Pacti: Assume-Guarantee Contracts for Efficient Compositional Analysis and Design

Inigo Incer, Apurva Badithela, Josefine B Graebener, Piergiuseppe Mallozzi, Ayush Pandey, Nicolas Rouquette, Sheng-Jung Yu, Albert Benveniste, Benoit Caillaud, Richard M Murray, et al.

Pacti: Assume-Guarantee Contracts for Efficient Compositional Analysis and Design

INIGO INCER, University of Michigan, Ann Arbor, USA
APURVA BADITHELA, Princeton University, USA
JOSEFINE B. GRAEBENER, California Institute of Technology, USA
PIERGIUSEPPE MALLOZZI, University of California, Berkeley, USA
AYUSH PANDEY, University of California, Merced, USA
NICOLAS ROUQUETTE, Jet Propulsion Laboratory, California Institute of Technology, USA
SHENG-JUNG YU, University of California, Berkeley, USA
ALBERT BENVENISTE and BENOIT CAILLAUD, INRIA/IRISA Rennes, France
RICHARD M. MURRAY, California Institute of Technology, USA
ALBERTO SANGIOVANNI-VINCENTELLI and SANJIT A. SESHIA, University of California, Berkeley, USA

Contract-based design is a method to facilitate modular design of systems. While there has been substantial progress on the theory of contracts, there has been less progress on practical algorithms for the algebraic operations in the theory. In this paper, we present 1) principles to implement a contract-based design tool at scale and 2) Pacti, a tool that can efficiently compute these operations. We illustrate the use of Pacti in a variety of case studies.

CCS Concepts: • **Theory of computation → Logic and verification**; **Abstraction**; • **Computer systems organization → Embedded and cyber-physical systems**.

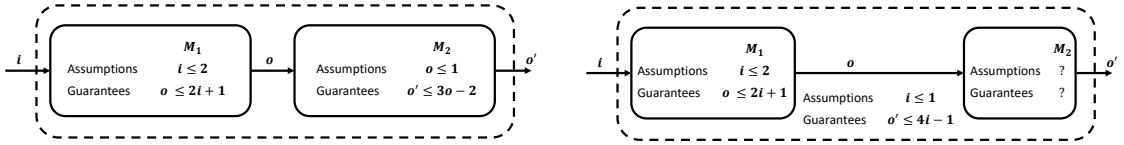
Additional Key Words and Phrases: assume-guarantee reasoning, contracts, system-level design, computational logic

1 INTRODUCTION

It has been more than fifteen years since contract-based design [3, 4] was proposed as a formal methodology to facilitate the compositional design of general cyber-physical systems. A fundamental idea is to specify cyber and physical components in a system using *assume-guarantee contracts* (AG contracts). Contracts enable two processes: *independent* design and *concurrent* design. By independent design, we mean that a set of contracts whose composition refines a top-level requirement is identified. These new contracts can be delivered to others in order to obtain an implementation, all while knowing in advance that the composition of these implementations

Authors' addresses: Inigo Incer, iir@umich.edu, University of Michigan, Ann Arbor, USA; Apurva Badithela, apurva.b@princeton.edu, Princeton University, USA; Josefine B. Graebener, jgraeben@caltech.edu, California Institute of Technology, USA; Piergiuseppe Mallozzi, mallozzi@berkeley.edu, University of California, Berkeley, USA; Ayush Pandey, ayushpandey@ucmerced.edu, University of California, Merced, USA; Nicolas Rouquette, nfr@jpl.nasa.gov, Jet Propulsion Laboratory, California Institute of Technology, USA; Sheng-Jung Yu, shengjungyu@berkeley.edu, University of California, Berkeley, USA; Albert Benveniste, albert.benveniste@inria.fr; Benoit Caillaud, benoit.caillaud@inria.fr, INRIA/IRISA Rennes, France; Richard M. Murray, murray@cds.caltech.edu, California Institute of Technology, USA; Alberto Sangiovanni-Vincentelli, alberto@berkeley.edu; Sanjit A. Seshia, sseshia@berkeley.edu, University of California, Berkeley, USA.

ACM 2378-9638/2024/11-ART
https://doi.org/10.1145/3704736



(a) (b)

Fig. 1. Two examples of computing assume-guarantee contracts: (a) Given the contracts for two components connected in series, we wish to compute the system-level contract. (b) Given a top-level contract and the contract for one of the subsystems, we wish to compute the contract of the missing component.

will meet the top-level objective. By concurrent design, we mean that the specification of a given subsystem can be broken into multiple specifications, each addressing a certain viewpoint (e.g., functionality or performance) of the design element in question. Analysis can then be carried out by using as much specification data as needed for the task at hand. A rich contract algebra [4, 23] has been developed to carry out operations of relevance for independent and concurrent system analysis and design.

While these algebraic operations satisfy certain optimality criteria, their results need to be post-processed before they are provided to the user or to subsequent computational steps. Consider the following example.

Running example. Figure 1a shows components M and M' connected in series. M assumes that its input i satisfies $i \leq 2$ and guarantees that its output o will satisfy $o \leq 2i + 1$. M' assumes that its input o satisfies $o \leq 1$ and guarantees that its output o' satisfies $o' \leq 3o - 2$.

An intuitive specification for the system obtained by interconnecting these two components is (assumptions: $i \leq 0$, guarantees: $o' \leq 6i + 1$). However, when we use the operation of contract composition, we obtain the system-level assumptions $(i \leq 2 \wedge o \leq 1) \vee (i \leq 2 \wedge \neg(o \leq 2i + 1)) \vee (o \leq 1 \wedge \neg(o' \leq 3o - 2))$ and the guarantees $(\neg(i \leq 2) \vee (o \leq 2i + 1)) \wedge (\neg(o \leq 1) \vee (o' \leq 3o - 2))$. ∎

We observe that the specification that results from the algebraic operation of composition has three drawbacks: 1) it is more difficult to understand than the original specifications, since now we have to reason about several statements in a disjunction, 2) it refers to variables that are not part of the interface of the system (i.e., variable o), and 3) it is considerably longer than the original specifications, possibly causing difficulties for further computation of specifications. Other contract operations suffer from the same complications. In this paper, we study mechanisms for efficiently computing contract specifications at the interfaces of systems and introduce Pacti, a tool that performs these contract computations. The techniques introduced in this paper and Pacti enable the analysis of systems using only the assume-guarantee specifications of the components. To illustrate these techniques, we will consider design problems across several domains (Section 7):

- In hardware implementations of signal processing applications, it is important to analyze the amount of rounding error in which we incur by translating the numerical representations of a signal processing algorithm from floating point to fixed point. Given the rounding-error specifications of several arithmetic blocks, we use contract composition to understand the total rounding error introduced by the fixed-point algorithm. We also use the quotient to identify the amount of rounding error that a component is allowed to add so that the entire algorithm meets a bound on its allowed error. This information is valuable to optimize the size of arithmetic blocks in the fixed-point implementation of the algorithm.
- Synthetic biology is increasingly applying component-based design. We consider the design of a genetic circuit with three components. Two of those components are fixed, and the other is chosen from a library containing several candidates. In order to understand what components from the library would yield a working circuit, we compose the existing component contracts with each of the contracts contained in the library. This composition filters out several library components that cannot be used to build a circuit. Components are

filtered out because their guarantees are insufficient to meet the assumptions of downstream components. Among those components that yield successful compositions, we pick the one that results in the best system-level performance, measured as the ratio of the steady-state output concentration level to the input level of the circuit.

- It can be challenging to identify specifications for perception components in autonomous vehicles. We consider the situation of an autonomous vehicle approaching a crosswalk. The vehicle is required to satisfy a safety property with a certain probability. We abstract the vehicle as a system that consists of three controllers and one perception component. We show how the contract quotient operation allows us to compute the optimal specification of the perception component from those of the vehicle and controllers. This specification can then be given to a perception expert for independent implementation.

Previous work. Several tools to support contract-based design have been developed. OCRA [10] supports refinement checking of temporal contracts; AGREE [11] uses assume-guarantee reasoning to hierarchically verify contract refinement using past-time linear temporal logic; CHASE [39] combines a front-end formal specification language with back-end requirement correctness, completeness, and refinement checking. The purpose of these tools is to verify whether a given contract decomposition is a valid refinement of a top-level specification; they do not return the result of contract operations to the user. CROME [35] allows engineers to refine and realize robotic-mission specifications using contracts and computes contract operations by concatenating LTL expressions, incurring in the drawbacks just discussed. As far as we know, there does not exist a tool that can return contract operations at the interfaces of the components being specified.

Contributions. This paper presents the theoretical foundations and design of Pacti, a requirement engineering tool that allows designers to efficiently manipulate assume-guarantee contracts while generating results at the interfaces of components. An important use case of specification-based analysis is the exploration of the design space of complex systems, a step in which designers consider the trade-offs between multiple architectures of a system. This motivated us to optimize the computational efficiency of Pacti's contract operations. Specifically, this paper makes the following contributions:

(1) The introduction of Pacti, a contract-based tool supporting requirement engineering, in which specifications consist of sets of requirements understood in a conjunctive sense. Pacti returns contracts in which both assumptions and guarantees are conjunctions of basic properties.
(2) The realization that contract specifications should only be given at the interfaces of components (Section 3). This means that, while AG contract operations meet algebraic optimality criteria, they are not necessarily what a tool should return to the user because they contain unnecessary detail, e.g., internal system variables. To the best of our knowledge, this remark and its implications have not been investigated in the contract literature.
(3) Efficient algorithms for the computation of contracts at the interfaces of components (Section 4), which increases the readability of specifications by human designers and reduces their complexity for further processing by tools. These algorithms have a clear separation between the implementation of the contract algebra and the formalism in which specifications are expressed. As far as we know, Pacti is the first tool that returns contracts whose assumptions and guarantees only refer to the interface variables of the results.
(4) The identification and solution of the context-aided variable elimination problem (Section 5), which is of relevance to requirement engineering. Our solution to the problem enables Pacti to return contracts in a user-friendly form.
(5) Efficient algorithms to solve context-aided variable elimination when contracts are expressed using polyhedral constraints (Section 6). We show that the algorithms we introduce are either polynomial-time or based on linear programming.

This paper is based on Chapter 7 of the PhD thesis [23] and preprints [24] and [25]. The structure of the paper is as follows: Section 2 provides a brief overview of assume-guarantee contracts and introduces a formalism to deal with syntactic representations of contracts. Section 3 introduces the three principles that allow us to compute contracts for complex systems and that form the basis of Pacti. In Section 4, we describe the algorithms used to efficiently compute the contract operations. Section 5 introduces the context-aided variable elimination problems and provides methods to solve it for general specification languages. Section 6 discusses how to support the algorithms introduced in Section 4 when contracts are expressed using polyhedral constraints. Section 7 discusses applications of Pacti to problems in digital circuit design, autonomous systems, and synthetic biology. Section 8 presents large-scale benchmarks of contract computations using Pacti. Of this content, only the paragraph on contract background of Section 2 is based on peer-reviewed material. Unless otherwise stated, the rest of that section and all subsequent material are contributions of this work.

2 ASSUME-GUARANTEE CONTRACTS

We provide here the standard definitions of assume-guarantee contracts and introduce a formalization for connecting contracts to syntactic representations.

Background on assume-guarantee contracts. We follow [4, 23] in our definitions. Let $\mathcal{B}$ be the universe of behaviors over all variables in our system, including cyber, physical, functional, and nonfunctional variables. The components over which we define predicates and the predicates themselves are defined as sets of behaviors. We define a component $M \subseteq \mathcal{B}$ as the collection of behaviors we can witness from it, whereas a property $P \subseteq \mathcal{B}$ is the set of behaviors meeting a given criterion, such as safety. We say a component M satisfies a property P, written $M \models P$, if $M \subseteq P$. Given two properties P and P', we say that P is a refinement of P' if for every component M, $M \models P \Rightarrow M \models P'$. Thus, P is a refinement of P' if $P \leq P'$, where $\leq$ is the subset order. Component composition is given by set intersection, i.e., for two components M, M', their composition is $M \parallel M' = M \cap M'$.

A contract is a pair of properties $C = (A, G)$, where A represents the *assumptions*, and G the *guarantees* of the contract. A component E is called an *environment* of the contract, denoted $E \models^E C$ if $E \models A$. A component M is an implementation of the contract, denoted $M \models^I C$, if $M \parallel E \models G$ for all $E \models^E C$. That is, a component is an implementation of a contract if it satisfies the contract's guarantees when operating in an environment that meets the contract's assumptions. We discuss the order and various operations of contracts.

(1) *Refinement*. Contracts are partially ordered [4]. Given contracts $C = (A, G)$ and $C' = (A', G')$, we say that C is a *refinement* of C' (or that C' is a relaxation of C), denoted $C \leq C'$, if any implementation of C is an implementation of C' and any environment of C' is an environment of C: $C \leq C' \Leftrightarrow (A' \leq A) \wedge (G \cup \neg A \leq G' \cup \neg A')$. This order generates a well-defined greatest-lower bound, called *conjunction*. It is given by $C \wedge C' = \big((A \cup A'), (G \cup \neg A) \cap (G' \cup \neg A')\big)$. Conjunction yields a contract that retains all information about the contracts being conjoined: the guarantees of the contracts being conjoined are required to hold when their corresponding assumptions hold. This operation is used to merge viewpoints that do not need to hold simultaneously.
(2) *Composition*. The operation of composition [4] allows us to obtain a specification for a system built by composing implementations of the contracts being composed. Its closed-form expression is $C \parallel C' = \big((A \cap A') \cup (A \cap \neg G) \cup (A' \cap \neg G'), (G \cup \neg A) \cap (G' \cup \neg A')\big)$.
(3) *Quotient*. The quotient [28] allows us to solve the following problem: given a top level specification C that we want a system to meet and given the specification C' of a partial implementation of the system, what is the specification of the component that we are missing to implement C? We can compute it as follows: $C/C' = (A \cap (\neg A' \cup G'), (A' \cap G) \cup \neg A \cup (A' \cap \neg G'))$.
(4) *Merging* (or strong merging) [42] can be used to handle multiple contract viewpoints that need to be enforced simultaneously. This operation yields a contract whose environments satisfy the assumptions

of both contracts and whose implementations are valid implementations for both contracts: $C \bullet C' = (A \cap A', (G \cap G') \cup \neg A \cup \neg A')$.

Syntax. The standard definitions of assume-guarantee contracts do not lend themselves to implementations. We need syntactic representations of contracts for this purpose. Here we establish the formal aspects of these representations.

Contracts are defined over a universe of behaviors, $\mathcal{B}$. We consider the construction of such a universe. The most fundamental concept in modeling is the variable. Variables are names associated with a concept in our system. Following the tagged signal model [33], we will define a variable as a tuple $(V, \mathcal{B}_V)$ of a name V and a set of behaviors $\mathcal{B}_V$ for that variable. For instance, a static variable with real values would have $\mathcal{B}_V = \mathbb{R}$, while a signal with discrete transitions and values taken in a domain D would have a universe $\mathbb{N} \to D$. Continuous functions are defined by changing the domain, e.g., $\mathbb{R}_{\geq 0} \to D$. Assume we have a set **VarSet** of variables in our system. Then we can build the universe of behaviors as $\mathcal{B} = \prod_{V \in \mathbf{VarSet}} \mathcal{B}_V$.

We assume we have access to a Boolean algebra T, called the *term algebra*, whose elements we will call *terms* or *constraints*. T will give us the syntax we will use to represent specifications on top of which contracts are built. This Boolean algebra comes equipped with a Boolean map $\mathbf{Den}\colon T \to 2^{\mathcal{B}}$ called the *denotation map*. The fact **Den** is a Boolean map means it commutes with the Boolean algebra structure of T. Also, the denotation map generates a preorder on T as follows: for $a, b \in T$, we say that $a \leq b$ if $\mathbf{Den}(a) \subseteq \mathbf{Den}(b)$. Given a term $a \in T$, we will use $\mathrm{FV}(a)$ to obtain the set of free variables that appear in a.

For $a, g \in T$, we can write contracts over T as $C = (a, g)$, where a and g are terms. Applying the denotation map element-wise enables us to connect contracts over T with contracts over $2^{\mathcal{B}}$: $\mathbf{Den}\colon (a, g) \mapsto (\mathbf{Den}(a), \mathbf{Den}(g))$. Observe that we can now express contracts over formulas of an arbitrary logic, such as LTL, polyhedral constraints, or representations of interacting state machines. We can compute the contract operations *in the term algebra* as follows: for contracts $C = (a, g)$ and $C' = (a', g')$ defined over the term algebra, using the operations stated in the background discussion, we have

$$C \leq C' \Leftrightarrow (a' \leq a) \wedge (g \vee \neg a \leq g' \vee \neg a') \tag{1}$$

$$C \parallel C' = ((a \wedge a') \vee (a \wedge \neg g) \vee (a' \wedge \neg g'), (g \vee \neg a) \wedge (g' \vee \neg a')) \tag{2}$$

$$C/C' = (a \wedge (\neg a' \vee g'), (a' \wedge g) \vee \neg a \vee (a' \wedge \neg g')) \tag{3}$$

$$C \bullet C' = (a \wedge a', (g \wedge g') \vee \neg a \vee \neg a') \tag{4}$$

3 COMPUTING CONTRACT OPERATIONS AT COMPONENT INTERFACES

The closed-form expressions for refinement (1), composition (2), quotient (3), and merging (4) over a term algebra T immediately suggest a difficulty: the results of these system construction/deconstruction operations are considerably more complex than the original specifications themselves. This brings problems to 1) the generation of specifications that a designer can readily understand and to 2) the further automated processing of specifications, as now the algorithms have to manipulate longer formulas.

