Proving Consistency Assertions for Automotive Product Data Management

Wolfgang Küchlin and Carsten Sinz
Symbolic Computation Group, WSI for Computer Science, University of Tübingen and Steinbeis Technology Transfer Center OIT,
Sand 13, D-72076 Tübingen, Germany
http://www-sr.informatik.uni-tuebingen.de

Abstract. We present a formal specification and verification approach for industrial product data bases containing Boolean logic formulae to express constraints. Within this framework, global consistency assertions about the product data are converted into propositional satisfiability problems. Today's state-of-the-art provers turn out to be surprisingly efficient in solving the SAT-instances generated by this process. Moreover, we introduce a method for encoding special non-monotonic constructs in traditional Boolean logic. We have successfully applied our method to industrial automotive product data management and could establish a set of commercially used interactive tools that facilitate the management of change and help raise quality standards.

Keywords: formal specification, verification, product data management, product configuration, industrial application

1. Introduction

The use of formal methods is still uncommon in industrial practice [17]. Analysis of safety-critical systems and verification of both processors and protocols seem to be among the rare exceptions [10, 13, 15, 19]. The reason for this is manifold: First, there are intrinsic preconditions imposed on the process to be formalized. One is the requirement of precise, symbolic input data; this contrasts with the imprecise data delivered by physical measurement. Another is the algorithmic complexity of symbolic techniques which is usually much higher than that of numerical or heuristic procedures. Second, the industrial process to be formalized has to be understood very precisely in every aspect, and modeling has to start on a well-founded basis. The language in which the process is to be described must have sufficient expressive power, but should exclude intractable logics—incomplete, undecidable, or merely too time- or space-consuming for the intended purpose.

Besides these intrinsic reasons there are more practical ones: On the one hand, one is faced with industrial prejudice against the power of formal verification [4]; on the other hand, research in the ATP community is often not directly concerned with real-world application aspects.

Further obstacles arise when communication between researchers and practitioners is hindered by different worlds of thought, or when incorporation of formal methods requires major changes in the actual production process. However, successful verification projects conducted in the realm of industry may further the acceptance of formal methods [3, 12].

In this article we describe a method to reveal inconsistencies in a data base used by DaimlerChrysler AG to check the constructibility of motor-vehicles of the Mercedes-Benz lines. This project was greatly helped by the fact that Boolean logic was already used to express constraints in the product data base. Difficulties were imposed by the mere complexity, as the data that had to be considered for some tests consisted of more than 18,000 rules (elementary Boolean formulae) and 1,700 propositional variables. In addition we had to formalize significant portions of the industrial process involved in order to prove assertions that could not be checked so far.

Although the general problem of product configuration has gained interest over the last years, we have not yet seen the formalization and verification of an existing large-scale industrial system.

The paper is organized as follows: First, we briefly describe the automobile constructibility data base, its integration into the order processing and production process, and show some potential inconsistencies that might occur in the data base. Then a formalization of the process to check individual car orders is presented, followed by a variety of global data base consistency criteria formulated as propositional satisfiability problems. Finally, we present experimental results achieved using a state-of-the-art propositional satisfiability checker.

2. Product Documentation and Order Processing

The Mercedes-Benz passenger car and commercial vehicle production encompasses a wide variety of different models customers can order. Apart from different model classes, design lines and engine variants, an extraordinary number of supplementary equipment may be selected. Not all theoretically possible combinations of variants can actually be produced, however. Geometrical, electrical or other engineering limitations are as common as legal or sales restrictions. Moreover, the availability of certain models can differ from country to country and undergo substantial temporal change. To automate administration and production tasks, exact knowledge about valid models is needed in electronic form. Thus, a data base (called product documentation) is employed

to draw the distinction between models that can be manufactured and those that cannot.

A second, but equally important purpose of the product documentation is to transform a customer's model description into a parts list for the requested vehicle, which can then be fed into the production planning process.

2.1. Processing a Customer's Order

A customer's order consists of a basic model class selection together with a set of further equipment codes describing additional features. As model classes can be decoded into a few special equipment codes, all rules in the product documentation are formulated on the basis of these codes, which are just propositional variables.

Ascertaining the constructibility of an individual order as well as parts list generation and other intermediate steps are performed by evaluating Boolean formulae stored in the product documentation data base.

Within our application constructibility may only be checked with respect to the rules of the product documentation, not with respect to physical reality. Therefore, we call an order *constructible* (or *valid*) if the corresponding vehicle model can be manufactured according to the product documentation; otherwise an order is called *invalid*.

The most important of the aforementioned intermediate steps—and the only one considered here—allows the completion of orders by appending additional codes. This supplementing process is mainly used to add codes that are implied by technical dependencies or that make up equipment packages.

The whole order processing procedure (slightly simplified) consists of the following steps:

1. *Order completion:* Extend the customer's order by additional 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.

These three steps are illustrated in Figure 1, schematically on the left hand side and by a concrete example on the right.

