SAT-Based Consistency Checking of Automotive Electronic Product Data

Carsten Sinz and **Andreas Kaiser** and **Wolfgang Küchlin**[1]

Abstract. Complex products such as motor vehicles or computers need to be configured as part of the sales process [3, 8]. If the sale is electronic, then the configuration and some validity checking of the order must be done electronically as part of an electronic product data management system (EPDMS). The EPDMS typically maintains a data base of sales options and parts together with a set of logical constraints expressing valid combinations of sales options and their transformation into manufacturable products. Due to the complexity of these constraints, creation and maintenance of the configuration data base is a nontrivial task and error-prone. We present our system BIS which is commercially used to check global consistency assertions about the product data base used by the EPDMS of a major car and truck manufacturer. The EPDMS uses Boolean logic to encode the constraints, and BIS translates the consistency assertions into problems which it solves using a propositional satisfiability checker. We expect our approach to be especially suited for rapidly changing complex products as they increasingly appear in electronic commerce.

1 Introduction

We present our system BIS, an extension to a commercially used electronic product data management system (EPDMS) at DaimlerChrysler AG, the manufacturer of the Mercedes lines of passenger cars and commercial vehicles. BIS is just now beginning commercial service to weed out residual defects in the product data base that is used by the main EPDMS for order checking and production planning. The EPDMS uses Boolean logic to encode the manufacturability constraints, and BIS translates global consistency assertions on the constraints into problems which it solves using a propositional satisfiability checker. BIS is programmed in modern object-oriented client/server technology and easily runs on a current laptop.

The automotive industry today manages to supply customers with highly individualized products at low prices by configuring each vehicle individually from a very large set of possible options. E.g., the Mercedes C-class of passenger cars knows far more than a thousand options, and on the average more than 30,000 cars will be manufactured before an order is repeated identically. Heavy commercial trucks are even more individualized, and every truck configuration is built only very few times on average.

Traditionally, a sales person will bespeak the individual order with the customer. Still, the space of possible variations is so great that the validity of each order needs to be checked electronically against a product data base which encodes the constraints governing legal combinations of options (e.g., no two radios; no steel sun-roof for convertibles). But the maintenance of a data base with thousands of logical rules is error-prone in itself, especially since it is under constant change due to the phasing in and out of models. Every fault in the data base may lead to a valid order rejected, or an invalid (non constructible) order accepted which may ultimately result in the assembly line to be stopped. Therefore, an EPDMS is employed to maintain the product data base and check orders.

DaimlerChrysler AG, for their Mercedes lines of cars and commercial trucks, employ a mainframe-based EPDMS which does the validity checking of each individual order (followed by initial production planning) in an off-line process. It also supports the on-line maintenance of the product data base by the staff of the product documentation department. The data base contains a large number of constraints formulated in Boolean logic. Some of the constraints represent general rules about valid combinations of sales options, other formulae are attached to parts and express the exact condition under which each part is needed for an order to be manufactured. Based on some years of experience with the current EPDMS, it was found that it is not humanly possible to keep such a large and constantly changing data base of logical rules and formulae absolutely defect-free without help from an automated reasoning system for the documentation logic.

Therefore our system BIS was created as an extension to the current EPDMS to help the product documentation staff in proving complex global consistency assertions about the data base. As an example, BIS can check for each of the thousands of sales options whether it can legally be contained in at least one valid (manufacturable) order. BIS can also deal with partially specified orders, checking e.g. which engine options are still valid given the preselected body and interior, or it can check which parts cannot possibly be part of any vehicles that go to a certain country.

While the current EPDMS is used off-line, it is clear that something similar must be used on-line in an electronic sales system for even simple configurable products, and a system like BIS is then needed to keep the product data base defect free. In an electronic market, there is no more knowledgeable sales-person to assist the customer and to prevent silly orders, and it may not even be feasible to contact the customer personally to sort out problems once an order is submitted. Hence we argue that the methods used in BIS today will be useful in electronic markets with even much simpler products tomorrow.

In Section 2, we shall now summarize, from [7], the capabilities of the core BIS system (BIS 1.0). In Section 3 we shall present some modifications and extensions that went into the current BIS 2.0, based on feedback from actual industrial use of BIS 1.0 on commercial product documentation data.

