Tightening a Contract Refinement

Alessandro Cimatti, Ramiro Demasi, and Stefano Tonetta

Fondazione Bruno Kessler, Trento, Italy
{cimatti,demasi,tonettas}@fbk.eu

Abstract. Contract-based design is an emerging paradigm for correct-by-construction hierarchical systems: components are associated with assumptions and guarantees expressed as formal properties; the architecture is analyzed by verifying that each contract of composite components is correctly refined by the contracts of its subcomponents.
The approach is very efficient, because the overall correctness proof is decomposed into proofs local to each component. However, part of the complexity is delegated to the designer, who has the burden of specifying the contracts. Typical problems include understanding which contracts are necessary, and how they can be simplified without breaking the correctness of the refinement.
In this paper, we tackle these problems by proposing a new technique to understand and simplify a contract refinement. The technique, called tightening, is based on parameter synthesis. The idea is to generate a set of parametric proof obligations, where each parameter evaluation corresponds to a variant of the original contract refinement, and to search for tighter variants of the contracts that still ensure the correctness of the refinement. We cast this approach in the OCRA framework, where contracts are expressed with LTL formulas, and we evaluate its performance and effectiveness on a number of benchmarks.

1 Introduction

Embedded systems are growing in number and technical complexity. They are becoming more and more sophisticated towards open, interconnected and networked systems. Such complexity requires a rigorous analysis especially for those functions that have safety-critical requirements. Formal architectural models provide an important means to guarantee the correct refinement of system requirements along the design development and decomposition of the system.

Contract-based design, first conceived for software specification by Meyer in [20] and nowadays also applied to embedded systems (cfr. e.g., [3,22,16,15,12,14,4,2]), is an emerging paradigm for correct-by-construction systems which structures components properties into contracts. A contract specifies the properties assumed to be satisfied by the component environment (assumptions), and the properties guaranteed by the component in response (guarantees). The architecture is analyzed by verifying that each contract of composite components is correctly refined by the contracts of its subcomponents.

In the contract framework proposed in [12,13], assumptions and guarantees are specified as temporal formulas. Checking the correctness of contracts refinement is supported by generating a set of necessary and sufficient conditions. These proof obligations are temporal formulas obtained from assumptions and guarantees, which are valid if and only if the refinement is correct. The approach is implemented in the OCRA tool [8] and is parametrized by a linear-time temporal logic, either propositional LTL [21], or LTL with SMT predicates [11], or HRELTL [10,11], a variant of LTL where formulas represent sets of hybrid traces, mixing discrete- and continuous-time steps, and therefore amenable to model properties of hybrid systems. The approach has been used in several contexts and domains. A significant case study is presented in [5], where different variants of an industrial-size architectural model of a wheel braking system are analyzed, following the example outlined in the avionic AIR6110 standard.

The approach is very efficient, because the overall correctness proof is decomposed into proofs local to each component. However, part of the complexity is delegated to the designer, who has the burden of specifying the contracts. Typical problems include understanding which contracts are necessary, and how they can be simplified without breaking the correctness of the refinement.

In this paper, we tackle these problems by proposing a new technique to understand and simplify a contract refinement. The technique, called tightening, is based on parameter synthesis. The idea is to generate a set of parametric proof obligations, where each parameter evaluation corresponds to a variant of the original contract refinement, and to search for tighter variants of the contracts that still ensures the correctness of the refinement. We cast this approach in the OCRA framework and we evaluate its performance and effectiveness on a number of benchmarks, including the industrial-size architectures described in [5].

Related Work We are not aware of similar works in the context of contract-based design. The problem of contract tightening is related to vacuity checking [18] and unsatisfiability core extraction [9]. The probably most related work is the notion of unsatisfiability core for LTL proposed in [23]. However, the design problem, the formal problem, and the technical solution are very different. First, differently from the above-mentioned problems, we are not weakening/strengthening the occurrence of a subformula, but we need to weaken/strengthen all occurrences of an assumption/guarantee inside the proof obligations in the same way. Second, we do not have just one property to simplify, but every assumption/guarantee that is simplified occurs in different proof obligations; this corresponds to different unsatisfiability or model checking problems to consider at the same time. Third, we reduce the problem to a parameter synthesis problem and we ensure the monotonicity of parameters to ensure scalable results.

Also the work described in [17] addresses the problem of simplifying a contract refinement, but with a different purpose and solution: the approach relies on a library of contracts and refinement relations considered as additional inputs to the refinement check problem, and simplifies the contract refinement based on such library. The main objective of the authors is to improve the performance

of the refinement check based on the library, while we search a tighter version of the contracts that still ensure the correctness of the refinement.

Outline The remainder of the paper is structured as follows. In Section 2 we introduce some notions used throughout the paper. In Section 3, we introduce the problem of tightening a contract refinement. We present in Section 4 the main algorithm for solving such problem. We describe the experimental evaluation performed in Section 5. Finally, we discuss some conclusions and directions for further work.

2 Background

2.1 Transition Systems

Given a finite set V of variables with a (potentially infinite) domain D, we denote with $\Sigma(V)$ the set of assignments to V, i.e. mapping from V to D. A *transition system* (TS) S is a tuple $S = \langle V, I, T\rangle$, where V is a set of (state) variables, $I \subseteq \Sigma(V)$ is the set of initial states, and $T \subseteq \Sigma(V) \times \Sigma(V)$ is the set of transitions. A state $s \in \Sigma(V)$ of S is an assignment to the variables V. A trace σ of S is an infinite sequence of states $\sigma = s_0, s_1, \cdots$ such that $s_0 \in I$ and for all $i \geq 0$, $\langle s_i, s_{i+1}\rangle \in T$. Given two transition systems $S_1 = \langle V_1, I_1, T_1\rangle$ and $S_2 = \langle V_2, I_2, T_2\rangle$, we define the synchronous product $S_1 \times S_2$ as $\langle V_1 \cup V_2, I_1 \wedge I_2, T_1 \wedge T_2\rangle$. Since the product is commutative and associative, it can be generalized to a set of transitions systems.

