Formally Proving Compositionality in Industrial Systems with Informal Specifications

Mattias Nyberg(✉), Jonas Westman, and Dilian Gurov

Royal Institute of Technology (KTH), Stockholm, Sweden
{matny,jowestm,dilian}@kth.se
https://www.kth.se/itm/rigorous-systems-engineering

Abstract. Based upon first-order logic, the paper presents a methodology and a deductive system for proving compositionality. Typical specifications found in industry are not expressed in any formal notation; rather most often in natural language. Therefore, the methodology does not assume specifications to be formal logical sentences. Instead, the methodology takes as input, properties of specifications and in particular, refinement relations. To cover general industrial heterogeneous systems, the semantics chosen is behavior based, originating in previous work on contract-based design for cyber-physical systems. In contrast to the previous work, implementation of specifications is non-monotonic with respect to composition. That is, even though a specification is implemented by one component, a composition with a second component may not implement the same specification. This kind of non-monotonicity is fundamentally important to support architectural specifications and so-called freedom-of-interference used in design of safety critical systems.

1 Introduction

To enable verification of large scale systems, the need for *compositional verification* is well known [5,13,14]. The survey paper [14], one of the most cited ones in the area of compositional verification, decribes the principle of *compositional verification* of software programs as presented below, but here by using the notation and terminology of the present paper.

Consider $\mathbf{c}$ to be a component being a composition of a set of components $\mathbf{c}_1, \ldots \mathbf{c}_n$. According to [14], to ensure by *compositional verification* that $\mathbf{c}$ implements a specification $\mathbf{S}$ amounts to the following steps:

a) Find specifications $\mathbf{S}_1, \ldots \mathbf{S}_n$ for $\mathbf{c}_1, \ldots \mathbf{c}_n$ such that steps (b) and (c) below can be executed.
b) Prove that any component c implements $\mathbf{S}$ whenever c is composed of any components c_i implementing specifications $\mathbf{S}_i$, $i = 1, \ldots n$. This is called the *compositionality proof problem* and involves the specifications $\mathbf{S}_1, \ldots \mathbf{S}_n$ and $\mathbf{S}$ only.
c) Verify that each $\mathbf{c}_i$, $i = 1, \ldots n$, implements specification $\mathbf{S}_i$.

T. Margaria and B. Steffen (Eds.): ISoLA 2020, LNCS 12478, pp. 348–365, 2020.
https://doi.org/10.1007/978-3-030-61467-6_22

In [14], all the specifications $\mathbf{S}$ and $\mathbf{S}_i$, $i = 1, \ldots n$, are assumed to be expressed using *logical sentences only*. It is also assumed that specifications are composed using a composition operator op^S such that, given that each component $\mathbf{c}_i$ implements $\mathbf{S}_i$, the composition of components implements $op^S(\mathbf{S}_1, \ldots, \mathbf{S}_n)$. Then, the compositionality proof problem reduces to showing that $op^S(\mathbf{S}_1, \ldots, \mathbf{S}_n)$ logically entails $\mathbf{S}$.

The problem we are solving in the present paper is the *compositionality proof problem*, but with the following generalizations and modifications in relation to [14].

- We consider the general domain of heterogeneous systems. That is, a component may be a software program, as in [14], but we allow also mechanical and electrical components.
- Specifications are industrial engineering objects, and as such we do not presume these to be expressed in any logical language. The motivation is that a typical specification found in industry is not expressed in any formal notation; instead it is most often informally expressed in natural language [11].

Exactly like [14], we deal with the compositionality proof problem, i.e. the problem of proving that the composition of a given set of specifications $\mathbf{S}_1, \ldots \mathbf{S}_n$ "implies" another given specification $\mathbf{S}$. However, *due to that we do not assume specifications to be expressed as logical sentences*, in contrast to [14], we can not rely on a solution based on proving that the composition $op^S(\mathbf{S}_1, \ldots, \mathbf{S}_n)$ *logically entails* $\mathbf{S}$. Instead, we represent specifications formally as constant symbols, and rather than formalizing the content of the specifications, we formalize refinement relations between specifications and certain properties of the specifications. These formalizations are then used to prove compositionality.

Our aim is industrial software-intensive heterogeneous systems, meaning the solution needs to support both discrete and continuous systems. We have therefore chosen a *behavior based semantics* of components and specifications originating in previous work developed to support "contract-based design for cyber-physical systems" [1,2,15–17].

The previous frameworks for contracts-based design [1,2,15,16] also give some support for proving compositionality in the form of their definition of "parallel composition" as a condition on the involved specifications. However, their condition is based upon the assumption that *implementation is monotonic with respect to component composition*. In contrast to these previous works, the present paper does *not* make this assumption; instead we support the case when *implementation is non-monotonic with respect to composition* [17]. That is, even though a component $\mathbf{c}_1$ implements a specification $\mathbf{S}$, it should *not* hold generally that the composition of $\mathbf{c}_1$ with another component $\mathbf{c}_2$ also implements $\mathbf{S}$.

A support for this kind of non-monotonicity is important since architectural specifications typically state that components shall not read and not write any other signals than those included in a defined interface. This is inheritly a "non-monotonic property" since, for example, consider a component $\mathbf{c}_1$ that does not write to a signal x, but another component $\mathbf{c}_2$ does. Component $\mathbf{c}_1$ clearly implements the specification "Signal x shall not be written to." but when composed

with component $\mathbf{c}_2$, this specification is no longer implemented. Thus, implementation is non-monotonic with respect to composition. The property that certain components shall not read or write certain variables is also fundamental when designing safety critical systems and in those cases referred to as *freedom of interference* [9]. In conclusion, any framework for proving compositionality, and that is capable of including also architectural specifications, needs to support the case when implementation is non-monotonic with respect to composition. The proposed solution generally supports the non-monotonic case, but it also gives specific support to, and can explicitly utilize, the case when implementation is indeed monotonic with respect to component composition, which is still a common case for normal functional specifications.

With all these observations in mind, we propose as a first contribution of the paper, a *formal framework* that enables formal reasoning about relationships between components and specifications, and most importantly, it does not require specifications to be formally expressed. The framework is based upon first-order predicate logic and consists of a *formal language*, presented in Sect. 2, a *formal semantics*, presented in Sect. 3, and a number of *derived formal properties*, presented in Sect. 4. Based upon the proposed formal framework, Sect. 5 presents the second contribution of the paper, a *methodology and a deductive system for proving compositionality*. Proofs of propositions and theorem in the paper have been left out but can be found in the report [12].

2 Syntax

In the framework of first-order predicate logic [8], this section presents the grammars defining the syntactic categories used in the class of languages considered. This forms a formal syntax for the contracts theory in [17] and is part of the proposed formal framework, i.e. the first contribution of the paper. We consider languages parameterized by two disjoint sets of symbols $\mathbb{C}$ and $\mathbb{S}$. A *component term* is formed by the grammar

$$c ::= \mathbf{c} \mid c \times c \mid q$$

where $\mathbf{c} \in \mathbb{C}$ and $\mathbb{C}$ is a set of component constant symbols, $\times$ is a component composition function symbol, and q ranges over component variables.