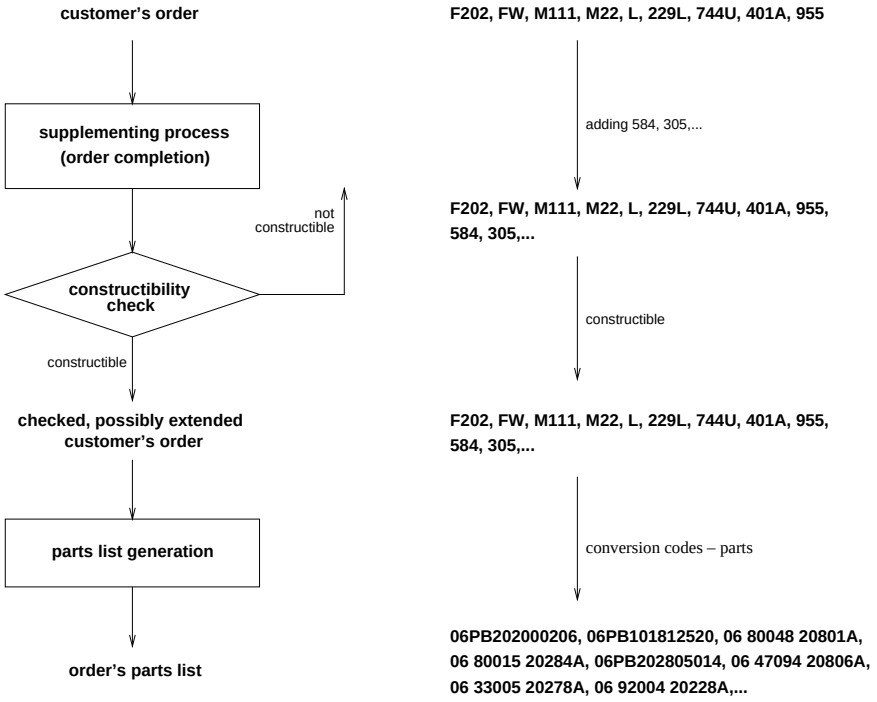

Figure 1. Processing customers' orders

2.2. Consistency of the Product Documentation

The mere complexity of the product documentation sometimes induces erroneous data base entries that are usually hard to find. Global usage and varying knowledge of the operating personnel aggravate this trend. In one of the evaluated settings, consisting of that part of the product documentation that specifies the limousines of the Mercedes-Benz C-class, 1,151 codes and formulae containing a total of more than 170,000 logical symbols had to be considered.

The formal specification of the complete process enables posterior verification of important aspects and thus offers new possibilities to reduce inconsistencies.

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

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

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

Consistency of the order completion process: Are there any constructible orders that 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 that cannot occur in any constructible order?

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

By using additional information further checks may be performed. Most of them require only minor changes with respect to the above criteria. The existence of valid orders with certain constraints falls under this class of checks. They are an easy generalization of the search for necessary or inadmissible codes.

We see our main contribution in showing how to formalize these consistency checks and how to apply theorem proving methods to improve the data quality within an existing industrial process.

3. A Formal View of Product Documentation

We will now describe the data base and its functionality more thoroughly. Starting with constructibility, and continuing with the order completion process, we are finally led to parts list generation.

Thus, we achieve a formal description of the complete actual order processing procedure for Mercedes-Benz cars and trucks. This formalization will afterwards serve as the basis on which product documentation consistency criteria can be formulated.

3.1. Preliminaries

Let $\mathcal{C}$ be the set of all equipment codes used in the product documentation. Then a customer's order O is the subset of $\mathcal{C}$ corresponding to the equipment selected in the order. Obviously each order O may be interpreted as a truth assignment by using the characteristic function $\chi_O : \mathcal{C} \to \{0, 1\}$ of O relative to $\mathcal{C}$, where we interpret 1 as truth and 0 as falsity.

All checks and modifications performed on a customer's order O depend on the evaluation of formulae (rules of the product documentation) under the truth assignment χ_O. For a formula F this evaluation is denoted by $\chi_O^*(F)$. We will also use the notation $\chi_O \models F$, or even shorter $O \models F$, instead of $\chi_O^*(F) = 1$.

Furthermore, we assume that each order contains a specification (exactly one of the codes L and R) of whether left or right hand side steering is demanded.

3.2. Constructibility

In general, constructibility[1] of a customer's order O 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 within O if all constraining rules associated with this code are fulfilled, that is, all of these rules evaluate to *true* under χ_O. For an order to be constructible, each code of the order must be constructible.

The constructibility check consists of two independent 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. The latter also depends on the kind of steering[2] (left or right hand side) under consideration.

The model class independent part of the constructibility check consists of a rule for each code indicating a constraint under which it may (or may not) be used. For a code c we use the notation $\mathrm{C}^{\mathrm{I}}(c)$ for the model class independent rule corresponding to c. To pass the model class independent constructibility check, an order O must fulfill the following condition:

$$O \models \mathrm{C}^{\mathrm{I}}(c) \quad \text{for each } c \in O \ . \tag{1}$$

The second part of the constructibility check is more complex, and the relevant rules are hierarchically organized, as shown in Figure 2.

There is a set $\mathcal{P}$ of geometric positions and a set $\mathcal{V}$ of variants. Positions are grouped reflecting common functionality or usage. With each position $p \in \mathcal{P}$ a unique code c is associated; however, the converse need not hold: hence, a code may occur at different positions. We use the function $\mathrm{Pos} : \mathcal{C} \to 2^{\mathcal{P}}$ to denote all positions associated with a code. Positions are unique within the whole product documentation, and for each position a (possibly empty) set of variants $V \subseteq \mathcal{V}$ exists. Each variant possesses an additional steering attribute (left or right, L or R). The function $\mathrm{Var} : \mathcal{P} \times \{L, R\} \to 2^{\mathcal{V}}$ selects all variants at a position with matching steering type. The model class dependent constructibility rules[3] $\mathrm{C}^{\mathrm{D}}(p, v)$ are indexed by a pair $(p, v) \in \mathcal{P} \times \mathcal{V}$.

[1] German: *Baubarkeit*
[2] German: *Lenkungsvariante*
[3] German: *Baubarkeitsregeln*

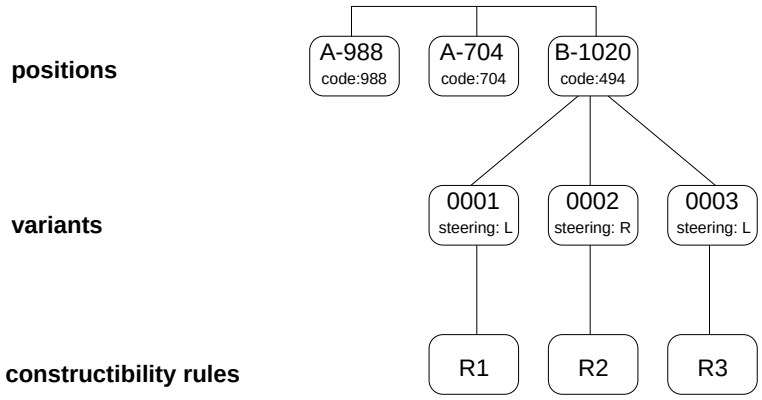

Figure 2. Structure of model class dependent constructibility

For a code c to be constructible, at least one variant v of matching steering type must exist for every position p associated with c, such that $\mathrm{C}^{\mathrm{D}}(p, v)$ is fulfilled. In a valid order O every code $c \in O$ must be constructible.