[1] Symbolic Computation Group, WSI for Computer Science, University of Tübingen and Steinbeis Technology Transfer Center OIT, Sand 13, 72076 Tübingen, Germany, HTTP://WWW-SR.INFORMATIK.UNI-TUEBINGEN.DE

2 BIS: A SAT-Based Consistency Checker for Product Documentation

Before turning to the description of the BIS system, we will need to give a rough picture of the underlying EPDM System which it complements. Then we present some consistency criteria that can be examined using the BIS system, and show how they translate into SAT instances. Thereafter we will shortly comment on the architecture of our system.

2.1 DaimlerChrysler's EPDM System DIALOG

In the following we will describe the EPDM system used for DaimlerChrysler's Mercedes lines more thoroughly.

A customer's order consists of a basic model class selection together with a set of further equipment codes describing additional features. Each equipment code is represented by a Boolean variable, and choosing some piece of equipment is reflected by setting the corresponding variable to *true*. As model classes can be decoded into a set of special equipment codes, all rules in the product documentation are formulated on the basis of codes.

Slightly simplified, each order is processed in three major steps, as is depicted in Figure 1:

1. *Order completion:* Supplement the customer's order by additional (implied) codes.
2. *Constructibility check:* Are all constraints on constructible models fulfilled by this order?
3. *Parts list generation:* Transform the (possibly supplemented) order into a parts list.

All of these steps are controlled by logical rules of the EPDMS. The rules are formulated in pure propositional logic using AND, OR and NOT as connectives, with additional restrictions placed on the rules depending on the processing step, as will be shown below.

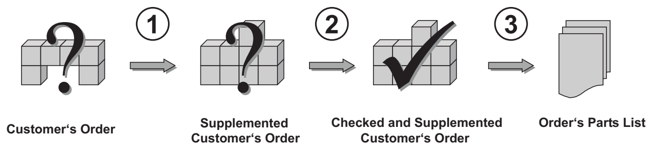

1
2
3
Customer's Order
Supplemented Customer's Order
Checked and Supplemented Customer's Order
Order's Parts List

Figure 1. Processing a customer's order.

Order Completion

The order completion (or supplementing) process adds implied codes to an order. The process is guided by special formulae associated with each code, which are of the following form:

$$Cond^S \longrightarrow c,$$

where c is a code (i.e. a propositional variable) and $Cond^S$ an arbitrary formula. The semantics of such a rule is that when condition $Cond^S$ evaluates to *true* under an order, then code c is added to that order. Thus, each rule application extends the order by exactly one code, and the whole completion process is iterated until no further changes result. Ideally, the relationship between original and augmented order should be functional. However, the result of the order completion process may depend critically on the ordering of rule application. We have shown in [7] how to identify potential instances of this problem.

Constructibility Check

In general, constructibility of a customer's order is checked according to the following scheme: For each code, there may be several rules indicating restrictions under which this code may be used. A code is called *constructible* (or *valid*) within a given order if all constraining rules associated with this code are fulfilled, i.e. all of these rules evaluate to *true*. For an order to be *constructible* (or *valid*), each code of the order must be valid. The constructibility check consists of two different parts: The first one is independent of the car model class considered, while the second one takes into account additional features of each car model class. Although the latter incorporates an additional hierarchical organization of rules, we will not elaborate on this. For our purpose the constructibility rules may be considered in a unified, simpler form:

$$c \longrightarrow Cond^C,$$

where c is a code and $Cond^C$ an arbitrary formula. Such a rule expresses the fact that whenever code c occurs in a customer's order, the order must fulfill condition $Cond^C$, i.e. $Cond^C$ must evaluate to *true* for this order.

Parts List Generation

The parts list is subdivided into modules, positions and variants, with decreasing generality from modules to variants. Parts are grouped in modules depending on functional and geometrical aspects. Each position contains all those parts which may be used alternatively in one place. The mutually exclusive parts of a position are specified using variants. Each variant is assigned a formula called a code rule and a part number. A parts list entry is selected for an order if its code rule evaluates to *true*. Thus, to construct the parts list for a completed and checked customer's order, one scans through all modules, positions, and variants, and selects those parts which possess a matching code rule.

2.2 Consistency of Product Documentation

Due to the complexity of the product documentation occurring in the automotive industry, some erroneous rules in the data base are almost unavoidable and usually quite hard to find. Moreover, the rule base changes frequently, and rules often introduce interdependencies between codes which at a first sight seem not to be related at all.

As the rule base not only reflects the knowledge of engineers, but also world wide legal, national and commercial restrictions, the complexity seems to be inherent to automotive product configuration, and is therefore hard to circumvent.

