Incremental Refinement of Goal Models with Contracts

Piergiuseppe Mallozzi, Pierluigi Nuzzo, Patrizio Pelliccione

Incremental Refinement of Goal Models with Contracts

Piergiuseppe Mallozzi[1], Pierluigi Nuzzo[2], and Patrizio Pelliccione[1,3]

Chalmers University of Technology | University of Gothenburg[1]
University of Southern California[2], Gran Sasso Science Institute (GSSI)[3]
mallozzi@chalmers.se, nuzzo@usc.edu, patrizio.pelliccione@gssi.it

Abstract. Goal models and contracts offer complementary approaches to requirement analysis. Goal modeling has been effectively used to capture designer's intents and their hierarchical structure. Contracts emphasize modularity and formal representations of the interactions between system components. In this paper, we present CoGoMo (Contract-based Goal Modeling), a framework for systematic requirement analysis, which leverages a new formal model, termed *contract-based goal tree*, to represent goal models in terms of hierarchies of contracts. Based on this model, we propose algorithms that use contract operations and relations to check goal consistency and completeness, and support incremental and hierarchical refinement of goals from a library of goals. Model and algorithms are implemented in a tool which enables incremental formalization and refinement of goals from a web interface. We show the effectiveness of our approach on an illustrative example motivated by vehicle platooning.

1 Introduction

Missing or erroneously formulated requirements can have a negative impact on the quality of a design. Designers are often faced with the challenge of ensuring the correctness of an implementation despite the growing complexity of the requirement corpora [29]. Existing requirement-management tools are mostly based on natural-language constructs that leave space for ambiguities, redundancies, and conflicts [13,23]. Furthermore, the requirement elicitation process is itself challenging, as it revolves around human-related considerations that are intrinsically difficult to capture.

Goal modeling (e.g., as in KAOS [31,9]) has been used over the years as an intuitive and effective means to capture the designer's intents and their hierarchical structure. The refinement process, however, mostly follows informal procedures, e.g., by posing *how* questions about existing high-level goals (top-down process) or *why* questions about low-level goals for the system under consideration (bottom-up process) [32,29]. The main modeling challenges are framed in terms of ensuring *completeness* and *consistency* of a specification. A set of hierarchically organised goals is *incomplete* when the high-level goal remains unsatisfied even if the low-level goals, which are expected to capture its decomposition, are satisfied, meaning that the designer could not anticipate all the possible operating scenarios for the design. There is, instead, a *conflict* when the satisfaction of a goal prevents the satisfaction of another goal [30]. The process of completely refining a goal into sub-goals is not straightforward [11]. On the other hand, independently-developed goals can include overlapping and conflicting behaviors [12]. Systematic methods to detect conflicts and incomplete requirements remain an active research area [21,4,14].

This paper presents a framework, CoGoMo (Contract-based Goal Modeling), which addresses these challenges by representing goals via contract models. Contract-based modeling has shown to enable formal requirement analysis in a modular way, rooted in sound representations of the system semantics and decomposition architecture [2,27,8,25,22,24,23]. A *contract* specifies the behavior of a component by distinguishing the responsibilities of the component (*guarantees*) from those of its environment (*assumptions*). Contract operations and relations provide formal support for notions such as stepwise *refinement* of high-level contracts into lower-level contracts, *compositional reasoning* about contract aggregations, and *reuse* of pre-designed components satisfying a contract. CoGoMo addresses correctness and completeness of goal models by formulating and solving contract consistency and refinement checking problems. Specifically, the contributions of the paper can be summarised as follows:

– A novel formal model, namely, *contract-based goal tree (CGT)*, which represents a goal model as a hierarchy of assume-guarantee (A/G) contracts.
– Algorithms that exploit the CGT as well as contract-based operations to detect conflicts and perform complete hierarchical refinements of goals. Specifically, we introduce mechanisms that help resolve inconsistencies between goals during refinement and a *goal extension* algorithm to automatically refine the CGT using new goals from a library.
– A tool, which implements the proposed model and algorithms to incrementally formalize and refine goals via an easy-to-use web-interface.

We illustrate the applicability of CoGoMo on a case study motivated by vehicle platooning.

2 Background

Goals. A goal is a prescriptive statement of intent that the system should satisfy, formulated in a declarative way. Goals can be decomposed, progressing from high-level objectives to fine-grained system prescriptions [31]. For example, an AND-refinement link relates a goal to a set of sub-goals. The parent can be satisfied if all the sub-goals in the refinement are also satisfied. Establishing *correctness* of the refinement amounts to ensuring that the sub-goals are *consistent*, i.e., there are no *conflicts* among them, and *complete*, i.e., there are no behaviours left unspecified that could result in a violation of the high-level goal even if the lower-level goals are satisfied. We only refer to *internal completeness*, i.e., we are not concerned with investigating whether all the information required to define a design problem is in the specification [34]. Formally, we say that the refinement of goal $\mathcal{G}$ into sub-goals $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$ is correct if and only if

$$\underbrace{\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\} \not\models \textbf{false}}_{\text{consistency}} \quad \wedge \quad \underbrace{\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\} \models \mathcal{G},}_{\text{completeness}}$$

where we denote by $\models$ the entailment operator between goals and say that $\{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$ entails $\mathcal{G}$ to mean that, if all $\mathcal{G}_1, \ldots, \mathcal{G}_n$ are satisfied then $\mathcal{G}$ is satisfied. Similarly, we write $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\} \not\models \textbf{false}$ to indicate that the logical conjunction of $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\}$ does not lead to false.

Contracts. A *contract* $\mathcal{C}$ is a triple (V, A, G) where V is a set of *variables*, and A and G are sets of behaviors over V. Behaviors are generic. For example, they can be traces generated by the execution of a finite state machine, i.e., infinite sequences of assignments to the variables in V, consisting of the input, output, and state variables of the state machine. For simplicity, whenever possible, we drop V from the definition and refer to contracts as pairs of assumptions and guarantees, i.e., $\mathcal{C} = (A, G)$. A expresses the behaviors that a system expects from its environment, while G expresses the behaviors that a system implementation promises under the environment assumptions. An environment E satisfies a contract $\mathcal{C}$ whenever E and $\mathcal{C}$ are defined over the same set of variables and all the behaviors of E are included in the assumptions of $\mathcal{C}$, i.e., when $|E| \subseteq A$, where $|E|$ is the set of behaviors of E. An implementation M satisfies a contract $\mathcal{C}$ whenever M and $\mathcal{C}$ are defined over the same set of variables and all the behaviors of M are included in the guarantees of $\mathcal{C}$ when considered in the context of the assumptions A, i.e., when $|M| \cap A \subseteq G$. A contract $\mathcal{C} = (A, G)$ can be placed in saturated form by re-defining its guarantees as $G_{sat} = G \cup \overline{A}$, where $\overline{A}$ denotes the complement of A. A contract and its saturated forms are semantically equivalent, i.e., they have the same set of environments and implementations. Therefore, in the rest of the paper, we assume that all the contracts are expressed in saturated form [2].