Our objective is to identify means for reducing the complexity of the computed operations to make them more understandable to designers, and more succinct, so that contract operations can be applied repeatedly without the constraints exploding in length. We adopt three principles towards this end. In this section, we discuss these principles and present algorithms for the efficient computation of some contract operations.

Principle 1: post-processing contract operations. All contract operations are defined as a contract satisfying certain optimality criteria. For example, composition is defined as the smallest contract such that the composition of the implementations of the contracts being composed satisfies the guarantees of the contract and the environments of the contract satisfy other criteria (see [4], Chapter 4 or [23], Chapter 6). Similarly, given contracts C and C', the quotient is defined as the largest contract C'' such that $C' \parallel C'' \leq C$, i.e., the quotient is the largest specification

of a missing component that will allow a system to meet a top-level specification. The key observation is that *the fact that contract operations are optimal does not mean they should be the end result we should communicate to the user or keep for subsequent processing by tools.* For example, a system that obeys the composite specification will also obey a more relaxed contract. Similarly, any specification that refines the quotient will also be the specification of a missing component. We conclude that *if an operation is defined as a minimum (resp. maximum), then a relaxation (resp. refinement) of the operation can be returned to the user.* Thus, we will relax or refine the contract operations in order to place contracts in a more desirable form.

This more desirable form only refers to the variables at the interface of the specification being computed. Thus, we will be relaxing/refining specifications as needed in order to eliminate certain variables from a contract. This process raises the question of optimality of the approximation. Because the optimality of approximation while eliminating variables depends on the elimination algorithms developed for specific logical formalisms, we consider the issue as orthogonal to the contract algorithms in Pacti and outside the scope of this work. More details are given in Section 4.

The next two principles address the form that contracts should have and that will yield an algorithm for contract post-processing.

Principle 2: contracts as lists of requirements. Requirements in industry are often expressed as conjunctions of constraints. In general, component datasheets state a list of requirements that must hold simultaneously on the environment (e.g., bounded temperature, input voltages, etc.) in order for a list of promises to hold. We will call *termlist* the elements of 2^T. The denotation of termlists will be given by the composed map $2^T \xrightarrow{\wedge} T \xrightarrow{\mathbf{Den}} 2^{\mathcal{B}}$ which works as follows: $\mathfrak{t} \mapsto \mathbf{Den}(\wedge_{t\in\mathfrak{t}} t) = \bigcap_{t\in\mathfrak{t}} \mathbf{Den}(t)$. Therefore, *we will express contracts in the form* $(\mathfrak{a}, \mathfrak{g})$*, where* $\mathfrak{a}, \mathfrak{g} \in 2^T$*, indicating that several promises hold as long as several assumptions hold.*

Principle 3: IO contracts. Influenced by IO Automata [34], Interface Automata [15], and Moore Interfaces [8], we will extend the definition of a contract to be

Definition 3.1. Let T be a term algebra and **VarSet** be a set of variables. An IO contract is a list $(I, O, \mathfrak{a}, \mathfrak{g})$, where $I, O \subseteq$ **VarSet** are disjoint sets of input and output variables, respectively, and $\mathfrak{a}, \mathfrak{g} \in 2^T$ are lists of terms representing the assumptions and guarantees of the contracts, respectively. The assumptions of IO contracts only depend on input variables, and the guarantees only depend on input and output variables. In other words, $\mathrm{FV}(\wedge\mathfrak{a}) \subseteq I$ and $\mathrm{FV}(\wedge\mathfrak{g}) \subseteq I \cup O$.

Next, we will investigate ways to compute for IO contracts the refinement relation and the operations of composition, quotient, and merging. The IO profile of a contract will play a key role when computing the contract operations, as discussed below.

4 IMPLEMENTING THE CONTRACT OPERATIONS

Our objective is to devise algorithms to compute the contract operations taking as inputs IO contracts and producing as outputs IO contracts. We start with composition.

Contract composition. Consider two IO contracts $C = (I, O, \mathfrak{a}, \mathfrak{g})$ and $C' = (I', O', \mathfrak{a}', \mathfrak{g}')$. Their composition will only be defined when O and O' are disjoint. When this happens, we have $C \parallel C' = (I_c, O_c, \mathfrak{a}_c, \mathfrak{g}_c)$, where $I_c = (I \cup I') \setminus (O \cup O')$ and $O_c = (O \cup O') \setminus (I \cup I')$. This operation will not keep in the composed contract's IO profile any output which serves as an input of the other contract being composed; this means that, in general, composition of IO contracts is not associative.[1]

$\mathfrak{a}_c$ and $\mathfrak{g}_c$ are computed as follows. Let $a = \wedge\mathfrak{a}$, $g = \wedge\mathfrak{g}$, $a' = \wedge\mathfrak{a}'$, and $g' = \wedge\mathfrak{g}'$. We form the T-contract (a_c, g_c) for $a_c, g_c \in T$, where the contract (a_c, g_c) is given by the standard contract composition (2). Since C and C' are IO

[1]Our implementation allows the user to specify which of the output signals should be kept in the composed contract.

contracts, a and g have terms depending on I and $I \cup O$, respectively, and similarly for a' and g'. This means that a_c and g_c may fail to produce an IO contract because the assumptions and guarantees may depend on variables other than those allowed (i.e., I_c for the assumptions and $I_c \cup O_c$ for the guarantees). In order to produce an IO contract after composition, we apply Principle 1. Since we know that composition is defined as a minimum, we can relax the operation (2) in order to obtain a well-defined IO contract. How should such a relaxation be computed? Per (1), to relax the contract means to refine the assumptions and loosen the guarantees. We observe that the assumptions of the composition (2) have three terms: $a \wedge a'$, $a \wedge \neg g$, and $a' \wedge \neg g'$. We refer to the first term as the *stem* of the assumptions since this term represents the simultaneous enforcement of the constraints of a and a' and thus means that we can expect the guarantees of both contracts to hold. The terms $a \wedge \neg g$ and $a' \wedge \neg g'$ are *failure terms*, as they state that the assumptions of a contract were met, but the component did not deliver its promises. These terms are used to carry out transformations in the stem, as shown in the theorem below, but we can safely remove them after we have used them. *We transform the stem in order to remove from it references to variables that do not belong to I_c.*

Theorem 4.1. *Let (a, g) and (a', g') be contracts defined over a term algebra T. Let $C_c = (a_c, g_c)$ be the composition, as computed by* (2)*. Suppose that $a'' \in T$ satisfies $a'' \wedge g \wedge a \leq a' \wedge g \wedge a$ and $g'' \in T$ satisfies $g \wedge g' \leq g''$. Then the contract $(a \wedge a'', g'')$ is a relaxation of C_c.*

Proof. We compute $a \wedge a'' \leq (a \wedge a'') \vee (a \wedge \neg g) \vee (a' \vee \neg g') = (a \wedge a'' \wedge g) \vee (a \wedge \neg g) \vee (a' \vee \neg g') \leq (a \wedge a' \wedge g) \vee (a \wedge \neg g) \vee (a' \vee \neg g') = (a \wedge a') \vee (a \wedge \neg g) \vee (a' \vee \neg g')$. Thus, $a \wedge a'' \leq a_c$. We also have $g'' \vee \neg(a \wedge a'') \geq (g \wedge g') \vee \neg a \vee \neg a'' \geq (g \wedge g') \vee \neg a \vee (a \wedge g \wedge \neg a'') \geq (g \wedge g') \vee \neg a \vee (a \wedge g \wedge \neg a') = (g \wedge g') \vee \neg a \vee (g \wedge \neg a') \geq (g \wedge g') \vee (g' \wedge \neg a) \vee (\neg a \wedge \neg a') \vee (g \wedge \neg a') = g_c \vee \neg a_c$. We conclude that the contract $(a \wedge a'', g'')$ is a relaxation of C_c. □

If we have IO contracts $C = (I, O, \mathfrak{a}, \mathfrak{g})$ and $C' = (I', O', \mathfrak{a}', \mathfrak{g}')$ and their composition is defined (the sets of output variables are disjoint), we use Theorem 4.1 to compute their composition $(I_c, O_c, \mathfrak{a} \cup \mathfrak{a}'', \mathfrak{g}'')$. This means we have to identify termlists $\mathfrak{a}''$ and $\mathfrak{g}''$ such that $(\wedge\mathfrak{a}'') \wedge (\wedge\mathfrak{g}) \wedge (\wedge\mathfrak{a}) \leq (\wedge\mathfrak{a}') \wedge (\wedge\mathfrak{g}) \wedge (\wedge\mathfrak{a})$ and $\wedge\mathfrak{g}'' \geq (\wedge\mathfrak{g}) \wedge (\wedge\mathfrak{g}')$. Using proof-theoretic notation, we can consider the sets of constraints $\mathfrak{g} \cup \mathfrak{a}$ as a context for the following inference: $\mathfrak{g}, \mathfrak{a}, \mathfrak{a}'' \vdash \mathfrak{a}'$; i.e., we use the context $\mathfrak{g} \cup \mathfrak{a}$ in order to refine the terms $\mathfrak{a}''$ from $\mathfrak{a}'$. Similarly, we use the terms $\mathfrak{a} \cup \mathfrak{a}''$ to relax $\mathfrak{g}''$ from $\mathfrak{g} \cup \mathfrak{g}'$. Observe that Theorem 4.1 only allows us to refine $\mathfrak{a}'$ using the context $\mathfrak{g}$ or $\mathfrak{a}$ using the context $\mathfrak{g}'$. We pick the context based on the interconnection of the contracts. If a component drives the inputs of another component, and the assumptions of the second component depend on this driven input, we use the former's guarantees as a context to refine the latter's assumptions. From our considerations so far, we will not allow both components to have outputs driving each other's inputs when the assumptions of both components depend on those inputs. Procedure ContractComposition (Algorithm 1) shows how we compute IO-contract composition.

Algorithm 1 Contract composition

Input: IO contracts $C = (I, O, \mathfrak{a}, \mathfrak{g})$ and $C' = (I', O', \mathfrak{a}', \mathfrak{g}')$
Output: Relaxation of the composition $C \parallel C'$

1: **procedure** ContractComposition$(I, O, \mathfrak{a}, \mathfrak{g}, I', O', \mathfrak{a}', \mathfrak{g}')$ **do**
2: $O^c \leftarrow (O \cup O') \setminus (I \cup I')$
3: $I^c \leftarrow (I \cup I') \setminus (O \cup O')$
4: $I_c, I_c' \leftarrow \text{Vars}(\mathfrak{a}), \text{Vars}(\mathfrak{a}')$ ▷ Constrained inputs
5: CyclePresent $\leftarrow (O' \cap I \neq \emptyset$ **and** $O \cap I' \neq \emptyset)$
6: **if** $O \cap O' \neq \emptyset$ **or** (CyclePresent **and** $(O' \cap I_c \neq \emptyset$ **or** $O \cap I_c' \neq \emptyset))$ **then**
7: **return** Error: Contracts are not composable

8: **else if** $O' \cap I \neq \emptyset$ and $O \cap I' = \emptyset$ **then**
▷ Refining $\mathfrak{a}$ using $\mathfrak{g}'$ and $\mathfrak{a}'$
9: $\bar{\mathfrak{a}} \leftarrow$ ElimVarsByRefining$(\mathfrak{a}, \mathfrak{a}' \cup \mathfrak{g}', I^c)$
10: $\mathfrak{a}^c \leftarrow$ Reduce$(\bar{\mathfrak{a}} \cup \mathfrak{a}', \emptyset)$
11: **else if** $O' \cap I = \emptyset$ and $O \cap I' \neq \emptyset$ **then**
▷ Refining $\mathfrak{a}'$ using $\mathfrak{g}$ and $\mathfrak{a}$
12: $\bar{\mathfrak{a}}' \leftarrow$ ElimVarsByRefining$(\mathfrak{a}', \mathfrak{a} \cup \mathfrak{g}, I^c)$
13: $\mathfrak{a}^c \leftarrow$ Reduce$(\bar{\mathfrak{a}}' \cup \mathfrak{a}, \emptyset)$
14: **else if** $(O' \cap I_c = \emptyset$ **and** $O \cap I'_c = \emptyset)$ **or** CyclePresent **then**
15: $\mathfrak{a}^c \leftarrow$ Reduce$(\mathfrak{a} \cup \mathfrak{a}', \emptyset)$
▷ Find $\mathfrak{g}^c$ such that $(\wedge\mathfrak{g}^c) \wedge (\wedge\mathfrak{a}^c) \geq (\wedge\mathfrak{g}) \wedge (\wedge\mathfrak{g}') \wedge (\wedge\mathfrak{a}^c)$
16: $\mathfrak{g}^c \leftarrow$ ElimVarsByRelaxing$(\mathfrak{g} \cup \mathfrak{g}', \mathfrak{a}^c, I^c \cup O^c)$
17: **return** $(I^c, O^c, \mathfrak{a}^c,$ Reduce$(\mathfrak{g}^c, \mathfrak{a}^c))$

ElimVarsByRefining$(\mathfrak{t}, \mathfrak{t}', S)$ and ElimVarsByRelaxing$(\mathfrak{t}, \mathfrak{t}', S)$ eliminate from $\mathfrak{t}$ any variable not appearing in S by refining or relaxing, respectively, these termlists in the context $\mathfrak{t}'$. The function Reduce$(\mathfrak{t}, \mathfrak{t}')$ is used to eliminate from $\mathfrak{t}$ any redundant constraints, using the context $\mathfrak{t}'$. The function IsRefinement$(\mathfrak{t}, \mathfrak{t}')$ tells whether the satisfaction of $\mathfrak{t}$ implies the satisfaction of $\mathfrak{t}'$. The implementations of these four functions depend on the specification theory in which the terms are expressed.

In the routines ElimVarsByRefining and ElimVarsByRelaxing, we observe that we do not use variable elimination as a strategy to refine or relax specifications; on the contrary, we refine and relax specifications only because we must eliminate certain variables from them, as discussed in Principle 1 of Section 3. Each specification formalism supported in Pacti must provide these procedures. We discuss these routines in Section 5.

Finally, we stated that Algorithm 1 outputs a relaxation of the algebraic composition (2). A question remains: how much is this output relaxed with respect to the algebraic operation? Under the assumption that the routines ElimVarsByRefining and ElimVarsByRelaxing produce optimal results, a discussion of which is given in Sections 5 and 6, the formula for the assumptions of the returned contract is the most refined formula satisfying the following requirements: *i*) its only free variables are top-level inputs of the system formed by interconnecting components with the IO profiles of the contracts being composed, and *ii*) the assertion of this formula guarantees that the assumptions of the contracts being composed will hold simultaneously when using the guarantees of the contracts being composed as context. Under the same assumption, the formula for the guarantees of the returned contract is the most relaxed formula satisfying the following: *i*) its only free variables are top-level inputs and outputs of the system formed by interconnecting components with the IO profiles of the contracts being composed, and *ii*) the assertion of this formula implies that of the guarantees of the contracts being composed in the context of the assumptions of the returned contract.

The following example shows how the routine ContractComposition computes the composition of two contracts:

Running example. In the circuit of Figure 1a, we have components M and M' obeying IO contracts $C = (\{i\}, \{o\}, \{i \leq 2\}, \{o \leq 2i + 1\})$ and $C' = (\{o\}, \{o'\}, \{o \leq 1\}, \{o' \leq 3o - 2\})$, respectively. We use ContractCompose (Algorithm 1) to obtain the system-level contract $(I^c, O^c, \mathfrak{a}^c, \mathfrak{g}^c)$. L2-L3 yield $I^c = \{i\}$ and $O^c = \{o'\}$. Condition L11 is active since M drives its outputs to the inputs of M' (and not vice versa). L12 yields $\bar{\mathfrak{a}}' =$ ElimVarsByRefining$(\{o \leq 1\}, \{i \leq 2, o \leq 2i + 1\}, \{i\}) = \{i \leq 0\}$, and from L13, we obtain $\mathfrak{a}^c = \{i \leq 0\}$. Finally, from L16, we get $\mathfrak{g}^c =$ ElimVarsByRelaxing$(\{o \leq 2i + 1, o' \leq 3o - 2\}, \{i \leq 0\}, \{i, o'\}) = \{o' \leq 6i + 1\}$. The resulting specification uses exclusively the inputs and outputs of the top-level system, as the routine outputs the contract $(\{i\}, \{o'\}, \{i \leq 0\}, \{o' \leq 6i + 1\})$. ∎

Contract quotient. The computation of the IO-contract quotient follows similar reasoning. Given IO contracts $C = (I, O, \mathfrak{a}, \mathfrak{g})$ and $C' = (I', O', \mathfrak{a}', \mathfrak{g}')$, we want to compute $C/C' = (I_q, O_q, \mathfrak{a}_q, \mathfrak{g}_q)$ applying (3). First, the quotient is defined only if I and O' are disjoint, as the outputs of O' cannot be inputs of the top-level. The inputs and outputs of the quotient are $I_q = (O' \setminus O) \cup (I \setminus I')$ and $O_q = (O \setminus O') \cup (I' \setminus I)$, respectively. Setting once again $a = \wedge\mathfrak{a}$, $g = \wedge\mathfrak{g}$, $a' = \wedge\mathfrak{a}'$, and $g' = \wedge\mathfrak{g}'$, the quotient assumptions and guarantees are given by (3). As with composition, this expression is not a valid IO contract. As the quotient is defined as a maximum, we will refine the quotient operation in order to transform it into a valid IO contract.

