Contract Replaceability for Ensuring Independent Design using Assume-Guarantee Contracts

Sheng-Jung Yu

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-238
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-238.html

December 27, 2024

Acknowledgement

I am deeply grateful to my advisor, Professor Alberto Sangiovanni-Vincentelli, for his guidance and encouragement, which shaped both my research and personal growth. I also thank Professor Sanjit A. Seshia for his valuable feedback that greatly improved this work.

Special thanks to Inigo Incer for our discussions on contracts, which enriched this project, and to Shaokai Lin, Zheng Liang, Tung-Wei Lin, and the members of Professor Alberto's group for their support and camaraderie.

Finally, I express my heartfelt gratitude to my family for their unwavering support, especially during the pandemic. Their love and belief in me made this milestone possible.

Contract Replaceability for Ensuring Independent Design using Assume-Guarantee Contracts

by Sheng-Jung Yu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Alberto Sangiovanni Vincentelli

Professor Alberto Sangiovanni-Vincentelli
Research Advisor

November 20, 2024

(Date)

* * * * * * *

Professor Sanjit Seshia
Second Reader

December 20, 2024

(Date)

Contract Replaceability for Ensuring Independent Design using Assume-Guarantee Contracts

by

Sheng-Jung Yu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alberto Sangiovanni-Vincentelli, Chair
Professor Sanjit Seshia

Spring 2025

Abstract

Contract Replaceability for Ensuring Independent Design using Assume-Guarantee Contracts

by

Sheng-Jung Yu

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Complexity and heterogeneity are fundamental challenges for system design, as they prolong the design process and increase its cost. *Independent design* is a promising design flow to address these challenges whereby a supplier can develop its component without exchanging system-level information with other suppliers. Recent research on assume-guarantee contracts and contract-based design has focused on algebraic concepts, such as refinement and composition, to achieve independent design. However, the conventional definition of assume-guarantee contracts may result in implementations that may not operate correctly in the targeted environment of the system, thus hindering independent design. In this report, we introduce the concept of *contract replaceability*, a binary relation on contracts that prevents this problem. We then extend the requirements to include *receptiveness* as a constraint on assume-guarantee contracts to ensure strong replaceability. The properties derived from the constraints ensure that strong replaceability is satisfied under contract refinement and cascade composition. Thus any assume-guarantee contract that satisfies this constraint permits independent design.

Contents

List of Figures

Acknowledgments

I would like to extend my gratitude to my advisor, Professor Alberto Sangiovanni-Vincentelli, for his guidance and support throughout the project. His insightful advice and encouragement have been instrumental in shaping both my research and personal growth. I am also deeply thankful to Professor Sanjit A. Seshia for his thoughtful feedback and valuable insights on my research, which have greatly improved the quality of this work.

I would like to express my sincere appreciation to my colleague, Inigo Incer, for our many discussions on contracts. His expertise and experience in the domain provided immense support and significantly enriched this project.

Additionally, I want to thank my colleagues Shaokai Lin, Zheng Liang, and Tung-Wei Lin, as well as all the members of Professor Alberto's research group, for their genuine support and camaraderie during my studies.

Finally, I would like to express my deepest gratitude to my beloved family for their constant support across the Pacific Ocean. Their encouragement was especially meaningful during the pandemic, a period marked by separation and isolation. I would not have reached this milestone without their love, strength, and belief in me.

Chapter 1

Introduction

As the needs for large-scale systems, such as autonomous driving, Industry 4.0, and artificial intelligence-based applications, increased over the last decades, complexity and heterogeneity have become the main challenges that prolong the design process and increase its cost [25, 26]. Several methodologies and algorithms have been proposed to cope with design complexity and heterogeneity in all design aspects including specification, verification, and synthesis [20, 21, 7, 26]. Among them, design specification is crucial, as it is the first stage in a rigorous design flow. Methodologies for design specification affect efficiency in verification and synthesis, the subsequent stages of a design flow.

Contract-based design [26, 22] tackles complexity and heterogeneity coupled with platform-based design and formal specifications and thus has become a promising candidate for facilitating complex and heterogeneous design. Contracts are formal specifications [3] for the design environment and its implementation. Contract-based design is a methodology that utilizes contracts in platform-based design. It applies refinement and abstraction to reduce complexity and separates orthogonal viewpoints, or aspects, of a design to handle the heterogeneity of the design [4]. Among many formalisms, assume-guarantee contracts, consisting of an assumption set and a guarantee set, are attractive in research because of their ease of use. The close-formed formulas of the assume-guarantee contract operations, such as composition and quotient [24, 15], have been derived to facilitate contract-based design.

Independent design [3] is a benefit brought by contract-based design. It allows earlier verification of the system and protects the trade secrets between designers and suppliers. In the independent design paradigm, system-level specifications are refined with more detailed information and decomposed into multiple parts where the composition of these parts satisfies the system-level specifications. The refinement and decomposition ensure that the system meets the top-level requirement once each part follows its local specification. Every supplier thus can independently develop the part under its specification without the system-level specifications or coordination between the suppliers. As a result, the paradigm captures design faults at the specification stage to avoid costly and time-consuming redesign processes and protects the high-level design ideas from leaking to the suppliers, which might be different companies.

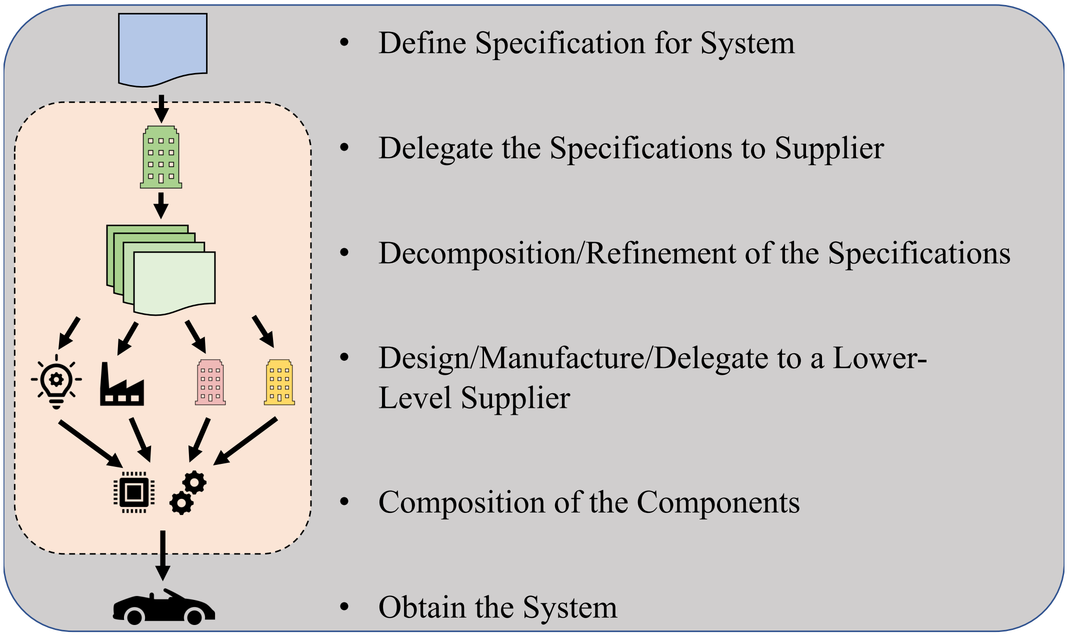

• Define Specification for System
• Delegate the Specifications to Supplier
• Decomposition/Refinement of the Specifications
• Design/Manufacture/Delegate to a Lower-Level Supplier
• Composition of the Components
• Obtain the System

Figure 1.1: Overview of the independent design flow.

Figure 1.1 shows the ideal flow of independent design. First, the top-level specification for the product is decomposed into the specifications of multiple subsystems or parts. These specifications of the parts are sent to different suppliers proficient in the domain knowledge to design and provide implementations for the parts. These suppliers can refine their specifications, add more detailed information to the specification, or further decompose the part specification into more part specifications and then delegate them to subsequent suppliers for implementation. After a provider completes an implementation, the implementation is sent back to the system integrator and composed into the target system.

However, the machinery of assume-guarantee contracts does not rule out that the implementations generated for systems or components may not operate correctly in their targeted environments. These implementations, which we call *vacuous implementations*, have empty sets of behaviors in the targeted environment (i.e., not compatible with the environment). Therefore, vacuous implementations should be avoided in the independent design flow. As these vacuous implementations are not excluded from the standard contract framework, additional requirements and constraints enforced on contracts are required to support independent design using assume-guarantee contracts.

In this work, we investigate the requirements and additional constraints on assume-guarantee contracts to provably avoid vacuous implementations. Our contributions are the following:

- We identify the vacuous implementation problem as an obstacle to the independent design paradigm. To the best of our knowledge, this is the first work that discovers and addresses this problem of the application of the contract-based design methodology to independent design.

- We introduce *replaceability*, a binary relation, as a sufficient condition to prevent the vacuous implementation problem. A refinement of a contract that follows *replaceability* is guaranteed to contain implementations compatible with the contract.

- We then introduce *strong replaceability*, a transitive binary relation, as a restriction on refinement to prevent the vacuous implementation problem in successive refinement steps and enable the independent design. Strong replaceability relieves the need for system contracts to enforce the requirement of replaceability. Once each refinement step follows *strong replaceability*, the resulting contract must follow replaceability with the system contract.

- We introduce the concept of *receptiveness* as a property of assume-guarantee contracts. We show that receptiveness is sufficient to ensure strong replaceability, and the independent design paradigm is permitted on receptive contracts for refinement and cascade composition.