We say that a contract is *well-formed* if and only if it is *compatible*, i.e., $A \neq \emptyset$ and *consistent*, i.e., $G \neq \emptyset$, that is, if and only if there exists at least an environment and an implementation that satisfy the contract. Contract *refinement* formalizes a notion of substitutability among contracts. Let $\mathcal{C} = (A, G)$ and $\mathcal{C}' = (A', G')$ be two contracts. $\mathcal{C}$ refines $\mathcal{C}'$ if and only if all the assumptions of $\mathcal{C}'$ are contained in the assumptions of $\mathcal{C}$ and all the guarantees of $\mathcal{C}$ are included in the guarantees of $\mathcal{C}'$, that is, $\mathcal{C} \preceq \mathcal{C}'$ if and only if $A \supseteq A'$ and $G \subseteq G'$. Refinement entails relaxing the assumptions and strengthening the guarantees. If $\mathcal{C} \preceq \mathcal{C}'$, we also say that $\mathcal{C}'$ is an *abstraction* of $\mathcal{C}$ and can be replaced by $\mathcal{C}$.

Contracts can be combined through the operations of *composition* and *conjunction*. Let $\mathcal{C}_1 = (A_1, G_1)$ and $\mathcal{C}_2 = (A_2, G_2)$ be two contracts. The composition $\mathcal{C}_{\parallel} = (A, G) = \mathcal{C}_1 \parallel \mathcal{C}_2$ can be computed using the following expressions: $A = (A_1 \cap A_2) \cup \overline{(G_1 \cap G_2)}$ and $G = G_1 \cap G_2$. The conjunction $\mathcal{C}_{\wedge} = \mathcal{C}_1 \wedge \mathcal{C}_2$ can instead be computed by taking the intersection of the guarantees and the union of the assumptions, that is, $\mathcal{C}_{\wedge} = (A_1 \cup A_2, G_1 \cap G_2)$. Intuitively, an implementation satisfying $\mathcal{C}_{\parallel}$ or $\mathcal{C}_{\wedge}$ must satisfy the guarantees of both $\mathcal{C}_1$ and $\mathcal{C}_2$, hence the operation of intersection. The situation is different for the environments. Composition requires that an environment satisfy the assumptions of both contracts, motivating the conjunction of A_1 and A_2. On the other hand, contract conjunction requires that an implementation operate under all the environments of $\mathcal{C}_1$ and $\mathcal{C}_2$, motivating the disjunction of A_1 and A_2.

In the following, we denote by $\mathcal{C}_i = (\psi_i, \phi_i)$ the contract formalizing a goal $\mathcal{G}_i$, where ψ_i and ϕ_i are logic formulas used to represent the assumptions and the guarantees, respectively. Finally, we perform operations among goals by translating them into operations on the corresponding contracts.

3 Running Example: Vehicle Platooning

We consider a case study inspired by vehicle platooning as an illustrative example throughout the paper. We define goals for a vehicle participating in a platoon in the *following mode*, which adjusts speed and steering angle based on what is communicated by the leading vehicle. We assume that all the vehicles in the platoon communicate via Vehicle Ad hoc Networks (VANETs), established with the IEEE 802.11p standard, a specially designed protocol for intelligent transportation systems (ITS) [33], offering at most 27 Mbps of data transmission rate. The propagation delay is the difference between the time-stamps for message reception, t_{Rx}, and message transmission, t_{Tx}, i.e., $d = t_{Rx} - t_{Tx}$. The necessary time interval $d_{(i,j)}$ for a successful end-to-end transmission of a message of L bits between a pair of vehicles (i, j) is:

$$d_{(i,j)} = \frac{L}{f_{(i,j)}}, \tag{1}$$

where $f_{(i,j)}$ is the *transmission rate* between the i-th and the j-th vehicle. We consider a platoon consisting of N vehicles, where the first one is the leader and the remaining $N-1$ are the followers. To reach all followers, a message generated by the leader propagates for at most $N-1$ hops. We assume the same transmission rate $f = 3$ Mbps between adjacent vehicles and a fixed message length size $L = 400$ bytes.

4 The CoGoMo Approach

CoGoMo revolves around a new formal model, termed *contract-based goal tree* (CGT), and a set of operations on it. A CGT is a tree $T = (\Upsilon, \Sigma)$, where each node $\upsilon \in \Upsilon = \Gamma \cup \Delta$ is either a *goal node* $\gamma \in \Gamma$ or an *operator node* $\delta \in \Delta$, with $\Gamma \cap \Delta = \emptyset$. Each goal node contains the formalization of a goal in terms of a contract. Each operator node assumes a value in the set $\{\|, \wedge, \curlywedge\}$ of available operators, namely, composition, conjunction, and refinement, respectively. Each edge $\sigma \in \Sigma$ connects a goal node in Γ to an operator node in Δ or *vice versa*. Figure 1 shows a portion of a CGT for the vehicle platooning example, where each goal node includes a textual description of a goal. A *library* $\mathcal{L}$ of goals, at the bottom of the figure, is used to automatically extend the CGT. A library of goals is a set of pre-defined goals, e.g., specification patterns, that can be