The assumptions of the quotient are $a \wedge (\neg a' \vee g')$. Refining the quotient means to enlarge the assumptions—see (1). In order to respect the IO contract structure, we keep the assumptions a, and we add to them the guarantees g' if $a \leq a'$. From the resulting list we remove terms containing irrelevant variables (i.e., those not in I_q), thus generating a relaxation of the assumptions. The guarantees of the quotient are $(a' \wedge g) \vee \neg a \vee (a' \wedge \neg g')$, an expression we refine using the following result:

Theorem 4.2. *Let (a, g) and (a', g') be contracts defined over a term algebra T. Let $C_q = (a_q, g_q)$ be the quotient $(a, g)/(a', g')$, as computed by* (3)*. Suppose that $a'' \in T$ satisfies $a_q \leq a''$ and $g'', g''' \in T$ satisfy $a' \wedge g' \wedge g'' \leq a' \wedge g' \wedge g$ and $g''' \wedge a \leq a' \wedge g'' \wedge a$. Then the contract (a'', g''') is a refinement of C_q.*

Proof. Since we are given $a_q \leq a''$, we just have to verify the guarantees: $g''' \vee \neg a'' \leq g''' \vee \neg a_q = g''' \vee \neg a \vee (a' \wedge \neg g') = (g''' \wedge a) \vee \neg a \vee (a' \wedge \neg g') \leq (a' \wedge g'' \wedge a) \vee \neg a \vee (a' \wedge \neg g') = (a' \wedge g'' \wedge g') \vee \neg a \vee (a' \wedge \neg g') \leq (a' \wedge g' \wedge g) \vee \neg a \vee (a' \wedge \neg g') = g_q \vee \neg a_q$. We conclude that $(a'', g''') \leq C_q$. □

This theorem tells us how to obtain a correct refinement of the quotient. Procedure ContractQuotient (Algorithm 2) shows how we compute the quotient of IO contracts using Theorem 4.2 and the considerations above.

Algorithm 2 Contract quotient

Input: IO contracts $C = (I, O, \mathfrak{a}, \mathfrak{g})$ and $C' = (I', O', \mathfrak{a}', \mathfrak{g}')$
Output: Refinement of the quotient C/C'

1: **procedure** ContractQuotient($I, O, \mathfrak{a}, \mathfrak{g}, I', O', \mathfrak{a}', \mathfrak{g}'$) **do**
2: **if** $I \cap O' \neq \emptyset$ **then**
3: **return** Error: The quotient is not defined
4: $O^q = (O \setminus O') \cup (I' \setminus I)$
5: $I^q = (O' \setminus O) \cup (I \setminus I')$
6: $\mathfrak{a}'' \leftarrow \mathfrak{a}$
7: **if** IsRefinement($\mathfrak{a}, \mathfrak{a}'$) **then**
8: $\mathfrak{a}'' \leftarrow$ Reduce($\mathfrak{a}'' \cup \mathfrak{g}', \emptyset$)
9: $\mathfrak{a}'' \leftarrow$ ElimVarsByRelaxing($\mathfrak{a}'', \emptyset, I^q$)
10: $\mathfrak{g}'' \leftarrow$ ElimVarsByRefining($\mathfrak{g}, \mathfrak{a}' \cup \mathfrak{g}', I^q \cup O^q$)
11: $\mathfrak{g}''' \leftarrow$ ElimVarsByRefining($\mathfrak{a}' \cup \mathfrak{g}'', \mathfrak{a}, I^q \cup O^q$)
12: **return** $(I^q, O^q, \mathfrak{a}'',$ Reduce$(\mathfrak{g}''', \mathfrak{a}''))$

The following example shows how ContractQuotient computes the quotient of two contracts.

Running example. In Figure 1b, we want to implement a system M with contract $(i, o', \{i \leq 1\}, \{o' \leq 4i - 1\})$ using a partial implementation M' with contract $(i, o, \{i \leq 2\}, \{o \leq 2i + 1\})$. We use ContractQuotient (Algorithm 2) to find the specification $(I^q, O^q, \mathfrak{a}'', \mathfrak{g}''')$ of the missing component. From L4-L5, we obtain $I^q = \{o\}$ and $O^q = \{o'\}$. L6-L9 result in $\mathfrak{a}'' =$ ElimVarsByRelaxing$(\{i \leq 1, o \leq 2i + 1\}, \emptyset, \{o\}) = \{o \leq 3\}$. L10 yields $\mathfrak{g}'' =$ ElimVarsByRefining$(\{o' \leq 4i - 1\}, \{i \leq 2, o \leq 2i + 1\}, \{o, o'\}) = \{o' \leq 2o - 3\}$. From L11, we get

$\mathfrak{g}''' = \text{RefineWithContext}(\{i \leq 2, o' \leq 2o-3\}, \{i \leq 1\}, \{o, o'\}) = \{o' \leq 2o-3\}$. We obtain a specification only depending on the missing component's inputs and outputs, as the routine outputs the contract $(\{o\}, \{o'\}, \{o \leq 3\}, \{o' \leq 2o-3\})$. ∎

Contract merging. The computation of strong merging (4) does not need refinements/relaxations and is given by the ContractMerging procedure that follows:

Algorithm 3 Contract merging

Input: IO contracts $C = (I, O, \mathfrak{a}, \mathfrak{g})$ and $C = (I', O', \mathfrak{a}', \mathfrak{g}')$
Output: Strong merging $C \bullet C'$

```
1: procedure ContractMerging(I, O, 𝔞, 𝔤, I′, O′, 𝔞′, 𝔤′) do
2:     if (I ≠ I′) or (O ≠ O′) then
3:         return Error: Merging is not defined
4:     a^m ← Reduce(𝔞 ∪ 𝔞′, ∅)
5:     return (I, O, a^m, Reduce(𝔤 ∪ 𝔤′, a^m))
```

Contract refinement. We consider contract refinement. We observe that a cumbersome issue with refinement is that (1) requires the computation of complements. Complements would require us to carry out expensive expansions of the termlists. The following proposition allows us to compute refinement without taking complements.

Proposition 4.3. *Let $C = (a, g)$ and $C' = (a', g')$ be contracts over the term algebra T. Then $C \leq C'$ if and only if $a' \leq a$ and $g \wedge a' \leq g' \wedge a'$.*

Proof. Suppose that $g \wedge a' \leq g' \wedge a'$. Then $g \vee \neg a' \leq g' \vee \neg a'$. Since $\neg a \leq \neg a'$, we have $g \vee \neg a \leq g' \vee \neg a'$. Conversely, suppose that $g \vee \neg a \leq g' \vee \neg a'$. Since $\neg a \leq \neg a'$, $g \vee \neg a' \leq g' \vee \neg a'$. Conjoining both sides with a yields $g \wedge a' \leq g' \wedge a'$. □

Verifying contract implementations. Finally, supporting the verification that a component M is a valid implementation of a contract $C = (a, g)$ requires the adoption of a representation for M. Suppose, for example, that the term algebra T over which a and g are defined is LTL; then M could be represented as a discrete transition system. If the term algebra were STL, M could be represented as a differential equation. In the first version of Pacti, we assume that components are abstracted into the term algebra T. Thus, $M \in T$, and the verification that M is an implementation of C means that $M \wedge a \leq g$.

5 CONTEXT-AIDED VARIABLE ELIMINATION FOR FIRST-ORDER LOGIC

Section 4 presented algorithms to compute contract operations whose return values are contracts having assumptions and guarantees only involving variables that appear at their interfaces. ElimVarsByRefining and ElimVarsByRelaxing are the key routines used in Algorithms 1 and 2 to make sure that only certain variables appear in the results of the contract operations. The purpose of these routines is to eliminate a given set of free variables Y from a given termlist ϕ by refining or relaxing, respectively, this termlist ϕ in the context of another termlist Γ.

In this section, we discuss the theory behind these routines. We will assume that ϕ and Γ are formulas instead of termlists; this is accomplished by taking the conjunction over the terms of a termlist, as discussed in Section 4. While the results of this section apply to general logical formalisms, in Section 6 we will discuss how to compute these routines when the terms are polyhedral constraints.

We state formally the problem that ElimVarsByRefining and ElimVarsByRelaxing are meant to solve. Given two formulas, α and β, we write $\alpha \models \beta$ to state that $\alpha \rightarrow \beta$ is a tautology. Let ϕ be a formula containing some variables that must be eliminated. In our application, these will be the variables that do not appear at the interface

of an object under specification. These variables to be eliminated will be called *irrelevant variables*, and the set of such variables will be denoted Y. In order to carry out the elimination, suppose we can use information from a set of formulas Γ called the *context*.

In this section, we will consider the problems of synthesizing missing formulas in the expressions

$$\Gamma \wedge \ ? \models \phi \quad \text{and} \quad \Gamma \wedge \phi \models \ ?$$

such that the result lacks irrelevant variables.

We will call the first problem *antecedent synthesis*, and the second *consequent synthesis*. If ψ is a solution to the antecedent synthesis problem, we will say that ψ is a Y-antecedent (or a Y-refinement) of ϕ in the context Γ. If ρ is a solution to the consequent synthesis problem, we will say that ρ is a Y-consequent (or a Y-relaxation) of ϕ in the context Γ. The fact that ψ and ρ lack irrelevant variables means that Y is disjoint from both $\mathrm{FV}(\psi)$ and $\mathrm{FV}(\rho)$.

The routine ElimVarsByRefining is meant to provide computational support for Y-antecedent synthesis, and ElimVarsByRelaxing for Y-consequent synthesis.

We first consider these problems for general first-order formulas and then will specialize to the situation when formulas are expressed as linear constraints in a context of linear inequalities. We provide efficient algorithms to address this problem.

5.1 Computing antecedents and consequents in first-order logic

Suppose ϕ is a formula in first-order logic with free variables $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$, and Γ a formula with free variables x, y, and $z = (z_1, \ldots, z_o)$. Thus, x is a list of m variables, y is a list of n variables, and z is a list of o variables. As a matter of notation, when the free variables of a formula are understood, we will simply write the name of the formula, e.g., $\phi(x, y)$ is synonymous with ϕ. We let Y be the set of *irrelevant variables* that we want to eliminate from ϕ. Throughout this section, the set of irrelevant variables is always $Y = \{y_i\}_i$, i.e., we are always interested in eliminating the y variables from ϕ.

Definition 5.1. We say that a formula $\psi(x, z)$ with free variables $x = (x_1, \ldots, x_m)$ and $z = (z_1, \ldots, z_o)$ is a Y-antecedent of ϕ in the context Γ if

$$\Gamma(x, y, z) \wedge \psi(x, z) \models \phi(x, y).$$

We say that $\rho(x, z)$ with free variables x and z is a Y-consequent of ϕ in the context Γ if

$$\Gamma(x, y, z) \wedge \phi(x, y) \models \rho(x, z).$$

Our objective is to eliminate the irrelevant variables y from ϕ by synthesizing a Y-antecedent or Y-consequent of ϕ in the context Γ.

This section contains the following results:

(1) a characterization of the optimal solutions for Y-antecedent/consequent synthesis in general (Proposition 5.3);
(2) a characterization of the optimal solution to this problem when the context can be expressed as a conjunction of a formula that depends on irrelevant variables and a formula that does not (Proposition 5.7); and
(3) a characterization of optimal solutions to this problem when ϕ monotonically depends on a function of y and does not depend on y in any other way (Proposition 5.9).

We now make use of ϕ, Γ, and Y to define two new formulas. We will then show that these formulas play a key role in the problems of antecedent and consequent synthesis in a context.

Definition 5.2. Given ϕ, Γ, and $Y = \{y_i\}_{i=1}^n$ as above, we define the formulas

$$\phi_\Gamma : \forall y_1, \ldots, y_n.\ (\Gamma \to \phi) \quad \text{and}$$
$$\phi^\Gamma : \exists y_1, \ldots, y_n.\ (\Gamma \wedge \phi)\,.$$

To simplify notation, we will often use $\forall y$ and $\exists y$ for the quantifications above. The following result shows the relevance of these definitions to the solutions of our problems.

Proposition 5.3. *A formula $\psi(x,z)$ is a Y-antecedent for ϕ in the context Γ if and only if $\psi \models \phi_\Gamma$. A formula $\rho(x,z)$ is a Y-consequent for ϕ in the context Γ if and only if $\phi^\Gamma \models \rho$.*

Proof. We have $\Gamma \wedge \psi \models \phi$ iff $\psi \models (\Gamma \to \phi)$ iff $\psi \models \phi_\Gamma$. Similarly, $\Gamma \wedge \phi \models \rho$ iff $\exists y.\ (\Gamma \wedge \phi) \models \rho$. □

This result means that ϕ_Γ is the weakest Y-antecedent of ϕ in the context Γ. Conversely, ϕ^Γ is the strongest Y-consequent.

Lemma 5.4. *The denotations of ϕ_Γ and ϕ^Γ are*

$$\mathbf{Den}\,(\phi_\Gamma) = \bigcap_{\substack{b \in \mathrm{dom}(y) \\ \Gamma(x,b,z)}} \mathbf{Den}\,(\phi(x,b)) \ \textit{and}\ \mathbf{Den}\left(\phi^\Gamma\right) = \bigcup_{\substack{b \in \mathrm{dom}(y) \\ \Gamma(x,b,z)}} \mathbf{Den}\,(\phi(x,b))\,.$$

Proof. We compute

$$\mathbf{Den}\,(\phi_\Gamma) = \mathbf{Den}\,(\forall y\ (\Gamma \to \phi)) = \bigcap_{b \in \mathrm{dom}(y)} \mathbf{Den}\,(\Gamma(x,b,z) \to \phi(x,b)) = \bigcap_{\substack{b \in \mathrm{dom}(y) \\ \Gamma(x,b,z)}} \mathbf{Den}\,(\phi(x,b))\,.$$

A similar reasoning applies to ϕ^Γ. □

Proposition 5.3 provides a universal characterization of Y-antecedents and consequents. The following example, however, shows that the optimal Y-antecedents and consequents may not necessarily be the results we want to return to users, as they may contain redundant information.

Example 5.5. Suppose that ϕ and Γ are formulas in linear arithmetic such that $\phi(x,y) = (x \le y)$ and $\Gamma(x,y,z) = \Gamma_1(x,z) \wedge \Gamma_2(x,y,z)$, where $\Gamma_1 = (z \le 2)$ and $\Gamma_2 = (z \le y)$. From Proposition 5.3, the optimal Y-antecedent of ϕ in Γ is $\phi_\Gamma = \forall y.\ ((z \le 2) \wedge (z \le y) \to (x \le y)) = ((z \le 2) \to (x \le z))$. ■

In the answer returned in Example 5.5, $z \le 2$ is redundant because this constraint is already guaranteed by the context Γ. We introduce a definition of optimality that will allows us to trim syntactic redundancies from results.

Definition 5.6. A formula $\psi(x,z)$ is an optimal Y-antecedent of ϕ in the context Γ if ψ is a Y-antecedent of ϕ in Γ and $\phi_\Gamma \wedge \Gamma \models \psi$. Similarly, a formula $\rho(x,z)$ is an optimal Y-consequent of ϕ in the context Γ if ρ is a Y-consequent of ϕ in Γ and $\rho \wedge \Gamma \models \phi^\Gamma$.

Thus, to say that ψ is an optimal Y-antecedent of ϕ in the context Γ means that $\psi \wedge \Gamma$ and $\phi_\Gamma \wedge \Gamma$ are semantically equivalent. A similar statement applies to optimal Y-consequents. The following result allows us to find optimal Y-antecedents and consequents using only partial information from the context.

