
UC Berkeley
alberto@berkeley.edu

This paper presents the algebra of assume-guarantee (AG) contracts. It defines contracts, provides new and known operations, and shows all ways in which these operations are related. Contracts are functorial: any Boolean algebra has an associated contract algebra. We study monoid and semiring structures in contract algebra—and the mappings between such structures. We discuss the actions of a Boolean algebra on its contract algebra.

1 Introduction

AG contracts can be understood as formal specifications split in two parts: (i) assumptions made on the environment, and (ii) responsibilities assigned to the object satisfying the specification when it is instantiated in an environment which meets the assumptions of the contract. Contracts were introduced to streamline the integration of complex systems and to support concurrent design. System integration pertains the composition of multiple design objects into a coherent whole. For example, suppose a company wishes to implement a system with a specification $\mathscr{C}$; they may realize that there are two sub-specifications $\mathscr{C}_1$ and $\mathscr{C}_2$ such that the composition of their implementations always yields an implementation for the top level specification. In the language of AG contracts, we would say that the composition of $\mathscr{C}_1$ and $\mathscr{C}_2$, written $\mathscr{C}_1 \parallel \mathscr{C}_2$, *refines* $\mathscr{C}$. This company may now develop an implementation M_1 for $\mathscr{C}_1$, and provide $\mathscr{C}_2$ to a third-party OEM for them to develop an implementation M_2. If M_2 is an implementation for $\mathscr{C}_2$, the original company knows that M_1 and M_2 can be composed and that this composition meets the top-level specification $\mathscr{C}$. In this setting, frictions in the supply chain are alleviated as companies exchange formal specifications expressed as contracts.

An additional use of contracts is as follows. Suppose our company wants to implement a system with a specification $\mathscr{C}$ using a component M_1 with specification $\mathscr{C}_1$ that is not sufficient to implement $\mathscr{C}$. Contracts provide an operation called *quotient* which yields the specification whose implementation is *exactly* the component M' such that M_1 composed with M' meets the specification $\mathscr{C}$. The operation of quotient has uses in synthesis (when we have made incremental progress towards meeting a goal) and in every situation where we need to find *missing components*.

To say that contracts support concurrent design refers to another aspect of the design process. The design of some components involves multiple engineers working on different aspects of the same object. For example, a team may work on the functionality aspects of an integrated circuit, while another works on its timing characterization. If the functionality team generates a specification $\mathscr{C}_f$, and the timing team generates a specification $\mathscr{C}_t$, the two teams can combine their specs into a single contract object $\mathscr{C}$ through an operation called *merging*. In contract theory, these various aspects of a component are called *viewpoints*.

Influenced by the assume-guarantee reasoning of Floyd-Hoare logic [6, 8], the work on assume-guarantee specifications of Lamport and Abadi [1, 12], design by contract by Meyer [14], interface

---

[*]This paper is based on Chapter 6 of [10].

automata by de Alfaro and Henzinger [2], and applications of formal specifications to cyber-physical systems [5], it was realized that formal assume-guarantee specifications could be used to design and analyze any cyber-physical system, including their discrete and continuous aspects. Assume-guarantee contracts were thus introduced for this purpose by Benveniste et al. in [3]. Cyber-physical-system design methodologies using contracts were described in [16, 18].

This paper presents the algebra of assume-guarantee contracts, arranged around two questions of practical relevance (the related discussions are written in blue). *Question 1: how can multiple specifications be combined to generate a system specification?* This leads us to consider binary operations on assume-guarantee contracts and their uses in the system design process. The question has a corollary: *when a system is decomposed across multiple suppliers, as well as across multiple viewpoints, is there a right order for applying the contract operations? Does the result depend on this order?* This will lead us to consider how the various operations interact among themselves.

The second question has to do with computational complexity. We know that abstracting specifications tends to yield computationally-friendlier semi decision procedures. *Question 2: how can we compute contract abstractions?* Part of our answer to this question will make use of semiring actions.

The structure of the paper is as follows. Section 2 begins to address Question 1 by covering the standard definitions of contracts [4] and all known contract operations. The content of this section is review, except for the operations of *implication* and *coimplication*, which, to the best of our knowledge, have not been published. Section 3 treats contracts as an algebra associated with any Boolean algebra and presents a study of monoid and semiring structures within a contract algebra. In other words, this section deals with how the various contract operations interact with each other, yielding insight into compositional design methodologies using contracts. We use these results to define contract actions, which play a role in answering Question 2. Section 4 covers two ways in which a Boolean algebra can act on its contract algebra, and Section 5 discusses contract abstractions. Sections 3-5 summarize the contributions of this paper.

2   Assume-guarantee contracts

This section defines assume guarantee contracts and addresses Question 1: how can multiple specifications be combined to generate a system specification?

We discuss four binary operations that allow us to combine contracts. We also explore adjoint operations that allow us to carry out contract decompositions optimally. The content in this section borrows from [4, 15], and the references in the text. Its contributions are the closed-form expressions for implication and coimplication and the use of duality to unify the presentation of the contract operations.

Let $\mathcal{B}$ be a set called the *universe of behaviors*. Its elements are called *behaviors*. The universe of behaviors fixes the modeling formalism we have chosen in our application. A property is defined as a subset of $\mathcal{B}$. A component is also a subset of $\mathcal{B}$. The difference is semantics: we think of components as the set of behaviors they can display. Properties contain behaviors meeting a certain criterion. We say that a component M satisfies a property P, written $M \models P$ if $M \subseteq P$. Assume-guarantee contracts are pairs of properties.

Definition 2.1. *A contract $\mathcal{C}$ is a pair of properties $\mathcal{C} = (A, G)$. We call A* assumptions*, and G* guarantees.

Components can have two types of relationships with respect to a contract.

Definition 2.2. *Let $\mathcal{C} = (A, G)$ be a contract. We say that a component E is an environment for $\mathcal{C}$, written $E \models^E \mathcal{C}$, if $E \models A$.*

Environments are those components which meet the assumptions of a contract. Implementations are those which meet the guarantees of the contract when operating in an environment accepted by the contract.

Definition 2.3. *Let $\mathscr{C} = (A, G)$ be a contract. We say that a component M is an implementation for $\mathscr{C}$, written $M \models^M \mathscr{C}$, if $M \parallel E \models G$ for every environment E of $\mathscr{C}$.*

Now that we have definitions for environments and implementations, we define a relation on contracts that declares two contracts equivalent when they have the same environments and the same implementations:

Definition 2.4. *Let $\mathscr{C}$ and $\mathscr{C}'$ be two contracts. We say they are equivalent when they have the same environments and the same implementations.*

This means that for $\mathscr{C} = (A, G)$ and $\mathscr{C}' = (A', G')$ to be equivalent, we must have $A = A'$ and $G \cap A = G' \cap A' = G' \cap A$ (because $A' = A$). The largest G' meeting this condition is $G' = G \cup \neg A$ (the complement is taken with respect to $\mathscr{B}$). Enforcing this constraint for a contract allows us to have a unique mathematical object for each set of environments and implementations. We thus define an AG contract in canonical form as follows:

Definition 2.5. *A contract in canonical form is a contract $\mathscr{C} = (A, G)$ satisfying $A \cup G = \mathscr{B}$.*

From now on, we assume all contracts are in canonical form.

2.1 Duality

There is a unary operation which is helpful in revealing structure for AG contracts.

Definition 2.6. *Let $\mathscr{C} = (A, G)$ be a contract. We define a unary operation called reciprocal as follows: $C^{-1} = (G, A)$.*

This operation flips environments and implementations, i.e., it gives us the "environment view" of the specification $\mathscr{C}$. Note that the reciprocal respects canonicity.

Definition 2.7. *Let $\circ$ and $\star$ be two binary operations on AG contracts, we say that the operations are dual when $(\mathscr{C}_a \circ \mathscr{C}_b)^{-1} = \mathscr{C}_a^{-1} \star \mathscr{C}_b^{-1}$.*

2.2 Order

Definition 2.8. *Suppose $\mathscr{C}$ and $\mathscr{C}'$ are two contracts. We say that $\mathscr{C}$ is a refinement of $\mathscr{C}'$, written $\mathscr{C} \leq \mathscr{C}'$, when all implementations of $\mathscr{C}$ are implementations of $\mathscr{C}'$ and all environments of $\mathscr{C}'$ are environments of $\mathscr{C}$.*

The association we make of a specification being a refinement is that it is harder to meet than another. This is why we say that a specification accepting more environments is a refinement of one accepting less. We can express this order relation using assumptions and guarantees.

Proposition 2.9 (Theorem 5.2 of [4])**.** *Let $\mathscr{C} = (A, G)$ and $\mathscr{C}' = (A', G')$ be two contracts. Then $\mathscr{C} \leq \mathscr{C}'$ when $G \subseteq G'$ and $A' \subseteq A$.*

2.3 Conjunction and disjunction

The notion of order provides a lattice structure to AG contracts in canonical form. Given contracts $\mathscr{C} = (A, G)$ and $\mathscr{C}' = (A', G')$, their meet (GLB) and join (LUB) are given by

$$\mathscr{C} \wedge \mathscr{C}' = (A \cup A', G \cap G') \quad \text{and} \quad \mathscr{C} \vee \mathscr{C}' = (A \cap A', G \cup G').$$

We leave it to the reader to verify that conjunction and disjunction are monotonic with respect to the refinement order. Also, conjunction and disjunction furnish our first example of dual operations: $\mathscr{C} \wedge \mathscr{C}' = (G \cap G', A \cup A')^{-1} = ((G, A) \vee (G', A'))^{-1} = (\mathscr{C}^{-1} \vee \mathscr{C}'^{-1})^{-1}$.

If we interpret a contract as the entailment $A \Rightarrow G$, then contract conjunction is the conjunction of such entailments. Conjunction can be used to combine viewpoints. Disjunction has an application in product lines.

2.4 Composition

The notion of composition of AG contracts yields the specification of systems obtained from composing implementations of each of the contracts being composed. Contract composition formalizes how contracts with suppliers result in a system level contract. This operation is defined by axiom as follows:

Suppose $\mathscr{C}_1$ and $\mathscr{C}_2$ are two specifications to be composed. Call $\mathscr{C}$ the composite specification. Let M_1 and M_2 be arbitrary implementations of $\mathscr{C}_1$ and $\mathscr{C}_2$, respectively, and let E be any environment of $\mathscr{C}$. We define $\mathscr{C}$ to be the smallest contract satisfying the following constraints: the composite $M_1 \parallel M_2$ is an implementation of $\mathscr{C}$; the composite $M_1 \parallel E$ is an environment of $\mathscr{C}_2$; and the composite $M_2 \parallel E$ is an environment of $\mathscr{C}_1$.