2.2 LTL

Given a set of variables V, we assume to be given a set $Expr(V)$ of Boolean expressions over V as in [19]. In particular, in this paper we consider standard arithmetic predicates $(<, \leq, >, \geq, \ldots)$ and functions $(+, -, \ldots)$ over integer and real variables, although the proposed methods can be applied to more general settings.

We define the set of LTL formulas over the variables V with the following grammar rule:

$$\phi := p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid X\phi \mid \phi U \phi \mid \phi R \phi$$

where p ranges in $Expr(V)$. We use the following standard abbreviations: $\top := p \vee \neg p$, $\bot := \neg\top$, $\phi \rightarrow \psi := (\neg\phi) \vee \psi$, $F\phi := \top U \phi$, $G\phi := \neg F \neg\phi$.

Traces over V are infinite sequences of assignments to V. Given a trace $\sigma = s_0, s_1, \ldots$, we denote with $\sigma[i]$ the $i+1$-th state s_i and with σ^i the suffix trace starting from $s[i]$.

Given a trace σ and an LTL formula ϕ over V, we define $\sigma \models \phi$ as follows:

- $\sigma \models p$ iff p evaluates to true given the assignment $\sigma[0]$
- $\sigma \models \neg\phi$ iff $\sigma \not\models \phi$

- $\sigma \models X\phi$ iff $\sigma^1 \models \phi$
- $\sigma \models \phi U \psi$ iff there exists $i \geq 0$ s.t. $\sigma^i \models \psi$ and for all j, $0 \leq j < i$, $\sigma^j \models \phi$
- $\sigma \models \phi R \psi$ iff for all $i \geq 0$ $\sigma^i \models \psi$ or there exists j, $0 \leq j < i$, s.t. $\sigma^j \models \phi$

The satisfiability problem is the problem of checking if for a given LTL formula ϕ there exists a trace σ such that $\sigma \models \phi$.

Given a TS $S = \langle V, I, T \rangle$ and an LTL formula ϕ over $V' \subseteq V$, $M \models \phi$ if for all traces σ of M, $\sigma \models \phi$. The satisfiability problem of an LTL formula over V can be reduced to model checking by considering the universal model as transition system: i.e., ϕ is satisfiable iff $\langle V, \Sigma(V), \Sigma(V) \times \Sigma(V) \rangle \not\models \neg\phi$.

Note that we are considering in general infinite-state transition systems for which these problems are undecidable. Our methods are based on SMT-based algorithms as those implemented in nuXmv [7].

2.3 Parameter Synthesis

The goal of parameter synthesis is to find the maximal set of values for some parameters, so that a given property is satisfied. Let S be a transition system and let U be a set of parameter, we define the parametric transition system $P = (V, U, I_U, T_U)$, where I and T are now defined on both the state variables and parameters. We define the parameters as frozen, i.e., we set their value in the initial state and preserve it during the execution of the system. Given a valuation for the parameters ($\gamma \in \Sigma(U)$), and a formula ψ we write $\gamma(\psi) = \psi[u \in U/\gamma(u)]$, to indicate that each parameter has been substituted with its value. Given a parametric transition system P and a valuation for the parameters γ, we can compute the *induced* transition system, by replacing the parameters with their valuation: $P_\gamma = (V, \gamma(I_U), \gamma(T_U))$. Given an LTL property ϕ expressed over the state variables and parameters, the parameter region ρ is the maximal set of assignments to the parameters, such that the property is satisfied by every trace of the induced system, formally: $\rho = \{\gamma \mid P_\gamma \models \gamma(\phi)\}$.

In this paper, we consider Boolean parameters and, with abuse of notation, we identify a parameter evaluation γ with the set $\{p \mid p \in U, \gamma(p) = \top\}$. The parameter region is monotonic iff whenever $\gamma \subseteq \gamma'$, if $\gamma \in \rho$ then $\gamma' \in \rho$. The monotonicity of the parameter region is typically exploited by parameter synthesis algorithms that enumerate the parameter evaluations γ such that $P_\gamma \not\models \gamma(\phi)$. In fact, one can proceed with γ of increasing cardinality and as soon as $P_\gamma \models \gamma(\phi)$ all γ' with $\gamma \subseteq \gamma'$ can be included in ρ.

2.4 Contract Refinement

In order to simplify the presentation, in this paper, we define a contract refinement independently from the component interfaces. In practice, in the tool support we consider, contracts are specified in terms of component input/output ports and the refinement has to take into account the connections among ports in component decomposition.

A contract C over the variables V is a pair $\langle A, G\rangle$ of LTL formulas over V_S representing respectively an *assumption* and a *guarantee*.

We also denote A by $\mathcal{A}(C)$, G by $\mathcal{G}(C)$, and the assertion $\neg A \vee G$ by $nf(C)$.

Let $C = \langle A, G\rangle$ be a contract over V. Let I and E be TS over V. We say that I is a correct implementation of C iff $I \models A \rightarrow G$. We say that E is a correct environment of C iff $E \models A$. We denote by $\mathcal{I}(C)$ and $\mathcal{E}(C)$, respectively, the set of correct implementations and the set of correct environments of C.

Given two contracts C and C' over V, we say that C refines C' (denoted by $C \preceq C'$) iff $\mathcal{I}(C') \subseteq \mathcal{I}(C)$ and $\mathcal{E}(C) \subseteq \mathcal{E}(C')$.

In a system architecture, each contract is associated to a component. If a component is decomposed into subcomponents, the associated contracts are implemented by the composition of the subcomponents' implementations. Similarly, the environment of the contract of a subcomponent is given by the composition of the environment of the composite component and the implementations of the other subcomponents. In order to prove that such decomposition is correct, we generalize the refinement notion to a set of contracts.

