Contract-based Specification and Description-Logic-Based Validation of Product Lines

Damir Nešić and Mattias Nyberg
School ITM, department MMK, division MDA


August, 2018
Contract-based Specification and Description-Logic-Based Validation of Product Lines

Damir Nešić\textsuperscript{1} and Mattias Nyberg\textsuperscript{1,2}

\textsuperscript{1} Royal Institute of Technology, Stockholm, Sweden
\textsuperscript{2} Scania CV AB, Södertälje, Sweden
\{damirn,matny\}@kth.se

Abstract. The complexity of critical systems is constantly increasing. Consequently, assuring properties like safety or security of such systems is increasingly difficult. The difficulties are only intensified in the \textit{Product-Line Engineering} (PLE) context, where properties of a complete family of systems, i.e., a \textit{Product Line} (PL), must be assured. \textit{Contract-Based Specification and Design} (CBSD) paradigm is a promising approach for alleviating these difficulties because it is a general-purpose, formal paradigm, developed purposely to support structured development of complex systems which are \textit{correct-by-design}. Starting from a general CBSD framework, we present an extension that supports using CBSD in PLE, and prove that the extension preserves the properties of the original framework. Then, as a step towards providing tool-support for CBSD specification of PLs, we define the encoding of an arbitrary CBSD model of a PL, together with the constraints which define a proper CBSD model, as a \textit{Tbox} of a description logic knowledge base. Finally, we show how verification of these constraints can be reduced to satisfiability verification of the corresponding knowledge base. In order to validate the presented approach, a CBSD specification of a small, but real, industrial PL is created, implemented as an OWL ontology, and an off-the-shelf reasoner was used to verify if the provided CBSD model is proper.

1 Introduction

The complexity and heterogeneity of critical systems is constantly increasing [18]. As a consequence, assuring that such systems satisfy properties like safety or security, by either using testing or formal verification methods, is increasingly difficult. When such systems are developed according to the \textit{Product-Line Engineering} [3, 25] paradigm this task becomes even more challenging. PLE is a development methodology which facilitates the development of a family of systems, referred to as a \textit{Product Line} (PL), where individual systems, referred to as \textit{product configurations}, share \textit{common} functionalities, but potentially each product configuration can contain unique functionality, referred to as \textit{variable}. Because the number of possible product configurations grows exponentially with the number of functionalities, creating an individual of each product configuration and establishing its properties is often not feasible [19, 28, 36].
One of the few approaches for tackling the above challenges is the Contract-Based Specification and Design (CBSD) paradigm [9, 13, 33, 42]. CBSD is a general-purpose, formal framework for the development of complex systems which are correct-by-design. By using pairs of specifications, referred to as contracts, CBSD explicitly separates the assumptions that systems, and each of its constituent component, place on their respective environment, in order to guarantee the properties under their responsibility. This leads to the ability to perform compositional verification, where the properties of a system do not have to be verified on the system level but instead inferred from the verified properties of the constituent components. This is particularly important in the PLE context because it allows verifying properties of each product configuration by verifying the properties of each common and variable functionality in isolation.

Although CBSD has received significant attention in the past decades, none of the proposed approaches support variability management according to the principles of PLE. Therefore, as a first contribution, we extend the concepts of CBSD with mechanisms to manage the PL variability and prove that under certain constraints, this extension preserves the compositional reasoning about the properties of each product configuration in a PL. Then, as a step towards general-purpose tool-support for specification of PLs using CBSD, the second contribution of the present paper is the encoding of an arbitrary CBSD model and the previously mentioned constraints, as a Description Logic (DL) Knowledge Base (KB) [6]. Furthermore, verifying that the given CBSD specification of a PL conforms to the defined constraints is reduced to verifying the satisfiability of the corresponding KB. Both the PLE extension of CBSD and the DL encoding are validated on a small, but real, industrial PL from Scania CV AB, a global heavy-vehicle manufacturer.

The relevance of contract-oriented thinking in PLE has been noted almost two decades ago in approaches presenting methodological contributions regarding components-based, UML-driven, PL development [2, 5]. However, the most notable related work regarding contract-based design in the PLE community is based on work by Thüm [37–39]. This line of work integrates contract-based thinking according to Meyer [27], with Feature-Oriented Programming (FOP) [32] which is a particular method for developing software PL. The approach focuses on annotating the so called feature modules primarily with pre-conditions and post-conditions and then reasons about the result of the composition of a feature module $F$ with a feature module $F'$, which corresponds to various types of refinement of the contract of $F$ through the contract of $F'$. The work in [37–39] builds on the traditional design-by-contract paradigm [27], which focuses on "producing correct and robust software", while the work in the present paper builds on the CBSD paradigm which has its origins in design-by-contract paradigm but is more general, supports a top-down approach to systems engineering, and considers any type of physical or logical component composition. Consequently, the extension we present here supports the development of arbitrary, heterogeneous PLs, where contracts are first-class entities and not just annotations in the source code.
Existing literature on using DL and DL-based reasoners for analyzing contract-based models is scarce. Examples of somewhat related work are [26, 44] which treat the issue of automatically brokering contracts between web services. However the notion of a contract and its usage is different and less formal compared to CBSD frameworks. Besides the MICA tool [1], which is highly specialized and does not focus on supporting general-purpose CBSD, we are unaware of other work which aims at automated verification of CBSD models. On the other hand, the PLE community has recognized OWL as a useful tool for analyzing the so-called variability models which represent the commonality and variability of a PL. Work in [34] encodes a set of Simulink [15] objects which represent PL variations, as a DL KB and uses an off-the-shelf reasoners to analyze the possible product configurations for inconsistencies various defects [8]. Approaches in [16,30,40] provide the encoding of a particular type of a variability model, the so-called feature model (FM) [7], as a DL KB and also use off-the-shelf reasoners to detect inconsistencies similar product configurations defects, while the approach in [12] performs a similar task but instead of a FM the considered variability model is an OVM model [31].

The remainder of the paper is structured as follows. Section 2 summarizes the formal concepts underlying PLE and CBSD. Section 3 presents the PLE extension of the general CBSD paradigm. Section 4 defines the semantics of several PLE concepts in order to enable their encoding as a DL KB. Section 5 presents the encoding of an arbitrary FM and an arbitrary CBSD model as a DL KB and it is followed by Section 6 which exemplifies how an arbitrary CBSD model can be verified against the defined constraints. Finally, Section 7 discusses the results and concludes the paper.

2 Background

The central idea of PLE is to declare features [3] which represent user-visible functional and non-functional characteristics of each product configuration in the PL, and express them, together with any mutual dependencies, in a variability model where FMs are the most frequently used. This representation of a PL is a part of the so-called problem space [4]. Given an FM, design and implementation artifacts can be labeled with formulas expressed in terms of features and these formulas are referred to as presence conditions. The representation of a PL in terms of artifacts annotated with presence conditions is a part of the so-called solution space [4]. By selecting features from an FM, subject to declared dependencies between features, a particular product configuration is selected, and the set of artifacts that describe or implement the selected product configuration are those whose presence conditions evaluate to true for the given feature selection. The remainder of this section presents FMs, as the representation of the problem space, and CBSD models, as the representation the solution space. From here on, a feature is considered to be a Boolean variable.
2.1 Feature models and presence conditions

Definition 1 (Feature Model). A feature model $V$ is a pair $V = (F, C)$ where $F = \{f_1, \ldots, f_n\}$ is a set of features and $C$ is a set of Boolean constraints over the features in $F$.

For a specific set of Boolean constraints, FMs define the corresponding graphical syntax which is used to structure the features from $F$ into a tree, where each parent-$f_i$ to child-$f_m$ relation corresponds to the propositional expression $f_m \rightarrow f_i$. Figure 1a shows the FM graphical syntax used in the present paper, together with corresponding propositional expressions, in accordance with [14]. Figure 1b, shows the FM running example, created using the FeatureIDE tool [24], which shows the FM representation of an excerpt of the variability model describing the problem space in Scania CV AB, a global heavy-vehicle manufacturer. The constructs in Figure 1a are referred to as: i) $f_m$ is a mandatory feature, ii) $f_m$ and $f_n$ are a group of alternative features where $\bigvee$ stands for exclusive or, iii) $f_m$ and $f_n$ are a group of or features, and iv) $f_m$ is an optional feature. Note that Figure 1b contains additional Boolean constraints, where the constraints like $V8 \rightarrow Advanced$ are referred to as the requires constraints.

