Formalizing ASP-based Product Configuration: Work in Progress

Nicolas Rühling[1,3], Torsten Schaub[1,2], and Philipp Wanko[1,2]

[1] University of Potsdam, Germany
[2] Potassco Solutions, Germany
[3] UP Transfer, Germany

Abstract. We present a mathematical formalization of product configuration problems inspired by an earlier formalization, and motivated by our work on the CoomSuite. The latter contains, among others, a pipeline for translating product models from the industrial configuration language COOM into ASP, and our formalization allows us to prove the correctness of the corresponding encoding. Different from our earlier formalization, our approach defines the semantics at an "atomic" level, where all configuration elements are explicitly defined. We argue that this offers a different perspective towards understanding the essential elements making up product configuration problems.

1 Introduction

We present a mathematical formalization for *product configuration* problems that is largely motivated by previous work in [1], which developed the workbench CoomSuite for experimenting with product configuration problems specified in the industrial configuration language Coom [3]. Among others, the CoomSuite includes a pipeline for translating and solving Coom product models with ASP. Using our new formalization, we provide a solid, theoretical foundation to this ASP encoding and were able to prove the correctness of a simplified version of it.

Another configuration formalization was presented in [9], inspired by frameworks in the literature [10, 7, 5] which aim at capturing the underlying problem in a compact way. However, here, our goal is to capture the semantics at an "atomic" level where all elements that make up the configuration are explicitly defined. As a result, our approach may result in a more verbose representation but we argue that this reduction provides another perspective towards understanding the essential elements defining a product configuration problem and may serve as a basis for studying more complex concepts like defaults and interactivity.

We intentionally limit ourselves to product configuration problems that have the structure of a tree, i.e., there exists a single root object representing the product to be configured, and all other objects are either direct subparts of the product or subparts of other parts, as these are the kinds of problems that can be modeled using Coom. This also means that we do not allow for a taxonomy and for associations [7, 5], although, we would like to investigate such extensions in the future.

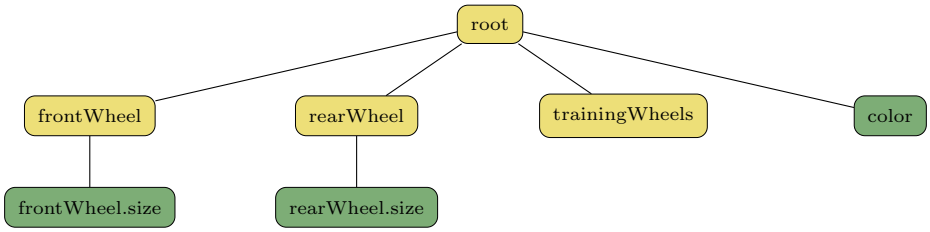

Fig. 1. Tree structure of the *KidsBike* example.

2 Configuration model and solutions

A *configuration model* $\mathcal{M}$ is a tuple (O, r, p, D, d, C), where

1. (O, r, p) is a rooted, directed tree, where
   (a) $O = P \cup A$ is a set of *objects* partitioned into a set P of *parts* and a set A of *attributes*,
   (b) $r \notin O$ is the *root object*,
   (c) $p : O \to P_r = P \cup \{r\}$ is a total function mapping objects to their *parents*,
2. D is a domain of *values*, with $D_{\boldsymbol{u}} = D \cup \{\boldsymbol{u}\}$ the extended domain which contains the special value $\boldsymbol{u}$ standing for *undefined*,
3. $d : A \to 2^D$ is a *domain function* that assigns to each attribute a set of values from the domain, and
4. C is a set of *table constraints* of the form $\langle (o_1, \ldots, o_n), R \rangle$, with
   (a) $o_1, \ldots, o_n \in O$ for $n \geq 1$, and
   (b) $R \subseteq D_{\boldsymbol{u}}^n$ an n-ary relation over $D_{\boldsymbol{u}}$.

Notet that the definition of the parent function p ensures that attributes are always leaf nodes that are directly associated to some part $o \in P_r$.

For the sake of illustration, consider a slightly adapted version of the *KidsBike* example from [1]. It has a front and rear wheel which are mandatory parts and optionally can be equipped with training wheels. The configurable attributes are the bike's color, which can be red, green, yellow, or blue, and the size of the front and rear wheel, which share the same domain of 14, 16, 18, or 20. We can express this formally as a model $\mathcal{M} = (O, r, p, D, d, C)$, where the set of objects O consists of the union of the parts $P = \{\mathit{frontWheel}, \mathit{rearWheel}, \mathit{trainingWheels}\}$ and the attributes $A = \{\mathit{color}, \mathit{frontWheel.size}, \mathit{rearWheel.size}\}$. The tree structure of the *KidsBike* is shown in Figure 1, where the yellow and green nodes represent parts and attributes, respectively. All three parts and the *color* attribute are direct children of the root object and the two attributes representing the wheel sizes both have their corresponding wheel part as parent,i.e., we have $p(\mathit{frontWheel}) = p(\mathit{rearWheel}) = p(\mathit{trainingWheels}) = p(\mathit{color}) = r$ and $p(\mathit{frontWheel.size}) = \mathit{frontWheel}$ and $p(\mathit{rearWheel.size}) = \mathit{rearWheel}$. Lastly, the attribute domains are defined as $d(\mathit{color}) = \{\mathit{Red}, \mathit{Green}, \mathit{Yellow}, \mathit{Blue}\}$ and $d(\mathit{frontWheel.size}) = d(\mathit{rearWheel.size}) = \{14, 16, 18, 20\}$.

