Maximilian Eibl, Martin Gaedke. (Hrsg.): INFORMATIK 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 1499

Adapting Binary Decision Diagrams for Visualizing Product Configuration Data

Daniel Bischoff[1] and Wolfgang Küchlin[2]

Abstract: This paper deals with the challenges of visualizing and understanding complex interacting Boolean formulæ for selecting parts in an automotive Bill-of-Materials (BoM). Our approach targets entire BoM positions containing all variants of a part, each with its own selection formula. A part variant is needed for a car if the selection formula evaluates to true under the option list (feature list) which defines the car variant. Understanding the formulæ is critical when editing or when trying to analyze and explain a bug, but it is non-trivial. SAT-solving is commonly used to *detect* bugs, but *explaining* the cause of bugs is a different matter. Our approach is to visualize all selection alternatives in a single diagram based on an adaptation of binary decision diagrams (BDDs). We also visualize the influence of the configuration constraints for car variants on the selection diagrams and show how they can help to reduce their size. Based on this method we implemented a visualization tool which additionally serves as a visual formula editor.

Keywords: Software Product Lines, Binary Decision Diagrams, Visualization

1 Introduction – Structure

Premium car manufacturers such as Daimler produce large numbers of highly customizable products in product lines out of parts that exist in many possible variations. Variability drivers are technical restrictions, sales options serving specific customer demands, legal requirements of the distribution country, marketing strategies, etc.

Product documentation both defines how a customer can configure a car order out of all available options, and which (variants of) parts must be selected from the overall Bill-of-Materials (BoM) to build each order. Automotive product configuration is therefore organized on two levels. High Level Configuration uses Boolean constraints to define a subspace of valid option combinations (car variants) within the Cartesian product of all options. Low Level Configuration uses Boolean formulæ to select from the BoM the right variant of each part based on the option list which defines each car variant.

Note that a part does not need to be a physical object in this context, since the software of a car's electronic control units (ECU) must also be adapted in order to fit the individual

[1] Daimler AG Group Research & MBC Development, 89073 Ulm, Germany, daniel.bischoff@daimler.com
[2] Universität Tübingen, Mathematisch-Naturwissenschaftliche Fakultät, Wilhelm-Schickard-Institut für Informatik, Symbolisches Rechnen, Sand 14, 72076 Tübingen, Germany, wolfgang.kuechlin@uni-tuebingen.de

doi:10.18420/in2017_149

car variant. One way to adapt the software is to use parameters to enable or disable certain functions, or to adapt the behavior of the ECU in general. The ECU is configured through predefined settings when an individual car is manufactured. These settings have to be selected for the respective car variants, based on the car's option list.

Since the features of a car are selected by a customer in the form of sales options, the terms "option" and "feature" can be used interchangeably. In state-of-the-art approaches for software variability management the Feature Oriented Domain Analysis (FODA) approach is used to model the problem space, whereas parameters and their values are part of the solution space [Ap13]. Feature models can be translated into Boolean formulæ [Ba05]. The vehicle variants, which result as possible selections of the feature model, are then mapped to the solution space.

The current approach to describe such mappings is by the use of Boolean formulæ. Visualizing and checking those formulæ is the subject of this paper. Broadly speaking, we present a way to visualize the underlying relations so that it becomes easy to check that they are indeed mappings, or to pinpoint defects in the formulæ.

The running example for this paper is as follows: Suppose one has implemented a special highway light software. This activates high beam mode as soon as the car's speed exceeds a specific limit. Since the limit has to be adapted to different car variants, the activation speed is a parameter. There are four Boolean features of the car which influence the activation speed. First it matters where the car is to get a MOT[3] approval, denoted here by the two features "GER" and "USA" respectively. The feature "NAVI" is used to indicate whether the car is equipped with a satellite navigation system which can help to recognize highways. One might also be able to predict whether the current road is a highway via a camera "CAM". We will use just those four features to influence our activation speed parameter. In practice, complex parameter configurations with more factors have to be taken into account.

Section 2 describes the mapping problem in detail and outlines the complexity. Section 3 presents different approaches for visualizing Boolean formulæ. Section 4 describes how our approach deals with feature constraints that limit the configuration space in practice.