A priori, i.e. without explicit knowledge of intended constraints on constructible models, the following data base consistency criteria may be checked:

Necessary codes: Are there codes which must invariably appear in each constructible order?

Inadmissible codes: Are there any codes which cannot possibly appear in any constructible order?

Consistency of the order completion process: Are there any constructible orders which are invalidated by the supplementing process? Does the outcome of the supplementing process depend on the (probably accidental) ordering in which codes are added?

Superfluous parts: Are there any parts which cannot occur in any constructible order?

Ambiguities in the parts list: Are there any orders for which mutually exclusive parts are simultaneously selected?

Note that the aforementioned criteria are not checked on the basis of existing (or virtual) orders, but constitute intrinsic properties of the product documentation itself.

By incorporating additional knowledge on which car models can be manufactured and which cannot, further checks may be performed. Besides requiring additional knowledge, these tests often do not possess the structural regularity of the abovementioned criteria and thus cannot be handled as systematically as the other tests.

2.3 SAT Encoding of Consistency Assertions

We will now show how to encode the consistency criteria developed in the last section as propositional satisfiability (SAT) problems.

Transformation of the consistency criteria into SAT problems seems to be a natural choice for two reasons: first, the rules of the underlying EPDM system are already presented in Boolean logic; and second, SAT solvers are applied in other areas of artificial intelligence with increasing success [1, 6].

The formulation of all these consistency assertions requires an integrated view of the documentation as a whole or, more precisely, a characterization of the set of orders as they appear having passed the order completion process and the constructibility check. So we first concentrate on a Boolean formula describing all valid, extended orders that may appear just before parts list generation.

Let the set of order completion (supplementing) rules be $SR = \{sr_1, \ldots, sr_n\}$ with $sr_i = Cond_i^S \longrightarrow c_i$. Then the set of completely supplemented orders is described by formula Z, where

$$Z \ := \bigwedge_{1 \leq i \leq n} \left(Cond_i^S \Rightarrow c_i \right) \ .$$

Now, let $CR = \{cr_1, \ldots, cr_m\}$ be the set of constructibility rules with $cr_j = c_j \longrightarrow Cond_j^C$. Then the set of constructible orders is described by formula C, where

$$C \ := \bigwedge_{1 \leq j \leq m} \left(c_j \Rightarrow Cond_j^C \right) \ .$$

Moreover, the set of all orders that have passed the supplementing process and the constructibility control are described by B, where

$$B \ := \ Z \wedge C \ .$$

We now have reached our goal to generate a propositional formula reflecting the state before parts list generation. The mapping of the consistency criteria to SAT instances is now straightforward. For example, code c is inadmissible, iff $B \wedge c$ is unsatisfiable. The other criteria are converted accordingly, but some of them require a more sophisticated translation, especially those tests concerning the order completion process. The complete set of transformations from our consistency assertions to SAT instances – as they are actually built into the BIS system – can be found in [7].

Finally, it should be noted that the process described here is simplified in comparison to the actual order processing that takes place in the DIALOG system. The general ideas should nevertheless be apparent.

2.4 Integration into Work-Flow

We will now briefly describe how the BIS system is integrated into the existing product documentation process.

After having made a change to the documentation rule base (or, alternatively, in regular temporal intervals) some or all of the abovementioned consistency criteria are checked, and potential flaws of the documentation are reported by BIS.

Each inconsistency indicated by BIS has then to be analyzed and interpreted by the product documentation experts: If the product documentation does not correctly reflect reality (in the sense that it does not properly classify what actually can be manufactured), the error has to be corrected – either by adapting the documentation rules or by modifying the product itself. Otherwise the reported inconsistency most likely is an intended exceptional case that does not need any further processing.

Even if not all such inconsistencies are – or even can be – handled, the quality of the product documentation is nevertheless improved. This is an important fact, considering that SAT is an NP-complete problem. Thus, it cannot be guaranteed that the system will find all inconsistencies within a suitably short amount of time. We experienced, however, that for our application worst-case behavior and unacceptably long run-times are the rare exception; the run-times for each proof are usually clearly below one second.

2.5 Architecture of the BIS System

The BIS system is constructed employing object-oriented client/server technology. It consists of a general prover module programmed in C++ with a propositional satisfiability checker as its core component; a C++ server which maintains product data in raw and pre-processed form and handles requests by building the appropriate formulae for the prover; and a graphical user interface programmed in Java, through which tests can be started and results can be displayed. The three components communicate via CORBA interfaces, thereby achieving a great flexibility, allowing e.g. to place each component on a different, suitable computer or to use multiple instances of a component (e.g. prover), if the workload demands this. Figure 2 shows a schematic view of the BIS system architecture.