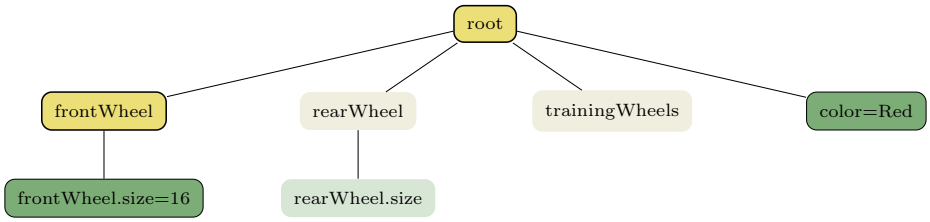

Fig. 2. Instantiation of the *KidsBike*.

Furthermore, the set of constraints is $C = \{c_0, c_1, c_2, c_w, c_t\}$, where the first three express knowledge about possible combinations of parts and attribute values, and the latter two are cardinality constraints. We describe them here in natural language only before giving a more detailed treatment of constraints and how to capture them within the formalism in Section 3.

c_0: A yellow bike is compatible with a front wheel of size 18 or 20.
c_1: If a bike has training wheels, then it must have a rear wheel size of at most 16.
c_2: The size of the front and rear wheel have to be equal.
c_w: The front and rear wheel are mandatory parts.
c_t: The training wheels are an optional part.

In the rest of the section, we provide definitions about instantiating a model and determining its solutions. An *instantiation* $\mathcal{I}$ of a configuration model $\mathcal{M}$ is a pair (I, v), where

1. $I \subseteq O_r = O \cup \{r\}$ is a set of *included objects* spanning a subtree of (O, r, p), such that
   (a) $r \in I$,
   (b) if $o \in I \setminus \{r\}$, then $p(o) \in I$, and
   (c) if $o \in I$ and $p(a) = o$ for some $a \in A$, then $a \in I$,
2. $v : A \to D_{\boldsymbol{u}}$ is a *valuation (function)* that assigns to each included attribute a value from its domain and undefined otherwise,
   (a) if $a \in I \cap A$, then $v(a) \in d(a)$,
   (b) if $a \in A \setminus I$, then $v(a) = \boldsymbol{u}$.

A possible instantiation for the *KidsBike* can be seen in Figure 2, where the grayed out objects represent objects that are not included. Formally, this instantiation is $\mathcal{I} = (I, v)$, where

$I = \{r, color, frontWheel, frontWheel.size\}$, and
$v(color) = Red$, $v(frontWheel.size) = 16$
$v(rearWheel.size) = \boldsymbol{u}$

We have to include the root object r and accordingly the *color* attribute because it is a direct child of r. Since we included the *frontWheel*, we also need to include its direct child attribute *frontWheel.size*. Lastly, our valuation function v assigns

values to both included attributes from their corresponding domain, and leaves the *rearWheel.size* undefined as it is not included.

Note that the conditions imposed on instantiations are only requirements about its basic structure. However, in practice we are also interested in whether an instantiation satisfies the given constraints. We say that an instantiation $\mathcal{I}$ of $\mathcal{M}$ *satisfies* a constraint $\langle (o_1, \ldots, o_n), R \rangle \in C$, if there exists $(v_1, \ldots, v_n) \in R$ such that for each $1 \leq i \leq n$ we have:

1. For $o_i \in P$,
   (a) $o_i \notin I$ if $v_i = \boldsymbol{u}$, and
   (b) $o_i \in I$ otherwise,
2. For $o_i \in A$, $v_i = v(o_i)$.

Thus, for parts we only check includedness and for attributes we check if the value assigned by v coincides with a table entry. Note that this assigned value can also be $\boldsymbol{u}$ which represents that the attribute is not included in the instantiation.

An instantiation $\mathcal{I}$ of $\mathcal{M}$ is *valid*, if $\mathcal{I}$ satisfies all constraints in C. Let $\mathcal{M}$ be a model and let $\mathcal{I} = (I, v)$ and $\mathcal{I}' = (I', v')$ be two instantiations of $\mathcal{M}$. We say that $\mathcal{I}'$ is an *extension* of $\mathcal{I}$, if

1. $I \subseteq I'$, and
2. if $v(a) \neq \boldsymbol{u}$, then $v(a) = v'(a)$ for each $a \in A$.

We abuse notation and write $\mathcal{I} \subseteq \mathcal{I}'$.

Lemma 1. *Let $\mathcal{M}$ be a fixed configuration model.*

Then, for any three instantiations $\mathcal{I}, \mathcal{I}', \mathcal{I}''$ of $\mathcal{M}$, it holds that $\mathcal{I} \subseteq \mathcal{I}'$ and $\mathcal{I}' \subseteq \mathcal{I}''$ implies $\mathcal{I} \subseteq \mathcal{I}''$.

