Journal of Intelligent Manufacturing manuscript No.
(will be inserted by the editor)

Applications of Non-monotonic Reasoning to Automotive Product Configuration Using Answer Set Programming

Eray Gençay · Peter Schüller · Esra Erdem

Received: date / Accepted: date

This is a post-print. The official final version of this paper is available from the publisher at
https://link.springer.com/article/10.1007/s10845-017-1333-3.

Abstract In automotive industry, validation and maintenance of product configuration data is a complex task. Both orders from the customers and new product line designs from the R&D department are subject to a set of configuration rules to be satisfied. In this work, non-monotonic computational logic, Answer Set Programming in particular, is applied to industrial-scale automotive product configuration problems. This methodology provides basic validation of the product configuration documentation and validation of single product orders, where Reiter style diagnosis provides minimal changes needed to correct an invalid order or a product configuration rule set. In addition, a method for discovering groups of product configuration variables that are strongly related can be obtained by small modification of the basic logic program, and by the usage of cautious and brave reasoning methods. As a result, options that are used in every, or respectively in no configuration, can easily be identified, as well as groups of options that are always used together or not at all. Finally it is possible to single out mandatory and obsolete options, relative to a preselected set of included or excluded options. Experimental results on an industrial dataset show applicability, example results, and computational feasibility with computation times on the order of seconds using a state-of-the-art answer set solver on standard PC hardware.

Keywords Product Configuration · Non-monotonic Logic · Answer Set Programming · Automotive Production · Configuration Management · Mass Customization

E. Gençay was supported by a TUBITAK-2218 Fellowship for Postdoctoral Researchers and by the Turkish-German University Scientific Research Projects Commission under grant no 2015BM0014. P. Schüller has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement no 678867 and by The Scientific and Technological Research Council of Turkey (TUBITAK) under grant 114E777.

E. Gençay
Turkish-German University
Istanbul
E-mail: gencay@tau.edu.tr

P. Schüller
Marmara University
Istanbul
E-mail: peter.schuller@marmara.edu.tr

E. Erdem
Sabanci University
Istanbul
E-mail: esraerdem@sabanciuniv.edu

1 Introduction

In contrast to conventional mass production, where a product is designed once and then produced in large quantities, products are now produced increasingly more in a customized manner according to customer demands. Stanley Davis used the term “mass customization” (mass personalization, personalized mass production), being the first to refer to this development [17]. Tseng and Jiao then defined mass customization as “producing goods and services to meet individual customer’s needs with near mass production efficiency” [18]. A variety of examples can be given for this kind of customizable products that are supplied to the mass market: computers, cars, furniture, cosmetic products, bicycles, modular kitchen systems. Examples for mass customization of services are Internet portals that adapt themselves to the preferences of individual users [81].

In recent years, this tendency has gained momentum with the introduction of the term “Industrie 4.0” (similarly the term “Industrial Internet” in the U.S.) which was originally a vision that brings the concepts of Cyber-Physical Systems [53, 72], Internet of Things and Internet of Services to enable an utterly flexible and individualized mass production (mass customization) at costs comparable to mass production [13]. In order to enable this vision, flexible product configuration management systems are needed, which can support the production system online with results to queries that arise spontaneously as the individualized production orders come in. In this paper it is shown how a rich set of queries about constructible car variants and dependencies between configuration parameters can be processed in feasible time.

Mass customization may seem self-contradictory at first glance, since it involves both terms, mass and customization. The key concept that eliminates the seeming contradiction is the modular production. Modular production becomes possible by standardizing the interfaces of the constituent parts of a product. Thus, by combining varieties of different parts of a product, it becomes theoretically possible to produce as many variations of a product as the cartesian product of the number of varieties of its constituent parts. In practice, there are many production rules that restrict the combination of the parts: thousands of such restrictive rules are used in the production of modern complex products. While there are many of these constraints, the number of potential product variations still remains high. The industrial benchmark used to demonstrate feasibility of the approach introduced in the following is the Renault car benchmark [3] which features about $1.4 \cdot 10^{12}$ variations.

Automotive product configurations are manufactured after they have been ordered as a result of customer choices. Customers can choose between different body types (5-doors, 4-doors, station wagon, SUV, etc.), trim levels, engine types, gear types, exterior colors, interior upholstery types, and additional to these, accessories and options that are offered in a large range. Despite all of these possibilities of choice, not all configurations are constructible. Production constraints, or restrictive production rules, can be legal, marketing related, geometric (space constraints), electrical or related to engineering (part compatibility). For example, a specific air conditioning system could only be installed in combination with a battery of a certain size or with an engine type that produces at least a certain level of horsepower. This rule would be an example of a compatibility constraint.

In an environment where the number of product lines increases, and even the distinction between them becomes blurred while the product customization is increasingly becoming more prevalent, product configuration rules have to be managed and analyzed permanently, because addition or change of a single rule can have far-reaching consequences for the set of feasible configurations:

- the configuration can become inconsistent, which means that there is no set of configuration parameters that satisfies all rules, and no product can be configured/produced according to such a set of rules;
- the configuration can contain possible parameter values that are never used because all possible usages are ruled out by one or another constraint.

As configuration rules concern the same set of parameters, small changes can have big and unforeseen effects, for example the modification of a rule about the seat system can eliminate all configurations for a certain country, without making the configuration inconsistent.

Therefore, after modifying configuration rules, the configuration rule set should be validated against inconsistencies and other potentially undesired effects of the new rule should be analyzed. The space of potential solutions is huge, therefore using explicit enumeration for checking is impossible. Besides validating the rule set, finding significant patterns in the configuration space and predicting potentially harmful effects of modifications of the rule set is a challenging task.

In this work, motivated by these challenges, a novel method for validating and analyzing industrial-scale configuration rule sets is presented. The method is based on Answer Set Programming (ASP) which is a declarative knowledge representation paradigm. The approach presented here is a translation of the Renault benchmark configuration rules into an ASP program that reproduces the semantics of the benchmark configuration space, and a proof of correctness of this transformation (the encoding is also applicable to similar configuration instances).

In case of an *inconsistency*, it is necessary to diagnose the cause of inconsistency in the rule set, so that it can be repaired (made satisfiable). Usually the aim is a repair with minimal changes. To find potential causes of inconsistency, Reiter style diagnosis [73] is realized in ASP rules, based on prior work on diagnosis with ASP [10, 19] and applications of ASP diagnosis in diverse areas such as cognitive robotics [24] and knowledge integration [21].

For identifying *options that are always used* or *options that are never used*, the method described in the following utilizes specific ASP querying modes (brave and cautious reasoning) which allow to obtain from answer sets those parameters that are always true (or false) over all constructible car variants, without enumerating all these variants explicitly. This analysis is useful as follows: once found, parts belonging to obsolete options can be removed from production warehouses to reduce administrative and storage-related costs. Mandatory parts are to be considered as such in product configuration management systems, ERP systems, storage management and production planning to decrease processing time and thus reduce costs.

Moreover, the method presented permits an even more detailed look into the structure of the configuration space: the identification of *find groups of options that are always used together*. If one option in such a group is not used, other options also cannot be part of any constructible car variant. This analysis result can be used to arrange the manufacturing logistics in the production environment more efficiently, and indeed the Renault benchmark contains several examples of such option groups.

Finally it is possible to apply all the above reasoning about the configuration space *relative to a set of preset configuration parameters*. This can be useful to analyze the configuration space only for a particular country or excluding a particular engine type, or combinations thereof.

Experiments with the Renault benchmark on standard PC hardware show that ASP can be applied to real-world industrial-scale automotive configuration problems in a feasible way, which even allows the integration of some of the applications into an interactive user

interface that can give immediate feedback about consistency and other properties of the configuration rule set at hand.

The rest of this paper is organized as follows. Preliminaries of product configuration rule sets, the Renault benchmark, and a short overview of Answer Set Programming is given in Section 2, related work is presented in Section 3, a transformation of the configuration problem into the ASP formalism, and a proof of correctness of this representation is given in Section 4, several applications based on this representation (validating single order configurations, finding rules that make a configuration problem inconsistent, identifying configuration settings that are always true or always false, discovering groups of parts/settings that are always used together (or not at all), and finally all these queries relative to a preselected set of parameter values) are brought out in Section 5, performance tests on the Renault benchmark are reported in Section 6, and conclusions and future directions of this research are described in Section 7.

2 Preliminaries

2.1 Industrial-scale Configuration Benchmark

The configuration rule set used in this work sources from a related work by Amilhastre *et al.* [3]. Originally, it was provided by the French Renault DVI car manufacturer. The rule set documents the configuration of the Renault Megane family of cars. In the file, information like variables, domains of variables and rules for the combinations of the variables are given. Variables represent different configuration options for a car such as the type of engine, the country the car is going to be built for, or the type of air conditioning. The configuration rule set is available in different formats in the CLib Configuration Benchmarks Library [88]. This work uses CP format syntax [47] which is beneficial for presentation reasons.