This report is based on the based on joint work with Inigo Incer, and Alberto Sangiovanni-Vincentelli [27].

The remainder of the report is organized as follows: Chapter 2 introduces assume-guarantee contracts. Chapter 3 describes the vacuous implementation problem and formulates the contract replaceability requirement. Chapter 4 proposes the notion of contract receptiveness. Chapter 5 and Chapter 6 show that contract receptiveness ensures strong replaceability in refinement and cascade composition, respectively. Finally, Chapter 8 concludes the report.

Chapter 2

Preliminaries

This chapter introduces the elements in assume-guarantee contracts, the contract-based design methodology, and their background.

Variables, Behaviors, and Projections

A system specification defines its interaction with the external environment. The interaction is represented by the behaviors over the variables of the system.

Variables For our purposes, the variable is the port of the system that interacts with the external environment. Each variable is associated with a variable type, the set of values the variable can take. The collection of all variables in the system is the *variable set* of the system, denoted by V. For example, a logical AND gate with input ports a and b, and output port c has the variable set $V = \{a, b, c\}$, and the variable types for a, b, and c are the Boolean domains $\mathbb{B}$.

Behaviors and Components A behavior is a successive assignment of values to the variables in the system. Take the logical AND gate for example, $(a, b, c) = (T, T, T), (T, F, F), (F, F, F) \ldots$ is one behavior. To simplify notation, in our examples, we will denote behaviors statically, i.e., taking constant values for all time steps. However, all of the definitions and properties used throughout the report can be applied to arbitrary behaviors.

For a variable $v \in V$, we denote the universe of the behaviors over the variable v by $\mathcal{B}_v$. The universe of system behaviors over the variable set V, represented as $\mathcal{B}_V$, is defined by the direct product of the universe of behaviors for all system variables, i.e., $\prod_{v \in V} \mathcal{B}_v$. For example, the universe of the behaviors over the variable set of the logical AND gate is $\mathcal{B}_V = \mathbb{B}^3$.

A component $M \subseteq \mathcal{B}_V$ is a set of behaviors. We understand the component as containing the behaviors that one can observe from it. The behavior set can be expressed as a predicate over the variables. All behaviors leading to the truth value of the predicate belong to the behavior set. In the logical AND gate example, the behavior set is $\{(a, b, c) \in \mathbb{B}^3 \mid a \wedge b = c\}$.

If the variable set is well-known in the context, we can express the behavior set using only the predicate: $a \wedge b = c$. Elaboration on these syntactic issues can be found in Chapter 7 of [12].

The composition of two components M_1 and M_2 is the intersection of both behavior sets $M_1 \cap M_2$. We understand this intersection as the simultaneous enforcement of the constraints imposed by M_1 and M_2 on the system's behaviors. In other words, the composition of two components can be represented as the conjunction of their predicates (i.e., constraints).

Projections of Behaviors We will often need to map behaviors to behaviors defined over a different set of variables. To define this map, we first define the behaviors restricted to a single variable $v \in V$.

Let $e \in \mathcal{B}_V$ be a behavior and $v \in V$, we denote e_v as the behavior restricted to the variable v.

Here we define the projection of behaviors. Let V and V' be two variable sets and $B_{V'} \subseteq \mathcal{B}_{V'}$ be a set of behaviors defined in V', the projection of behaviors $B_{V'}$ to variables V is as follows:

$$\Pi_V(B_{V'}) = \left\{ e \in \mathcal{B}_V \;\middle|\; \begin{array}{l} \exists e' \in B_{V'} \ (\forall v \in V \cap V' \ e_v = {e'}_v) \ \wedge \\ (\forall v \in V \setminus V' \ {e'}_v \in \mathcal{B}_v) \end{array} \right\}$$

Each behavior e after the projection corresponds to some behavior $e' \in B_{V'}$, sharing the same assignments for the common variables and having arbitrary assignments for the variables not in V'.

Take the logical AND gate as an example again, the projection of its behavior to variables $\{b, c\}$ is $\{(b, c) \in \mathbb{B}^2 \mid \neg c \vee b\}$. Consider another example of the removal of a variable and the inclusion of an additional variable at the same time. If we are considering the system which contains an additional input d that is unrelated to this logical AND gate, the projection to variables $\{b, c, d\}$ is $\{(b, c, d) \in \mathbb{B}^3 \mid \neg c \vee b\}$.

Regarding notation, if e is a behavior, we will sometimes write $\Pi_V(e)$ to mean $\Pi_V(\{e\})$. Similarly, when $\Pi_V(e)$ is a singleton, we will sometimes use $\Pi_V(e)$ to denote the element it contains. In those cases, we may see statements such as $\Pi_V(\{e\}) \in \mathcal{B}_V$.

Assume-Guarantee Contracts and Contract-Based Design

An assume-guarantee contract $\mathcal{C}$, as a formal specification for a system, is a pair of behavior sets (A, G), where A is the assumption set, and G is the guarantee set. Both behavior sets share the same contract variable set $V_{\mathcal{C}}$, the collection of all ports in the system. The assumption set specifies the targeted environments where the system is expected to operate. The guarantee set describes the acceptable behaviors when the system operates under the targeted environment. When the system operates in an environment not specified by the assumption set, all behaviors are acceptable. As a result, the acceptable behavior set of a contract $\mathcal{C} = (A, G)$ is $G \cup \overline{A}$. An implementation of the contract is a component $M_{\mathcal{C}}$

which meets the contract. Therefore, the behavior set of an implementation must satisfy $M_{\mathcal{C}} \cap A \subseteq G$, and it is thus any subset of the acceptable behavior set: $M_{\mathcal{C}} \subseteq G \cup \overline{A}$.

A contract is a saturated contract if it satisfies $G \cup \overline{A} = G$, meaning that the guarantee set includes all the acceptable behaviors. Any contract can be saturated by replacing $\mathcal{C} = (A, G)$ with $(A, G \cup \overline{A})$. The semantics of the specification remains unchanged, as saturation does not change the acceptable behaviors.

Contract-based design is a design methodology that specifies a system using contracts and exploits the theory of contracts to operate on the contracts during the design process. The refinement and the composition are the main operations in contract-based design. The refinement increases the details of the specifications, and composition integrates multiple subsystem contracts into a contract for the entire system. The contract-based design process is as follows: First, the designers specify a system-level contract to reflect the goal of the design. The contract is then refined and decomposed into multiple subsystem contracts whose composition refines the original contracts. The process continues until the detail of the contract for each subsystem is sufficient for actual implementation, such as a circuit layout, a CAD model for a mechanical part, and the control parameters for a controller design. The contract-based design leverages this hierarchical flow and abstraction of the system to tackle design complexity and capture design faults earlier to avoid long design cycles.

The assume-guarantee contracts support the composition and the refinement for contract-based design using the set operations [26]. A contract $\mathcal{C}_1 = (A_1, G_1)$ is a refinement of an abstract contract $\mathcal{C}_2 = (A_2, G_2)$, denoted by $\mathcal{C}_2 \succeq \mathcal{C}_1$, if $A_1 \supseteq A_2$ and $G_1 \subseteq G_2$. The composition of two contracts, denoted by $\mathcal{C}_1 \parallel \mathcal{C}_2$, can be computed as $((A_1 \cap A_2) \cup \overline{(G_1 \cap G_2)}, G_1 \cap G_2)$.

For a given contract $\mathcal{C} = (A, G)$ we define the non-assumption variables V_G and the assumption variable set V_A. The non-assumption variable set is a subset of $V_{\mathcal{C}}$ that is insensitive to the assumption set, defined formally as follows:

$$V_G = \left\{ v \in V_{\mathcal{C}} \;\middle|\; \begin{array}{l} \forall e \in \mathcal{B}_V \\ (\Pi_{V_{\mathcal{C}}}(\Pi_{V_{\mathcal{C}}/\{v\}}(e)) \subseteq A) \vee \\ (\Pi_{V_{\mathcal{C}}}(\Pi_{V_{\mathcal{C}}/\{v\}}(e)) \subseteq \overline{A}) \end{array} \right\}.$$

The intuition of the definition is that the value of the non-assumption variable does not affect the satisfaction of the assumption for all behaviors. The set $\Pi_{V_{\mathcal{C}}}(\Pi_{V_{\mathcal{C}}/\{v\}}(e))$ contains all behaviors that have the same value as e for all variables except for v.

The assumption variable set V_A is defined as $V_A = V_{\mathcal{C}} \setminus V_G$, the set difference of $V_{\mathcal{C}}$ and V_G.

For example, considering the contract $\mathcal{C} = (A, G) = (x \geq 0 \wedge y \geq 0, z = x + y), V_{\mathcal{C}} = \{x, y, z\}$, the non-assumption variable set V_G is $\{z\}$ and the assumption variable V_A is $\{x, y\}$.

Related Work

The notion of contracts derives from a software engineering technique using preconditions and postconditions to specify and verify a program method. Once the preconditions, the responsibilities of the caller, are satisfied, the method ensures the postconditions. Variants of these specifications are also proposed to reason different systems. For example, Jones [16] proposes rely-guarantee reasoning for concurrent programming, adding rely-conditions, the assumptions on the changes of global states made by other processes, and guarantee conditions, the changes of global states that can be made by the programs itself. Meyer [18] was the first to use the term "contract" in its proposed design methodology, as an analogy to business contracts between the caller of function and the function itself.