Given a contract C and a set of contracts $Sub = \{C_1, \ldots, C_n\}$, we say that Sub is a refinement of C, written $Sub \preceq C$, iff the following conditions hold:

1. the correct implementations of the sub-contracts form a correct implementation of C:

$$\{S_1 \times \ldots \times S_n \mid S_1 \in \mathcal{I}(C_1), \ldots, S_n \in \mathcal{I}(C_n)\} \subseteq \mathcal{I}(C)$$

2. for every $C_i \in Sub$, the correct implementation of the other sub-contracts and a correct environment of C form a correct environment of C_i:

$$\{E \times S_1 \times \ldots \times S_{j\neq i} \times \ldots \times S_n \mid E \in \mathcal{E}(C), \text{ for all } j, 1 \leq j \leq n, j \neq i, S_j \in \mathcal{I}(C_j)\} \subseteq \mathcal{E}(C_i)$$

In [12,13], we proved that the refinement is correct if and only if the following proof obligations are valid temporal formulas:

$$nf(C_1) \wedge \ldots \wedge nf(C_n) \rightarrow nf(C)$$

$$A \wedge \bigwedge_{1 \leq j \leq n, j \neq i} nf(C_j) \rightarrow A_i \text{ (for every } i, 1 \leq i \leq n)$$

3 The Problem of Tightening a Refinement

3.1 Motivation

Contract-Based Design The contract-based design flow is depicted in Figure 1, using the example of a Wheel Braking System (WBS), which takes care of translating the brake signals of the braking pedals into physical brake of the wheel. The brake pedal position is electrically fed to the braking computer, which in turn produces corresponding control signals to the brakes. This computer is named the Braking System Control Unit (BSCU). The BSCU is implemented with two redundant sub-systems, called subBSCU. Therefore, the BSCU takes as input two redundant `Pedal_Pos` brake positions and outputs a pressure on the `Brake_Line`.

The design starts with the view of the system as a whole black box with ports to interact with its environment. Then, it is decomposed into `BSCU` and `Hydraulic` components. The `BSCU` is in turn decomposed into two redundant `subBSCU` and a switch. The decomposition also defines how the ports of the component being decomposed are mapped down into the decomposition. For example, the "left" ports of the `WBS` are mapped onto the "left" ports of the `BSCU`.

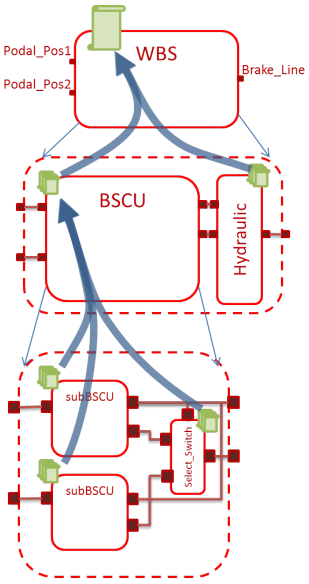

Fig. 1: Contract-based design flow.

Each component in the hierarchy is associated with a set of *contracts*, depicted in green, specifying the acceptable behaviors for the component and its environment. Contracts are refined, following the decomposition of components. For example, the contracts of the WBS are refined by some contracts of the BSCU and the Hydraulic subcomponents. The framework guarantees that, under specific conditions (corresponding to correct contract refinement), if the contracts of the subcomponents hold, then the contract of the parent component also holds.

Need for Tightening The typical design of a system follows a top-down approach starting from the system requirements and iteratively deriving the requirements of subcomponents. The process is however quite expensive, especially if the requirements are formalized into formal properties. It may happen therefore to specify contracts on the subcomponents that are more demanding than necessary or that contain unwanted redundancies. It may happen also that the designer specifies a very strong assumption on the system to make the refinement correct and she/he wants to relax such assumption while keeping the design correct. In general, given a correct contract refinement, we would like to understand if the guarantees of subcomponents or assumption on the composite component can be weakened. We call this problem *top-down tightening* of a contract refinement.

In some cases, the guarantees of subcomponents or the assumption of the system are fixed. For example, the designer used the contract specification of an existing component. After having verified the contract refinement, the designer would like to understand if, using this subcomponents' specification, the system properties can be strengthened. Similarly, a given subcomponent specification can entail stronger assumptions on other subcomponents, which would allow the designer to choose alternative design solutions. We call *bottom-up tightening*

of a contract refinement to this problem of strengthening the guarantees of a composite component and the assumptions of the subcomponents.

3.2 Formal Definition

Tightening We now define formally the problem of tightening a contract refinement as follows. Given a contract C, and a set of contracts $C_1, \ldots, C_n$ such that $\{C_1, \ldots C_n\} \preceq C$, a *tightening* of this contract refinement is given by a set of contracts $C', C'_1, \ldots, C'_n$ such that:

- $\{C'_1, \ldots C'_n\} \preceq C'$
- $C' \preceq C$ and, for every i, $1 \leq i \leq n$, $C_i \preceq C'_i$.

A *top-down tightening* is a tightening as defined above such that $\mathcal{G}(C) = \mathcal{G}(C')$ and, for all i, $1 \leq i \leq n$, $\mathcal{A}(C_i) = \mathcal{A}(C'_i)$. We can easily prove that, equivalently, a top-down tightening is given by a set of contracts $C', C'_1, \ldots, C'_n$ such that:

- $\{C'_1, \ldots C'_n\} \preceq C'$
- $\mathcal{A}(C) \models \mathcal{A}(C')$ and, for every i, $1 \leq i \leq n$, $\mathcal{G}(C_i) \models \mathcal{G}(C'_i)$.

A *bottom-up tightening* is a tightening as defined above such that $\mathcal{A}(C) = \mathcal{A}(C')$ and, for all i, $1 \leq i \leq n$, $\mathcal{G}(C_i) = \mathcal{G}(C'_i)$. We can easily prove that, equivalently, a bottom-up tightening is given by a set of contracts $C', C'_1, \ldots, C'_n$ such that:

- $\{C'_1, \ldots C'_n\} \preceq C'$
- $\mathcal{G}(C') \models \mathcal{G}(C)$ and, for every i, $1 \leq i \leq n$, $\mathcal{A}(C'_i) \models \mathcal{A}(C_i)$.