The product documentation consists of a set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$ where each variable v_i has a domain definition $\mathcal{D}_i = \{d_{i_1}, \ldots, d_{i_m}\}$, moreover there is a set of rules $\mathcal{R} = \{r_1, \ldots, r_k\}$.

The following is an example for a rule in CP format.

```
(( (var3==M5 && var5==FRAN && var6==DG)
|| (var3==M5 && var5==DOTO && var6==DG)
                    .
                    .
                    .
|| (var3==ND1G && var5==CETI && var6==DG)
));
```

A rule comprises a disjunction of conjunctions of variable assignments. Thus, each rule is in disjunctive normal form (DNF), if variable assignments are interpreted as propositions, more precisely as positive literals. Since all rules together must hold, the logic formula to be satisfied is a conjunction of DNF formulae. Conjunctions of variable assignments are called *minterms* in the rest of the paper.

The logical formula for a rule set formalizing the product documentation containing k rules then can be formulated as follows:

$$\Phi = \bigwedge_{1 \leq l \leq k} r_l$$

where rules of the form above are disjunctions of conjunctions of literals of the form $v_i == d_{i_j}$ where $v_i \in \mathcal{V}$ and $d_{i_j} \in D_i$.

The Renault benchmark [3] has the following properties.

- It contains 101 variables with their domain sizes varying from 2 to 43, and 113 rules.
- The number of solutions of the problem is about $1.4 * 10^{12}$ [3].
- The rule set in CP format occupies 23.9 MB with nearly 195 500 lines of code.

2.2 Answer Set Programming

Answer Set Programming (ASP) is a KRR (Knowledge Representation and Reasoning) formalism that aims to provide a balance between expression power, ease of use, and computing performance [37,56,31,36,11,15]. In the following, ASP features that are relevant for this work are presented.

An atom is of the form $p(x_1, \ldots, x_l)$ with $0 \leq l$, and if $l = 0$ the atom is written short as p. An ASP program P consists of a set of rules, where a rule r is of the form

$$\alpha \leftarrow \beta_1, \ldots, \beta_n, \texttt{not}\ \beta_{n+1}, \ldots, \texttt{not}\ \beta_m \tag{1}$$

where α and β_i are atoms, called head and body atoms of r, respectively. $H(r) = \{\alpha\}$ is called the *head* of r. $B^+(r) = \{\beta_1, \ldots, \beta_n\}$ and $B^-(r) = \{\beta_{n+1}, \ldots, \beta_m\}$ are called the positive and the negative *body* of r, respectively. A rule is a *fact* if $m = 0$, in that case the arrow of $a \leftarrow$ is usually omitted, a rule is a *constraint* if $k = 0$. Atoms can contain constants, variables, and function terms, and a program must permit a finite instantiation. Semantics of an ASP program P is defined based on the ground instantiation $\mathrm{grnd}(P)$ and Herbrand base HB_P of P: an interpretation $I \in HB_P$ satisfies a rule r iff $H(r) \cap I \neq \emptyset$ or $B^+(r) \not\subseteq I$ or $B^-(r) \cap I \neq \emptyset$; I is a model of P if it satisfies all rules in P. The reduct P^I of P wrt. I is the set of rules

$$P^I = \{H(r) \leftarrow B^+(r) \mid B^-(r) \cap I = \emptyset\}$$

and an interpretation I is an answer set iff it is a $\subseteq$-minimal model of P^I.

Intuitively, an atom a is true in an answer set iff there is at least one rule with a in the head and a satisfied body, and the truth of the atoms satisfying the body is not due to a positive cycle through that atom a, i.e., ASP prevents self-justifications of atoms in cycles such as $a \leftarrow a$.

ASP allows recursive definitions without becoming undecidable, thus it permits to express transitivity which can not be modeled in first order logic [1,15] as follows.

$$part\_of(X, Y) \leftarrow part\_of(X, Z), part\_of(Z, Y).$$

Atoms can contain variables, such programs are evaluated by first instantiating them with a 'grounder' tool.

Constraints are rules without a head.

$$\leftarrow b_1, \ldots, b_n, \texttt{not}\ b_{n+1}, \ldots, \texttt{not}\ b_m.$$

Intuitively, a set of atoms that satisfies the body of this constraint cannot be an answer set.

A useful feature of the ASP language are choice rules: these rules have heads with choice constructions that contain multiple atoms and a cardinality limit:

$$l\{a_1, \ldots, a_n\}u \leftarrow B.$$

Intuitively, such a rule states, that if the body B is satisfied in an answer set, then at least l and at most u of the atoms in the set $\{a_1, \ldots, a_n\}$ must be true in the answer set. If the body condition B is not given, such a rule directly generates a space of potential solutions.

Additionally to the language features discussed above, Answer Set Programming has been enhanced with a variety of extensions:

- Strong negation allows mixing of $\neg a$ and $\texttt{not}\, a$ for more concise and more intuitive knowledge representation. Strong negation can be rewritten to weak (`not`) negation [38].
- Aggregates are grouping functions similar to those in database language SQL: $\#\mathrm{count}$, $\#\mathrm{sum}$, $\#\mathrm{max}$, and $\#\mathrm{min}$ are examples of group functions that are implemented by Answer Set Programming languages [16].
- Furthermore, optimization criteria can be specified as $\#maximize$ or $\#minimize$ expressions or as *weak constraints* which are constraints that impose a cost on the answer set when they are violated.

Details of syntactic restrictions, syntactic elements, and semantics of ASP are described in the ASP-Core-2 standard [16].

3 Literature Survey

Research in product configuration goes back to the end of the 1970's and is still an active research area. There are many works both in the theory, comprising definitions and models of product configuration [63,66,44,49,82,50,77,93], and in the practice of the field in form of product configurator software systems. According to Sabin *et al.* product configurators can be classified to rule-based, model-based, and case-based approaches [76].

Rule-based configurators have been developed as expert systems that execute rules in form 'IF condition THEN consequence' to generate valid product configurations. Some of the prominent examples of rule-based configurators are R1/XCON, a configurator for computer systems [8,60,85], and the configurator system VT (vertical tranportation) by the Westinghouse Elevator Company to configure elevator systems [59]. As expert systems using production rules in a forward-chaining manner, rule-based systems had no separation between representation of domain knowledge and control strategy [76]. This drawback in the chosen knowledge representation resulted in problems of knowledge base development and maintenance [40,45].

Model-based knowledge representation technologies have emerged as a solution to address these shortcomings of the early rule-based configurators. Model-based configurators introduced a clear separation of the business logic representing the product configuration knowledge and the problem solving knowledge needed to find consistent configurations. One of the major representatives of the model-based approaches are the constraint-based systems. Representing a product configuration problem as a Constraint Satisfaction Problem (CSP) in terms of variables, domains, and constraints, and solving it using search algorithms independent of the product knowledge has proven to be successful for configuring processes with a fixed and pre-defined set of component types [87,63]. Mittal and Falkenhainer [62] proposed an extension to the classical CSP, called dynamic CSP, to represent complex configuration problems, where additional component types are to be included in the configuration according to the decisions in the configuration process. Furthermore, Aldanondo *et al.* proposed the requirements for modeling continuous variables, and numerical constraints over discrete and continuous variables [2]. Xie, Henderson, and Kernahan [99] extended the classical CSP-based modeling with an approach to represent continuous variables.

Case-based reasoning tries to solve a new configuration problem by finding a similar, previously solved problem and adapting its solution to the requirements of the new problem [76]. A relatively recent example to the case-based reasoning is the approach of Tseng *et al.* [95] to product configuration and BOM (bill of material) generation problems.

Apart from these main categories, there have also been completely different approaches to product configuration problems. Kusiak *et al.* apply clustering as a data mining method to identify product configurations that can increase the effectiveness of the enterprise resources analyzing large amounts of sales data available in companies [52,86].

Hadzic *et al.* developed a two-phase solution for interactive product configuration problems [42]. In the first off-line phase, they compiled the solution space of the problem to a Boolean domain using BDDs (Binary Decision Diagrams). Second, if the resulting BDD was small enough, they embedded this representation in an online configurator and utilized for fast, complete, and backtrack-free interactive product configuration. The solution has been extended in a later work with scenarios involving cost functions, which express user preferences using multi-valued decision diagrams (MDDs) [4].

Felfernig *et al.* showed how the techniques from software engineering like object-oriented modeling and knowledge-based systems can be employed to design knowledge-based configuration systems [26,27]. They proposed the application of model-based methods for the validation of knowledge bases and diagnosis of unfeasible customer requirements. The developed system represents the configuration data in XML corresponding to their UML representation. This model is then translated into the representation of a constraint-based off-the-shelf configuration tool (ILOG Configurator) to solve configuration tasks. They also provided formal definitions for configuration problems and their related terms.