The contracts are later adopted in the (cyber-physical) system design domain for formal verification and specifications of components operating in parallel. Abadi *et al.* [2, 1] was the first to represent specifications as assumptions and guarantees for Transition Systems and their composition. Sangiovanni-Vincentelli *et al.* [26] then developed the contract-based design methodology, which utilizes contracts in the platform-based design for cyber-physical system designs. The contracts in the methodology include horizontal contracts and vertical contracts. The horizontal contracts describe the interactions of the components in the same level of abstraction, and the vertical contracts capture additional assumptions between different levels of abstraction. Applications of the contract-based design methodology to build a systematic design flow, such as aircraft Electricity Power System [20] and analog system interface design [21], have demonstrated its potentials in handling complex design problems. Extensions of contracts formalism are also proposed in different applications, such as optimization of controller designs [23], stochastic systems [22], and hyperproperties [13] We refer interested readers to Chapter 3 of the monograph by Benveniste *et al.* [3], which provides a comprehensive background of contracts in software engineering and their adoption by the cyber-physical domain in formal verification and specification.

This report focuses on the problem in the application of refinement in contract-based design. Many works have proposed algorithms for verifying and generating refinement of contracts. Cimatti *et al.* [6, 5] proposed the property-based proof systems to check whether a system is refined by the submodule contracts. The algorithm tests whether the guarantees generated by all submodules satisfy the top-level guarantees, and any top-level environments operating with all submodules create an environment for each submodule. Le *et al.* [17] proposed a similar paradigm more generically by defining a set of metatheoretical operators which allows the proof strategy to apply in different contract frameworks. Iannopollo *et al.* [11] adopted a hierarchical verification strategy and proposed a library-based contract refinement checking algorithm. The algorithm utilizes pre-checked refinement relations in the library to accelerate the verification. Iannopollo *et al.* [9, 8] also proposed a counter-example guided inductive synthesis-based constrained synthesis flow to synthesize contracts from a library of components or contracts specified using linear temporal logic. Their subsequent work [10] improves the synthesis efficiency by hierarchically decomposing the contracts into smaller contracts. These works are not aware of the potential vacuous implementation problem in the refinement process, the key enabler of the independent design paradigm. To the best of our knowledge, this is the first work that formally defines the requirement for independent design using contracts and introduces constraints to address the problem. As a result, this work complements the algorithms and tools by identifying the requirements for ensuring independent design. By enforcing the requirements, independent design can be ensured using the contract-based design without worrying about the vacuous implementation problem.

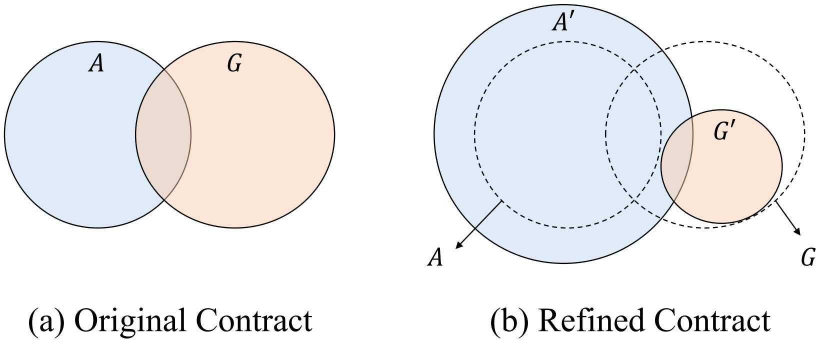
Receptiveness is the foundation for our proposed receptive contracts. The concept of receptiveness, which originates from the implementation point of view, was first proposed by Benveniste *et al.* [4], where receptiveness is defined over behaviors as any values for the specified variables corresponding to some behaviors restricted by an assertion. Then the consistency of a contract is defined as the guarantee being **u**-receptive, where **u** stands for the uncontrolled variables in the variable set. The same notion for receptiveness was also mentioned in the later works [3, 5, 7, 26], while Benveniste *et al.* [3] and Inigo *et al.* [15] define the contract consistency differently as contracts containing nonempty implementations set.

However, these works do not show the relation of receptiveness to the ability of independent design, as they intend to ensure receptive implementations and semantically separate the responsibilities of the assumption and guarantees instead of independent design. Their definition based on predefined partitioning of variables also limits the application of contracts as it cannot apply to components without rigorous input-output ports such as ones with bidirectional ports. Furthermore, being **u**-receptive requires the guarantees to include behaviors rejected by its assumptions, and thus the guarantees have larger behavior set sizes and contain redundant information. Therefore, taking the notion of receptiveness for behaviors as the foundation, our work defines receptiveness for contracts which does not contain redundant information by requiring receptiveness only for the behaviors accepted by assumptions. We show that our proposed receptive contracts ensure independent design and it does not rely on predefined partitioning of controlled and uncontrolled variables.

Chapter 3

The Vacuous Implementation Problem

As introduced before, the empty sets of behaviors could be problematic in the set-based definition of contracts. Given a contract $\mathcal{C} = (A, G)$, an implementation of the contract $\mathcal{C}$ is a component $M_{\mathcal{C}}$ such that $M_{\mathcal{C}} \cap A \subseteq G$. Since an empty set is a subset of G, a component $M'_{\mathcal{C}}$ such that $M'_{\mathcal{C}} \cap A = \emptyset$ is by definition an implementation of $\mathcal{C}$. However, this implementation is not compatible with the targeted environment A. We call such an implementation a "vacuous implementation" of the contract. We also define a "strict implementation" of the contract $\mathcal{C}$ as an implementation $M_{\mathcal{C}}$ such that $M_{\mathcal{C}} \cap A \neq \emptyset$.



(a) Original Contract

(b) Refined Contract

Figure 3.1: A scenario that all implementations in the refined contract are vacuous implementations since they form an empty set when intersecting with the original contract assumption.

During the design process, we should avoid vacuous implementations and guarantee strict implementations. However, the refinement of contracts results in smaller acceptable behavior sets, and thus we may lose all strict implementations for the original contracts. Consider a scenario where the contract is $\mathcal{C} = (A, G)$ and its refinement contract is $\mathcal{C}' = (A', G')$ such that the acceptable behavior set of the refined contract and the original assumption set are

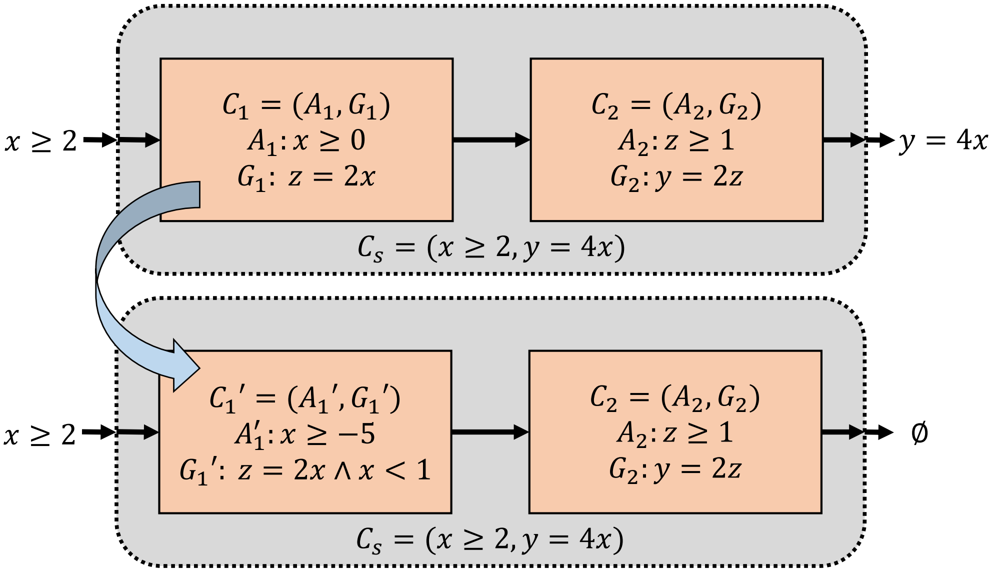


Figure 3.2: A motivating example that shows the vacuous implementation problem in contract refinement. All implementations based on the refined composition $\mathcal{C}_1 \parallel \mathcal{C}_2$ are vacuous implementations for $\mathcal{C}_s$.

disjoint, as illustrated in Figure 3.1. All implementations M' of the contract $\mathcal{C}'$ are vacuous implementations since $M' \cap A \subseteq (G' \cup \overline{A'}) \cap A = \emptyset$.

In this section, we first provide a motivating example of the vacuous implementation problem in the independent design and then formulate the requirements to avoid the problem.

Motivating Example

Figure 3.2 shows an example that the refinement of contracts results in vacuous implementations. Let the system contract be $\mathcal{C}_s = (A_s, G_s) = (x \geq 2, y = 4x)$, and two contracts be $\mathcal{C}_1 = (A_1, G_1) = (x \geq 0, z = 2x)$ and $\mathcal{C}_2 = (A_2, G_2) = (z \geq 1, y = 2z)$ as its subsystems whose composition refines the system contract. All contracts are defined on the variable set $V_s = \{x, y, z\}$. The two subsystem contracts are then sent to different suppliers for independent development. If the supplier for $\mathcal{C}_1$ refines the contract as $\mathcal{C}_1' = (A_1', G_1') = (x \geq -5, z = 2x \wedge x < 1)$ during the design process, the composition of $\mathcal{C}_1'$ and $\mathcal{C}_2$ remains a refinement for the system-level contracts. However, all implementations of the composition are vacuous implementations for $\mathcal{C}_s$, as shown in the following derivation:

$$\begin{aligned}(M_1' \cap M_2) \cap A_s &\subseteq (G_1' \cup \overline{A_1'}) \cap (G_2 \cup \overline{A_2}) \cap A_s \\ &\subseteq (G_1' \cup \overline{A_1'}) \cap A_s \\ &\subseteq ((z = 2x \wedge x < 1) \vee (x < -5)) \wedge (x \geq 2) \\ &= \emptyset,\end{aligned}$$

where M_1' is any implementation for $\mathcal{C}_1'$ and M_2 denotes any implementation for $\mathcal{C}_2$.