4 The Algorithm

4.1 Overview

In this section, we present the main algorithm for tightening a contract refinement for the two variants of the problem we defined (*top-down* and *bottom-up*). The procedure first injects a set P of parameters in the contract specification to create a search space of weakened or strengthened assumptions and guarantees. Second, it creates the related proof obligations that are now parametrized by P and we want to find for which configurations of the parameters the contract refinement holds. This is a multiple parameter synthesis problem, because we have to search for the assignment to P such that all proof obligations are valid. Thus, as third step, we convert the problem to a single standard parameter synthesis problem and we call an off-the-shelf algorithm to solve it. In the first step, we make sure that the injection creates a monotonic parameter region by construction, which can be exploited by the synthesis algorithm.

These steps are formalized as follows, while the pseudo-code is shown in Algorithm 1. Suppose we want to obtain a top-down tightening of $Sub \preceq C$.

1. We transform C and Sub into a parametrized version C_P and Sub_P such that for every evaluation γ of P, if $\gamma(Sub_P) \preceq \gamma(C_P)$, then $\langle \gamma(C_P), \gamma(Sub_P) \rangle$ is a top-down tightening of $\langle C, Sub \rangle$.
2. We generate the proof obligations $PO(V, P)$ of $\gamma(Sub_P) \preceq \gamma(C_P)$.
3. We generate a single proof obligation ϕ that is equivalent to $PO(V, P)$ in the sense that $\{\gamma \in \Sigma(P) \text{ s.t. } \models \gamma(\phi) \text{ for every } \phi \in PO(V, P)\} = \{\gamma \in \Sigma(P) \text{ s.t. } \models \gamma(\phi_{PO})\}$.

Algorithm 1 Tightening a Contract Refinement

Input: a contract C, a set of contracts $Sub = \{C_1, \ldots, C_n\}$ such that $Sub \preceq C$, and $T =$ bottom-up or top-down
Output: $Sub' = \{C'_1, \ldots C'_n\} \preceq C'$and $C' \preceq C$ and, for every i, $1 \leq i \leq n$, $C_i \preceq C'_i$.

```
{Calling top-down or bottom-up alg. on Sub and C}
if T = top-down then
    ⟨⟨Sub_P, C_P⟩, P⟩ = Top_down_tightening(Sub, C)
else {T = bottom-up}
    ⟨⟨Sub_P, C_P⟩, P⟩ = Bottom_up_tightening(Sub, C)
end if
{Construction of the Proof Obligations}
POs = ConstructPOs(Sub_P, C_P)
{Encodes all POs into a single PO}
PO = BuildSinglePO(POs)
{Calling Parameter Synthesis Algorithm}
param_region = ComputeParamRegion(PO, P)
{Generate output}
GenerateTightenedContractRef(PO, param_region)
```

4.2 Generation of the Parametric Problem

In this section, we describe how we introduce parameters in the contracts and generate a monotonic parameter synthesis problem. The high-level transformation is described in Algorithms 2 and 3 where assumptions and guarantees are weakened or strengthened depending on whether we are targeting a top-down or a bottom-up tightening of the contract refinement.

If the target is the top-down tightening of $Sub \preceq C$, the Algorithm 2 weakens every guarantee of the subcontracts in Sub and the assumption of the C. If the target is the bottom-up tightening, the Algorithm 3 strengthens the guarantee of C and every assumption of Sub.

The *Weaken* and *Strengthen* functions are described respectively in Algorithms 4 and 5. They take as input a formula and they return a parametric formula and a set of injected parameters. The definition assumes that every new parameter p is a fresh symbol. The number of parameters is linear in the size of the formula.

Parameters are injected so that every parameter evaluation yields a respectively weaker or stronger formula.

Algorithm 2 Top-down tightening ($Top\_down\_tightening(Sub, C)$)

Input: a contract C and a set of contracts $Sub = \{C_1, \ldots, C_n\}$
Output: $\langle\langle Sub', C'\rangle, P\rangle$

$Sub' = \emptyset$
$P = \emptyset$ {Set of parameters}
for all $C_i \in Sub$ **do**
  $\langle \mathcal{G}(C_i'), P'\rangle = Weaken(\mathcal{G}(C_i))$
  $Sub' = Sub' \cup \{\langle \mathcal{A}(C_i), \mathcal{G}(C_i')\rangle\}$
  $P = P \cup P'$
end for
$\langle \mathcal{A}(C'), P'\rangle = Weaken(\mathcal{A}(C))$
$C' = \langle \mathcal{A}(C'), \mathcal{G}(C)\rangle$
$P = P \cup P'$
return $\langle\langle Sub', C'\rangle, P\rangle$

Algorithm 3 Bottom-up tightening ($Bottom\_up\_tightening(Sub, C)$)

Input: a contract C and a set of contracts $Sub = \{C_1, \ldots, C_n\}$
Output: $\langle\langle Sub', C'\rangle, P\rangle$

$Sub' = \emptyset$
$P = \emptyset$ {Set of parameters}
for all $C_i \in Sub$ **do**
  $\langle \mathcal{A}(C_i'), P'\rangle = Strengthen(\mathcal{A}(C_i))$
  $Sub' = Sub' \cup \{\langle \mathcal{A}(C_i'), \mathcal{G}(C_i)\rangle\}$
  $P = P \cup P'$
end for
$\langle \mathcal{G}(C'), P'\rangle = Strengthen(\mathcal{G}(C))$
$C' = \langle \mathcal{A}(C), \mathcal{G}(C')\rangle$
$P = P \cup P'$
return $\langle\langle Sub', C'\rangle, P\rangle$

We remark that we do not aim to obtain the weakest or strongest version of a formula. In our approach, the definition of *Weaken* and *Strengthen* functions is pattern-based where new patterns can be investigated to complement or improve the current ones.