Towards the end of the 1990's, the industry began to adopt product configuration technologies. As a result of this, industrial strength configuration environments like Tacton [68], ConfigIT [64], EngCon [5], and ILOG [48,58] have emerged. These solutions were also increasingly integrated into ERP (Enterprise Resource Planning) systems like SAP [41], BAAN [100], and ORACLE.

Product configuration problems from the automotive industry have been investigated before:

Amilhastre *et al.* extended the CSP framework to solve product configuration problems as interactive decision support problems. The Renault benchmark configuration rule set originates from their work [3]. They used an automaton model to represent the set of solutions of the configuration problem as a CSP. Through exploitation of this data structure, they could ensure both consistency maintenance and computation of maximal consistent subsets of user's choices efficiently. Whereas they use CSP methods in their approach, this work uses ASP and defines and realizes additional applications based on special ASP reasoning modes.

Pargamin used a product knowledge compilation approach to develop a configuration engine [70]. Through compilation, the configuration variables are divided into clusters organized as a tree. Inferences are done locally in each cluster, and then the new state of the node is propagated to other nodes to reach a global solution. In a following work, the solution has been extended with non-Boolean variables to enable preference-based optimization [71].

Sinz *et al.* demonstrated that formal methods can be applied to validate real-world automotive configuration data using a SAT-solver [51,80,81]. They also enhanced the performance of their solutions using parallel implementations [79,12]. Their consistency support tool (BIS) works on an existing database containing Boolean constraints about valid configurations and about how they can be transformed into constructible configurations. BIS can check inconsistencies in product documentation using a SAT-solver that is modified for that

kind of problems. While the approach presented in this work is similar to their application, the methods presented here are based on a different formalism and additional applications are brought out here.

Walter *et al.* applied MaxSAT to automotive configuration. Having this approach, they could not only obtain an answer about the satisfiability but also repair suggestions for inconsistent configuration constraints or invalid orders computing the maximum subset of constraints that can be satisfied [98,97,96].

Especially in the last decade, there has been impressive progress in the development of efficient satisfiability solvers (SAT-solvers) that solve formulae written in propositional logic [39]. State-of-the-art SAT-solvers can solve hard problem instances with more than one million variables and several millions of constraints. Thus, it became possible to apply these solvers on large-scale real-world problems. These achievements in the SAT community have greatly influenced the logic programming and KRR communities and paved the way to the ASP paradigm.

ASP has been used in several works to address product configuration problems [7,28, 65,78]. Soininen *et al.* proposed a rule-based language for product configuration using ASP [83,84,92]. In the scope of their work, they realized a prototype implementation for the conversion of rules into normal programs. They used computer system configuration as an example for their prototype implementation. The configurator software WiCoTin [91] and its commercial version VariSales [90] enable a component-oriented representation of configuration problems, and were proven to be applicable in several application domains. The method described in this paper is focused on application scenarios that are beneficial in the analysis of industrial-scale automotive product configuration.

Some examples to the application of ASP to diagnosis are as follows. Syrjänen [89] and Gebser *et al.* [35] used ASP for debugging ASP programs, Eiter *et al.* described a generic diagnosis frontend for the DLV solver [19], Erdem *et al.* used ASP for diagnosing plan failures based on causal reasoning in cognitive robotics [24], and Eiter *et al.* described a method for diagnosing inconsistency in non-monotonic heterogeneous multi-context system [21].

Other than in product configuration and diagnosis, ASP has also been applied successfully to a wide range of areas like robotics [22,25,43,101], decision support systems [67], workforce management [74], intelligent call routing [55], and configuration and reconfiguration of railway safety systems [6]. For a more detailed account of the applications of the ASP, the reader is referred to [23].

Main contribution of the present work can be summarized as follows. Cautious and brave reasoning methods are used in novel applications to discover groups of product configuration variables that are strongly related to each other. In this manner, options that are used in every, or respectively no configuration, and groups of options that are always used together or not at all, can be easily identified, and it is possible to single out mandatory or obsolete options relative to a preselected set of options. Furthermore, as a theoretical contribution, a proof is provided for the proposition, that for every configuration problem that is defined as a conjunction of DNF-rules, an answer set program can be created, answer sets of which correspond 1-1 with the admissible configurations of that configuration problem.

A limitation of the standard ASP approach, as in this paper, is that the continuous parameters can not be represented in the language, so that optimization with real values is not applicable. A solution to this limitation would be using a hybrid solver that combines the Boolean solving capacities of the ASP with techniques for using continuous variables from the area of Constraint Programming or SMT (Satisfiability Modulo Theories). Examples of such systems are Clingcon [69], EZCSP [9], and Dingo [46]. Another limitation of

the ASP is that it lacks non-Herbrand functions, i.e., functions in the mathematical sense, where the value of a term changes with its arguments. ASP uses Herbrand models, therefore distinct terms can never be equal. To address this limitation, ASPMT (Answer Set Programming Modulo Theories) [54], application-specific reasoning interfaces [29], and the HEX framework [20] as a generalization of the standard ASP have been proposed. Such hybrid reasoning techniques are necessary only for application scenarios which contain continuous parameters and constraints on them, therefore they are not considered in this work for the Renault benchmark.

In the diagnosis of inconsistencies, cardinality-minimal diagnosis is performed, as opposed to subset-minimal diagnosis due to Reiter [73] to focus on the domain-specific encoding in automobile configuration. Brewka *et al.* describe the *asprin* framework for adding preferences to ASP programs [14], for example subset-minimality or cardinality-minimality, and report that their encoding computes the former by a more complex encoding, but in shorter time, compared to the latter. Hence the approach presented in this paper can easily be extended without efficiency drawbacks, to subset-minimal diagnoses.

4 Representation of Industrial Configuration in ASP

In Section 2.1, a formalization of the product configuration rule set was provided. In this section, a transformation from this formalization into an ASP program is described, and a proof that this transformation represents the semantics of the original configuration problem is provided.

The transformation is performed as follows.

1. Replace all variable and value comparisons of the form $v_i == d_{i_j}$ with single literals of the form $v_i\_d_{i_j}$.
2. Number all minterms of all rules of the form $(p_1 \wedge p_2 \wedge \cdots \wedge p_{n-1} \wedge p_n)$ with $t = 1, \ldots, z$ and for each minterm create an ASP rule of the form

$$c_t \leftarrow p_1, p_2, ..., p_{n-1}, p_n. \tag{2}$$

3. For minterms $k = l, \ldots, m$ of a single rule, define the following ASP constraint.

$$\leftarrow \texttt{not}\, c_l, \texttt{not}\, c_{l+1}, \ldots, \texttt{not}\, c_{m-1}, \texttt{not}\, c_m. \tag{3}$$

4. For each variable $v_i \in \mathcal{V}$ with domain $\{d_{i_1}, \ldots, d_{i_u}\}$, define an ASP choice rule as follows.

$$1\{v_i\_d_{i_1}; v_i\_d_{i_2}; \ldots; v_i\_d_{i_u}\}1 \leftarrow . \tag{4}$$

Intuitively, in step 2 the truth value of minterm t is represented by atom c_t. Step 3 ensures that no solution violates a rule, and step 4 defines potential solutions as an assignment of exactly one domain element for each variable v_i.

Example 1 Parts of the rules of the Renault product documentation in CP format is as follows.

$$\vdots$$

```
(( (var1==S64 && var3==M9 && var81==BVM5
    && var94==EU93 && var95==CRIT1870CC
    && var96==CRIT095CV && var99==JC5
    && var100==CRIT784 && var101==F8Q)
|| (var1==S64 && var3==M9
    && var81==BVM5 && var94==EU93
    && var95==CRIT1870CC && var96==CRIT095CV
    && var99==JC5 && var100==CRIT786
    && var101==F8Q)
                                   ⋮
|| (var1==V25 && var3==MJ && var81==BVM5
    && var94==EU96 && var95==CRIT1870CC
    && var96==CRIT070CV && var99==JB1
    && var100==CRIT788 && var101==F8Q)
));
                                   ⋮
```

Steps 1 and 2 yield the following.

$$
\begin{aligned}
c_{208} \leftarrow & \mathit{var1\_S64}, \mathit{var3\_M9}, \mathit{var81\_BVM5}, \\
& \mathit{var94\_EU93}, \mathit{var95\_CRIT1870CC}, \\
& \mathit{var96\_CRIT095CV}, \mathit{var99\_JC5}, \\
& \mathit{var100\_CRIT784}, \mathit{var101\_F8Q}. \\
c_{209} \leftarrow & \mathit{var1\_S64}, \mathit{var3\_M9}, \mathit{var81\_BVM5}, \\
& \mathit{var94\_EU93}, \mathit{var95\_CRIT1870CC}, \\
& \mathit{var96\_CRIT095CV}, \mathit{var99\_JC5}, \\
& \mathit{var100\_CRIT786}, \mathit{var101\_F8Q}. \\
& \vdots \\
c_{371} \leftarrow & \mathit{var1\_V25}, \mathit{var3\_MJ}, \mathit{var81\_BVM5}, \\
& \mathit{var94\_EU96}, \mathit{var95\_CRIT1870CC}, \\
& \mathit{var96\_CRIT070CV}, \mathit{var99\_JB1}, \\
& \mathit{var100\_CRIT788}, \mathit{var101\_F8Q}.
\end{aligned}
$$

The first rule in the CP format is converted into the ASP rules with rule heads from c_{208} to c_{371}. (In this example, 207 minterms were produced before converting this rule.)