Thus we obtain the following condition for an order O of steering type s to pass the model class dependent constructibility check:

$$O \models \mathcal{B}(c, s) \quad \text{for each } c \in O, \tag{2}$$

where $\mathcal{B}(c, s)$ is defined as

$$\mathcal{B}(c, s) = \bigwedge_{p \in \mathrm{Pos}(c)} \bigvee_{v \in \mathrm{Var}(p,s)} \mathrm{C}^{\mathrm{D}}(p, v) \ . \tag{3}$$

The cardinalities of $\mathrm{Pos}(c)$ and $\mathrm{Var}(p, s)$ are usually less than a dozen, the individual rules $\mathrm{C}^{\mathrm{D}}(p, v)$ normally consist of much fewer than a hundred symbols.

For an order to be valid, it must pass both the model dependent and the independent check.

3.3. Order Completion Process

The order completion process adds implied codes to an order. The process is guided by special formulae, called supplementing rules[4], associated with each code. These rules are structurally organized in the same way as those of the model dependent constructibility check, that is, in positions and variants. Each rule application extends the order

[4] German: *Zusteuerungsregeln*

by exactly one code, so 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 order of rule application, whereas the exact sequence in which individual codes are added is sometimes obscure. Therefore, the functional relationship cannot be assumed to hold in general. We will show below how to identify potential instances of this problem.

As is the case with the model class dependent constructibility check, supplementing rules $\mathrm{S}(p, v)$ are hierarchically organized in positions $p \in \mathcal{P}$ and variants $v \in \mathcal{V}$.[5] The semantics of positions is different, though. In order to activate a supplementing step for code c, it is sufficient that any rule for code c is fulfilled, independent of the position or variant of the rule. To avoid invalidation of correct orders, constructibility is also considered during such a step. This works as follows: Besides the supplementing rule $\mathrm{S}(p, v)$ for code c, the corresponding formulae $\mathrm{C}^{\mathrm{D}}(p, v)$ and $\mathrm{C}^{\mathrm{I}}(c)$ must evaluate to *true* under order O.

Thus, code c is added to order O of steering type s, if

$$O \models \mathcal{Z}(c, s), \tag{4}$$

where $\mathcal{Z}(c, s)$ is defined as

$$\mathcal{Z}(c, s) = \Bigg(\bigvee_{\substack{p \in \mathrm{Pos}(c) \\ v \in \mathrm{Var}(p,s)}} \big(\mathrm{S}(p, v) \wedge \mathrm{C}^{\mathrm{D}}(p, v) \big) \Bigg) \wedge \mathrm{C}^{\mathrm{I}}(c) \ . \tag{5}$$

Symbolically, we can express the admissible steps of the supplementing process as a rewrite relation $\longrightarrow_S \subseteq 2^{\mathcal{C}} \times 2^{\mathcal{C}}$. Thus $O \longrightarrow_S O'$ iff there is a code c such that $O \models \mathcal{Z}(c, s)$, where s is the unique steering code occurring in O, $c \notin O$ and $O' = O \,\dot\cup\, \{c\}$. We also use the notation $O \xrightarrow{c}_S O'$ if O' is obtained by adding code c to order O. Hence we have

$$\longrightarrow_S \quad = \quad \bigcup_{c \in \mathcal{C}} \xrightarrow{c}_S \ .$$

Note that relation $\longrightarrow_S$ is a terminating reduction-relation. The reflexive-transitive closure of $\longrightarrow_S$ is denoted by $\longrightarrow_S^*$, and the n-fold product of $\longrightarrow_S$ by $\longrightarrow_S^n$. Furthermore, we write $\longrightarrow_S^{\leq n}$ for $\bigcup_{0 \leq i \leq n} \longrightarrow_S^i$. Definitions of basic notions regarding rewrite systems can be found in [8].

[5] $\mathrm{C}^{\mathrm{D}}(p, v)$ corresponds to $\mathrm{S}(p, v)$

3.4. Parts List Generation

The parts list[6] 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. Admissible variants depend, as is the case with model class dependent constructibility, on the steering type under consideration. Each variant is assigned a formula called a code rule[7], and a part number. The structure of the parts list is depicted in Figure 3.

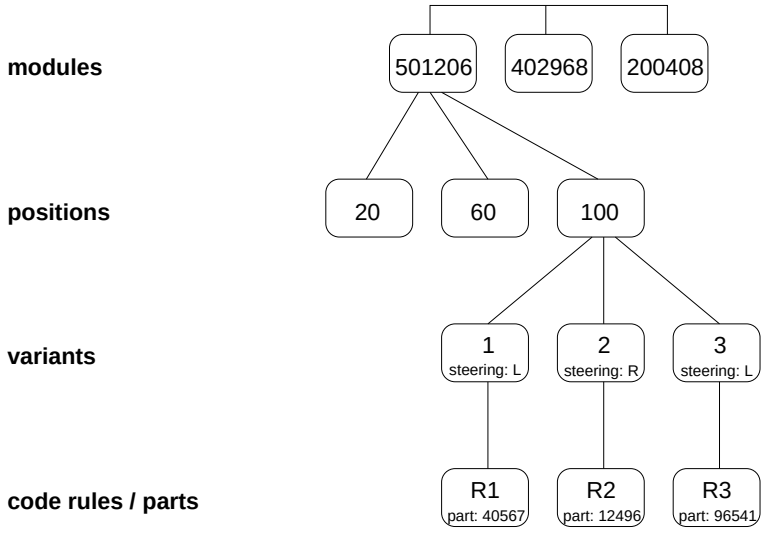

Figure 3. Structure of the parts list

The sets of part numbers, modules, positions, and variants of the parts list are denoted by $\mathcal{N}$, $\mathcal{M}$, $\mathcal{I}$, and $\mathcal{W}$, respectively. All variants of module $m \in \mathcal{M}$ at position $p \in \mathcal{I}$ with steering type s are delivered by function $\mathrm{PV} : \mathcal{M} \times \mathcal{I} \times \{L, R\} \to 2^{\mathcal{W}}$, the steering type of a variant v of module m and position p is determined by $\lambda(m, p, v)$. The code rule of variant v at position p in module m is accessible through $\mathrm{R}(m, p, v)$, the corresponding part is $\tau(m, p, v)$. The mapping $\tau : (m, p, v) \mapsto \tau(m, p, v)$ need not be injective: for each position there is a part, but a part may be used at many positions. Moreover, let the function $\Omega : \mathcal{N} \to 2^{\mathcal{M} \times \mathcal{I} \times \mathcal{W}}$ select all occurrences of a certain part in the parts list.