The first requirement states that composing implementations of the specs being composed yields an implementation of the composite specification. The second requirement states that instantiating an implementation of $\mathscr{C}_1$ in an environment of the composite specification yields an environment for $\mathscr{C}_2$. And the last requirement is the analogous statement for $\mathscr{C}_1$. This principle, which states how to compose specifications split between environment and implementation requirements, was stated for the first time by M. Abadi and L. Lamport [1]. We can obtain a closed-form expression of this principle for AG contracts:

Proposition 2.10 (Theorem 5.2 of [4])**.** *Let* $\mathscr{C}_1 = (A_1, G_1)$ *and* $\mathscr{C}_2 = (A_2, G_2)$ *be two AG contracts. Their composition, denoted* $\mathscr{C}_1 \parallel \mathscr{C}_2$, *is given by* $\mathscr{C}_1 \parallel \mathscr{C}_2 = (A_1 \cap A_2 \cup \neg(G_1 \cap G_2), G_1 \cap G_2)$.

We state without proof an important property of composition:

Proposition 2.11. *Composition of AG contracts is monotonic with respect to the refinement order.*

2.5 Strong merging (or merging)

We said that AG contracts are used to handle the specifications of the various viewpoints of the same design element. Suppose $\mathscr{C}_1$ and $\mathscr{C}_2$ are specifications corresponding to different aspects to the same design object, e.g., functionality and power. We define their merger, denoted $\mathscr{C}_1 \bullet \mathscr{C}_2$, to be the contract which guarantees the guarantees of both specifications when the assumptions of both specifications are respected:

$$\mathscr{C}_1 \bullet \mathscr{C}_2 = (A_1 \cap A_2, G_1 \cap G_2 \cup \neg(A_1 \cap A_2)).$$

This contract is equivalent to contract $(A_1 \cap A_2, G_1 \cap G_2)$, which is exactly what we defined merging to be. Merging can be used to combine viewpoints. Merging and composition are duals, as pointed out in [15].

2.6 Adjoints

We have introduced four operations on AG contracts: two were obtained from the partial order, and two by axiom. Now we obtain the adjoints of these operations. Adjoints are used to compute optimal decompositions of contracts.

2.6.1 Quotient (or residual)

The adjoint of composition is called *quotient*. Let $\mathscr{C}$ and $\mathscr{C}'$ be two AG contracts. The quotient (also called residual in the literature), denoted $\mathscr{C}/\mathscr{C}'$, is defined as the largest AG contract $\mathscr{C}''$ satisfying $\mathscr{C}' \parallel \mathscr{C}'' \leq \mathscr{C}$.

Due to the fact that the quotient is the largest contract with this property, Proposition 2.11 tells us that any of its refinements has this property.

If we interpret $\mathscr{C}$ as a top-level specification that our system has to meet (e.g., the specification of a vehicle), and $\mathscr{C}'$ as the specification of a subset of the design for which we already have an implementation (e.g., a powertrain), then the quotient is the specification whose implementations are exactly those components that, if added to our partial design, would yield a system meeting the top-level specification. The following proposition gives us a closed-form expression for the quotient of AG contracts:

Proposition 2.12 (Theorem 3.5 of [9]). *Let $\mathscr{C} = (A, G)$ and $\mathscr{C}' = (A', G')$ be two AG contracts. The quotient, denoted $\mathscr{C}/\mathscr{C}'$, is given by $\mathscr{C}/\mathscr{C}' = (A \cap G', G \cap A' \cup \neg(A \cap G'))$.*

For an in-depth study of the notion of a quotient across several compositional theories, see [11]. We can readily show that

$$\mathscr{C}/\mathscr{C}' = \mathscr{C} \bullet (\mathscr{C}')^{-1}. \tag{1}$$

2.6.2 Separation

Just like composition has an adjoint operation (the quotient), merging has an adjoint. For contracts $\mathscr{C}$ and $\mathscr{C}'$, we define the operation of *separation*, denoted $\mathscr{C} \div \mathscr{C}'$, as the smallest contract $\mathscr{C}''$ satisfying $\mathscr{C} \leq \mathscr{C}' \bullet \mathscr{C}''$. This operation has a closed-form solution:

Proposition 2.13 (Theorem 3.12 of [17]). *Let $\mathscr{C} = (A, G)$ and $\mathscr{C}' = (A', G')$ be two AG contracts. Then $\mathscr{C} \div \mathscr{C}' = (A \cap G' \cup \neg(G \cap A'), G \cap A')$.*

Separation obeys $\mathscr{C} \div \mathscr{C}' = \mathscr{C} \parallel (\mathscr{C}')^{-1}$. From this identity and (1), it follows that quotient and separation are duals. For examples of merging and separation, see [17].

2.6.3 Implication and coimplication

Given contracts $\mathscr{C}$ and $\mathscr{C}'$, the definition of implication, denoted $\mathscr{C}' \to \mathscr{C}$, in a lattice is $\forall \mathscr{C}''. \mathscr{C}'' \wedge \mathscr{C}' \leq \mathscr{C} \Leftrightarrow \mathscr{C}'' \leq (\mathscr{C}' \to \mathscr{C})$. In other words, $\mathscr{C}' \to \mathscr{C}$ is the largest contract $\mathscr{C}''$ satisfying $\mathscr{C}'' \wedge \mathscr{C}' \leq \mathscr{C}$. The following proposition tells us how to compute this object:

Proposition 2.14. *Let $\mathscr{C} = (A, G)$ and $\mathscr{C}' = (A', G')$ be two contracts. Implication has the closed form expression $\mathscr{C}' \to \mathscr{C} = ((A \cap \neg A') \cup (G' \cap \neg G), G \cup \neg G')$.*

Dually, we can ask what is the smallest contract $\mathscr{C}''$ satisfying $\mathscr{C}'' \vee \mathscr{C}' \geq \mathscr{C}$. We will denote this object $\mathscr{C}' \nrightarrow \mathscr{C}$. A similar proof yields the following proposition.

Proposition 2.15. *Let $\mathscr{C} = (A, G)$ and $\mathscr{C}' = (A', G')$ be two contracts. The smallest contract $\mathscr{C}''$ satisfying $\mathscr{C}'' \vee \mathscr{C}' \geq \mathscr{C}$ has the closed form expression $\mathscr{C}' \nrightarrow \mathscr{C} = (A \cup \neg A', (G \cap \neg G') \cup (A' \cap \neg A))$.*

We observe that $\mathscr{C}' \to \mathscr{C} = (G \cup \neg G', (A \cap \neg A') \cup (G' \cap \neg G))^{-1} = ((\mathscr{C}')^{-1} \nrightarrow \mathscr{C}^{-1})^{-1}$, which shows that implication and coimplication are duals.

Conjunction	Disjunction
$\mathscr{C} \wedge \mathscr{C}' = (a \vee a', g \wedge g')$	$\mathscr{C} \vee \mathscr{C}' = (a \wedge a', g \vee g')$
Composition	Merging
$\mathscr{C}_1 \parallel \mathscr{C}_2 = (a_1 \wedge a_2 \vee \neg(g_1 \wedge g_2), g_1 \wedge g_2)$	$\mathscr{C}_1 \bullet \mathscr{C}_2 = (a_1 \wedge a_2, g_1 \wedge g_2 \vee \neg(a_1 \wedge a_2))$
Quotient	Separation
$\mathscr{C}/\mathscr{C}' = (a \wedge g', g \wedge a' \vee \neg(a \wedge g'))$	$\mathscr{C} \div \mathscr{C}' = (a \wedge g' \vee \neg(g \wedge a'), g \wedge a')$
Implication	Coimplication
$\mathscr{C}' \to \mathscr{C} = ((a \wedge \neg a') \vee (g' \wedge \neg g), g \vee \neg g')$	$\mathscr{C}' \nrightarrow \mathscr{C} = (a \vee \neg a', (g \wedge \neg g') \vee (a' \wedge \neg a))$

Table 1: Closed-form expressions of operations for contracts over a Boolean algebra

2.7 Summary of binary operations

The following diagram shows how all AG contract operations are related.
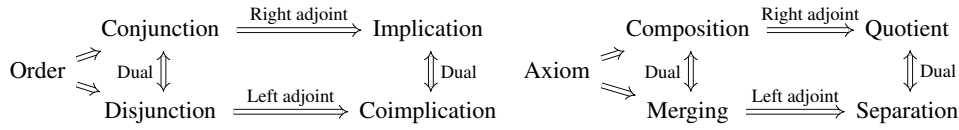
Two operations—conjunction and disjunction—come from the definition of order, and two—composition and merging—are defined by axiom. The rest of the operations are adjoints of these four.

3 Algebraic structures within contracts

In this section, we investigate the corollary of Question 1: Since both viewpoints and subsystems specifications need to be combined, does the order among these operations influence the result? If yes, is there a best order?

Inspection of the various binary formulas for AG contracts suggests that contracts can be defined over any Boolean algebra, not just that corresponding to properties over a set of behaviors. From now on, we deal with contracts defined over an arbitrary Boolean algebra and embark on a study of the relations between the various contract operations.

Let B be a Boolean algebra with bottom and top elements 0_B and 1_B, respectively. We form the contract algebra $\mathbf{C}(B)$ associated with B. The elements of $\mathbf{C}(B)$ are all pairs $(a,b) \in B^2$ such that $a \vee b = 1_B$. The notions of order and the binary operations work exactly the same as for AG contracts over sets of behaviors. Table 1 summarizes these operations.

The contract $1 = (0_B, 1_B)$ is larger than any contract. $0 = (1_B, 0_B)$ is smaller than any contract. The contract $e = (1_B, 1_B)$ is an identity for composition and merging. 1 is an identity for conjunction, and 0 for disjunction. Table 2 shows how various operations behave with respect to the distinguished elements.

3.1 Monoids

We begin to study the interactions among the various operations. We recall that a monoid is a semigroup with identity, i.e., a set together with an associative binary operation and an identity element for that operation. A contract algebra contains several monoids:

Proposition 3.1. $\mathbf{C}^M_\wedge(B) = (\mathbf{C}(B), \wedge, 1_B)$, $\mathbf{C}^M_\vee(B) = (\mathbf{C}(B), \vee, 0_B)$, $\mathbf{C}^M_\parallel(B) = (\mathbf{C}(B), \parallel, e)$, *and* $\mathbf{C}^M_\bullet(B) = (\mathbf{C}(B), \bullet, e)$ *are idempotent, commutative monoids.*

It turns out these monoids are isomorphic:

0	1	e
$\mathscr{C} \wedge 0 = 0$	$\mathscr{C} \wedge 1 = \mathscr{C}$	$(a,g) \wedge e = (1_B, g)$
$\mathscr{C} \vee 0 = \mathscr{C}$	$\mathscr{C} \vee 1 = 1$	$(a,g) \vee e = (a, 1_B)$
$\mathscr{C} \parallel 0 = 0$	$(a,g) \parallel 1 = (\neg g, g)$	$\mathscr{C} \parallel e = \mathscr{C}$
$(a,g) \bullet 0 = (a, \neg a)$	$\mathscr{C} \bullet 1 = 1$	$\mathscr{C} \bullet e = \mathscr{C}$
$\mathscr{C}/0 = 1$	$(a,g)/1 = (a, \neg a)$	$\mathscr{C}/e = \mathscr{C}$
$0/(a,g) = (g, \neg g)$	$1/\mathscr{C} = 1$	$e/\mathscr{C} = \mathscr{C}^{-1}$
$(a,g) \div 0 = (\neg g, g)$	$\mathscr{C} \div 1 = 0$	$\mathscr{C} \div e = \mathscr{C}$
$0 \div \mathscr{C} = 0$	$1 \div (a,g) = (\neg a, a)$	$e \div \mathscr{C} = \mathscr{C}^{-1}$
$(a,g) \to 0 = (g, \neg g)$	$\mathscr{C} \to 1 = 1$	$(a,g) \to e = (\neg a, 1_B)$
$0 \to \mathscr{C} = 1$	$1 \to \mathscr{C} = \mathscr{C}$	$e \to (a,g) = (\neg g, g)$
$\mathscr{C} \nrightarrow 0 = 0$	$(a,g) \nrightarrow 1 = (\neg a, a)$	$(a,g) \nrightarrow e = (1_B, \neg g)$
$0 \nrightarrow \mathscr{C} = \mathscr{C}$	$1 \nrightarrow \mathscr{C} = 0$	$e \nrightarrow (a,g) = (a, \neg a)$