Given a (possibly empty) instantiation $\mathcal{I}$ of $\mathcal{M}$, we say that $\mathcal{I}'$ is a *solution* of $\mathcal{M}$ wrt $\mathcal{I}$, if

1. $\mathcal{I}'$ is a valid instantiation of $\mathcal{M}$, and
2. $\mathcal{I} \subseteq \mathcal{I}'$.

We denote the set of all solutions of $\mathcal{M}$ wrt $\mathcal{I}$ as $S(\mathcal{M}, \mathcal{I})$.

Lemma 2. *Let $\mathcal{M}$ be a configuration model.*

Then, for any two instantiations $\mathcal{I}, \mathcal{I}'$ of $\mathcal{M}$, we get that $\mathcal{I} \subseteq \mathcal{I}'$ implies $S(\mathcal{M}, \mathcal{I}') \subseteq S(\mathcal{M}, \mathcal{I})$.

3 Types of constraints

Apart from the required tree structure, our definition of a configuration model is very similar to that of a Constraint Satisfaction Problem (CSP; [4]), especially in the way that constraints are expressed as (possibly infinite) tables.

This allows us to capture a wide range of constraint types such as Boolean, arithmetic, cardinality, and (directional) assignment constraints. We demonstrate this by means of the *KidsBike*'s constraints which we introduced in the previous section using natural language. Next, we show how to express them within the formalism. Throughout this section, let $\langle (o_1, \ldots, o_n), R \rangle$ be a constraint. We call $(o_1, \ldots, o_n)$ the *scope* of the constraint.

Attribute table constraints An *attribute table constraint* is a constraint where the scope of the constraint is restricted to attributes, i.e., $o_1, \ldots, o_n \in A$. An example of this is contraint c_0, which can be represented by $\langle (\mathit{color}, \mathit{frontWheel.size}), R \rangle$, with

$$R = \{(\mathit{Yellow}, 18), (\mathit{Yellow}, 20)\} \cup \{(c, s) \mid c \in \{\mathit{Red}, \mathit{Green}, \mathit{Blue}\} \text{ and } s \in d(\mathit{frontWheel.size})\}\,.$$

Written out extensionally, the relation R consists of $2 + 3 \times 4 = 14$ tuples.

The special value *undefined* In our formalism, we allow for attributes to be undefined and this is represented by the special value $\boldsymbol{u}$. This value can be used to make explicit different constraint semantics, as regards to the way they deal with undefined variables (which can be parts or attributes).

We say that a constraint $\langle (o_1, \ldots, o_n), R \rangle$ is *strict*, if $\boldsymbol{u}$ does not occur in R, thereby enforcing definedness, respectively, inclusion of all $o_1, \ldots, o_n$ in the solution. Otherwise, we say that a constraint is *non-strict*, thus the constraint can be satisfied even if some objects are undefined. For example, we can express constraint c_1 by $\langle (\mathit{trainingWheels}, \mathit{rearWheel.size}), R \rangle$ with

$$R = \{(1, 14), (1, 16), (\boldsymbol{u}, 14), (\boldsymbol{u}, 16), (\boldsymbol{u}, 18), (\boldsymbol{u}, 20)\}\,.$$

For parts (e.g., *trainingWheels*), there are only two cases wrt constraint satisfaction. Either they are included, which is represented by the value 1 in R, or they are undefined, which is represented by $\boldsymbol{u}$.[4] If R consisted just of the first two tuples, then the constraint would be strict and enforce the inclusion of the *trainingWheels* and a *rearWheel* of size 14 or 16. However, the remaining four tuples allow for the *trainingWheels* to be undefined and in that case for the size of the *rearWheel* to vary freely. The constraint therefore can be seen as a the material implication $\mathit{trainingWheels} \rightarrow \mathit{rearWheel.size} \leq 16$, which is vacuously satisfied if the antecedent is false, i.e., the *trainingWheels* are not included.

Boolean and arithmetic formulas As in the example above, we can encode Boolean and arithmetic formulas using table constraints. We do not define a language for such constraints but illustrate this with another example. Constraint c_2 can be expressed by the formula $\mathit{frontWheel.size} = \mathit{rearWheel.size}$, which can be represented by $\langle (\mathit{frontWheel.size}, \mathit{rearWheel.size}), R \rangle$ with

$$R = \{(14, 14), (16, 16), (18, 18), (20, 20)\}\,.$$

Cardinality constraints Another important type of constraints in configuration problems, are *cardinality constraints*. In many approaches, they are part of the formalism itself but for the sake of simplicity, we decided to treat them like other constraints. In that sense, they are defined as constraints whose scope is restricted

[4] In fact, we could choose any value other than $\boldsymbol{u}$, but for simplicity use 1.

to parts and whose relations range over $\boldsymbol{u}$ and the value 1. More precisely, we require $o_1, \ldots, o_n \in P$ and $R \subseteq \{\boldsymbol{u}, 1\}^n$.

We define the following syntactic sugar: A *group cardinality constraint* is a function $\mathit{card} : P_r \times 2^P \to 2^{\mathbb{N}}$ which defines a cardinality for a "group" of parts $G \subseteq P$ wrt to another part $o \in P_r, o \notin G$, such that $o \in I$ implies $|I \cap G| \in \mathit{card}(o, G)$. Although, this allows to specify general cardinality constraints across the tree structure, in practice, the parts in G are often (direct) children of o. For example, we can encode constraint c_w as a group cardinality constraint $\mathit{card}(r, \{\mathit{frontWheel}, \mathit{rearWheel}\}) = \{2\}$ stating that exactly both wheels must be included, and constraint c_t as $\mathit{card}(r, \{\mathit{trainingWheels}\}) = \{0, 1\}$ expressing that the inclusion of the training wheels is optional.