[6] German: *Teileliste, Stückliste*
[7] German: *Coderegel*

Variant v at position p of module m is selected for an order O of steering type s, if

$$\lambda(m,p,v) = s \quad \text{and} \quad O \models \mathrm{R}(m,p,v) \ . \tag{6}$$

In order to construct the parts list for a completed and checked customer's order O, one thus scans through all modules, positions, and variants, and selects those parts which possess a matching steering attribute and a code rule that evaluates to *true* under O.

Usually, exactly one variant has to be selected for each module position.

Let us end this section with two notes:

1. The product data base as described above is in use for the Mercedes-Benz passenger cars only. Commercial vehicles are documented in a slightly different fashion.

2. In the actual process some minor refinements and exceptional cases occur which we did not consider here.

4. Formalization of Consistency Assertions

In this section we want to develop criteria expressing various consistency aspects of the product documentation data base as a whole. These aspects were already briefly mentioned in Section 2.2.

Most of these consistency assertions require a characterization of the set of orders having passed the order completion process and the constructibility check, just before parts list generation.

Starting with a formula representing all constructible models, and extending it by a partial formula concerning the supplementing process, we finally reach the required characterization. Having this formula at hand, we can formulate the aforementioned consistency assertions as propositional satisfiability problems.

Throughout this section we assume the set of Boolean variables to be fixed to $\mathcal{C}$. The set of propositional formulae over $\mathcal{C}$ is denoted by $\mathcal{F}(\mathcal{C})$ or simply $\mathcal{F}$. Formula variables are implicitly assumed to range over $\mathcal{F}$, sets of orders are subsets of $2^{\mathcal{C}}$.

Furthermore, we assume $\wedge, \vee, \neg$ and $\bot$ as basic logical constants, whose semantics are defined, as usual, by

$$\begin{aligned}
\chi^*_O(F \wedge G) &= \min(\chi^*_O(F), \chi^*_O(G)) \\
\chi^*_O(F \vee G) &= \max(\chi^*_O(F), \chi^*_O(G)) \\
\chi^*_O(\neg F) &= 1 - \chi^*_O(F) \\
\chi^*_O(\bot) &= 0
\end{aligned}$$

Other symbols ($\Rightarrow$, $\top$) and their semantics are derived from the logical constants as usual.

Moreover, we restrict our attention to only one model class. This is no limitation in practice, because model classes are mutually independent.

DEFINITION 4.1. *A propositional formula F describes a set S of orders if S is the set of models of F (i.e., $O \models F$ iff $O \in S$).*

PROPOSITION 4.2. *The set of model class independent constructible orders is described by formula $\mathcal{B}^{\mathrm{I}}$, where*

$$\mathcal{B}^{\mathrm{I}} \;:=\; \bigwedge_{c \in \mathcal{C}} (c \Rightarrow \mathrm{C}^{\mathrm{I}}(c)) \;\; . \tag{7}$$

Proof. Let S be the set of model class independent constructible orders. By (1) we know that $O \in S$ iff $O \models \mathrm{C}^{\mathrm{I}}(c)$ for each $c \in O$. So we have to show that

$$O \models \mathcal{B}^{\mathrm{I}} \quad \text{iff} \quad O \models \mathrm{C}^{\mathrm{I}}(c) \;\text{ for each }\; c \in O \;\; . \tag{$*$}$$

At first, note that evaluation of $\mathcal{B}^{\mathrm{I}}$ under any χ_O yields

$$\chi^*_O(\mathcal{B}^{\mathrm{I}}) = \min_{c \in \mathcal{C}} \{\max(1 - \chi_O(c), \chi^*_O(\mathrm{C}^{\mathrm{I}}(c)))\} \;\; .$$

We now prove both implications of ($*$).

"⇒": Let $O \models \mathcal{B}^{\mathrm{I}}$ and $c' \in O$ arbitrary. Then $\chi^*_O(\mathcal{B}^{\mathrm{I}}) = 1$ and, in particular, $\max(1 - \chi_O(c'), \chi^*_O(\mathrm{C}^{\mathrm{I}}(c'))) = 1$. As $c' \in O$ implies $\chi_O(c') = 1$, we have $\chi^*_O(\mathrm{C}^{\mathrm{I}}(c')) = 1$ and thus $O \models \mathrm{C}^{\mathrm{I}}(c')$.

"⇐": Let $O \subseteq \mathcal{C}$ with $O \models \mathrm{C}^{\mathrm{I}}(c)$ for each $c \in O$. Choose any $c' \in \mathcal{C}$. If $c' \notin O$ then $\chi_O(c') = 0$. Otherwise we have $\chi^*_O(\mathrm{C}^{\mathrm{I}}(c')) = 1$. In both cases $\max(1 - \chi_O(c'), \chi^*_O(\mathrm{C}^{\mathrm{I}}(c))) = 1$ holds. As c' was chosen arbitrarily we have $\chi^*_O(\mathcal{B}^{\mathrm{I}}) = 1$.

PROPOSITION 4.3. *The set of model class dependent constructible orders is described by formula $\mathcal{B}^{\mathrm{D}}$, where*

$$\mathcal{B}^{\mathrm{D}} \;:=\; \bigwedge_{c \in \mathcal{C}} \left(c \;\Rightarrow\; \bigwedge_{\substack{p \in \mathrm{Pos}(c) \\ s \in \{L,R\}}} \left(s \;\Rightarrow\; \bigvee_{v \in \mathrm{Var}(p,s)} \mathrm{C}^{\mathrm{D}}(p, v) \right) \right) \tag{8}$$

$$\;=\; \bigwedge_{\substack{c \in \mathcal{C} \\ s \in \{L,R\}}} \Big(c \wedge s \;\Rightarrow\; \mathcal{B}(c, s) \Big), \tag{9}$$

and $\mathcal{B}(c, s)$ is defined by Equation (3).