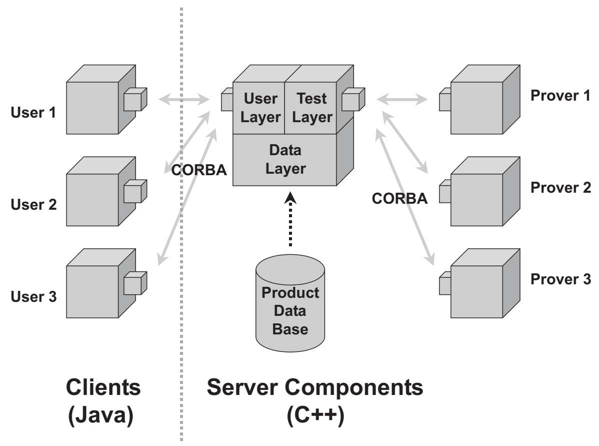
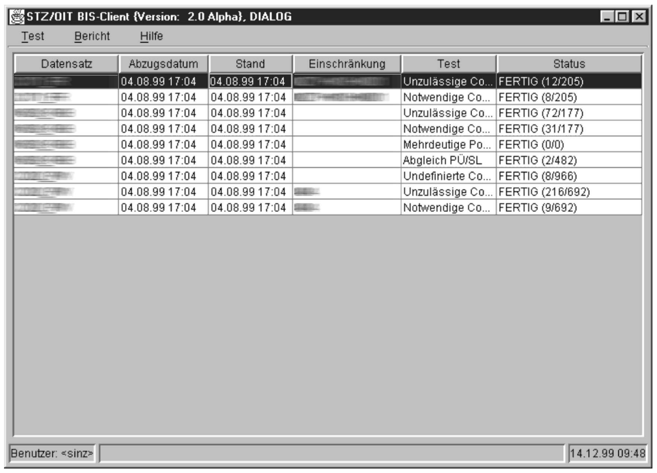


Figure 2. BIS system architecture.

Within the server, the UserLayer is responsible for authentication and handles user requests by starting the appropriate consistency tests. Therefore it employs the TestLayer which in turn is responsible for managing (i.e. scheduling, starting) all consistency checks. The data layer is used as a mediator between the TestLayer and the EPDM system, and supports caching of pre-computed data.

3 Extensions Based on Experience

Since the first evaluation deployment of the BIS 1.0 system (and even before), we have received a lot of feedback from DIALOG users at DaimlerChrysler. This helped us greatly in improving the system in various aspects. We will now describe relevant user feedback as well as experience-induced changes in greater detail.



Datensatz	Abzugsdatum	Stand	Einschränkung	Test	Status
[redacted]	04.08.99 17:04	04.08.99 17:04	[redacted]	Unzulässige Co...	FERTIG (12/205)
[redacted]	04.08.99 17:04	04.08.99 17:04	[redacted]	Notwendige Co...	FERTIG (8/205)
[redacted]	04.08.99 17:04	04.08.99 17:04		Unzulässige Co...	FERTIG (72/177)
[redacted]	04.08.99 17:04	04.08.99 17:04		Notwendige Co...	FERTIG (31/177)
[redacted]	04.08.99 17:04	04.08.99 17:04		Mehrdeutige Po...	FERTIG (0/0)
[redacted]	04.08.99 17:04	04.08.99 17:04		Abgleich PÜ/SL	FERTIG (2/482)
[redacted]	04.08.99 17:04	04.08.99 17:04		Undefinierte Co...	FERTIG (8/966)
[redacted]	04.08.99 17:04	04.08.99 17:04	[redacted]	Unzulässige Co...	FERTIG (216/692)
[redacted]	04.08.99 17:04	04.08.99 17:04	[redacted]	Notwendige Co...	FERTIG (9/692)

Benutzer: <sinz> 14.12.99 09:48

Figure 3. BIS system client.

The following items appeared to be indispensable for a broad everyday use:

Push-button technology: The logical prover component can be completely hidden from the user, and it needs no assistance in finding a proof. Interaction with the prover is done in terms familiar to the operating personnel.

Graphical user interface: The BIS system offers an elaborated graphical user interface, as can be seen from Figure 3. No cryptic command lines have to be typed by the user.

Short response times of the system: As BIS 1.0 was used more and more interactively, consistency checks had to exhibit short and predictable run-times. We will describe below in more detail how we could achieve this.