Proposition 5.7. *Suppose that the context Γ can be written as $\Gamma(x,y,z) = \Gamma_1(x,z) \wedge \Gamma_2(x,y,z)$. Then $\phi_\Gamma = \Gamma_1 \to \phi_{\Gamma_2}$ and $\phi^\Gamma = \Gamma_1 \wedge \phi^{\Gamma_2}$. Moreover, ϕ_{Γ_2} and ϕ_{Γ^2} are optimal Y-antecedents and consequents, respectively, of ϕ in Γ.*

Proof. Regarding the first part of the statement, from Definition 5.1, we have

$$\begin{aligned}\phi_\Gamma &= \forall y.\ ((\Gamma_1 \wedge \Gamma_2) \to \phi) = \forall y.\ (\Gamma_1(x,z) \to (\Gamma_2(x,y,z) \to \phi(x,y))) \\ &= \Gamma_1 \to (\forall y.\ (\Gamma_2(x,y,z) \to \phi(x,y))) = \Gamma_1 \to \phi_{\Gamma_2}\end{aligned}$$

and

$$\phi^{\Gamma} = \exists y.\ ((\Gamma_1(x,z) \wedge \Gamma_2(x,y,z)) \wedge \phi(x,y)) = \Gamma_1 \wedge (\exists y.\ (\Gamma_2(x,y,z)) \wedge \phi(x,y)) = \Gamma_1 \wedge \phi^{\Gamma_2}.$$

Regarding the second part, we have $\Gamma \wedge \phi_\Gamma = \Gamma_1 \wedge \Gamma_2 \wedge (\Gamma_1 \rightarrow \phi_{\Gamma_2}) \models \Gamma \wedge \phi_{\Gamma_2}$, so ϕ_{Γ_2} is an optimal Y-antecedent. Similarly $\Gamma \wedge \phi^{\Gamma_2} = \Gamma_1 \wedge \Gamma_2 \wedge \Gamma_1 \wedge \phi^{\Gamma_2} = \Gamma \wedge \phi^{\Gamma} \models \phi^{\Gamma}$, so ϕ_{Γ_2} is an optimal Y-consequent. □

The following example is a continuation of Example 5.5.

Example 5.8 (Continuation of Example 5.5). We consider the computation of the Y-antecedent using a subset of the context. Per Proposition 5.7, ϕ_{Γ_2} is an optimal Y-antededent. As we just discussed, since it is optimal, we may as well use ϕ_{Γ_2} instead of ϕ_Γ as the Y-antecedent of ϕ. We obtain

$$\phi_{\Gamma_2} = \forall y.\ ((z \leq y) \rightarrow (x \leq y)) = (x \leq z).$$

For requirement engineering, it is preferable to output ϕ_{Γ_2} instead of ϕ_Γ because it is syntactically simpler. ■

Now we discuss an optimization formulation of antecedent/consequent synthesis.

Proposition 5.9. *Suppose that ϕ can be expressed as $\phi(x, g(y))$, where ϕ is monotonic in the second argument. Define*

$$g^-(a,c) = \begin{cases} \mathit{minimize} & g(b) \\ {\scriptstyle b \in \mathrm{dom}(y)} & \\ \mathit{subject\ to} & [x \coloneqq a, y \coloneqq b, z \coloneqq c] \models \Gamma \end{cases} \quad \textit{and} \quad g^+(a,c) = \begin{cases} \mathit{maximize} & g(b) \\ {\scriptstyle b \in \mathrm{dom}(y)} & \\ \mathit{subject\ to} & [x \coloneqq a, y \coloneqq b, z \coloneqq c] \models \Gamma. \end{cases}$$

Then $\phi(x, g^-(x,z))$ is an optimal Y-antecedent of ϕ in Γ, and $\phi(x, g^+(x,z))$ is an optimal Y-consequent of ϕ in Γ.

Proof. If $[x \coloneqq a, y \coloneqq b, z \coloneqq c] \not\models \Gamma$ for all $b \in \mathrm{dom}(y)$, then $[x \coloneqq a, z \coloneqq c] \models \phi_\Gamma$. Otherwise, from Lemma 5.4, we have $\mathbf{Den}\,(\phi_\Gamma(a,c)) = \bigcap_{\substack{b \in \mathrm{dom}(y) \\ \Gamma(a,b,c)}} \mathbf{Den}\,(\phi(a, g(b))) = \mathbf{Den}\,(\phi(a, g^-(a,c)))$. The second part is proved similarly. □

In addition to yielding optimal Y-antecedents and consequents, Proposition 5.9 has two advantages. First, it enables us to use tools and intuition from optimization in order to compute Y-antecedents and consequents, as we will do in Section 6. Second, the Y-antecedents and consequents obtained from the optimization formulation are the result of a substitution of the irrelevant variables by $g^-(x,z)$ or $g^+(x,z)$. This means that the optimization formulation yields results that are syntactically similar to the original expression ϕ. We believe that keeping this similarity is important in requirement engineering, as the syntax of requirements entered by users has a close connection to the semantics they want to express. The following example illustrates this point.

Example 5.10. Suppose we want to compute an antecedent of the formula

$$\phi\colon (p \wedge q) \vee r$$

in the context $\Gamma\colon s \rightarrow q$, where q is an irrelevant variable. Let $g(q) = q$ and $\phi(x, g(q)) = (p \wedge q) \vee r$. Then ϕ is monotonic in its last argument (i.e., changing the value of q from 0 to 1 can never change the value of ϕ from 1 to 0). To apply Proposition 5.9, we compute g^-:

$$g^-(a,c) = \begin{cases} \text{minimize} & q \\ {\scriptstyle b \in \{0,1\}} & \\ \text{subject to} & [p \coloneqq a, q \coloneqq b, s \coloneqq c] \models s \rightarrow q \end{cases} = c.$$

By Proposition 5.9, we conclude that $\phi(p, g^-(p,s))$ is a Y-antecedent of ϕ in the given context, i.e., we get the antecedent $(p \wedge s) \vee r$. In contrast, ϕ_Γ is $\forall y.\ ((s \rightarrow q) \rightarrow ((p \wedge q) \vee r)) = (p \vee r) \wedge (s \vee r)$. ■

In Example 5.10, ϕ_Γ and $\phi(x, g^-(x,z))$ match. Yet, we observe that the latter immediately yields a result in a syntactic form which is closer to the original ϕ. This happens because this expression is obtained by simply replacing q with $g^-(x,z)$ in ϕ.

5.2 Compositional results

We now express ϕ and Γ as $\phi = \bigvee_j \bigwedge_i \phi_i^j(x,y)$, where the $\phi_i^j(x,y)$ are clauses (i.e., terms formed from atomic formulas and Boolean connectives), and $\Gamma = \bigvee_k \Gamma^k(x,y,z)$, where each $\Gamma^k(x,y,z)$ is a conjunction of clauses. Instead of synthesizing an antecedent or consequent for ϕ directly, we will seek methods to do this compositionally.

Proposition 5.11. *Let* $\tilde{\phi}_\Gamma \colon \bigvee_j \bigwedge_{i,k} \forall y.\ \left(\Gamma^k \to \phi_i^j\right)$ *and* $\tilde{\phi}^\Gamma \colon \bigvee_{k,j} \bigwedge_i \exists y.\ \left(\Gamma^k \wedge \phi_i^j\right)$. $\tilde{\phi}_\Gamma$ *is an antecedent and* $\tilde{\phi}^\Gamma$ *a consequent of* ϕ *in the context* Γ.

Proof. $\tilde{\phi}_\Gamma = \bigvee_j \bigwedge_{i,k} \forall y.\ (\Gamma^k \to \phi_i^j) = \bigvee_j \forall y.\ (\Gamma \to \bigwedge_i \phi_i^j) \models \forall y.\ \bigvee_j (\Gamma \to \bigwedge_i \phi_i^j) = \phi_\Gamma$. By applying Proposition 5.3, we proved the first part. We also have $\phi^\Gamma = \exists y.\ (\Gamma \wedge \phi) = \exists y.\ \bigvee_{k,j} \bigwedge_i (\Gamma^k \wedge \phi_i^j) = \bigvee_{k,j} \exists y.\ \bigwedge_i (\Gamma^k \wedge \phi_i^j) \models \bigvee_{k,j} \bigwedge_i \exists y.\ (\Gamma^k \wedge \phi_i^j) = \tilde{\phi}^\Gamma$, which shows the second part after applying Proposition 5.3. □

The compositional result of Proposition 5.11 allows us to focus on the case in which ϕ is a clause and Γ is a conjunction of clauses. *We shall assume this from now on.*

To further exploit compositionality, express ϕ as

$$\phi(x,y) = \phi(x, g_1(y), \ldots, g_p(y)), \tag{5}$$

where ϕ is required to be monotonic in all arguments, except the first. We assume that our language allows any clause to be expressed in this way. This is true for linear arithmetic and for real arithmetic with a ReLu (rectified linear unit) function.

Example 5.12. Suppose ϕ is the real-arithmetic formula $x^2 - 4xy + y^2 \leq 0$. This is equivalent to $x^2 - 4r(x)y + 4r(-x)y + y^2 \leq 0$, where r is the ReLu function. We can thus write ϕ as $\phi(x, g_1(y), g_2(y), g_3(y))$ with $g_1(y) = y$, $g_2(y) = -y$, and $g_3(y) = -y^2$. ■

We now study how to exploit the structure (5) to compute Y-antecedents and consequents for ϕ in Γ.

Proposition 5.13. *Let*

$$g_i^-(a,c) = \begin{cases} \textit{minimize} & g_i(b) \\ b \in \mathrm{dom}(y) & \\ \textit{subject to} & [x \coloneqq a, y \coloneqq b, z \coloneqq c] \models \Gamma \end{cases} \quad \textit{and} \quad g_i^+(a,c) = \begin{cases} \textit{maximize} & g_i(b) \\ b \in \mathrm{dom}(y) & \\ \textit{subject to} & [x \coloneqq a, y \coloneqq b, z \coloneqq c] \models \Gamma. \end{cases}$$

The formulas $\phi(x, g_1^-(x,z), \ldots, g_p^-(x,z))$ *and* $\phi(x, g_1^+(x,z), \ldots, g_p^+(x,z))$ *are, respectively, y-antecedents and consequents of* ϕ *in* Γ.

Proof. From Lemma 5.4, we have

$$\mathbf{Den}(\phi_\Gamma) = \bigcap_{\substack{b \in \mathrm{dom}(y) \\ \Gamma(x,b,z)}} \mathbf{Den}(\phi) \supseteq \mathbf{Den}\left(\phi(x, g_1^-(x,z), \ldots, g_p^-(x,z))\right).$$

We conclude that $\phi(x, g_1^-(x,z), \ldots, g_p^-(x,z)) \models \phi_\Gamma$. Proposition 5.3 yields the first part. The second part is proved similarly. □

Proposition 5.13 tells us that we can independently optimize over the monotonic functions composing ϕ to compute Y-antecedents and consequents. This result does not yield the optimality guarantees of Proposition 5.9, but it allows us to compute antecedents/consequents compositionally.

Example 5.14. Continuing Example 5.12, suppose we want to compute a consequent of ϕ in the context $\Gamma \colon (y \leq 2) \wedge (1 \leq y)$. We apply Proposition 5.13 by optimizing over the g_i separately. We obtain $g_1^+(x) = 2$, $g_2^+(x) = -1$, and $g_3^+(x) = -1$. The formula $\phi(x, g_1^+(x), g_2^+(x), g_3^+(x)) = x^2 - 8r(x) + 4r(-x) + 1 \leq 0$ is a Y-consequent of ϕ in Γ.

Finally, we consider an example in LTL.

Example 5.15. Suppose we want to compute an antecedent of the formula

$$\phi : \mathbf{F}((q \wedge \neg s) \vee \mathbf{G}p)$$

in the context $\Gamma : \mathbf{X}^5(r \rightarrow q)$, where q is an irrelevant variable. Observe that ϕ is equal to $\mathbf{F}(q \wedge \neg s) \vee \mathbf{FG}p = \left(\bigvee_{i=0}^{\infty} X^i(q \wedge \neg s)\right) \vee \mathbf{FG}p$. Define $g_i = \mathbf{X}^i(q \wedge \neg s)$. We observe that ϕ is monotonic in all g_i. We compute $g_i^- = \begin{cases} 0 & i \neq 5 \\ X^5 r & i = 5 \end{cases}$. Per proposition 5.13, the formula $\mathbf{X}^5\ (r \wedge \neg s) \vee \mathbf{FG}p$ is a solution to our problem. ∎

6 COMPUTING CONTRACT OPERATIONS EXPRESSED AS POLYHEDRAL CONSTRAINTS

Section 4 discussed algorithms to implement the contract operations according to the principles laid out in Section 3. Section 5 discussed the mathematical meaning and solution of the routines ElimVarsByRefining and ElimVarsByRelaxing used in Algorithms 1 and 2. This section discusses how Pacti supports specifications expressed as polyhedral constraints. To do this, we explain how the specification-processing routines of Algorithms 1 and 2 are implemented in the case of polyhedral constraints.

By a polyhedral constraint, we mean linear inequalities with real coefficients connected by conjunctions or disjunctions. Polyhedral constraints are an intuitive formalism for writing specifications for complex systems, as they allow us to place piecewise linear bounds on quantities of interest. To implement Reduce, we use standard methods for the elimination of redundant terms, e.g., [36, 47]. To compute IsRefinement, i.e., to verify whether a polyhedron is contained inside another, one can use linear programming, as shown by [18] in their solution of the "HH formulation" of the optimal containment problem. Most of this section is devoted to the implementation of ElimVarsByRefining and ElimVarsByRelaxing for polyhedral constraints. Our starting point is the general solutions given in Section 5.

6.1 Linear inequality constraints

Per Proposition 5.11, we will focus our attention on algorithms for the efficient computation of antecedents and consequents when ϕ is an atom (i.e., a linear inequality) and Γ a conjunction of atoms. Thus, ϕ will have the form

$$\phi : \quad \sum_{i=1}^{m} p_i x_i + \sum_{i=1}^{n} q_i y_i + r \leq 0,$$

where r and the p_i and q_i are constants. The set of irrelevant variables to be eliminated is $Y = \{y_i\}_i$. The context Γ is a set of linear inequalities of the form

$$\Gamma = \left\{ \sum_{j=1}^{m} \alpha_{ij} x_i + \sum_{j=1}^{n} \beta_{ij} y_i + \sum_{j=1}^{o} \gamma_{ij} z_i + K_i \leq 0 \right\}_{i=1}^{N},$$

where the K_j, α_i^j, β_i^j, and γ_i^j are constants.

Let $A = (\alpha_{ij}) \in \mathbb{R}^{N \times m}$, $B = (\beta_{ij}) \in \mathbb{R}^{N \times n}$, $C = (\gamma_{ij}) \in \mathbb{R}^{N \times o}$, $K \in \mathbb{R}^N$, $p \in \mathbb{R}^m$, and $q \in \mathbb{R}^n$. We also let $x = (x_i)$, $y = (y_i)$, $z = (z_i)$ be m-, n-, and o-dimensional vectors of variables, respectively.

Our problem is to eliminate the y variables from

$$\phi : \quad p^\intercal x + q^\intercal y + r \leq 0 \tag{6}$$

using the context

$$\Gamma : \quad Ax + By + Cz + K \leq 0 \tag{7}$$

by computing Y-antecedents/consequents.

This problem can be addressed via Fourier-Dines-Motzkin elimination [16, 20, 38], to which many improvements have been made—see [9, 14, 17, 22, 32]. As known algorithms have at least exponential worst case complexity, we seek alternative methods to enable fast contract computations.

Let $b(x, z) = -K - Ax - Cz$. We obtain the following corollary from Proposition 5.9.

Corollary 6.1. *Let ϕ and Γ be as above. Let*

$$g^-(x,z) = \begin{cases} \underset{y\in\mathbb{R}^n}{\text{minimize}} & -q^\top y \\ \text{subject to} & By \le b(x,z) \end{cases} \tag{8}$$

and

$$g^+(x,z) = \begin{cases} \underset{y\in\mathbb{R}^n}{\text{maximize}} & -q^\top y \\ \text{subject to} & By \le b(x,z). \end{cases} \tag{9}$$

Then the formula $p^\top x - g^-(x,z) \le r$ is an optimal Y-antecedent of ϕ in the context Γ and $p^\top x - g^+(x,z) \le r$ is an optimal Y-consequent of ϕ in the context Γ.

Example 6.2. Suppose we wish to eliminate variables y_1 and y_2 from $2x + y_1 - 2y_2 \le 5$ through antecedent computation, using the context $\{x - 2y_1 + y_2 + z \le 1, 3y_1 - 4y_2 \le 6\}$. We compute

$$g^-(x,z) = \begin{cases} \underset{y_1,y_2\in\mathbb{R}}{\text{minimize}} & -(y_1 - 2y_2) \\ \text{subject to} & x - 2y_1 + y_2 + z \le 1 \\ & 3y_1 - 4y_2 \le 6 \end{cases} = -\left(4 - \frac{2}{5}(x+z)\right).$$

The antecedent formula is $2x + 4 - \frac{2}{5}(x + z) \le 5$, which becomes $8x - 2z \le 5$. ∎

Example 6.3. Suppose we wish to eliminate variables y_1 and y_2 from $x + 5y_1 - 2y_2 \le 5$ by the computation of a consequent, using the context $\{x - 2y_1 + y_2 + z \le 1, 3y_1 - 4y_2 \le 6\}$. We compute

$$g^+(x,z) = \begin{cases} \underset{y_1,y_2\in\mathbb{R}}{\text{maximize}} & -(5y_1 - 2y_2) \\ \text{subject to} & x - 2y_1 + y_2 + z \le 1 \\ & 3y_1 - 4y_2 \le 6 \end{cases} = -\left(-4 + \frac{14}{5}(x+z)\right).$$

The consequent is $x - 4 + \frac{14}{5}(x + z) \le 5$, or $19x + 14z \le 45$. ∎

6.2 Solving the symbolic optimization problems

Corollary 6.1 provides an explicit expression to compute optimal Y-antecedents and consequents. The next issue we face is the computation of (8) and (9). Both are linear programs, but their solutions are symbolic due to the presence of $b(x, z)$. We observe that if we have a context Γ' such that $\Gamma = \Gamma' \wedge \Gamma''$ and if ϕ' is a Y-antecedent/consequent of ϕ in the context Γ', then ϕ' is also a Y-antecedent/consequent in the context Γ. This means that we can develop techniques to find a subset Γ' of the context Γ in which solving the problems (8) and (9) is efficient. In Section 6.2.1, we introduce a condition (Definition 6.4) on a context Γ' that ensures that the solutions of the LPs can be expressed as the solutions of linear systems of equations (Lemma 6.5). The question then becomes how to identify a subset Γ' of the context Γ meeting the requirements of Definition 6.4. We discuss two techniques to do this in Sections 6.2.2 and 6.2.3.