A *specification term* is formed by the grammar

$$S ::= \mathbf{S} \mid S \sqcap S \mid (S, S) \mid \ S \| S \ \mid \ V$$

where $\mathbf{S} \in \mathbb{S}$ and $\mathbb{S}$ is a set of specification constants; $\sqcap$, $(\cdot, \cdot)$, and $\|$ are specification composition function symbols; and V ranges over specification variables. A specification term of the kind $S_1 \sqcap S_2$ will be referred to as *conjunction* of specifications. A specification term of the kind (S_1, S_2) will be referred to as *assume-guarantee contract*, or for short, *contract*. The first specification term in (S_1, S_2) is called *assumption* and the second *guarantee*. A specification term of the kind $S_1 \| S_2$ will be referred to as *parallel composition* of specifications.

As an extension to the contracts theory in [17], we introduce two new types of specification constants, namely ©, called the *compatibility specification*, and $\top_{\|}$, called the *top specification*. As will be explained in detail in Sect. 3 and 4, the compatibility specification © is used to enforce composed components to be compatible with each other, and the top specification $\top_{\|}$ is used to allow contracts to be a most general form of specification. Both © and $\top_{\|}$ are included among the specification constants $\mathbb{S}$.

We introduce three predicate symbols; the only ones considered in the paper. First, *implementation* written $c : S$, which reads *c implements S*. Second, *refinement* written $S_1 \sqsubseteq S_2$, which reads S_1 *refines* S_2. Third, $\mathsf{Assertional}(S)$, which is a new concept and an extension of the contracts theory in [17]. As will be explained in detail in Sect. 3 and 4, the purpose of $\mathsf{Assertional}(S)$ is to give explicit support for cases when implementation is monotonic with respect to component composition, since in general, the proposed framework otherwise assumes non-monotonicity.

As usual in predicate logic [8], *formulas* are recursively built by combining the three kind of predicates with the first-order logical symbols $\forall$, $\exists$, $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $=$. Furthermore, a *sentence* is a formula without free variables.

In summary, we have introduced a class of formal languages in which each language, denoted $\mathcal{L}_{\mathbb{C},\mathbb{S}}$, has the *signature*, i.e. non-logical symbols:

- the function symbols $\times$, $\sqcap$, $\|$, and $(\cdot,\cdot)$
- the predicate symbols $:$, $\sqsubseteq$, and $\mathsf{Assertional}(\cdot)$
- component constant symbols instantiated by a set $\mathbb{C}$
- specification constant symbols instantiated by a set $\mathbb{S}$, which includes the two specification constant symbols © and $\top_{\|}$.

As seen each language is completely specified by the disjoint sets $\mathbb{C}$ and $\mathbb{S}$.

3 Semantics

In order to formally reason about compositionality, and as part of the proposed formal framework, the present section introduces semantics for the syntactic categories given in the previous section.

In a given engineering context, component terms and specification terms represent real-world components and specifications respectively. When reasoning about whether real-world components implement real-world specifications, we are not so interested in the components and specifications themselves. Instead, of relevance are only the *behaviors* of components and behaviors that specifications specify. Therefore, given an engineering context, we define a *semantics* based upon *behavior of a real-world component* and *behavior set of real-world specification*. Formally, the engineering context is represented by a *model* $\mathcal{M}$. Since our interest is only the behaviors, the model $\mathcal{M}$ "bypasses" the real-world objects and provides a mapping directly from terms to behaviors and behavior sets, and from predicate symbols to relations between behaviors and behavior sets.

The first subsection below defines *behavior* formally, by reusing concepts from previous work in [1,2,16] and [17]. The next subsections then use this definition of behavior to define the semantics of terms and predicates. Many more detailed explanations and motivations follow later in Sect. 4.

3.1 Behavior and Behavior Set

Let *the universal set of variables* $\Xi = \{x_1, \ldots, x_{N_v}\}$, $N_v \geq 1$, denote the set of variables considered. These variables represent measurable or immeasurable quantities of interest in the context studied. As such, a variable is typically a function of time representing the fact that its value changes over a time window. Next, in accordance with [16], let a *run* be a vector, with the elements being variables in Ξ. For example, a run can be a *trace* [3,4,18] or an *execution* [10].

Let Ω denote the considered set of possible runs over the variables Ξ. Note that for the same set of variables Ξ, infinitely many different sets Ω can be chosen by varying the domain of each variable and the time window over which runs are defined. Now, a *behavior* B is a, possibly empty, set of runs, i.e. $\mathsf{B} \subseteq \Omega$. A *behavior set* $\mathcal{Q}$ is a, possibly empty, set of behaviors, i.e. $\mathcal{Q} \subseteq \mathcal{P}(\Omega)$, where $\mathcal{P}$ denotes *power set*.

3.2 The Model Class *Behavior Semantics*

In general and according to standard predicate logic [8], a model $\mathcal{M}$ for a language $\mathcal{L}$ is a pair of an *interpretation* for the language $\mathcal{L}$ and a *domain of discourse* $\mathcal{D}$.

For a given arbitrary set Ω, we will here consider a class of models $\mathbb{M}_{\Omega,\mathbb{C},\mathbb{S}}$ for the language $\mathcal{L}_{\mathbb{C},\mathbb{S}}$. The class is defined by the following two constraints applying to each model in $\mathbb{M}_{\Omega,\mathbb{C},\mathbb{S}}$:

- The domain of discourse is $\mathcal{D}_\Omega = \mathcal{P}(\Omega) \cup \mathcal{P}(\mathcal{P}(\Omega))$, i.e. $\mathcal{D}_\Omega$ is the union of the set of all possible behaviors and the set of all possible behavior sets.
- The interpretation conforms to a set of constraints (1), (2), and (3), presented in the next section.

Let *behavior semantics* refer to the class $\mathbb{M}_{\mathbb{C},\mathbb{S}}$ that we define to be the union of all model classes $\mathbb{M}_{\Omega,\mathbb{C},\mathbb{S}}$ for all possible sets Ω.

3.3 Constraints on the Interpretation

We will below introduce constraints applying to the interpretation of each model $\mathcal{M} \in \mathbb{M}_{\Omega,\mathbb{C},\mathbb{S}}$. As stated above, each such model $\mathcal{M}$ has a domain of discourse generated by the set Ω. Furthermore, since $\mathcal{M}$ is a model for some language $\mathcal{L}_{\mathbb{C},\mathbb{S}}$, the interpretation is a mapping from each of the symbols in $\mathcal{L}_{\mathbb{C},\mathbb{S}}$, as presented in Sect. 2, to $\mathcal{D}_\Omega$.