2 The Problem – Configuration of Alternatives

The challenge at hand consists of mapping each vehicle variant to the correct parameter value. Due to the high number of possible variants this is done by functions (given as Boolean formulæ). These cover entire subsets of cars, rather than listing individual cars and their parameter values in variant tables. A typical configuration table shown in Tab. 1, which corresponds to a BoM position, holds all four possible values for our activation speed parameter p, consisting of values $70, 80, 90, 100$ and Boolean selection formulæ $\varphi_{70}, \ldots, \varphi_{100}$. We use upper case letters to denote vehicle features, $\wedge$ for conjunction, $\vee$ for

[3] Ministry of Transportation

disjunction, $\rightarrow$ for implication and $\neg$ for negation. $\top$ denotes the constant with value *true* and $\bot$ denotes *false*.

value		selection formula
$p = 70$	$\varphi_{70} =$	$(\neg GER \wedge NAVI \wedge CAM) \vee (USA \wedge GER \wedge NAVI \wedge CAM)$
$p = 80$	$\varphi_{80} =$	$(\neg USA \wedge GER \wedge NAVI \wedge CAM) \vee (\neg GER \wedge \neg NAVI \wedge CAM)$ $\vee(\neg GER \wedge \neg CAM \wedge NAVI) \vee (USA \wedge CAM \wedge \neg NAVI)$
$p = 90$	$\varphi_{90} =$	$(\neg USA \wedge \neg GER \wedge NAVI \wedge \neg CAM) \vee (\neg USA \wedge GER \wedge \neg NAVI \wedge CAM)$ $\vee(USA \wedge \neg NAVI \wedge \neg CAM)$
$p = 100$	$\varphi_{100} =$	$(\neg USA \wedge GER \wedge NAVI \wedge \neg CAM) \vee (\neg USA \wedge \neg NAVI \wedge \neg CAM)$

Tab. 1: Example configuration for a parameter with four possible values.

These formulæ denote a relation between vehicle variants and the parameter values. However, additional properties are required:

1. Since the parameter can only hold one value at a time, it is expected that the different options are alternatives.
2. The formulæ need to assign a value for every variant.

Property 1 requires the relation to be right-unique, Property 2 requires left-totality. I.e., in combination the selection formulæ have to be constructed in such a way that for all assignments of the vehicle features exactly one formula evaluates to true.[4] These properties ensure that the relation is a (total) mapping. More formally we can write this as

$$\underset{v \in 2^F}{\forall} \left(\sum_i \texttt{int}(\varphi_i|_v) = 1 \right). \tag{1}$$

- F is the set of vehicle features (here $F = \{GER, USA, NAVI, CAM\}$).
- `int` is a function that transforms a Boolean constant to an integer, i.e. $\texttt{int}(\top) = 1$ and $\texttt{int}(\bot) = 0$.
- $\varphi_i|_v$ denotes the evaluation of φ_i given the vehicle feature selection in v.
- 2^F is the power set of F and therefore a $v \in 2^F$ is a vehicle variation. If a vehicle feature $f \in F$ is in v, it is included in the vehicle variant, else it is absent.

$$f|_v = \begin{cases} \top \text{ if } f \in v \\ \bot \text{ otherwise} \end{cases} \tag{2}$$

  In this way all variables in the various φ are assigned a value and the $\varphi|_v$ evaluate to a constant.

[4] Here this is not the case in order to show how our approach deals with those errors.

For simplicity, the given example is kept short as it is used throughout the rest of the paper. However, one can easily imagine that such configuration tables can become large and the corresponding formulæ can become quite complex. Large and complex selection formulæ are hard to read and even harder to maintain for a human documentarist [HB05].

The process of error checking, i.e. ensuring that exactly one formula is true in each table for all possible assignments, is error-prone, if not outright impossible to do by hand for complex formulæ. Moreover, there are as many formulæ as there are parts in the BoM, between about 30,000 and 100,000 in practice. Fortunately, this property can be efficiently checked with a SAT-solver even for large numbers of large formulæ [KS00; MC03; SKK03].