Proof. The equivalence of (8) and (9) is easily shown. The rest of the proof is similar to that of Proposition 4.2. Let s_O be the uniquely determined steering type of order O. Using Definition (2) we have to show that

$$O \models \mathcal{B}^{\mathrm{D}} \quad \text{iff} \quad O \models \mathcal{B}(c, s_O) \text{ for each } c \in O \ . \qquad (**)$$

Formula (9) yields for arbitrary O

$$\chi^*_O(\mathcal{B}^{\mathrm{D}}) = \min_{c \in \mathcal{C}, s \in \{L,R\}} \{\max(1 - \chi_O(c), 1 - \chi_O(s), \chi^*_O(\mathcal{B}(c,s)))\} \ .$$

We again show the two implications of $(**)$ independently:

"⇒": Let $O \models \mathcal{B}^{\mathrm{D}}$ and $c' \in O$ arbitrary. Furthermore, let s_O be the uniquely determined steering type of O. Then $\chi^*_O(\mathcal{B}^{\mathrm{D}}) = 1$ and $\max(1 - \chi_O(c'), 1 - \chi_O(s_O), \chi^*_O(\mathcal{B}(c', s_O))) = 1$. As $\chi_O(c') = \chi_O(s_O) = 1$ we have $\chi^*_O(\mathcal{B}(c', s_O))) = 1$ and thus $O \models \mathcal{B}(c', s_O)$.

"⇐": Let $O \subseteq \mathcal{C}$ with $O \models \mathcal{B}(c, s_O)$ for each $c \in O$ and uniquely determined $s_O \in O \cap \{L, R\}$. Choose any $c' \in \mathcal{C}, s' \in \{L, R\}$. For $s' \neq s_O$ we obtain $\chi_O(s') = 0$. If $c' \notin O$ then $\chi_O(c') = 0$. Otherwise ($s' = s_O, c' \in O$) we have $\chi^*_O(\mathcal{B}(c', s')) = 1$. In each case $\max(1 - \chi_O(c'), 1 - \chi_O(s'), \chi^*_O(\mathcal{B}(c', s')))\} = 1$ holds. As c' and s' were chosen arbitrarily we have $\chi^*_O(\mathcal{B}^{\mathrm{D}}) = 1$.

Formula $\mathcal{B}^{\mathrm{D}}$ allows orders with either no or multiple steering codes. To enforce exactly one $s \in \{L, R\}$ in each order, we use the following formula:

PROPOSITION 4.4. *The set of all orders containing exactly one steering code $s \in \{L, R\}$ is described by formula $\mathcal{B}^{\mathrm{S}}$, where*

$$\mathcal{B}^{\mathrm{S}} := (L \vee R) \wedge \neg(L \wedge R) \ . \qquad (10)$$

Proof. By case distinction whether or not $L, R \in O$.

COROLLARY 4.5. *The set of (effectively) constructible orders is described by formula $\mathcal{B}^{\mathrm{C}}$, where*

$$\mathcal{B}^{\mathrm{C}} := \mathcal{B}^{\mathrm{I}} \wedge \mathcal{B}^{\mathrm{D}} \wedge \mathcal{B}^{\mathrm{S}} \ . \qquad (11)$$

Proof. By Propositions 4.2, 4.3 and 4.4, and the fact that $\wedge$ corresponds to set intersection.

What we have achieved so far is the formalization of the constructibility check; the order completion process, however, is not yet integrated. At first glance, it may seem that the supplementing process requires

modeling some kind of state transition. Describing each step in this way inevitably necessitates the use of non-monotonic or modal logic. Neither way seems to be desirable due to complexity considerations.

4.1. Avoiding State Transitions

The major part of the criteria to be checked requires no knowledge of the dynamic development caused by the supplementing process. This holds, e.g., for the (static) description of all possible states appearing after completion of the supplementing process. Moreover, formalization of changes caused by a fixed number of supplementing steps is also possible within propositional logic, as is shown later.

Before resuming with further criteria, we want to comment on some simple properties of relation $\longrightarrow_S$.

At first, note that an order is left unchanged by the supplementing process iff it is in $\longrightarrow_S$-normal form. Furthermore, an $\longrightarrow_S$-normal form of an order O of steering type s is characterized by the following property (compare with Equation (4)):

$$O \not\models \mathcal{Z}(c,s) \quad \text{for all } c \in \mathcal{C} \setminus O \ . \tag{12}$$

We therefore obtain

PROPOSITION 4.6. *The set of orders left unchanged by the supplementing process is described by formula $\mathcal{Z}^{\mathrm{U}}$, where*

$$\mathcal{Z}^{\mathrm{U}} := \bigwedge_{\substack{c\in\mathcal{C}\\ s\in\{L,R\}}} \left(s \wedge \Big(\bigvee_{\substack{p\in\mathrm{Pos}(c)\\ v\in\mathrm{Var}(p,s)}} \big(\mathrm{S}(p,v) \wedge \mathrm{C}^{\mathrm{D}}(p,v)\big)\Big) \wedge \mathrm{C}^{\mathrm{I}}(c) \ \Rightarrow \ c \right) \tag{13}$$

$$= \bigwedge_{\substack{c\in\mathcal{C}\\ s\in\{L,R\}}} \Big(s \wedge \mathcal{Z}(c,s) \Rightarrow c \Big), \tag{14}$$

and $\mathcal{Z}(c,s)$ is defined by Equation (5).

Proof. Formula (13) is just the version of (14) with the definition of $\mathcal{Z}(c,s)$ expanded, so equivalence is obvious. Furthermore, let s_O be the unique steering type of order O. Using (12) we now have to show that

$$O \models \mathcal{Z}^{\mathrm{U}} \quad \text{iff} \quad O \not\models \mathcal{Z}(c,s_O) \ \text{ for all } \ c \in \mathcal{C} \setminus O \ . \tag{$***$}$$

Evaluation yields for arbitrary O

$$\chi^*_O(\mathcal{Z}^{\mathrm{U}}) = \min_{c\in\mathcal{C},s\in\{L,R\}} \{\max(1-\chi_O(s), 1-\chi^*_O(\mathcal{Z}(c,s)), \chi_O(c))\} \ .$$

As in similar proofs before, equivalence $(***)$ is shown by proving the two implications directly, which we feel free to omit here.