![FM graphical syntax and running example](image)

Fig. 1: Used FM constructs and the FM running-example.

Definition 2 (Product Configuration). Given a feature model $V = (F, C)$, a product configuration, denoted as $PC$, is a set of features such that each $f_i \in F$ is assigned with a value $v_i$, where $v_i \in \{true, false\}$. A product configuration for which each Boolean constraint in $C$ evaluates to true is valid.

The intuition behind a product configuration is that it represents, in terms of features, all individual systems that are identically configured. From here on, we consider only product configuration that are valid. As discussed previously, the relation between the problem space and the solution space is established through the concept of presence conditions.

Definition 3 (Presence Condition). A presence condition $\varphi$ is a boolean expression over features $f_i \in F$, structured according to the grammar $\varphi ::= f_i \neg(\varphi)(\varphi \land \varphi)(\varphi \lor \varphi)$. 

Verifying Contract-Based Specifications of Product Lines using DL
2.2 Basic CBSD framework

In order to support the development of increasingly complex and heterogeneous systems, several lines of research have presented CBSD-based approaches [9, 10, 13, 33, 42, 43]. This section summarizes the CBSD concepts relevant for the present paper and introduces the needed notation.

Basic concepts of CBSD are:

i) **Component** $C$, representing any logical or physical component,

ii) **Specification** $S$, representing any requirement,

iii) component implements a specification, denoted as $C \triangleright S$,

iv) specification $S_i$ fulfills specification $S_j$, denoted as $\text{fulfills}(S_i, S_j)$,

v) n-ary component composition that results in new components, denoted as $C' = C_1 \otimes C_2 \ldots \otimes C_n$.

A **Specification** specifies the intended behavior of a **Component**. It should be noted that what we refer here to as "specification" is in other approaches referred to as an "assertion". The notion of implementation is similar across all CBSD-based approaches, and it corresponds to the expectation that the implementation of a component will exhibit the behavior specified in the corresponding specification. The notion of fulfillment between specifications $S_i$ and $S_j$ represents the intention that if a component implements the property expressed as specification $S_i$ then it will also implement the property expressed as specification $S_j$. In [9], term refinement is used to describe the same concept. Finally, composition correspond to the process of component integration into new components and ultimately into a complete system.

Given the aforementioned concepts, the following definitions are introduced.

**Definition 4 (Contract).** A **Contract** $K$ is an ordered pair of specifications, denoted $(A, G)$, where $A$ is called an **Assumption** and $G$ is called a **Guarantee**.

Note that some CBSD-based approaches consider contracts also to be specifications. Here, we explicitly distinguish these two concepts because each requirement is a specification but often, a single requirement does not correspond to a assume-guarantee relation specified by a contract.

**Definition 5 (Satisfy Contract).** Component $C$ satisfies a contract $(A, G)$, denoted as $C \triangleright (A, G)$, if $\forall C_e. C_e \triangleright A \Rightarrow C_e \otimes C \triangleright G$.

Definition 5 shows the main benefit of using contracts. Given the contract $K = (A, G)$, the component $C$ can be developed independently of any other component and whenever it is composed with some component $C_e$ which implements $A$, often referred as the environment of $C$, the contract will be satisfied. We will consider that each contract is intended to be satisfied by one and only one component. This intention is expressed by saying that a contract is allocated to a component, denoted as $\text{allocatedTo}(K, C)$.

The intended semantics of the statement $\text{fulfills}(S_i, S_j)$ is that the property expressed in $S_i$ logically entails the property expressed in $S_j$, denoted as $S_i \models S_j$. Given that specifications are implemented by components, the fulfills relation between specifications $S_i$ and $S_j$ expresses the intention that $\forall C. C \triangleright S_i \Rightarrow C \triangleright S_j$. 


Assumption 1. The composition operator is commutative and associative.

The composition operator must be commutative and associative in order to enable that independently developed components can be composed in any order. To denote composition of multiple components, we introduce the shorthand $C' = \bigotimes_{i=1}^{N} C_i$ that replaces $C' = C_1 \otimes \ldots \otimes C_n$ notation. Furthermore, we refer to components that are not composed of other components as Atomic components and we refer to components composed of other components as Composite components. Atomic components usually represent low-level implementation like SW, HW, or mechanical components, but depending on the considered level of abstraction, sometimes a SW module can be an atomic component while in other cases a single SW function can be an atomic component. On the other hand, composite components can represent everything from a high level system functionality, for example a feature in PLE, down to a composition of two SW functions. Without loss of generality, in the remainder of the paper we consider only systems that correspond to a single composite component composed from an arbitrary number of atomic components.

Definition 6 (Monotonic composition). Composition of two components is monotonic with respect to implementation if, $\forall C, C', \forall S.C \triangleright S \Rightarrow C' \otimes C \triangleright S$.

Frequently, it is desirable that component composition is monotonic [43]. However, it should be noted that in the general case, component composition is not monotonic but such scenario is not considered in the remainder of the paper.

Assumption 2. The composition operator is monotonic with respect to implementation.

2.3 Illustrative example

Given the concepts introduced so far, Figure 2 presents an illustrative example, a simple but real industrial system, the Fuel Level Display (FLD) [41] from Scania CV AB, a global heavy trucks manufacturer.

Figure 2 shows a composite component FLD, which is composed of atomic components FuelSensor, COO, short for COOrdinator electronic control unit, and ICL, short for Instrumentation CLuster. The intended functionality of the FLD system is to display the current fuel level on the display of the ICL and guarantee that the displayed value corresponds to the actual fuel level. White rectangles represent contracts and the contract overlaying component FLD captures the above-mentioned specification as guarantee $G'$ under the assumption $A'$ that the ignition key is turned on. Note that contract $(A', G')$ is classified as safety related because displaying higher fuel level than the actual can lead to running out of fuel that in turn leads to sudden engine stop and loss of servo-steering which is essential for heavy vehicles.

Contract $(A_1, G_1)$ guarantees that the measured fuel level in volts will correspond to the actual fuel level if the power supply is nominal. Contract $(A_2, G_2)$ guarantees that the estimated value in liters will be in a predetermined $[0 - max]$ range and will correspond to the value measured by the sensor. Contract
Fig. 2: Running example; the FLD system

(A_3, G_3) guarantees that the displayed value will correspond to the actual value if the value in liters, received via the communication buss, is updated at predetermined time periods and corresponds to the actual value. Consequently, if \( G_3 \) is satisfied, \( G' \) is satisfied, i.e. \( G_3 \) fulfills \( G' \). Each fulfills relation is as indicated by an arrow labeled \( f \). Component \( C'' \), guarantee \( G'' \), and the fulfill relation from \( G'' \) to \( A' \) illustrate the expectation that once the FLD system is integrated into the vehicle, some other component must implement \( A'' \).

2.4 Specification structure

The only element of Figure 2 that has not been introduced formally so far are the arrows labeled \( a \) which represent the assumption of relation, denoted as assumptionOf\((A_i, G_i)\), which are intrinsic in each contract \( K = (A_i, G_i) \). Given an explicit representation of the assumption of relation, assumptions and guarantees connected with fulfills and assumptionOf relations form a graph that will be referred to as Specification Structure. We will also say that a set of contracts forms a specification structure.

**Definition 7 (Specification structure).** Let \( C \) be a possibly empty set of atomic components, \( K \) be a possibly empty set of contracts such that each contract \( K_i \in K \) is allocated to a single component in \( C \) where each \( A_i \) and \( G_i \) is unique. Furthermore, let \( C'' \) be a composite component such that \( C'' = \bigotimes_i C_i \), and \( (A'', G'') \) be a contract allocated to \( C'' \). Then, an edge-labeled directed graph \( D = (\mathcal{N}, \mathcal{E}) \) is a specification structure, if

1. each node \( n \in \mathcal{N} \) is either an assumption \( A_i \) or \( A' \) or a guarantee \( G_i \) or \( G' \),
2. each edge \( e = (n_i, n_j) \in \mathcal{E} \) corresponds to either a single assumptionOf\((S_i, S_j)\) or a single fulfills\((S_i, S_j)\) relation,
3. for each edge \( e = (n_i, n_j) \in \mathcal{E} \) it holds that \( n_i \neq n_j \),
4. for each assumptionOf\((S_i, S_j)\) relation, \( S_i \) is an assumption, \( S_j \) is a guarantee and the ordered pair \( (S_i, S_j) \) is a contract,
5. for each edge fulfills\((S_i, S_j)\), if
a) $S_i$ is a guarantee then $S_i$ is a guarantee of a contract $K_i$ and $S_j$ is either an assumption $A_j$ or the guarantee $G'$.

b) $S_i$ is an assumption then $S_i$ is the assumption $A'$ and $S_j$ is any $A_i$.