Step 3 introduce the following ASP constraints which specify that at least one of the variables denoting the minterms of a CP rule is true.

$$\leftarrow \texttt{not}\, c_{208}, \texttt{not}\, c_{209}, \ldots, \texttt{not}\, c_{371}.$$

After this step, there are as many constraints of this kind as there are rules in CP format.

Step 4 creates the following choice rules which intuitively choose exactly one domain element for each variable.

$$
\begin{aligned}
1\{&var1\_B64; var1\_D64; var1\_E64;\\
&var1\_F64; var1\_J64; var1\_K25;\\
&var1\_L64; var1\_S64; var1\_V25\}1.\\
1\{&var2\_E0; var2\_E1; var2\_E2;\\
&var2\_E3; var2\_E5\}1.\\
&\vdots\\
1\{&var101\_D7F; var101\_E7J; var101\_F3R;\\
&var101\_F4R; var101\_F7R; var101\_F8Q;\\
&var101\_F9Q; var101\_K4J; var101\_K4M;\\
&var101\_K7M\}1.
\end{aligned}
$$

Concrete examples for answer sets are given in Section 5.

Below the correct representation of the configuration problem by the above rules is shown.

The formal definition for configuration problems and resulting configurations that are admissible with respect to a set of configuration rules is as follows. (Note that these were described intuitively in Section 2.)

Definition 1 (Configuration Problem): A configuration problem is a triplet $\mathcal{C} := (\mathcal{V}, \mathcal{D}, \mathcal{R})$, where $\mathcal{V} := \{v_1, v_2, \ldots, v_n\}$ is the set of variables, $\mathcal{D} := \{D_1, D_2, \ldots, D_n\}$ is the set of the variable domains, and $\mathcal{R}$ is a propositional logic formula using only Boolean variables $p_{ij} \in \mathcal{P}$ representing sets of equivalences $v_i == d_{i_j}$, where $v_i \in \mathcal{V}$ and $d_{i_j} \in D_i$.

Definition 2 (Configuration Problem with DNF rules): A configuration problem with DNF rules is a triplet $\mathcal{C}_{DNF} := (\mathcal{V}, \mathcal{D}, \mathcal{R}_{DNF})$, where $\mathcal{V} := \{v_1, v_2, \ldots, v_n\}$ is the set of variables, $\mathcal{D} := \{D_1, D_2, \ldots, D_n\}$ is the set of the variable domains, and $\mathcal{R}_{DNF}$ is in the following form:

$$\mathcal{R}_{DNF} := \bigwedge_{1 \leq l \leq k} r_l$$

r_l is a formula in disjunctive normal form, i.e., r_l is a set of sets of equivalences $v_i == d_{i_j}$, where $v_i \in \mathcal{V}$ and $d_{i_j} \in D_i$.

Definition 3 (Admissible Configuration): Given a configuration problem $\mathcal{C}$, an admissible configuration C_{adm} is an assignment of $p_{ij} \in \mathcal{P}$ to $\{0, 1\}$ that satisfies $\mathcal{R}$.

Proposition 1 *Given a configuration problem $\mathcal{C}_{DNF}$, the answer sets $AS(P(\mathcal{C}_{DNF}))$ of program $P(\mathcal{C}_{DNF})$ correspond 1-1 with admissible configurations of $\mathcal{C}_{DNF}$.*

For the proof of correctness, a propositional theory is created from the rules in CP format using the Tseitin transformation [94], moreover the ASP formulation is also converted into a propositional theory using the transformation proposed by Lin and Zhao [57]. At the end, it is shown that these two propositional theories are equivalent to each other.

Proof (Proposition 1) The rules in CP format can be seen as a conjunction of DNF formulae. Tseitin's transformation yields a propositional theory as follows: for every minterm in Step 2 above, new variables c_t are added, where t is a running number from 1 to the number of all minterms in all rules. Thus, a unique variable is obtained for every minterm.

$$c_t \equiv (p_1 \wedge p_2 \wedge \cdots \wedge p_{n-1} \wedge p_n)$$

According to the Tseitin transformation, the following clause with all the variables that represent the minterms of a rule, which are numbered with indices $k = l, \ldots, m$, is created for each rule:

$$(c_l \vee c_{l+1} \vee \cdots \vee c_{m-1} \vee c_m)$$

Finally, since each variable in CP format can only take a single value, the following equivalences are included in the formula:

$$\begin{aligned}
v_i_d_{i_1} &\equiv \neg v_i_d_{i_2} \wedge \cdots \wedge \neg v_i_d_{i_u} \\
v_i_d_{i_2} &\equiv \neg v_i_d_{i_1} \wedge \cdots \wedge \neg v_i_d_{i_u} \\
v_i_d_{i_3} &\equiv \neg v_i_d_{i_1} \wedge \cdots \wedge \neg v_i_d_{i_u} \\
&\vdots \\
v_i_d_{i_u} &\equiv \neg v_i_d_{i_1} \wedge \cdots \wedge \neg v_i_d_{i_{u-1}}
\end{aligned}$$

Note that, for readability, the literals that were represented as variables $v_i\_d_{i_j}$ are here replaced with literals $p_1, p_2, \ldots, p_{n-1}, p_n$, equal to those literals used in Step 2 above.

In this way, the conjunctions of formulae created above represent the rules that were originally in CP format.

Conversion of the ASP formulation. In the second stage of the proof, the ASP formulation is converted to propositional logic using the approach of Lin and Zhao [57]. Note that the ASP formulation does not include any loops. As mentioned above unique variables were created for minterms.

In the second step of the CP-ASP transformation, the following ASP rules were created:

$$c_t \leftarrow p_1, p_2, \ldots, p_{n-1}, p_n.$$

According to Lin and Zhao, these are converted into the following form.

$$c_t \equiv (p_1 \wedge p_2 \wedge \cdots \wedge p_{n-1} \wedge p_n)$$

In the third step of the CP-ASP transformation, the following constraints were created:

$$\leftarrow \texttt{not}\, c_l, \texttt{not}\, c_{l+1}, \ldots, \texttt{not}\, c_{m-1}, \texttt{not}\, c_m.$$

These constraints will be converted according to Lin and Zhao into the form below.

$$\neg(\neg c_l \wedge \neg c_{l+1} \wedge \cdots \wedge \neg c_{m-1} \wedge \neg c_m)$$

Using the De Morgan law, this is equivalent to the following.

$$(c_l \vee c_{l+1} \vee \cdots \vee c_{m-1} \vee c_m)$$

In the forth step of the transformation, the following choice rules are added to assure that each CP variable can only have a single value:

$$1\{v_i\_d_{i_1}; v_i\_d_{i_2}; \ldots; v_i\_d_{i_u}\}1.$$

This constraint can be alternatively represented with the following series of rules according to [16]:

$$\begin{aligned}
v_i_d_{i_1} &\leftarrow \texttt{not}\ v_i_d_{i_2}, \ldots, \texttt{not}\ v_i_d_{i_u} \\
v_i_d_{i_2} &\leftarrow \texttt{not}\ v_i_d_{i_1}, \ldots, \texttt{not}\ v_i_d_{i_u} \\
v_i_d_{i_3} &\leftarrow \texttt{not}\ v_i_d_{i_1}, \ldots, \texttt{not}\ v_i_d_{i_u} \\
&\vdots \\
v_i_d_{i_u} &\leftarrow \texttt{not}\ v_i_d_{i_1}, \ldots, \texttt{not}\ v_i_d_{i_{u-1}}
\end{aligned}$$

In this manner, every literal in the program has been used in the rule heads. Since literals that represent the minterms (which start with the prefix c) are never used in the rule bodies of these rules, still no loops are present in the program after this step of the transformation.

The rules representing the constraints above can be converted to the following formulae in propositional logic according to Lin and Zhao:

$$\begin{aligned}
v_i_d_{i_1} &\equiv \neg v_i_d_{i_2} \wedge \cdots \wedge \neg v_i_d_{i_u} \\
v_i_d_{i_2} &\equiv \neg v_i_d_{i_1} \wedge \cdots \wedge \neg v_i_d_{i_u} \\
v_i_d_{i_3} &\equiv \neg v_i_d_{i_1} \wedge \cdots \wedge \neg v_i_d_{i_u} \\
&\vdots \\
v_i_d_{i_u} &\equiv \neg v_i_d_{i_1} \wedge \cdots \wedge \neg v_i_d_{i_{u-1}}
\end{aligned}$$

When the formulae that were created using Tseitin transformation and according to the approach of Lin and Zhao are compared, one will see that they are equivalent. ■

5 Applications

Other than the basic applications like the validation of a given production order or of the complete product documentation, e.g. the validation of the configuration rule base, there are also a series of applications that are relevant to the production planning and warehouse management. In the following, details about applications and their implementation in ASP are given.