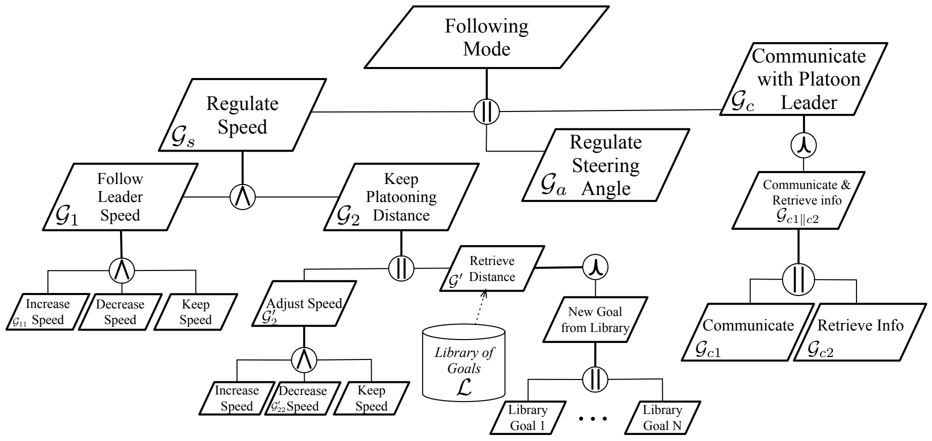

Following Mode
Regulate Speed $\mathcal{G}_s$
Communicate with Platoon Leader $\mathcal{G}_c$
Regulate Steering Angle $\mathcal{G}_a$
Follow Leader Speed $\mathcal{G}_1$
Keep Platooning Distance $\mathcal{G}_2$
Communicate & Retrieve info $\mathcal{G}_{c1||c2}$
Increase Speed $\mathcal{G}_{11}$
Decrease Speed
Keep Speed
Adjust Speed $\mathcal{G}'_2$
Retrieve Distance $\mathcal{G}'$
New Goal from Library
Communicate $\mathcal{G}_{c1}$
Retrieve Info $\mathcal{G}_{c2}$
Library of Goals $\mathcal{L}$
Increase Speed
Decrease Speed $\mathcal{G}'_{22}$
Keep Speed
Library Goal 1
...
Library Goal N

Fig. 1: Portion of the CGT for the vehicle platooning example.

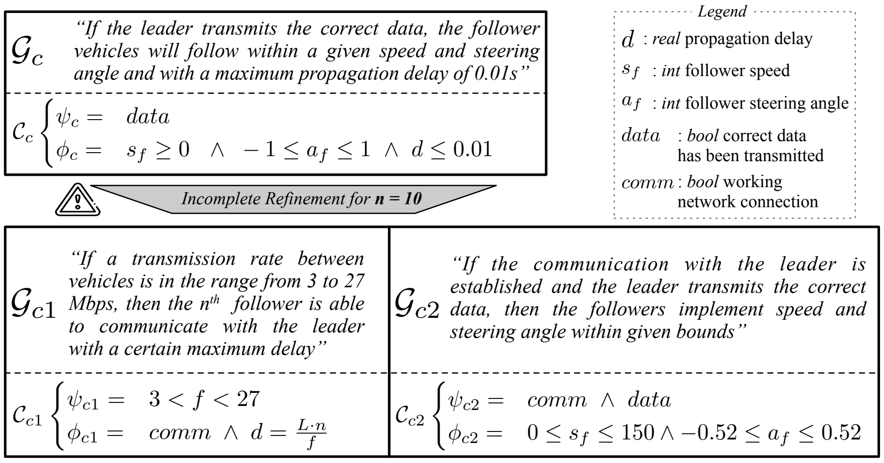
labelled by a *cost*, capturing the overhead incurred when employing a certain goal to extend the CGT, as further illustrated below.

A CGT can be built interactively using a web interface and with the support of a proof-of-concept tool, which is released open-source [18]. A designer can insert (new) goals by typing or uploading a structured text file, while the contracts are specified by formulas expressed in the language of an SMT solver [10]. The specification formalization process then iterates between two activities: (i) goal identification and formalization with A/G contracts, and (ii) goal manipulation, incrementally linking goals via composition, conjunction, and refinement. The outcome is a formal specification in terms of a CGT.

4.1 Goal Formalization

CoGoMo enables requirement formalization by representing goals in terms of contracts. It then solves contract verification problems to detect conflicts and incompleteness among goals. Specifically, completeness and consistency checking problems translate into checking the satisfiability and validity of logic formulas via an SMT solver [10]. In this paper, we express contract assumptions and guarantees as formulas in propositional logic, where atomic propositions include Boolean variables or arithmetic predicates on real variables.

Detecting Conflicts. CoGoMo detects conflicts by checking the compatibility, consistency, and feasibility of each contract formalizing a goal in the CGT. *Feasibility* checking aims to verify that there exists at least an implementation which does not violate the assumptions, that is, for contract $\mathcal{C} = (A, G)$, $A \cap G \neq \emptyset$ holds. CoGoMo verifies compatibility, consistency, and feasibility of a contract (ψ, ϕ) by checking whether ψ, ϕ, and $\psi \wedge \phi$ are satisfiable, respectively. In case of conflict, the SMT solver provides an explanation of infeasibility


Legend
d : *real* propagation delay
s_f : *int* follower speed
a_f : *int* follower steering angle
$data$: *bool* correct data has been transmitted
$comm$: *bool* working network connection
Incomplete Refinement for **n = 10**

$\mathcal{G}_c$ *"If the leader transmits the correct data, the follower vehicles will follow within a given speed and steering angle and with a maximum propagation delay of 0.01s"*

$$\mathcal{C}_c \begin{cases} \psi_c = & data \\ \phi_c = & s_f \geq 0 \ \wedge \ -1 \leq a_f \leq 1 \ \wedge \ d \leq 0.01 \end{cases}$$

$\mathcal{G}_{c1}$ *"If a transmission rate between vehicles is in the range from 3 to 27 Mbps, then the nth follower is able to communicate with the leader with a certain maximum delay"*

$$\mathcal{C}_{c1} \begin{cases} \psi_{c1} = & 3 < f < 27 \\ \phi_{c1} = & comm \ \wedge \ d = \frac{L \cdot n}{f} \end{cases}$$

$\mathcal{G}_{c2}$ *"If the communication with the leader is established and the leader transmits the correct data, then the followers implement speed and steering angle within given bounds"*

$$\mathcal{C}_{c2} \begin{cases} \psi_{c2} = & comm \ \wedge \ data \\ \phi_{c2} = & 0 \leq s_f \leq 150 \wedge -0.52 \leq a_f \leq 0.52 \end{cases}$$

Fig. 2: Examples of three goals formalized by contracts. After linking the goals via composition and refinement, CoGoMo detects that the refinement is incomplete.

in terms of an *unsat core*, i.e., a subset of clauses that are mutually unsatisfiable [10]. CoGoMo then links the conflicting clauses to the goals that generated them and presents these goals to the designer.

Checking Completeness. CoGoMo checks completeness by verifying that all the refinement links of the CGT are correct. Given two contracts, $\mathcal{C} = (\psi, \phi)$ and $\mathcal{C}' = (\psi', \phi')$, $\mathcal{C} \preceq \mathcal{C}'$ holds if and only if $\psi' \rightarrow \psi$ and $\phi \rightarrow \phi'$ are *valid* formulas, i.e., they are tautologies for the language, where $\rightarrow$ denotes the implication. Validity checking can be translated into checking the satisfiability of $\overline{\psi' \rightarrow \psi}$ or $\overline{\phi \rightarrow \phi'}$. If no solution is found, then the refinement is correct; otherwise, the returned solution serves as a certificate of incompleteness, and is exhibited to the designer.