The interpretation of component constant symbols is a mapping from component constant symbols to behaviors. The interpretation of the function symbol *component composition* $\times$ is a function from $(\mathcal{P}(\Omega))^2$ to $\mathcal{P}(\Omega)$. That is,

$$q^{\mathcal{M}} \qquad \in \mathcal{P}(\Omega) \tag{1a}$$

$$\mathsf{B}_1 \times^{\mathcal{M}} \mathsf{B}_2 = \mathsf{B}_1 \cap \mathsf{B}_2. \tag{1b}$$

The interpretation of the component composition symbol reveals that it is associative, commutative, and idempotent, i.e. $(c_1 \times c_2) \times c_3 = c_1 \times (c_2 \times c_3)$, $c_1 \times c_2 = c_2 \times c_1$ and $c \times c = c$ respectively.

The interpretation of specification constant symbols is a mapping from specification constant symbols to behavior sets. The interpretation of the function symbols *specification conjunction*, *specification parallel composition*, and *contract* are functions from $(\mathcal{P}(\mathcal{P}(\Omega)))^2$ to $\mathcal{P}(\mathcal{P}(\Omega))$. The interpretation of specification parallel composition is defined using a *double intersection of sets*, i.e. $\mathcal{Q}_1 \Cap \mathcal{Q}_2 = \{\mathsf{B}_1 \cap \mathsf{B}_2 |\ \mathsf{B}_1 \in \mathcal{Q}_1, \mathsf{B}_2 \in \mathcal{Q}_2\}$.

$$p^{\mathcal{M}} \qquad \in \mathcal{P}(\mathcal{P}(\Omega)) \tag{2a}$$

$$\top_{\|}^{\mathcal{M}} \qquad = \{\Omega\} \tag{2b}$$

$$\textcircled{c}^{\mathcal{M}} \qquad = \{\mathsf{B} \in \mathcal{P}(\Omega) |\ \mathsf{B} \neq \emptyset\} \tag{2c}$$

$$\mathcal{Q}_1 \sqcap^{\mathcal{M}} \mathcal{Q}_2 = \mathcal{Q}_1 \cap \mathcal{Q}_2 \tag{2d}$$

$$\mathcal{Q}_1 \|^{\mathcal{M}} \mathcal{Q}_2 = \mathcal{Q}_1 \Cap \mathcal{Q}_2 \tag{2e}$$

$$(\mathcal{A}, \mathcal{G})^{\mathcal{M}} = \{\mathsf{B} \in \mathcal{P}(\Omega) |\ \forall \mathsf{B}' \in \mathcal{A}.\, \mathsf{B} \cap \mathsf{B}' \in \mathcal{G}\} \tag{2f}$$

We can note that both the interpretations of the conjunction and parallel composition symbols are associative, commutative, and idempotent. However, the interpretation of contract has none of these properties.

As common in predicate logic, we extend the interpretation to non-constant terms, e.g. $(c_1 \times c_2)^{\mathcal{M}} = c_1^{\mathcal{M}} \times^{\mathcal{M}} c_2^{\mathcal{M}}$ and $(A, G)^{\mathcal{M}} = (A^{\mathcal{M}}, G^{\mathcal{M}})^{\mathcal{M}}$.

Next, we consider the interpretation of the three predicate symbols. In predicate logic, interpretation of each predicate is usually defined by a relation over $\mathcal{D}^n$ where n is the arity of the predicate. Using this principle, the interpretation of the three predicate symbols is as follows:

$$:^{\mathcal{M}} \qquad = \{(\mathsf{B}, \mathcal{Q}) \in \mathcal{P}(\Omega) \times \mathcal{P}(\mathcal{P}(\Omega)) \,|\, \mathsf{B} \in \mathcal{Q}\} \tag{3a}$$

$$\sqsubseteq^{\mathcal{M}} \qquad = \{(\mathcal{Q}_1, \mathcal{Q}_2) \in \mathcal{P}(\mathcal{P}(\Omega)) \times \mathcal{P}(\mathcal{P}(\Omega)) \,|\, \mathcal{Q}_1 \subseteq \mathcal{Q}_2\} \tag{3b}$$

$$\mathsf{Assertional}^{\mathcal{M}} = \{\mathcal{Q} \in \mathcal{P}(\mathcal{P}(\Omega)) \,|\, \mathcal{Q} \text{ is downward closed}\} \tag{3c}$$

where *downward closed* refers to the general set property that for each $\mathsf{B} \in \mathcal{Q}$, it holds that each subset $\mathsf{B}' \subseteq \mathsf{B}$ is also in $\mathcal{Q}$, i.e. $\mathsf{B}' \in \mathcal{Q}$.

3.4 Evaluation

As standard in predicate logic, given a model $\mathcal{M}$ for a language $\mathcal{L}_{\mathbb{C},\mathbb{S}}$, evaluation of a *sentence* ϕ in $\mathcal{L}_{\mathbb{C},\mathbb{S}}$, is done by first using the interpretation of the constants

to find the corresponding concrete elements in $\mathcal{D}_\Omega$, i.e. behaviors and behavior sets. If ϕ is a single predicate, by iteratively using the interpretation of the function symbols, we compute one element, or a pair of elements, depending on the arity of the actual predicate symbol. The element(s) are then checked in the relation obtained from the interpretation of the predicate symbol. If and only if the element(s) are in the relation, we say that $\mathcal{M}$ *satisfies* ϕ , or ϕ is *true* in $\mathcal{M}$, and write $\mathcal{M} \models \phi$. If ϕ is a formula built with several predicates combined by logical symbols, it is evaluated by using the standard usage of the logical symbols, e.g. $\mathcal{M} \models \phi_1 \vee \phi_2$ if and only if $\mathcal{M} \models \phi_1$ or $\mathcal{M} \models \phi_2$.

A set of formulas $\Psi = \{\psi_1, \ldots \psi_N\}$ *semantically entails* a formula ϕ, denoted $\Psi \models \phi$, if it holds that any model $\mathcal{M}$ that satisfies each formula $\psi_i \in \Psi$ also satisfies the formula ϕ.

3.5 Theory

According to standard first order logic, we use the concept of *theory* [6] in order to obtain a syntactical characterization of the model class considered. Let $T_{\mathbb{C},\mathbb{S}}$ be the *theory* of model class $\mathbb{M}_{\mathbb{C},\mathbb{S}}$. That is, $T_{\mathbb{C},\mathbb{S}}$ is the set of all first-order sentences such that each sentence is satisfied by every model in $\mathbb{M}_{\mathbb{C},\mathbb{S}}$. The following proposition explains how general semantical-entailment properties can be proven by refering to the theory $T_{\mathbb{C},\mathbb{S}}$, and is the basis for all other propositions and theorem in the paper.