Table 2: Contract operations and the distinguished elements

Proposition 3.2. *The monoids* $\mathbf{C}_\wedge^M(B)$, $\mathbf{C}_\vee^M(B)$, $\mathbf{C}_\parallel^M(B)$, *and* $\mathbf{C}_\bullet^M(B)$ *are isomorphic. Moreover, the following diagram commutes, where* $\theta_g(a,g) = (\neg(a \wedge g), g)$ *and* $\theta_a(a,g) = (a, \neg(a \wedge g))$:

$$
\begin{array}{ccc}
\mathbf{C}_\wedge^M(B) & \xleftarrow[\simeq]{(\cdot)^{-1}} & \mathbf{C}_\vee^M(B) \\
\theta_g \big\uparrow \wr & & \wr \big\downarrow \theta_a \\
\mathbf{C}_\parallel^M(B) & \xleftarrow[(\cdot)^{-1}]{\simeq} & \mathbf{C}_\bullet^M(B)
\end{array}
\tag{2}
$$

3.2 Maps between monoids

The previous result showed how to express contract operations in terms of others. Now we explore how to map contract monoids across their underlying Boolean algebras.

Suppose we have two Boolean algebras, B and B'. Due to Proposition 3.2, it is sufficient to study the structure of the maps between the contract monoids $\mathbf{C}_\parallel^M(B)$ and $\mathbf{C}_\parallel^M(B')$ in order to understand the structure of the maps between all contract monoids associated with each Boolean algebra. First we study maps that allow us to construct and split contracts. Then we consider the general maps.

Let $\mathbf{M}_\wedge(B)$ and $\mathbf{M}_\vee(B)$ be the monoids $\mathbf{M}_\wedge(B) = (B, \wedge, 1_B)$ and $\mathbf{M}_\vee(B) = (B, \vee, 0_B)$. We define the two monoid maps

$$
\iota_a : \mathbf{M}_\wedge(B) \to \mathbf{C}_\parallel^M(B) \quad a \mapsto (a, 1_B)
$$
$$
\iota_g : \mathbf{M}_\wedge(B) \to \mathbf{C}_\parallel^M(B) \quad g \mapsto (1_B, g).
$$

These maps generate an epic monoid map $\pi : \mathbf{M}_\wedge(B) \times \mathbf{M}_\wedge(B) \to \mathbf{C}_\parallel^M(B)$ defined as

$$
(a,g) \mapsto \iota_a(a) \parallel \iota_g(g) = (g \to a, g).
$$

Similarly, we have monoid maps that allow us to split a contract:

$$
\pi_g : \mathbf{C}_\parallel^M(B) \to \mathbf{M}_\wedge(B) \quad \pi_g(a,g) = g
$$
$$
\pi_a : \mathbf{C}_\wedge^M(B) \to \mathbf{M}_\vee(B) \quad \pi_a(a,g) = a.
$$

We use the monoid isomorphisms (2) to obtain a map $\mathbf{C}_\wedge^M(B) \to \mathbf{M}_\wedge(B)$ from the last morphism:

$$\neg \circ \pi_a \circ \theta_g \colon \mathbf{C}_\|^M(B) \to \mathbf{M}_\wedge(B)$$
$$(a,g) \mapsto a \wedge g.$$

The two maps $\mathbf{C}_\|^M(B) \to \mathbf{M}_\wedge(B)$ yield the monic monoid map

$$\iota \colon \mathbf{C}_\|^M(B) \to \mathbf{M}_\wedge(B) \times \mathbf{M}_\wedge(B)$$
$$(a,g) \mapsto (a \wedge g, g).$$

This map is left-invertible:

$$\pi \circ \iota = e.$$

The elementary maps just described enable us to find the general structure between the monoid maps between the parallel monoids corresponding to two Boolean algebras.

Theorem 3.3. *Let* $f \colon \mathbf{C}_\|^M(B) \to \mathbf{C}_\|^M(B')$. *Then we can write f as*

$$f = \pi \circ (l_a(ag)l_g(g)r_a(ag)r_g(g), r_a(ag)r_g(g)) \circ \iota,$$

where $l_a, l_g, r_a, r_g \colon \mathbf{M}_\wedge(B) \to \mathbf{M}_\wedge(B')$ *are monoid morphisms.*

3.3 Semirings

Now that we have four isomorphic monoids, we look for additional algebraic structure within the contract algebra, namely, the existence of semirings. This will capture the interactions between algebraic operations. First we study the distributivity of the binary operations. Distributivity and semi-distributivity answer the corollary of Question 1: Does the order between subspecification composition and viewpoint combination matter?

	Conjunction	Disjunction	Composition	Merging
Conjunction	$\mathscr{C} \wedge (\mathscr{C}' \wedge \mathscr{C}'') =$ $(\mathscr{C} \wedge \mathscr{C}') \wedge (\mathscr{C} \wedge \mathscr{C}'')$	$\mathscr{C} \wedge (\mathscr{C}' \vee \mathscr{C}'') =$ $(\mathscr{C} \wedge \mathscr{C}') \vee (\mathscr{C} \wedge \mathscr{C}'')$	$\mathscr{C} \wedge (\mathscr{C}' \| \mathscr{C}'') =$ $(\mathscr{C} \wedge \mathscr{C}') \| (\mathscr{C} \wedge \mathscr{C}'')$	$e \wedge (1 \bullet 0) \neq$ $(e \wedge 1) \bullet (e \wedge 0)$
Disjunction	$\mathscr{C} \vee (\mathscr{C}' \wedge \mathscr{C}'') =$ $(\mathscr{C} \vee \mathscr{C}') \wedge (\mathscr{C} \vee \mathscr{C}'')$	$\mathscr{C} \vee (\mathscr{C}' \vee \mathscr{C}'') =$ $(\mathscr{C} \vee \mathscr{C}') \vee (\mathscr{C} \vee \mathscr{C}'')$	$e \vee (1 \| 0) \neq$ $(e \vee 1) \| (e \vee 0)$	$\mathscr{C} \vee (\mathscr{C}' \bullet \mathscr{C}'') =$ $(\mathscr{C} \vee \mathscr{C}') \bullet (\mathscr{C} \vee \mathscr{C}'')$
Composition	$\mathscr{C} \| (\mathscr{C}' \wedge \mathscr{C}'') =$ $(\mathscr{C} \| \mathscr{C}') \wedge (\mathscr{C} \| \mathscr{C}'')$	$\mathscr{C} \| (\mathscr{C}' \vee \mathscr{C}'') =$ $(\mathscr{C} \| \mathscr{C}') \vee (\mathscr{C} \| \mathscr{C}'')$	$\mathscr{C} \| (\mathscr{C}' \| \mathscr{C}'') =$ $(\mathscr{C} \| \mathscr{C}') \| (\mathscr{C} \| \mathscr{C}'')$	$1 \| (0 \bullet e) \neq$ $(1 \| 0) \bullet (1 \| e)$
Merging	$\mathscr{C} \bullet (\mathscr{C}' \wedge \mathscr{C}'') =$ $(\mathscr{C} \bullet \mathscr{C}') \wedge (\mathscr{C} \bullet \mathscr{C}'')$	$\mathscr{C} \bullet (\mathscr{C}' \vee \mathscr{C}'') =$ $(\mathscr{C} \bullet \mathscr{C}') \vee (\mathscr{C} \bullet \mathscr{C}'')$	$0 \bullet (1 \| e) \neq$ $(0 \bullet 1) \| (0 \bullet e)$	$\mathscr{C} \bullet (\mathscr{C}' \bullet \mathscr{C}'') =$ $(\mathscr{C} \bullet \mathscr{C}') \bullet (\mathscr{C} \bullet \mathscr{C}'')$

Table 3: Distributivity of contract operations

Proposition 3.4. *Table 3 shows whether the binary operations displayed in the rows distribute over the binary operations in the columns.*

If we use conjunction to combine viewpoints, then the distributivity of composition over conjunction says that combining viewpoints and composing subspecifications can be performed at will, in any order. This is wrong if we use merging to combine viewpoints. Nevertheless, the following proposition expresses that, if we use merging to combine viewpoints, it is preferred to combine viewpoints first, and then compose subspecifications, than the converse.

Proposition 3.5. *Let $\mathscr{C}_i$ be a contract for $1 \le i \le 4$. We have the following semi-distributions:*

$$(C_1 \wedge C_2) \star (C_3 \wedge C_4) \le (C_1 \star C_3) \wedge (C_2 \star C_4) \quad and \quad (C_1 \vee C_2) \star (C_3 \vee C_4) \ge (C_1 \star C_3) \vee (C_2 \star C_4),$$

where $\star$ is any of the operations conjunction, disjunction, composition, or merging.

The sub-distributivity of parallel composition over conjunction is proved in Chapter 4 of [4]. We will now use the distributivity results to look for semiring structure within the algebra of contracts. We recall the definition of a semiring (see, e.g., [7]):

Definition 3.6. *A semiring $(R, \cdot, +, 1_R, 0_R)$ is a nonempty set R where (i) $(R, +, 0_R)$ is a commutative monoid; (ii) $(R, \cdot, 1_R)$ is a monoid; (iii) $r(s+t) = rs + rt$ and $(s+t)r = sr + tr$ for all $r, s, t \in R$; (iv) $r \cdot 0_R = 0_R \cdot r = 0_R$ for all $r \in R$; and (v) $0_R \neq 1_R$.*

A map of semirings $f : (R, \cdot, +, 1_R, 0_R) \to (R', \cdot, +, 1_{R'}, 0_{R'})$ satisfies $f(0_R) = f(0_{R'})$, $f(1_R) = f(1_{R'})$, $f(r+s) = f(r) + f(s)$, and $f(r \cdot s) = f(r) \cdot f(s)$.

The following result provides the semiring structures available in the contract algebra.

Proposition 3.7. *Using the operations of conjunction, disjunction, composition, and merging, we have exactly four semirings: the conjunction semiring $\mathbf{C}_\wedge^S(B) = (\mathbf{C}(B), \wedge, \vee, 1, 0)$, the disjunction semiring $\mathbf{C}_\vee^S(B) = (\mathbf{C}(B), \vee, \wedge, 0, 1)$, the composition semiring $\mathbf{C}_\|^S(B) = (\mathbf{C}(B), \|, \vee, e, 0)$, and the merging semiring $\mathbf{C}_\bullet^S(B) = (\mathbf{C}(B), \bullet, \wedge, e, 1)$.*

These four semirings have two isomorphisms.