Theorem 1. *For any parameter evaluation* γ, $\phi \to \gamma(Weaken(\phi))$ *and* $\gamma(Strengthen(\phi)) \to \phi$.

Proof. We prove the theorem by induction on the structure of the formula. If $Weaken(\phi) = \langle \phi^W, P\rangle$, we denote with ϕ' the instantiation of ϕ^W with some evaluation γ. Similarly, if $Strengthen(\phi) = \langle \phi^S, P\rangle$, we denote with ϕ'' the instantiation of ϕ^S with some evaluation γ.

The result of *Weaken* and *Strengthen* is outlined in Tables 1 and 2. It is routine to check line by line that $\phi \to \phi'$ and $\phi'' \to \phi$, based on the inductive hypothesis that $\phi_1 \to \phi_1'$, $\phi_2 \to \phi_2'$, $\phi_1'' \to \phi_1$. □

Algorithm 4 *Weaken*(ϕ)

Input: a formula ϕ
Output: $\langle \phi^W, P \rangle$

if $\phi = a > b$ (similar for $\phi = a < b$) **then**
  $\phi^W = p_1 \rightarrow (a > b) \wedge p_2 \rightarrow (a \geq b)$
  return $\langle \phi^W, \{p_1, p_2\} \rangle$
else if $\phi = \phi_1 \wedge \phi_2$ **then**
  $\langle \phi_1^W, P_1 \rangle = Weaken(\phi_1)$, $\langle \phi_2^W, P_2 \rangle = Weaken(\phi_2)$
  $\phi^W = p_1 \rightarrow \phi_1^W \wedge p_2 \rightarrow \phi_2^W$
  return $\langle \phi^W, P_1 \cup P_2 \cup \{p_1, p_2\} \rangle$
else if $\phi = \phi_1 \vee \phi_2$ **then**
  $\langle \phi_1^W, P_1 \rangle = Weaken(\phi_1)$, $\langle \phi_2^W, P_2 \rangle = Weaken(\phi_2)$
  $\phi^W = \phi_1^W \vee \phi_2^W$
  return $\langle \phi^W, P_1 \cup P_2 \rangle$
else if $\phi = \phi_1 \ \mathcal{R} \ \phi_2$ **then**
  $\langle \phi_1^W, P_1 \rangle = Weaken(\phi_1)$, $\langle \phi_2^W, P_2 \rangle = Weaken(\phi_2)$
  $\phi^W = p_1 \rightarrow (\phi_1^W \wedge \phi_2^W) \wedge p_2 \rightarrow (\phi_1^W \ \mathcal{R} \ \phi_2^W)$
  return $\langle \phi^W, P_1 \cup P_2 \cup \{p_1, p_2\} \rangle$
else if $\phi = \phi_1 \ \mathcal{U} \ \phi_2$ **then**
  $\langle \phi_1^W, P_1 \rangle = Weaken(\phi_1)$, $\langle \phi_2^W, P_2 \rangle = Weaken(\phi_2)$
  $\phi^W = \phi_1^W \ \mathcal{U} \ \phi_2^W$
  return $\langle \phi^W, P_1 \cup P_2 \rangle$
else if $\phi = \neg\phi_1$ **then**
  $\langle \phi_1^S, P_1 \rangle = Strengthen(\phi_1)$
  return $\langle \neg\phi_1^S, P_1 \rangle$
else
  return $\langle p \rightarrow \phi, \{p\} \rangle$
end if

It follows immediately that Algorithm 2 and 3 yield a correct top-down/bottom-up tightening, as stated in the following corollary.

Corollary 1. *Let C be a contract and Sub a set of contracts. Let $\langle \langle Sub', C' \rangle, P \rangle$ be the result of $Top\_down\_tightening(Sub, C)$ or $Bottom\_up\_tightening(Sub, C)$. Then, for any evaluation γ of the parameters P, if $\gamma(Sub') \preceq \gamma(C')$ then $\langle \gamma(Sub'), \gamma(C') \rangle$ is a top-down or bottom-up tightening of $\langle Sub, C \rangle$, respectively.*

Moreover, the parameter injection is designed so that the semantics of the parametric formulas is monotonic with respect to the parameter evaluations.

Theorem 2. *If $\gamma \subseteq \gamma'$, $\gamma'(Weaken(\phi)) \rightarrow \gamma(Weaken(\phi))$ and $\gamma(Strengthen(\phi)) \rightarrow \gamma'(Strengthen(\phi))$.*

Proof. Looking again at Table 1 and 2, one can check the monotonicity case by case. In fact, for each type of formula, the lines reporting the result of *Weaken* and *Strengthen* are sorted according to the strength of the parameter evaluation

(third column). More precisely, if γ is below γ', then either they are incomparable or $\gamma \subset \gamma'$. Therefore it is routine to prove that, in the second case, $\gamma'(\mathit{Weaken}(\phi)) \rightarrow \gamma(\mathit{Weaken}(\phi))$ and $\gamma(\mathit{Strengthen}(\phi)) \rightarrow \gamma'(\mathit{Strengthen}(\phi))$ (fourth column). □