Proposition 1. *Let* $\phi_i(t_1, ..., t_m)$, $i = 1..n$, *and* $\varphi(t_1, ..., t_m)$ *represent sentences in* $\mathcal{L}_{\mathbb{C},\mathbb{S}}$. *If for each model* $\mathcal{M} \in \mathbb{M}_{\mathbb{C},\mathbb{S}}$, *such that* $\mathcal{M} \models \phi_i(t_1, ..., t_m)$, *it holds also* $\mathcal{M} \models \varphi(t_1, ..., t_m)$, *then* $T_{\mathbb{C},\mathbb{S}}, \phi_1(t_1, ..., t_m), \ldots, \phi_n(t_1, ..., t_m) \models \varphi(t_1, ..., t_m)$. □

4 Explanations and Connection to Real-World Engineering

This section presents more detailed explanations of and motivations for the behavior semantics presented in Sect. 3. We also present a number of propositions stating important properties of the proposed formal framework, i.e. the first contribution of the paper. In the sequel, most results and discussions will be presented *without* reference to a specific language $\mathcal{L}_{\mathbb{C},\mathbb{S}}$. Still, it means that each result is valid only for a given implicit language $\mathcal{L}_{\mathbb{C},\mathbb{S}}$, but since that language is arbitrary in the class considered, the results and discussions will be valid for all such languages.

We consider *real-world components* and *real-world specifications* to be engineering artifacts existing in an engineering context. In the following three subsections, with the ambition to strive for clarity, we carefully distinguish between component *terms*, *real-world* components that are represented by component terms, and *behavior* of real-world components represented by the interpretation of the terms. Similarly we carefully distinguish between specification terms, real-world specifications, and behavior sets. However, in the rest of paper, when the context is sufficiently precise, we will often refer to just *component* or *specification*.

4.1 Components and Component Compositions

We are considering an engineering context represented by a model $\mathcal{M}$. The component constant symbol $\mathbf{c}$ represents a real-world component in that context. The interpretation $\mathbf{c}^{\mathcal{M}}$ is the behavior of this real-world component. As seen, the real-world component is not explicit in our formalism, although we could have introduced a second interpretation containing a mapping from the symbols $\mathbb{C}$ to real-world components. However, since all our reasoning is about *behavior* components, such a second interpretation would not bring any extra usefulness.

The real-world component, represented by the symbol $\mathbf{c}$, has the behavior $\mathbf{c}^{\mathcal{M}}$. In accordance with Sect. 3.1, this behavior captures, through the runs it includes, the dynamic and static constraints imposed by the real-world component on the variables in Ξ, independent of constraints imposed by other real-world components.

Example 1. Consider an electrical amplifier that takes an input signal u and creates an output signal y, with amplification factor 2. The relationship between the input u and output y can be described by the equation $y = 2u$. Let $\Xi = \{u, y\}$, where u and y are real variables. For simplicity, we consider only one point in time, i.e. each run is a vector of values, not a vector of functions of time. The behavior of the amplifier is then the infinite set of all vectors (u, y) that are solutions to $y = 2u$. That is $\mathsf{B} = \{(0,0), (0.1, 0.2), (1,2), (2,4), (15,30), \dots\}$. □

Example 2. Let $\Xi = \{x, y\}$, where x and y are Boolean variables. For simplicity, we consider only one point in time. Examples of behaviors are $\mathsf{B}_1 = \{(0,0)\}$, $\mathsf{B}_2 = \{(0,1)\}$, $\mathsf{B}_3 = \{(0,0), (0,1)\}$, $\mathsf{B}_4 = \{(0,0), (1,1)\}$, and $\mathsf{B}_5 = \emptyset$. □

Example 3. Let $\Xi = \{x, y\}$, where u and y are real variables. Examples of runs are $\omega_1 = (x(t), y(t)) = (t, e^t)$ and $\omega_2 = (t, 2e^t)$ defined on a time window $[0, 10]$. These two runs can be combined to form four different behaviors $\mathsf{B}_1 = \{\}$, $\mathsf{B}_2 = \{(t, e^t)\}$, $\mathsf{B}_3 = \{(t, 2e^t)\}$, and $\mathsf{B}_4 = \{(t, e^t), (t, 2e^t)\}$. □

Conceptually, composing two real world components means combining the constraints imposed individually by the components. Thus the behavior $(\mathbf{c}_1 \times \mathbf{c}_2)^{\mathcal{M}}$ captures the combined dynamic and static constraints imposed by both real world components. That is, the first real world component allows the runs $\mathbf{c}_1^{\mathcal{M}}$ and the second allows the runs $\mathbf{c}_2^{\mathcal{M}}$. Together, the two real world components allow only runs that are in both $\mathbf{c}_1^{\mathcal{M}}$ and $\mathbf{c}_2^{\mathcal{M}}$. Thus, the behavior of the composed real world component is the intersection $(\mathbf{c}_1 \times \mathbf{c}_2)^{\mathcal{M}} = \mathbf{c}_1^{\mathcal{M}} \cap \mathbf{c}_2^{\mathcal{M}}$.

In the paper, we will mostly write expressions involving general component terms. A term c may therefore be either a constant symbol or a composition of constant symbols. If it is a composition, it represents a corresponding composition of real-world components. Even though the term c is a composition,

representing a real-world composition, we will refer to it as a real-world component. That is, we always consider a composition of real-world components to be a new real-world component even though there may be no explicit component constant symbol representing this composed real-world component.

4.2 Specification

Real-world specifications are usually defined in some requirements management system. A specification constant symbol $\mathbf{S}$ represents such a real-world specification. A real-world specification expresses, formally or informally, an *intended property* in terms of the variables Ξ. Typical examples are functional requirements or interface requirements. Note that, regardless of if a real-world specification is expressed formally or informally in natural language, we presume that the specified intended property is *unambiguous*, in line with standards on requirements engineering e.g. [9].

The intended property is in our framework characterized by $\mathbf{S}^{\mathcal{M}}$, which is the set of behaviors consistent with the intended unambiguous property. For example, if a real world specification represented by $\mathbf{S}$ is expressed as "The signal x shall be larger than zero.", then the behavior set $\mathbf{S}^{\mathcal{M}}$ consists of all behaviors in which all runs have the value of the variable x larger than 0 at all time points.

Example 4. Let $\Xi = \{x, y, z\}$ where each variable is Boolean, and consider only one point in time. Consider a real-world specification represented by $\mathbf{S}_1$ and expressed as "The variable x shall be 0, y shall be constrained to either 0 or 1, and z shall not be constrained at all.". Consider also real-world specification represented by $\mathbf{S}_2$ and expressed as "Either x shall be 0 and z not be constrained, or z shall be 0 and x not be constrained. In both cases, y shall be 1.". This corresponds to the behavior sets

$$\mathbf{S}_1^{\mathcal{M}} = \{\{(0,0,0),(0,0,1)\},\{(0,1,0),(0,1,1)\}\} \tag{4a}$$

$$\mathbf{S}_2^{\mathcal{M}} = \{\{(0,1,0),(0,1,1)\},\{(0,1,0),(1,1,0)\}\} \tag{4b}$$

□

As seen in the example, the framework allows specifications that specify properties, both of the kind that variables shall take certain values, but also of the kind stating that some variables shall *not* be constrained. The latter is a typical property enforced by architecture specifications and requirements of freedom of interference, and as further discussed in the following sections, closely related to implementation being non-monotic with respect to composition.