Proposition 3.8. *We have the isomorphisms $\mathbf{C}_\wedge^S(B) \cong \mathbf{C}_\vee^S(B)$ and $\mathbf{C}_\|^S(B) \cong \mathbf{C}_\bullet^S(B)$. There are no isomorphisms between these two pairs.*

3.4 Some semiring maps

Let $\mathbf{S}_\wedge(B)$ and $\mathbf{S}_\vee(B)$ be, respectively, the semirings $(B, \wedge, \vee, 1, 0)$ and $(B, \vee, \wedge, 0, 1)$. We first observe that complementation is a semiring isomorphism for $\mathbf{S}_\wedge(B)$ and $\mathbf{S}_\vee(B)$. We define maps $\Delta_g : \mathbf{S}_\wedge(B) \to \mathbf{C}_\wedge^S(B)$ and $\iota_g : \mathbf{S}_\wedge(B) \to \mathbf{C}_\|^S(B)$ as follows: $\Delta_g(b) = (\neg b, b)$ and $\iota_g(b) = (1_B, b)$.

Proposition 3.9. *Δ_g and ι_g are semiring homomorphisms.*

Observe that Δ_g can be used to obtain a semiring map from $\mathbf{S}_\vee(B)$ to $\mathbf{C}_\vee^S(B)$ using the semiring isomorphisms $\neg : \mathbf{S}_\wedge(B) \xrightarrow{\sim} \mathbf{S}_\vee(B)$ and $(\cdot)^{-1} : \mathbf{C}_\wedge^S(B) \xrightarrow{\sim} \mathbf{C}_\vee^S(B)$ as follows:

$$\mathbf{S}_\vee(B) \xrightarrow{\neg} \mathbf{S}_\wedge(B) \xrightarrow{\Delta_g} \mathbf{C}_\wedge^S(B) \xrightarrow{(\cdot)^{-1}} \mathbf{C}_\vee^S(B)$$

$$b \longmapsto \neg b \longmapsto (b, \neg b) \longmapsto (\neg b, b)$$

This means that Δ_g is also a semiring homomorphism $\mathbf{S}_\vee(B) \xrightarrow{\Delta_g} \mathbf{C}_\vee^S(B)$. The following diagram commutes in the category of semirings:

$$
\begin{array}{ccc}
\mathbf{S}_\wedge(B) & \xleftarrow[\simeq]{\neg} & \mathbf{S}_\vee(B) \\
{\scriptstyle \Delta_g} \downarrow & \searrow{\scriptstyle \Delta_a} & \downarrow {\scriptstyle \Delta_g} \\
\mathbf{C}_\wedge^S(B) & \xleftarrow[(\cdot)^{-1}]{\simeq} & \mathbf{C}_\vee^S(B)
\end{array}
$$

The commutativity of the diagram gives rise to the diagonal arrow $\Delta_a = (\cdot)^{-1} \circ \Delta_g = \Delta_g \circ \neg$. This map is a semiring homomorphism from $\mathbf{S}_\wedge(B)$ to $\mathbf{C}_\vee^S(B)$ and from $\mathbf{S}_\vee(B)$ to $\mathbf{C}_\wedge^S(B)$. Explicitly, this map is

$$\Delta_a b = (\Delta_g(b))^{-1} = (\neg b, b)^{-1} = (b, \neg b) \quad (b \in B).$$

If we use the map ι_g, we can obtain $\iota_a' : \mathbf{S}_\vee(B) \to \mathbf{C}_\bullet^S(B)$ as follows:

$$\mathbf{S}_\vee(B) \xrightarrow{\;\neg\;} \mathbf{S}_\wedge(B) \xrightarrow{\;\iota_g\;} \mathbf{C}_\parallel^S(B) \xrightarrow{\;(\cdot)^{-1}\;} \mathbf{C}_\vee^S(B)$$

$$b \longmapsto \neg b \longmapsto (1_B, \neg b) \longmapsto (\neg b, 1_B)$$

We obtain the diagram below.

$$
\begin{array}{ccc}
\mathbf{S}_\wedge(B) & \xleftrightarrow[\simeq]{\;\neg\;} & \mathbf{S}_\vee(B) \\
\iota_g \Big\downarrow \;\; \iota_a \;\; \times \;\; \iota_g' \;\; \Big\downarrow \iota_a' \\
\mathbf{C}_\parallel^S(B) & \xleftarrow[(\cdot)^{-1}]{\;\simeq\;} & \mathbf{C}_\bullet^S(B)
\end{array}
$$

The commutativity of the diagram provides the semiring maps $\;\mathbf{S}_\wedge(B) \xrightarrow{\iota_a} \mathbf{C}_\bullet^S(B)\;$ and $\;\mathbf{S}_\vee(B) \xrightarrow{\iota_g'} \mathbf{C}_\parallel^S(B)$

given by $\iota_a = (\cdot)^{-1} \circ \iota_g$ and $\iota_g' = \iota_g \circ \neg$.

Now we consider maps to $\mathbf{S}_\wedge(B)$. The map $\pi_g : (a, g) \mapsto g$ is a semiring homomorphism from $\mathbf{C}_\wedge^S(B)$ to $\mathbf{S}_\wedge(B)$ and from $\mathbf{C}_\parallel^S(B)$ to $\mathbf{S}_\wedge(B)$. Similarly, the map $\pi_a' : (a, g) \mapsto \neg a$ is a semiring homomorphism from $\mathbf{C}_\wedge^S(B)$ to $\mathbf{S}_\wedge(B)$. The following diagrams commute and define the maps not specified before.

$$
\begin{array}{ccc}
\mathbf{S}_\wedge(B) \xleftrightarrow[\simeq]{\neg} \mathbf{S}_\vee(B) & \mathbf{S}_\wedge(B) \xleftrightarrow[\simeq]{\neg} \mathbf{S}_\vee(B) & \mathbf{S}_\wedge(B) \xleftrightarrow[\simeq]{\neg} \mathbf{S}_\vee(B) \\
\pi_g \uparrow \; \pi_g' \times \pi_a \; \uparrow \pi_a' & \pi_g \uparrow \; \pi_g' \times \pi_a \; \uparrow \pi_a' & \pi_a' \uparrow \; \pi_a \times \pi_g' \; \uparrow \pi_g \\
\mathbf{C}_\parallel^S(B) \xleftarrow[(\cdot)^{-1}]{\simeq} \mathbf{C}_\bullet^S(B) & \mathbf{C}_\wedge^S(B) \xleftarrow[(\cdot)^{-1}]{\simeq} \mathbf{C}_\vee^S(B) & \mathbf{C}_\wedge^S(B) \xleftarrow[(\cdot)^{-1}]{\simeq} \mathbf{C}_\vee^S(B)
\end{array}
$$

4   Actions

One of the questions we sought to answer was, how can we compute contract abstractions? Abstractions are useful to carry out refinement verification in less costly computational environments. A key way to define abstractions will be through contract actions.

The semiring maps just described can be used to generate actions of the semirings $\mathbf{S}_\wedge(B)$ and $\mathbf{S}_\vee(B)$ over the contract semirings. Consider, for example the map $\Delta_g : \mathbf{S}_\wedge(B) \to \mathbf{C}_\wedge^S(B)$. For a contract $\mathscr{C} = (a, g)$, we have $\Delta_g(b) \wedge \mathscr{C} = (\neg b, b) \wedge (a, g) = (b \to a, b \wedge g)$.

Now consider the map $\iota_g : \mathbf{S}_\wedge(B) \to \mathbf{C}_\parallel^S(B)$: $\iota_g(b) \parallel \mathscr{C} = (1_B, b) \parallel (a, g) = ((b \wedge g) \to a, b \wedge g) = (b \to a, b \wedge g)$. We observe that $\mathbf{S}_\wedge(B)$ acts in the same way on the semirings $\mathbf{C}_\wedge^S(B)$ and $\mathbf{C}_\parallel^S(B)$. This leads us to

Definition 4.1. *The right action of B on $\mathbf{C}(B)$ is $(a, g) \cdot b = (b \to a, b \wedge g)$.*

Similarly, the semiring map $\Delta_a : \mathbf{S}_\wedge(B) \to \mathbf{C}_\vee^S(B)$ yields $\Delta_a(b) \vee \mathscr{C} = (b, \neg b) \vee (a, g) = (b \wedge a, b \to g)$, and the semiring map $\iota_a : \mathbf{S}_\wedge(B) \to \mathbf{C}_\bullet^S(B)$ results in $\iota_a(b) \bullet \mathscr{C} = (b, 1_B) \bullet (a, g) = (b \wedge a, b \to g)$. $\mathbf{S}_\wedge(B)$ acts in the same way on the semirings $\mathbf{C}_\vee^S(B)$ and $\mathbf{C}_\bullet^S(B)$. We thus obtain

Order	
$b \cdot \mathscr{C} \geq \mathscr{C}$	$\mathscr{C} \cdot b \leq \mathscr{C}$
$\mathscr{C} \leq \mathscr{C}' \Rightarrow b \cdot \mathscr{C} \leq b \cdot \mathscr{C}'$	$\mathscr{C} \leq \mathscr{C}' \Rightarrow \mathscr{C} \cdot b \leq \mathscr{C}' \cdot b$
Reciprocal	
$(b \cdot \mathscr{C})^{-1} = \mathscr{C}^{-1} \cdot b$	
Associativity	
$(b \wedge b') \cdot \mathscr{C} = b \cdot (b' \cdot \mathscr{C})$	$\mathscr{C} \cdot (b \wedge b') = (\mathscr{C} \cdot b) \cdot b'$
Distributivity over the Boolean algebra	
$(b \vee b') \cdot \mathscr{C} = (b \cdot \mathscr{C}) \wedge (b' \cdot \mathscr{C})$	$\mathscr{C} \cdot (b \vee b') = (\mathscr{C} \cdot b) \vee (\mathscr{C} \cdot b')$
Actions and the contract operations	
$b \cdot (\mathscr{C} \wedge \mathscr{C}') = b \cdot \mathscr{C} \wedge b \cdot \mathscr{C}'$	$(\mathscr{C} \wedge \mathscr{C}') \cdot b = \mathscr{C} \cdot b \wedge \mathscr{C}'$
$b \cdot (\mathscr{C} \vee \mathscr{C}') = b \cdot \mathscr{C} \vee \mathscr{C}'$	$(\mathscr{C} \vee \mathscr{C}') \cdot b = \mathscr{C} \cdot b \vee \mathscr{C}' \cdot b$
$b \cdot (\mathscr{C} \parallel \mathscr{C}') = b \cdot \mathscr{C} \parallel b \cdot \mathscr{C}'$	$(\mathscr{C} \parallel \mathscr{C}') \cdot b = \mathscr{C} \cdot b \parallel \mathscr{C}'$
$b \cdot (\mathscr{C} \bullet \mathscr{C}') = b \cdot \mathscr{C} \bullet \mathscr{C}'$	$(\mathscr{C} \bullet \mathscr{C}') \cdot b = (\mathscr{C} \cdot b) \bullet (\mathscr{C}' \cdot b)$
Actions and the adjoint operations	
$b \cdot (\mathscr{C}/\mathscr{C}') = \mathscr{C}/(\mathscr{C}' \cdot b) = (b \cdot \mathscr{C})/\mathscr{C}'$	$(\mathscr{C}/\mathscr{C}') \cdot b = (\mathscr{C} \cdot b)/(b \cdot \mathscr{C}')$
$b \cdot (\mathscr{C} \div \mathscr{C}') = (b \cdot \mathscr{C}) \div (\mathscr{C}' \cdot b)$	$(\mathscr{C} \div \mathscr{C}') \cdot b = (\mathscr{C} \cdot b) \div \mathscr{C}' = \mathscr{C} \div (b \cdot \mathscr{C}')$
$b \cdot (\mathscr{C}' \to \mathscr{C}) = \mathscr{C}' \to b \cdot \mathscr{C} = \mathscr{C}' \cdot b \to \mathscr{C}$	$(\mathscr{C}' \to \mathscr{C}) \cdot b = b \cdot \mathscr{C}' \to \mathscr{C} \cdot b$
$b \cdot (\mathscr{C}' \nrightarrow \mathscr{C}) = \mathscr{C}' \cdot b \nrightarrow b \cdot \mathscr{C}$	$(\mathscr{C}' \nrightarrow \mathscr{C}) \cdot b = \mathscr{C}' \nrightarrow \mathscr{C} \cdot b = b \cdot \mathscr{C}' \nrightarrow \mathscr{C}$