Formula ϕ	$\mathit{Weaken}(\phi) = \langle \phi^W, P \rangle$	Evaluation γ	$\gamma(\mathit{Weaken}(\phi))$
$a < b$	$p_1 \rightarrow (a < b) \wedge p_2 \rightarrow (a \leq b)$	$\{p_1, p_2\}$ $\{p_1\}$ $\{p_2\}$ $\emptyset$	$a \leq b$ $a < b$ $a \leq b$ $\top$
$\phi_1 \wedge \phi_2$	$p_1 \rightarrow \phi_1^W \wedge p_2 \rightarrow \phi_2^W$	$\{p_1, p_2\}$ $\{p_1\}$ $\{p_2\}$ $\emptyset$	$\phi_1' \wedge \phi_2'$ ϕ_1' ϕ_2' $\top$
$\phi_1 \vee \phi_2$	$\phi_1^W \vee \phi_2^W$	NA	$\phi_1' \vee \phi_2'$
$\phi_1 \mathcal{R} \phi_2$	$p_1 \rightarrow (\phi_1^W \wedge \phi_2^W) \wedge p_2 \rightarrow (\phi_1^W \mathcal{R} \phi_2^W)$	$\{p_1, p_2\}$ $\{p_2\}$ $\{p_1\}$ $\emptyset$	$\phi_1' \wedge \phi_2'$ $\phi_1' \mathcal{R} \phi_2'$ $\phi_1' \wedge \phi_2'$ $\top$
$\phi_1 \mathcal{U} \phi_2$	$\phi_1^W \mathcal{U} \phi_2^W$	NA	$\phi_1' \mathcal{U} \phi_2'$
$\neg\phi_1$	$\neg\phi_1^S$	NA	$\neg\phi_1'$

Table 1: Simplification table for $\mathit{Weaken}(\phi)$, where ϕ_i' denotes the instantiation of ϕ_i^W with some evaluation γ.

Formula ϕ	$\mathit{Strengthen}(\phi) = \langle \phi^S, P \rangle$	Evaluation γ	$\gamma(\mathit{Strengthen}(\phi))$
$a \leq b$	$\neg p_1 \rightarrow (a < b) \wedge \neg p_2 \rightarrow (a = b) \wedge$ $(p_1 \wedge p_2) \rightarrow (a \leq b)$	$\{p_1, p_2\}$ $\{p_2\}$ $\{p_1\}$ $\emptyset$	$a \leq b$ $a < b$ $a = b$ $\bot$
$\phi_1 \vee \phi_2$	$\neg p_1 \rightarrow \phi_1^S \wedge \neg p_2 \rightarrow \phi_2^S \wedge$ $(p_1 \wedge p_2) \rightarrow (\phi_1^S \vee \phi_2^S)$	$\{p_1, p_2\}$ $\{p_2\}$ $\{p_1\}$ $\emptyset$	$\phi_1'' \vee \phi_2''$ ϕ_1'' ϕ_2'' $\phi_1'' \wedge \phi_2''$
$\phi_1 \wedge \phi_2$	$\phi_1^S \wedge \phi_2^S$	NA	$\phi_1'' \wedge \phi_2''$
$\phi_1 \mathcal{U} \phi_2$	$\neg p \rightarrow \phi_2^S \wedge p \rightarrow \phi_1^S \mathcal{U} \phi_2^S$	$\{p\}$ $\emptyset$	$\phi_1'' \mathcal{U} \phi_2''$ ϕ_2''
$\phi_1 \mathcal{R} \phi_2$	$\phi_1^S \mathcal{R} \phi_2^S$	NA	$\phi_1'' \mathcal{R} \phi_2''$
$\neg\phi_1$	$\neg\phi_1^W$	NA	$\neg\phi_1''$

Table 2: Simplification table for $\mathit{Strengthen}(\phi)$, where ϕ_i'' denotes the instantiation of ϕ_i^S with some evaluation γ.

Note that parameters are introduced per contract, so they are shared by difference occurrences of the assumption/guarantee in the proof obligations. It

is immediate to show that, thanks to the structured way in which formulas are either strengthened or weakened according to the target top-down/bottom-up tightening, the resulting synthesis problem is monotonic, as stated in the following corollary.

Corollary 2. *Let C be a contract and Sub a set of contracts. Let $\langle\langle Sub', C'\rangle, P\rangle$ the result of $Top\_down\_tightening(Sub, C)$ or $Bottom\_up\_tightening(Sub, C)$. Then, for any evaluation γ, γ' of the parameters P such that $\gamma \subseteq \gamma'$, if $\gamma(Sub') \preceq \gamma(C')$ then $\gamma'(Sub') \preceq \gamma'(C')$.*

Algorithm 5 *Strengthen*(ϕ)

Input: a formula ϕ
Output: $\langle \phi^S, P\rangle$

if $\phi = a \leq b$ (similar for $a \geq b$) **then**
  $\phi^S = \neg p_1 \rightarrow (a < b) \wedge \neg p_2 \rightarrow (a = b) \wedge (p_1 \wedge p_2) \rightarrow (a \leq b)$
  return $\langle \phi^S, \{p_1, p_2\}\rangle$
else if $\phi = \phi_1 \vee \phi_2$ **then**
  $\langle \phi_1^S, P_1\rangle = Strengthen(\phi_1)$, $\langle \phi_2^S, P_2\rangle = Strengthen(\phi_2)$
  $\phi^S = \neg p_1 \rightarrow \phi_1^S \wedge \neg p_2 \rightarrow \phi_2^S \wedge (p_1 \wedge p_2) \rightarrow (\phi_1^S \vee \phi_2^S)$
  return $\langle \phi^S, P_1 \cup P_2 \cup \{p_1, p_2\}\rangle$
else if $\phi = \phi_1 \wedge \phi_2$ **then**
  $\langle \phi_1^S, P_1\rangle = Strengthen(\phi_1)$, $\langle \phi_2^S, P_2\rangle = Strengthen(\phi_2)$
  $\phi^S = \phi_1^S \wedge \phi_2^S$
  return $\langle \phi^S, P_1 \cup P_2\rangle$
else if $\phi = \phi_1 \ \mathcal{U}\ \phi_2$ **then**
  $\langle \phi_1^S, P_1\rangle = Strengthen(\phi_1)$, $\langle \phi_2^S, P_2\rangle = Strengthen(\phi_2)$
  $\phi^S = \neg p \rightarrow \phi_2^S \wedge p \rightarrow \phi_1^S \ \mathcal{U}\ \phi_2^S$
  return $\langle \phi^S, P_1 \cup P_2 \cup \{p\}\rangle$
else if $\phi = \phi_1 \ \mathcal{R}\ \phi_2$ **then**
  $\langle \phi_1^S, P_1\rangle = Strengthen(\phi_1)$, $\langle \phi_2^S, P_2\rangle = Strengthen(\phi_2)$
  $\phi^S = \phi_1^S \ \mathcal{R}\ \phi_2^S$
  return $\langle \phi^S, P_1 \cup P_2\rangle$
else if $\phi = \neg\phi_1$ **then**
  $\langle \phi_1^W, P_1\rangle = Weaken(\phi_1)$
  return $\langle \neg\phi_1^W, P_1\rangle$
else
  return $\langle \neg p \rightarrow \phi, \{p\}\rangle$
end if