For a part $o \in P_r$ and a group $G = \{g_1, \ldots, g_n\} \subseteq P$ with $o \notin G$, a group cardinality constraint $\mathit{card}(o, G)$ can be translated into a table constraint $\langle (o, g_1, \ldots, g_n), R \rangle$ with

$$\begin{aligned} R = &\{(\boldsymbol{u}, v_1, \ldots, v_n) \in \{\boldsymbol{u}, 1\}^{n+1}\} \\ &\cup \{(1, v_1, \ldots, v_n) \in \{\boldsymbol{u}, 1\}^{n+1} \mid \exists k \in \mathit{card}(o, G) : \sum_{v_i \neq \boldsymbol{u}} 1 = k\}\,. \end{aligned}$$

The first set of tuples handles the case when the "reference" part o is not included and the parts in G can be in- or excluded freely. When $o = r$, this expression could be omitted as $r \in I$ always holds. The second set of tuples ensures that when o is included, exactly a number $k \in \mathit{card}(o, G)$ of parts in G are included.

User requirements A common feature in configuration problems, are so-called *user requirements*. In their most simple case, they usually set values for specific attributes or specify certain parts that need to be included but they can also consist of more complex relations between objects. For simplicity, we choose here to encode them as normal constraints as they do not alter the general behavior of the configuration model. However, note that in some use-cases a user might want to give user requirements higher priority over other constraints, for example, if there are conflicts. This is an aspect we do not consider here but which we want to investigate in future work. Furthermore, note that some forms of user requirements can also be given by providing an instantiation $\mathcal{I}$ that needs to be extended.

Coming back to the *KidsBike* example, let us additionally define the following user requirement

$$c_u = \langle (\mathit{trainingWheels}), \{(1)\} \rangle\,,$$

enforcing that the *trainingWheels* must be included. Then, our set of constraints is $C = \{c_0, c_1, c_2, c_w, c_s, c_u\}$.

Our model is $\mathcal{M} = \langle O, r, p, D, d, C \rangle$. The instantiation $\mathcal{I}$ we had defined earlier is not a valid instantiation because, among others, it does not satisfy the cardinality constraint c_w. However, we can extend $\mathcal{I}$ by including *rearWheel* and *rearWheel.size*, and assigning a value to the latter. Furthermore, we need to include the *trainingWheels* to satisfy user requirement c_u. More precisely, let

$\mathcal{I}' = (I', v')$ be defined as $I' = I \cup \{rearWheel, rearWheel.size, trainingWheels\}$ and $v'(color) = Red$, $v'(frontWheel.size) = v'(rearWheel.size) = 16$. Then, $\mathcal{I} \subseteq \mathcal{I}'$ and $\mathcal{I}'$ is a valid instantiation of $\mathcal{M}$ because it satisfies all constraints in C. It follows that $\mathcal{I}'$ is a solution of $\mathcal{M}$ wrt $\mathcal{I}$. Note that in addition to constraint c_u, our previous instantiation $\mathcal{I}$ also served as a way of expressing user requirements.

4 Proving the CoomSuite encoding

In this section, we provide a simplified version of the ASP encoding from [1] and prove its correctness with regard to our formalization.

4.1 Fact format

Let $\mathcal{M} = (P \cup A, r, p, D, d, C)$ be a configuration model. We split the set of constraints C into two disjoint subsets: A set C_t of non-cardinality constraints (or table constraints) and a set C_g of group cardinality constraints. We represent $\mathcal{M}$ as a set of facts $\Gamma(\mathcal{M})$ containing

- `part(`o`).` `parent(`o`,`$p(o)$`).` for each part $o \in P$,
- `attribute(`a`).` `parent(`a`,`$p(a)$`).` for each attribute $a \in A$, and `domain(`a`,`c`).` for each domain element $c \in d(a)$,
- for a non-cardinality constraint $\langle (o_1, \ldots, o_n), R \rangle \in C_t$, we define a *constraint identifier* c and we assign an index j to each tuple $r_j = (v_1, \ldots, v_n) \in R$,
  - `scope(`c`,(`i`,`o_i`)).` for each object $o_i \in (o_1, \ldots, o_n)$,
  - `relation(`c`,(`i`,`j`),`v_i`).` for each $r_j \in R$ and $v_i \in r_j$,
- for a group cardinality constraint, $card(o, G)$, we define a *group identifier* g, and
  - `cardinality((`o`,`g`),`n`).` for each allowed cardinality $n \in card(o, G)$,
  - `group((`o`,`g`),`p`).` for each part $p \in G$.

We represent an instantiation $\mathcal{I} = (I, v)$ of $\mathcal{M}$ as a set of facts $\Gamma(\mathcal{I})$ that contains

- `include(`o`).` for each included object $o \in I$, and
- `value(`a`,`$v(a)$`).` for each included attribute $a \in I \cap A$.

Listing 1.1 shows a simplified version of the ASP encoding from [1]. In Line 1, a choice rule is used to guess which parts and attributes are included and in Lines 2-4 the three conditions on included objects are encoded (cf. 1a-1c of the definition of an instantiation in Section 2).