The primary purpose of configuration constraints is, to determine what a feasible configuration is. Hence validating a given configuration against a set of rules is the first possible usage for the ASP encoding of the configuration instance.

To validate a particular order configuration, the contents of this order are represented as rules without bodies, namely as facts. Subsequently, this rule set is added to the answer set program that represents the product documentation. Enumeration of answer sets of this new program yields valid configurations or no answers if the configuration is inconsistent.

An order can be formalized with a set of literals denoting the key value pairs of the form $p_{ij} = v_i\_d_{i_j}$:

$$o = \{p_{ij} | i \in \{1, 2, \ldots, |\mathcal{V}|\}, j \in \{1, 2, \ldots, |D_i|\}\}$$

The ASP program representing the product documentation can then be complemented by adding each of these literals in the order set as single facts. The formalization of an order with a number of z facts can be represented as below:

$p_1. p_2. \ \ldots \ p_{z-1}. p_z.$

For example, all Diesel car models for Switzerland have the configuration

```
var5=SUIS var4=DIESEL
```

which is represented in the ASP formulation by the following atoms.

```
var5_SUIS var4_DIESEL
```

The ASP program comprising the transformed product documentation and such facts can be checked now for satisfiability. Note that not every variable in $\mathcal{V}$ has to be set through a literal in the order. An order can leave some of the variables unset. In this case, the answer set computed by the solver can be interpreted as one possible feasible solution with assignments for unset variables.

The applications described in the following go beyond validating a single configuration: the following encodings make analyses of all potential configurations at once using the power of modern computational logic techniques and ASP solvers.

5.1 Diagnosis (APP1)

In the case of an incorrect car order configuration or an invalid product documentation, it is usually interesting to know which rules or rule parts have caused the test to fail. Since there are multiple ways to make an invalid configuration valid again, it is also desirable to find the minimum number of changes in variable settings in order to do so. To achieve this, the answer set program is complemented with additional variables denoting abnormality according to the theory of diagnosis from the first principles by Reiter.

In Step 3 of the transformation from CP format to ASP, constraints for the minterms of each rule were defined. Additionally to the negative literals denoting the minterms, here a variable to each constraint to symbolize abnormality of the constraint.

$$\leftarrow \texttt{not}\, c_l, \texttt{not}\, c_{l+1}, \ldots, \texttt{not}\, c_{m-1}, \texttt{not}\, c_m, ab(i,1).$$

Applied to the example above, some of the constraints generated this way are as follows.

$$\begin{aligned} &\leftarrow \texttt{not}\, c_{208}, \texttt{not}\, c_{209}, \ldots, \texttt{not}\, c_{371}, ab(1,1). \\ &\leftarrow \texttt{not}\, c_{376}, \texttt{not}\, c_{377}, \ldots, \texttt{not}\, c_{1425}, ab(2,1). \\ &\vdots \end{aligned}$$

The predicate ab has two arguments. The first argument holds the running number for the rule that contains the minterms, denoted by the negative literals $\texttt{not}\, c_l, \cdots, \texttt{not}\, c_m$. The second argument gives the weight of the abnormality. Uniform weight of cost 1 was chosen for all rules, but alternatively the number of different variables in each minterm or a priority value for the importance of a rule for the production could be chosen as the value of the second argument.

In ASP, constraints can only be defined to confine atoms that are populated by variable binding as it is specified in the program. In order to be able to use the ab predicate in the constraints, the following ASP rule generates all possible combinations of abnormalities.

$$\{ab(1,1); ab(2,1); \ldots; ab(n,1)\}.$$

Since the Renault benchmark contains 113 rules, the guess obtained this way is as follows:

$$\{ab(1,1); ab(2,1); \ldots; ab(113,1)\}.$$

Having no limiting numbers on each side of the rule, any subset of the literals $ab(1,1)$, $\ldots$, $ab(n,1)$ can be guessed as true. This modification ensures that the new answer set program is always satisfiable, since the constraints that fired previously can now be deactivated thanks to their $ab(\cdot,\cdot)$ literals.

As a result, a number of abnormality literals appears in the answer sets instead of the mere output `UNSATISFIABLE` for inconsistent configuration rule sets. As the next step, a minimization statement eliminates all but the answer sets with the minimal number of violated rules.

Many answer set solvers allows the use of $\#maximize$ and $\#minimize$ expressions and the assignment of weights to literals. For each answer set, a sum is going to be calculated adding up the weights of its literals. The answer set that provides the optimum can be then found according to the answer set weights and the choice between the expressions $\#maximize$ and $\#minimize$. The following statement restricts solutions to minimal ones.

$$\#minimize\{C@1, X : ab(X,C)\}.$$

Here, the argument C denotes the cost to minimize, i.e. the sum of weights of all abnormality literals. The number 1 indicates the optimization level of the rule which is the same for all abnormalities (it is possible to prioritize certain abnormalities this way).

Using this optimization constraint, the cost of the optimal answer sets corresponds with the number of rules violated, since cost 1 was specified as the weight of each rule.

Since the Renault benchmark is consistent, applying the ASP formulation yields the empty set. Accordingly no constraints need to be deactivated to obtain a consistent product documentation. A case yielding more interesting results is the usage of this formulation relative to a preselected set of options which is discussed in Section 5.4.

5.2 Options that are used in every/no configuration (APP2)

Many ASP solvers can compute brave or cautious consequences of a logic program. Brave consequences of a logic program are the atoms that are true in some answer sets, whereas the cautious consequences are the atoms that are true in every answer set of that program. In other words, the intersection of all answer sets of a program represent the cautious consequences, whereas the union of all answer sets of it represent the brave consequences.

Given these definitions, it is obvious that the cautious consequences of the logic program represent exactly the set of options that are used in every car variant. This method identifies the following seven options to be present in every constructible car variant in the Renault benchmark:

```
var71=FSTPO var50=SASURV var17=SPRTEL var24=SSARCE
var10=SANCOA var8=CPLN var36=PTCAV
```

The concrete output of the Clingo ASP solver for this application is as follows.

```
var71_FSTPO var50_SASURV var17_SPRTEL var24_SSARCE
var10_SANCOA var8_CPLN var36_PTCAV
```

There are different ways to find the options that are not used in any car variant. One of them would be computing the brave consequences of the program and then finding out the difference of the set of all atoms representing the configuration options to them. The resulting set would be exactly the options that are not used in any constructible car variant. Using Clingo and the logic program as it is (see Section 6 for details on the experimental setup used), it took 2 hours 36 minutes to compute the brave consequences. The result included not only the atoms representing the options but also the atoms that were standing for the minterms of each rule in the CP format. Thus, the result set is confined to the first using the $\#show$ declarative. The $\#show$ declarative allows to selectively include the atoms of a certain predicate in the resulting answer sets. After complementing the logic program with $\#show$ declaratives for atoms representing the options, it takes Clingo around 10 seconds to compute the brave consequences of the program.

Another way to find these options is first to define new variables for the (default) negations of every option variable, and then to compute the cautious consequences of the program that is now complemented with the rules for variable negations. An example definition for a variable negation is given below:

$$nvar1\_B64 \leftarrow \texttt{not}\ var1\_B64\,.$$

Using these negations and cautious reasoning, the concrete output of the Clingo ASP solver is as follows.

```
nvar8_CPLG nvar10_temp nvar17_temp nvar24_temp
nvar36_temp nvar50_temp nvar55_CRIT2RHENF
nvar71_temp nvar80_CRIT4X25KI nvar100_CRIT731
```

From this, the following options that are never used in any constructible car variant can be obtained:

```
var8=CPLG var10=temp var17=temp var24=temp
var36=temp var50=temp var55=CRIT2RHENF
var71=temp var80=CRIT4X25KI var100=CRIT731
```

Many of these options seem to be false positives, as the word “temp” indicates that these option variations represent some kind of temporary place holders.

The discovery of options that are not used in any product configuration will enable savings in production warehouse management, inventory management, production management, and labor costs related to the parts affected by the identified options, since these parts can now be released from the production warehouse, for example to supply them as spare parts. On the other hand, when the options were really options that are needed for potential product configurations (e.g. in case of false positives), their discovery will prevent possible loss of opportunity, when customers were interested in products including these options.

5.3 Groups of options that are used always together (APP3)

Based on the encoding above, additional information about the configuration space can be obtained: pairs and groups of options that are always used together, or are always absent together from configurations.