4.3 Multiple Validity Parameter Synthesis Problem

The approach to solve the tightening problem proposed in Section 4.1 introduces the problem of finding the parameter evaluations γ such that each formula $\phi(P, V) \in PO$ instantiated with γ is valid. Each validity problem can be reduced to a model checking problem but the parameter evaluation is shared by the different verification problems. This is different from the standard parameter

synthesis problem where only one verification problem is considered. We called this problem a multiple validity parameter synthesis problem (not to be confused with multiple objective parameter synthesis problem).

We propose to reduce the multiple validity to one validity problem by renaming the variables in V and taking the conjunction of the proof obligations. Namely, if $PO = \{\phi_1, \ldots, \phi_n\}$ we create the formula $\phi_{PO}(P, V_1, \ldots, V_n) = \bigwedge_{1 \leq j \leq n} \phi_j[V_j/V]$, where V_j contains one copy v_j for each variable $v \in V$ and $\phi_j[V_j/V]$ is the formulas obtained by substituting every variable $v \in V$ with v_j (while the parameters P remain unchanged).

Theorem 3. *For all parameter evaluation γ, $\gamma(\phi_{PO})$ is valid iff, for all formulas $\phi \in PO$, $\gamma(\phi)$ is valid.*

Proof. $\Rightarrow$) Suppose for some $\phi_j \in PO$, $\gamma(\phi_j)$ is not valid. Let σ be a trace over V satisfying $\neg\gamma(\phi_j)$. Let us define the trace σ_j such that, for every $i \geq 0$, for all $v \in V$, $\sigma_j[i](v_j) = \sigma[j](v)$. Let us extend σ_j to a trace σ'_j over $V_1 \cup \ldots V_n$ assigning variables not in V_j in an arbitrary way. Then σ'_j satisfies $\neg\gamma(\phi_{PO})$.

$\Leftarrow$) Suppose ϕ_{PO} is not valid. Let σ be a trace over $V_1 \cup \ldots \cup V_n$ satisfying $\neg\gamma(\phi_{PO})$. Then, there exists j, $1 \leq j \leq n$, such that $\sigma \models \neg\gamma(\phi_j[V_j/V])$. Let us define the trace σ_j such that, for every $i \geq 0$, for all $v \in V$, $\sigma_j[i](v) = \sigma[j](v_j)$. Then σ'_j satisfies $\neg\gamma(\phi_j)$. □

5 Experimental Evaluation

5.1 Details of the Implementation

We have implemented the algorithms described in the previous section on top of OCRA [8], a tool for architectural design based on contract-based design. In more details, we implemented a new command in OCRA that takes as input an OCRA specification, a contract's name, a component's name, and a desired variant of tightening (top-down or bottom-up) and produces as output an OCRA specification containing the tightened version of the given contract and its subcontracts. Regarding the parameter synthesis algorithm, we have used as backend an implementation reported in [6]. Since the synthesis is quite expensive for large number of parameters, we arbitrarily limit the injection to 350 parameters. This allows to get a tightening also in cases in which the definitions would produce many more parameters making the synthesis blow up.

We also implemented self checks to validate the results: first, we automatically check that each tightened contract refinement is correct; second, we automatically check for each tightened specification that the original formula entails the weaken formula (top-down tightening) and the strengthened formula entails the original formula (bottom-up tightening), see Theorem 1.

5.2 Description of benchmarks

We have taken several benchmarks from several case studies developed manually using the OCRA language. Some examples are: several versions of the Wheel

Brake System described in Section 3, a Lift System, a system with Redundant Sensors, and Airbag system [1]. Particularly, an interesting case study is taken from [5], where the authors presented a complete formal analysis of the AIR6110, a document describing the informal design of a Wheel Brake System, covering all the phases of the process, and modeled the case study by means of a combination of formal methods including contract-based design using OCRA, model checking and safety analysis.

5.3 Experimental Results

We carried out an experimental evaluation for 875 contract refinements taking into account the simplification obtained on each tightened contract refinement with respect to the length of the formulas on the original contracts involved [1]. The results of applying top-down (red crosses) and bottom-up (grey circles) tightening are shown in Figure 2. From the results, we can clearly see a significant simplification for top-down tightening. As for bottom-up tightening, we did not get important simplification, but we observed that the main reason is that the size of formulas of the contracts involved are much smaller compared to the ones involved on the top-down tightening.

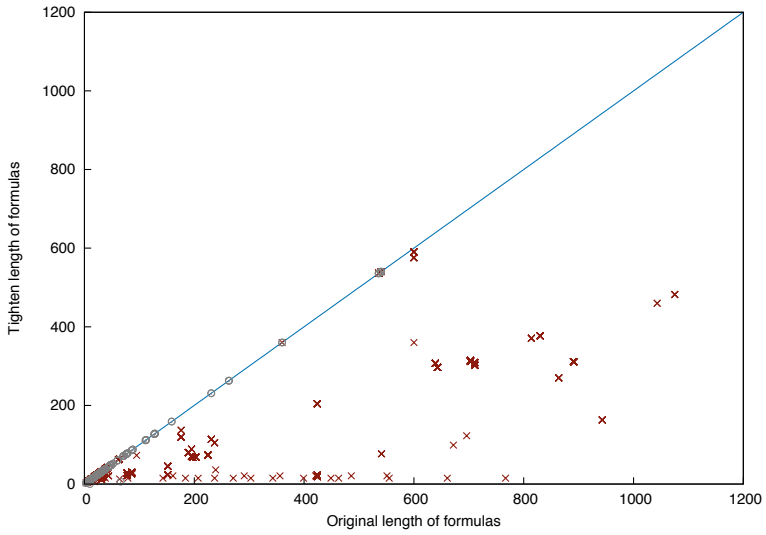

Fig. 2: Analysis of length of formulas for top-down and bottom-up.