Line 2 ensures that $r \in I$, Line 3 ensures that parents of included parts are also included, and Line 4 ensures that attributes are included if their parents are included. Next, in Line 6, the valuation of included attributes is guessed from the corresponding attribute domain.

Lastly, the encoding contains rules for checking constraints. First, non-cardinality constraints are encoded as table constraints. Lines 8-13 check whether

```
1  { include(X) : attribute(X); include(X) : part(X) }.
2  include("root").
3  :- include(X),   parent(X,P), not include(P).
4  :- attribute(X), parent(X,P), include(P), not include(X).

6  { value(X,V) : domain(X,V) } = 1 :- attribute(X), include(X).

8  -hit(C,Row) :- relation(C,(Col,Row),"undefined"), scope(C,(Col,X)),
9                 include(X) : part(X); include(X) : attribute(X).
10 -hit(C,Row) :- relation(C,(Col,Row),V), scope(C,(Col,X)),
11                V != "undefined", attribute(X), not value(X,V).
12 -hit(C,Row) :- relation(C,(Col,Row),V), scope(C,(Col,X)),
13                V != "undefined", part(X), not include(X).

15 violated(C) :- scope(C,_), -hit(C,Row) : relation(C,(_,Row),_).

17 satisfied(G) :- cardinality(G,N), N = #count{Y : group(G,Y), include(Y)}.
18 violated(G)  :- cardinality(G,_), G=(X,_), include(X), not satisfied(G).

20 :- violated(C).
```

Listing 1.1. Encoding for solving configuration problems

some row of the table is not satisfied by testing for the three different cases of violations. Note that we do not assign a value $\boldsymbol{u}$ to attributes in the ASP encoding.

1. for a table entry whose value is $\boldsymbol{u}$ the corresponding object is included (cf. Lines 8-9)
2. for a table entry corresponding to an attribute and whose value is not $\boldsymbol{u}$, the attribute is is not assigned the correct value (cf. Lines 10-11)
3. for a table entry corresponding to a part and whose value is not $\boldsymbol{u}$, the part is not included (cf. Lines 12-13)

A table constraint is violated if no row in the table is satisfied (cf. Line 15). A group cardinality constraint is satisfied if the number of included parts in the group is one of the specified cardinalities (Line 17). Then, a group cardinality constraint is violated if its "reference object" o is included and the constraint is not satisfied (Line 18). For both types of constraint we derive a predicate `violated/1` if it is violated, and so we can use the integrity constraint in Line 20 for both.

Note that this encoding differs mostly from the one in the CoomSuite in the way that constraints are represented. While here we only allow for table constraints, the CoomSuite encoding has a proper representation of cardinality constraints as well as Boolean and arithmetic formulas, including aggregates. See [2] and [1] for comparison.

Next, we show that this encoding is sound and complete with regards to our formalization.

Proposition 1. *(Soundness)* Given a configuration model $\mathcal{M}$, let P be the union of $\Gamma(\mathcal{M})$ and the encoding in Listing 1.1.

If X is a stable model of P, then (I, v) is a valid instantiation of $\mathcal{M}$ with

1. $I = \{o \mid \texttt{include(o)} \in X\}$
2. $v(a) = \begin{cases} c & \text{if } \texttt{value(a,c)} \in X \\ \boldsymbol{u} & \text{if } \texttt{include(a)} \notin X \end{cases}$

Proposition 2. *(Completeness)* Given a configuration model $\mathcal{M} = (P \cup A, r, p, D, d, C_t \cup C_c)$, let P be the union of $\Gamma(\mathcal{M})$ and the encoding in Listing 1.1. If $\mathcal{I} = (I, v)$ is a valid instantiation of $\mathcal{M}$, then

$$\begin{aligned}
&\Gamma(\mathcal{M}) \cup \{\texttt{include(}o\texttt{)} \mid o \in I\} \cup \{\texttt{value(}a\texttt{,}c\texttt{)} \mid a \in I \cap A, v(a) = c\} \\
&\cup \{\texttt{-hit(}c\texttt{,}j\texttt{)} \mid c \in C_t, r_j \in R, \text{ and } r_j \text{ is not satisfied by } \mathcal{I}\} \\
&\cup \{\texttt{violated(}c\texttt{)} \mid c \in C_t \text{ and } c \text{ is not satisfied by } \mathcal{I}\} \\
&\cup \{\texttt{satisfied(}c\texttt{)} \mid c = \mathit{card}(o, G) \text{ and } |I \cap G| \in \mathit{card}(o, G)\} \\
&\cup \{\texttt{violated(}c\texttt{)} \mid c = \mathit{card}(o, G) \text{ and } c \text{ is not satisfied by } \mathcal{I}\}
\end{aligned}$$

is a stable model of P.

5 Related work

Configuration frameworks in the literature [7, 5, 9] usually aim to capture the underlying problem in a compact way such as for example OOASP [5] which allows for modeling configuration problems in an object-oriented manner. These frameworks usually contain the notion of an abstract *configuration model* consisting of at least a partonomy (plus in some cases a taxonomy), expressed through a notion of abstract types or classes, which are connected by "part-of" (and "is-a") relations. Furthermore, types or classes can contain attributes and the model may contain constraints that restrict the possible configurations.