4.3 Implementation

As stated above, a real world component, represented by a term c, has a behavior $c^{\mathcal{M}}$ that is the set of all runs that are possible with the constraints imposed by the component. A real world specification, represented by a symbol S, expresses an

intended property, and the behavior set $S^{\mathcal{M}}$ is the set of all behaviors consistent with this intended property. Based upon these notions, a real-world component, represented by c, *implements* a real-world specification, represented by S, if the behavior $c^{\mathcal{M}}$ is in the set $S^{\mathcal{M}}$. We express this as $\mathcal{M} \models c{:}S$.

Example 5. Consider again the real-world specifications represented by $\mathbf{S}_1$ and $\mathbf{S}_2$ from Example 4. Consider also three real-world components represented by $\mathbf{c}_3$, $\mathbf{c}_4$, and $\mathbf{c}_5$, having behaviors $\mathbf{c}_3^{\mathcal{M}} = \{(0,0,0),(0,0,1)\}$, $\mathbf{c}_4^{\mathcal{M}} = \{(0,1,0),(0,1,1)\}$, and $\mathbf{c}_5^{\mathcal{M}} = \{(0,1,0),(1,1,0)\}$ respectively. Notably, it holds that $\mathcal{M} \models \mathbf{c}_3 : \mathbf{S}_1$ and $\mathcal{M} \models \mathbf{c}_4 : \mathbf{S}_1$, but since $\mathbf{c}_3^{\mathcal{M}} \cap \mathbf{c}_4^{\mathcal{M}} = \emptyset \notin \mathbf{S}_1^{\mathcal{M}}$, it does *not* hold that $\mathcal{M} \models \mathbf{c}_3 \times \mathbf{c}_4 : \mathbf{S}_1$. Furthermore, it holds that $\mathcal{M} \models \mathbf{c}_4 : \mathbf{S}_2$ and $\mathcal{M} \models \mathbf{c}_5 : \mathbf{S}_2$, but since $\mathbf{c}_4^{\mathcal{M}} \cap \mathbf{c}_5^{\mathcal{M}} = \{(0,1,0)\} \notin \mathbf{S}_2^{\mathcal{M}}$, it does *not* hold that $\mathcal{M} \models \mathbf{c}_4 \times \mathbf{c}_5 : \mathbf{S}_2$. □

Example 5 highlights the fact that even though two real-world components individually satisfy the same real-world specification, their composition does in general *not*. That is, *implementing a specification is non-monotonic with respect to composition*.

4.4 Refinement

From now on, we will stop being so careful in the dinstinction between terms and real-world objects.

In general, *refinement* is the single most important property used to prove compositionality, as will be further highlighed in Sect. 5.1. Refinement means, in words, that if any component c implements specification S_1, and S_1 *refines* S_2, then c will implement also S_2. The following proposition confirms that the interpretation constraints (2) and (3) match this notion of refinement.

Proposition 2 (Refinement Elimination (re) **and Introduction).** *It holds*

a) $T_{\mathbb{C},\mathbb{S}}, c{:}S_1, S_1 \sqsubseteq S_2 \models c{:}S_2$ *and* (re)
b) $T_{\mathbb{C},\mathbb{S}}, \forall q(q{:}S_1 \rightarrow q{:}S_2) \models S_1 \sqsubseteq S_2$. □

4.5 Assertional Specification

The proposed framework, as noted many times by now, supports generally the case when implementation is non-monotonic with respect to component composition. However, in many cases, implementation of a specification S is indeed monotonic and in order to support reasoning utilizing this fact, the framework incorporates the predicate $\mathsf{Assertional}(S)$. If a specification S is assertional and a component c_1 implements S, i.e. $c_1 : S$, then any composition with any other component c_2 will also implement S, i.e. $c_1 \times c_2 : S$. Thus, if a specification S is assertional, then implementation of S will be monotonic with respect to component composition. This relationship is indeed a consequence of the interpretation constraints (1), (2), and (3), as confirmed by the following proposition.

Proposition 3 (Assertional vs Monotonic (am)**).** *It holds*

a) $T_{\mathbb{C},\mathbb{S}},\ \forall q_1 \forall q_2 \big(q_1 : S \rightarrow q_1 \times q_2 : S\big)\ \models\ \mathsf{Assertional}(S)$ *and*
b) $T_{\mathbb{C},\mathbb{S}}, \mathsf{Assertional}(S), c_1 : S\ \models\ c_1 \times c_2 : S$. (am)

□

Consider a specification **S** expressing an intended property of the kind of a simple relation between variables in Ξ, e.g. "x shall be larger than y". According to Sect. 4.2 the behavior set $\mathbf{S}^{\mathcal{M}}$ contains all behaviors consistent with the relation, which means all behaviors in which each run respects the relation. Since any subset, including the empty set, of such a behavior will also have all its runs respecting the relation, any such subset is also a behavior in $\mathbf{S}^{\mathcal{M}}$. Recall from Sect. 3.3 that this corresponds to the definition of downward-closed set and the interpretation of $\mathsf{Assertional}(S)$. Thus, in general it holds that any property of individual runs lifted to sets of runs, is assertional. This includes all properties that are expressible in linear-time temporal logic. In particular, a specification becomes assertional if it is possible to express the specification as a simple relation between variables in Ξ, or a combination of such relations, and this should be the case for a majority of industrial specifications. Note however, that the empty behavior is rarely a desired behavior, but it can be excluded simply by means of the specification © as will be explained in Sect. 4.8.

An important exception is specifications expressing architectural contraints, e.g. "signal x shall not be sent out" or "memory location 0x2AF3 shall not be written to", sometimes refered to as a requirement on *freedom of interference* [9]. The specifications $\mathbf{S}_1$ and $\mathbf{S}_2$ in Example 4 are two examples of specifications that are not assertional. This can be seen by studying the behavior sets (4) and also by observing that their natural language formulations include the phrase "not constrained" which is a type of architectural specification.

4.6 Conjunction and Parallel Composition of Specifications

Conjunction of two specifications simply means that the component shall implement both specifications, as confirmed by the following proposition. The following proposition confirms that the interpretation constraints (2) and (3) match this notion of refinement.

Proposition 4 (Conjunction Introduction ($\sqcap$i) **and Elimination** ($\sqcap$e)**).** *It holds*

a) $T_{\mathbb{C},\mathbb{S}}, c : S_1, c : S_2 \models c : S_1 \sqcap S_2$ *and* ($\sqcap$i)
b) $T_{\mathbb{C},\mathbb{S}}, c : S_1 \sqcap S_2 \models c : S_1$. ($\sqcap$e)

□

Parallel composition of two specifications means that, if a component c implements $S_1 \,\|\, S_2$, the behavior of the component is possible to "factorize" into two behaviors $\mathsf{B}_1 \in S_1^{\mathcal{M}}$ and $\mathsf{B}_2 \in S_2^{\mathcal{M}}$ such that $c^{\mathcal{M}} = \mathsf{B}_1 \cap \mathsf{B}_2$. In practice, parallel composition of specification $S_1 \,\|\, S_2$ is typically used as part of a refinement relation $S_1 \,\|\, S_2 \sqsubseteq S$, which means that, if the refinement relation holds, we can check that components c_1 and c_2 implement S_1 and S_2 respectively, and then

it follows that the composition $c_1 \times c_2$ implements S. The following proposition confirms that the interpretation constraints (1), (2), and (3) match both these views of parallel composition.