[1] We consider the standard definition of the length of a formula (number of symbols), apart from the length of $\top$ which is set to 0.

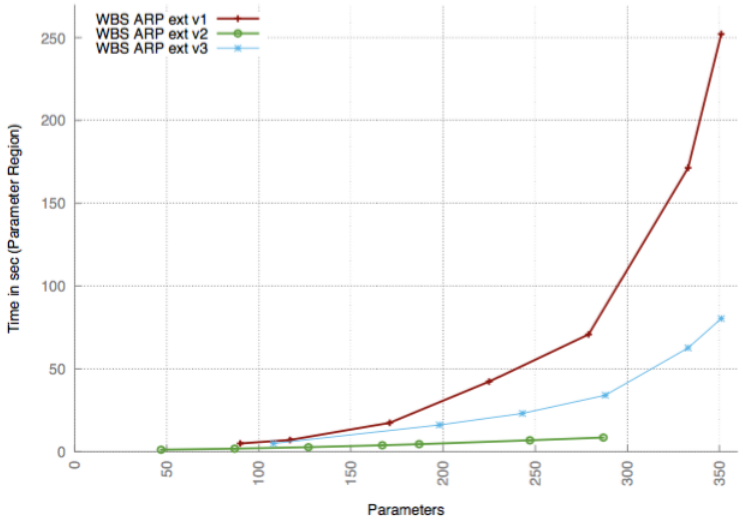

Fig. 3: Parameter scalability.

In Figure 3 it is shown how our approach scales with respect to the number of parameters used for tightening a contract refinement and the time for computing the parameter region for three extended versions of the WBS example. All benchmarks have been performed with a time limit of 5 minutes for checking the contract refinement before and after tightening, the computation of the parameter region, and the check of the entailments properties. For the 875 contract refinements, 68 could not be completed within the timeout. We have run our experiments on a Linux machine with 8 CPU of 3.40 Ghz Intel Xeon, with a memory of 15Gb.

6 Conclusions and Future Work

Motivated by validation problems of contract-based design, we defined the problem of tightening a contract refinement. We provided a solution based on the synthesis of parameters of temporal satisfiability problems. We evaluated the approach on a number of benchmarks and showed that the solution is effective and scalable. For future work, we will extend the approach to consider also the tightening of metric operators and the preservation of realizability.

References

1. T. Arts, M. Dorigatti, and S. Tonetta. Making Implicit Safety Requirements Explicit - An AUTOSAR Safety Case. In *SAFECOMP*, pages 81–92, 2014.
2. S.S. Bauer, A. David, R. Hennicker, K.G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Moving from Specifications to Contracts in Component-Based Design. In *FASE*, pages 43–58, 2012.

3. A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple Viewpoint Contract-Based Specification and Design. In *FMCO*, pages 200–225, 2007.
4. A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K.G. Larsen. Contracts for System Design. Technical Report RR-8147, INRIA, November 2012.
5. M. Bozzano, A. Cimatti, A.F. Pires, D. Jones, G. Kimberly, T. Petri, R. Robinson, and S. Tonetta. Formal Design and Safety Analysis of AIR6110 Wheel Brake System. In *CAV*, pages 518–535, 2015.
6. Marco Bozzano, Alessandro Cimatti, Alberto Griggio, and Cristian Mattarei. Efficient Anytime Techniques for Model-Based Safety Analysis. In *CAV*, pages 603–621, 2015.
7. R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuXmv Symbolic Model Checker. In *CAV*, pages 334–342, 2014.
8. A. Cimatti, M. Dorigatti, and S. Tonetta. OCRA: A Tool for Checking the Refinement of Temporal Contracts. In *ASE*, pages 702–705, 2013.
9. A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. Boolean Abstraction for Temporal Logic Satisfiability. In *CAV*, pages 532–546, 2007.
10. A. Cimatti, M. Roveri, and S. Tonetta. Requirements Validation for Hybrid Systems. In *CAV*, pages 188–203, 2009.
11. A. Cimatti, M. Roveri, and S. Tonetta. HRELTL: A temporal logic for hybrid systems. *Inf. Comput.*, 245:54–71, 2015.
12. A. Cimatti and S. Tonetta. A Property-Based Proof System for Contract-Based Design. In *SEAA*, 2012.
13. A. Cimatti and S. Tonetta. Contracts-refinement proof system for component-based embedded systems. *Sci. Comput. Program.*, 97:333–348, 2015.
14. D.D. Cofer, A. Gacek, S.P. Miller, M.W. Whalen, B. LaValley, and L. Sha. Compositional Verification of Architectural Models. In *NFM*, pages 126–140, 2012.
15. W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *DATE*, pages 1023–1028, 2011.
16. S. Graf, R. Passerone, and S. Quinton. Contract-Based Reasoning for Component Systems with Complex Interactions . In *TIMOBD'11*, 2011.
17. A. Iannopollo, P. Nuzzo, S. Tripakis, and A.L. Sangiovanni-Vincentelli. Library-based scalable refinement checking for contract-based design. In *DATE*, pages 1–6, 2014.
18. O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2):224–233, 2003.
19. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.
20. B. Meyer. Applying "Design by Contract". *Computer*, 25(10):40–51, 1992.
21. A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.
22. S. Quinton and S. Graf. Contract-Based Verification of Hierarchical Systems of Components. In *SEFM*, pages 377–381, 2008.
23. V. Schuppan. Towards a notion of unsatisfiable and unrealizable cores for LTL. *Sci. Comput. Program.*, 77(7-8):908–939, 2012.