Definition 7 corresponds to the so-called contract structure in [42] and it provides an intuitive way to visualize how the property in the form of a guarantee $G'$ allocated to the system, is partitioned across the atomic components that comprise the system. Moreover, the specification structure matches well with the ideas in safety standards like ISO 26262 [23] or IEC 61508 [22] which require the decomposition of higher level requirements into lower level requirements with explicitly managed traceability links between them. In order to ensure that satisfying the contract allocated to the system is a consequence of satisfying the contracts allocated to the atomic components, the following conditions must hold.

Definition 8 (Proper specification structure). A specification structure $\mathcal{D}$ is proper if

i) for each assumption $A_i$ of a contract $K_i$, there exists a specification $S_k$ such that $\text{fulfills}(S_k, A_i)$, where $S_k$ is either $A'$ or $G_i$,

ii) there exists a guarantee $G_i$ such that $\text{fulfills}(G_i, G')$,

iii) the graph of $\mathcal{D}$ is acyclic.

Two consequences of previous definitions, which will be used later, are the following.

Proposition 1. If a specification structure $\mathcal{D}$ is proper, then there exists an edge $\text{fulfills}(A', A_i)$ where $A_i$ is an assumption of a contract $(A_i, G_i)$ allocated to an atomic component.

Proof. Assume that each assumption $A_i$ is such that edge $\text{fulfills}(A', A_i)$ does not exist. Given that $\mathcal{D}$ is proper, according to Definition 8, there exists a guarantee $G_j$ of a contract $(A_j, G_j)$ such that there exists an edge $\text{fulfills}(G_j, G')$. Furthermore, according to Definition 8, there exists a guarantee $G_k$ of a contract $(A_k, G_k)$ such that $G_k \neq G_j$ and $G_k \neq G'$. In other words, $G_k$ cannot belong to the path from $A_j$ to $G'$. If this process is repeated for each $A_i$, an assumption $A_m$ will be reached such that an edge $\text{fulfills}(G_n, A_m)$ would introduce a cycle in $\mathcal{D}$ because $G_n$ belongs to the path from $A_m$ to $G'$. Because this contradicts condition (iii) from Definition 8, it follows that there must exist an edge $\text{fulfills}(A', A_i)$.

Proposition 2. If a specification structure $\mathcal{D}$ is proper then each edge $\text{fulfills}(S_i, S_j)$ is such that $S_j \neq A'$.

Proof. From Proposition 1 it follows that there exists a path from $A'$ to $A_i$ and $G_i$ of each contract $(A_i, G_i)$ allocated to an atomic component $C_i$. Furthermore, $\mathcal{D}$ contains the edge assumptionOf$(A', G')$. Consequently, if there would exist an edge from any $A_i$, $G_i$, or $G'$ to $A'$, that would introduce a cycle in $\mathcal{D}$. Hence, there is no edge $\text{fulfills}(S_i, S_j)$ where $S_j = A'$. 
The following theorem shows the key idea of CBSD, i.e. the compositional reasoning about system properties. It corresponds to the dominance relation defined in [17].

**Theorem 1.** Given a set of atomic components \( C = \{C_i\}_{i=1}^{N} \), a set of contracts \( K = \{(A_i, G_i)\}_{i=1}^{N} \) allocated to components from \( C \), a contract \( K' = (A', G') \) allocated to a composite component \( C' = \bigotimes_{i=1}^{N} C_i \), if

\[
\begin{align*}
&i) \forall K_i, \exists C_i. C_i \triangleright K_i, \\
&ii) \forall \text{fulfills}(S_i, S_j). S_i \models S_j, \\
&iii) \text{contracts in } K \text{ together with } K' \text{ form a proper specification structure},
\end{align*}
\]

then it holds that: \( C' \triangleright K' \).

**Proof.** According to Definition 5, we can rewrite the theorem claim as \( \forall C''. C'' \triangleright A' \Rightarrow C'' \otimes C' \triangleright G' \). Let \( C'' \) be an arbitrary component such that \( C'' \triangleright A' \). From premise (iii), according to Proposition 1, it follows that there exist an assumption \( A_j \) and an edge fulfills\((A', A_j)\). From premise (ii) and the semantics of fulfill relation it follows that \( C'' \triangleright A_j \). From this and from premise (i), according to Definition 5, \( C'' \otimes C_j \triangleright G_j \). Without loss of generality, let \( A_j \) be such that there exists a path from \( A_j \) to some \( G' \). From this, and premise (iii), it follows that there exists an edge assumptionOf\((A_j, G_j)\) and furthermore an edge fulfills\((G_j, A_k)\) where \( A_k \) is the assumption of an some contract \((A_k, G_k)\). By following the path from \( A_j \) to \( G' \) and repeating previous steps, we obtain the expression \((C'' \otimes C_j) \otimes ... \otimes C_n \triangleright G_n \) where \( G_n \) is the guarantee of some contract \((A_n, G_n)\). This expression is either a composition of each \( C_i \in C \) and \( C'' \) or a subset of components \( C_s \subseteq C \) and \( C'' \). In the former case, the expression corresponds to \( C'' \otimes C' \triangleright G_n \). In the latter case, because according to Assumption 2 the composition operator is monotonic with respect to implementation, the expression can be extended with components in \( C \setminus C_s \) which also yields \( C'' \otimes C_1 \otimes C_2 \otimes ... \otimes C_n \triangleright G_n \Leftrightarrow C'' \otimes C' \triangleright G_n \). Given that \( G_n \) is an arbitrary guarantee and in accordance with condition (ii) from Definition 8, assume that \( G_n \) fulfills \( G' \). From premise (ii) and the semantics of fulfill relation it follows that \( C'' \otimes C' \triangleright G' \Leftrightarrow C' \triangleright K' \) which concludes the proof.

\( \square \)

The theory that has been presented cannot be used for specification of PLs. In the next section, the presented theory is extended with PLE constructs.

### 3 CBSD extension for PLE

In order to enable the use of CBSD concepts in the PLE context, and consequently enable construction of assurance cases, this section presents the first contribution of the present paper which is a PLE oriented extension of the theory presented in Section 2.2. Before we introduce the formal definitions of necessary extensions, we illustrate a representative PLE scenario and the challenges it introduces.
Consider that CBSD is used for the development of a PL, and that components and specifications are artifacts in the development lifecycle. According to Section 2, these artifacts can be labeled with arbitrary presence conditions, according to Definition 3. Figure 3 presents an extension of Figure 2 in which an additional atomic component BMS, which stands for Body Management System electronic control unit has been added, and each component and each specification is labeled with a presence condition denoted as $\phi_i$ written in terms of features from the FM in Figure 1b.

The component BMS is responsible for calculating the fuel level in liters in truck configurations with gas fuel while the component COO is responsible for the same calculation in truck configurations with diesel fuel. Consequently, the composite component FLD can be configured in one of the two valid ways; one in which component BMS is present but COO is not and $G_2$ fulfills $A_4$, and one in which component COO is present but BMS is not and $G_3$ fulfills $A_4$. In other words, the model in Figure 3 specifies the solution space of a PL with two product configurations.

Given this scenario, several mismatches between the presence conditions, and consequently the corresponding artifacts can occur. For example, in order for guarantee $G_2$ to fulfill assumption $A_4$, and because component BMS is intended to satisfy contract $(A_2, G_2)$, presence conditions $\phi_4, \phi_5, \phi_6$ and $\phi_{11}$ must simultaneously evaluate to true or false for each product configuration. If this is not
the case, that would mean that either: i) the guarantee $G_2$ is intended to fulfill assumption $A_2$ but the component $BMS$ is not expected to satisfy the contract $(A_2, G_2)$ for the same product configuration, ii) there exists a product configuration such that the presence conditions of $A_2$ and $G_2$ do not simultaneously evaluate to true. In general, presence conditions of assumptions, guarantees, and components must be such that mismatches like the ones described are avoided. Otherwise, the compositional reasoning about the property of the FLD system, according to Theorem 1, will not be possible.

In the following two subsections, we present the constraints to which presence conditions must conform, in order for the above mentioned mismatches.