Proposition 5 (Parallel Specification Composition). *It holds*

a) $T_{\mathbb{C},\mathbb{S}}, c{:}S_1 \,\|\, S_2 \models \exists q_1, q_2 (q_1{:}S_1 \wedge q_2{:}S_2 \wedge q_1 \times q_2 = c)$ *and*
b) $T_{\mathbb{C},\mathbb{S}}, c_1{:}S_1, c_2{:}S_2 \models c_1 \times c_2{:}S_1 \,\|\, S_2$. □

The following proposition investigates the relationship between conjunction and parallel composition.

Proposition 6 (Conjunction vs Parallel Composition). *It holds*

a) $T_{\mathbb{C},\mathbb{S}} \models S_1 \sqcap S_2 \sqsubseteq S_1 \,\|\, S_2$ *and*
b) $T_{\mathbb{C},\mathbb{S}},\ \mathsf{Assertional}(\mathsf{S}_1),\ \mathsf{Assertional}(\mathsf{S}_2)\ \models\ S_1 \,\|\, S_2 \sqsubseteq S_1 \sqcap S_2$. □

Part (b) of the proposition is interesting since it, in the case S_1 and S_2 are assertional, provides an indirect way to prove $S_1 \,\|\, S_2 \sqsubseteq S$ from knowing that $S_1 \sqcap S_2 \sqsubseteq S$, which in some cases might be simpler to prove.

4.7 Contracts

Traditionally in the literature, e.g. see [1,2,15,16], an assume-guarantee contract (A, G) is defined by the property that: a component c_2 implements a contract (A, G) if for any component c_1 that implements A, the composition $c_1 \times c_2$ implements G. The following proposition confirms that our way of defining contract *as the constraint* (2f) *on the interpretation*, leads to this property used as definition in the previous literature.

Proposition 7 (Contract Elimination (ce) **and Introduction** (ci)**).** *It holds*

a) $T_{\mathbb{C},\mathbb{S}} \models A \,\|\, (A, G) \sqsubseteq G$ *alternatively expressed as*
$T_{\mathbb{C},\mathbb{S}}, c_1{:}A, c_2{:}(A, G) \models c_1 \times c_2{:}G$, (ce)
b) $T_{\mathbb{C},\mathbb{S}}, \forall q_1 (q_1{:}A \to q_1 \times c_2{:}G) \models c_2{:}(A, G)$. (ci)
□

As noted in Sect. 1, and as will be seen in Sect. 5.1, contracts are of particular importance when proving compositionality. Therefore, results in Sect. 5.1 will be derived assuming that all specifications are contracts. However, this is no limitation according to the following proposition.

Proposition 8 (Generality of Contracts).
It holds that $T_{\mathbb{C},\mathbb{S}} \models S = (\top_{\|}, S)$. □

Thus, any specification S can always be written as the contract $(\top_{\|}, S)$. In fact, this is the reason why the specification constant symbol $\top_{\|}$ was introduced in the syntax in Sect. 2.

According to Proposition 3, assertional specifications are tightly connected with implementation being monotonic with respect to component composition.

This monotonicity is highly useful for proving compositionality, as will be seen in Sect. 5.1. Therefore, it is important to know if a contract is assertional, and according to the following proposition, a contract is in fact assertional if its guarantee is assertional.

Proposition 9 (Assertional Contracts). *It holds* $T_{\mathbb{C},\mathbb{S}}, \mathsf{Assertional}(G) \models \mathsf{Assertional}((A, G))$. □

4.8 Compatibility Specification ©

Two components may impose constraints incompatible with each other. This corresponds to that there is no single run that is in both behaviors of the components, i.e. $c_1^{\mathcal{M}} \cap c_2^{\mathcal{M}} = \emptyset$. We want to be able to reason about this case, in order to avoid it. Therefore we need support in the syntax to express and specify that components shall be compatible, and to express that components are or are not compatible. This syntactic support is provided by the specification constant symbol © whose interpretation, according to (2), is the set of all behavior sets except the empty set. Consequently, for any model $\mathcal{M} \in \mathbb{M}_{\mathbb{C},\mathbb{S}}$, it holds $\mathcal{M} \models c_1 \times c_2 : ©$ if and only if $c_1^{\mathcal{M}} \cap c_2^{\mathcal{M}} \neq \emptyset$.

Note that even though the purpose of the notion of compatible components is to avoid *two or more* components to get the behavior empty set. However, in the framework, the notion of compatibility is formally a property of *general* component terms, including *single* components. Therefore we will simply refer to compatible or incompatible *components*, irrespective of whether the component is a composition or not.

Note that an assertional specification will always accept an incompatible component, i.e. an incompatible component will always implement any assertional specification, since the interpretation of an assertional specification is a downward-closed set, which always includes the empty set as one of the elements. Therefore, the specification © comes in handy in conjunction with assertional specifications, i.e. $S \sqcap ©$ allows S to be assertional while at the same time enforces compatibility.

5 Compositionality

With grammar and behavior semantics defined, this section will present the second contribution of the paper: a methodology and a deductive system for proving compositionality without presuming specifications to be expressed formally. We start by defining compositionality formally.

Definition 1 (Compositionality). *Consider a model* $\mathcal{M}$ *such that* $\mathcal{M} \in \mathbb{M}_{\mathbb{C},\mathbb{S}}$. *In the model* $\mathcal{M}$*, the specifications* $S_1, \ldots S_N$ *are* composable *into the specification* S*, or equivalently, the specification* S *is* decomposable *into the specifications* $S_1, \ldots S_N$*, if*

$$\mathcal{M} \models S_1 \| \ldots \| S_N \sqsubseteq S \ . \tag{5}$$

□

Note that in Definition 1, compositionality is defined for a specific model $\mathcal{M}$. In fact, in the framework of the paper, we *can not* define compositionality to be a property only of the specifications $S_1, \ldots, S_N, S$ such as

$$T_{\mathbb{C},\mathbb{S}} \models S_1 \| \ldots \| S_N \sqsubseteq S$$

since this relation does *not* hold in general. It holds in some particular cases, such as $T_{\mathbb{C},\mathbb{S}} \models A \| (A, G) \sqsubseteq G$. However it does not hold if $S_1, \ldots, S_N, S$ are distinct constant terms, i.e. $T_{\mathbb{C},\mathbb{S}} \models \mathbf{S}_1 \| \ldots \| \mathbf{S}_N \sqsubseteq \mathbf{S}$ does *never* hold, since there is always possible to find one model that does not satisfy $\mathbf{S}_1 \| \ldots \| \mathbf{S}_N \sqsubseteq \mathbf{S}$.