Table 4: Identities for the left and right actions of a Boolean algebra B over its contract algebra $(b, b' \in B$ and $\mathscr{C}, \mathscr{C}' \in \mathbf{C}(B))$

Definition 4.2. *The left action of B on $\mathbf{C}(B)$ is given by $b \cdot (a, g) = (b \wedge a, b \to g)$.*

The left and right actions have the practical meaning of adding assumptions or guarantees, respectively, to a contract. The following proposition shows several properties of these actions.

Proposition 4.3. *The identities for the left and right actions shown in Table 4 hold.*

The left action on contracts can be used to generate an abstraction for the guarantees of a contract. Given a contract $\mathscr{C}$ and $b \in B$, we know from Table 4 that $\mathscr{C} \leq b \cdot \mathscr{C}$. This tells us that we can obtain a more relaxed contract by adding assumptions. Subsequently, we may make use of this additional assumption to coarsen the guarantees of the contract. This operation is rich in algebraic properties, as shown in Table 4. For algorithmic manipulations of contracts, see Chapter 7 of [10].

5 Contract abstractions

It is often useful to vary the level of detail used to represent objects. More detail may be needed to carry out some analysis tasks, but too much detail may hinder computation. We will explore Question 2: how can we compute contract abstractions?.

5.1 The Galois-connection abstraction

Suppose we have a monotone operator $\alpha \colon B_c \to B_a$, where B_a and B_c are Boolean algebras called abstract and concrete domains, respectively. Suppose that α is also a monoid map $\alpha \colon \mathbf{M}_\vee(B_c) \to \mathbf{M}_\vee(B_a)$, i.e., it commutes with disjunction and maps 1_{B_c} to 1_{B_a}. We can define a map $\bar{\alpha} \colon \mathbf{C}(B_c) \to \mathbf{C}(B_a)$ as

$$\bar{\alpha} \colon (a, g) \mapsto (\alpha(a), \alpha(g)). \tag{3}$$

The map is well defined since $\alpha(a) \vee \alpha(g) = \alpha(a \vee g) = \alpha(1_{B_c}) = 1_{B_a}$. Moreover, the monotonicity of $\bar{\alpha}$ follows from the monotonicity of α. This abstraction is a slight generalization of that proposed in Chapter 5 of [4].

How can such a map α be obtained? Suppose that α is a monotone map that maps 1_{B_c} to 1_{B_a}. If α is a left element of a Galois connection pair, then it commutes with disjunction (because left adjoints commute with colimits). Thus, such an α generates a valid contract abstraction (3).

5.2   Contract Galois connections from Boolean algebra maps

Let $f : B \to B'$ be a Boolean algebra map. f induces a map $f^* : \mathbf{C}(B) \to \mathbf{C}(B')$ between their contract algebras given by $f^*(a,g) = (f(a), f(g))$. Observe that $f(a) \vee f(g) = f(a \vee g) = f(1_B) = 1_{B'}$. As f commutes with the Boolean algebra operations, f^* commutes with the contract operations. Thus, f^* is well-defined.

Let $\gamma : B_a \to B_c$ and $\alpha : B_c \to B_a$ be Boolean algebra maps. These maps generate contract maps

$$\mathbf{C}(B_a) \xrightarrow{\gamma^*} \mathbf{C}(B_c) \quad \text{and} \quad \mathbf{C}(B_c) \xrightarrow{\alpha^*} \mathbf{C}(B_a), \quad \text{as described before.}$$

We are interested in exploring the conditions needed for these maps to form a Galois connection. Specifically, for $\mathscr{C}_a = (a_a, g_a) \in \mathbf{C}(B_a)$ and $\mathscr{C}_c = (a_c, g_c) \in \mathbf{C}(B_c)$, we want $\alpha^*(\mathscr{C}_c) \leq \mathscr{C}_a$ if and only if $\mathscr{C}_c \leq \gamma^*(\mathscr{C}_a)$.

This means that $(\alpha a_c, \alpha g_c) \leq (a_a, g_a)$ if and only if $(a_c, g_c) \leq (\gamma a_a, \gamma g_a)$. If we set $a_c = 1_{B_c}$ and $a_a = 1_{B_a}$, we get $\alpha g_c \leq g_a$ if and only if $g_c \leq \gamma g_a$, and if we set $g_c = 1_{B_c}$ and $g_a = 1_{B_a}$, we obtain $a_a \leq \alpha a_c$ if and only if $\gamma a_a \leq a_c$.

This means that α and γ must be simultaneously the left and right adjoints of each other. By setting $g_c = \gamma g_a$ and $a_c = \gamma a_a$ in the equations above, we obtain that $\alpha \circ \gamma$ is the identity map. Similarly, by setting $g_a = \alpha g_c$ and $a_a = \alpha a_c$, we get that $\gamma \circ \alpha$ is the identity map. This means that B_a and B_c are isomorphic. We conclude that contract Galois connections generated from Boolean algebra maps impose very rigid constraints on the Boolean algebras over which the contracts are defined.

6   Conclusions

Assume-guarantee contracts are an appealing tool in engineering design. This paper describes the algebra of contracts that can provide a framework for manipulating these objects in a structured way.

We explored in-depth this algebraic structure *per se*, while establishing a mathematically sound methodology for contract-based system design. The rich algebraic structure of assume-guarantee contracts provides sound support for a number of operations in system design: combining viewpoints, composing sub-specifications, and patching a design. At the same time, it raises a number of methodological issues: does a designer have freedom in ordering the different design steps? Alternatively, does the theory impose or recommend an ordering? Abstractions are a well established way to provide semi-decision procedures at a lower computational cost. We answered the following question: is there a systematic way to lift abstractions of properties to abstractions of contracts?

One question remains open: is there a sound way to lift testing from properties—where this is well developed—to assume-guarantee contracts? Our algebra of assume-guarantee contracts does not provide answers to this question as yet.

References

[1] Martín Abadi & Leslie Lamport (1993): *Composing Specifications*. ACM Transactions on Programming Languages and Systems 15(1), pp. 73–132, doi:10.1145/151646.151649. Available at `http://doi.acm.org/10.1145/151646.151649`.

[2] Luca de Alfaro & Thomas A. Henzinger (2001): *Interface Automata*. In: *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-9, Association for Computing Machinery, New York, NY, USA, pp. 109–120, doi:10.1145/503209.503226. Available at `https://doi.org/10.1145/503209.503226`.

[3] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone & Christos Sofronis (2008): *Multiple Viewpoint Contract-Based Specification and Design*. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf & Willem-Paul de Roever, editors: *Formal Methods for Components and Objects, 6th International Symposium (FMCO 2007), Amsterdam, The Netherlands, October 24–26, 2007, Revised Papers*, Lecture Notes in Computer Science 5382, Springer Verlag, Berlin Heidelberg, pp. 200–225, doi:10.1007/978-3-540-92188-2.

[4] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto L. Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger & Kim G. Larsen (2018): *Contracts for System Design*. Foundations and Trends® in Electronic Design Automation 12(2-3), pp. 124–400.

[5] W. Damm (2005): *Controlling speculative design processes using rich component models*. In: *Fifth International Conference on Application of Concurrency to System Design (ACSD'05)*, pp. 118–119, doi:10.1109/ACSD.2005.35.

[6] R. W. Floyd (1967): *Assigning meanings to programs*. Mathematical Aspects of Computer Science, ed. Schwartz, JT, Amer. Math. Soc.

[7] Jonathan S. Golan (1999): *Semirings and their Applications*, 1st ed edition. Springer, Dordrecht, doi:10.1007/978-94-015-9333-5.

[8] C. A. R. Hoare (1969): *An Axiomatic Basis for Computer Programming*. Commun. ACM 12(10), pp. 576–580, doi:10.1145/363235.363259. Available at `http://doi.acm.org/10.1145/363235.363259`.

[9] I. Incer, A. L. Sangiovanni-Vincentelli, C.-W. Lin & E. Kang (2018): *Quotient for Assume-Guarantee Contracts*. In: *16th ACM-IEEE International Conference on Formal Methods and Models for System Design*, MEMOCODE'18, pp. 67–77, doi:10.1109/MEMCOD.2018.8556872.

[10] Inigo Incer (2022): *The Algebra of Contracts*. Ph.D. thesis, EECS Department, University of California, Berkeley.

[11] Inigo Incer, Leonardo Mangeruca, Tiziano Villa & Alberto L. Sangiovanni-Vincentelli (2020): *The Quotient in Preorder Theories*. In Jean-Francois Raskin & Davide Bresolin, editors: *Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, Brussels, Belgium, September 21-22, 2020*, Electronic Proceedings in Theoretical Computer Science 326, Open Publishing Association, Brussels, Belgium, pp. 216–233, doi:10.4204/EPTCS.326.14.

[12] Leslie Lamport (1989): *A Simple Approach to Specifying Concurrent Systems*. Commun. ACM 32(1), pp. 32–45, doi:10.1145/63238.63240. Available at `http://doi.acm.org/10.1145/63238.63240`.

[13] Piergiuseppe Mallozzi, Pierluigi Nuzzo & Patrizio Pelliccione (2021): *Incremental Refinement of Goal Models with Contracts*. In Hossein Hojjat & Mieke Massink, editors: *Fundamentals of Software Engineering*, Springer International Publishing, Cham, pp. 35–50.

[14] Bertrand Meyer (1992): *Applying "Design by Contract"*. IEEE Computer 25(10), pp. 40–51, doi:10.1109/2.161279.