Customized special checks: Although we offered a general-purpose interface to the prover[2] which could be used to perform a lot of non-standard consistency checks on the product documentation, acceptance of this tool was rather poor. Thus, we implemented a set of further customized special checks and extended the client accordingly.

3.1 Additional Functionality

We will now report on functional extensions that went into BIS 2.0. As we already mentioned above, most of these additional tests could in principle have been performed with the general-purpose test facility of BIS 1.0, but required some kind of – frequently almost trivial – logical problem encoding by the user; an interpretation of the result reported by the client; and often an annoying manual generation of a series of tests.

[2] This interface allowed queries about the existence of valid orders with special properties, where the demanded property is an arbitrary propositional formula.

Restricting the Set of Valid Orders

The formalization mentioned above allows the analysis of the set of all valid orders. However, as it turned out, it is often necessary to restrict the set of orders to be considered to some subset of all valid orders. This may be needed, for example, to check assertions about all valid orders of a certain country, or about all valid orders with a special motor variant.

The formalization of order restrictions could easily be realized by adding a formula R describing the additional restriction to the constructibility formula B, and running the tests on $B \wedge R$.

Valid Additional Equipment Options

Not only for an individual order, but also for a whole class of orders, it may be interesting to know what kind of additional equipment may be selected without making the order invalid. This can be used on the one hand to analyze the product data, but may also serve a customer to find possible extensions of a partially specified order. This can be achieved as follows: Using the restriction possibility of the last paragraph, the partially specified order serves as a restriction R on the set of valid orders to be considered. Then it is checked for all codes c whether or not the formula $B \wedge R \wedge c$ is satisfiable. If this is the case, then code c is a valid extension of the partially specified order.

Combinations of Codes

Upon creation and maintenance of parts list entries, the following question frequently has to be answered: Given a fixed set of codes, which combinations of these codes may possibly occur in a valid order? The answer to this question decides over which parts list entries have to be documented and which have not.

Performing this kind of test one by one for each combination manually appears to be rather cumbersome. An automatic generation of all possible combinations seems to be more appropriate. Again, this can be simply realized by generating systematically all combinations one after the other and checking the corresponding formulae for satisfiability.

Groups of Codes

Although not reflected by the documentation structure, we found it characteristic for automotive product data that certain codes are symmetrically related. We call a set C of codes symmetrically related (with respect to a rule-based product documentation) if there is a non-empty subset R of those rules containing at least one code of set C, such that R is invariant under all permutations of the codes of set C.

A typical case for a set of symmetrically related codes is a set of mutually exclusive codes, where one of the codes must appear in each valid order, i.e. each order must contain exactly one code of the set. For example, in the DIALOG documentation system each order must contain exactly one code that determines the country in which the customer has ordered the car.

Since these kinds of symmetric relations can not be explicitly stated in the EPDM system, but are implicitly given by several rules formed with the binary connectives `NOT`, `AND` and `OR`, we added a possibility to check for a given list of codes (a group of codes) whether or not each valid order contains exactly one of these codes.

This is realized by checking the satisfiability of formula B that describes all valid orders, extended by the additional constraint that the order contains none or at least two of the codes specified in the group.

3.2 Extending the Propositional Language

While sets of mutually exclusive codes represent the most prominent example for a symmetrical relation, one can also think of situations where other symmetrical relations are applicable. For example, a customer can choose exactly one of a set of audio systems or he can completely dispense with audio systems. This means he can choose *at most one* of a set of options. Another example is a valid order that needs exactly k of a set of n colors specified. Obviously giving these kinds of restrictions in standard propositional logic leads to excessive growth in formula size which is certainly unacceptable for user interaction, and may even in the simplest case exhaust the available resources. The fact that the mutual exclusiveness of country codes in DaimlerChrysler's current product documentation is not explicitly stated underlines this.

To address this drawback we added – as is described in detail in [5] – the abovementioned expressions to standard propositional language. This extends the language by expressions of the form $Rk : X_1, \ldots, X_n$, where $R \in \{=, \neq, \leq, <, \geq, >\}$, k is a positive number and $X_1, \ldots, X_n$ are arbitrary formulae of the extended language. The semantics of such an expression is that exactly Rk of the n formulae are *true*. Thus, the fact that at most one of three possible audio systems A_1, A_2 and A_3 should appear in an order corresponds to the expression

$$\leq 1 : A_1, A_2, A_3 \ ,$$

which is equivalent to writing

$$\neg(A_1 \wedge A_2) \wedge \neg(A_1 \wedge A_3) \wedge \neg(A_2 \wedge A_3)$$

in pure propositional logic.