We are now able to describe the set of orders that appear just before parts list generation:

COROLLARY 4.7. *The set of orders which have passed the supplementing process and the constructibility check are described by formula* $\mathcal{B}^{\mathrm{Z}}$*, where*

$$\mathcal{B}^{\mathrm{Z}} := \mathcal{B}^{\mathrm{C}} \wedge \mathcal{Z}^{\mathrm{U}} \quad . \tag{15}$$

Proof. Corollary 4.5, Proposition 4.6.

Four of the five initially posed problems can now be solved:

Necessary codes: Code c is necessary iff the formula

$$\mathcal{B}^{\mathrm{Z}} \Rightarrow c$$

is a tautology, which is equivalent to $\mathcal{B}^{\mathrm{Z}} \wedge \neg c$ being unsatisfiable.

Inadmissible codes: Code c is inadmissible iff the formula

$$c \Rightarrow \neg\mathcal{B}^{\mathrm{Z}}$$

is a tautology. This holds iff $\mathcal{B}^{\mathrm{Z}} \wedge c$ is not satisfiable.

Superfluous parts: Part t is superfluous iff

$$\mathcal{B}^{\mathrm{Z}} \wedge \bigvee_{(m,p,v)\in\Omega(t)} \Big(\lambda(m,p,v) \wedge \mathrm{R}(m,p,v)\Big)$$

is unsatisfiable.

Ambiguities in the parts list generation process: There is an ambiguity between variants v_1 and v_2 at position p of module m iff the formula

$$\mathcal{B}^{\mathrm{Z}} \wedge \lambda(m,p,v_1) \wedge \mathrm{R}(m,p,v_1) \wedge \lambda(m,p,v_2) \wedge \mathrm{R}(m,p,v_2)$$

is satisfiable.

The interpretation of these formulae is straightforward using the set-of-orders view suggested by Definition 4.1.

Note that $\mathcal{B}^{\mathrm{Z}}$ occurs exclusively positive in all these cases, when formulated as satisfiability problems. We can greatly profit from this fact as follows. Most propositional provers require the input to be in

conjunctive normal form. As $\mathcal{B}^{\mathrm{Z}}$ is a conjunction of smaller formulae, it can be converted relatively easily to conjunctive normal form (CNF), at least much more easily than its negation $N = \neg\mathcal{B}^{\mathrm{Z}}$. The CNF of N, which is structurally equivalent to the disjunctive normal form of $\mathcal{B}^{\mathrm{Z}}$, can be understood as an implicit enumeration of all valid orders. It seems hardly surprising that the enumeration-of-orders view is less efficient than the constraint view using $\mathcal{B}^{\mathrm{Z}}$.

We now concentrate on the remaining open question concerning the consistency of the supplementing process.

4.2. Ordering Dependencies in the Supplementing Process

At first, we investigate dependencies on the ordering in which the supplementing steps are performed. As the relation $\longrightarrow_S$ is terminating, it suffices to show that $\longrightarrow_S$ is locally confluent to assure (global) confluence and thus ordering independence.

LEMMA 4.8. *If $\longrightarrow_S$ is locally confluent, every order O has a unique $\longrightarrow_S$-normal form.*

Proof. This is just a variation of Newman's Diamond Lemma [14] applied to the terminating reduction relation $\longrightarrow_S$.

As we want to avoid the encoding of different states during the supplementing process in a modal or non-monotonic logic, we have to resort to constructs available in propositional logic. But a purely propositional formalization of the local confluence property of relation $\longrightarrow_S$ is hard to give, as the number of supplementing steps and thus supplementing possibilities can be quite large.

Therefore, we consider a stronger precondition by limiting the number of reduction steps:

DEFINITION 4.9. *The relation $\longrightarrow_S$ is n-step locally confluent if for all $O, O_1, O_2 \subseteq \mathcal{C}$ there is an $O_{12} \subseteq \mathcal{C}$ with*

$$O_1 \longleftarrow_S O \longrightarrow_S O_2 \quad \Rightarrow \quad O_1 \longrightarrow_S^{\leq n} O_{12} \longleftarrow_S^{\leq n} O_2 \ .$$

NOTE 4.10. *Obviously n-step local confluence implies local confluence.*

Let us now concentrate on developing a criterion describing 1-step local confluence. Before we can give this criterion, we need the following

DEFINITION 4.11. *Let $F \in \mathcal{F}(\mathcal{C})$, $x, y \in \mathcal{C}$ and $k \in \{\top, \bot\}$.*[8] *The restriction $F|_{x=k}$ is defined as the homomorphic extension of*

$$y|_{x=k} \ = \ \begin{cases} k & \text{if } y = x \\ y & \text{otherwise} \end{cases}$$

[8] $\top$ denotes truth, $\bot$ falsity.

to the set $\mathcal{F}(\mathcal{C})$ of formulae.

Using this definition we can formalize properties about successor states in the supplementing process.

LEMMA 4.12. *Let $O \xrightarrow{c}_S O'$ and $F \in \mathcal{F}$. Then*

$$O \models F|_{c=\top} \quad \textit{iff} \quad O' \models F \ .$$

Proof. At first, note that $O' = O \dot{\cup} \{c\}$. We proceed by structural induction. The lemma is obvious for $F = \top$ and $F = \bot$. If $F = x \neq c$ is a propositional variable, then $x|_{c=\top} = x$ and $\chi_O(x) = \chi_{O'}(x)$. If $F = c$, we have $c|_{c=\top} = \top$ and $\chi_{O'}(c) = 1 = \chi_O(\top)$. Now, assume $F = \neg G$. Since $(\neg G)|_{c=\top} \equiv \neg(G|_{c=\top})$, the induction hypothesis already proves the lemma. The cases $F = G \vee H$ and $F = G \wedge H$ are handled accordingly using the fact that the restriction is a homomorphism.

Using Lemma 4.12 we can represent the effect of individual supplementing steps in propositional logic. The validity of a property F after adding code c is thus expressed by the validity of $F|_{c=\top}$ before supplementing c.