Solutions to the configuration problem are then defined as concrete instantiations of the model, i.e., as sets of objects that are instances of the abstract types, and which satisfy the constraints imposed by the model.

Different to this, the intention of our approach is to capture the semantics of product configuration problems at an "atomic" level, i.e., we explicitly define all elements that make up the configuration. Therefore, our approach contains the same notion of a configuration model, but with the difference that this model consists not of abstract but concrete entities.

A way of looking at this, is to take the abstract configuration problem and to transform it into a concrete representation where all knowledge is expressed explicitly. This can be seen as a "compilation" or "grounding" of the abstract model. As a result, our approach mights result in a more verbose representation. We only sketch this idea here and leave a more detailed study to future work. A simple demonstration can be seen in Figure 3, where on the left side we have a simple, abstract UML-style configuration model of a machine and on the right

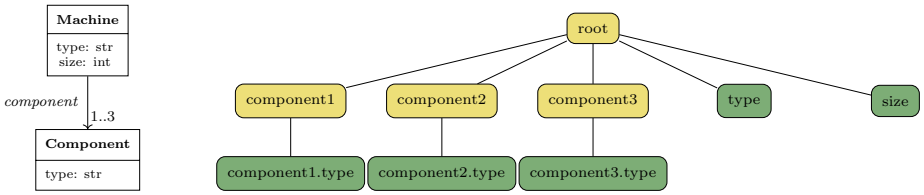

Fig. 3. Abstract configuration model and its representation in our formalism

side its corresponding representation in our formalism. The machine has two attributes *type* and *size*, and between one and three *Component* subparts which in turn have an attribute *type* as well. The intuitive idea of the translation from the abstract into the explicit representation is to consider all possible objects, i.e., the maximal cardinalities, and accordingly instantiate these together with their corresponding cardinality constraints and associated attributes.

6 Discussion

We presented a mathematical formalization of product configuration problems that is inspired by earlier work [9]. This formalization provides a solid, theoretical foundation to the ASP encoding of the CoomSuite [1] and enabled us to prove the correctness of a simplified version of it.

While our current formalization only allows for a partonomy, we would like to investigate extensions with taxonomy and general associations in the future. Other future work includes a study of defaults and/or a general methodology to express preferences between constraints, such as for example user requirements and "normal" constraints. Furthermore, we would like to compare our formalization to other approaches from the literature such as OOASP [5] and our previously published configuration formalization [9]

References

1. Baumeister, J., Herud, K., Ostrowski, M., Reutelshoefer, J., Rühling, N., Schaub, T., Wanko, P.: Towards industrial-scale product configuration. In: Dodaro, C., Gupta, G., Martinez, M. (eds.) Proceedings of the Seventeenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'24). Lecture Notes in Artificial Intelligence, vol. 15245, pp. 71–84. Springer-Verlag (2024). https://doi.org/10.1007/978-3-031-74209-5_6
2. Baumeister, J., Herud, K., Ostrowski, M., Reutelshöfer, J., Rühling, N., Schaub, T., Wanko, P.: CoomSuite. `https://github.com/potassco/coom-suite` (2025)
3. Baumeister, J., Herud, K., Reutelshöfer, J., Belli, V.: Coom language. `https://www.coom-lang.org/` (2025)
4. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
5. Falkner, A., Ryabokon, A., Schenner, G., Shchekotykhin, K.: OOASP: connecting object-oriented and logic programming. In: Calimeri, F., Ianni, G., Truszczyński, M.

(eds.) Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15). Lecture Notes in Artificial Intelligence, vol. 9345, pp. 332–345. Springer-Verlag (2015). https://doi.org/10.1007/978-3-319-23264-5_28
6. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K. (eds.) Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88). pp. 1070–1080. MIT Press (1988). https://doi.org/10.1201/b10397-6
7. Junker, U.: Configuration. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, chap. 24, pp. 837–873. Elsevier Science (2006). https://doi.org/10.1016/s1574-6526(06)80028-3
8. Niemelä, I., Simons, P., Soininen, T.: Stable model semantics of weight constraint rules. In: Gelfond, M., Leone, N., Pfeifer, G. (eds.) Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning. Lecture Notes in Artificial Intelligence, vol. 1730, pp. 317–333. Springer-Verlag (1999)
9. Rühling, N., Schaub, T., Stolzmann, T.: Towards a formalization of configuration problems for ASP-based reasoning: Preliminary report. In: Horcas, J., Galindo, J., Comploi-Taupe, R., Fuentes, L. (eds.) Proceedings of the Twenty-fifth International Configuration Workshop (CONF'23). vol. 3509, pp. 85–94. CEUR Workshop Proceedings (2023), `https://ceur-ws.org/Vol-3509/paper12.pdf`
10. Soininen, T., Tiihonen, J., Männistö, T., Sulonen, R.: Towards a general ontology of configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing **12**(4), 357–372 (1998). https://doi.org/10.1017/s0890060498124083

A Proofs

Proof of Lemma 1 We need to show that $I \subseteq I''$, and for any $a \in A$, if $v(a) \neq \boldsymbol{u}$, then $v(a) = v''(a)$.