6.2.1 Optimization in a subset of the context. A linear program achieves its optimal value on the boundary of its constraints. If the context Γ contains N constraints and is a bounded polyhedron, then the optimal value of the linear program will occur at one of the $\binom{N}{n}$ possible vertices. We will look for ways to choose n constraints from Γ such that the optimization problems achieve optimal values at the vertex determined by those n constraints. First, we focus on solving symbolic LPs when the context contains n constraints. The following definition will be useful:

Definition 6.4. Let $M \in \mathbb{R}^{n\times n}$ and $v \in \mathbb{R}^n$. We say that (M, v) is a refining pair if M is invertible and $(M^\intercal)^{-1}v$ has nonnegative entries. We say that the pair (M, v) is a relaxing pair if M is invertible and $-(M^\intercal)^{-1}v$ has nonnegative entries.

As the next result shows, these conditions are sufficient to solve the problems (8) and (9) when there are as many context formulas as irrelevant variables (i.e., when $N = n$). Suppose $J \subseteq \{1, \dots, N\}$ has cardinality n. We let $B_J = (\beta_{J_i,j})_{i,j=1}^n$ and $b_J = (b_{J_i})_{i=1}^n$ be the J-indexed rows of B and b, respectively.

Lemma 6.5. *Suppose (B_J, q) is a refining pair. Then*

$$\begin{cases} \underset{y\in\mathbb{R}^n}{\text{maximize}} & q^\intercal y \\ \text{subject to} & B_J y \le b_J(x,z) \end{cases} = q^\intercal B_J^{-1} b_J(x,z).$$

Suppose (B_J, q) is a relaxing pair. Then

$$\begin{cases} \underset{y\in\mathbb{R}^n}{\text{minimize}} & q^\intercal y \\ \text{subject to} & B_J y \le b_J(x,z) \end{cases} = q^\intercal B_J^{-1} b_J(x,z).$$

Proof. Let (B_J, q) be a refining pair. We consider the first problem and its Lagrange dual (see [5], Section 5.2.1):

$$\text{primal} \begin{cases} \underset{y}{\text{minimize}} & -q^\intercal y \\ \text{subject to} & B_J y \le b_J(x,z) \end{cases} \qquad \text{dual} \begin{cases} \underset{\lambda}{\text{maximize}} & -b_J^\intercal \lambda \\ \text{subject to} & B_J^\intercal \lambda - q = 0 \\ & \lambda \ge 0 \end{cases}$$

The dual problem only admits the solution $\lambda^\star = (B_J^\intercal)^{-1}q$ if $\lambda^\star \ge 0$, which is the case, as (B_J, q) is a refining pair. Thus, the optimal value of the dual problem is $v^\star = -q^\intercal(B_J^{-1}b_J)$. As strong duality holds for any linear program (see [5], Section 5.2.4), $v^\star$ is also the optimal value of the primal problem. The statement of the theorem follows.

Now suppose (B_J, q) is a relaxing pair. We consider the second problem and its dual:

$$\text{primal} \begin{cases} \underset{y}{\text{minimize}} & q^\intercal y \\ \text{subject to} & B_J y \le b_J(x,z) \end{cases} \qquad \text{dual} \begin{cases} \underset{\lambda}{\text{maximize}} & -b_J^\intercal \lambda \\ \text{subject to} & B_J^\intercal \lambda + q = 0 \\ & \lambda \ge 0 \end{cases}$$

The dual only admits the solution $\lambda^\star = -(B_J^\intercal)^{-1}q$ if $\lambda^\star \ge 0$, which is the case because (B_J, q) is a relaxing pair. The optimal value of the dual problem is $v^\star = q^\intercal(B_J^{-1}b)$. Due to strong duality, $v^\star$ is also the optimal value of the primal problem. □

As a consequence of Corollary 6.1 and Lemma 6.5, we obtain the following result:

Corollary 6.6. *With all definitions as above, if (B_J, q) is a refining pair, then $p^\intercal x + q^\intercal B_J^{-1} b_J(x,z) \le r$ is a Y-antecedent of $p^\intercal x + q^\intercal y \le r$ in the context $By \le b(x,z)$. If (B_J, q) is a relaxing pair, then $p^\intercal x + q^\intercal B_J^{-1} b_J(x,z) \le r$ is a Y-consequent of $p^\intercal x + q^\intercal y \le r$ in the context $By \le b(x,z)$.*

Table 1. Execution time in seconds of Algorithm 4 for a given total number of variables, number of constraints in the context Γ, and number of irrelevant variables (or variables that need to be eliminated). For each combination of these quantities, we give two numbers: the first corresponds to the execution time of the algorithm when the context is a matrix with 25% density, and the second to the execution time when the context is a matrix with 100% density.

	Constraints in context	Total number of variables					
		5	10	15	20	25	30
2 irrelevant variables	5	0.15|0.34	0.21|0.63	0.31|0.92	0.36|1.25	0.42|1.60	0.51|1.97
	10	0.15|0.32	0.21|0.62	0.34|0.96	0.39|1.27	0.44|1.62	0.51|2.00
	20	0.16|0.34	0.23|0.66	0.31|0.97	0.38|1.54	0.57|1.77	0.51|2.09
	100	0.27|0.71	0.30|1.17	0.38|1.68	0.48|2.24	0.61|2.95	0.69|3.56
	300	0.84|3.45	0.51|4.65	0.65|6.24	0.76|8.27	0.93|10.66	1.01|13.45
4 irrelevant variables	5	0.27|0.69	0.39|1.23	0.53|2.02	0.73|2.87	0.85|3.13	1.14|3.77
	10	0.25|0.65	0.45|1.23	0.63|1.83	0.77|2.48	1.02|3.20	1.26|3.96
	20	0.24|0.66	0.44|1.29	0.56|1.89	0.78|2.85	1.17|3.27	1.34|4.02
	100	0.40|1.06	0.49|1.88	0.58|2.58	0.69|3.45	1.22|4.37	1.37|5.42
	300	0.98|3.82	0.74|5.35	0.82|7.25	1.00|9.41	1.38|12.25	1.74|15.33

Corollary 6.6 gives explicit formulas for computing Y-antecedents/consequents of a formula in a context. This result is missing methods for computing J, the set of the indices of formulas in Γ, in such a way that it yields refining or relaxing pairs (B_J, q), as needed. We now consider a method to identify J.

6.2.2 Computing J by seeking positive solutions to linear equations. We will construct J by identifying constraints yielding linear systems of equations whose solutions are guaranteed to be nonnegative. We will use the following result.

Theorem 6.7 (Kaykobad [29]). *Let $M = (\mu_{ij}) \in \mathbb{R}^{n\times n}$ and $v \in \mathbb{R}^n$. Suppose the entries of M are nonnegative, its diagonal entries are positive, the entries of v are positive, and $v_i > \sum_{j\neq i} \mu_{ij} \frac{v_j}{\mu_{jj}}$ for all $i \leq n$. Then M is invertible and $M^{-1}v$ has positive entries.*

Definition 6.8. A pair (M, v), where $M = (\mu_{ij}) \in \mathbb{R}^{n\times n}$ and $v \in \mathbb{R}^n$, satisfying the conditions of Theorem 6.7 is called a *Kaykobad pair.*

We have the following result.

Lemma 6.9. *Let Q be an $n \times n$ diagonal matrix whose i-th diagonal entry is $\textsc{sign}(q_i)$. Let $\bar{B}_J = B_J Q$ and $\bar{q} = Qq$. If $(\bar{B}_J^\intercal, \bar{q})$ is a Kaykobad pair, then (B_J, q) is a refining pair. If $(-\bar{B}_J^\intercal, \bar{q})$ is a Kaykobad pair, then (B_J, q) is a relaxing pair.*

Proof. Suppose $(\bar{B}_J^\intercal, \bar{q})$ is a Kaykobad pair. Then $\bar{B}_J^\intercal$ is invertible. We have $B_J(\bar{B}_J Q)^{-1} = B_J Q(\bar{B}_J)^{-1} = I$ and $(\bar{B}_J Q)^{-1} B_J = Q(\bar{B}_J)^{-1}(B_J Q)Q = I$, so B_J is invertible. Moreover, we have

$$0 < (\bar{B}_J^\intercal)^{-1}\bar{q} = (QB_J^\intercal)^{-1}(Qq) = (B_J^\intercal)^{-1}q,$$

which means that (B_J, q) is a refining pair.

If $(-\bar{B}_J^\intercal, \bar{q})$ is a Kaykobad pair, then $\bar{B}_J^\intercal$ is invertible, which means that so is B_J. Moreover,

$$0 < -(\bar{B}_J^\intercal)^{-1}\bar{q} = -(QB_J^\intercal)^{-1}(Qq) = -(B_J^\intercal)^{-1}q.$$

Thus, (B_J, q) is a relaxing pair. □

Algorithm 4 Antecedents and consequents for linear inequality constraints by identifying systems of equations with positive solutions

Input: Term to transform $p^\intercal x + q^\intercal y \leq r$, context Γ,
transform instruction s (**true** for antecedents and **false** for consequents)
Output: Transformed term t' lacking any y variables

MatrixRowTerms $\leftarrow \emptyset$ ▷ Rows of the context matrix A
PartialSums $\leftarrow$ zeros(length(y))
TCoeff $\leftarrow -1$
if s **then**
    TCoeff $\leftarrow 1$
for $i = 1$ to $i =$ length(y) **do** ▷ One iteration per row of context matrix
    IthRowFound $\leftarrow$ **false** ▷ Indicate whether we could add the i-th row
    for $\gamma \in \Gamma \setminus$ MatrixRowTerms **do**
        ▷ 1. Verifying Kaykobad pair: sign of nonzero matrix terms
        TermIsInvalid $\leftarrow$ **false**
        for $j = 1$ to $j =$ length(y) **do**
            if coeff(γ, y_j) $\neq 0$ and sign(coeff(γ, y_j)) $\neq$ sign(q_j) $\cdot$ TCoeff **then**
                TermIsInvalid $\leftarrow$ **true**
                break
        ▷ 2. Verifying Kaykobad pair: matrix diagonal terms
        if coeff(γ, y_i) $= 0$ or TermIsInvalid **then**
            next
        ▷ 3. Verifying Kaykobad pair: relationship between matrix and vector entries
        Residuals $\leftarrow$ zeros(length(y))
        for $j = 1$ to $j =$ length(y) **do**
            if $j \neq i$ **then**
                Residuals$[j] \leftarrow$ sign(q_j) $\cdot$ TCoeff $\cdot$ coeff(γ, y_j) $\cdot \frac{q_i}{\text{coeff}(\gamma, y_i)}$
            if $|q_j| \cdot$ TCoeff $\leq$ PartialSums$[j]$ + Residuals$[j]$ **then**
                TermIsInvalid $\leftarrow$ **true**
                break
        if not TermIsInvalid **then**
            ▷ Resulting matrix is meeting Kaykobad pair conditions at i-th row
            IthRowFound $\leftarrow$ **true**
            for $j = 1$ to $j =$ length(y) **do**
                PartialSums$[j] \leftarrow$ PartialSums$[j]$ + Residuals$[j]$
            MatrixRowTerms.append(γ)
            break
    if not IthRowFound **then**
        return Error: Cannot transform term
$B \leftarrow$ MatrixFromTerms(MatrixRowTerms, y)
$b \leftarrow$ VectorFromTerms(MatrixRowTerms, y)
return $p^\intercal x + q^\intercal B^{-1} b \leq r$

Corollary 6.6 and Lemma 6.9 yield a method for computing Y-antecedents and consequents of formulas in a context. To use it, we must construct J such that (B_J, q) meets the corollary's conditions. We construct J by choosing n formulas from the context Γ; these formulas must meet the conditions of a Kaykobad pair. One advantage of the Kaykobad condition is that it allows us to incrementally identify suitable constraints to add to the context Γ', i.e., we don't have to select n constraints before we run the verification. That is, when we have identified $k < n$ constraints, we can easily verify whether a candidate $(k+1)$-th formula would be acceptable for constructing a Kaykobad pair. Algorithm 4 computes Y-antecedents and consequents for linear inequality constraints based on Corollary 6.6. Lines 6–30 search the context Γ for n constraints meeting the Kaykobad conditions. The rest of the algorithm computes the Y-antecedents/consequents. If there are n variables to be eliminated, and $N = |\Gamma|$ constraints in the context Γ, the algorithm has complexity $O(n^2N + N^3)$. The function $\textsc{coeff}(\gamma, y_j)$ extracts the coefficient of the variable y_j from the term γ. The call $\textsc{MatrixFromTerms}(\text{MatrixRowTerms}, y)$ extracts all coefficients of the y variables contained in MatrixRowTerms and makes these coefficients the rows of the resulting matrix. The call $\textsc{VectorFromTerms}(\text{MatrixRowTerms}, y)$ returns a vector of all expressions contained in MatrixRowTerms with their y variables removed. These are the elements of $b(x, z)$. Finally, $\textsc{diag}(v)$ returns a diagonal matrix whose entries are the vector v.

We implemented Algorithm 4 in Python and generated benchmarks for it. We considered formulas ϕ of the form (6) and contexts of the form (7). We varied the number N of constraints in the context, the number m of variables to be eliminated, and the total number of variables present in the problem $n + m + o$ (in our experiments ϕ and Γ had the same variables, so we had $o = 0$). We randomly generated coefficients for all variables in ϕ and Γ. We executed the algorithm for the situations in which we wanted to eliminate 2 irrelevant variables and 4 irrelevant variables. The results of the experiments are shown in Table 1. The first of the two numbers shown for each entry in the table corresponds to the execution time of the algorithm when each constraint in the context has an average of $0.25n$ variables with nonzero coefficients. The second element corresponds to the situation when all variables of all constraints in the context have nonzero coefficients. We observe that sparsity has a significant effect on execution time.

6.2.3 Computing J via linear programming. Now we will build J by numerically solving (8) and (9) for fixed values of x and z.

Lemma 6.10. *Let $a \in \mathbb{R}^m$ and $c \in \mathbb{R}^o$.*

- *Suppose $g^-(a, c)$ is finite and the optimum of the LP (8) (with $x = a$ and $z = c$) is attained at $y^\star$. Let $J = \left\{ i \;\middle|\; b_i(a, c) - \sum_{j=1}^{n} \beta_{ij} y_j^\star = 0 \right\}$ and assume that $|J| = n$, where n is the number of optimization variables y in g^-. If B_J is invertible, then (B_J, q) is a refining pair.*
- *Similarly, suppose $g^+(a, c)$ is finite and the optimum of the LP (9) (with $x = a$ and $z = c$) is attained at $y^\star$. Let $J = \left\{ i \;\middle|\; b_i(a, c) - \sum_{j=1}^{n} \beta_{ij} y_j^\star = 0 \right\}$ and assume that $|J| = n$. If B_J is invertible, then (B_J, q) is a relaxing pair.*

Proof. We prove the first part. Consider the following problems:

$$\text{primal} \begin{cases} \underset{y}{\text{minimize}} & -q^\intercal y \\ \text{subject to} & By \le b(a, c) \end{cases} \qquad \text{dual} \begin{cases} \underset{\lambda}{\text{maximize}} & -b(a, c)^\intercal \lambda \\ \text{subject to} & B^\intercal \lambda - q = 0 \\ & \lambda \ge 0 \end{cases}$$

Since $g^-(a, c)$ is finite, the primal is feasible. By strong duality, so is the dual. Let $\lambda^\star$ be the value of λ where the dual attains its optimum. Then $\lambda^\star \ge 0$ and $0 = B^\intercal \lambda^\star - q = B_J^\intercal \lambda_J^\star + B_{\hat{J}}^\intercal \lambda_{\hat{J}}^\star - q$, where $\hat{J} = \{1, \ldots, N\} \setminus J$. Due to complementary slackness, we know that $\lambda_{\hat{J}}^\star = 0$. Thus, $0 = B_J^\intercal \lambda_J^\star - q$. By assumption, B_J is invertible. Then (B_J, q) is a refining pair. The proof of the second part is similar. □

Lemma 5 allows us to obtain the solution to a linear programming problem with symbolic constraints $By \leq b(x,z)$ in a reduced context $B_J y \leq b_J(x,z)$, where we identify J by solving a numerical LP. Lemma 5 and Corollary 6.6 yield a method for computing Y-antecedents and consequents. This method is reflected in Algorithm 5. As before, MatrixFromTerms(Γ, y) and VectorFromTerms(Γ, y) extract from the context Γ the matrix B and symbolic vector $b(x,z)$ of the constraints $By \leq b(x,z)$. Evaluate(b,a,c) returns the vector $b(a,c) \in \mathbb{R}^N$. LinearProgramming(q,B,b_e) solves the LP $\min_y q^\intercal y$ subject to $By \leq b_e$ and returns a success variable and the value $y^\star$ where the minimum is attained. The success variable is true when the LP is feasible and has a finite solution. MatrixInv computes matrix inverses. Its success variable is false when the matrix is not invertible.