To obtain such information, the previous formulation is extended by the following rules.

$$pair_{v_i\_d_{i_j},v_k\_d_{k_l}} \leftarrow v_i\_d_{i_j}, v_k\_d_{k_l}.$$
$$pair_{v_i\_d_{i_j},v_k\_d_{k_l}} \leftarrow \texttt{not}\ v_i\_d_{i_j}, \texttt{not}\ v_k\_d_{k_l}.$$
$$\text{for } i,k \in \{1,\ldots,|\mathcal{V}|\},\ j \in \{1,\ldots,|D_i|\}, \text{ and } l \in \{1,\ldots,|D_k|\}$$

This defines a new atom $pair_{v_i\_d_{i_j},v_k\_d_{k_l}}$ that is true in two cases: if variable v_i has value d_{i_j} and variable v_k has value d_{k_l}, and if both variables do not have these values. Essentially this new atom is true if the variable/value combinations have equivalent truth values (both true or both false) in a solution.

By performing cautious reasoning on the resulting ASP program all pairs of variable/-value combinations that are always 'synchronized' in configurations are obtained: either both variables have the indicated values, or both do not have these values. These observations, translated back into the domain of car manufacturing, provide sets of components (or other variable settings) that are always either used together in a car, or not at all.

From pairs of components that are used together, groups of such components can be computed: if component A and B are always used together, and components B and C are always used together, A and C are also always used together (mathematically the components form an equivalence class).

Performing this cautious computation yields 29 equivalence relations of components that are always used together or not at all. Two of these groups are trivial groups: the group of items that is used in no configuration, and the group of items that is used in all configurations.

Examples for nontrivial groups of items that are always used together or not at all, are the following three groups.

```
{var3=ME, var100=CRIT764, var75=MOCY02}
{var5=SUIS, var82=NMAS04}
{var5=ALLE, var82=NMAS03, var72=PARALL}
```

As an example, the first group is indicated in the cautious answer set by the following atoms.

```
pair_var3_ME_var75_MOCY02   pair_var3_ME_var100_CRIT764
pair_var75_MOCY02_var3_ME   pair_var75_MOCY02_var100_CRIT764
pair_var100_CRIT764_var3_ME pair_var100_CRIT764_var75_MOCY02
```

In practice, such information can be valuable for arranging storage space to minimize transport distances and logistic effort during assembly. Another use of this information would be the consideration of the options in such a group for the unification to a single option, since they are provably used always together. This way, the efficiency of the configuration process would be increased.

5.4 Queries relative to a preselected set of options (APP4)

By restricting the configuration space, for example enforcing usage of a certain option, or forbidding usage of another option, application scenarios APP1–APP3 can be applied relative to a hypothetical change of the configuration space.

For example, it can be computed which options are mandatory or obsolete (APP1) under the assumption that a certain option should be used, or is no longer available.

Such reasoning can be achieved by adding constraints to the ASP encodings of the above applications. These constraints require certain variable/value combinations or forbid them.

Table 1 Results for the Validation of the Product Documentation.

Application	APP1	APP2	APP2	APP2	APP3	APP4	APP4
Reasoning Mode	Diagnosis	Brave	Cautious	+negation	Cautious pairs	+var5_SUIS	−var4_DIESEL
Time (sec)	0.4	10.2	5.4	11.4	55.9	10.4	6.4
Memory (GB)	1.5	1.5	1.5	1.5	2.0	2.0	2.0

To require that variable $i \in \{1, \ldots, |\mathcal{V}|\}$ obtains value $j \in \{1, \ldots, |D_i|\}$, the only required change to the respective application is to add the following constraint.

$$\leftarrow \texttt{not}\ v_i\_d_{i_j}.$$

Intuitively this declares solutions where $v_i\_d_{i_j}$ is false as invalid, or in other words all answer sets satisfy the condition that they represent $v_i = d_{i_j}$.

To require that variable v_i does not have value d_{i_j}, it suffices to add the following constraint.

$$\leftarrow v_i\_d_{i_j}.$$

Moreover multiple variable/value combinations can be restricted like this at the same time by adding the respective constraints.

In the Renault rule set, if constraint $\leftarrow \texttt{not}\ var5\_SUIS$ is added, this requires the variation $var5\_SUIS$ to be true. Cautious querying then yields a set of equivalence relations different from the original equivalence relations, i.e., the set of options used together or not at all in all potential configurations for car models in Switzerland.

In the result of this query 'relative to Switzerland' 18 of the original equivalence classes are obtained, for example {`var3=ME, var100=CRIT764, var75=MOCY02`} is still part of the solution. In addition, the following previously non-existing equivalence relation become apparent.

```
{var1=E64, var42=SSPHAN}
```

Moreover all other country-specific equivalences (e.g., `var5=ALLE`, `var82=NMAS03`, and `var72=PARALL`) become part of the 'always false' set of parameters.

In total, when restricting to `var5_SUIS`, 17 configuration parameters become always true and 75 parameters become always false relative to the unrestricted set of product variations.

The configuration space can be restricted further, for example by ruling out Diesel engine models, using the following constraint.

$$\leftarrow var4\_DIESEL.$$

As a result, further novel synchronized configuration parameters become apparent. In the above example, the following new equivalence classes can be retrieved from the answer set (a concrete example for atoms of such an answer set is given in Section 5.3).

```
{var3=MM, var100=CRIT622}
{var3=MT, var100=CRIT624, var96=CRIT070CV}
```

These are two equivalence classes where it becomes visible that choice of `var3` completely determines choice of `var100` and this is not the case in Diesel or non-Switzerland variations.

Moreover, an additional 41 configuration parameters become permanently false and two parameters (`var26=SANCL` and `var4_ESS`) become permanently true in Switzerland without Diesel.

This ASP reasoning mode together with pair-finding rules provides useful information for optimizing configuration and manufacture, and using the method described here, this information can be obtained from the product configuration rule set in a non-complicated way. For example, through an analysis of the remaining options based on the sales market (i.e. the country option is preselected), it can be examined if every product line the producer offers can also be produced for a certain country.

6 Results and Discussion

Based on the above encodings and based on the industrial-scale Renault benchmark [3] experiments were conducted to verify the feasibility of the approach. A laptop system with Intel®Core™i7-2760QM processor with 2.4 GHz and 8 GB main memory was used as hardware, for solving Answer Set Programs the Clingo 4.5.0[1] was used. Clingo is a compact program that combines the instantiator program Gringo [32] and the solver Clasp [33,30] and is currently the best-performing tool in the majority of benchmarks [34]. The default configuration of Clingo was used, which means that Clingo performs automatic selection of the (predicted) best solver parameter portfolio. Manual tuning yielded slightly different and mostly worse performance, so these experiments are not reported. Clingo was used in single-threaded mode.

The results of experiments with all above mentioned application scenarios is summarized in Table 1. In particular the solving time and memory consumption is shown for all experiments that are performed. Columns of the table have been described in the respective application section (Sections 5.1–5.4).

The results are encouraging, even though the underlying problems are NP-complete: most application scenarios require a time around or below 10 seconds for reasoning. As a consequence such reasoning can be integrated into interactive tools for editing product documentations. In particular querying with respect to a selection of variations (APP4) requires a moderate amount of time.

Memory consumption is similar among APP1–APP2 (1.5 GB) while a bit more memory is required in APP3–APP4. This can be explained by the pair-detecting rules in the program which are only required in APP3–APP4.

In detail, column APP1 of Table 1 shows resources required for computing a diagnosis over the product documentation, i.e., finding which rules are causing inconsistency of the documentation.[2] Column APP2/brave pertains to identifying options that are used in at least one constructible car variant, APP2/cautious is about computing those options that are used in every constructible car variant, and APP2/+negation shows results for cautious querying with negation, which means to find options that are absent in all constructible car variants. The most time-consuming computation is clearly application APP3 which is the search for pairs of options that are related in the sense that in all constructible cars, either both options are present, or both are absent. For selective querying APP4 the computational resources for computing the examples shown in Section 5.4 are given: column APP4/`+var5_SUIS` is the search for pairs of options that are related (as in APP3) in all constructible car variants

[1] `http://potassco.sourceforge.net/`

[2] As the benchmark is consistent, the result is empty.

that are intended for Switzerland, and APP4/`-var4_DIESEL` additionally constraints this search to car variants that are not Diesel cars.

With APP4, the more the configuration space is restricted by requiring or forbidding certain options, the faster reasoning becomes. This easier solving with more constrained space is a well-known phenomenon in ASP, SAT, and constraint solving [75].

7 Conclusion and Future Directions

In this work, Answer Set Programming was applied to an industrial-scale automotive product configuration rule set. The configuration rule set was formalized as a propositional logic formula and then transformed into an ASP encoding. Using this encoding, four applications within automotive product configuration were defined. The investigation of that kind of questions has the potential to increase productivity and decrease costs by minimizing storage and transportation distances in logistics of producing highly configurable products. For example, the discovery of options that are not used in any product configuration will enable savings in management, storage, inventory, and labor costs, or when the options were really options that ought to be used (in case of false positives), their discovery will have positive consequences like the prevention of possible loss of opportunity.