The example shows that the refinement can result in the vacuous implementation problem during independent design. Therefore, we hope to restrict the refinement to guarantee strict implementations, and the vacuous implementation problem can avoid the problem in independent development as long as all suppliers follow the restriction. In the next part, we formulate the requirement to avoid vacuous implementations and develop a restriction based on the requirement.

Requirement for Independent Design

We define contract replaceability as the requirement to guarantee strict implementation:

Definition 1. *Let $\mathcal{C}_1 = (A_1, G_1)$ and $\mathcal{C}_2 = (A_2, G_2)$ be saturated contracts that satisfy $\mathcal{C}_1 \succeq \mathcal{C}_2$ and share the same variable set $V_\mathcal{C}$ and assumption variable set V_A. We say that $\mathcal{C}_1$ is replaceable by $\mathcal{C}_2$, or that $\mathcal{C}_2$ replaces $\mathcal{C}_1$, if the following condition is satisfied:*

$$\exists e \in \Pi_{V_A}(A_1), \Pi_{V_\mathcal{C}}(e) \cap G_2 \neq \emptyset,$$

or, equivalently, $A_1 \cap G_2 \neq \emptyset$.

Contract replaceability requires a projected behavior e in the assumption set A_1 such that the intersection of the guarantee set and the behavior projected back to the entire variable set $V_\mathcal{C}$ is not an empty set. As a result, a behavior with the assignments of the assumption variables satisfying A_1, the targeted environment, can be found in G_2, the refined guarantee. A binary relation called the contract replaceability relation is defined as the set containing all contract pairs $(\mathcal{C}_1, \mathcal{C}_2)$ such that $\mathcal{C}_1$ replaces $\mathcal{C}_2$.

Contract replaceability ensures that the strict implementations for the original contracts can be found using the refined contract, summarized in Theorem 1:

Theorem 1. *Let $\mathcal{C}_1 = (A_1, G_1)$ and $\mathcal{C}_2 = (A_2, G_2)$ be saturated contracts over the same variable set $V_\mathcal{C}$ and assumption variable set V_A such that $\mathcal{C}_2$ refines $\mathcal{C}_1$ and $\mathcal{C}_2$ replaces $\mathcal{C}_1$. Any implementation M_2 of $\mathcal{C}_2$ such that $M_2 \supseteq (G_2 \cap A_2)$ is a strict implementation for $\mathcal{C}_1$.*

Proof. We prove Theorem 1 by showing that $M_2 \cap A_1$ is not an empty set:

$$M_2 \cap A_1 \supseteq (G_2 \cap A_2) \cap A_1 = G_2 \cap A_1 \neq \emptyset,$$

where the equality is by the definition of refinement that $A_1 \subseteq A_2$, and the inequality is by the definition of contract replaceability. Therefore, $M_2 \cap A_1$ is a superset of a non-empty set, which means $M_2 \cap A_1$ is not an empty set and thus a strict implementation for $\mathcal{C}_1$. □

As a result, once the system contract is replaceable by the refined contract, we can find a strict implementation for the system contract using the refined contract.

However, we need the assumption set from the system contract to ensure contract replaceability. In independent design, the supplier does not obtain the system contract but relies on a refined contract. Intuitively, we can require that the supplier guarantees contract replaceability for the refined contract instead of the system contract. Unfortunately, the contract replaceability relation is not transitive. A contract replacing the refined contract is not guaranteed to replace the system contract, as shown in the following example:

Example 1. *Consider the following three contracts $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$, where $\mathcal{C}_1 \succeq \mathcal{C}_2 \succeq \mathcal{C}_3$:*

$$\begin{aligned}
\mathcal{C}_1 &= (A_1, G_1) = (x \geq 0, y = 2x) \\
\mathcal{C}_2 &= (A_2, G_2) = (x \geq -2, (y = 2x \wedge x \leq 4) \vee (x < -2)) \\
\mathcal{C}_3 &= (A_3, G_3) = (x \geq -4, (y = 2x \wedge x \leq -1) \vee (x < -4)).
\end{aligned}$$

We can see that $\mathcal{C}_2$ replaces $\mathcal{C}_1$, and that $\mathcal{C}_3$ replaces $\mathcal{C}_2$. However, $\mathcal{C}_3$ does not replace $\mathcal{C}_1$ as $A_1 \cap G_3 = (x \geq 0) \wedge ((y = 2x \wedge x \leq -1) \vee (x < -4)) = \emptyset$ ■

To address the problem, a transitive relation that guarantees strict implementation is required. Thus, we propose *strong replaceability*:

Definition 2. *Let $\mathcal{C}_1 = (A_1, G_1)$ and $\mathcal{C}_2 = (A_2, G_2)$ be saturated contracts that satisfy $\mathcal{C}_1 \succeq \mathcal{C}_2$ and share the same variable set $V_{\mathcal{C}}$ and assumption variable set V_A. We say that $\mathcal{C}_1$ is strongly replaceable by $\mathcal{C}_2$, or $\mathcal{C}_2$ strongly replaces $\mathcal{C}_1$, if the following condition is satisfied:*

$$\forall e \in \Pi_{V_{A_1}}(A_1), \Pi_{V_{\mathcal{C}_1}}(e) \cap G_2 \neq \emptyset.$$

Strong replaceability requires that for all projected behaviors e in the assumption set A_1, the intersection of the guarantee set and the behavior projected back to the entire variable set $V_{\mathcal{C}_1}$ is not an empty set. As a result, for each assignment of the assumption variables satisfying A_1, we can always find a satisfying behavior in G_2. A binary relation called the strong replaceability relation is defined as the set containing all contract pairs $(\mathcal{C}_1, \mathcal{C}_2)$ such that $\mathcal{C}_1$ strongly replaces $\mathcal{C}_2$. The difference between replaceability and strong replaceability is the quantification of the projected behavior.

We can show that the strong replaceability relation is transitive:

Proposition 1. *Let $\mathcal{C}_1 = (A_1, G_1)$, $\mathcal{C}_2 = (A_2, G_2)$, and $\mathcal{C}_3 = (A_3, G_3)$ be saturated contracts over the same variable set $V_{\mathcal{C}}$ and assumption variable set V_A such that $\mathcal{C}_1 \succeq \mathcal{C}_2 \succeq \mathcal{C}_3$. If $\mathcal{C}_2$ strongly replaces $\mathcal{C}_1$ and $\mathcal{C}_3$ strongly replaces $\mathcal{C}_2$, then $\mathcal{C}_3$ strongly replaces $\mathcal{C}_1$.*

Proof. Since $\mathcal{C}_3$ strongly replaces $\mathcal{C}_2$, by the definition of strong replaceability:

$$\forall e_2 \in \Pi_{V_A}(A_2), \Pi_{V_\mathcal{C}}(e_2) \cap G_3 \neq \emptyset. \tag{3.1}$$

And $A_1 \subseteq A_2$ by the definition of contract refinement, so their projected behavior sets also hold the subset relation: $\Pi_{V_A}(A_1) \subseteq \Pi_{V_A}(A_2)$.

Therefore, $\forall e_1 \in \Pi_{V_A}(A_1), e_1 \in \Pi_{V_A}(A_2)$, and thus e_1 satisfies the qualification for (3.1): $\forall e_1 \in \Pi_{V_A}(A_1), \Pi_{V_\mathcal{C}}(e_1) \cap G_3 \neq \emptyset$. By the definition of strong replaceability, $\mathcal{C}_3$ strongly replaces $\mathcal{C}_1$. □

Combining Theorem 1 and Proposition 1, we conclude that strong replaceability guarantees strict implementations during independent design in Theorem 2

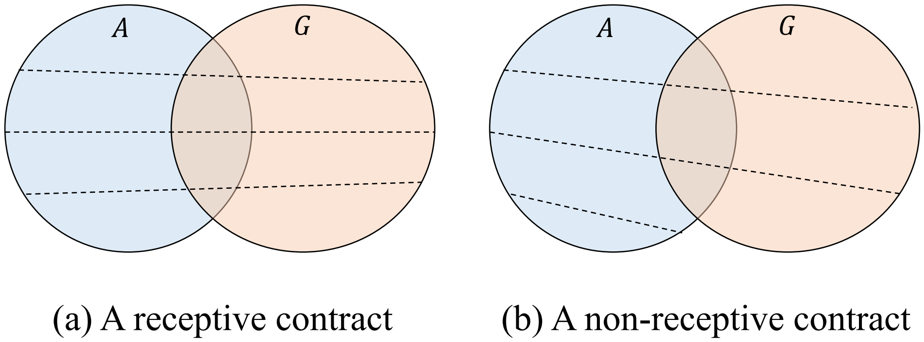
Theorem 2. *Let $\mathcal{C}_1 = (A_1, G_1)$, $\mathcal{C}_2 = (A_2, G_2)$, $\mathcal{C}_3 = (A_3, G_3)$, $\ldots, \mathcal{C}_n = (A_n, G_n)$ be saturated contracts over the same variable set $V_\mathcal{C}$ and assumption variable set V_A such that $\mathcal{C}_i \succeq \mathcal{C}_{i+1}$ for $i = 1 \ldots n-1$. If $\mathcal{C}_{i+1}$ strongly replaces $\mathcal{C}_i$ for $i = 1 \ldots n-1$, then any implementation M_n such that $M_n \supseteq A_n \cap G_n$ strictly implements $\mathcal{C}_1$.*

Therefore, we propose strong replaceability as the restriction for suppliers to perform contract refinement. As long as all the suppliers follow the restriction to ensure strong replaceability, strict implementations for the system contracts can be found by $A_n \cap G_n$.