[15] Radu Negulescu (1995): *Process spacess*. Technical Report CS-95-48, University of Waterloo.

[16] P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti & T. Villa (2015): *A Platform-Based Design Methodology With Contracts and Related Tools for the Design of Cyber-Physical Systems*. Proceedings of the IEEE 103(11), pp. 2104–2132, doi:10.1109/JPROC.2015.2453253.

[17] Roberto Passerone, Inigo Incer & Alberto L. Sangiovanni-Vincentelli (2019): *Coherent Extension, Composition, and Merging Operators in Contract Models for System Design*. ACM Trans. Embed. Comput. Syst. 18(5s), doi:10.1145/3358216.

[18] Alberto L. Sangiovanni-Vincentelli, Werner Damm & Roberto Passerone (2012): *Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems*. European Journal of Control 18(3), pp. 217 – 238, doi:http://dx.doi.org/10.3166/ejc.18.217-238. Available at `http://www.sciencedirect.com/science/article/pii/S0947358012709433`.

A   Proofs

Proof of Proposition 2.14. Let $\mathscr{C}_i$ be the contract stated in the proposition. Observe that $\mathscr{C}_i \wedge \mathscr{C}' = \mathscr{C} \wedge \mathscr{C}'$. Thus, by the monotonicity of conjunction, $\mathscr{C}'' \leq \mathscr{C}_i$ implies that $\mathscr{C}'' \wedge \mathscr{C}' \leq \mathscr{C}$.

Now write $\mathscr{C}''$ as $\mathscr{C}'' = (A'', G'')$ and assume that $\mathscr{C}'' \wedge \mathscr{C}' \leq \mathscr{C}$. Then

$$G'' \cap G' \leq G \text{ and}$$
$$A'' \cup A' \geq A.$$

From this we conclude that

$$G'' \leq G \cup \neg G' \text{ and} \tag{4}$$
$$A'' \geq A \cap \neg A'. \tag{5}$$

From (4) and the fact that $A'' \geq \neg G''$ (which follows from the definition of AG contracts), we obtain $A'' \geq G' \cap \neg G$. This result and (5) yield

$$A'' \geq (A \cap \neg A') \cup (G' \cap \neg G).$$

This expression and (4) mean that $\mathscr{C}'' \leq \mathscr{C}_i$, completing the proof. $\qquad\square$

Proof of Proposition 3.1. We already know that 1 and 0 are, respectively, the identity elements of conjunction and disjunction. e is the identity for composition and merging. The idempotence of these operations follows immediately from their definitions. It remains to show that these operations are associative.

Let $\mathscr{C} = (a, g)$, $\mathscr{C}' = (a', g')$, and $\mathscr{C}'' = (a'', g'')$ be contracts.

- Conjunction.

$$\begin{aligned}
\mathscr{C} \wedge (\mathscr{C}' \wedge \mathscr{C}'') &= (a, g) \wedge (a' \vee a'', g' \wedge g'') \\
&= \left(a \vee (a' \vee a''), g \wedge (g' \wedge g'')\right) \\
&= \left((a \vee a') \vee a'', (g \wedge g') \wedge g''\right) \\
&= \left(a \vee a', g \wedge g'\right) \wedge \mathscr{C}'' = (\mathscr{C} \wedge \mathscr{C}') \wedge \mathscr{C}''
\end{aligned}$$

- Disjunction.

$$\begin{aligned}
\mathscr{C} \vee (\mathscr{C}' \vee \mathscr{C}'') &= \left(\mathscr{C}^{-1} \wedge ((\mathscr{C}')^{-1} \wedge (\mathscr{C}'')^{-1})\right)^{-1} \\
&= \left((\mathscr{C}^{-1} \wedge (\mathscr{C}')^{-1}) \wedge (\mathscr{C}'')^{-1}\right)^{-1} = (\mathscr{C} \vee \mathscr{C}') \vee \mathscr{C}''
\end{aligned}$$

- Composition.

$$\mathscr{C} \parallel (\mathscr{C}' \parallel \mathscr{C}'')$$
$$= (a,g) \parallel (\neg g' \vee \neg g'' \vee (a' \wedge a''), g' \wedge g'')$$
$$= (\neg g \vee \neg g' \vee \neg g'' \vee (a \wedge a' \wedge a''), g \wedge (g' \wedge g''))$$
$$= (\neg g \vee \neg g' \vee \neg g'' \vee ((a \wedge a') \wedge a''), (g \wedge g') \wedge g'')$$
$$= (\neg g \vee \neg g' \vee (a \wedge a'), g \wedge g') \parallel \mathscr{C}'' = (\mathscr{C} \parallel \mathscr{C}') \parallel \mathscr{C}''$$

- Merging.

$$\mathscr{C} \bullet (\mathscr{C}' \bullet \mathscr{C}'') = \left(\mathscr{C}^{-1} \parallel ((\mathscr{C}')^{-1} \parallel (\mathscr{C}'')^{-1}) \right)^{-1}$$
$$= \left((\mathscr{C}^{-1} \parallel (\mathscr{C}')^{-1}) \parallel (\mathscr{C}'')^{-1} \right)^{-1} = (\mathscr{C} \bullet \mathscr{C}') \bullet \mathscr{C}'' \qquad \Box$$

Proof of Proposition 3.2. Due to the duality relations between conjunction and disjunction and between composition and merging, the reciprocal map provides monoid isomorphisms between $(\mathbf{C}(B), \wedge, 1)$ and $(\mathbf{C}(B), \vee, 0)$ and between $(\mathbf{C}(B), \parallel, e)$ and $(\mathbf{C}(B), \bullet, e)$.

We now show that the map $\theta_g : \mathbf{C}_\parallel^M(B) \to \mathbf{C}_\wedge^M(B)$ defined as

$$\theta_g : (a,g) \mapsto (\neg(a \wedge g), g)$$

is a monoid isomorphism.

Observe that $\theta_g^2(a,g) = \theta_g(\neg(a \wedge g), g) = (\neg(\neg(a \wedge g) \wedge g), g) = (a,g)$, so θ_g is an involution. We proceed to show it is a monoid homomorphism. Let $\mathscr{C} = (a,g)$ and $\mathscr{C}' = (a',g')$.

- $\theta_g(e) = \theta_g(1,1) = (0,1) = 1$

- We verify whether θ_g commutes with the multiplications:

$$\theta_g(\mathscr{C} \parallel \mathscr{C}') = \theta_g(\neg(g \wedge g') \vee (a \wedge a'), g \wedge g')$$
$$= (\neg(a \wedge g \wedge a' \wedge g'), g \wedge g')$$
$$= (\neg(a \wedge g) \vee \neg(a' \wedge g'), g \wedge g')$$
$$= (\neg(a \wedge g), g) \wedge (\neg(a' \wedge g'), g') = \theta_g(\mathscr{C}) \wedge \theta_g(\mathscr{C}').$$

As θ_g is an involution, we have to check that it is a monoid map from $\mathbf{C}_\wedge^M(B)$ to $\mathbf{C}_\parallel^M(B)$.

$$\theta_g(\mathscr{C} \wedge \mathscr{C}')$$
$$= \theta_g(a \vee a', g \wedge g') = (\neg(g \wedge g' \wedge (a \vee a')), g \wedge g')$$
$$= (\neg(g \wedge g') \vee (\neg a \wedge \neg a'), g \wedge g')$$
$$= (\neg(g \wedge g') \vee (\neg(a \wedge g) \wedge \neg(a' \wedge g')), g \wedge g')$$
$$= (\neg(a \wedge g), g) \parallel (\neg(a' \wedge g'), g') = \theta_g(\mathscr{C}) \parallel \theta_g(\mathscr{C}')$$

The isomorphism θ_a is defined using the diagram (2), i.e., $\theta_a(a,g) = \left(\theta_g(a,g)^{-1} \right)^{-1} = (\theta_g(g,a))^{-1} = (\neg(a \wedge g), a)^{-1} = (a, \neg(a \wedge g))$. $\qquad \Box$

Proof of Theorem 3.3. Because ι is monic, f generates a unique monoid map $f^{\#}$:

$$
\begin{array}{ccc}
\mathbf{M}_{\wedge}(B) \times \mathbf{M}_{\wedge}(B) & & \mathbf{M}_{\wedge}(B') \times \mathbf{M}_{\wedge}(B') \\
\downarrow{\scriptstyle \pi} & \nearrow{\scriptstyle f^{\#}} & \uparrow{\scriptstyle \iota} \\
\mathbf{C}_{\|}^{M}(B) & \xrightarrow[\;f\;]{} & \mathbf{C}_{\|}^{M}(B')
\end{array}
$$

Because π is epic, $f^{\#}$ generates a unique monoid map $f^{\flat}$

$$
\begin{array}{ccc}
\mathbf{M}_{\wedge}(B) \times \mathbf{M}_{\wedge}(B) & \dashrightarrow^{f^{\flat}} & \mathbf{M}_{\wedge}(B') \times \mathbf{M}_{\wedge}(B') \\
\downarrow{\scriptstyle \pi} & \nearrow{\scriptstyle f^{\#}} & \uparrow{\scriptstyle \iota} \\
\mathbf{C}_{\|}^{M}(B) & \xrightarrow[\;f\;]{} & \mathbf{C}_{\|}^{M}(B')
\end{array}
$$

Thus, we have the diagram

$$
\begin{array}{ccc}
\mathbf{M}_{\wedge}(B) \times \mathbf{M}_{\wedge}(B) & \dashrightarrow^{f^{\flat}} & \mathbf{M}_{\wedge}(B') \times \mathbf{M}_{\wedge}(B') \\
{\scriptstyle \iota}\uparrow\downarrow{\scriptstyle \pi} & \nearrow{\scriptstyle f^{\#}} & {\scriptstyle \pi}\downarrow\uparrow{\scriptstyle \iota} \\
\mathbf{C}_{\|}^{M}(B) & \xrightarrow[\;f\;]{} & \mathbf{C}_{\|}^{M}(B')
\end{array}
\qquad (6)
$$

$f^{\flat}$ can be factored as the product of two maps

$$\mathbf{M}_{\wedge}(B) \times \mathbf{M}_{\wedge}(B) \to \mathbf{M}_{\wedge}(B').$$

We also observe that $f^{\flat}(a,g) = f^{\flat}((a,1) \wedge (1,g)) = f^{\flat}(a,1) \wedge f^{\flat}(1,g)$. This means there are monoid maps $l_a, l_g, r_a, r_g \colon \mathbf{M}_{\wedge}(B) \to \mathbf{M}_{\wedge}(B')$ such that

$$f^{\flat}(a,g) = (l_a(a)l_g(g), r_a(a)r_g(g)).$$

To obtain further restrictions on these maps, we use (6): $f^{\flat}(a,g) = \iota \circ f \circ \pi(a,g) = \iota \circ \pi \circ f^{\flat} \circ \iota \circ \pi(a,g) = (l_a(ag)l_g(g)r_a(ag)r_g(g), r_a(ag)r_g(g))$. $\qquad\square$

Proof of Proposition 3.4. Let $\mathscr{C} = (a,g)$, $\mathscr{C}' = (a',g')$, and $\mathscr{C}'' = (a'',g'')$ be contracts.
• Conjunction.