4.2 Goal Manipulation via Composition and Refinement

CoGoMo uses composition to capture with a single goal the composite behaviors resulting from the composition of modules (implementations) that are separately specified by different goals. For example, goal $\mathcal{G}_c$ in Figure 1 can be refined by the composition of $\mathcal{G}_{c1}$ with $\mathcal{G}_{c2}$. Figure 2 proposes an initial formalization of $\mathcal{G}_c$ as a contract $\mathcal{C}_c$ and its further decomposition into two goals, $\mathcal{G}_{c1}$ and $\mathcal{G}_{c2}$, establishing requirements on the network connection and the follower's speed and angle ranges, respectively. $\mathcal{G}_{c1}$ specifies the propagation delay d according to the transmission rate f (in Mbps), the message length L, and the position n of the follower participating in the platoon. Assuming a working network connection, $\mathcal{G}_{c2}$ guarantees that the speed of the follower is at most 150 km/h and its steering angle at most 30° (0.52 rad). We would like to show that $\mathcal{G}_{c1} \parallel \mathcal{G}_{c2} = \mathcal{G}_{c1\|c2}$ can be connected to the top goal $\mathcal{G}_c$ via refinement, i.e., $\mathcal{G}_{c1\|c2} \preceq \mathcal{G}_c$. To do so, CoGoMo first checks for conflicts between $\mathcal{G}_{c1}$ and $\mathcal{G}_{c2}$ by verifying that the contract associated with $\mathcal{G}_{c1\|c2}$ is compatible, consistent, and feasible, which is the case in our example. However, CoGoMo detects that $\mathcal{G}_{c1\|c2}$ is not a refinement of $\mathcal{G}_c$ and provides a certificate showing that, for the 10^{th} follower in the platoon, the propagation delay is slightly larger than $d = 0.01$, which violates the guarantees of $\mathcal{C}_c$. It is, however, still possible to circumvent the incompleteness of the refinement by strengthening the assumptions of $\mathcal{G}_{c1\|c2}$ to limit the size of the platoon to less than 10 vehicles, and by "propagating" this restriction to $\mathcal{G}_c$ via a mechanism of *assumption propagation*, as detailed below.

Assumptions Propagation. When eliciting the top-level goals of a specification in a hierarchical way, we may discover additional assumptions, associated with lower-level goals in the hierarchy, which were not known *a priori.* This inconsistency between assumptions at different levels of the hierarchy may be a reason for incompleteness in the refinement. Let $\mathcal{G}_a$ and $\mathcal{G}_r$ be two goals, and $\mathcal{C}_a = (\psi_a, \phi_a)$ and $\mathcal{C}_r = (\psi_r, \phi_r)$ their respective contracts. Assumption propagation ensures that $\psi_a \rightarrow \psi_r$ is valid, by propagating the assumption formula from the lower-level contract to the upper-level contract and conjoining it with the assumptions of the higher-level contract so that behaviors that are not in ψ_r are removed from ψ_a and ψ_a is redefined as $(\psi_a \wedge \psi_r)$. After assumption propagation, the top-level guarantees will also be updated by bringing $\mathcal{C}_a$ again in saturated form, i.e., by setting the guarantees to $\phi_a \vee \overline{(\psi_a \wedge \psi_r)}$. In our example, the new assumptions for $\mathcal{C}_c$ after propagation become

$$\psi_c = \quad 3 < f < 27 \quad \wedge \quad data \quad \wedge \quad n < 10,$$

which makes refinement complete and allows creating a refinement node in the CGT as in Figure 1.

4.3 Goal Manipulation via Conjunction

CoGoMo uses conjunction to generate a goal that refines multiple goals or *scenarios*, which are active under different assumptions, and may not be simultaneously satisfiable. In our example, the goal $\mathcal{G}_s$, 'Regulate Speed,' in Figure 1 can be decomposed into two sub-goals, $\mathcal{G}_1$ and $\mathcal{G}_2$, specifying how the speed of a vehicle s should be regulated according to the leader's speed s_l or according to the distance d_{front} to the front vehicle, respectively. Because $\mathcal{G}_1$ and $\mathcal{G}_2$ should both be satisfied, albeit in different situations, we can define a single goal $\mathcal{G}_s = \mathcal{G}_1 \wedge \mathcal{G}_2$ for the system. The same procedure applies to the decomposition of $\mathcal{G}_1$ and $\mathcal{G}_2$. For example, the goal 'Adjust Speed' in Figure 1 can be achieved by satisfying 'Increase Speed' or 'Decrease Speed' or 'Keep Speed,' which are not simultaneously satisfiable but are active under different assumptions.

If the assumptions of $\mathcal{G}_1$ and $\mathcal{G}_2$ are not mutually exclusive, the conjunction may require that potentially conflicting guarantees be satisfied simultaneously.

For example, one of the contracts contributing to $\mathcal{G}_1$ may prescribe an *increase* of the follower's speed when there is an increase in the leader's speed, i.e., $\mathcal{C}_{11} = (s_t < s_l,\ \ s_{t+1} > s_t)$ where s_l, s_t and s_{t+1} represent the current speed of the leader and the speed of follower vehicle at times t and $t+1$, respectively. On the other hand, one of the contracts contributing to $\mathcal{G}_2$ may prescribe a *decrease* in the follower's speed when the distance to the vehicle in front is detected and it is less than a certain threshold, i.e., $\mathcal{C}'_{22} = (dist \wedge d_{\text{front}} < D_p,\ \ s_{t+1} < s_t)$, where $dist$ evaluates to true if and only if the distance from the vehicle in front was detected correctly and $d_{\text{front}} < D_p$ indicates that the distance should be less than a constant (i.e., "the platooning distance"). The assumptions of $\mathcal{C}_{11}$ and $\mathcal{C}'_{22}$ can be satisfied simultaneously, possibly causing conflicts in the guarantees of the joint contract $(\psi_{11} \rightarrow (s_{t+1} > s_t))\ \wedge\ (\psi'_{22} \rightarrow (s_{t+1} < s_t))$. CoGoMo prevents such conflicts via a *goal priority* mechanism.

Goal Priority. A *goal priority* mechanism $\mathcal{P}(\mathcal{G}_1, \mathcal{G}_2)$ avoids such potential conflicts by making the assumptions mutually exclusive so that only one goal is effective under any environment. For instance, a priority mechanism may set $\mathcal{C}_2 = (\psi_2 \wedge \overline{\psi_1}, \phi_2)$, assuming that $\psi_2 \wedge \overline{\psi_1}$ is satisfiable, so that $\mathcal{C}_1$ is granted higher priority and dominates whenever both ψ_2 and ψ_1 hold. Because the assumptions of $\mathcal{C}_2$ become stronger, an assumption propagation step may also be needed to keep the refinement relations correct across the CGT. In our example, prioritizing $\mathcal{G}_1$ versus $\mathcal{G}_2$ solves the potential conflict. $\mathcal{G}_s$ can then be satisfied if the vehicle adjusts its speed according to the information provided by the distance sensor. When d_{front} is not available, as denoted by $dist$, the vehicle regulates its speed according to the speed of the leader of the platoon.