Chapter 4

Receptive Contracts

We have formulated strong replaceability as a restriction to ensure strict implementations in independent design. However, the problem that the conventional operations in assume-guarantee contracts cannot ensure strict implementations is worth exploring. In this chapter, we propose contract receptiveness as a constraint for assume-guarantee contracts so that any operations in independent design under the constraint ensure strict implementations. We will show that the receptive contract guarantees strong replaceability for refinement in Chapter 5 and cascade composition in Chapter 6.



(a) A receptive contract

(b) A non-receptive contract

Figure 4.1: Illustrations of a receptive contract and a non-receptive contract. (a) A receptive contract as all its areas separated by the dashed lines intersect with the guarantee set. (b) A non-receptive contract as the area at the bottom of A does not intersect with the guarantee set.

Contract receptiveness is defined as follows:

Definition 3. *A receptive contract is a contract $\mathcal{C} = (A, G)$ satisfying the following condition:*

$$\forall e \in \Pi_{V_A}(A), \Pi_{V_\mathcal{C}}(e) \cap G \neq \emptyset.$$

A receptive contract requires that every assignment to the assumption variable set allowed by the assumption set corresponds to at least a behavior in the guarantee set. Figure 4.1 illustrates the concept of the receptive contract. The areas between the dashed line represent all the assumption set assignments, $\Pi_{V_A}(A)$ Each area in the receptive contract, as shown in Figure 4.1 (a), must contain a behavior in G, while some areas in a non-receptive contract, as shown in Figure 4.1 (b), do not contain any behavior in G.

Example 2. *The contract $\mathcal{C}_1$ in Example 1 is a receptive contract while the contracts $\mathcal{C}_2$ and $\mathcal{C}_3$ are not receptive contracts. To check the receptiveness of $\mathcal{C}_1$, we first find the assumption set assignments $\Pi_{V_{A_1}}(A_1) = \{x \mid x \geq 0\}$. For all assignments $x \geq 0$, we can find a behavior $(x, y) = (x, 2x)$ that is in G_1 and $\Pi_{V_{\mathcal{C}_1}}(x)$. Therefore, $\mathcal{C}_1$ is a receptive contract.*

Then we check the receptiveness of the contracts $\mathcal{C}_2$ and $\mathcal{C}_3$ in Example 1, The assignments of the assumption variable allowed by $\mathcal{C}_2$ is $\{x \mid x \geq -2\}$. However, as the guarantee set requires $x \leq 4$, any behavior with assignments of the assumption variable being $x > 4$ is not in the guarantee set. Similarly, for $\mathcal{C}_3$, the guarantee set requires $x \leq -1$, and thus any behavior with assignments of the assumption variable being $x > -1$ is not in the guarantee set. Therefore, the two contracts are not receptive. ■

Chapter 5

Refinement with Receptive Contracts

In this chapter, we show that the proposed receptive contracts guarantee strong replaceability for refinement during independent design and allow the suppliers to discover design faults in the specifications.

Theorem 3 states that receptive contracts guarantee strong replaceability in the refinement operation:

Theorem 3. *Let $\mathcal{C}_1 = (A_1, G_1)$ and $\mathcal{C}_2 = (A_2, G_2)$ be saturated contracts over the same variable set $V_\mathcal{C}$ and assumption variable set V_A such that $\mathcal{C}_1 \succeq \mathcal{C}_2$. If $\mathcal{C}_2$ is a receptive contract, then $\mathcal{C}_2$ strongly replaces $\mathcal{C}_1$.*

Proof. We prove Theorem 3 by converting the condition for receptiveness to the condition for strong replaceability:

As $\mathcal{C}_2$ is receptive, using the definition for the receptive contract we get:

$$\forall e_1 \in \Pi_{V_A}(A_2), G_2 \cap \Pi_{V_\mathcal{C}}(e_1) \neq \emptyset.$$

By the definition of contract refinement, $A_1 \subseteq A_2$, so $\Pi_{V_A}(A_1) \subseteq \Pi_{V_A}(A_2)$. Therefore, $\forall e_2 \in \Pi_{V_A}(A_1), e_2 \in \Pi_{V_A}(A_2)$. Combining the above results, we get

$$\forall e_2 \in \Pi_{V_A}(A_1), G_2 \cap \Pi_{V_\mathcal{C}}(e_2) \neq \emptyset.$$

Therefore, $\mathcal{C}_2$ strongly replaces $\mathcal{C}_1$ by the definition of strong replaceability. □

Receptive contracts guarantee strong replaceability not only for the abstract contracts before refinement but also for the system contract. To see this, we first show that a receptive refined contract implies that its abstract contract is also receptive:

Proposition 2. *Let $\mathcal{C}_1 = (A_1, G_1)$ and $\mathcal{C}_2 = (A_2, G_2)$ be saturated contracts over the same variable set $V_\mathcal{C}$ and assumption variable set V_A such that $\mathcal{C}_1 \succeq \mathcal{C}_2$. If $\mathcal{C}_2$ is a receptive contract, then $\mathcal{C}_1$ is also a receptive contract.*

Proof. During the proof in Theorem 3, we have derived the following:

$$\forall e_2 \in \Pi_{V_A}(A_1), G_2 \cap \Pi_{V_\mathcal{C}}(e_2) \neq \emptyset.$$

By the definition of contract refinement, $G_1 \supseteq G_2$:

$$\forall e_2 \in \Pi_{V_A}(A_1), G_1 \cap \Pi_{V_\mathcal{C}}(e_2) \supseteq G_2 \cap \Pi_{V_\mathcal{C}}(e_2) \neq \emptyset.$$

Therefore, $\mathcal{C}_1$ is a receptive contract by Definition 3. □

Strong replaceability for the system contracts, summarized in Theorem 4, can thus be derived by combining Proposition 2 and Theorem 3:

Theorem 4. *Let $\mathcal{C}_1 = (A_1, G_1)$, $\mathcal{C}_2 = (A_2, G_2)$, $\mathcal{C}_3 = (A_3, G_3)$, $\ldots, \mathcal{C}_n = (A_n, G_n)$ be saturated contracts over the same variable set $V_\mathcal{C}$ and assumption variable set V_A such that $\mathcal{C}_i \succeq \mathcal{C}_{i+1}$ for $i = 1 \ldots n-1$. If $\mathcal{C}_n$ is a receptive contract, then $\mathcal{C}_n$ strongly replaces $\mathcal{C}_1$.*

Proof. We prove Theorem 4 by induction. When $n = 2$, the statement holds by Theorem 3. Assume that the statement holds for $n = k$. When $n = k + 1$, $\mathcal{C}_{k+1}$ strongly replaces $\mathcal{C}_2$ by the assumption. $\mathcal{C}_2$ is a receptive contract as $\mathcal{C}_{k+1}$ is a receptive contract by Proposition 2. Applying Theorem 3 on $\mathcal{C}_2$ and $\mathcal{C}_1$, $\mathcal{C}_2$ strongly replaces $\mathcal{C}_1$. Therefore, by the transitivity of strong replaceability in Proposition 1, $\mathcal{C}_{k+1}$ strongly replaces $\mathcal{C}_1$. By mathematical induction, Theorem 4 holds for any $n \geq 2$. □

Furthermore, Theorem 4 implies that the suppliers can discover faults in the specification. We can impose receptiveness as a constraint on the assume-guarantee contract. The suppliers can rest assured that strong replaceability holds as long as the received abstract contract is receptive. In contrast, if the suppliers receive a non-receptive abstract contract, some faults must have occurred before the abstract contract was generated. Accordingly, the supplier can alert the specification provider to check for faults during the design process.

Chapter 6

Cascade Composition with Receptive Contracts

We have presented requirements for independent design under the refinement operations. However, in contract-based design, an abstract contract can be decomposed into several contracts whose composition refines the abstract contract. The decomposition of a contract is analogous to decomposing a system into several subsystems. Each subsystem follows the decomposed contracts. If a supplier receives one of the subsystem contracts and refines the subsystem contract, the supplier cannot check the strong replaceability of the composition without the other subsystem contracts. In this chapter, we discuss strong replaceability in composition using receptive contracts.

A composition is either a cascade composition or a feedback composition, depending on the topology of the subsystems. A cascade composition has subsystem order such that the assumption variable set of each subsystem only connects to the variables from the assumption variable set of the environment or the variables set from the preceding subsystems. We will use the subscript s to denote the system contract and numbers as subscripts to denote the order for the subsystem in a cascade composition. For example, let a system $\mathcal{C}_s$ be a cascade composition of two subsystem contracts $\mathcal{C}_1 \parallel \mathcal{C}_2$. Then $\mathcal{C}_1$ precedes $\mathcal{C}_2$, and thus by the definition of cascade composition, V_{A_1} must be a subset of V_{A_s}. A feedback composition is any composition that is not a cascade composition, meaning that subsystem order cannot be defined.