$$
\begin{aligned}
\mathscr{C} \wedge (\mathscr{C}' \vee \mathscr{C}'') &= (a,g) \wedge (a' \wedge a'', g' \vee g'') \\
&= \left((a \vee a') \wedge (a \vee a''), (g \wedge g') \vee (g \wedge g'') \right) \\
&= (\mathscr{C} \wedge \mathscr{C}') \vee (\mathscr{C} \wedge \mathscr{C}'') \\
\mathscr{C} \wedge (\mathscr{C}' \| \mathscr{C}'') & \\
&= (a,g) \wedge \left((g' \wedge g'') \to (a' \wedge a''), g' \wedge g'' \right) \\
&= \left(a \vee (a' \wedge a'') \vee \neg g' \vee \neg g'', g \wedge g' \wedge g'' \right) \\
&= \left(a \vee \neg g \vee (a' \wedge a'') \vee \neg g' \vee \neg g'', g \wedge g' \wedge g'' \right) \\
&= \left((g \wedge g' \wedge g'') \to ((a \vee a') \wedge (a \vee a'')), g \wedge g' \wedge g'' \right) \\
&= (a \vee a', g \wedge g') \| (a \vee a'', g \wedge g'') \\
&= (\mathscr{C} \wedge \mathscr{C}') \| (\mathscr{C} \wedge \mathscr{C}'') \\
e \wedge (1 \bullet 0) &= e \wedge 1 = e \neq 0 \\
&= e \bullet 0 = (e \wedge 1) \bullet (e \wedge 0)
\end{aligned}
$$

- Disjunction.

$$\mathscr{C} \vee (\mathscr{C}' \wedge \mathscr{C}'') = \left(\mathscr{C}^{-1} \wedge ((\mathscr{C}')^{-1} \vee (\mathscr{C}'')^{-1})\right)^{-1}$$
$$= \left((\mathscr{C}^{-1} \wedge (\mathscr{C}')^{-1}) \vee (\mathscr{C}^{-1} \wedge (\mathscr{C}'')^{-1})\right)^{-1}$$
$$= (\mathscr{C} \vee \mathscr{C}') \wedge (\mathscr{C} \vee \mathscr{C}'')$$
$$\mathscr{C} \vee (\mathscr{C}' \bullet \mathscr{C}'') = \left(\mathscr{C}^{-1} \wedge ((\mathscr{C}')^{-1} \parallel (\mathscr{C}'')^{-1})\right)^{-1}$$
$$= \left((\mathscr{C}^{-1} \wedge (\mathscr{C}')^{-1}) \parallel (\mathscr{C}^{-1} \wedge (\mathscr{C}'')^{-1})\right)^{-1}$$
$$= (\mathscr{C} \vee \mathscr{C}') \bullet (\mathscr{C} \vee \mathscr{C}'')$$
$$e \vee (1 \parallel 0) = e \vee 0 = e \neq 1$$
$$= 1 \parallel e = (e \vee 1) \parallel (e \vee 0)$$

- Composition.

$$\mathscr{C} \parallel (\mathscr{C}' \wedge \mathscr{C}'') = (a, g) \parallel (a' \vee a'', g' \wedge g'')$$
$$= \begin{pmatrix} \neg(g \wedge g') \vee \neg(g \wedge g'') \vee (a \wedge a') \vee (a \wedge a''), \\ (g \wedge g') \wedge (g \wedge g'') \end{pmatrix}$$
$$= \left((g \wedge g') \to (a \wedge a'), (g \wedge g')\right) \wedge$$
$$\quad \left((g \wedge g'') \to (a \wedge a''), (g \wedge g'')\right)$$
$$= (\mathscr{C} \parallel \mathscr{C}') \wedge (\mathscr{C} \parallel \mathscr{C}'')$$
$$\mathscr{C} \parallel (\mathscr{C}' \vee \mathscr{C}'') = (a, g) \parallel (a' \wedge a'', g' \vee g'')$$
$$= \begin{pmatrix} \neg(g \wedge g') \wedge \neg(g \wedge g'') \vee ((a \wedge a') \wedge (a \wedge a'')), \\ (g \wedge g') \vee (g \wedge g'') \end{pmatrix}$$
$$= \begin{pmatrix} \left(\neg(g \wedge g') \vee (a \wedge a')\right) \wedge \left(\neg(g \wedge g'') \vee (a \wedge a'')\right), \\ (g \wedge g') \vee (g \wedge g'') \end{pmatrix}$$
$$= \left((g \wedge g') \to (a \wedge a'), (g \wedge g')\right)$$
$$\quad \vee \left((g \wedge g'') \to (a \wedge a''), (g \wedge g'')\right)$$
$$= (\mathscr{C} \parallel \mathscr{C}') \vee (\mathscr{C} \parallel \mathscr{C}'')$$
$$1 \parallel (0 \bullet e) = 1 \parallel 0 = 0 \neq 1 = 0 \bullet 1$$
$$= (1 \parallel 0) \bullet (1 \parallel e)$$

The distributivity of composition over conjunction was shown in [13].

- Merging.

$$\mathscr{C} \bullet (\mathscr{C}' \wedge \mathscr{C}'') = \left(\mathscr{C}^{-1} \parallel ((\mathscr{C}')^{-1} \vee (\mathscr{C}'')^{-1})\right)^{-1}$$
$$= \left((\mathscr{C}^{-1} \parallel (\mathscr{C}')^{-1}) \vee (\mathscr{C}^{-1} \parallel (\mathscr{C}'')^{-1})\right)^{-1}$$
$$= (\mathscr{C} \bullet \mathscr{C}') \wedge (\mathscr{C} \bullet \mathscr{C}'')$$
$$\mathscr{C} \bullet (\mathscr{C}' \vee \mathscr{C}'') = \left(\mathscr{C}^{-1} \parallel ((\mathscr{C}')^{-1} \wedge (\mathscr{C}'')^{-1})\right)^{-1}$$
$$= \left((\mathscr{C}^{-1} \parallel (\mathscr{C}')^{-1}) \wedge (\mathscr{C}^{-1} \parallel (\mathscr{C}'')^{-1})\right)^{-1}$$

$$= (\mathscr{C} \bullet \mathscr{C}') \vee (\mathscr{C} \bullet \mathscr{C}'')$$
$$0 \bullet (1 \parallel e) = 0 \bullet 1 = 1 \neq 0 = 1 \parallel 0$$
$$= (0 \bullet 1) \parallel (0 \bullet e) \qquad \qquad \square$$

Proof of Proposition 3.5. Due to the distributivity of $\star$ over conjunction and disjunction (Proposition 3.4), we have $(C_1 \wedge C_2) \star (C_3 \wedge C_4) = (C_1 \star C_3) \wedge (C_1 \star C_4) \wedge (C_2 \star C_3) \wedge (C_2 \star C_4) \leq (C_1 \star C_3) \wedge (C_2 \star C_4)$ and $(C_1 \vee C_2) \star (C_3 \vee C_4) = (C_1 \star C_3) \vee (C_1 \star C_4) \vee (C_2 \star C_3) \vee (C_2 \star C_4) \geq (C_1 \star C_3) \vee (C_2 \star C_4)$. $\qquad \square$

Proof of Proposition 3.7. Tables 2 and 3 tell, respectively, how operations behave with respect to the distinguished elements and how operations distribute.

Suppose conjunction is the multiplication operation. Since $\mathscr{C} \wedge e \neq e$, neither merging nor composition can be the addition operations. On the other hand, $\mathscr{C} \wedge 0 = 0$, and conjunction distributes over disjunction. Thus, $(\mathbf{C}(B), \wedge, \vee, 1, 0)$ is a semiring.

Now we assume disjunction is the multiplication operation. Since $\mathscr{C} \vee e \neq e$, neither merging nor composition can be the addition operations. However, $\mathscr{C} \vee 1 = 1$, and disjunction distributes over conjunction. Thus, $(\mathbf{C}(B), \vee, \wedge, 0, 1)$ is a semiring.

Suppose composition is the multiplication operation. Since composition does not distribute over merging, merging cannot be addition. Since $\mathscr{C} \parallel 1 \neq 1$, conjunction cannot be addition. However, $\mathscr{C} \parallel 0 = 0$ and composition distributes over disjunction. Thus, $(\mathbf{C}(B), \parallel, \vee, e, 0)$ is a semiring.

Now suppose that merging is the multiplication. Since merging does not distribute over composition, composition cannot be addition. Also, since $\mathscr{C} \bullet 0 \neq 0$, conjunction cannot be addition. However, $\mathscr{C} \bullet 1 = 1$ and merging distributes over conjunction. Thus, $(\mathbf{C}(B), \bullet, \wedge, e, 1)$ is a semiring. $\qquad \square$

Proof of Proposition 3.8. The two semiring isomorphisms are given by the reciprocal map. Suppose there is a semiring map $\beta \colon \mathbf{C}_{\parallel}^S(B) \to \mathbf{C}_{\wedge}^S(B)$. Then

$$\beta(a, 1_B) = \beta((a, 1_B) \vee e) = \beta(a, g) \vee 1 = 1,$$

which means that β is not invertible. $\qquad \square$

Proof of Proposition 3.9. Let $b, b' \in B$.

- $\Delta_g(0_B) = (1_B, 0_B) = 0$ and $\Delta_g(1_B) = (0_B, 1_B) = 1$
- $\Delta_g(b \wedge b') = (\neg(b \wedge b'), b \wedge b') = (\neg b \vee \neg b', b \wedge b') = \Delta_g(b) \wedge \Delta_g(b')$
- $\Delta_g(b \vee b') = (\neg(b \vee b'), b \vee b') = (\neg b \wedge \neg b', b \vee b') = \Delta_g(b) \vee \Delta_g(b')$

This shows that Δ_g is a semiring homomorphism. Now we study ι_g:

- $\iota_g(0_B) = (1_B, 0_B) = 0$ and $\iota_g(1_B) = (1_B, 1_B) = e$
- $\iota_g(b \wedge b') = (1_B, b \wedge b') = (1_B, b) \parallel (1_B, b') = \iota_g(b) \parallel \iota_g(b')$
- $\iota_g(b \vee b') = (1, b \vee b') = (1_B, b) \vee (1_B, b') = \iota_g(b) \vee \iota_g(b')$

We conclude that ι_g is a semiring homomorphism as well. $\qquad \square$

Proof of Proposition 4.3. Let $b, b' \in B$, $\mathscr{C} = (a, g)$, and $\mathscr{C}' = (a', g')$. We have the following properties:

- Order.

$$b \cdot \mathscr{C} = b \cdot (a,g) = (b \wedge a, b \to g) \geq (a,g) = \mathscr{C}$$
$$\mathscr{C} \cdot b = (a,g) \cdot b = (b \to a, b \wedge g) \leq (a,g) = \mathscr{C}$$

Now suppose $\mathscr{C} = (a,g) \leq \mathscr{C}' = (a',g')$. We have $a' \leq a$ and $g \leq g'$. Since the operations $b \wedge (\cdot)$ and $b \to (\cdot)$ are monotonic, we have $b \cdot \mathscr{C} \leq b \cdot \mathscr{C}'$ and $\mathscr{C} \cdot b \leq \mathscr{C}' \cdot b$.