When composing a goal that was previously obtained by conjunction, e.g., when composing $\mathcal{G}_2$ with $\mathcal{G}_c$ in Figure 1, it may be useful to separately reason about the different scenarios involved in the composition. To do so, we use the fact that, given three contracts $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_c$, $[(\mathcal{C}_1 \wedge \mathcal{C}_2) \parallel \mathcal{C}_c] = [(\mathcal{C}_1 \parallel \mathcal{C}_c) \wedge (\mathcal{C}_2 \parallel \mathcal{C}_c)]$ holds, and we can separately identify the scenarios associated with $(\mathcal{C}_1 \parallel \mathcal{C}_c)$ and $(\mathcal{C}_2 \parallel \mathcal{C}_c)$ in the composition. In words, while composition is not distributive over conjunction in general [2], the distributive property holds in the special case of three contracts as above. A proof of this result is in the appendix.

5 CGT Extension

Given a leaf node $\mathcal{G}'$ in a CGT and a library of goals $\mathcal{L}$, the extension problem consists in finding a set of goals $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$ in $\mathcal{L}$, that, once composed, refine $\mathcal{G}'$. The CGT is then extended by linking $\mathcal{G}'$ to a node $\mathcal{G}_s$ via refinement, and $\mathcal{G}_s$ to $\mathcal{G}_1$, $\mathcal{G}_2$,…,$\mathcal{G}_n$ via composition. Formally, we require:

$$\mathcal{C}_s = \mathcal{C}_1 \parallel \mathcal{C}_2 \parallel \ldots. \parallel \mathcal{C}_n = (\psi_s, \phi_s),$$

$$\text{where} \quad \psi_s \wedge \psi' \text{ is satisfiable} \quad \text{and} \quad \phi_s \rightarrow \phi' \text{ is valid.}$$

Algorithm 1: Goals Selection

```
Input: Library of goals $\mathcal{L} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\}$ with contracts $\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$
       respectively, input-goal $\mathcal{G}'$ with contract $\mathcal{C}' = (\psi', \phi')$
Output: Set of goals $\mathcal{L}_c \subseteq \mathcal{L}$ that composed would result in a goal $\mathcal{G}_l$ and
        contract $\mathcal{C}_l = (\psi_l, \phi_l)$ such that $\psi_l \wedge \psi'$ is satisfiable and $\phi_l \rightarrow \phi'$ is
        valid.
1 $\mathcal{L}_c = \emptyset$ is the set of goals to be returned
2 $R = \{(\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_4), (\mathcal{G}_1, \mathcal{G}_5), (\mathcal{G}_2, \mathcal{G}_3), ...\}$ is a set of candidate compositions,
    where each element $R_i \subseteq \mathcal{L}$ and the composition of the goals contained in $R_i$
    results in a goal with a contract $\mathcal{C}_p = (\psi_p, \phi_p)$, such that
    – $\psi_p \wedge \psi'$ is a satisfiable formula
    – $\phi_p \rightarrow \phi'$ is a valid formula

3 $K = \{\mathcal{G}_i, \mathcal{G}_j, .., \mathcal{G}_m\} =$ optimal_selection($R$)
4 $\mathcal{L}_c \leftarrow K$      adds the selected goals to $\mathcal{L}_c$
5 $\mathcal{S} \leftarrow K$      where $\mathcal{S}$ is the set of goals whose assumptions need to be searched
  in the library
6 while $S \neq \emptyset$ do
7   for goal $\mathcal{G}_s$ in $\mathcal{S}$ where $\mathcal{C}_s = (\psi_s, \phi_s)$ do
7.1     $Q = \{(\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3), (\mathcal{G}_2, \mathcal{G}_5), (\mathcal{G}_1, \mathcal{G}_3), ...\}$ is a set of candidate
        compositions, where each element $Q_i \subseteq \mathcal{L}$ and the composition of the
        goals in $Q_i$ has a contract $\mathcal{C}_q = (\psi_q, \phi_q)$ such that:

          – $\psi_q \wedge \psi' \wedge \psi_s$ is a satisfiable formula
          – $\phi_q \rightarrow \psi_s$ is a valid formula

7.2     $H = \{\mathcal{G}_i, \mathcal{G}_j, .., \mathcal{G}_m\} =$ optimal_selection($Q$)
7.3     $\mathcal{L}_c \leftarrow \mathcal{L}_c \cup H$,   $\mathcal{S} \leftarrow S \cup H$
7.4     $\mathcal{S} \leftarrow \mathcal{S} - \{\mathcal{G}_s\}$              removes the $\mathcal{G}_s$ from $\mathcal{S}$
8 return $\mathcal{L}_c$
```

When connecting $\mathcal{G}_s$ to $\mathcal{G}'$ via a refinement edge, CoGoMo also uses assumption propagation, as described in Section 4.2, to ensure that $\psi' \rightarrow \psi_s$ is valid.

Algorithm 1 proposes a procedure to automatically extend a CGT leaf node. The algorithm takes as inputs a goal node $\mathcal{G}'$ and a library of goals $\mathcal{L}$, and returns the set of goals to be composed as output. We assume that each goal in the library is labeled by a *cost* that is proportional to the number of clauses in the assumptions. The cost of a solution is the sum of the costs of all the selected goals. We first choose the lowest-cost selection of goals from $\mathcal{L}$ whose guarantees, once composed, imply the guarantees of $\mathcal{C}'$. Then, we choose the lowest-cost selection of goals from $\mathcal{L}$ whose guarantees, once composed, imply the assumptions of the goals selected in the previous iteration, and repeat this step until there are no more goals in the library or there are no assumptions that can be relaxed, i.e., discharged by the guarantees of another contract from the library. Concretely, given a input-goal $\mathcal{G}'$, where $\mathcal{C}' = (\psi', \phi')$, we look for all the combinations of goals in $\mathcal{L}$ such that their composition $\mathcal{C}_l = (\psi_l, \phi_l)$, has guarantees ϕ_l that imply either the guarantees of the input goal ϕ' (line 2) or its