In this chapter, we discuss the following problem: Let the system contract be $\mathcal{C}_s = (A_s, G_s)$, Let $\mathcal{C}_1 = (A_1, G_1)$ and $\mathcal{C}_2 = (A_2, G_2)$ be saturated receptive contracts in a cascade composition such that $\mathcal{C}_s \succeq \mathcal{C}_1 \parallel \mathcal{C}_2$. Let $\mathcal{C}'_1 = (A'_1, G'_1)$ and $\mathcal{C}'_2 = (A'_2, G'_2)$ be saturated receptive contracts such that $\mathcal{C}_1 \succeq \mathcal{C}'_1$ and $\mathcal{C}_2 \succeq \mathcal{C}'_2$. All the behavior sets are defined in the variable set $V_{\mathcal{C}_s}$. We will show that the composition $\mathcal{C}'_1 \parallel \mathcal{C}'_2$ strongly replaces the system contract $\mathcal{C}_s = (A_s, G_s)$. For the feedback composition, we will present an example showing that the composition of the refined receptive contracts does not ensure strong replaceability. More constraints are thus required for general composition. We leave these additional constraints for feedback composition as future work.

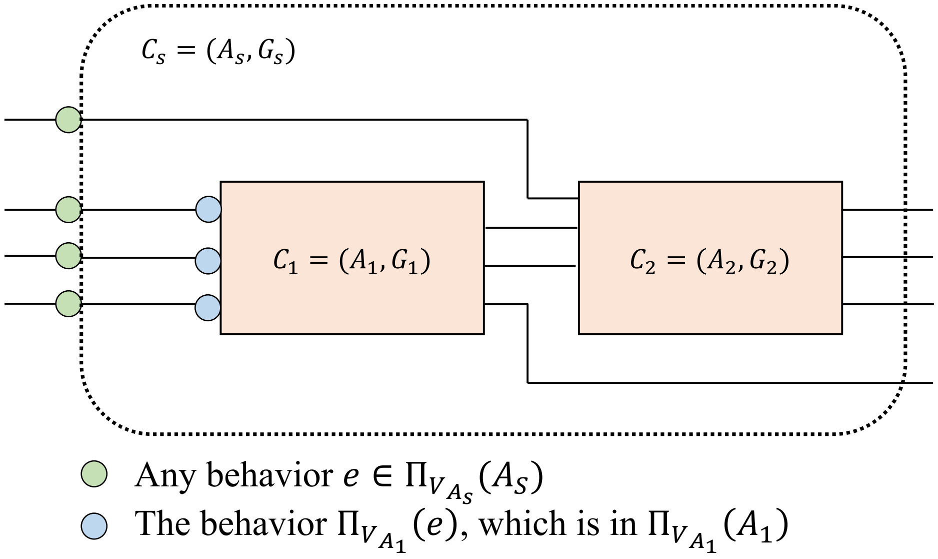

$C_s = (A_s, G_s)$
$C_1 = (A_1, G_1)$
$C_2 = (A_2, G_2)$
○ Any behavior $e \in \Pi_{V_{A_s}}(A_S)$
○ The behavior $\Pi_{V_{A_1}}(e)$, which is in $\Pi_{V_{A_1}}(A_1)$

Figure 6.1: Visualization of Lemma 1. Any behavior from the targeted assumption satisfies the assumption of $\mathcal{C}_1$.

We develop two lemmas to show strong replaceability of the cascade composition by receptive contracts. The first lemma, summarized in Lemma 1, states that any assignments to the assumption variable set of the system contract must satisfy the assumption of the first contracts.

Lemma 1. *Let $\mathcal{C}_s = (A_s, G_s)$, $\mathcal{C}_1 = (A_1, G_1)$, and $\mathcal{C}_2 = (A_2, G_2)$ be saturated receptive contracts such that $\mathcal{C}_s \succeq \mathcal{C}_1 \parallel \mathcal{C}_2$, then $\forall e \in \Pi_{V_{A_s}}(A_s), \Pi_{V_{A_1}}(e) \in \Pi_{V_{A_1}}(A_1)$.*

Figure 6.1 illustrates the concept in Lemma 1. The lemma is proved by contradiction: if $\Pi_{V_{A_1}}(e) \notin \Pi_{V_{A_1}}(A_1)$, then $e \notin \Pi_{V_{A_s}}(A_s)$.

Proof. Assume that e is a counterexample of Lemma 1 such that $e \in \Pi_{V_{A_s}}(A_s)$ and $\Pi_{V_{A_1}}(e) \notin \Pi_{V_{A_1}}(A_1)$. We want show that $e \notin \Pi_{V_{A_s}}(A_s)$, and thus the assumption leads to a contradiction.

First, we show that $\Pi_{V_{\mathcal{C}_s}}(e) \subseteq \overline{A_1}$ and $\Pi_{V_{\mathcal{C}_s}}(e) \subseteq G_1$. Since $\Pi_{V_{A_1}}(e) \notin \Pi_{V_{A_1}}(A_1)$, we can project the two sides back to $V_{\mathcal{C}_s}$:

$$\begin{aligned}
&\Pi_{V_{A_1}}(e) \notin \Pi_{V_{A_1}}(A_1)\\
\Longrightarrow\ &\Pi_{V_{\mathcal{C}_s}}(\Pi_{V_{A_1}}(e)) \subseteq \overline{\Pi_{V_{\mathcal{C}_s}}(\Pi_{V_{A_1}}(A_1))}\\
\Longrightarrow\ &\Pi_{V_{\mathcal{C}_s}}(e) \subseteq \overline{\Pi_{V_{\mathcal{C}_s}}(A_1)}\\
\Longrightarrow\ &\Pi_{V_{\mathcal{C}_s}}(e) \subseteq \overline{A_1} \subseteq G_1 \cup \overline{A_1} \subseteq G_1.
\end{aligned} \tag{6.1}$$

Then we discuss whether e can satisfy the assumption set of the second contract in two cases. The first case is $\Pi_{V_{A_2}}(e) \subseteq \overline{\Pi_{V_{A_2}}(A_2)}$ and the second case is $\Pi_{V_{A_2}}(e) \nsubseteq \overline{\Pi_{V_{A_2}}(A_2)}$.

Case 1

$$\begin{aligned}
&\Pi_{V_{A_2}}(e) \subseteq \overline{\Pi_{V_{A_2}}(A_2)} \\
\Longrightarrow\ &\Pi_{V_{\mathcal{C}_s}}(\Pi_{V_{A_2}}(e)) \subseteq \overline{\Pi_{V_{\mathcal{C}_s}}(\Pi_{V_{A_2}}(A_2))} \\
\Longrightarrow\ &\Pi_{V_{\mathcal{C}_s}}(e) \subseteq \overline{\Pi_{V_{\mathcal{C}_s}}(A_2)} \\
\Longrightarrow\ &\Pi_{V_{\mathcal{C}_s}}(e) \subseteq \overline{A_2} \subseteq G_2 \cup \overline{A_2} = G_2.
\end{aligned} \tag{6.2}$$

Combining (6.1) and (6.2), we get:

$$\begin{aligned}
\Pi_{V_{\mathcal{C}_s}}(e) &\subseteq (\overline{A_1} \cup \overline{A_2}) \cap (G_1 \cap G_2) \\
&\subseteq \overline{((A_1 \cap A_2) \cup \overline{(G_1 \cap G_2)})} \\
&\subseteq \overline{A_s}.
\end{aligned}$$

Therefore, $\Pi_{V_{\mathcal{C}_s}}(e) \subseteq \overline{A_s}$, and thus $e \notin \Pi_{V_{A_s}}(A_s)$, which contradicts our assumption that e is a counterexample.

Case 2 When $\Pi_{V_{A_2}}(e) \not\subseteq \overline{\Pi_{V_{A_2}}(A_2)}$, $\Pi_{V_{A_2}}(e) \cap \Pi_{V_{A_2}}(A_2) \neq \emptyset$. We can find a behavior $e_2 \in \Pi_{V_{A_2}}(e) \cap \Pi_{V_{A_2}}(A_2)$. Since $\mathcal{C}_2$ is a receptive contract, we can find a behavior $e_3 \in \Pi_{V_{\mathcal{C}_2}}(G_2) \cap \Pi_{V_{\mathcal{C}_2}}(e_2)$. Considering the behavior $e_4 = \Pi_{V_{\mathcal{C}_s}}(e_3) \cap \Pi_{V_{\mathcal{C}_s}}(e)$, we can get $e_4 \in \Pi_{V_{\mathcal{C}_s}}(e)$ and $e_4 \in \Pi_{V_{\mathcal{C}_s}}(e_3)$. Also, $e_4 \in \Pi_{V_{\mathcal{C}_s}}(e)$ implies $e_4 \in \overline{A_1} \subseteq G_1$.

Therefore, we can get:

$$e_3 \in \Pi_{V_{\mathcal{C}_2}}(G_2) \implies \Pi_{V_{\mathcal{C}_s}}(e_3) \subseteq G_2 \implies e_4 \in G_2.$$

As a result, we can derive that e_4 is not a behavior in A_s:

$$\begin{aligned}
e_4 &\in \overline{A_1} \cap (G_1 \cap G_2) \\
&\in \overline{((A_1 \cap A_2) \cup \overline{(G_1 \cap G_2)})} \\
&\in \overline{A_s}.
\end{aligned}$$

Therefore, $e_4 \in \overline{A_s}$, and thus $e = \Pi_{V_{A_s}}(e_4) \notin \Pi_{V_{A_s}}(A_s)$, which contradicts the assumption that e is a counterexample.