3.1 Presence conditions in CBSD framework

Assume that a PL has $n$ product configurations $P_{C_1}, ..., P_{C_n}$. Consequently, for each product configuration a specification structure $D_1, ..., D_n$ can be constructed. In order to avoid maintaining $n$ different specifications structures, individual specification structures $D_i$ can be merged into a specification structure $\hat{D}$ such that if there are two identical nodes or two identical edges in two distinct individual specification structures $D_i, D_j, j \neq i$, then there exists a single corresponding node or a single edge, respectively, in the specification structure $\hat{D}$. The remaining nodes and edges in $\hat{D}$ correspond to nodes and edges specific to each $D_i$. The consequence of merging specification structures is that the sets of components $C_1, ..., C_n$ to which the contracts forming $D_1, ..., D_n$ are allocated, are also merged, i.e. $\mathcal{C} = \bigcup_{i=1}^{N} C_i$, and the composite component $C'$ becomes a composition of components from $\mathcal{C}$.

The resulting specification structure $\hat{D}$ then represents the specification structure of the complete PL, and hereinafter we refer to it as PL specification structure. Note that even if the original specification structures $D_i$ were proper, $\hat{D}$ is not necessarily proper because the merging operation could introduce cycles in the graph of $\hat{D}$.

Assumption 3. A PL specification structure $\hat{D}$ is acyclic.

Assumption 3 is introduced because showing acyclicity is rather involved and space-consuming but conceptually not central for the presented theory. Consequently, explicitly establishing acyclicity of $\hat{D}$ is left as future work.

Given a PL specification structure $\hat{D}$, in order to be able to identify the subgraph of $\hat{D}$ that applies to a particular product configuration, it is necessary to relate the artifacts forming specification structure $\hat{D}$ and product configurations $P_{C_1}, ..., P_{C_n}$. More specifically, given a specification structure $\hat{D}$ formed by the set of contracts $K = \{(A_i, G_i)\}_i$, allocated to components $C_i$, let $\Phi$ be a labeling function that labels each $A_i$, $G_i$, and $C_i$, with a presence conditions $\varphi$. The notation $\Phi(S)$ will be used to denote the presence condition of specification $S$ while the notation $\Phi_{P_{C_i}}(S)$ will be used to denote the true or false value of the presence condition of $S$ for product configuration $P_{C_i}$. From here on, we consider that each assumption $A_i$, guarantee $G_i$, and component $C_i$ is labeled with a presence conditions that evaluates to true or false for any given product configuration $P_{C_i}$.
3.2 Constraints on presence conditions

As outlined in section 2, the idea behind labeling artifacts with presence conditions is to be able to automatically obtain the set of artifacts that describe or implement a product configuration $P_C$ by evaluating each presence condition and selecting the ones whose presence conditions evaluate to true for the given $P_C$.

We say that a PL specification structure $\hat{D}$ is instantiated for a product configuration $P_C$, if each $A_i$, $G_i$, and $C_i$ whose presence condition evaluates to false for $P_C$, is removed from $\hat{D}$. The result of instantiation, denoted as $D_{P_C}$, is an edge-labeled, directed graph which is not necessarily a specification structure according to Definition 7. For example, presence conditions of $A_i$ and $G_i$ of a contract $(A_i, G_i)$ could evaluate to true but the presence condition of the component $C_i$ to which the contract $(A_i, G_i)$ is allocated, evaluates to false.

In order to ensure that instantiation results in a specification structure we introduce the following definitions, and we refer to them as invariance with respect to configurations, hereinafter only invariant. The entailment between presence conditions $\varphi_i$ and $\varphi_j$ is denoted as $\varphi_i \models^C \varphi_j$ as a short hand for $(\mathcal{C}, \varphi_i) \models \varphi_j$, i.e. entailment must hold under the constraints in $\mathcal{C}$. For instance, in Figure 3, it holds that $\varphi_2 \not\models^C \varphi_1$, but when constraints $\text{Truck} \rightarrow \text{Fuel}$ and $\text{Fuel} \rightarrow \text{Gas} \lor \text{Diesel}$ from Figure 1b are considered, then the entailment holds, i.e. $\varphi_2 \models^C \varphi_1$.

Definition 9 (Invariant Contract). A contract $K = (A, G)$ is invariant if it holds that $\Phi(G) \models^C \Phi(A)$ and $\Phi(A) \models^C \Phi(G)$.

Definition 10 (Invariant Allocation). An allocation of an invariant contract $(A, G)$ to a component $C$ is invariant if $\Phi(G) \models^C \Phi(C)$ and equally if $\Phi(A) \models^C \Phi(C)$.

Definition 11 (Invariant Composition). Composition of atomic components $C_1, \ldots, C_n$ into a composite component $C' = \bigotimes_i C_i$ is invariant if it holds that $\forall C_i, \Phi(C_i) \models^C \Phi(C')$.

Given a PL specification structure $\hat{D}$ with invariant contracts, allocation, and composition, we show that instantiating $\hat{D}$ for an arbitrary valid product configuration $P_C$, results in a specification structure according to Definition 7.

Theorem 2. Given a feature model $V$, and a PL specification structure $\hat{D}$, if

i) each contract is invariant,

ii) each allocation of a contract $(A_i, G_i)$ is invariant,

iii) component composition is invariant,

then for each valid product configuration $P_{C_i}$, each instantiation $\mathcal{D}_{P_{C_i}}$ is a specification structure.

Proof. Let $P_{C_i}$ be an arbitrary valid product configuration specified by $V$. Given $\hat{D}$, according to Definition 7, it follows that there exists a contract $(A'_i, G'_i)$ allocated to a composite component $C' = \bigotimes_i C_i$. If the presence condition of
each $A_i$, $G_j$, evaluate to false for the given $P_{C_k}$, then this corresponds to the case when the set of contracts $\mathcal{K}$, allocated to atomic components, is empty and the graph only contains nodes $A'$ and $G'$ and the edge assumptionOf($A'$, $G'$). Otherwise, the presence condition of at least one assumption $A_i$ or a guarantee $G_j$ evaluates to true. For each $\Phi_{P_{C_k}}(A_i) = true$, and according to premise (i), the corresponding $\Phi_{P_{C_k}}(G_i)$ of the contract $(A_i, G_i)$, must simultaneously evaluate to true. Then it follows that there exists an edge-labeled, directed graph with nodes $A_i, G_i$ and edges assumptionOf($A_i$, $G_i$). According to premise (i), identical consideration holds for each $G_j$ such that $\Phi_{P_{C_k}}(G_j) = true$.

So far we have established that given premise (i), $\mathcal{D}_{P_{C_k}}$ forms an edge-labeled directed graph according to Definition 7. However Definition 7 also requires that the contracts that form $\mathcal{D}_{P_{C_k}}$ are allocated to a single atomic component $C_i$ or the composite component $C'$ and that $C'$ is such that $C' = \bigotimes_i C_i$. Given premise (ii), for each $\Phi_{P_{C_k}}(A_i) = true$ or $\Phi_{P_{C_k}}(G_i) = true$ of a contract $(A_i, G_i)$ allocated to a component $C_i$, it follows that $\Phi_{P_{C_k}}(C_i) = true$. From this, and given premise (iii), for each $C_i$ such that $\Phi_{P_{C_k}}(C_i) = true$ it follows that $\Phi_{P_{C_k}}(C') = true$, i.e. $C' = \bigotimes_{j=1}^M C_j$ where $M$ is an arbitrary number of atomic components whose presence conditions evaluate to true for the given $P_{C_k}$. Finally, condition (iii) from Definition 7 holds because $\mathcal{D}$ is constructed from specification structures that are well-formed with respect to Definition 7. This concludes the proof.

Theorem 2 established the conditions under which instantiating the PL specification structure for any product configuration will result in a specification structure. However, as shown by Theorem 1, in order to prove that a property, in the form of a guarantee of a contract, allocated to the system is satisfied, instantiating a PL specification structure should result in a proper specification structure.

**Theorem 3.** Given a feature model $\mathcal{V}$, and a PL specification structure $\hat{\mathcal{D}}$ such that each instantiation $\mathcal{D}_{P_{C_k}}$ is a specification structure according to Definition 7, if

\begin{itemize}
  \item[i)] $\Phi(A_i) \models \bigvee_{\text{fulfills}(S_k, A_i)}\Phi(S_k)$ where $S_k$ is some $G_j$ or $A'$,
  \item[ii)] $\Phi(G') \models \bigvee_{\text{fulfills}(G_i, G')} \Phi(G_i)$,
\end{itemize}

then each $\mathcal{D}_{P_{C_k}}$ is proper.

