Detection of Inconsistencies in Complex Product Configuration Data Using Extended Propositional SAT-Checking

Carsten Sinz and **Andreas Kaiser** and **Wolfgang Küchlin**
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

Abstract

We present our consistency support tool BIS, an extension to the electronic product data management system (EPDMS) used at DaimlerChrysler AG to configure the Mercedes lines of passenger cars and commercial vehicles. BIS allows verification of certain integrity aspects of the product data as a whole. The underlying EPDMS maintains a data base of sales options and parts together with a set of logical constraints expressing valid configurations and their transformation into manufacturable products. Due to the complexity of the products and the induced complexity of the constraints, maintenance of the data base is a nontrivial task and error-prone. By formalizing DaimlerChrysler's order processing method and converting global consistency assertions about the product data base into formulae of an extended propositional logic, we are able to employ a satisfiability checker integrated into BIS to detect inconsistencies, and thus increase the quality of the product data.

Introduction

Today's automotive industry manages to supply customers with highly individualized products by personalizing each vehicle using a very large set of configuration options. E.g., the Mercedes C-class of passenger cars allows far more than a thousand options, and on the average more than 30,000 cars will be manufactured before an order is repeated identically. 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 (Freuder 1998). 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. 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. 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 express the condition under which a part is included into the order's parts list. It turned out that it is very hard to keep such a large and constantly changing data base of logical rules defect-free without help from an additional 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 consistency assertions about the product data. BIS offers a set of pre-formulated integrity conditions, and allows verification of these conditions for the current constraint rule system of a model line. These integrity conditions encompass—among others—easily comprehensible properties of valid orders requiring no special engineering expert knowledge as well as consistency aspects which are hardly observable without an integrated look on the data base as a whole. The main goal of BIS is to reduce the number of errors in the documentation rules and thus increase the documentation quality.

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 outline the architecture of our system.

DaimlerChrysler's EPDM System DIALOG

In the following we will describe the EPDM system DIALOG, that is 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 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 list of parts.

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.

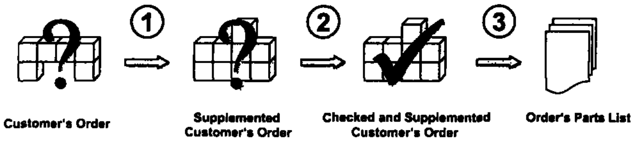

①
②
③
Customer's Order
Supplemented Customer's Order
Checked and Supplemented Customer's Order
Order's Parts List

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 elsewhere how to identify potential instances of this problem (Küchlin & Sinz 2000).

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.

Consistency of Product Documentation

Due to the complexity of automotive product documentation, 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.

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 (Biere *et al.* 1999; Kautz & Selman 1992). SAT can be seen as a specialization of constraint satisfaction (Wallace 1996), and many ideas are shared between these two research areas.

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

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

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

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

Moreover, the set of all orders that have passed the supplementing process and the constructibility check 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 can be found in (Küchlin & Sinz 2000).

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.

Integration into Work-Flow

We will now briefly describe how our 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. Each inconsistency indicated by BIS must then 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 reasonably 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-time for each proof is usually clearly below one second.

Architecture of the BIS System

The BIS system has been constructed employing object-oriented client/server technology. It consists of a general prover module programmed in C++ with a SAT-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 (Obj 1995), 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.

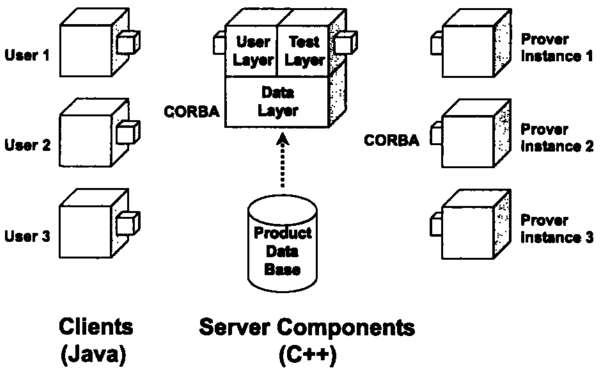


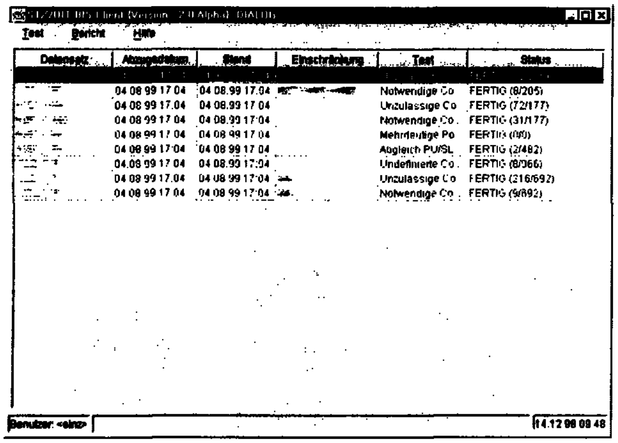
Figure 2: BIS system architecture.

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.

Extensions Based on Experience

Since the first evaluation deployment of BIS 1.0 (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. 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[1] 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.



Datensatz	Abzugsdatum	Stand	Einschränkung	Test	Status
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	04 08 99 17 04	04 08.99 17.04	[illegible]	Notwendige Co	FERTIG (8/205)
[illegible]	04 08 99 17.04	04 08.99 17 04		Unzulassige Co	FERTIG (72/177)
[illegible]	04 08 99 17 04	04 08.99 17:04		Notwendige Co	FERTIG (31/177)
[illegible]	04 08 99 17 04	04 08 99 17 04		Mehrdeutige Po	FERTIG (0/0)
[illegible]	04 08 99 17:04	04 08 99 17 04		Abgleich PU/SL	FERTIG (2/482)
[illegible]	04.08 99 17 04	04 08.99 17:04		Undefinierte Co	FERTIG (8/066)
[illegible]	04 08 99 17.04	04 08 99 17:04	[illegible]	Unzulassige Co	FERTIG (216/692)
[illegible]	04 08 99 17 04	04 08 99 17:04	[illegible]	Notwendige Co	FERTIG (9/692)

Figure 3: BIS system client.

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.

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. 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 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 arises: 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. Setting up these tests manually for each combination is rather cumbersome, whereas an automatic enumeration is trivial.

Groups of symmetrically related 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, we added a possibility to check for a given set 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.

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 ap-

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

plicable. 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 laying down such restrictions in standard propositional logic leads to excessive growth in formula size which is often unacceptable, and may even in simple cases 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 (Kaiser 2000)—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 (Davis & Putnam 1960). 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 compaction 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 nontrivial.

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.

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 verify 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 (Gent & Walsh 2000), 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 an area of adaptations and improvements on prover technology and possibly further speed gains, brought forward by applicational needs.

For the future, we expect 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 high quality electronic product documentation—which is made possible by our techniques—receives increased attention.

References

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

Davis, M., and Putnam, H. 1960. A computing procedure for quantification theory. In *J. ACM*, volume 7, 201–215.

Freuder, E. 1998. The role of configuration knowledge in the business process. *IEEE Intelligent Systems* 13(4):29–31.

Gent, I., and Walsh, T. 2000. Satisfiability in the year 2000. *J. Automated Reasoning* 24(1–2):1–3.

Kaiser, A. 2000. A sat-based propositional prover for consistency checking of automotive product data. Unpublished manuscript.

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

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

Object Management Group. 1995. *The Common Object Request Broker: Architecture and Specification.*

Wallace, M. 1996. Practical applications of constraint programming. *Constraints* 1(1–2).