In the first application (APP1), the theory of diagnosis from first principles by Reiter was applied to give explanations to possible inconsistencies in the configuration rule set. Combined with the minimization mechanism of ASP more relevant repair suggestions can be obtained. In the second application (APP2), the mandatory or obsolete options that appear in every (or respectively no) car variants are computed. The third application (APP3) brings out which pairs or groups of options must be used or must be absent together in all constructible car variants. Finally, the last application (APP4) computes mandatory or obsolete options under the assumption that a certain set of options should be used or no longer available. Empirical experiments showed that ASP as an approach of non-monotonic reasoning can be used in these applications in a feasible and efficient way.

The presented work in this paper could be extended in two different directions. Methods of probabilistic inductive logic programming [61] can be used to determine probability distribution over answer sets of a given answer set program. Through analysis of this information, one can automatically discover product line candidates from a set of product configuration rules. Another future direction would be to extend the approach in this paper with techniques allowing the use of non-Boolean variables with ASP. Hybrid solvers like Clingcon, that combine ASP with CSP are already available. Using such capabilities, the analysis of the configuration options can be extended with optimization based on criteria with real values.

References

1. Aho, A.V., Ullman, J.D.: Universality of data retrieval languages. In: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pp. 110–119. ACM (1979)
2. Aldanondo, M., Hadj-Hamou, K., Moynard, G., Lamothe, J.: Mass customization and configuration: Requirement analysis and constraint based modeling propositions. Integrated Computer-Aided Engineering **10**(2), 177–189 (2003)
3. Amilhastre, J., Fargier, H., Marquis, P.: Consistency restoration and explanations in dynamic CSPs–application to configuration. Artificial Intelligence **135**(1), 199–234 (2002)
4. Andersen, H.R., Hadzic, T., Pisinger, D.: Interactive cost configuration over decision diagrams. Journal of Artificial Intelligence Research **37**(1), 99–140 (2010)

5. Arlt, V., Günter, A., Hollmann, O., Wagner, T., Hotz, L.: EngCon - Engineering & Configuration. In: Workshop on Configuration at AAAI (1999)
6. Aschinger, M., Drescher, C., Friedrich, G., Gottlob, G., Jeavons, P., Ryabokon, A., Thorstensen, E.: Optimization methods for the partner units problem. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 4–19. Springer (2011)
7. Aschinger, M., Drescher, C., Gottlob, G., Vollmer, H.: LoCo-A logic for configuration problems. ACM Transactions on Computational Logic (TOCL) **15**(3), 20 (2014)
8. Bachant, J., McDermott, J.: R1 revisited: Four years in the trenches. AI magazine **5**(3), 21 (1984)
9. Balduccini, M.: Industrial-size scheduling with ASP+CP. In: International Conference on Logic Programming and Nonmonotonic Reasoning, pp. 284–296. Springer (2011)
10. Balduccini, M., Gelfond, M.: Diagnostic reasoning with A-Prolog. Theory and Practice of Logic Programming **3**(4), 425–461 (2003)
11. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press (2004)
12. Blochinger, W., Sinz, C., Küchlin, W.: Parallel consistency checking of automotive product data. In: Proc. of the International Parallel Computing Conference, pp. 50–57 (2001)
13. BMBF: Zukunftsbild Industrie 4.0. Broschüre 99999-1679, Bundesministerium für Bildung und Forschung (BMBF), Bonn (2013)
14. Brewka, G., Delgrande, J., Romero, J., Schaub, T.: asprin: Customizing Answer Set Preferences without a Headache. In: AAAI Conference on Artificial Intelligence, pp. 1467–1474 (2015)
15. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
16. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2: Input language format (2012)
17. Davis, S.M.: Future perfect. Addison-Wesley Reading, MA (1987)
18. Du, X., Jiao, J., Tseng, M.M.: Architecture of product family: fundamentals and methodology. Concurrent Engineering **9**(4), 309–325 (2001)
19. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: The Diagnosis Frontend of the dlv System. AI Communications **12**(1-2), 99–111 (1999)
20. Eiter, T., Fink, M., Ianni, G., Krennwallner, T., Redl, C., Schüller, P.: A model building framework for Answer Set Programming with external computations. Theory and Practice of Logic Programming **16**(04), 418–464 (2016). DOI 10.1017/S1471068415000113
21. Eiter, T., Fink, M., Schüller, P., Weinzierl, A.: Finding Explanations of Inconsistency in Multi-Context Systems. Artificial Intelligence **216**, 233–274 (2014)
22. Erdem, E., Aker, E., Patoglu, V.: Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. Intelligent Service Robotics **5**(4), 275–291 (2012)
23. Erdem, E., Gelfond, M., Leone, N.: Applications of asp. AI Magazine **37**(3) (2016)
24. Erdem, E., Haspalamutgil, K., Patoglu, V., Uras, T.: Causality-Based Planning and Diagnostic Reasoning for Cognitive Factories. In: IEEE Conference on Emerging Technologies & Factory Automation (2012)
25. Erdem, E., Patoglu, V., Saribatur, Z.G., Schüller, P., Uras, T.: Finding optimal plans for multiple teams of robots through a mediator: A logic-based approach. Theory and Practice of Logic Programming **13**(4-5), 831–846 (2013)
26. Felfernig, A., Friedrich, G., Jannach, D.: Conceptual modeling for configuration of mass-customizable products. Artificial Intelligence in Engineering **15**(2), 165–176 (2001)
27. Felfernig, A., Friedrich, G.E., Jannach, D., Stumptner, M.: Exploiting structural abstractions for consistency based diagnosis of large configurator knowledge bases. In: Proc. International Configuration Workshop at ECAI, pp. 23–28 (2000)
28. Friedrich, G., Ryabokon, A., Falkner, A.A., Haselböck, A., Schenner, G., Schreiner, H.: (Re)configuration using Answer Set Programming. In: Second Workshop on Logics for Component Configuration (LoCoCo 2011), pp. 26–35 (2011)
29. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory Solving made easy with Clingo 5. In: International Conference on Logic Programming: Technical Communications. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). DOI 10.4230/OASIcs.ICLP.2016.2
30. Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., Schaub, T.: Progress in clasp series 3. In: Logic Programming and Nonmonotonic Reasoning, pp. 368–383. Springer (2015)
31. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer set solving in practice. Morgan & Claypool Publishers (2012)
32. Gebser, M., Kaminski, R., König, A., Schaub, T.: Advances in gringo series 3. In: Logic Programming and Nonmonotonic Reasoning, pp. 345–351. Springer (2011)

33. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: clasp: A conflict-driven answer set solver. In: Logic Programming and Nonmonotonic Reasoning, pp. 260–265. Springer (2007)
34. Gebser, M., Maratea, M., Ricca, F.: What's hot in the Answer Set Programming Competition. In: AAAI Conference on Artificial Intelligence (2016)
35. Gebser, M., Pührer, J., Schaub, T., Tompits, H.: A meta-programming technique for debugging answer-set programs. In: AAAI, pp. 448–453 (2008)
36. Gelfond, M., Kahl, Y.: Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach. Cambridge University Press (2014)
37. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: International Conference and Symposium on Logic Programming (ICLP/SLP), pp. 1070–1080 (1988)
38. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Deductive Databases. New Generation Computing **9**, 365–385 (1991)
39. Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability solvers. Handbook of Knowledge Representation **3**, 89–134 (2008)
40. Günter, A., Kühn, C.: Knowledge-based configuration-survey and future directions. In: German Conference on Knowledge-Based Systems, pp. 47–66. Springer (1999)
41. Haag, A.: Sales configuration in business processes. IEEE Intelligent Systems and their Applications **13**(4), 78–85 (1998)
42. Hadzic, T., Subbarayan, S., Jensen, R.M., Andersen, H.R., Møller, J., Hulgaard, H.: Fast backtrack-free product configuration using a precompiled solution space representation. In: PETO Conference, pp. 131–138 (2004)
43. Havur, G., Ozbilgin, G., Erdem, E., Patoglu, V.: Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 445–452. IEEE (2014)
44. Heinrich, M., Jungst, E.: A resource-based paradigm for the configuring of technical systems from modular components. In: IEEE Conference on Artificial Intelligence Applications, pp. 257–264. IEEE (1991)
45. Hotz, L., Felfernig, A., Günter, A., Tiihonen, J.: A short history of configuration technologies. Knowledge-based Configuration–From Research to Business Cases pp. 9–19 (2014)
46. Janhunen, T., Liu, G., Niemelä, I.: Tight integration of non-ground answer set programming and satisfiability modulo theories. Proceedings of GTTV **11**, 1–13 (2011)
47. Jensen, R.M.: CLab: A C++ library for fast backtrack-free interactive product configuration. In: Principles and Practice of Constraint Programming–CP 2004, pp. 816–816. Springer (2004)
48. Junker, U., Mailharro, D.: The logic of ilog (j) configurator: Combining constraint programming with a description logic. In: Workshop on Configuration at IJCAI, pp. 13–20 (2003)
49. Klein, R.: Model representation and taxonomic reasoning in configuration problem solving. In: Fachtagung für Künstliche Intelligenz (GWAI-91), pp. 182–194. Springer (1991)
50. Klein, R., Buchheit, M., Nutt, W.: Configuration as model construction: The constructive problem solving approach. In: Artificial Intelligence in Design, pp. 201–218. Springer (1994)
51. Küchlin, W., Sinz, C.: Proving consistency assertions for automotive product data management. Journal of Automated Reasoning **24**(1-2), 145–163 (2000)
52. Kusiak, A., Smith, M.R., Song, Z.: Planning product configurations based on sales data. IEEE Transactions on Systems, Man, and Cybernetics, Part C **37**(4), 602–609 (2007)
53. Lee, E.A.: Cyber Physical Systems: Design Challenges. In: IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pp. 363–369 (2008)
54. Lee, J., Meng, Y.: Answer set programming modulo theories and reasoning about continuous changes. In: Proc. International Joint Conference on Artificial Intelligence (IJCAI), pp. 990–996 (2013)
55. Leone, N., Ricca, F.: Answer set programming: a tour from the basics to advanced development tools and industrial applications. In: Reasoning Web International Summer School, pp. 308–326. Springer (2015)
56. Lifschitz, V.: What is answer set programming?. In: AAAI, vol. 8, pp. 1594–1597 (2008)
57. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. Artificial Intelligence **157**(1), 115–137 (2004)
58. Mailharro, D.: A classification and constraint-based framework for configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing **12**(4), 383–397 (1998)
59. Marcus, S., Stout, J., McDermott, J.: Vt: An expert elevator designer that uses knowledge-based backtracking. AI magazine **8**(4), 41 (1987)
60. McDermott, J.: R1: A rule-based configurer of computer systems. Artificial Intelligence **19**(1), 39–88 (1982)
61. Mileo, A., Nickles, M.: Probabilistic inductive answer set programming by model sampling and counting. In: International Workshop on Learning and Nonmonotonic Reasoning (LNMR), pp. 5–16 (2013)