**Proof.** The proof consists of showing that conditions from Definition 8 hold for each instantiation of $\hat{\mathcal{D}}$. Let $P_{C_k}$ be an arbitrary product configuration specified by $\mathcal{V}$ and let $\mathcal{D}_{P_{C_k}}$ be an instantiation of $\hat{\mathcal{D}}$ for product configuration $P_{C_k}$. From premise (i) it follows that there exists a specification $S_k$ and an edge fulfills to each $A_i$ in $\mathcal{D}_{P_{C_k}}$. Because $\mathcal{D}_{P_{C_k}}$ is obtained from $\mathcal{D}$, it follows that for each fulfills edge to each $A_i$ there exists a $G_j$ or $A'$ from which each such fulfills edge originates from. This shows that condition (i) from Definition 8 holds for $\mathcal{D}_{P_{C_k}}$.

The fact that condition (ii) from Definition 8 holds for $\mathcal{D}_{P_{C_k}}$ follows directly from premise (ii). Finally, because $\mathcal{D}$ does not contain cycles according to Assumption 3, then $\mathcal{D}_{P_{C_k}}$ does not contain cycles and consequently condition (iii)
of Definition 8 holds. Given that $P_{C_k}$ was selected arbitrarily, the above proof holds for each instantiation of $\mathcal{D}$.

A direct consequence of Theorem 2 and Theorem 3 in conjunction with Theorem 1 is the following corollary.

**Corollary 1.** Given a feature model $\mathcal{V}$, and a PL specification structure $\mathcal{D}$, if

1. each contract is invariant,
2. each allocation of a contract $(A_i, G_i)$ is invariant,
3. component composition is invariant,
4. $\Phi(A_i) \models \bigvee_{\text{fulfills}(S_k, A_i)} \Phi(S_k)$ where $S_k$ is some $G_j$ or $A'$,
5. $\Phi(G') \models \bigvee_{\text{fulfills}(G_i, G')} \Phi(G_i)$,
6. $\forall_{\text{fulfills}(S_i, S_j)} S_i \models S_j$,
7. each contract $K_i$ allocated to an atomic component $C_i$ is such that $C_i \triangleright K_i$,

then it holds that $C' \triangleright (A', G')$ for each valid product configuration $P_{C'}$.

Less formally, if a PL specification structure is such that each instantiation is a proper specification structure, and if each pair of specifications in fulfills relation logically entail each other, then by verifying that each component of each product configuration satisfies the contract allocated to it, it can be inferred that each product configuration satisfies the contract allocated to it. In other words, if a PL is specified using CBSD and if each constraint from Corollary 1 holds, then verifying that each product configuration satisfies the properties allocated to it can be reduced to verifying the properties of its constituent component.

The remainder of the paper aims at providing automated support for verifying that each instantiation of an arbitrary CBSD model, is a proper specification structure. In other words, given an arbitrary CBSD model, a way to verify constraints in Definition 7, Definition 8, and constraints (i)-(v) from Corollary 1 is defined. It should be noted that due to Assumption 3, it is assumed that constraint (iii) from Definition 8 is verified elsewhere.

### 4 Semantics of presence conditions

This and the following sections present the second contribution of the present paper. In order to be able to verify that an arbitrary CBSD model is a proper specifications structure, it is necessary to define a common semantic interpretation of the constructs representing the problem space and the solution space.

Because an FM and a CBSD model jointly represent the PL design, a potentially infinite number of real-world individuals, denoted as $p$, can be created by instantiating the PL design. From this it follows that each product configuration corresponds to a set of individual products. Furthermore, each $p$ can be represented as a Component, either atomic or composite. For example, each individual Scania product, which is a truck or a bus, is a Composite component composedOf an individual of the FLD system, which is also a Composite component.
In order to define the semantics of presence conditions, let $\text{Config}$ be a function which given an individual product $p$ returns its product configuration $P_C$. Moreover, let $\text{Eval}$ be a function that takes as arguments a presence condition $\varphi$ and a product configuration $P_C$ and returns the value of $\varphi$ for the given $P_C$.  

**Definition 12 (Presence condition semantics).** The semantics of a presence condition, denoted as $[\varphi]$, is $[\varphi] = \{p_i | \text{Eval}(\varphi, \text{Config}(p_i)) = \text{true}\}$.

A presence condition $\varphi$ corresponds to a set of individual products to which the artifact labeled with $\varphi$ applies. This interpretation of presence conditions corresponds to the concept of *product groups* in Nešić [29].

### 5 Encoding an FM and a CBSD model as Tbox axioms

This section presents the encoding of an arbitrary FM and an arbitrary CBSD model as a set of Tbox axioms. In order to provide a formal encoding, a CBSD model is represented as a tuple $M = (O, R, \{\cdot\}^T)$, where $O$ is a set of objects, $R$ is a set of pairs representing relations, and $\{\cdot\}^T$ is a function which returns the type of an object or a relation. For instance, a fragment of the example from Figure 3 would be $\{G_1, A_2\} \subseteq O$, $(G_1, A_2) \in R$, and $G_1^T = \text{Specification}$, $A_1^T = \text{Specification}$, $(G_1, A_1)^T = \text{fulfills}$.

Regarding DL notation, let $N_C, N_R$ and $N_I$ be mutually disjoint sets of concept names, role names, and individual names, respectively. Then the tuple $(N_C, N_R, N_I)$ is referred to as the KB signature. In the remainder of the paper we use the DL notation and the semantics defined in [6].

#### 5.1 General CBSD entities and relations as Tbox axioms

Before defining the encoding of an arbitrary FM or an arbitrary $M$ as Tbox axioms, the Tbox is complemented with concepts and roles that correspond to general CBSD entities and relations. Table 1 and Table 2 show the corresponding concepts and roles.

Roles $R_{\text{hasS}}, R_{\text{hasA}}, R_{\text{hasG}}$ correspond to the *containment* relation between a specification structure and specifications, denoted $\text{hasSpecification}(D, S)$, and the containment relation between contracts and corresponding assumptions and guarantees, denoted $\text{hasAssumption}(K, A)$ and $\text{hasGuarantee}(K, G)$. Role $R_{\text{comp}}$, short for *comprises*, represents the relation between each atomic component $C_i$ and the composite component $C'$ such that $C' = \bigoplus C_i$, i.e. $\text{comprises}(C_i, C')$. Roles $R_{\text{partOf}}, R_{\text{gPartOf}}, R_{\text{fullBy}}$ are inverse to roles $R_{\text{hasA}}, R_{\text{hasG}}, R_{\text{full}}$, respectively. As will be shown later, expressing several conditions from Definition 7 and Definition 8 will require these roles.

According to the discussion in Section 4, the Tbox is complemented with the concept $D_P \in N_C$ which represents all individual products, and the axiom $D_P \subseteq D_C$ in order to capture the fact that each individual product is a component. Moreover, because specifications are partitioned into assumptions and guarantees, the axiom $D_S \equiv D_{SA} \cup D_{SG}$ is added to the Tbox. Similarly, because
Table 1: DL concepts corresponding to entities in the CBSD framework

<table>
<thead>
<tr>
<th>CBSD entity</th>
<th>concept</th>
<th>Inclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spec. Struct.</td>
<td>$D_{SS}$</td>
<td>$\subseteq$</td>
</tr>
<tr>
<td>Specification</td>
<td>$D_S$</td>
<td>$\subseteq$</td>
</tr>
<tr>
<td>Assumption</td>
<td>$D_{SA}$</td>
<td>$\subseteq$</td>
</tr>
<tr>
<td>Guarantee</td>
<td>$D_{SG}$</td>
<td>$\subseteq$</td>
</tr>
<tr>
<td>Component</td>
<td>$D_C$</td>
<td>$\subseteq$</td>
</tr>
<tr>
<td>Composite Comp.</td>
<td>$D_{CC}$</td>
<td>$\subseteq$</td>
</tr>
<tr>
<td>Atomic Comp.</td>
<td>$D_{CA}$</td>
<td>$\subseteq$</td>
</tr>
<tr>
<td>Contract</td>
<td>$D_K$</td>
<td>$\subseteq$</td>
</tr>
</tbody>
</table>

Disjoint: $D_{SA} \cap D_{SG} \subseteq \bot$, $D_{CC} \cap D_{CA} \subseteq \bot$

Disjoint: $D_{SS}, D_S, D_C, D_K$ are pairwise disjoint

Table 2: DL roles corresponding to relations in CBSD framework with domain and range restrictions.