- Reciprocal.

$$(b \cdot \mathscr{C})^{-1} = (b \wedge a, b \to g)^{-1} = (b \to g, b \wedge a)$$
$$= (g,a) \cdot b = \mathscr{C}^{-1} \cdot b$$

- Associativity.

$$(b \wedge b') \cdot \mathscr{C} = \big((b \wedge b') \wedge a, (b \wedge b') \to g\big)$$
$$= \big(b \wedge (b' \wedge a), b \to (b' \to g)\big)$$
$$= b \cdot \big(b' \cdot (a,g)\big) = b \cdot \big(b' \cdot \mathscr{C}\big)$$
$$\mathscr{C} \cdot (b \wedge b') = \big((b \wedge b') \cdot \mathscr{C}^{-1}\big)^{-1} = \big((b' \wedge b) \cdot \mathscr{C}^{-1}\big)^{-1}$$
$$= \big(b' \cdot (b \cdot \mathscr{C}^{-1})\big)^{-1} = \big(b \cdot \mathscr{C}^{-1}\big)^{-1} \cdot b' = (\mathscr{C} \cdot b) \cdot b'$$

- Distributivity over the Boolean algebra B.

$$(b \vee b') \cdot \mathscr{C} = \big((b \vee b') \wedge a, (b \vee b') \to g\big)$$
$$= \big((b \wedge a) \vee (b' \wedge a), (b \to g) \wedge (b' \to g)\big)$$
$$= (b \cdot \mathscr{C}) \wedge (b' \cdot \mathscr{C})$$
$$\mathscr{C} \cdot (b \vee b') = \big((b \vee b') \cdot \mathscr{C}^{-1}\big)^{-1}$$
$$= \big(b \cdot \mathscr{C}^{-1} \wedge b' \cdot \mathscr{C}^{-1}\big)^{-1}$$
$$= \mathscr{C} \cdot b \vee \mathscr{C} \cdot b'$$

- Distributivity over the contract operations.
    - Conjunction.

$$b \cdot (\mathscr{C} \wedge \mathscr{C}') = (b \wedge (a \vee a'), b \to (g \wedge g'))$$
$$= ((b \wedge a) \vee (b \wedge a'), (b \to g) \wedge (b \to g'))$$
$$= b \cdot \mathscr{C} \wedge b \cdot \mathscr{C}'$$
$$(\mathscr{C} \wedge \mathscr{C}') \cdot b = (b \to (a \vee a'), b \wedge (g \wedge g'))$$
$$= ((b \to a) \vee a', (b \wedge g) \wedge g')$$
$$= (\mathscr{C} \cdot b) \wedge \mathscr{C}'$$

- Disjunction.

$$b \cdot (\mathscr{C} \vee \mathscr{C}') = \left(\left(\mathscr{C}^{-1} \wedge (\mathscr{C}')^{-1} \right) \cdot b \right)^{-1}$$
$$= \left(\mathscr{C}^{-1} \cdot b \wedge (\mathscr{C}')^{-1} \right)^{-1}$$
$$= b \cdot \mathscr{C} \vee \mathscr{C}'$$
$$(\mathscr{C} \vee \mathscr{C}') \cdot b = \left(b \cdot \left(\mathscr{C}^{-1} \wedge (\mathscr{C}')^{-1} \right) \right)^{-1}$$
$$= \left(b \cdot \mathscr{C}^{-1} \wedge b \cdot (\mathscr{C}')^{-1} \right)^{-1}$$
$$= \mathscr{C} \cdot b \vee \mathscr{C}' \cdot b$$

- Composition.

$$b \cdot (\mathscr{C} \parallel \mathscr{C}')$$
$$= \left(b \wedge \left((g \wedge g') \to (a \wedge a') \right), b \to (g \wedge g') \right)$$
$$= \left(\begin{matrix} (b \to (g \wedge g')) \to (b \wedge a \wedge a'), \\ b \to (g \wedge g') \end{matrix} \right)$$
$$= \left(\begin{matrix} ((b \to g) \wedge (b \to g')) \to ((b \wedge a) \wedge (b \wedge a')), \\ (b \to g) \wedge (b \to g') \end{matrix} \right)$$
$$= b \cdot \mathscr{C} \parallel b \cdot \mathscr{C}'$$
$$(\mathscr{C} \parallel \mathscr{C}') \cdot b$$
$$= \left(b \to \left((g \wedge g') \to (a \wedge a') \right), b \wedge g \wedge g' \right)$$
$$= \left((b \wedge g \wedge g') \to (a \wedge a'), b \wedge g \wedge g' \right)$$
$$= \left((b \wedge g \wedge g') \to ((b \to a) \wedge a'), b \wedge g \wedge g' \right)$$
$$= (\mathscr{C} \cdot b) \parallel \mathscr{C}'$$

- Merging.

$$b \cdot (\mathscr{C} \bullet \mathscr{C}') = \left(\left(\mathscr{C}^{-1} \parallel (\mathscr{C}')^{-1} \right) \cdot b \right)^{-1}$$
$$= \left(\mathscr{C}^{-1} \cdot b \parallel (\mathscr{C}')^{-1} \right)^{-1} = b \cdot \mathscr{C} \bullet \mathscr{C}'$$
$$(\mathscr{C} \bullet \mathscr{C}') \cdot b = \left(b \cdot \left(\mathscr{C}^{-1} \parallel (\mathscr{C}')^{-1} \right) \right)^{-1}$$
$$= \left(b \cdot \mathscr{C}^{-1} \parallel b \cdot (\mathscr{C}')^{-1} \right)^{-1} = \mathscr{C} \cdot b \bullet \mathscr{C}' \cdot b$$

- Distributivity over the adjoint operations.
    - Quotient.

$$b \cdot (\mathscr{C} / \mathscr{C}') = b \cdot (\mathscr{C} \bullet (\mathscr{C}')^{-1}) = b \cdot \mathscr{C} \bullet (\mathscr{C}')^{-1}$$
$$= (b \cdot \mathscr{C}) / (\mathscr{C}')$$
$$b \cdot (\mathscr{C} / \mathscr{C}') = b \cdot (\mathscr{C} \bullet (\mathscr{C}')^{-1}) = \mathscr{C} \bullet b \cdot (\mathscr{C}')^{-1}$$
$$= \mathscr{C} \bullet (\mathscr{C}' \cdot b)^{-1} = \mathscr{C} / (\mathscr{C}' \cdot b)$$
$$(\mathscr{C} / \mathscr{C}') \cdot b = (\mathscr{C} \bullet (\mathscr{C}')^{-1}) \cdot b = \mathscr{C} \cdot b \bullet (\mathscr{C}')^{-1} \cdot b$$
$$= \mathscr{C} \cdot b \bullet (b \cdot \mathscr{C}')^{-1} = (\mathscr{C} \cdot b) / (b \cdot \mathscr{C}')$$

– Separation.

$$b \cdot (\mathscr{C} \div \mathscr{C}') = b \cdot (\mathscr{C} \parallel (\mathscr{C}')^{-1}) = b \cdot \mathscr{C} \parallel b \cdot (\mathscr{C}')^{-1}$$
$$= b \cdot \mathscr{C} \parallel (\mathscr{C}' \cdot b)^{-1} = (b \cdot \mathscr{C}) \div (\mathscr{C}' \cdot b)$$
$$(\mathscr{C} \div \mathscr{C}') \cdot b = (\mathscr{C} \parallel (\mathscr{C}')^{-1}) \cdot b = \mathscr{C} \cdot b \parallel (\mathscr{C}')^{-1}$$
$$= (\mathscr{C} \cdot b) \div \mathscr{C}'$$
$$(\mathscr{C} \div \mathscr{C}') \cdot b = (\mathscr{C} \parallel (\mathscr{C}')^{-1}) \cdot b$$
$$= \mathscr{C} \parallel (\mathscr{C}')^{-1} \cdot b = \mathscr{C} \div (b \cdot \mathscr{C}')$$

– Implication.

$$b \cdot (\mathscr{C}' \rightarrow \mathscr{C}) = b \cdot \big((a \wedge \neg a') \vee (g' \wedge \neg g), g \vee \neg g'\big)$$
$$= \big(b \wedge (a \wedge \neg a') \vee (b \wedge g' \wedge \neg g), (b \rightarrow g) \vee \neg g'\big)$$
$$= \begin{pmatrix} ((b \wedge a) \wedge \neg a') \vee (g' \wedge \neg(b \rightarrow g)), \\ (b \rightarrow g) \vee \neg g' \end{pmatrix}$$
$$= \mathscr{C}' \rightarrow b \cdot \mathscr{C}$$
$$b \cdot (\mathscr{C}' \rightarrow \mathscr{C}) = b \cdot \big((a \wedge \neg a') \vee (g' \wedge \neg g), g \vee \neg g'\big)$$
$$= \begin{pmatrix} (a \wedge \neg(b \rightarrow a')) \vee (b \wedge g' \wedge \neg g), \\ (b \wedge g') \rightarrow g \end{pmatrix}$$
$$= \mathscr{C}' \cdot b \rightarrow \mathscr{C}$$
$$(\mathscr{C}' \rightarrow \mathscr{C}) \cdot b = \big((a \wedge \neg a') \vee (g' \wedge \neg g), g \vee \neg g'\big) \cdot b$$
$$= \begin{pmatrix} (b \rightarrow (a \wedge \neg a')) \vee (b \rightarrow (g' \wedge \neg g)), \\ (b \wedge g) \vee \neg(b \rightarrow g') \end{pmatrix}$$
$$= \begin{pmatrix} (b \rightarrow a) \wedge \neg(b \wedge a') \vee (b \rightarrow g') \wedge \neg(b \wedge g), \\ (b \wedge g) \vee \neg(b \rightarrow g') \end{pmatrix}$$
$$= b \cdot \mathscr{C}' \rightarrow \mathscr{C} \cdot b$$

– Coimplication.

$$b \cdot (\mathscr{C}' \nrightarrow \mathscr{C}) = \big(((\mathscr{C}')^{-1} \rightarrow \mathscr{C}^{-1}) \cdot b\big)^{-1}$$
$$= \big(b \cdot (\mathscr{C}')^{-1} \rightarrow \mathscr{C}^{-1} \cdot b\big)^{-1} = \mathscr{C}' \cdot b \nrightarrow b \cdot \mathscr{C}$$
$$(\mathscr{C}' \nrightarrow \mathscr{C}) \cdot b = \big(b \cdot ((\mathscr{C}')^{-1} \rightarrow \mathscr{C}^{-1})\big)^{-1}$$
$$= \big((\mathscr{C}')^{-1} \rightarrow b \cdot \mathscr{C}^{-1}\big)^{-1} = \mathscr{C}' \nrightarrow \mathscr{C} \cdot b$$
$$= \big((\mathscr{C}')^{-1} \cdot b \rightarrow \mathscr{C}^{-1}\big)^{-1} = b \cdot \mathscr{C}' \nrightarrow \mathscr{C} \qquad \square$$