The reference to a model in Definition 1 stands in contrast to [14], in which specifications themselves are expressed as formulas of predicates, meaning that in [14], no reference to a specific model $\mathcal{M}$ is needed to define or evaluate compositionality. This difference is further highlighted in the following example.

Example 6. Consider an engineering context represented by a model $\mathcal{M}_1$ in which the specification constant symbol $\mathbf{S}$ represents a real-world specification expressed as "The value of signal y shall be two times the value of signal u.". That is, the interpretation $\mathbf{S}^{\mathcal{M}_1}$ is the behavior set of all behaviors matching this natural-language specification. Furthermore, $\mathbf{S}_1$ represents a specification expressed as "The value of signal x shall equal the value of signal u.", and $\mathbf{S}_2$ represents a specification expressed as "The value of signal y shall be two times the value of signal x." In this model $\mathcal{M}_1$, it holds $\mathcal{M}_1 \models \mathbf{S}_1 \| \mathbf{S}_2 \sqsubseteq \mathbf{S}$. However, we can not say anything about any *other* model $\mathcal{M}_2$ since we do not know the interpretation of $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}$ in any such model. For example, in $\mathcal{M}_2$, it might be the case that $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}$ represents real-world specifications "x shall equal 0.", "y shall equal 0.", and "z shall equal 0" respectively. Thus, it does not hold generally that $\models \mathbf{S}_1 \| \mathbf{S}_2 \sqsubseteq \mathbf{S}$. □

5.1 Proving Compositionality

Even though compositionality is a property of a model, we want to utilize standard methods and tools, developed for first-order logic, and these work solely on a syntactic level. Thus, when proving compositionality, we need a way to avoid making reference to a particular model.

To solve this problem, we below present a theorem, which explains how compositionality can be proven without reference to any model $\mathcal{M}$. To remove the model $\mathcal{M}$ from the reasoning, we abstract it into a set Γ of sentences that $\mathcal{M}$ satisfy. More specifically, the sentences in Γ represent given knowledge about *refinement relations* between specifications, and also given knowledge that some specifications are *assertional*. The theorem refers to a hypothetical deductive system that is *strongly* sound for $T_{\mathbb{C},\mathbb{S}}$, which means that, for any sentence ϕ, if $T_{\mathbb{C},\mathbb{S}} \vdash \phi$, then also $T_{\mathbb{C},\mathbb{S}} \models \phi$.

Theorem 1. (Proving Contract Compositionality). *Let Γ be a set of sentences where each sentence is either in the form $\sqcap_j \alpha_j \sqsubseteq \beta$ or* $\mathsf{Assertional}(\gamma)$*, where each α_j, β, and γ is a specification term. If there is a proof of*

$$\Gamma,\ q_1{:}(A_1, G_1), \ldots,\ q_N{:}(A_N, G_N)\ \vdash\ q_1 \times \ldots \times q_N{:}(A, G), \tag{6}$$

in a deductive system, strongly sound for $T_{\mathbb{C},\mathbb{S}}$*, then it holds*

$$T_{\mathbb{C},\mathbb{S}}, \Gamma \models (A_1, G_1) \| \ldots \| (A_N, G_N) \sqsubseteq (A, G). \tag{7}$$

□

Note that (7) means, according to Definition 1, that in any model $\mathcal{M} \in \mathbb{M}_{\mathbb{CS}}$, satisfying Γ, the specifications $(A_1, G_1), \ldots (A_N, G_N)$ are composable into the specification (A, G).

To illustrate how the theorem can be used in an engineering context, assume that the specifications $A_1, G_1, \ldots, A_N, G_N, A, G$ are stored in a requirements management system. Assume also that the requirements management system contains the knowledge of refinement relations between the specifications and of which specifications that are assertional. This knowledge can be based on manual informal analysis made by engineers, which is the only option when specifications are informal, or it can be based upon formal analysis. However, it is important to note that any such formal analysis is done outside the proposed framework and outside the scope of Theorem 1. This allows using the most appropriate analysis technique for the problem at hand, e.g. non-linear analysis in case specifications are expressed as non-linear differential equations.

Whatever analysis methods that are used, manual or formal, the obtained knowledge is taken as input to the compositionality proof problem by inserting it as sentences in the set Γ. Then a theorem prover such as HOL4 can be used with a deductive system to find a proof of the sequent (6). That a proof has been found means formally that in any model $\mathcal{M} \in \mathbb{M}_{\mathbb{C},\mathbb{S}}$, such that $\mathcal{M}$ satisfies each sentence in Γ, the contract (A, G) is decomposable into the contracts (A_i, G_i). Note that the model $\mathcal{M}$ is the mathematical object representing the engineering context but it is not needed explicitly. Instead we can express the conclusion simply as: in the engineering context considered, the contract (A, G) is decomposable into the contracts (A_i, G_i).

5.2 Deductive System for Proving Compositionality

Theorem 1 presented the solution to prove compositionality (5) by finding a proof of the sequent (6). In order to find such proofs, we consider a deductive system for the language $\mathcal{L}_{\mathbb{C},\mathbb{S}}$, having no axioms but a set of inference rules R. We choose R to include seven rules based upon the propositions in Sect. 4: $R = \{\mathsf{re}, \mathsf{am}, \sqcap\mathsf{i}, \sqcap\mathsf{e}, \mathsf{ce}, \mathsf{ci}, \mathsf{cre}\}$. The first six match exactly the corresponding proposition and rule cre is a rule derived by combining $\sqcap$e, re, and $\sqcap$i. Two examples of the inference rules are Contract Elimination (ce), based upon Proposition 2a, and the derived rule Conjunction Refinement Elimination (cre):

$$\frac{c_1 : S_1 \quad c_2 : (S_1, S_2)}{c_1 \times c_2 : S_2}\ \mathsf{ce} \qquad \frac{c : S_1 \sqcap S_2 \quad S_2 \sqsubseteq S_3}{c : S_2 \sqcap S_3}\ \mathsf{cre}.$$

Each rule in R is not sound generally, but according to the underlying propositions 2, 3, 4, and 7, which have all been proven using HOL4, see [7], each rule is

sound when $T_{\mathbb{C},\mathbb{S}}$ is added to the premises. This means that each rule is strongly sound for $T_{\mathbb{C},\mathbb{S}}$, and consequently, the deductive system is strongly sound for $T_{\mathbb{C},\mathbb{S}}$.

In general, more inference rules can be needed to prove compositionality. For instance, the utilization of interface specifications is highly useful but this is out of the scope of the present paper. However, for the example presented in Sect. 5.3 below, the seven rules in R are sufficient.