<table>
<thead>
<tr>
<th>DL role</th>
<th>Domain</th>
<th>Range</th>
<th>Inverse of</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{fulls}$</td>
<td>$\exists R_{fulls}. T \subseteq D_S$</td>
<td>$T \subseteq \forall R_{fulls}. D_S$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$R_{assOf}$</td>
<td>$\exists R_{assOf}. T \subseteq D_{SA}$</td>
<td>$T \subseteq \forall R_{assOf}. D_{SG}$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$R_{alcTo}$</td>
<td>$\exists R_{alcTo}. T \subseteq D_K$</td>
<td>$T \subseteq \forall R_{alcTo}. D_C$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$R_{comp}$</td>
<td>$\exists R_{comp}. T \subseteq D_{CA}$</td>
<td>$T \subseteq \forall R_{comp}. D_{CC}$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$R_{hasA}$</td>
<td>$\exists R_{hasA}. T \subseteq D_K$</td>
<td>$T \subseteq \forall R_{hasA}. D_{SA}$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$R_{hasG}$</td>
<td>$\exists R_{hasG}. T \subseteq D_K$</td>
<td>$T \subseteq \forall R_{hasG}. D_{SG}$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$R_{hasS}$</td>
<td>$\exists R_{hasS}. T \subseteq D_{SS}$</td>
<td>$T \subseteq \forall R_{hasS}. D_S$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$R_{gPartOf}$</td>
<td>$\exists R_{gPartOf}. T \subseteq D_{SG}$</td>
<td>$T \subseteq \forall R_{gPartOf}. D_K$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$R_{aPartOf}$</td>
<td>$\exists R_{aPartOf}. T \subseteq D_{SA}$</td>
<td>$T \subseteq \forall R_{aPartOf}. D_K$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$R_{fullBy}$</td>
<td>$\exists R_{fullBy}. T \subseteq D_S$</td>
<td>$T \subseteq \forall R_{fullBy}. D_S$</td>
<td>$\top$</td>
</tr>
</tbody>
</table>

components are partitioned into Atomic and Composite components, the axiom $D_C \equiv D_{CA} \sqcup D_{CC}$ is added to the Tbox.

Finally, Definition 4 and condition (iii) from Definition 7 can be encoded as Tbox axioms independently of the provided FM or a model $M$. The axiom $D_K \subseteq 1 \ R_{hasA}. D_{SA} \cap 1 \ R_{hasG}. D_{SG}$ corresponds to Definition 4, and axioms $\exists R_{assOf}. T \subseteq \bot$ and $\exists R_{fullBy}. T \subseteq \bot$ correspond to condition (iii) of Definition 7. The semantics of the construct $\exists R.Self$ is defined in [20].

5.2 Arbitrary FM as Tbox axioms

In order to verify conditions (i)-(v) from Corollary 1, it is necessary to consider all the dependencies between the features, i.e. the FM. Several publications [16,30,40] have considered the encoding of a feature model as Tbox axioms with the purpose of establishing various properties [8] of the set of product configurations defined by the FM. In other words, these approaches try to establish certain properties of the problem space. On the contrary, the present paper aims at assuring properties
of the solution space but in order to reason about presence conditions it is still necessary to have access to the information about the constraints declared in an FM. Because the use of the FM is different, the encoding of the FM in the present paper is more compact, and unlike work [16,30,40], the semantics of each feature \( f_i \) is interpreted as the set of product individuals whose product configuration \( P_C \) is such that \( (f = \text{true}) \in P_C \). The Tbox encoding of an arbitrary feature model \( \mathcal{V} = (\mathcal{F}, \mathcal{C}) \) is shown in Table 3.

<table>
<thead>
<tr>
<th>FM construct</th>
<th>Tbox encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature ( f_i )</td>
<td>( D_{f_i} \in \mathcal{N}<em>C ), ( D</em>{f_i} \subseteq D_P )</td>
</tr>
<tr>
<td>Parent ( f_i ) - child ( f_m )</td>
<td>( D_{f_m} \subseteq D_{f_i} )</td>
</tr>
<tr>
<td>Parent ( f_i ) - mandatory child feature ( f_i )</td>
<td>( D_{f_i} \subseteq D_{f_m} )</td>
</tr>
<tr>
<td>Parent ( f_i ) - optional child feature ( f_m )</td>
<td>-</td>
</tr>
<tr>
<td>Parent ( f_i ) - alternative feature group ( f_1, \ldots, f_n )</td>
<td>( D_{f_i} \equiv \bigcup_{1}^{n} D_{f_j}, D_{f_i} \cap D_{f_k \neq j} \subseteq \bot )</td>
</tr>
<tr>
<td>Parent ( f_i ) - or feature group ( f_1, \ldots, f_n )</td>
<td>( D_{f_i} \equiv \bigcup_{1}^{n} D_{f_j} )</td>
</tr>
<tr>
<td>Feature ( f_i ) requires feature ( f_j )</td>
<td>( D_{f_j} \subseteq D_{f_i} )</td>
</tr>
</tbody>
</table>

Encoding arbitrary Boolean constraints in \( \mathcal{C} \in \mathcal{E} \) is exemplified using the running example from Figure 1b. Consider the arbitrary constraint \( \text{Diesel} \land \text{V8} \rightarrow \text{LeftTank} \land \text{RightTank} \). The corresponding Tbox axiom is \( D_{\text{Track}} \subseteq \neg(D_{\text{Diesel} \cap D_{\text{V8}}} \cup (D_{\text{LeftTank}} \cap D_{\text{RightTank}})) \), where \( D_{\text{Track}} \) is the root of the FM.

### 5.3 Arbitrary model \( \mathcal{M} \) as Tbox axioms

Given an arbitrary model \( \mathcal{M} = (\mathcal{O}, \mathcal{R}, [\,]^{T}) \), the Tbox encoding consists of the encoding of: objects in \( \mathcal{O} \), relations in \( \mathcal{R} \), presence conditions of objects in \( \mathcal{O} \), the absence of relations because of the underlying open-world assumption, and inverse relation described in the beginning of section 5.1. The encoding is exemplified using particular types of objects and relations but the encoding principles are similar for all other types of objects and relations.

a) **Encoding objects.** For each \( o_i \in \mathcal{O} \) such that \( o_i^{T} = \text{Guarantee} \), the Tbox is complemented with axioms \( D_{o_i} \in \mathcal{N}_C \), \( D_{o_i} \subseteq D_{\text{Gu}} \), \( D_{\text{Gu}} \equiv \bigcup_i D_{o_i} \), and for each \( D_{o_i} \) and \( D_{o_j \neq i} \), \( D_{o_i} \cap D_{o_j} \subseteq \bot \).

b) **Encoding relations.** For each \( o_i \in \mathcal{O} \) such that \( o_i^{T} = \text{Guarantee} \) and each pair \( (o_1, o_1), \ldots, (o_1, o_n) \in \mathcal{R} \) where \( o_1, o_1 \rangle^T = \cdots = (o_1, o_n)^T \) fulfills, the Tbox is complemented with axioms \( D_{o_i} \subseteq \bigcap_{j=1}^{n} D_{o_j} \), and \( D_{o_i} \subseteq \forall R_{\text{fals}}. \langle \bigcup_{j=1}^{n} D_{o_j} \rangle \).

### Encoding presence conditions.** For each \( o_i \in \mathcal{O} \) labeled with a presence condition \( \varphi \), the Tbox is complemented with axioms \( D_{\varphi} \in \mathcal{N}_C \), \( D_{\varphi} \subseteq D_{P} \), and \( D_{\varphi} \equiv \varphi_{DL} \), where \( \varphi_{DL} \) is the DL encoding of the presence condition such that each feature is replaced with the corresponding DL concept, each
∧ operator is replaced with \( \cap \) operator, and each \( \lor \) operator is replaced with a \( \cup \) operator.

d) **Encoding the absence of relations.** For each \( o_i \in \mathcal{O} \) such that \( o_i^T = \text{Guarantee} \) and \( D_{o_i} \subseteq D_S \), where \( D_S \) is the domain restriction of role \( R_{\text{fulls}} \), if there is no \( o_j \in \mathcal{O} \) such that \( (o_i, o_j) \in \mathcal{R} \) and \( (o_i, o_j)^T = \text{fulfills} \), the Tbox is complemented with the axiom \( D_{o_i} \subseteq \neg(\exists R_{\text{fulls}}. D_S) \) where \( D_S \) is the range restriction of the role \( R_{\text{fulls}} \).

e) **Encoding inverse relations.** As mentioned, verifying certain rules will require the relations inverse to \text{fulfills}, \text{hasAssumption}, and \text{hasGuarantee}. Because, each pair \((o_i, o_j) \in \mathcal{R}\) implicitly defines its inverse pair \((o_j, o_i)\), for each pair \((o_i, o_j) \in \mathcal{R}\) such that \((o_i, o_j)^T \in \{\text{fulfills}, \text{hasAssumption}, \text{hasGuarantee}\}\), its inverse pair can be uniquely determined, and then for each such pair the encoding principle for relations can be used to encode relations inverse to \text{fulfills}, \text{hasAssumption}, and \text{hasGuarantee} which are represented by roles \( R_{\text{fullBy}}, R_{a\text{PartOf}}, \) and \( R_{g\text{PartOf}} \).