The first part follows directly. For the second part, pick any $a \in A$ and assume that $v(a) \neq \boldsymbol{u}$. Then, from $\mathcal{I} \subseteq \mathcal{I}'$ and $\mathcal{I}' \subseteq \mathcal{I}''$ we get that $v(a) = v'(a) = v''(a)$. □

Proof of Lemma 2 Pick any $\mathcal{J} \in S(\mathcal{M}, \mathcal{I}')$. Per definition $\mathcal{J}$ is a valid instantiation of $\mathcal{M}$ and $\mathcal{I}' \subseteq \mathcal{J}$. From $\mathcal{I} \subseteq \mathcal{I}'$ and Lemma 1, it follows that $\mathcal{I} \subseteq \mathcal{J}$. Therefore, we get that $\mathcal{J} \in S(\mathcal{M}, \mathcal{I})$. □

Proof of Proposition 1 First, we check that (I, v) satisfies the conditions imposed on instantiations. Note that the set I is built up using the `include/1` predicate which only occurs in the heads of the rules in Lines 1 and 2. Since X is a stable model of P, this ensures that $I \subseteq O_r$.

Furthermore, Lines 2-4 ensure that in every stable model the three conditions on included objects are imposed (cf. 1a-1c of the definition of an instantiation in Section 2). Next, the valuation function v is built up using the `value/2` predicate which only occurs in the head of the rule in Line 6. This rule ensures that for

each included attribute $a \in I \cap A$, exactly one value $c \in d(a)$ is chosen such that $v(a) = c$.

Lastly, we need to show that all constraints in C are satisfied by (I, v). We split C into a set C_t of non-cardinality constraints (or table constraints) and into a set C_g of group cardinality constraints. Pick any non-cardinality constraint $c = \langle (o_1, \ldots, o_n), R \rangle \in C_t$. We need to check that there exists a $(v_1, \ldots, v_n) \in R$ such that for each $1 \leq i \leq n$ it holds that:

1. If $o_i \in P$, then
   (a) $o_i \notin I$ if $v_i = \boldsymbol{u}$, and
   (b) $o_i \in I$ otherwise,
2. If $o_i \in A$, then $v_i = v(o_i)$.
   (a) $o_i \notin I$ if $v_i = \boldsymbol{u}$, and
   (b) $v_i = v(o_i)$ otherwise,

Note that this definition is slightly adapted but equivalent to the one from Section 2 since we do not have an explicit value $\boldsymbol{u}$ for attributes in the ASP encoding. The negation of statements (1a) and (2a) are encoded in Lines 8-9. The negation of statement (2b) is encoded in Lines 12-13 and the negation of statement (1b) is encoded in Lines 10-11. A constraint is violated if all tuples in its relation are violated (Line 15).

A group cardinality constraint $\mathit{card}(o, G) \in C_g$ is satisfied if $o \in I$ implies $|I \cap G| \in \mathit{card}(o, G)$ and violated if $o \in I$ and it is not satisfied (Line 18). In both cases, the integrity constraint in Line 20 enforces that constraints cannot be violated in a stable model. The above shows that (I, v) is a valid instantiation of $\mathcal{M}$. □

Proof of Proposition 2 Let

$$
\begin{aligned}
X = & \Gamma(\mathcal{M}) \cup \{\texttt{include(}o\texttt{)} \mid o \in I\} \\
& \cup \{\texttt{value(}a\texttt{,}c\texttt{)} \mid a \in I \cap A, v(a) = c\} \\
& \cup \{\texttt{-hit(}c\texttt{,}j\texttt{)} \mid c \in C_t, r_j \in R, \text{ and } r_j \text{ is not satisfied by } \mathcal{I}\} \\
& \cup \{\texttt{violated(}c\texttt{)} \mid c \in C_t \text{ and } c \text{ is not satisfied by } \mathcal{I}\} \\
& \cup \{\texttt{satisfied(}c\texttt{)} \mid c = \mathit{card}(o, G) \text{ and } |I \cap G| \in \mathit{card}(o, G)\} \\
& \cup \{\texttt{violated(}c\texttt{)} \mid c = \mathit{card}(o, G) \text{ and } c \text{ is not satisfied by } \mathcal{I}\}
\end{aligned}
$$

and let G be the set of all rule instances obtained from rules in P, and G^X be the Gelfond-Lifschitz reduct of G with respect to X.

We need to show that X is a stable model of P.

Therefore, we show that X is a model of G (Item 1) and that it is a subset-minimal model of G^X (Item 2). [5]

[5] We use the fact that every stable model is a model and that any model of a program is a model of its reduct ([6], Theorem 1).