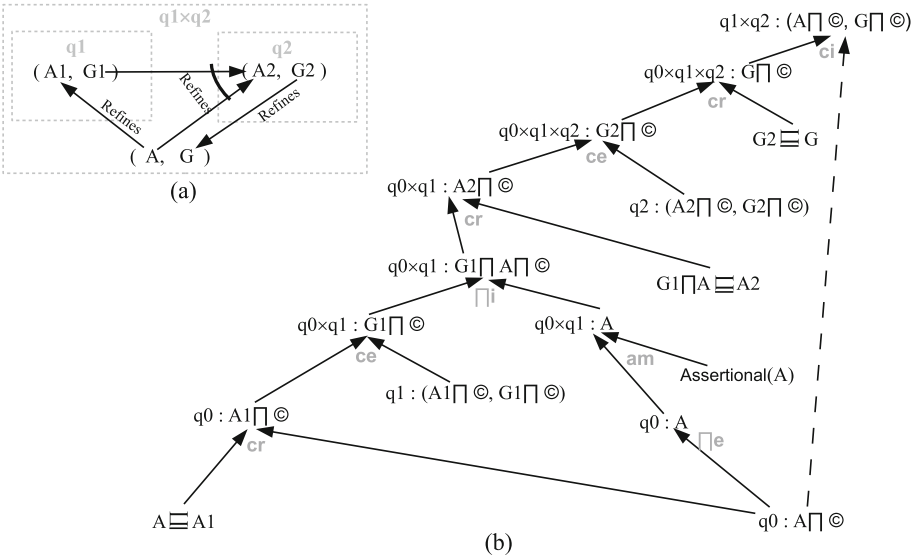

Fig. 1. (a) A compositionality problem with refinement relations in (8) illustrated as a directed graph. (b) A proof DAG for a proof of the sequent (9) corresponding to the compositionality problem in (a) but with © added to each contract.

5.3 Compositionality Proof Example

We consider an example of a basic compositionality problem as illustrated in Fig. 1a, where arrows represent known refinement relations, and component terms in the sequent to prove (9) are represented as dashed boxes. The refinement relations and a known fact that the specification A is assertional are collected in

$$\Gamma = \{\text{Assertional}(A), A \sqsubseteq A_1, A \sqcap G_1 \sqsubseteq A_2, G_2 \sqsubseteq G\}. \tag{8}$$

To ensure compatibility as explained in Sect. 4.8, we add the compatibility specification © to the assumption and guarantee of each contract. According to

Theorem 1, this gives us the following sequent to prove:

$$\Gamma,\ x_1{:}(A_1 \sqcap ©, G_1 \sqcap ©), x_2{:}(A_2 \sqcap ©, G_2 \sqcap ©) \ \vdash\ x_1 \times x_2{:}(A \sqcap ©, G \sqcap ©). \quad (9)$$

The deductive system has been implemented in the theorem prover HOL4, as further explained in [7], and by using this implementation, we automatically obtain a proof of the sequent (9). The proof is shown in Fig. 1 as a proof DAG in which arrows point from the premises used in the inference rule shown at the corresponding arrow head. The dashed arrow illustrates that $q_0 : A \sqcap ©$ is temporarily assumed when applying the rule contract introduction ci.

6 Conclusions

Based upon first-order predicate logic, the paper has presented a formal framework and methodology for proving compositionality. To support general heterogeneous systems, the semantics chosen is behavior based, originating in previous work on contract-based design for cyber-physical systems [1,2,15,16]. However, in contrast to the previous work, we treat implementation of specifications to be *non-monotonic* with respect to composition. That is, even though a specification is implemented by one component, a composition with a second component may not implement the same specification. This kind of non-monotonicity is fundamentally important to support architectural specifications and so called freedom of interference used in design of safety critical systems.

With the contributions of the paper, i.e. the framework and methodology for proving compositionality, it is now possible to prove compositionality for industrial systems, even though specifications themselves are not formal objects. Instead, we rely on given *refinement* relations between specifications and the property *assertional* of the specifications.

Our view is that in industrial heterogenuous systems, there is in general a mix between informal and formal specification. Furthermore, for the specifications that indeed are formal, the kind of formalism differ between different parts of the system, e.g. one part might be described by differential equations, while another by linear-time logic. Therefore, within each formalism, some individual refinement relations may indeed be possible to prove, although there is no universal formalism in which all refinement relations can be proven and certainly not the whole compositionality. In this case, the proposed framework provides a unifying framework allowing reasoning utilizing the *results from* individual refinement proofs, and also *results from* informal analyses, in order to prove compositionality.

References

1. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2007. LNCS, vol. 5382, pp. 200–225. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92188-2_9

2. Benveniste, A., Caillaud, B., Passerone, R.: Multi-viewpoint state machines for rich component models. In: Model-Based Design for Embedded Systems, pp. 487–518. Taylor & Francis (2009)
3. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**(3), 560–599 (1984)
4. Dill, D.L.: Trace theory for automatic hierarchical verification of speed-independent circuits. In: Proceedings of the fifth MIT Conference on Advanced Research in VLSI, pp. 51–65. MIT Press, Cambridge, MA, USA (1988)
5. Furia, C.A.: A Compositional World - a survey of recent works on compositionality in formal methods. Technical Report 22, Dipartimento di Elettronica e Informazione, Politecnico di Milano (2005)
6. Galton, A.: Logic for Information Technology. John Wiley & Sons Inc., Hoboken (1990)
7. Hedengren, G.: Verifying Correctness of Contract Decompositions. Master's thesis, Royal Institute of Technology (KTH) (2020)
8. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, Cambridge (2004)
9. ISO 26262: "Road vehicles - Functional safety" (2018)
10. Negulescu, R.: Process spaces. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 199–213. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44618-4_16
11. Nyberg, M., Gurov, D., Lidström, C., Rasmusson, A., Westman, J.: Formal verification in automotive industry: enablers and obstacles. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11247, pp. 139–158. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03427-6_14
12. Nyberg, M., Westman, J., Gurov, D.: Formally proving compositionality in industrial systems with informal specifications. Technical report, Royal Institute of Technology (KTH) (2020). http://www.kth.se/profile/matny
13. Peng, H., Tahar, S.: A survey on compositional verification. Technical report, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, November 1998
14. Roever, W.-P.: The need for compositional proof systems: a survey. In: de Roever, W.-P., Langmaack, H., Pnueli, A. (eds.) COMPOS 1997. LNCS, vol. 1536, pp. 1–22. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49213-5_1
15. Sangiovanni-Vincentelli, A.L., Damm, W., Passerone, R.: Taming Dr. Frankenstein: contract-based design for cyber-physical systems. Eur. J. Control **18**(3), 217–238 (2012)
16. Westman, J., Nyberg, M.: Conditions of contracts for separating responsibilities in heterogeneous systems. Formal Methods Syst. Des. **52**(2), 147–192 (2017). https://doi.org/10.1007/s10703-017-0294-7
17. Westman, J., Nyberg, M.: Preserving contract satisfiability under non-monotonic composition. In: Baier, C., Caires, L. (eds.) FORTE 2018. LNCS, vol. 10854, pp. 181–195. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92612-4_10
18. Wolf, E.S.: Hierarchical Models of Synchronous Circuits for Formal Verification and Substitution. Ph.D. thesis, Stanford University, Stanford, CA, USA (1996)