Using the running example, we further exemplify the usage of the presented encoding principles. For example, object \text{FuelSensor} from Figure 3 is encoded as \( D_{\text{FuelSensor}} \subseteq D_{C_A} \) and declared to be pairwise disjoint with each of the three remaining concepts \( D_{\text{BMS}}, D_{\text{COO}}, \) and \( D_{\text{ICL}} \). Regarding the encoding of relations, consider the pair \((A_1, G_1)^T = \text{assumptionOf}\) whose corresponding encoding is \( D_{A_1} \subseteq 1\ R_{\text{assOf}}. D_{G_1} \) and \( D_{A_1} \subseteq \forall R_{\text{assOf}}. D_{G_1} \). It should be noted that the reason for encoding relations using both value restrictions, e.g. \( \forall R_{\text{assOf}}. D_{G_1} \), and number restriction, e.g. \( 1 R_{\text{assOf}}. D_{G_1} \), is because value restriction does not guarantee the existence of the relation. Regarding presence condition encoding, consider \( \Phi(A_2) = \varphi_5 = \text{Truck} \land \text{V8} \) which is encoded as \( D_{\Phi(A_2)} \equiv D_{\text{Truck}} \cap D_{\text{V8}} \). Finally, encoding the absence of a relation is exemplified using assumption \( A_1 \). Because each \text{Assumption} is a \text{Specification}, and the concept \( D_S \) which corresponds to \text{Specification} is the domain restriction of the role \( R_{\text{fulls}} \), because there is no \text{Specification} \( S_k \) such that \((A_1, S_k)^T = \text{fulfills}\), then the Tbox is complemented with the axiom \( D_{A_1} \subseteq \neg(\exists R_{\text{fulls}}. D_S) \) where \( D_S \) is the range restriction of the role \( R_{\text{fulls}} \).

The presence condition semantics defined in Definition 12 is preserved by the corresponding Tbox encoding due to the following theorem.

**Theorem 4.** Given a presence condition \( \varphi \) expressed over features \( f_1, \ldots, f_n \), and corresponding concepts \( D_\varphi \) and \( D_{f_1}, \ldots, D_{f_n} \), if \( D_\varphi \equiv \varphi_{\text{DL}} \), where \( \varphi_{\text{DL}} \) is obtained by replacing each feature \( f_i \) from \( \varphi \) with the concept \( D_{f_i} \), each \( \land \) operator with \( \cap \), and each \( \lor \) operator with \( \cup \), then it holds that \( \llbracket \varphi \rrbracket = D_\varphi^\mathbb{I} \).

Given the correspondence between the set of individual products defined by a presence condition and the interpretation of the concept representing the presence condition, it is possible to draw conclusions about each such individual product without actually creating any real-world individuals.
6 Verifying that $\mathcal{M}$ is a proper specification structure

After an arbitrary FM and an arbitrary $\mathcal{M}$ are encoded as a Tbox axioms, the constraints from Definition 7, Definition 8, and condition (i)-(v) from Corollary 1, must be encoded as Tbox axioms. In what follows, the encoding of each constraint is presented.

6.1 Encoding the constraints

Definition 7: Let $\mathcal{C}$ be a possibly empty set of atomic components, $\mathcal{K}$ be a possibly empty set of contracts such that each contract $K_i \in \mathcal{K}$ is allocated to a single component in $\mathcal{C}$ where each $A_i$ and $G_i$ is unique. Furthermore, let $C'$ be a composite component such that $C' = \bigotimes_i C_i$, and $(A', G')$ be a contract allocated to $C'$. Then, an edge-labeled directed graph $\mathcal{D} = (\mathcal{N}, \mathcal{E})$ is a specification structure, if

(i): Each node $n \in \mathcal{N}$ is either an assumption $A_i$, $A'$ or a guarantee $G_i$, $G'$. Given this constraint, for $o_i \in \mathcal{O}$, such that $o_i^T = \text{SpecificationStructure}$, the following entailment should hold:

$$\mathcal{K} \models D_{o_i} \subseteq \forall R_{\text{hasS}.}((\exists R_{\text{partOf}.}(\exists R_{\text{alcTo}.}(\exists R_{\text{comp}.}D_{C'})) \cup D_{C'})) \cup (\exists R_{\text{partOf}.}(\exists R_{\text{alcTo}.}(\exists R_{\text{comp}.}D_{C'})) \cup D_{C'}))$$

(ii): Each edge $e = (n_i, n_j) \in \mathcal{E}$ corresponds to either a single fulfills($S_i, S_j$) or a single assumptionOf($S_i, S_j$) relation. Because the range restriction of the role $R_{\text{hasS}}$ is $D_S$ where $D_S = D_{S_A} \cap D_{S_G}$, and given the domain and range restrictions in Table 2, the only roles that can exists between objects of type Specification are $R_{\text{fulls}}$ and $R_{\text{assOf}}$.

(iii): see Section 5.1.

(iv): For each assumptionOf($S_i, S_j$) relation, $S_i$ is an assumption, $S_j$ is a guarantee and the ordered pair ($S_i, S_j$) is a contract. The first part of the constraint, regarding the types of specifications that can be related via the assumption of relation, is enforced through the domain and range restrictions on the $R_{\text{assOf}}$ role. Regarding the second part of the rule, for each pair $\{o_i, o_j\} \subseteq \mathcal{O}$ such that $(o_i, o_j) \in \mathcal{R}$ and $(o_i, o_j)^T = \text{assumptionOf}$, the following entailment should hold:

$$\mathcal{K} \models D_{o_i} \subseteq 1 R_{\text{partOf}.}(= 1 R_{\text{hasG}.}D_{o_i})$$
$$\mathcal{K} \models D_{o_j} \subseteq 1 R_{\text{partOf}.}(= 1 R_{\text{hasA}.}D_{o_j})$$

(v)-a: For each edge fulfills$(S_i, S_j)$ if $S_i$ is a guarantee then $S_j$ is a guarantee of a contract $K_i$ and $S_j$ is either an assumption $A_i$ or the guarantee $G'$. For each $(o_i, o_j) \in \mathcal{R}$ where $(o_i, o_j)^T = \text{fulfills}$ and $o_i^T = \text{Guarantee}$, the following inclusion should be entailed by the knowledge base:

$$\mathcal{K} \models D_{o_i} \subseteq (\exists R_{\text{partOf}.}(\exists R_{\text{alcTo}.}(\exists R_{\text{comp}.}D_{C'}))) \cap (\forall R_{\text{fulls}.}(\exists R_{\text{partOf}.}(\exists R_{\text{alcTo}.}(\exists R_{\text{comp}.}D_{C'})) \cup D_{C'}))$$
(v)-b: For each edge fulfills\((S_i, S_j)\) if \(S_i\) is an assumption then \(S_j\) is the assumption \(A'\) and \(S_j\) is any \(A_i\). For each \((o_i, o_j)\) \(\in R\) where \((o_i, o_j)^T = \text{fulfills}\) and \(o_i^T = \text{Assumption}\), the following inclusion should be entailed by the knowledge base:

\[
\mathcal{K} \models D_{o_i} \subseteq (\exists R_{a \text{PartOf}}(\exists R_{\text{alcTo}}D_{C'})) \sqcap
(\forall R_{\text{fullBy}}(\exists R_{g \text{PartOf}}(\exists R_{\text{alcTo}}D_{C'}))))
\]