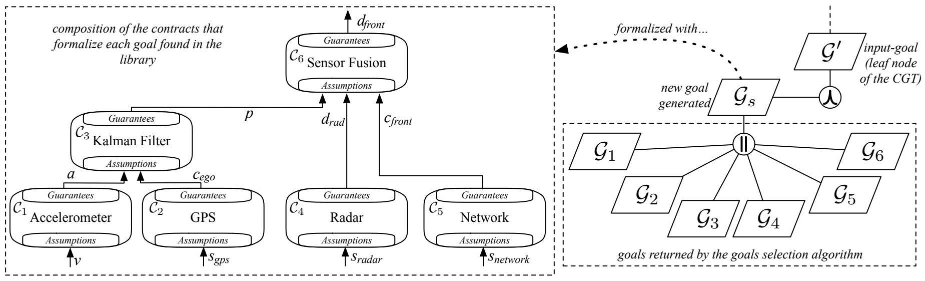


Fig. 3: Example of CGT extension via a library of goals. The left-hand side shows the composition of contracts used to formalize $\mathcal{G}_s$ and extend the CGT on the right-hand side.

assumptions ψ' (line 7.1). We evaluate the cost of all the candidate compositions and select the candidate with the lowest cost (lines 3, 7.2). If multiple candidates have the same cost, we compose all the goals in each candidate set and select the composition that has the weakest assumptions.

Our cost metric favors the selection of goals with weaker assumptions (shorter assumption formulas), as they pose less constraints to the environment and support a larger number of contexts. On the other hand, a goal with a longer assumption formula tends to accept a smaller set of environments and require a more complex aggregation of goals from the library to discharge the assumptions. However, other cost functions are also possible. Searching for a composition of goals in the library that minimizes a cost function can be exponential in the size of the library. We circumvent the worst-case complexity by adopting a greedy strategy, which select the lowest-cost goal at each iteration, even if this does not necessarily lead to a globally optimal solution. As new goals are selected, they are aggregated via composition. Therefore, at each iteration, Algorithm 1 searches for the weakest-assumption contract whose guarantees discharge the assumptions of the composite contract obtained in the previous iteration.

As an example, we extend $\mathcal{G}'$ in Figure 1, which specifies the precision with which the distance from the vehicle in front is retrieved, when this information is available. The associated contract $\mathcal{C}' = (--, \ \ dist \ \wedge \ d_{\text{front}} > 0 \ \wedge \ |d_{\text{front}} - d_{\text{real}}| < \delta)$ guarantees that the information on the distance $dist$ is available and that the perceived distance with the vehicle in front d_{front} is positive and has a precision δ in all contexts, where δ is a constant. We then use a library of goals specifying GPS modules, accelerometers, several kinds of radars with different levels of accuracy, and communication components. The extension algorithm returns 6 goals whose composition $\mathcal{G}_s$ is linked via refinement to $\mathcal{G}'$ in Figure 3.

The left-hand side of Figure 3 shows the contracts formalizing the new goal $\mathcal{G}_s$ and their interconnection structure. Each edge between contracts is labeled with a proposition that represents the logic predicate forming the assumptions or the guarantees of a contract (or both in case the guarantees of one contract imply some of the propositions in the assumptions of another contract). For example, the contract $\mathcal{C}_3$ in Figure 3, which specifies a Kalman Filter component, has

	100	200	300	400	500	600	700	800	900	1000
4	0.49	0.94	1.32	1.78	2.14	2.53	2.99	3.45	3.82	4.09
8	1.42	2.71	3.89	5.20	6.49	7.73	9.09	10.41	11.82	13.10
16	4.84	10.74	16.04	18.97	23.27	29.05	35.16	36.85	41.30	46.61
20	6.33	12.62	18.40	24.58	32.70	39.83	45.45	56.09	63.83	70.33
24	8.79	17.14	25.59	34.07	49.45	54.98	64.34	71.91	84.10	94.13

Table 1: Average execution times (sec) of 100 runs for different configurations of library size (number of goals in the columns) and contract complexity (rows).

assumptions $a \wedge c_{ego}$ and guarantees p. The composition of $\mathcal{C}_3$ with $\mathcal{C}_1 = (v, a)$ and $\mathcal{C}_2 = (s_{gps}, c_{ego})$ results in a contract, where a and c_{ego} are no longer present in the assumptions, since they are already supported by the guarantees. The net result is a simpler assumption formula. By composing all the goals retrieved from the library we obtain a new goal $\mathcal{G}_s$ and associated contract $\mathcal{C}_s = (\psi_s, \phi_s)$, where

$$\begin{aligned}\psi_s &= v \wedge s_{gps} \wedge s_{radar} \wedge s_{network} \wedge a \wedge c_{ego} \wedge p \wedge d_{rad} \wedge c_{front} \\ &\quad \vee \overline{(a \wedge c_{ego} \wedge p \wedge d_{rad} \wedge c_{front} \wedge d_{front})} \\ \phi_s &= a \wedge c_{ego} \wedge p \wedge d_{rad} \wedge c_{front} \wedge d_{front}.\end{aligned}$$

As in the previous example, the assumption formula reduces to $\psi_s = (v \wedge s_{gps} \wedge s_{radar} \wedge s_{network} \vee \overline{d_{front}})$. We observe that ϕ_s refines the guarantees of $\mathcal{C}'$ because $d_{front} = dist \wedge d_{\text{front}} > 0 \wedge |d_{\text{front}} - d_{\text{real}}| < \epsilon$ where ϵ is a constant and $\epsilon \leq \delta$. Finally, to preserve the completeness of the refinement, CoGoMo propagates the assumptions ψ_s to $\mathcal{C}'$ and then to the parent nodes of $\mathcal{G}'$ recursively, by following the edges of the CGT up to the root.

Numerical Validation. Algorithm 1 is sound and complete. The soundness is provided by the SMT solver, which checks the validity and satisfiability of the formulas. The completeness is given by the fact that the algorithm searches over the entire goal library. Because of the greedy procedure, the computation time scales linearly with the number of goals in the library. We performed numerical evaluation of synthetically generated libraries of different sizes populated by randomly generated goals. Goals are captured by simple propositional logic contracts, whose assumptions and guarantees are conjunctions of Boolean propositions. We use the length of these formulas, i.e., to quantify the complexity of each contract. A configuration is defined by the number of goals in the library and the complexity of the contracts. We evaluated the algorithm on up to 1000 library goals and up to 24 logical propositions in each contract for a total of 50 different configurations. For each configuration, we ran Algorithm 1 with 100 different input goals. Table 1 shows the average execution time for each configuration normalized by the number of goals returned by the algorithm. Figure 4 shows the execution times for 3 configurations, which scale linearly with the number of returned goals and the size of the library.[1]

[1] Complete results: http://bit.ly/3s5XD0L.