Algorithm 5 Antecedents and consequents for linear inequality constraints through linear programming

Input: Term to transform $p^\intercal x + q^\intercal y \leq r$, context Γ, $a \in \mathbb{R}^m$, $c \in \mathbb{R}^o$,
transform instruction s (**true** for antecedents and **false** for consequents)
Output: Transformed term t' lacking any y variables

$B \leftarrow$ MatrixFromTerms(Γ, y)
$b \leftarrow$ VectorFromTerms(Γ, y)
$b_e \leftarrow$ Evaluate(b,a,c)
if s **then**
    (success, $y^\star$) $\leftarrow$ LinearProgramming$(-q,B,b_e)$
else
    (success, $y^\star$) $\leftarrow$ LinearProgramming(q,B,b_e)
if not success **then**
    return Error: LP is unfeasible
$S \leftarrow b_e - By^\star$
$J \leftarrow \emptyset$
for $j = 1$ to $j =$ length(b) **do**
    if $S_j = 0$ **then**
        $J \leftarrow J \cup \{j\}$
(success, $\hat{B}_J$) $\leftarrow$ MatrixInv(B_J)
if not success **then**
    return Error: cannot invert B_J
return $p^\intercal x + q^\intercal \hat{B}_J b_J \leq r$

7 CASE STUDIES

We implemented Pacti, a Python package that allows us to carry out system-level reasoning using assume-guarantee contracts, and made it available open source[2] under a BSD 3-Clause license. Pacti supports all IO contract operations described in Section 4. Its implementation of the contract algebra is orthogonal from that of the specification formalism in which contracts are expressed. The first specification theory supported by Pacti is polyhedral constraints.

The main object represented in Pacti is the contract. In the tool, the contract $(\{o\}, \{o'\}, \{o \leq 3\}, \{o' \leq 2o - 3\})$ is represented as

[2]https://github.com/pacti-org/pacti

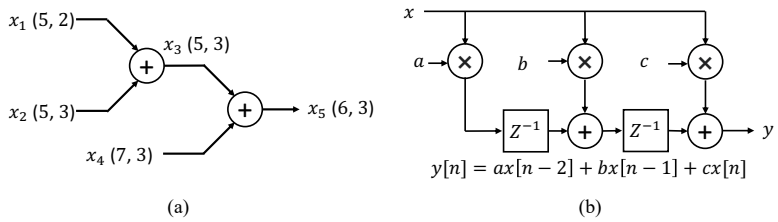


Fig. 2. (a) An example of two adders. The word-length is denoted beside the name of the number. (b) An example of a 3-tap digital filter.

```
c1 = PolyhedralIoContract.from_strings(
     InputVars=["o"],
     OutputVars=["o_p"],
     assumptions=["o <= 3"],
     guarantees=["o_p <= 2o - 3"])
```

Given contracts c1 and c2, we compute contract composition, quotient, and merging through the following functions, respectively:

```
c_system = c1.compose(c2)
c_merged_viewpoints = c1.merge(c2)
c_missing_component = c1.quotient(c2)
```

The descriptions of these basic operations are given in Section 2. In this section, we consider the application of contracts and Pacti in several case studies. For every case study, we consider how contract operations yield insight to designers. In general, our conceptual use of contract operations is as follows: We use contract quotient to find the specification of a component that needs to be added to a system in order to satisfy a top-level objective. Contract composition is used in two situations: (i) to compute the specifications of systems generated as an interconnection of subsystems and (ii) to identify interconnection problems. Regarding the latter point, Pacti will sometimes conclude that the guarantees of an interconnected contract are insufficient to satisfy the assumptions of the contract whose inputs it is driving. In this case, the tool will output an error that indicates an interconnection issue. Contract merging is used to write specifications separately for each of the viewpoints of our components. Detailed descriptions of the case studies covered in this section together with their accompanying code can be found on Pacti's website[3].

7.1 Signal processing pipelines in digital integrated circuits

The numerical representations of digital signal processing algorithms must often be translated from floating-point to fixed-point when the algorithms are implemented in hardware. To reduce implementation costs and increase performance, one of the objectives of this translation is to use the smallest possible fixed-point representations that allow the algorithm to operate within acceptable error bounds. This is a time-consuming and error-prone step that relies on statistical quantities gathered from simulations [6, 12, 30, 31, 44, 46]. As simulation yields limited coverage, techniques for verifying digital signal processing design have been proposed in the last decades [13, 19, 45].

This case study demonstrates the use of Pacti to find error bounds of fixed-point digital signal processing algorithms and perform local word-length optimization efficiently through contract *composition*, *quotient*, and *refinement*. We have two use cases. The first computes the bound in arithmetic error of a system obtained by composing several arithmetic operations, each introducing its own fixed-point error. The second imposes a total

[3]https://www.pacti.org/case_studies

error budget on the system and uses the quotient to find the maximum error that one of the components may have while meeting the system-level budget. This operation is helpful to size circuits. For simplicity, all numbers in this case study are unsigned.

Contract formulation of fixed-point operations. We first formulate fixed-point numbers and operations as contracts. The word-length of a fixed-point variable x is defined as a tuple (n_x, p_x). n_x denotes the number of bits to encode the fixed-point number, while p_x is the number of bits to encode the integer part [12]. Accordingly, the fractional part is encoded in $n_x - p_x$ bits.

We model a fixed-point number x using two variables x_e and x_a to represent the relationship between inputs and outputs. x_a is the maximum value that the variable can take, and x_e is the maximum error between the value of the fixed-point number and the ideal value of x. When x is a constant coefficient, x_e is the quantization error of the coefficient.

(1) *General Operation.* Given two input numbers x and y and an output number z with given word-lengths (n_x, p_x), (n_y, p_y), and (n_z, p_z), respectively, we form the contract for operation $z = f(x, y)$ as $C_{op} = (a_{op}, g_{op})$ that encodes the relation between variables $x_e, x_a, y_e, y_a, z_e, z_a$ as follows:

$$\begin{aligned} a_{op} &= \max_{\substack{0\le x\le x_a\\ 0\le y\le y_a}} f(x,y) < 2^{p_z} \\ g_{op} &= (z_e \le C) \wedge (z_a \le 2^{p_z} - 2^{p_z - n_z}) \wedge \left(z_a \le \max_{0\le x\le x_a, 0\le y\le y_a} f(x,y)\right), \end{aligned} \tag{10}$$

where $C = \max_{0\le x\le x_a, 0\le y\le y_a}(f(x,y) - f(x - x_e, y - y_e)) + \max(0, 2^{p_z}(2^{-n_z} - 2^{-(n_w - p_w + p_z)}))$ and (p_w, n_w) is the ideal minimum word-length that ensures no truncation loss and overflow hazards [12].
The assumption ensures no overflow occurs by checking that the maximal possible result can be fit into the range of the output variable. The guarantee is a conjunction of three clauses. The first clause bounds the output error considering the errors propagating from the input and the truncation error. The second clause states that the maximum output value is bounded by its fixed-point number representation. The third clause bounds the output value using the bounds on the inputs. Once we determine n_w, p_w, and the maximization terms for an operation, we can form the contract for it, even without the implementation details of the operation.

(2) *Addition.* Using (10), we can derive the contract C_{add} for addition as follows:

$$\begin{aligned} a_{add} &= x_a + y_a < 2^{p_z} \\ g_{add} &= (z_e \le x_e + y_e + C_{add}) \wedge (z_a \le 2^{p_z} - 2^{p_z - n_z} \wedge (z_a \le x_a + y_a)), \end{aligned} \tag{11}$$

where $C_{add} = 2^{p_z}(2^{-n_z} - 2^{p_y - \max(n_x, n_y - p_y + p_x) - \min(p_y - p_x, p_x - p_y) - p_z}$. Note that the contract for addition includes only polyhedral constraints.

(3) *Multiplication.* The contract $C_{mult} = (a_{mult}, g_{mult})$ for multiplication can be defined from (10) as follows:

$$\begin{aligned} a_{mult} &= x_a y_a < 2^{p_z} \\ g_{mult} &= (z_e \le y_a x_e + x_a y_e - x_e y_e + C_{mult}) \wedge (z_a \le 2^{p_z} - 2^{p_z - n_z}) \wedge (z_a \le x_a y_a), \end{aligned} \tag{12}$$

where $C_{mult} = 2^{p_z}(2^{-n_z} - 2^{-(n_x + n_y - p_x - p_y + p_z)})$. This contract includes non-polyhedral constraints due to the multiplications of the variables in the guarantee. However, if one of the inputs to the multiplication is a constant, the resulting contracts involve only polyhedral constraints. The operation of multiplying an input with a constant coefficient is common in signal processing.

Applying Pacti in the verification and optimization of fixed-point operations. We consider how we can use Pacti to reason about fixed-point error specifications and to generate meaningful results both for designers and optimization tools. Consider the two fixed-point adders shown in Figure 2a. We encode the contracts of

the two adders using the formulation in (11) as C^1 and C^2. Then we encode three contracts C^{x_1}, C^{x_2}, and C^{x_4} to represent the constraints on the three inputs x_1, x_2, and x_4 based on the word-lengths and assuming that the input has no errors. For example, the contract C^{x_1} for the input x_1 is formulated as $(\text{True}, 0 \leq x_{1a} \leq 7.75 \wedge x_{1e} = 0)$.

To obtain the error specification for the entire system, we compute the composition $C^{x_1} \parallel C^{x_2} \parallel C^{x_4} \parallel C^1 \parallel C^2$ using Pacti. The tool gives an error indicating that a system cannot be built because the guarantees of the first adder are insufficient to satisfy the assumptions of the second adder. This error means that an overflow might occur in x_3. Indeed, when $x_1 = 11.111$ and $x_2 = 111.11$, the result 1011.101 would cause an overflow in x_3, which only has three bits for the integer part. Therefore, the system does not work under general inputs and the designer should either modify the system or identify the possible range of input, which might be available from the top-level system specification or other contracts that specify these input variables. For example, if the maximum values of the inputs are constrained by $x_1 \leq 2$, $x_2 \leq 3.75$, and $x_4 \leq 0.03125$, we obtain the system-level contract $(\text{True}, g = x_{5a} \leq 5.8125 \wedge x_{5e} \leq 0.1875)$. The contract gives bounds for the value and the maximum error of the system-level output.

We consider a different situation. In the system of Figure 2a, suppose that we have a top-level contract C_{sys}, requiring that the resulting system has an output error smaller than 0.1, and we have the same input constraints as before. Our objective is to find the word-length of x_3 that would allow the system to meet its objective. First, we compute the quotient $C_{local} = C_{sys}/(C^{x_1} \parallel C^{x_2} \parallel C^{x_4})$ to get the local specification for components affected by x_3. From this specification, we update the word-length of x_3, compute the corresponding local contracts $C^1 \parallel C^2$ for the two adders (both contracts depend on x_3), and check if the local specification is satisfied using refinement $(C^1 \parallel C^2) \leq C_{local}$. This procedure continues until we find a word-length that satisfies the refinement relation. This way we locally optimize the word-length of x_3. The result is that $x_{3n} = 6$.

Now consider the system the weighted moving average filter shown in Figure 2b with coefficients $a = 0.2$, $b = 0.6$, and $c = 0.2$. Using a similar approach as described, we compute the system-level error from subsystem specifications. Our contract-based methodology yields a maximum error of 0.769, while the enumeration of all input combinations yields 0.688. Our obtained bound is pessimistic because each contract considers the worst-case scenario, which might not occur at the same time. On the other hand, its computation is vastly more tractable.

These examples illustrate that we can use contract operations to obtain upper bounds for variable errors without enumerating all input combinations, which is crucial for performing optimization with many iterations. In other words, Pacti can leverage contract-based design for combining formal methods with optimization to reason about fixed-point representations in digital signal processing system design.

7.2 Specification-based synthetic biology

In this case study, we explore the application of Pacti to aid the design of synthetic biological systems. The field of synthetic biology concerns the engineering of biological systems such as bacteria, yeast, plant cells, and mammalian cells to achieve desired behavior. Commonly, scientists genetically engineer biological systems to produce or control proteins that execute desired actuation and control tasks. We refer to these biological circuits as "biocircuits." Current experimental design approaches in synthetic biology are largely heuristics and trial-and-error based, hence difficult to scale. To alleviate this, various modeling and analysis [41] and logic gate design tools [7] have been proposed, but the successful design of large biological circuits remains challenging. We believe that the Pacti framework holds the potential to address this issue of scale due to its generality in expressing system specifications and in decoupling the modeling details of components from their specifications. Towards that end, we consider a biocircuit case study where we synthesize the specification of a genetic circuit and predict the system performance using this specification.

Consider the system schematic shown in Figure 3a, where we are interested in measuring the level of a protein expressed by a plant under control of bacteria that live near its roots. To design such a complex system, many parts

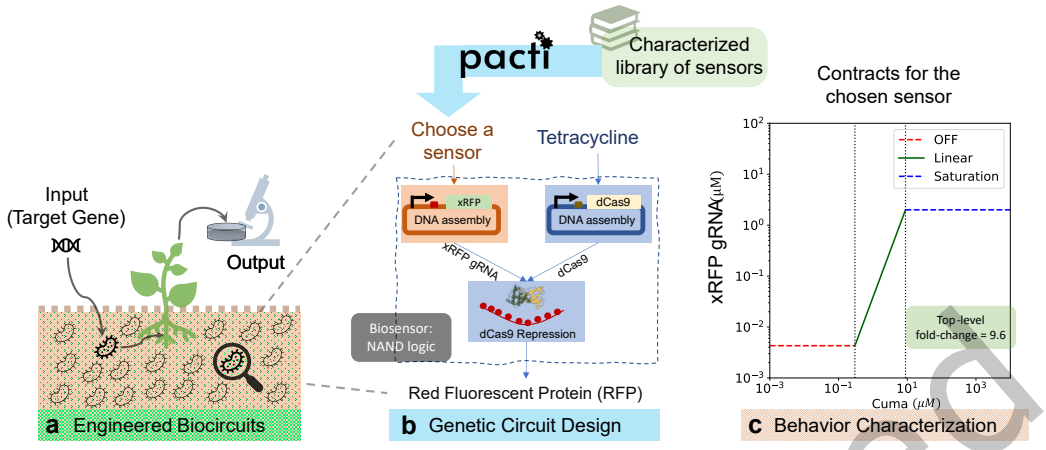


Fig. 3. **Specification-based synthetic biology using Pacti**. (a) The goal with the biocircuit design is to measure the plant protein expression under control of engineered bacteria near its roots. (b) The bacterial cell is engineered as a NAND logic gate. (c) Using Pacti, we can choose a sensor from a characterized library to achieve the highest fold-change for the NAND logic gate. Three contracts that model the specification of the chosen sensor are shown.

need to be designed in different laboratories and then integrated together. This is analogous to electronic circuit design where original equipment manufacturers synthesize their components according to set specifications and then product integration companies use these components to design a larger system. Hence, this system design aligns well with the contract-based design approach with Pacti. For the biocircuit that we consider in this case study, a desired specification has been given by the integration lab for the bacterial subsystem. So, in this case study, we only focus on the bacterial control logic subsystem—a biological NAND logic gate.

This system has three subsystems, as shown in Figure 3b. Two of the three subsystems can be chosen from a library of sensors that we have constructed using experimental data. We use the *quotient* operation on contracts to synthesize a specification of the missing third subsystem that needs to be designed. Further, once a nominal set of three subsystems have been chosen, we use contract *composition* as part of an optimization strategy to predict the maximum system fold-change, defined as the ratio between the on and off levels of the system's output. We say that an input or an output is "ON" when its level is higher than a minimum threshold, and is "OFF" when its level is lower than a maximum threshold. We formalize these notions with polyhedral constraints in the contract descriptions.

The three subsystems in the NAND gate are 1) a sensor with aTc input that outputs a dCas9 protein, 2) a sensor with Sal input that outputs the xRFP-gRNA protein, and 3) a repressor subsystem, which takes as inputs the xRFP-gRNA and the dCas9 proteins. When both inputs to this repressor subsystem are ON, it suppresses the output, RFP, a red fluorescent protein. RFP is also the output of the top-level system. RFP is ON only when either of the sensors inputs are OFF. In this way, the system behavior is that of a NAND logic gate. We denote the contract for the sensors as C_{aTc} and C_{Sal}, while the contract for repressor as C_{dCas9}. Our first task is to construct a library of sensors from the experimental data to write the contracts for C_{aTc} and C_{Sal}. Then, we find the missing subsystem contract (C_{dCas9}) using the quotient of the top-level system contract from the composition of the two sensors. Finally, for a given implementation of C_{dCas9} and for a fixed choice of subsystem 1, C_{aTc}, we aim to find the second sensor (choose spec for subsystem 2) that maximizes the top-level system fold-change.