We therefore obtain

PROPOSITION 4.13. *The supplementing process (i.e., $\longrightarrow_S$) is 1-step locally confluent iff*

$$\bigwedge_{\substack{c_1,c_2 \in \mathcal{C} \\ c_1 \neq c_2}} \Bigg(\neg c_1 \wedge \neg c_2 \quad \Rightarrow \quad \bigwedge_{s \in \{L,R\}} \Bigg(s \quad \Rightarrow \quad \Big(\mathcal{Z}(c_1, s) \wedge \mathcal{Z}(c_2, s) \ \Rightarrow \ \mathcal{Z}(c_2, s)|_{c_1=\top} \wedge \mathcal{Z}(c_1, s)|_{c_2=\top} \Big) \Bigg) \Bigg) \tag{16}$$

is a tautology.

Proof. First we want to show that 1-step local confluence implies (16). We assume, for a contradiction, that (16) is not a tautology. Then there is a counterexample model O of steering type s_O, and codes c_1, c_2 with $c_1 \neq c_2$, such that $c_1, c_2 \notin O$, $O \models \mathcal{Z}(c_1, s_O)$, $O \models \mathcal{Z}(c_2, s_O)$ and $O \not\models \mathcal{Z}(c_2, s_O)|_{c_1=\top} \wedge \mathcal{Z}(c_1, s_O)|_{c_2=\top}$. Therefore, we also have $O \xrightarrow{c_1}_S O_1$ and $O \xrightarrow{c_2}_S O_2$ for appropriate O_1 and O_2. Moreover we get by using Lemma 4.12: $O_1 \not\xrightarrow{c_2}_S O_{12}$ or $O_2 \not\xrightarrow{c_1}_S O_{12}$. As $O_1, O_2 \neq O_{12}$ we have shown that $\longrightarrow_S$ is not 1-step locally confluent, a contradiction. The converse, namely that (16) implies 1-step local confluence is shown using a similar argumentation in reverse order.

Combining Lemma 4.8 and Note 4.10 with Proposition 4.13 we have a criterion at hand that captures the largest part of ordering dependencies occurring in our application.[9]

COROLLARY 4.14. *Assume (16) holds. Then the relation* $\longrightarrow_S$ *generates unique normal forms.*

4.3. Consistency of the Supplementing Process

The supplementing process adds codes to orders, thereby possibly converting non-constructible to constructible ones. The opposite, however, namely the conversion of constructible to non-constructible orders, may indicate a flaw in the process, especially since the supplementing formula (5) takes constructibility information $(\mathrm{C}^{\mathrm{D}}, \mathrm{C}^{\mathrm{I}})$ into consideration. We now want to elaborate on this.

Supplementing code $c \in \mathcal{C}$ while retaining constructibility is expressed by the validity of

$$\left(\mathcal{B}^{\mathrm{C}} \wedge \neg c \wedge \bigwedge_{s\in\{L,R\}} \Big(s \Rightarrow \mathcal{Z}(c,s)\Big)\right) \;\Rightarrow\; \mathcal{B}^{\mathrm{C}}|_{c=\top} \tag{17}$$

The subformula $\mathcal{B}^{\mathrm{C}}|_{c=\top}$ can be further simplified in this context by omitting certain subformulae as follows. For arbitrary formulae F we have $F \Rightarrow F|_{c=\top}$ as long as c does not occur negatively in F (this is easily proved by induction on F). As $\mathcal{B}^{\mathrm{C}}$ is a big conjunction $B_1 \wedge \cdots \wedge B_n$ we can drop those subformulae B_i not containing c negatively from the restriction $\mathcal{B}^{\mathrm{C}}|_{c=\top} = (B_1 \wedge \cdots \wedge B_n)|_{c=\top}$, without changing the validity of (17).

PROPOSITION 4.15. *The supplementing process preserves constructibility (in the abovementioned sense) iff*

$$\bigwedge_{c\in\mathcal{C}} \left(\left(\mathcal{B}^{\mathrm{C}} \wedge \neg c \wedge \bigwedge_{s\in\{L,R\}} \Big(s \Rightarrow \mathcal{Z}(c,s)\Big)\right) \;\Rightarrow\; \mathcal{B}^{\mathrm{C}}|_{c=\top}\right) \tag{18}$$

is a tautology.

Proof. Following the same idea as in the proof of Lemma 4.13.

[9] Multiple supplementing steps can be simulated accordingly, although at the cost of increased complexity.

5. Experimental Results

In order to solve the decision problems developed in the last section, we made experiments with different proving techniques, including resolution [16], term rewriting (using Stone polynomials to represent formulae [11]), BDDs [1, 5], and variations of the Davis-Putnam (DP) algorithm [7, 6]. Some of these techniques (e.g., BDDs) did not even permit a representation of the generated formulae (like $\mathcal{B}^{\mathrm{Z}}$). In general, run-times and memory requirements differed by orders of magnitude between these procedures. The DP algorithm turned out to be by far best-suited to solve the generated problems. This corroborates conclusions from [22].

Results of the experiments with a state-of-the-art implementation of the DP algorithm (SATO [20, 21]) are shown in Table I.

We used the test set

$$\mathcal{T} \quad = \quad \left\{ \mathcal{B}^{\mathrm{Z}} \wedge c \;\middle|\; c \in \mathcal{C} \right\} \cup \left\{ \mathcal{B}^{\mathrm{Z}} \wedge \neg c \;\middle|\; c \in \mathcal{C} \right\}$$

(representing the computation of necessary and inadmissible codes) as input for the propositional prover and split $\mathcal{T}$ into two disjoint subsets SAT and UNSAT containing satisfiable and unsatisfiable instances, respectively.

Table I. SATO average and maximal run-times for consistency proofs, in seconds.

data set	#vars	#clauses	$\bar{t}_{\mathrm{SAT}}$	$\bar{t}_{\mathrm{UNSAT}}$	$t_{\max}$
C-Class	1151	22036	0.49	0.38	0.68
Actros	1734	14264	0.19	0.13	0.21
Atego	1684	8973	0.12	0.09	0.15

The input data comprise the complete product documentation of the limousines of the Mercedes-Benz C-class, and parts of the product documentation of the Mercedes-Benz heavy and light trucks Actros and Atego, respectively (each restricted to one model line). To give an idea of the problem sizes, we have also included the number of propositional variables and the number of clauses in Table I.