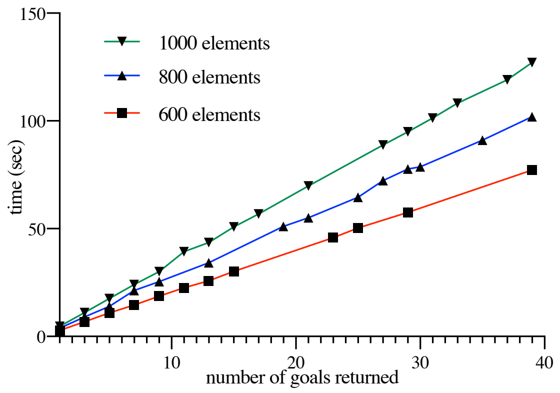


Fig. 4: Execution times as a function of the number of returned goals for 3 simulation configurations, with libraries of 600, 800, and 1000 elements, respectively, and contract complexity equal to 4.

6 Related Work

In the context of contract-based verification, tools like OCRA [5] and AGREE [7] use contracts to model system components and their aggregations and formally prove the correctness of contract refinements [6] by using model checkers [3]. Related to system engineering, CONDEnSe [28] propose a methodology and a tool that leverages the algebra of contracts to integrate artifacts developed in mechatronic systems. More oriented toward requirement engineering, the CHASE [23] framework combines a front-end formal specification language based on patterns with rigorous, contract-based verification and synthesis. It uses a declarative style to define the top-level requirements that are then translated into temporal logic, verified for consistency and, when possible, synthesized into a reactive model. CoGoMo's hierarchical and incremental approach to refinement of goal models is complementary and can be naturally incorporated into CHASE.

In the context of goal-oriented requirement engineering, significant work has addressed completeness and conflict detection using formal methods for goal models based on KAOS [12,11,1]. While these approaches mostly focus on algorithms that operate on a fixed set of requirements and environment expectations, CoGoMo proposes a step-wise approach where refinement checking and conflict analysis are performed contextually in an incremental way as the goal tree is built. Frameworks like COVER [20] use TROPOS as a goal modeling framework, Modal Transition Systems (MTS) to model the system design, and Fluent Linear Temporal Logic (FLTL) as a specification language for functional requirements. COVER checks the satisfaction of all the requirements by verifying the properties on the system model. Requirement verification using formal methods is common to the goal-oriented approaches above; however, to the best of our knowledge, CoGoMo is the first effort toward formalizing goal models using contracts, thus enhancing modularity and reuse in goal models.

Compositional synthesis of reactive systems, i.e., finding generic aggregations of reactive components such that their composition realizes a given specification, is an undecidable problem [26,17]. The problem becomes decidable by imposing a bound on the total number of component instances that can be used, but remains difficult due to its combinatorial nature [16]. Our approach relates to the one by Iannopollo et al. [16,15], proposing scalable algorithms for compositional synthesis and refinement checking of temporal logic specifications out of contract libraries. Our goal selection algorithm is, however, different, as it uses a cost function based on the complexity of a specification, and a greedy procedure that favors more compact and generic specifications (i.e., contracts with weakest assumptions) to refine the goal tree, while keeping the problem tractable.

7 Conclusions

We presented CoGoMo, a framework that guides the designer in building goal models by leveraging a contract-based formalism. CoGoMo leverages contract operations and relations to check goal consistency, completeness, and support the incremental and hierarchical refinement of goals from a library of goals. An example motivated by vehicle platooning shows its effectiveness for incrementally constructing contract-based goal trees in a modular way, with formal guarantees of correctness. Numerical results also illustrate the scalability of the proposed greedy heuristic to further extend a goal tree out of a library of goals. As future work, we plan to extend the expressiveness of CoGoMo by i) supporting contracts expressed in temporal logic and ii) supporting OR-refinements between goals by allowing optional refinement relations between multiple candidate contracts. Furthermore, we plan to improve the tool and incorporate its features into CROME [19], our recent framework for formalizing, analyzing, and synthesizing robotic missions.

Acknowledgments

The authors wish to acknowledge Alberto Sangiovanni-Vincentelli, Antonio Iannopollo, and Íñigo Íncer Romeo for helpful discussions. This work was supported in part by the Wallenberg Al Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation. In addition, the authors gratefully acknowledge the support by the US National Science Foundation (NSF) under Awards 1846524 and 1839842, the US Defense Advanced Projects Agency (DARPA) under Award HR00112010003, the US Office of Naval Research (ONR) under Award N00014-20-1-2258, and Raytheon Technologies Corporation. The views, opinions, or findings contained in this article should not be interpreted as representing the official views or policies, either expressed or implied, by the US Government. This content is approved for public release; distribution is unlimited.

References

1. Alrajeh, D., Kramer, J., Van Lamsweerde, A., Russo, A., Uchitel, S.: Generating obstacle conditions for requirements completeness. In: 2012 34th International Conference on Software Engineering (ICSE). pp. 705–715. IEEE (2012)
2. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., et al.: Contracts for system design. Foundations and Trends in Electronic Design Automation **12**(2-3), 124–400 (2018)
3. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: CAV. pp. 334–342 (2014)
4. Cheng, B.H., Atlee, J.M.: Research directions in requirements engineering. In: 2007 Future of Software Engineering. pp. 285–303. IEEE Computer Society (2007)
5. Cimatti, A., Dorigatti, M., Tonetta, S.: OCRA: A tool for checking the refinement of temporal contracts. In: ASE 2013. pp. 702–705. IEEE (2013)
6. Cimatti, A., Tonetta, S.: Contracts-refinement proof system for component-based embedded systems. Science of computer programming **97**, 333–348 (2015)
7. Cofer, D., Gacek, A., Miller, S., Whalen, M.W., LaValley, B., Sha, L.: Compositional verification of architectural models. In: NASA Formal Methods Symposium. pp. 126–140. Springer (2012)
8. Damm, W., Hungar, H., Josko, B., Peikenkamp, T., Stierand, I.: Using contract-based component specifications for virtual integration testing and architecture design. In: 2011 Design, Automation & Test in Europe. pp. 1–6. IEEE (2011)
9. Darimont, R., Van Lamsweerde, A.: Formal refinement patterns for goal-driven requirements elaboration. ACM SIGSOFT Software Engineering Notes **21**(6), 179–190 (1996)
10. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems (2008)
11. DeVries, B., Cheng, B.H.: Automatic detection of incomplete requirements via symbolic analysis. In: MODELS (2016)
12. DeVries, B., Cheng, B.H.: Automatic detection of feature interactions using symbolic analysis and evolutionary computation. In: 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). pp. 257–268. IEEE (2018)
13. Feldman, Y.A., Broodney, H.: A cognitive journey for requirements engineering. In: INCOSE International Symposium. vol. 26, pp. 430–444. Wiley Online Library (2016)
14. Ferrari, A., dell'Orletta, F., Spagnolo, G.O., Gnesi, S.: Measuring and improving the completeness of natural language requirements. In: Int. Working Conference on Requirements Engineering: Foundation for Software Quality. Springer (2014)
15. Iannopollo, A., Nuzzo, P., Tripakis, S., Sangiovanni-Vincentelli, A.: Library-based scalable refinement checking for contract-based design. In: 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1–6. IEEE (2014)
16. Iannopollo, A., Tripakis, S., Sangiovanni-Vincentelli, A.: Constrained synthesis from component libraries. Science of Computer Programming **171**, 21–41 (2019). https://doi.org/10.1016/j.scico.2018.10.003, `https://doi.org/10.1016/j.scico.2018.10.003`
17. Lustig, Y., Vardi, M.Y.: Synthesis from component libraries. International Journal on Software Tools for Technology Transfer **15**(5-6), 603–618 (2013)