Modeling the specifications and constructing a library of sensors. First we represent a set of sensors using contracts. We build a library of sensor contracts using the experimental data for the sensors in the *Marionette* bacterial cell strain [37]. We model the sensor behavior as assume-guarantee contracts. We observe that each

sensor has three characteristic behaviors: 1) the off state, where the output stays close to zero (the non-zero expression in this state is termed as "leaky response"), 2) the linear rate of output, where the output responds linearly to the input (in log scale), and 3) the saturation state, where the output saturates to a maximum constant value. The three contracts for a sensor, s, with input u and output y are

$$C_s^{\text{off}} = (u \leq u_{\text{start}}, y \leq y_{\text{leak}}),\ C_s^{\text{sat}} = (u \geq u_{\text{K}}, y \geq y_{\text{max}}),\ \text{and}\ C_s^{\text{lin}} = (u_{\text{start}} \leq u \leq u_{\text{K}}, y = mu + b),$$

where m and b are constants. The subscripts "start" and "K" denote the input threshold values of the end of the OFF behavior and the linear regime, respectively, whereas "leak" and "max" denote the leaky and the maximum values of the output. The "off", "lin", and "sat" superscripts represent the off, linear, and saturated viewpoints. Figure 3c shows the three contracts of one of the sensors constructed from the experimental data.

Contract quotient to find the specifications of missing parts. Suppose that we have chosen the two sensors (subsystems 1 and 2), and we are also given a desired top-level system contract C_{sys} that the system must meet. We use the quotient in Pacti to find the specification of the missing object: the dCas9 repression mechanism (subsystem 3). For example, with sensors "Sal" and "aTc", we have

$$C_{\text{Sal}}^{\text{lin}} = (0.9 \leq \text{Sal} \leq 43.0,\ \ 0.03\,\text{Sal} - \text{xRFP} + 0.02 = 0),\ \text{and}$$
$$C_{\text{aTc}}^{\text{lin}} = (0.0018 \leq \text{aTc} \leq 0.013,\ \ 88.84\,\text{aTc} - \text{dCas9} + 0.15 = 0).$$

For the top-level system, we have

$$C_{\text{sys}}^{\text{lin}} = (0.909 \leq \text{Sal} \leq 42.57 \wedge 0.0018 \leq \text{aTc} \leq 0.012, \text{RFP} + \text{Sal} + \text{aTc} \leq 1.29).$$

Using the quotient, we have that $C_{\text{dCas9}}^{\text{lin}} = C_{\text{sys}}^{\text{lin}} / (C_{\text{Sal}}^{\text{lin}} \parallel C_{\text{aTc}}^{\text{lin}})$, which Pacti computes as

$$C_{\text{dCas9}}^{\text{lin}} = (0.05 \leq \text{xRFP} \leq 1.33 \wedge 0.31 \leq \text{dCas9} \leq 1.29, \text{RFP} + 0.01\,\text{dCas9} + 32.5\,\text{xRFP} \leq 1.29).$$

This is the contract for the dCas9 mechanism when it represses the RFP level. For brevity, we have only shown the condition when both sensors are switched on in the linear regime. The resulting contract for subsystem 3 guarantees that it represses the RFP level dependent on its inputs, xRFP and dCas9. We can provide this missing-component contract to an expert for independent implementation. Further, using this synthesized component contract, we can optimize the choice of other components in the system.

Contract composition and sensor selection. From the library of 14 sensors that we constructed, we choose and fix the aTc sensor for our subsystem 1. With the synthesized contract for C_{dCas9} available, we can go through the library of remaining 13 sensors to choose the sensing behavior for subsystem 2 that maximizes the top-level fold change (as illustrated in Figure 3b). The contract for aTc, C_{aTc}, is given in three viewpoints as described above by using $K = 0.013, y_{\text{max}} = 1, \text{start} = 0.0018, y_{\text{leak}} = 4.9 \times 10^{-3}, m = 88.84, b = -0.15$. For subsystem 3, we write the dCas9 repression mechanism contract as $C_{\text{dCas9}}^{\text{lin}} = (0.3 \leq \text{xRFP} \leq 1 \wedge 0.1 \leq \text{dCas9} \leq 0.8,\ \ \text{RFP} + 2\,\text{xRFP} + 0.1\,\text{dCas9} \leq 5)$, with similar contracts for other viewpoints.

For each sensor in the library with contract C_{s_i}, we use Pacti to compute the system-level contract by composing the chosen sensor contract with the available subsystem contracts: $C_{\text{sys}} = C_{\text{aTc}} \parallel C_{s_i} \parallel C_{\text{dCas9}}$. When computing this composition for some sensors in the library, Pacti returns the error "unsatisfiable in the given context." This means that the guarantees of this sensor are insufficient to meet the assumptions of the component to which it drives outputs (the dCas9 repressor subsystem). Thus, Pacti allows us to identify potential design errors. In order to choose a sensor among those that yield valid compositions, we use the fold change of the system $F = \text{RFP}_{\text{on}}/\text{RFP}_{\text{off}}$ as a performance criterion and select the sensor that maximizes this number. The final chosen sensor that achieves the highest fold-change for the NAND gate is the "Cuma" sensor shown in Figure 3c. This designer-friendly interface of Pacti for contract algebra is crucial for biologists. An experimental validation of these predictions is an ongoing research endeavor [40].

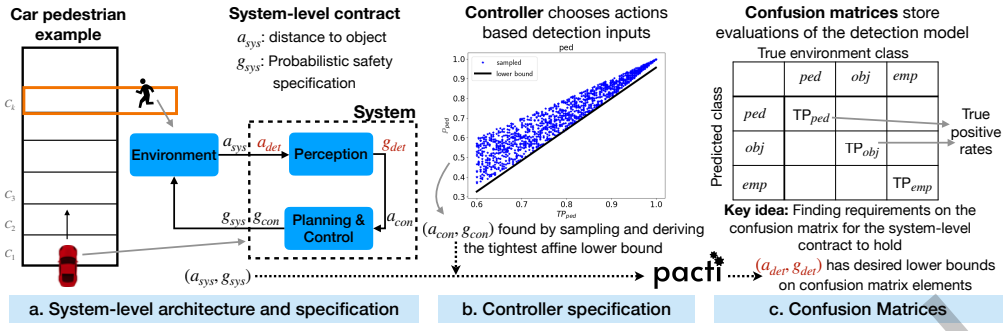


Fig. 4. Given a system-level specification $C_{\text{sys}} = (a_{\text{sys}}, g_{\text{sys}})$, and a specification for the controller $C_{\text{con}} = (a_{\text{con}}, g_{\text{con}})$, derive the object detection specification $C_{\text{det}} = (a_{\text{det}}, g_{\text{det}})$.

Note that although some design tools exist for synthetic biocircuits (like Cello [7]), none of the existing approaches exhibit the design features that we have demonstrated here with Pacti. Cello can synthesize large logic gate circuit designs but cannot be applied to design dynamical systems with analog behavior. On the other hand, the specification formalism with Pacti gives a framework for multiple viewpoints that can be used to model timing specifications along with functional objectives. More significantly, the ability to synthesize missing parts in a given design and find incompatible components are novel design features that Pacti imparts for the design of biocircuits.

7.3 Evaluating Perception in a System-level Safety-Critical Context

Compositional reasoning is crucial to scaling up formal methods to learning-enabled autonomous systems [43]. In this section, we introduce a case study on the design of an autonomous vehicle that must satisfy a safety property with a given probability. We abstract the vehicle as a system comprising two subsystems: a perception component (for object detection) and a controller, as shown in Figure 4a. Given a system-level safety contract and the specification of the control component, the *quotient* operator is used to derive a specification for the perception component.

Car-Pedestrian Example. Figure 4a shows the situation we are considering: a vehicle approaches a crosswalk and must act safely. If a pedestrian is on the crosswalk, the car is required to stop at the crosswalk. Otherwise, the car must keep driving and not stop at any point. The perception component must correctly detect the object at the crosswalk, which can belong to three classes: a pedestrian (denoted *ped*), an object that is not a pedestrian (denoted *obj*), and the background or empty class (denoted *emp*).

We encode the notion of system-level safety in linear temporal logic formulas φ_c, where $c \in \{ped, obj, emp\}$. This way, we can specify safe behavior when an element of each class is present on the crosswalk. We synthesized controllers to satisfy these safety properties, assuming perfect perception. The details of the properties and our synthesis approach can be found in [1].

System-level contract. Let $\mathbb{P}_c$ be the probability that the car will satisfy requirement φ_c when the crosswalk object has true class c. We set the system-level contract to

$$C_{\text{sys}} = (d_l \leq d \leq d_u,\ g_{ped} \wedge g_{obj} \wedge g_{emp}),$$

where d_l, d_u are bounds on the distance d to the object in the crosswalk, and g_c characterizes an affine lower bound of $\mathbb{P}_c$. This system-level contract assumes bounded distance to the object of interest, and guarantees affine lower bounds (as a function of d) on probabilistic satisfaction of safety properties. In other words, at the

system-level we allow the probability of satisfaction of the safety property to degrade if the vehicle is far away from the crosswalk.

Controller contract. As mentioned, we synthesize three controllers, each making sure that property φ_c would be satisfied under perfect perception. In order to write a contract for each of the controllers, we make use of the fact that the perception component is not perfect. As a result, the controller satisfies its safety specification probabilistically.

To correlate probabilities of property satisfaction to perception errors, we base our approach on [1, 2]. The satisfaction probability $\mathbb{P}_c$ for the safety property φ_c is computed by constructing a Markov chain with transition probabilities derived from the *true positive rates*[4] of the perception component, and then invoking standard statistical model-checking tools. For this example, $\mathbb{P}_c$ depends mainly on the true positive rate TP_c of the class c. We determine a tight affine lower bound for $\mathbb{P}_c$ as a function of TP_c by sampling and solving a linear program. The data for the linear program is generated by sampling false negatives for each value of TP_c and computing the corresponding $\mathbb{P}_c$ (see Figure 4b). This procedure yields the following controller contract corresponding to each object class c:

$$C_{\mathrm{c}} = (l_c \leq \mathsf{TP}_c,\ a_c(\mathsf{TP}_c) + b_c \leq \mathbb{P}_c),$$

where l_c, a_c, and b_c are reals. The three contracts are composed to find the overall control contract: $C_{\mathrm{con}} = C_{ped} \parallel C_{obj} \parallel C_{emp}$.

Object detection contract. Now that we have the specifications for the system and for the three controllers, we use contract operations to obtain the specification of the perception component. The detection component contract is found via the quotient $C_{\mathrm{det}} = C_{\mathrm{sys}}/C_{\mathrm{con}}$, where C_{sys} is the system-level contract and C_{con} is the controller contract. C_{det} imposes lower bounds on the true positive rates TP_c of each object class c. We illustrate the results numerically for an instance of the car-pedestrian example. The system contract is set to

$$C_{sys} = (1 \leq d \leq 10,\ 0.99(1 - 0.1d) \leq \mathbb{P}_{ped} \wedge 0.8(1 - 0.1d) \leq \mathbb{P}_{obj} \wedge 0.95(1 - 0.1d) \leq \mathbb{P}_{emp}),$$

i.e., the contract assumes the distance to the crosswalk is bounded between 1 and 10 units, and specifies desired system-level probabilities $\mathbb{P}_c$ as a function of distance d. The controller contracts are computed to be $C_{ped} = (0.6 \leq \mathsf{TP}_{ped},\ 1.58\mathsf{TP}_{ped} - 0.622 \leq \mathbb{P}_{ped})$, $C_{obj} = (0.3 \leq \mathsf{TP}_{obj},\ 0.068\mathsf{TP}_{obj} + 0.93 \leq \mathbb{P}_{obj})$, and $C_{emp} = (0.6 \leq \mathsf{TP}_{emp},\ 0.2\mathsf{TP}_{emp} + 0.799 \leq \mathbb{P}_{emp})$. These contracts impose affine lower bounds on $\mathbb{P}_c$ with respect to the true positive rates TP_c. The quotient results in an object detection contract with true positive rates lower bounded by affine functions of the distance d:

$$C_{\mathrm{det}} = \big(1 \leq d \leq 10,\ (1.02 - 0.063d \leq \mathsf{TP}_{ped}) \wedge (0.6 \leq \mathsf{TP}_{ped}) \wedge (0.3 \leq \mathsf{TP}_{obj}) \wedge (0.6 \leq \mathsf{TP}_{emp})\big).$$

Now that we have obtained the contract for the perception component, we can give this contract to designers responsible for object detection. The designers can develop the perception component and verify that it satisfies the requirements on true positive bounds as in C_{det}. If it does, we can infer that the overall system with controller designed according to C_{con} will satisfy the system-level requirements.

Extensions. Finally, we discuss extensions of this problem when system requirements or detection requirements are partially specified. We will consider three cases. The first case corresponds to the situation when when we add to the system a new set of requirements that have to be satisfied under the same environments as the existing ones. This happens, for instance, when we require the system to satisfy a certain safety specification (e.g., the car shall not collide with obstacles) and then we also ask it to satisfy a liveness specification (e.g., the car shall reach its destination). The second case corresponds to the addition of requirements to the top-level system when these requirements can be met under different environments from those of the existing requirements. For instance,

[4]The true positive rate of a perception component for an object class is defined as the probability that the component correctly detects an object to be of that class.

suppose that the original system contract promises certain functionality under the assumption of low-visibility conditions, and we add a new contract that makes promises under the assumption of high-visibility conditions. In this case, the environments "low-visibility" and "high-visibility" cannot hold simultaneously. Formally, the difference between the first case and the second is that we have to use contract merging to fuse the viewpoints in the first case, and we have to use the binary operation of *conjunction*, denoted $\wedge$, to address the second—see Chapter 5 of [23] for an extended discussion of these two operations. Finally, the third case corresponds to having access to a partial specification of the missing component. We will need the following proposition in our discussion.

Proposition 7.1. *The quotient operator distributes over conjunction and over merging.*

Proof. Given contracts C_1, C_2, and C_3, we have $(C_1 \wedge C_2)/C_3 = (C_1 \wedge C_2) \bullet C_3^{-1} = (C_1 \bullet C_3^{-1}) \wedge (C_2 \bullet C_3^{-1}) = (C_1/C_3) \wedge (C_2/C_3)$, where the first equality is the *preordered heap* identity of contracts (see Section 3.1 of [27]) and the second comes from Proposition 6.10.4 of [23].

Similarly, $(C_1 \bullet C_2)/C_3 = (C_1 \bullet C_2) \bullet C_3^{-1} = (C_1 \bullet C_3^{-1}) \bullet (C_2 \bullet C_3^{-1}) = (C_1/C_3) \bullet (C_2/C_3)$, where the second equality comes from the the fact that merging is idempotent and commutative. □

Case 1: Up to now, we assumed we had a single top-level system contract and three contracts for the controllers—one for each. From this data, we were able to identify the specification of the object detection component. Now suppose that we have computed the detection contract and that our system-level requirements change. How should we adjust the detection component in light of new system-level requirements? Let $C_{\text{sys}}^{\text{orig}}$ be the original system-level contract, and C_{con} the overall controller contract. The detection contract based on the original system requirements is $C_{\text{det}}^{\text{orig}} = C_{\text{sys}}^{\text{orig}}/C_{\text{con}}$. Suppose the top-level system is now required to satisfy a new set of requirements on top of the requirements we had before; moreover, we assume that these new requirements have to be satisfied under the same environments, e.g., we add safety requirements to liveness requirements. We can express the new system-level contract as $C_{\text{sys}} = C_{\text{sys}}^{\text{add}} \bullet C_{\text{sys}}^{\text{orig}}$. The revised detection contract is $C_{\text{det}}^{\text{new}} = (C_{\text{sys}}^{\text{orig}} \bullet C_{\text{sys}}^{\text{add}})/C_{\text{con}} = C_{\text{det}}^{\text{orig}} \bullet (C_{\text{sys}}^{\text{add}}/C_{\text{con}})$, per Proposition 7.1. In other words, the new detection contract can be obtained by merging the original detection contract with the quotient corresponding to the new system requirements.

Case 2: Suppose we add to the original system requirements a new set of requirements that do not necessarily have to be satisfied for the same environments as the existing ones. That is, we add the new requirements via conjunction. We thus have $C_{\text{sys}}^{\text{new}} = C_{\text{sys}}^{\text{orig}} \wedge C_{\text{sys}}^{\text{add}}$, and the new detection contract is $C_{\text{det}}^{\text{new}} = (C_{\text{sys}}^{\text{orig}} \wedge C_{\text{sys}}^{\text{add}})/C_{\text{con}} = C_{\text{det}}^{\text{orig}} \wedge (C_{\text{sys}}^{\text{add}}/C_{\text{con}})$, per Proposition 7.1.

Case 3: Suppose that we have the system-level contract $C_{\text{sys}}^{\text{orig}}$ and the controller contract C_{con} and that the detection contract is partially specified as $C_{\text{det}}^{\text{known}}$. This partial specification of the detection contract can represent the requirements on the detection model on a specific set of scenarios, e.g., confusion matrices when the visibility is high. However, this specification may be insufficient to satisfy the requirements of the system, which may make promises under the assumption of low-visibility, for instance. The question is, what is the most relaxed contract that we have to conjoin with $C_{\text{det}}^{\text{known}}$ such that the result is a missing component specification? In other words, we want to obtain the most relaxed specification that solves the problem $? \wedge C_{\text{det}}^{\text{known}} \leq C_{\text{sys}}^{\text{orig}}/C_{\text{con}}$. The answer to this problem is given in closed form by $C_{\text{det}}^{\text{known}} \rightarrow (C_{\text{sys}}^{\text{orig}}/C_{\text{con}})$, where $\rightarrow$ is the implication operator of contracts, introduced in Section 6.8.3 of [23].