1. *X is a model of G.*
   The facts in P are satisfied by X as they are part of the translation $\Gamma(\mathcal{M})$. By construction of X, for any $o \in I$, there exists an atom `include(o)` $\in X$. Therefore, any rule instance obtained from the choice rule in Line 1 is satisfied by X. Per definition of I, we get that $r \in I$, and for each part $o \in I \cap P$, $p(o) \in I$, and for each attribute $a \in I \cap A$, $p(a) \in I$. Therefore, any instance obtained from Lines 2-4 is satisfied by X. By construction of X, for any $a \in A \cap I$, there exists exactly one atom `value(a,c)` $\in X$ for $c = v(a)$. Therefore, any rule instance obtained from the rule in Line 6 is satisfied by X. Consider a rule instance obtained from Lines 10-11 whose head is `-hit(`c`,`j`)` for some $c \in C_t$ and $r_j \in R$. We assume its body satisfied, i.e., we have `relation(`c`,(`i`,`j`),`v_i`)` and `scope(`c`,(`i`,`o`))` in X for some $v_i \in r_j$ such that $v_i \neq \boldsymbol{u}$, $o \in A$, and $v(o) \neq v_i$. Then, the tuple $r_j \in R$, where R is the relation of c is not satisfied by $\mathcal{I}$ and by construction of X, we get that `-hit(`c`,`j`)` $\in X$. It follows that the rule instance is satisfied. By similar arguments we can prove that rule instances of Lines 8-9 and Lines 12-13 are satisfied.
   Next, consider a rule instance of Line 15 whose head is `violated(`c`)` for some constraint c. We assume its body satisfied, i.e., for all tuples $r_j \in R$, we get that `-hit(`c`,`j`)` $\notin X$. Then, no $r_j \in R$ is satisfied by $\mathcal{I}$ and c is not satisfied by $\mathcal{I}$. Per construction of X, it follows that `violated(`c`)` $\in X$, and thus the rule instance is satisfied. By similar arguments we can also show that any instances from Lines 17 and Lines 18 are satisfied.
   Lastly, note that in any valid instantiation all constraints are satisfied. Therefore, `violated(`c`)` $\notin X$ for any $c \in C$ and any instance from Line 20 is satisfied.
2. *X is a subset-minimal model of G^X.*
   We proof by contradiction assuming that there exists an $X' \subset X$ that is a model of G^X. Without loss of generality, we assume that X' is a subset-minimal model of G^X. Clearly, we get that $\Gamma(\mathcal{M}) \subseteq X'$.
   Assume that `include(`o`)` $\in X \setminus X'$ for some $o \in Included$. This is a contradiction as X' does not satisfy the reduct of the instance of the rule in Line 2 with `include(`o`)` in the head. [6]
   Assume that `value(`a`,`c`)` $\in X \setminus X'$ for some $a \in A \cap I$ and $c \in d(a)$. From $a \in I$ it follows that `include(`a`)` $\in X'$ as proven in the previous statement. Then, since X' is a model of G^X, we obtain `value(`a`,`c`)` $\in X'$ via the rule in Line 6. This is a contradiction.
   Assume that `-hit(`c`,`j`)` $\in X \setminus X'$ for some $c \in C_t$ and $r_j \in R$. Since `-hit(`c`,`j`)` $\in X$, we get that $r_j \in R$ is not satisfied by $\mathcal{I}$. Therefore, there exists some $v_i \in r_j$ that is not satisfied by $\mathcal{I}$. Consider the case $v_i = \boldsymbol{u}$. Then, `scope(`c`,(`i`,`o_i`))`, `relation(`c`,(`i`,`j`),`$\boldsymbol{u}$`)` $\in X'$ for some $o_i \in O$. Furthermore, since v_i is not satisfied by $\mathcal{I}$, it follows that $o_i \in I$, and `include(`o_i`)` $\in X'$. Since X' is a model of G^X, we obtain `-hit(`c`,`j`)` $\in X'$ via the rule in Lines 8-9. This is a contradiction. The other cases are similar.

[6] We refer to [8] for a reduct definition for choice/weight rules.

Assume that $\texttt{violated}(c) \in X \setminus X'$ for some $c \in C$. Since $\texttt{violated}(c) \in X$, we get that c is not satisfied by $\mathcal{I}$. We proceed by cases.
Case 1. $c \in C_t$. Then, for all $r_j \in R$, we get that r_j is not satisfied by $\mathcal{I}$. By construction of X, we get that $\texttt{-hit}(c,j) \in X$ for all j and by the previous bullet it follows that $\texttt{-hit}(c,j) \in X'$ for all j. Since X' is a model of G^X, we obtain $\texttt{violated}(c) \in X'$ via the rule in Line 15. This is a contradiction.
Case 2. $c \in C_g$. Let $c = \mathit{card}(o, G)$ for some $o \in O$ and group $G \subseteq P$. Since c is not satisfied by $\mathcal{I}$, we get that $o \in I$ and $|I \cap G| \notin \mathit{card}(o, G)$. By the construction of X and the first bullet, it follows that $\texttt{included}(o) \in X'$ and $\texttt{satisfied}(c) \notin X'$. Since X' is a model of G^X, we obtain $\texttt{violated}(c) \in X'$ via the rule in Line 18. This is a contradiction.
Assume that $\texttt{satisfied}(c) \in X \setminus X'$ for some $c \in C_g$. Let $c = \mathit{card}(o, G)$ for some $o \in O$ and group $G \subseteq P$. Since $\texttt{satisfied}(c) \in X$, we get that $|I \cap G| \in \mathit{card}(o, G)$. It follows that $\texttt{include}(o) \in X'$ for $o \in I \cap G$ and $\texttt{include}(o) \notin X'$ for every other $o \in G \setminus I$. Since X' is a model of G^X, we obtain $\texttt{satisfied}(c) \in X'$ via the rule in Line 17. This is a contradiction.
We have shown that X is a model of G and that it is a subset-minimal model of G^X. Therefore, X is a stable model of P. This concludes the proof.

□