As both cases lead to contradictions, Lemma 1 is thus proved. □

The other lemma, as shown in Lemma 2, states that the behaviors of the first contract satisfy the assumption of the second contract if the behaviors meet the assumption of the system contract.

Lemma 2. *Let $\mathcal{C}_s = (A_s, G_s)$, $\mathcal{C}_1 = (A_1, G_1)$, and $\mathcal{C}_2 = (A_2, G_2)$ be saturated receptive contracts such that $\mathcal{C}_s \succeq \mathcal{C}_1 \parallel \mathcal{C}_2$, then $\forall e_1 \in \Pi_{V_{A_s}}(A_s), \forall e_2 \in \Pi_{V_{\mathcal{C}_1}}(G_1) \cap \Pi_{V_{\mathcal{C}_1}}(e_1), \Pi_{V_{A_2}}(e_1) \cap \Pi_{V_{A_2}}(e_2) \in \Pi_{V_{A_2}}(A_2)$.*

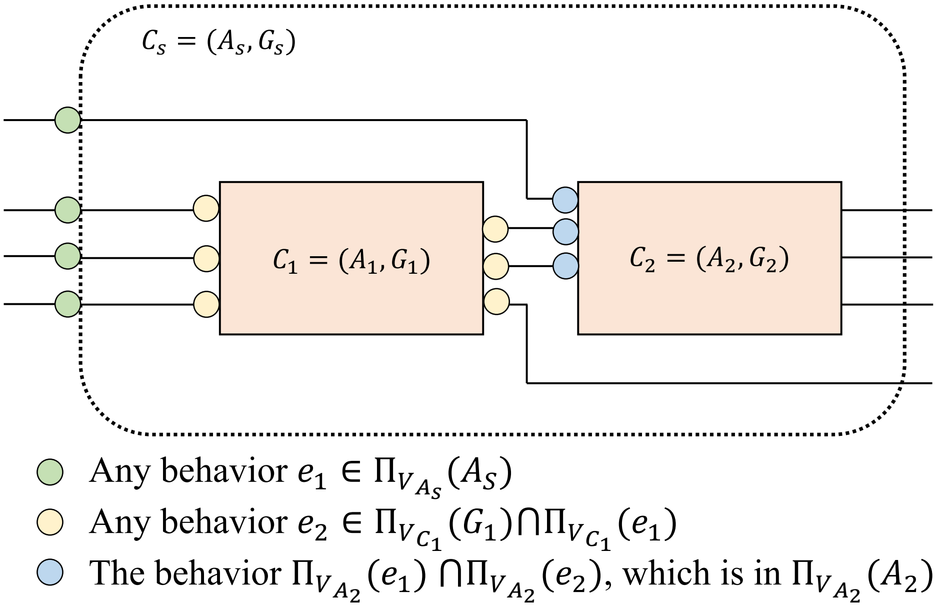

$C_s = (A_s, G_s)$
$C_1 = (A_1, G_1)$
$C_2 = (A_2, G_2)$
🟢 Any behavior $e_1 \in \Pi_{V_{A_s}}(A_S)$
🟡 Any behavior $e_2 \in \Pi_{V_{C_1}}(G_1) \cap \Pi_{V_{C_1}}(e_1)$
🔵 The behavior $\Pi_{V_{A_2}}(e_1) \cap \Pi_{V_{A_2}}(e_2)$, which is in $\Pi_{V_{A_2}}(A_2)$

Figure 6.2: Visualization of Lemma 2. The combined behavior of any behavior from the targeted assumption and the corresponding behavior generated by $\mathcal{C}_1$ satisfies the assumption of $\mathcal{C}_2$.

Figure 6.2 illustrates the concept in Lemma 2. The projected assumption from the system contracts and all the corresponding behaviors generated by G_1 must be in the assumption set of the second contract.

The lemma is proved by contradiction that if $\Pi_{V_{A_1}}(e_1) \notin \Pi_{V_{A_1}}(A_1)$, then $e \notin \Pi_{V_{A_s}}(A_s)$:

Proof. Assume that $e \in \Pi_{V_{A_s}}(A_s)$ and that $e_2 \in \Pi_{V_{\mathcal{C}_1}}(G_1) \cap \Pi_{V_{\mathcal{C}_1}}(e_1)$ forms a counterexample such that $\Pi_{V_{A_2}}(e_1) \cap \Pi_{V_{A_2}}(e_2) \notin \Pi_{V_{A_2}}(A_2)$. Therefore, we can derive the following:

$$\begin{aligned} &\Pi_{V_{A_2}}(e_1) \cap \Pi_{V_{A_2}}(e_2) \notin \Pi_{V_{A_2}}(A_2) \\ \Longrightarrow\ &\Pi_{V_{\mathcal{C}_s}}(e_1) \cap \Pi_{V_{\mathcal{C}_s}}(e_2) \subseteq \overline{A_2} \subseteq G_2 \end{aligned}$$

and

$$\begin{aligned} &e_2 \in \Pi_{V_{\mathcal{C}_1}}(G_1) \\ \Longrightarrow\ &\Pi_{V_{\mathcal{C}_s}}(e_2) \subseteq G_1 \\ \Longrightarrow\ &\Pi_{V_{\mathcal{C}_s}}(e_1) \cap \Pi_{V_{\mathcal{C}_s}}(e_2) \subseteq G_1. \end{aligned}$$

Similar to the proof for Lemma 1, we can show that $\Pi_{V_{\mathcal{C}_s}}(e_1) \cap \Pi_{V_{\mathcal{C}_s}}(e_2) \subseteq \overline{A_s}$:

$$\begin{aligned}\Pi_{V_{\mathcal{C}_s}}(e_1) \cap \Pi_{V_{\mathcal{C}_s}}(e_2) &\subseteq (\overline{A_2}) \cap (G_1 \cap G_2)\\ &\subseteq \overline{((A_1 \cap A_2) \cup \overline{(G_1 \cap G_2)})}\\ &\subseteq \overline{A_s}.\end{aligned}$$

Therefore, $\Pi_{V_{\mathcal{C}_s}}(e_1) \subseteq \overline{A_s}$, and thus $e_1 \notin \Pi_{V_{A_s}}(A_s)$ contradicts the assumption. □

With Lemmas 1 and 2, we conclude that any refinement to the receptive contracts ensures strong replaceability, as shown in Theorem 5:

Theorem 5. *Let $\mathcal{C}_s = (A_s, G_s)$, $\mathcal{C}_1 = (A_1, G_1)$, and $\mathcal{C}_2 = (A_2, G_2)$ be saturated receptive contracts such that $\mathcal{C}_s \succeq \mathcal{C}_1 \parallel \mathcal{C}_2$, and let $\mathcal{C}_1' = (A_1', G_1')$ and $\mathcal{C}_2' = (A_2', G_2')$ be saturated receptive contracts such that $\mathcal{C}_1 \succeq \mathcal{C}_1'$ and $\mathcal{C}_2 \succeq \mathcal{C}_2'$, then $\mathcal{C}_1' \parallel \mathcal{C}_2'$ strongly replaces $\mathcal{C}_s$.*

Proof. Using the proposition of contract refinement, $\mathcal{C}_s \succeq \mathcal{C}_1' \parallel \mathcal{C}_2'$. By Lemma 1, for every $e \in \Pi_{V_{A_s}}(A_s)$, $\Pi_{V_{A_1}}(e) \in \Pi_{V_{A_1}}(A_1')$. Since $\mathcal{C}_1'$ is a receptive contract, we can find e_2 such that $e_2 \in \Pi_{V_{\mathcal{C}_1}}(e) \cap \Pi_{V_{\mathcal{C}_1}}(G_1')$, and thus:

$$\Pi_{V_{\mathcal{C}_s}}(e_2) \in G_1'. \tag{6.3}$$

By Lemma 2, e_2 and e satisfies $\Pi_{V_{A_2}}(e) \cap \Pi_{V_{A_2}}(e_2) \in \Pi_{V_{A_2}}(A_2)$. Similarly, since $\mathcal{C}_2'$ is a receptive contract, we can find e_3 such that $e_3 \in \Pi_{V_{\mathcal{C}_2}}(\Pi_{V_{A_2}}(e) \cap \Pi_{V_{A_2}}(e_2)) \cap \Pi_{V_{\mathcal{C}_2}}(G_2')$, and thus:

$$\Pi_{V_{\mathcal{C}_s}}(e_3) \in G_2'. \tag{6.4}$$

Considering the behavior $\Pi_{V_{\mathcal{C}_s}}(e) \cap \Pi_{V_{\mathcal{C}_s}}(e_2) \cap \Pi_{V_{\mathcal{C}_s}}(e_3)$ and combining the results in (6.3) and (6.4), we get:

$$\begin{aligned}\Pi_{V_{\mathcal{C}_s}}(e) \cap \Pi_{V_{\mathcal{C}_s}}(e_2) \cap \Pi_{V_{\mathcal{C}_s}}(e_3) &\in G_1' \cap G_2'\\ &\subseteq G_s.\end{aligned}$$

As $\Pi_{V_{\mathcal{C}_s}}(e) \cap \Pi_{V_{\mathcal{C}_s}}(e_2) \cap \Pi_{V_{\mathcal{C}_s}}(e_3) \in \Pi_{V_{\mathcal{C}_s}}(e)$, the condition for strong replaceability is satisfied. Therefore, $\mathcal{C}_1' \parallel \mathcal{C}_2'$ strongly replaces $\mathcal{C}_s$. □

Finally, we show an example of a feedback composition using receptive contracts that contains only vacuous implementations after refinement:

Example 3. *Let $\mathcal{C}_s$ be the system contract, $\mathcal{C}_1$ and $\mathcal{C}_2$ be the subsystem contracts, and $\mathcal{C}_1'$ be the refined contract for $\mathcal{C}_1$:*

$$\begin{aligned}&\mathcal{C}_s = (True, y = \frac{x}{1-x}), V_s = \{x, y\},\\ &\mathcal{C}_1 = (True, (y = b+1) \vee (y = xb)), V_1 = \{x, y, b\},\\ &\mathcal{C}_2 = (True, b = y+1), V_2 = \{y, b\},\\ &\mathcal{C}_1' = (True, y = b+1), V_1{}' = \{x, y, b\}.\end{aligned}$$

The compositions $\mathcal{C}_1' \parallel \mathcal{C}_2$ and $\mathcal{C}_1 \parallel \mathcal{C}_2$ both refine $\mathcal{C}_1$. But $\mathcal{C}_1' \parallel \mathcal{C}_2 = (True, \emptyset)$, and thus the only implementation is a vacuous implementation $M_1' = \emptyset$, even though the refined contract is a receptive contract. ■

We believe that additional constraints are needed for feedback composition such that the strong replaceability of any composition is ensured. The constraints for the feedback composition will be material for future work.