8 PERFORMANCE

One way to get insight into the efficiency of Pacti's routines is to compare Pacti with existing software. As far as we know, there does not exist software that computes contract specifications at the interfaces of components. As a result, we ran benchmarks of Pacti using the case studies described in Section 7. We added lightweight instrumentation to the algorithms presented in Section 3 in order to track the number of operation invocations and the minimum and maximum sizes of the operations' arguments, with contract size defined as the tuple of the number of unique variables (inputs and outputs) and the number of constraints (assumptions and guarantees). We executed repeatedly the case studies described in Section 7, varying the constants in the contract definitions. We ran parallel versions of the case studies on a workstation with an AMD Ryzen Threadripper PRO 3955WX 16-Cores @ 3.8927 GHz and up to 32 threads.

The signal processing case study (see Section 7.1) ran in 14.3 seconds, invoking a total of 51,027 composition operations. The smallest input contracts had 3 constraints and 2 variables, and the largest had 7 constraints and 6 variables. The biocircuits case study (see Section 7.2) ran in 4.4 seconds, invoking a total of 82,428 composition operations. The smallest contract had 2 constraints and 2 variables, and the largest had 6 constraints and 6 variables. The autonomy case study (see Section 7.3) ran in 8.5 seconds, invoking a total of 18,750 quotient operations, where the smallest contract had 5 constraints and 4 variables and the largest had 9 constraints and 6 variables; and a total of 39,720 merge operations where the smallest contract had 3 constraints and 2 variables and the largest had 6 constraints and 4 variables.

To understand the growth of execution time with the size of contracts, we carried out an additional experiment. We defined the following initial contract:

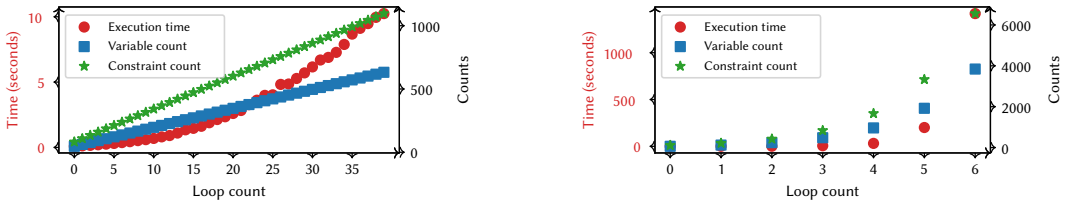
```
Inputs : [dt1,dt2,dt3,dt4,t_in,trtd_in]
Outputs: [err1_res,ertd2_res,dv3_res,t1,t2,t3,t_out,trtd1_res,trtd2_res,trtd3_res,trtd_out]
A: [ dt1 >= 0 & dt2 >= 0 & dt3 >= 0 & dt4 >= 0 & t_in >= 0 ]
G: [ t1 = t_in + dt1 & t2 = t1 + dt2 & t3 = t2 + dt3 & t_out = t3 + dt4 &
     trtd_in = trtd1_res = trtd2_res = trtd3_res &
     0.9 dv3_res <= trtd3_res - trtd_out <= 1.1 dv3_res &
     | err1_res | <= 0.005 trtd_in & ertd2_res = trtd1_res + err1_res &
     dv3_res <= 5 & 0.2 ertd2_res <= dv3_res <= 0.3 ertd2_res ]
```

We replicated this contract several times in such a way that the inputs of one contract matched the outputs of the next one, yielding a sequence of contracts that we composed into an overall contract. We experimented with two strategies for constructing the overall contract. First, in a sequential strategy, we iteratively composed a fixed-sized contract with a growing contract, the result of the previous iteration. This strategy roughly added a constant number of variables and constraints with each composition. Figure 5a shows the execution time and the number of variables and constraints involved in each contract composition when performing this experiment. Second, in a geometric strategy, we composed a contract with a copy of itself with variables renamed such that the inputs of the new contract correspond to the outputs of the original contract. This strategy roughly doubled the number of variables and constraints with each composition. Figure 5b shows the results of this experiment. In both experiments, the resulting contracts involved similar ratios of 1, 2, and 3 variable constraints: 16%, 38%, and 46% respectively, resulting in a matrix density of less than 5% initially down to 0.3% for the largest geometric construction.[5]

9 DISCUSSION AND CONCLUDING REMARKS

This paper introduced principles and algorithms to implement a contract-based design tool at scale. In doing this, we identified the problems of variable elimination via antecedent and consequent synthesis as relevant to the computation of specifications for requirement engineering. We provided efficient algorithms for the solutions

[5]Experiment details available here: https://github.com/pacti-org/cs-space-mission/tree/main/performance_benchmarking



(a) Linear construction (b) Geometric construction

Fig. 5. Execution time when composing contracts of various sizes

of these problems. We also presented Pacti as a tool to ease the broader use of assume-guarantee reasoning for system analysis and design. Currently, Pacti supports the verification of refinement and the operations of composition, quotient, and strong merging. The program supports specifications written as polyhedral constraints. Our results indicate that Pacti can efficiently compute contract operations, making it a suitable tool for early design-space exploration and trade-off analysis of complex systems. Our objective is to develop a platform that can grow over time as more features are implemented. Absent from the current implementation are support for temporal logic and hypercontracts [26], as well as integration with other modeling frameworks and languages (e.g., [21]). We plan to address them in the future.

ACKNOWLEDGMENTS

This work was supported by NSF and ASEE through an eFellows postdoctoral fellowship; by the DARPA LOGiCS project under contract FA8750-20-C-0156; by the Air Force Office of Scientific Research (AFOSR) under MURI grant FA9550-22-1-0316 and grant FA9550-22-1-0333; by Toyota under the iCyPhy Center at UC Berkeley, and by the Wallenberg AI Autonomous Systems and Software Program (WASP). During the completion of this work, I. Incer was with the California Institute of Technology and the University of California, Berkeley; A. Badithela and A. Pandey were with the California Institute of Technology.

REFERENCES

[1] Apurva Badithela, Tichakorn Wongpiromsarn, and Richard M. Murray. 2021. Leveraging Classification Metrics for Quantitative System-Level Analysis with Temporal Logic Specifications. In *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, Austin, TX, USA (virtual), 564–571. https://doi.org/10.1109/cdc45484.2021.9683611

[2] Apurva Badithela, Tichakorn Wongpiromsarn, and Richard M. Murray. 2023. Evaluation Metrics of Object Detection for Quantitative System-Level Analysis of Safety-Critical Autonomous Systems. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Detroit, MI, USA, To Appear.

[3] Albert Benveniste, Benoit Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. 2008. Multiple Viewpoint Contract-Based Specification and Design. In *Formal Methods for Components and Objects: 6th International Symposium, FMCO 2007, Amsterdam, The Netherlands, October 24-26, 2007, Revised Lectures*, Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 200–225. https://doi.org/10.1007/978-3-540-92188-2_9

[4] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. 2018. Contracts for System Design. *Foundations and Trends® in Electronic Design Automation* 12, 2-3 (2018), 124–400. https://doi.org/10.1561/1000000053

[5] Stephen P. Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press, New York, NY, USA.

[6] Marc-André Cantin, Yvon Savaria, and Pierre Lavoie. 2002. A comparison of automatic word length optimization procedures. In *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, Vol. 2. IEEE, Phoenix-Scottsdale, AZ, USA, II–II. https://doi.org/10.1109/ISCAS.2002.1011427

[7] Jeronimo Cello, Aniko V Paul, and Eckard Wimmer. 2002. Chemical synthesis of poliovirus cDNA: generation of infectious virus in the absence of natural template. *Science* 297, 5583 (2002), 1016–1018.

[8] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. 2002. Synchronous and Bidirectional Component Interfaces. In *Computer Aided Verification*, Ed Brinksma and Kim Guldstrand Larsen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 414–427. https://doi.org/10.1007/3-540-45657-0_34

[9] Vijay Chandru. 1993. Variable elimination in linear constraints. *Comput. J.* 36, 5 (1993), 463–472.

[10] Alessandro Cimatti, Michele Dorigatti, and Stefano Tonetta. 2013. OCRA: A tool for checking the refinement of temporal contracts. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Palo Alto, CA, USA, 702–705. https://doi.org/10.1109/ase.2013.6693137

[11] Darren Cofer, Andrew Gacek, Steven Miller, Michael W. Whalen, Brian LaValley, and Lui Sha. 2012. Compositional Verification of Architectural Models. In *NASA Formal Methods*, Alwyn E. Goodloe and Suzette Person (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 126–140. https://doi.org/10.1007/978-3-642-28891-3_13

[12] George A. Constantinides, Peter Y.K. Cheung, and Wayne Luk. 2003. Wordlength optimization for linear digital signal processing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, 10 (2003), 1432–1442. https://doi.org/10.1109/TCAD.2003.818119

[13] Arlen Cox, Sriram Sankaranarayanan, and Bor-Yuh Evan Chang. 2012. A Bit Too Precise? Bounded Verification of Quantized Digital Filters. In *Tools and Algorithms for the Construction and Analysis of Systems*, Cormac Flanagan and Barbara König (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 33–47. https://doi.org/10.1007/978-3-642-28756-5_4

[14] George B. Dantzig and B. Curtis Eaves. 1973. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory, Series A* 14, 3 (1973), 288–297. https://doi.org/10.1016/0097-3165(73)90004-6

[15] Luca de Alfaro and Thomas A. Henzinger. 2001. Interface Automata. *SIGSOFT Softw. Eng. Notes* 26, 5 (Sept. 2001), 109–120. https://doi.org/10.1145/503271.503226

[16] Lloyd L. Dines. 1919. Systems of linear inequalities. *Annals of Mathematics* (1919), 191–199.

[17] Richard J. Duffin. 1974. On Fourier's analysis of linear inequality systems. In *Pivoting and Extension: In honor of A.W. Tucker*, M. L. Balinski (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 71–95. https://doi.org/10.1007/BFb0121242

[18] B. Curtis Eaves and Robert M. Freund. 1982. Optimal scaling of balls and polyhedra. *Mathematical Programming* 23, 1 (1982), 138–147. https://doi.org/10.1007/bf01583784

[19] Claire F. Fang, Rob A. Rutenbar, and Tsuhan Chen. 2003. Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs. In *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*. IEEE, San Jose, CA, USA, 275–282. https://doi.org/10.1109/ICCAD.2003.159701

[20] Joseph Fourier. 1826. Solution d'une question particulière du calcul des inégalités. *Nouveau Bulletin des sciences par la Société philomathique de Paris, p. 99* (1826), 317–319.

[21] Daniel J. Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2022. Scenic: A Language for Scenario Specification and Data Generation. *Machine Learning* 112, 10 (2022), 3805–3849. https://doi.org/10.1007/s10994-021-06120-5

[22] Jean-Louis Imbert. 1993. Fourier's elimination: Which to choose?. In *Principles and Practices of Constraint Programming*, Vol. 1. Citeseer, Newport, Rhode Island, USA, 117–129.

[23] Inigo Incer. 2022. *The Algebra of Contracts*. Ph. D. Dissertation. EECS Department, University of California, Berkeley.

[24] Inigo Incer, Apurva Badithela, Josefine Graebener, Piergiuseppe Mallozzi, Ayush Pandey, Sheng-Jung Yu, Albert Benveniste, Benoit Caillaud, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2023. Pacti: Scaling Assume-Guarantee Reasoning for System Analysis and Design. arXiv:2303.17751

[25] Inigo Incer, Albert Benveniste, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2023. Context-Aided Variable Elimination for Requirement Engineering. arXiv:2305.17596

[26] Inigo Incer, Albert Benveniste, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2022. Hypercontracts. In *Proceedings of the 14th NASA Formal Methods Symposium (NFM)*. Springer International Publishing, Cham, 674–692. https://doi.org/10.1007/978-3-031-06773-0_36

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

[28] Inigo Incer, Alberto L. Sangiovanni-Vincentelli, Chung-Wei Lin, and Eunsuk Kang. 2018. Quotient for Assume-Guarantee Contracts. In *16th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE'18)*. ACM, Beijing, China, 67–77. https://doi.org/10.1109/MEMCOD.2018.8556872

[29] Mohammad Kaykobad. 1985. Positive solutions of positive linear systems. *Linear Algebra Appl.* 64 (1985), 133–140. https://doi.org/10.1016/0024-3795(85)90271-X

[30] Holger Keding, Markus Willems, Martin Coors, and Heinrich Meyr. 1998. FRIDGE: a fixed-point design and simulation environment. In *Proceedings Design, Automation and Test in Europe*. IEEE, Paris, France, USA, 429–435. https://doi.org/10.1109/DATE.1998.655893

[31] Seehyun Kim, Ki-Il Kum, and Wonyong Sung. 1998. Fixed-point optimization utility for C and C++ based digital signal processing programs. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 45, 11 (1998), 1455–1464. https://doi.org/10.1109/82.735357

[32] Jean-Louis Lassez and Michael J Maher. 1992. On Fourier's algorithm for linear arithmetic constraints. *Journal of Automated Reasoning* 9 (1992), 373–379.
[33] E.A. Lee and A. Sangiovanni-Vincentelli. 1998. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17, 12 (1998), 1217–1229. https://doi.org/10.1109/43.736561
[34] Nancy A. Lynch and Mark R. Tuttle. 1989. An introduction to input/output automata. *CWI Quarterly* 2 (1989), 219–246.
[35] Piergiuseppe Mallozzi, Pierluigi Nuzzo, Patrizio Pelliccione, and Gerardo Schneider. 2020. Crome: contract-based robotic mission specification. In *2020 18th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. IEEE, Jaipur, India (virtual), 1–11. https://doi.org/10.1109/memocode51338.2020.9315065
[36] Alexandre Maréchal and Michaël Périn. 2017. Efficient Elimination of Redundancies in Polyhedra by Raytracing. In *Verification, Model Checking, and Abstract Interpretation*, Ahmed Bouajjani and David Monniaux (Eds.). Springer International Publishing, Cham, 367–385. https://doi.org/10.1007/978-3-319-52234-0_20
[37] Adam J. Meyer, Thomas H. Segall-Shapiro, Emerson Glassey, Jing Zhang, and Christopher A. Voigt. 2019. Escherichia coli "Marionette" strains with 12 highly optimized small-molecule sensors. *Nature Chemical Biology* 15, 2 (2019), 196–204. https://doi.org/10.1038/s41589-018-0168-3
[38] Theodore Samuel Motzkin. 1936. *Beiträge zur Theorie der linearen Ungleichungen.* Ph. D. Dissertation. University of Basel.
[39] Pierluigi Nuzzo, Michele Lora, Yishai A. Feldman, and Alberto L. Sangiovanni-Vincentelli. 2018. CHASE: Contract-based requirement engineering for cyber-physical system design. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Dresden, Germany, 839–844. https://doi.org/10.23919/DATE.2018.8342122
[40] Ayush Pandey. 2024. *Modeling Frameworks for Modular and Scalable Biological Circuit Design.* Ph. D. Dissertation. California Institute of Technology.
[41] Ayush Pandey, Makena L. Rodriguez, William Poole, and Richard M Murray. 2023. Characterization of integrase and excisionase activity in a cell-free protein expression system using a modeling and analysis pipeline. *ACS Synthetic Biology* 12, 2 (2023), 511–523.
[42] Roberto Passerone, Inigo Incer, and Alberto L. Sangiovanni-Vincentelli. 2019. Coherent Extension, Composition, and Merging Operators in Contract Models for System Design. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 86 (Oct. 2019), 23 pages. https://doi.org/10.1145/3358216
[43] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. 2022. Toward Verified Artificial Intelligence. *Commun. ACM* 65, 7 (2022), 46–55.
[44] Changchun Shi and Robert W Brodersen. 2004. A perturbation theory on statistical quantization effects in fixed-point DSP with non-stationary inputs. In *2004 IEEE International Symposium on Circuits and Systems (ISCAS)*, Vol. 3. IEEE, Vancouver, BC, CA, III–373.
[45] Stella Simić, Omar Inverso, and Mirco Tribastone. 2021. Bit-Precise Verification of Discontinuity Errors Under Fixed-Point Arithmetic. In *Software Engineering and Formal Methods*, Radu Calinescu and Corina S. Păsăreanu (Eds.). Springer International Publishing, Cham, 443–460. https://doi.org/10.1007/978-3-030-92124-8_25
[46] Wonyong Sung and Ki-Il Kum. 1995. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Transactions on Signal Processing* 43, 12 (1995), 3087–3090. https://doi.org/10.1109/78.476465
[47] Jan Telgen. 1983. Identifying Redundant Constraints and Implicit Equalities in Systems of Linear Constraints. *Manage. Sci.* 29, 10 (oct 1983), 1209–1222. https://doi.org/10.1287/mnsc.29.10.1209

Received 8 September 2023; revised 30 June 2024; accepted 26 October 2024