18. Mallozzi, P.: CoGoMo tool - Web Interface and source code. http://cogomo.duckdns.org/, https://github.com/pierg/cogomo (2020)
19. Mallozzi, P., Nuzzo, P., Pelliccione, P., Schneider, G.: Crome: Contract-based robotic mission specification. In: 2020 18th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE). IEEE (2020)
20. Menghi, C., Spoletini, P., Ghezzi, C.: Integrating goal model analysis with iterative design. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 112–128. Springer (2017)
21. Moitra, A., Siu, K., Crapo, A., Chamarthi, H., Durling, M., Li, M., Yu, H., Manolios, P., Meiners, M.: Towards development of complete and conflict-free requirements. In: 2018 IEEE 26th International Requirements Engineering Conference (RE). pp. 286–296. IEEE (2018)
22. Nuzzo, P., Finn, J., Iannopollo, A., Sangiovanni-Vincentelli, A.L.: Contract-based design of control protocols for safety-critical cyber-physical systems. In: Proc. Design, Automation & Test in Europe Conference (DATE). pp. 1–4 (Mar 2014)
23. Nuzzo, P., Lora, M., Feldman, Y.A., Sangiovanni-Vincentelli, A.L.: CHASE: Contract-based requirement engineering for cyber-physical system design. In: Proceedings of DATE 2018. IEEE (2018)
24. Nuzzo, P., Sangiovanni-Vincentelli, A., Bresolin, D., Geretti, L., Villa, T.: A platform-based design methodology with contracts and related tools for the design of cyber-physical systems. Proc. IEEE **103**(11) (Nov 2015)
25. Nuzzo, P., Xu, H., Ozay, N., Finn, J.B., Sangiovanni-Vincentelli, A.L., Murray, R.M., Donzé, A., Seshia, S.A.: A contract-based methodology for aircraft electric power system design. IEEE Access **2**, 1–25 (2014). https://doi.org/10.1109/ACCESS.2013.2295764
26. Pneuli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: 31st Annual Symposium on Found. of Computer Science. IEEE (1990)
27. Sangiovanni-Vincentelli, A., Damm, W., Passerone, R.: Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. Eur. jou. of control **18**(3) (2012)
28. Ribeiro dos Santos, C.A., Saleh, A., Schrijvers, T., Nicolai, M.: Condense: Contract-based design synthesis. In: Proceedings of Models2019 (2019)
29. Van Lamsweerde, A.: Requirements engineering in the year 00: a research perspective. In: Proc. of the 22nd Int. conference on Software engineering (2000)
30. Van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Int. symposium on requirements engineering (2001)
31. Van Lamsweerde, A.: Requirements engineering: From system goals to UML models to software, vol. 10. Chichester, UK: John Wiley & Sons (2009)
32. Van Lamsweerde, A., Darimont, R., Massonet, P.: Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In: RE'95 (1995)
33. Wang, Y., Duan, X., Tian, D., Lu, G., Yu, H.: Throughput and Delay Limits of 802.11p and its Influence on Highway Capacity. Procedia - Social and Behavioral Sciences **96**(Cictp), 2096–2104 (2013)
34. Zowghi, D., Gervasi, V.: On the interplay between consistency, completeness, and correctness in requirements evolution. Information and Software Technology **45**(14), 993–1009 (2003). https://doi.org/10.1016/S0950-5849(03)00100-9

A Distribution of Composition over Conjunction

Given contracts $\mathcal{C}_1 = (\psi_1, \phi_1)$, $\mathcal{C}_2 = (\psi_2, \phi_2)$, and $\mathcal{C}_3 = (\psi_3, \phi_3)$, we show that

$$(\mathcal{C}_1 \wedge \mathcal{C}_2) \parallel \mathcal{C}_3 = (\mathcal{C}_1 \parallel \mathcal{C}_3) \wedge (\mathcal{C}_2 \parallel \mathcal{C}_3). \qquad (2)$$

Proof. Let (L_A, L_G) and (R_A, R_G) be the contracts on the left and right side of (2), respectively. We prove that $L_A = R_A$ and $L_G = R_G$. Both the composition and conjunction operations requires the conjunction of the guarantees, hence we obtain $L_G = R_G = \phi_1 \wedge \phi_2 \wedge \phi_3$. The assumptions of the contract on the left side can be computed as

$$L_A = (\psi_1 \vee \psi_2) \wedge \psi_3 \vee \overline{(\phi_1 \wedge \phi_2 \wedge \phi_3)} = (\psi_1 \wedge \psi_3) \vee (\psi_2 \wedge \psi_3) \vee \overline{\phi}_1 \vee \overline{\phi}_2 \vee \overline{\phi}_3.$$

On the right side, we obtain

$$\mathcal{C}_1 \parallel \mathcal{C}_3 = \Big((\psi_1 \wedge \psi_3) \vee \overline{(\phi_1 \wedge \phi_3)}, \phi_1 \wedge \phi_3\Big) \text{ and}$$
$$\mathcal{C}_2 \parallel \mathcal{C}_3 = \Big((\psi_2 \wedge \psi_3) \vee \overline{(\phi_2 \wedge \phi_3)}, \phi_2 \wedge \phi_3\Big),$$

which leads to

$$\begin{aligned} R_A &= (\psi_1 \wedge \psi_3) \vee \overline{(\phi_1 \wedge \phi_3)} \vee (\psi_2 \wedge \psi_3) \vee \overline{(\phi_2 \wedge \phi_3)} \\ &= (\psi_1 \wedge \psi_3) \vee (\psi_2 \wedge \psi_3) \vee \overline{\phi}_1 \vee \overline{\phi}_2 \vee \overline{\phi}_3. \end{aligned}$$

Finally, we also obtain $L_A = R_A$, which concludes our proof (2).