Average run times for the elements of the two test sets SAT and UNSAT as well as the maximal run times are reported in seconds. All tests were performed on a SUN Ultra 1 running at 140 MHz under Solaris 2.5.1. Experiments using encodings of other consistency criteria resulted in similar prover run-times.

Most Davis-Putnam-style implementations require the input data to be in conjunctive normal form. We experimented with different transformation procedures including BDD-based techniques (as they are used, e.g., for two-level hardware minimization) and the traditional satisfiability-conserving transformation method due to Tseitin [18]. Neither method showed clear advantages, be it in conversion times or in the impact on prover behaviour. We nevertheless believe that the relationship between CNF transformation technique and proof complexity could be an interesting field of research.

The statistical information about the data sets shown in Table I refers to a BDD-based CNF transformation (which results in fewer variables, but more and longer clauses).

Further experiments with the DP algorithm using different data structures (lists vs. tries), different variable selection heuristics, and different types of search-space pruning, indicated that most improvements on the DP algorithm (e.g., backjumping [2] or sophisticated literal selection heuristics [9]) were vital to achieve reasonable run-times for the complete set of generated SAT-instances.

6. Conclusions

In this paper we developed the formalization of an existing industrial process in automotive product data management. The formalization allows us to automatically check consistency criteria that could not be handled so far. We were thus able to increase the quality of the product documentation. Moreover, by establishing interactive tools for product data management, we could provide a push-button technology that helps the professionals cope with the continuous change of data.

Of course, practical acceptance of such tools crucially depends on reasonable run times, even more when used interactively. The state of development of current satisfiability checkers proved to be sufficiently advanced for our project.

But the integration into the users' workflow plays an equally important role. Users interact with our information system through a graphical interface in terms of their every-day work. The product documentation is imported through a standard interface to the data base. So, only little additional knowledge is demanded from the operating personnel.

As the product documentation's constraints and rules were already presented in Boolean logic, we could restrict our work to formalizing the process-inherent logic. A further encoding of the data was not necessary in order to apply propositional provers. While it would be

comparatively easy to switch from one encoding to another, the critical issue is, however, how precise the industrial data and process are laid down initially.

Although the experiments were tailored to automotive product documentation, the underlying concepts should be applicable to neighboring fields of product data management (aviation, avionics, electronics, etc.), and especially to automotive supply industries.

Acknowledgments

The authors would like to thank Alfons Geser for fruitful discussions and suggestions during the early phase of the project. Special thanks are due to Dirk Bendrich from debis Systemhaus Industry GmbH (now with DaimlerChrysler AG), who was instrumental in initiating the project. Throughout the project, Alexander Krewitz (debis Systemhaus Industry GmbH) has been an extremely supportive industrial project leader. We are grateful to Ralph Wüsthofen (DaimlerChrysler AG) for patiently checking our processes and terminology from an industrial point of view.

References

1. S. B. Akers. Binary decision diagrams. In *IEEE Transactions on Computers*, volume C-27(6), pages 509–516, June 1978.
2. R. J. Bayardo Jr. and R. C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208. AAAI Press, 1997.
3. A. Borälv. The industrial success of verification tools based on Stålmarck's method. In O. Grumberg, editor, *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 7–10. Springer-Verlag, 1997.
4. J. P. Bowen and M. G. Hinchey. Seven more myths of formal methods: Dispelling industrial prejudices. In M. Naftalin, T. Denvir, and M. Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, volume 873 of *Lecture Notes in Computer Science*, pages 105–117. Springer-Verlag, 1994.
5. R. E. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computers*, volume C-35(8), pages 677–691, Aug. 1986.
6. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. In *Communications of the ACM*, volume 5, pages 394–397, July 1962.
7. M. Davis and H. Putnam. A computing procedure for quantification theory. In *Journal of the ACM*, volume 7, pages 201–215, 1960.
8. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Formal Models and Semantics*, volume 2 of *Handbook of Theoretical Computer Science*, chapter 6. Elsevier, 1990.

9. J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, May 1995.
10. A. Geser and W. Küchlin. Structured formal verification of a fragment of the IBM 390 Clock Chip. Technical Report 97-50, RISC-Linz Report Series, Schloß Hagenberg bei Linz, Austria, Oct. 1997.
11. J. Hsiang. *Topics in Automated Theorem Proving and Program Generation*. PhD thesis, University of Illinois, Urbana, Illinois, Dec. 1982.
12. F. E. Marschner. Practical challenges for industrial formal verification tools. In O. Grumberg, editor, *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 1–2. Springer-Verlag, 1997.
13. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, 1993.
14. M. H. A. Newman. On theories with a combinatorial definition of "equivalence". In *Annals of Mathematics*, volume 43, pages 223–243. Princeton University Press, 1942.
15. R. Pugliese and E. Tronci. Automatic verification of a hydroelectric power plant. In M.-C. Gaudel and J. Woodcock, editors, *FME'96: Industrial Benefit and Advances in Formal Methods*, volume 1051 of *Lecture Notes in Computer Science*, pages 425–444. Springer-Verlag, 1996.
16. J. A. Robinson. A machine-oriented logic based on the resolution principle. In *Journal of the ACM*, volume 12, pages 23–41, 1965.
17. H. Saiedian. An invitation to formal methods. In E. A. Parrish, editor, *Computer*, volume 29, pages 16–30. IEEE Computer Society, Apr. 1996.
18. G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Silenko, editor, *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1970.
19. A. L. Turk, S. T. Probst, and G. J. Powers. Verification of a chemical process leak test procedure. In O. Grumberg, editor, *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 84–94. Springer-Verlag, 1997.
20. H. Zhang. SATO: A decision procedure for propositional logic. In *Association for Automated Reasoning Newsletter*, volume 22, pages 1–3, Mar. 1993.
21. H. Zhang. SATO: An efficient propositional prover. In *CADE'97: 14th International Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.
22. H. Zhang and M. Stickel. Implementing the Davis-Putnam algorithm by tries. Technical report, Department of Computer Science, The University of Iowa, Iowa City, IA, Aug. 1994.