62. Mittal, S., Falkenhainer, B.: Dynamic constraint satisfaction. In: Proceedings Eighth National Conference on Artificial Intelligence, pp. 25–32 (1990)
63. Mittal, S., Frayman, F.: Towards a generic model of configuraton tasks. In: Proc. International Joint Conference on Artificial Intelligence (IJCAI), pp. 1395–1401 (1989)
64. Møller, J., Andersen, H.R., Hulgaard, H.: Product configuration over the internet. Proceedings of the 6th INFORMS (2001)
65. Myllärniemi, V., Asikainen, T., Männistö, T., Soininen, T.: Kumbang configurator – a configuration tool for software product families. In: Proc. International Joint Conference on Artificial Intelligence (IJCAI), pp. 51–56 (2005)
66. Najmann, O., Stein, B.: A theoretical framework for configuration. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pp. 441–450. Springer (1992)
67. Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., Barry, M.: An a-prolog decision support system for the space shuttle. In: International Symposium on Practical Aspects of Declarative Languages, pp. 169–183. Springer (2001)
68. Orsvärn, K., Axling, T.: The tacton view of configuration tasks and engines. In: Workshop on Configuration at National Conference on Artificial Intelligence (AAAI) (1999)
69. Ostrowski, M., Schaub, T.: ASP modulo CSP: The clingcon system. Theory and Practice of Logic Programming **12**(4-5), 485–503 (2012)
70. Pargamin, B.: Vehicle sales configuration: the cluster tree approach. In: Configuration Workshop at ECAI at ECAI, pp. 35–40 (2002)
71. Pargamin, B.: Extending cluster tree compilation with non-boolean variables in product configuration: A tractable approach to preference-based configuration. In: Proc. International Joint Conference on Artificial Intelligence (IJCAI) (2003)
72. Rajkumar, R.R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical systems. In: Design Automation Conference (DAC), pp. 731–736 (2010)
73. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence **32**(1), 57–95 (1987)
74. Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., Leone, N.: Team-building with answer set programming in the gioia-tauro seaport. Theory and Practice of Logic Programming **12**(03), 361–381 (2012)
75. Russel, S., Norvig, P.: Artificial Intelligence - A Modern Approach, third edn. Prentice Hall (2009)
76. Sabin, D., Weigel, R.: Product configuration frameworks – a survey. IEEE Intelligent Systems **13**(4), 42–49 (1998)
77. Schreiber, A.T., Terpstra, P., Magni, P., Van Velzen, M.: Analysing and implementing VT using COMMON-KADS. In: Proc. Workshop on Knowledge Acquisition for Knowledge-Based Systems, pp. 44–1 (1994)
78. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1), 181–234 (2002)
79. Sinz, C., Blochinger, W., Küchlin, W.: PaSAT - Parallel SAT-checking with lemma exchange: Implementation and applications. Electronic Notes in Discrete Mathematics **9**, 205–216 (2001)
80. Sinz, C., Kaiser, A., Küchlin, W.: Detection of Inconsistencies in Complex Product Configuration Data Using Extended Propositional SAT-Checking. In: FLAIRS Conference, pp. 645–649 (2001)
81. Sinz, C., Kaiser, A., Küchlin, W.: Formal methods for the validation of automotive product configuration data. Artificial Intelligence for Engineering Design, Analysis and Manufacturing **17**(1), 75–97 (2003)
82. Snavely, G.L., Papalambros, P.Y.: Abstraction as a configuration design methodology. Advances in Design Automation **1**, 1993 (1993)
83. Soininen, T., Niemelä, I.: Developing a declarative rule language for applications in product configuration. In: Practical Aspects of Declarative Languages, pp. 305–319. Springer (1998)
84. Soininen, T., Niemelä, I., Tiihonen, J., Sulonen, R.: Representing configuration knowledge with weight constraint rules. Answer Set Programming **1** (2001)
85. Soloway, E., Bachant, J., Jensen, K.: Assessing the maintainability of XCON-in-RIME: Coping with problems of a very large rule base. In: Proc. International Conference on Artificial Intelligence, pp. 824–829. Morgan Kaufman (1987)
86. Song, Z., Kusiak, A.: Optimising product configurations with a data-mining approach. International Journal of Production Research **47**(7), 1733–1751 (2009)
87. Stumptner, M.: An overview of knowledge-based configuration. AI Communications **10**(2), 111–125 (1997)
88. Subbarayan, S.: CLib: configuration benchmarks library (2004)
89. Syrjänen, T.: Debugging inconsistent answer set programs. In: Proc. NMR, vol. 6, pp. 77–83 (2006)

90. Tiihonen, J., Anderson, A.: VariSales. Knowledge-based Configuration–From Research to Business Cases pp. 377–388 (2014)
91. Tiihonen, J., Heiskala, M., Anderson, A., Soininen, T.: WeCoTin–A practical logic-based sales configurator. AI Communications **26**(1), 99–131 (2013)
92. Tiihonen, J., Soininen, T., Niemelä, I., Sulonen, R.: A practical tool for mass-customising configurable products. In: Proc. International Conference on Engineering Design, pp. 1290–1299 (2003)
93. Tong, C., Sriram, D.: Artificial Intelligence in Engineering Design, Volume 1: Design Representation and Models of Routine Design. San Diego, CA: Academic Press (1992)
94. Tseitin, G.S.: On the complexity of proof in prepositional calculus. Zapiski Nauchnykh Seminarov POMI **8**, 234–259 (1968)
95. Tseng, H.E., Chang, C.C., Chang, S.H.: Applying case-based reasoning for product configuration in mass customization environments. Expert Systems with Applications **29**(4), 913–925 (2005)
96. Walter, R., Felfernig, A., Küchlin, W.: Constraint-based and sat-based diagnosis of automotive configuration problems. Journal of Intelligent Information Systems pp. 1–32 (2016)
97. Walter, R., Küchlin, W.: ReMax – A MaxSAT aided Product (Re-)Configurator. In: International Configuration Workshop, p. 59 (2014)
98. Walter, R., Zengler, C., Küchlin, W.: Applications of MaxSAT in Automotive Configuration. 15 th International Configuration Workshop **1**(2), 21 (2013)
99. Xie, H., Henderson, P., Kernahan, M.: Modelling and solving engineering product configuration problems by constraint satisfaction. International Journal of Production Research **43**(20), 4455–4469 (2005)
100. Yu, B., Skovgaard, H.J.: A configuration tool to increase product competitiveness. IEEE Intelligent Systems **13**(4), 34–41 (1998)
101. Zhang, S., Sridharan, M., Wyatt, J.L.: Mixed logical inference and probabilistic planning for robots in unreliable worlds. IEEE Transactions on Robotics **31**(3), 699–713 (2015)