Chapter 7

Discussion

In this chapter, we discuss the impacts of the discovery and proposed concept on the contract-based design process, and thus the need for the development of new algorithms and tools for supporting contract-based design.

Design Faults in Refinement

The vacuous implementation problem should be regarded as a type of design fault, which might be caused by the designer or problems in the automation tools to generate refined contracts not satisfying the replaceability relation. The replaceability relation is crucial for the refinement process to guarantee the compatibility of its subsystems and thus avoid vacuous implementations after system integration. Only verifying the refinement relation cannot capture this type of design fault. Therefore, existing contract-based design methodologies [26, 22] that propose using refinement in the design process, should include a stage for verifying the replaceability of the top-level specification. The transitive strong replaceability breaks down the problem of verifying the replaceability of the top-level specification into verifying the strong replaceability between each refinement step and thus can be applied in the independent design paradigm. If the design faults are not captured in this early stage of design, the vacuous implementation would result in huge costs and design time overhead.

Applying Receptive Contracts

In Chapter 4– 6, we have shown that receptive contracts guarantee strong replaceability in cascade composition and pure contract refinement. The theory indicates that using receptive contracts can further simplify the process of verifying the replaceability relation. As long as the system does not contain feedback composition, receptive subsystem contracts guarantee the replaceability of the refined systems to the top-level specifications. In many application fields, the specifications should be receptive by their definitions, such as controller design and sequential programs. The inputs and outputs are explicitly defined for every system in these fields. Therefore, verifying receptive contracts can serve two roles at the same time, one is verifying the design faults, and the other is maintaining the semantics of the components, as it is meaningless for a controller or a program method to have no outputs for any allowable inputs.

The Need for Development of New Algorithms and Tools

With the proposed theory, we suggest the development of new algorithms and tools to facilitate the contract-based design. Existing contract tools and algorithms [5, 11, 14] do not include the functionality to verify the replaceability relation, and thus are unable to detect the design faults of vacuous implementation. The universal quantification in the strong replaceability and receptiveness is challenging for algorithm development as its decidability depends on the representations of contracts. For example, the Presburger arithmetic is decidable while it becomes undecidable if multiplication is involved [19]. Therefore, research on tools and algorithms for different representations of contracts is required to prevent design faults and enable independent design using contracts.

Chapter 8

Conclusion

We identified the vacuous implementation problem using assume-guarantee contracts under the independent design paradigm. We first explored the notion of contract replaceability. This notion was shown to be the requirement to ensure strict implementations, but it is not transitive, thus limiting its applicability in independent design. The stricter notion of strong replaceability also ensures strict implementations and is transitive, thus fitting the independent design paradigm. We then proposed the notion of contract receptiveness, which guarantees strong replaceability. Moreover, we showed that receptive contracts can be implemented independently and that the composition of their implementations will not be vacuous in the case of cascade composition. A supplier receiving a contract as the specification for implementation can check whether this contract is receptive. If so, the supplier knows in advance that it can proceed to develop an implementation and that this implementation will integrate correctly into the system integrator's design. Our areas of future work include finding constraints for feedback composition, developing tools to support independent design, and investigating the replaceability in different contract formalisms.

Bibliography

[1] Martin Abadi and Leslie Lamport. "Composing specifications". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 15.1 (1993), pp. 73–132.

[2] Martín Abadi, Leslie Lamport, and Pierre Wolper. "Realizable and unrealizable specifications of reactive systems". In: *International Colloquium on Automata, Languages, and Programming.* 1989, pp. 1–17. ISBN: 978-3-540-46201-9.

[3] Albert Benveniste et al. "Contracts for system design". In: *Foundations and Trends® in Electronic Design Automation* 12.2-3 (2018), pp. 124–400.

[4] Albert Benveniste et al. "Multiple viewpoint contract-based specification and design". In: *International Symposium on Formal Methods for Components and Objects.* 2007, pp. 200–225.

[5] Alessandro Cimatti, Michele Dorigatti, and Stefano Tonetta. "OCRA: A tool for checking the refinement of temporal contracts". In: *IEEE/ACM International Conference on Automated Software Engineering (ASE).* 2013, pp. 702–705.

[6] Alessandro Cimatti and Stefano Tonetta. "A property-based proof system for contract-based design". In: *Euromicro Conference on Software Engineering and Advanced Applications.* 2012, pp. 21–28.

[7] Werner Damm et al. "Using contract-based component specifications for virtual integration testing and architecture design". In: *Design, Automation & Test in Europe Conference Exhibition (DATE).* 2011, pp. 1–6.

[8] Antonio Iannopollo. "A Platform-Based Approach to Verification and Synthesis of Linear Temporal Logic Specifications". PhD thesis. University of California, Berkeley, 2018.

[9] Antonio Iannopollo, Stavros Tripakis, and Alberto Sangiovanni-Vincentelli. "Constrained Synthesis from Component Libraries". In: *Formal Aspects of Component Software.* 2017, pp. 92–110. ISBN: 978-3-319-57666-4.

[10] Antonio Iannopollo, Stavros Tripakis, and Alberto Sangiovanni-Vincentelli. "Specification decomposition for synthesis from libraries of LTL Assume/Guarantee contracts". In: *Design, Automation & Test in Europe Conference Exhibition (DATE).* 2018, pp. 1574–1579.

[11] Antonio Iannopollo et al. “Library-based scalable refinement checking for contract-based design”. In: *Design, Automation & Test in Europe Conference Exhibition (DATE)*. IEEE. 2014, pp. 1–6.

[12] Inigo Incer. “The Algebra of Contracts”. PhD thesis. EECS Department, University of California, Berkeley, May 2022.

[13] Inigo Incer et al. “Hypercontracts”. In: *NASA Formal Methods*. 2022, pp. 674–692.

[14] Inigo Incer et al. “Pacti: Assume-Guarantee Contracts for Efficient Compositional Analysis and Design”. In: *ACM Trans. Cyber-Phys. Syst.* (Nov. 2024).

[15] Inigo Incer et al. “Quotient for assume-guarantee contracts”. In: *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. IEEE. 2018, pp. 1–11.

[16] C. B. Jones. “Tentative Steps toward a Development Method for Interfering Programs”. In: *ACM Transactions on Programming Languages and Systems* 5.4 (Oct. 1983), pp. 596–619. ISSN: 0164-0925.

[17] Thi Thieu Hoa Le et al. “Contract-Based Requirement Modularization via Synthesis of Correct Decompositions”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 15.2 (2016). ISSN: 1539-9087.

[18] Bertrand Meyer. “Applying ’design by contract’”. In: *Computer* 25.10 (1992), pp. 40–51.

[19] James Donald Monk. *Mathematical logic*. Vol. 37. Springer Science & Business Media, 2012.

[20] Pierluigi Nuzzo et al. “A contract-based methodology for aircraft electric power system design”. In: *IEEE Access* 2 (2013), pp. 1–25.

[21] Pierluigi Nuzzo et al. “Methodology for the Design of Analog Integrated Interfaces Using Contracts”. In: *IEEE Sensors Journal* 12.12 (2012), pp. 3329–3345.

[22] Pierluigi Nuzzo et al. “Stochastic assume-guarantee contracts for cyber-physical system design”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 18.1 (2019), pp. 1–26.

[23] Chanwook Oh et al. “Optimizing assume-guarantee contracts for cyber-physical system design”. In: *Design, Automation & Test in Europe Conference Exhibition (DATE)*. 2019, pp. 246–251.

[24] Roberto Passerone, Inigo Incer, and Alberto L Sangiovanni-Vincentelli. “Coherent Extension, Composition, and Merging Operators in Contract Models for System Design”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 18.5 (2019), pp. 1–23.

[25] Alberto Sangiovanni-Vincentelli. “Quo vadis, SLD? Reasoning about the trends and challenges of system level design”. In: *Proceedings of the IEEE* 95.3 (2007), pp. 467–506.

[26] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. “Taming Dr. Frankenstein: Contract-based design for cyber-physical systems”. In: *European journal of control* 18.3 (2012), pp. 217–238.

[27] Sheng-Jung Yu, Inigo Incer, and Alberto Sangiovanni-Vincentelli. “Contract Replaceability for Ensuring Independent Design using Assume-Guarantee Contracts”. In: *ACM/IEEE International Conference on Formal Methods and Models for System Design*. 2023, pp. 111–121.