**Definition 8:** A specification structure is proper if

(i): For each assumption \(A_i\) of a contract \(K_i\), there exists a specification \(S_k\) such that fulfills\((S_k, A_i)\), where \(S_k\) is either \(A'\) or \(G_i\). For each \\{(o_k, o_c), (o_k, o_o)\}\(\subseteq R\) where \((o_k, o_c)^T = \text{allocatedTo}\), \((o_k, o_o)^T = \text{hasAssumption}\), \(o_k^T = \text{Atomic}\), and \(o_o^T = \text{Assumption}\), the following entailment should hold:

\[
\mathcal{K} \models D_{o_k} \subseteq (\exists R_{\text{fullBy}}(\exists R_{\text{gPartOf}}(\exists R_{\text{alcTo}}D_{C'}))) \sqcap
(\exists R_{\text{aPartOf}}(\exists R_{\text{alcTo}}D_{C'})))
\]

(ii): There exists a guarantee \(G_i\) such that fulfills\((G_i, G')\). For each \\{(o_k, o_c), (o_k, o_o)\}\(\subseteq R\) where \((o_k, o_c)^T = \text{allocatedTo}\), \((o_k, o_o)^T = \text{hasGuarantee}\), \(o_k^T = \text{Composite}\), and \(o_o^T = \text{Guarantee}\), the following entailment should hold:

\[
\mathcal{K} \models D_{o_k} \subseteq (\exists R_{\text{fullBy}}(\exists R_{\text{gPartOf}}(\exists R_{\text{alcTo}}D_{C'})))
\]

(iii): see last paragraph in Section 3.2.

**Corollary 1:** Given a feature model \(V\), and a PL specification structure \(\mathcal{D}\), if

(i): For each contract \(K = (A_i, G_i)\), it holds that \(\Phi(A_i) \models \varepsilon \Phi(G_i)\) and \(\Phi(G_i) \models \varepsilon \Phi(A_i)\). For each \\{(o_k, o_o), (o_k, o_o)\}\(\subseteq R\) where \((o_k, o_o)^T = \text{hasAssumption}\), \((o_k, o_o)^T = \text{hasGuarantee}\), \(o_k^T = \text{Composite}\), and \(o_o^T = \text{Guarantee}\), the following entailment should hold:

\[
\mathcal{K} \models D_{\Phi(o_k)} \subseteq D_{\Phi(o_o)} \quad \text{and} \quad \mathcal{K} \models D_{\Phi(o_o)} \subseteq D_{\Phi(o_k)}
\]

(ii): For each contract \(K = (A_i, G_i)\) allocated to a component \(C_i\) it holds that \(\Phi(A_i) \models \varepsilon \Phi(C_i)\) and \(\Phi(G_i) \models \varepsilon \Phi(C_i)\). For each \\{(o_k, o_o), (o_k, o_o)\}\(\subseteq R\) where \((o_k, o_o)^T = \text{hasAssumption}\), \((o_k, o_o)^T = \text{hasGuarantee}\), \(o_k^T = \text{allocatedTo}\) and \(\Phi(o_k), \Phi(o_o)\) are the corresponding presence conditions, the following entailment should hold:

\[
\mathcal{K} \models D_{\Phi(o_k)} \subseteq D_{\Phi(o_o)} \quad \text{and} \quad \mathcal{K} \models D_{\Phi(o_o)} \subseteq D_{\Phi(o_k)}
\]

(iii): For the composite component \(C'\) composed of \(C_i\), for each \(C_i\) it holds that \(\Phi(C_i) \models \varepsilon \Phi(C')\). For each \((o_i, o_j)\) \(\in R\) where \((o_i, o_j)^T = \text{comprises}\) and \(\Phi(o_i), \Phi(o_j)\) are the corresponding presence conditions, the following should be entailed by the knowledge base \(\mathcal{K} \models D_{\Phi(o_i)} \subseteq D_{\Phi(o_j)}\).

(iv): For each contract \(K_i = (A_i, G_i)\) allocated to an atomic component \(C_i\) and for each specification \(S_k\) such that fulfills\((S_k, A_i)\) it holds that \(\Phi(A_i) \models \varepsilon\)
\[ \bigvee_{\text{fulfills}(S_k, A_i)} \Phi(S_k). \]

Given \( M \), for each set of pairs \( \{(o_1, o_i), \ldots, (o_n, o_i)\} \subseteq R \)
such that \( o_i^T = \text{Assumption} \), where \( (o_1, o_i)^T = \ldots = (o_n, o_i)^T = \text{fulfills} \), and \( \Phi(o_i), \Phi(o_1), \ldots, \Phi(o_n) \) are the corresponding presence conditions, the following
should be entailed by the knowledge base \( K \models D\Phi(o_i) \sqsubseteq \sqcup_{i=1}^n D\Phi(o_i) \).

(v): For the contract \( K' = (A', G') \) allocated to the composite component \( C' \)
and for each guarantee \( G_i \) of a contract \( C_i \) such that \( \text{fulfills}(G_i, G') \)
it holds that \( \Phi(G') \models \bigvee_{\text{fulfills}(G_i, G')} \Phi(G_i) \). For the set of pairs \( \{(o_k, o_g), (o_k, o_c)\} \subseteq R \)
where \( (o_k, o_g)^T = \text{hasGuarante} \), \( (o_k, o_c)^T = \text{allocatedTo} \), \( o_c^T = \text{Composite} \), and a set of
pairs \( \{(o_1, o_g), \ldots, (o_n, o_g)\} \subseteq R \) where \( (o_1, o_g)^T = \ldots = (o_n, o_g)^T = \text{fulfills} \)
and \( \Phi(o_g), \Phi(o_1), \ldots, \Phi(o_n) \) are the corresponding presence conditions, the following
should be entailed by the knowledge base \( D\Phi(o_g) \sqsubseteq \sqcup_{i=1}^n D\Phi(o_i) \).

### 6.2 Verifying the constraints

As can be seen from the previous section, given an arbitrary model \( M \), each
constraint from Definition 7, Definition 8, and constraints (i)-(v) from Corollary 1 can be formulated as inclusion axioms \( D_i \sqsubseteq D_j \). In accordance with [21],
verifying that a KB entails the above inclusions is equivalent to verifying that
the KB is not satisfiable if the Tbox is complemented with a fresh concept
\( D_{\text{Fresh}} \equiv D_i \sqcap \neg D_j \). Let \( R \) be a set of all such axioms required to verify if a
model \( M \) is a proper specification structure.

**Theorem 5.** Given a model \( M \), a corresponding knowledge base \( K = (T, A) \),
and the set \( R \), if \( K \) is not satisfiable for each \( r \in R \), then the model \( M \) is a
proper specification structure.

### 6.3 Implementation

In order to validate the presented encoding, the running example was manually
implemented as an OWL ontology using the Protege [11] ontology editor and
HermiT [35] reasoner was used to perform the reasoning tasks. The resulting
ontology was \( SRIQ \) DL and it contained 463 axioms with 45 additional axioms
needed to verify if the running example is a proper specification structure.
Because presence conditions of assumption \( A_4 \) and guarantee \( G_4 \) do not entail the
presence condition of the component \( ICL \), the running example is not a proper
specification structure. The resulting ontology is available online\(^3\).

### 7 Conclusion

Modern industry is facing significant challenges in designing dependable sys-
tems due to their increasing complexity and heterogeneity, especially when PL
development is considered.

In order to treat these challenges we have used CBSD, which although formal, is perhaps the only paradigm which strikes a balance between structured,
formal systems engineering and well-established engineering principles of design refinement and composition, i.e., integration. By building on established CBSD concepts, the first contribution of the present paper is a CBSD extension with variability management capabilities, which allows CBSD specifications of arbitrary PLs. Corollary 1 defines the conditions on CBSD models of PLs that allow inferring the properties satisfied by each product configuration from the properties satisfied by its constituent components. Then, as a step towards providing first, general-purpose tool-support for CBSD specification of PLs, the second contribution is the definition of an encoding of arbitrary CBSD model of PLs, and arbitrary FMs as a set of Tbox axioms. Then the constraints on CBSD models, which allows the previously described inference were also encoded as Tbox axioms and verification of these constraints was reduced to the verification of the KB satisfiability. Finally, a small, but real, industrial PL was modeled using CBSD, implemented as an OWL ontology, and an off-the-shelf reasoner was successfully used to verify the CBSD model. We believe that the defined encoding can form the basis for a novel tool for creation and automated analysis of CBSD models. Future work includes generalizations of the presented PLE extension of CBSD and consequently extending the presented DL encoding.

References