A closer analysis even shows that any formula in standard propositional logic can be transformed to an equivalent formula based solely on the additional connectives, which differs in size from the original formula only by a constant factor. Thus, these connectives provide us with a method to represent formulae for automotive product data management in a compact, structure-preserving and uniform way.

As a consequence of introducing additional connectives, we refrain from conversion to clausal normal form (CNF) for satisfiability checking – in contrast to most of the commonly used Davis-Putnam-style propositional theorem provers [2]. Although this step involves a more complex prover implementation using a tree data structure (as opposed to integer arrays for CNF representation), its benefit is beyond the mere compactification of formula representation. On formulae generated from automotive product data our prover showed in most cases similar or better performance. Moreover, we avoided an additional data structure to represent the CNF of the formula and therefore could reduce the complexity of the overall system as well as the space requirements and improve response time, because CNF conversion of very large formulae is non-trivial.

Even beyond consistency checking, we consider the introduction of a logical connective that reflects symmetrical relations to be essential to efficiently document product data on the basis of Boolean constraints (see also [9]).

4 Conclusion and Future Work

We presented BIS, a system to complement DaimlerChrysler's automotive EPDM system DIALOG. BIS serves as a tool to increase the quality of the product documentation by allowing to check certain global consistency conditions of the documentation data base as a whole. In BIS 2.0, the consistency assertions and the product documentation rules are translated to formulae of an extended language of propositional logic, which additionally includes a connective for symmetric relations.

Feedback from the documentation personnel showed us which features – among others – should be preferably included into a support tool for product documentation: ease of use via a graphical user interface; good integration into existing work-flow; push-button technology; and short response times.

Although current satisfiability checkers are quite advanced and SAT is still – and increasingly – an area of active research [4], we could learn from the special applicational needs how to improve propositional SAT tools and how to optimize prover techniques. Special constructs occurring frequently in product documentation, such as selection of one out of a set of n entities, are usually not appropriately supported by generic Boolean SAT checkers. Therefore, we see here a wide area of adaptations and improvements on prover technology and possibly further speed gains, brought forward by applicational needs.

For the future, we assume a system like BIS to be indispensable for electronic sales over the World Wide Web. Complex products need to be configured and checked electronically in large numbers, and thus the presence of a correct electronic product documentation receives increased attention. But electronic product documentation shows up to be complex and rapidly changing, thus making maintenance of the product data base difficult. Electronic business may increase the speed of product changes as well as the complexity of products, as product cycles and life times are likely to shrink down. Moreover, the needs of customers may vary considerably around the globe, thus forcing a diversification of the product palette.

Additionally, as we move towards fully electronic sales, configuration and on-line validity checking of even simple products may be required to be aided by electronic support systems in the future, and there will probably be no supporting human sales person. Such systems heavily depend on the quality of electronic product data, which we try to improve with our techniques.

REFERENCES

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, 'Symbolic model checking without BDDs', in *Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99)*, number 1579 in LNCS. Springer-Verlag, (1999).

[2] M. Davis and H. Putnam, 'A computing procedure for quantification theory', in *Journal of the ACM*, volume 7, pp. 201–215, (1960).

[3] F. Freuder, 'The role of configuration knowledge in the business process', *IEEE Intelligent Systems*, **13**(4), 29–31, (July/August 1998).

[4] I. Gent and T. Walsh, 'Satisfiability in the year 2000', *J. Automated Reasoning*, **24**(1–2), 1–3, (February 2000).

[5] A. Kaiser. Saturation-based satisfiability checking without CNF conversion, 2000. (unpublished manuscript).

[6] H. Kautz and B. Selman, 'Planning as satisfiability', in *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, pp. 359–363. John Wiley and Sons, (1992).

[7] W. Küchlin and C. Sinz, 'Proving consistency assertions for automotive product data management', *J. Automated Reasoning*, **24**(1–2), 145–163, (February 2000).

[8] J. McDermott, 'A rule-based configurer of computer systems', *Artificial Intelligence*, **19**(1), 39–88, (1982).

[9] T. Soininen and I. Niemelä, 'Developing a declarative rule language for applications in product configuration', in *Practical Aspects of Declarative Languages, First International Workshop (PADL'99)*, number 1551 in LNCS, pp. 305–319. Springer-Verlag, (1999).