However, there is still the problem of explaining each error to the user – a terse answer of SAT or UNSAT is hardly sufficient — and there is the related problem of supporting the user in developing or correcting these complex formulæ. In addition, an approach like in [SKK03] reports errors on the level of individual position variants and does not provide a comprehensive view of the entire situation at the position. A SAT solver will also report an overlap error in the form of any single car which selects two part variants, but it will not provide a representation of *all* cars which produce the same error.

The next section will show different approaches to visualize such configuration tables in order to make checking for errors easy.

3 Binary Decision Diagrams

The well known Binary Decision Diagrams (BDDs) [Ak78] provide a graphical representation of Boolean formulæ. The representation is even unique (within a Boolean equivalence class) if Reduced Ordered BDDs (ROBDDs) are used [Br92]. From an ROBDD the user can read off the answer to the following question: "which *true* / *false* decision do I have to make for each Boolean variable in a predetermined order to reach a result of *true* (resp. *false*) for the value of the formula?"

If we wanted to draw OBDDs for the four formulæ from Tab. 1, we could do this for example with the arbitrary ordering $CAM < NAVI < GER < USA$ as seen in Fig. 1. One could also combine these OBBDs into a shared OBDD (cf. [Kn09]) depicted in Fig. 2. These BDDs are interpreted by starting at the root of the tree, following the dashed line if a variable is assigned *false*, and following the solid line for *true*. It is still very hard to see if there is any assignment (i.e. car) where multiple formulæ or none at all evaluate to *true*. Therefore these traditional visualization approaches are not suitable for our problem.

Instead we propose a multi terminal BDD approach (as in [Cl93] and [Ba93]). For this purpose we construct a single formula by combining the selection formulæ φ_i from Tab. 1 and new variables $P_{70}, P_{80}, P_{90}, P_{100}$. Variable P_i encodes the fact that parameter P is set to value i. We call the result *parameter formula* since it describes all possible values and their

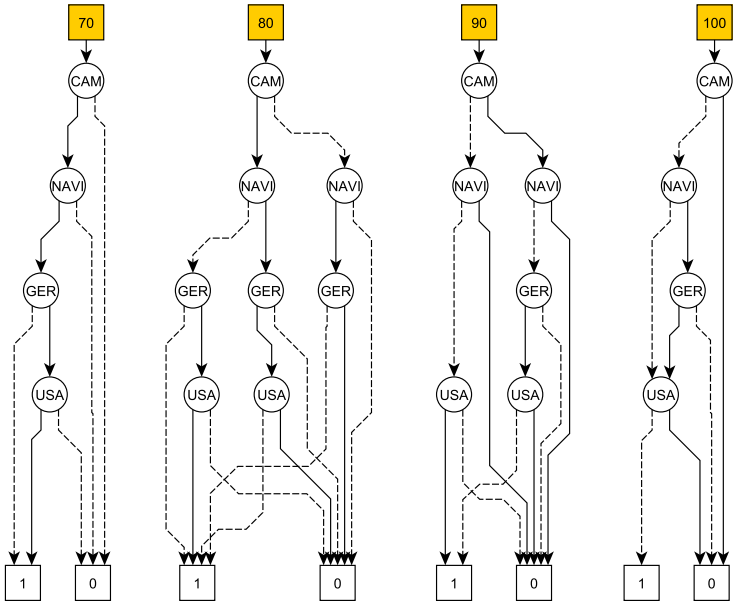

Fig. 1: OBDDs for $\varphi_{70}, \varphi_{80}, \varphi_{90}, \varphi_{100}$. The numbered roots show where each BDD starts.

conditions for the parameter. In general the parameter formula ψ for a parameter with some designated values is constructed as follows:

$$\psi = \bigwedge_i (\varphi_i \rightarrow P_i) = \bigwedge_i (\neg\varphi_i \vee P_i) \tag{3}$$

This ensures that whenever a selection formula evaluates to *true*, the corresponding part variable is also forced to *true*. In our case from Tab. 1 this results in

$$\psi = (\neg\varphi_{70} \vee P_{70}) \wedge (\neg\varphi_{80} \vee P_{80}) \wedge (\neg\varphi_{90} \vee P_{90}) \wedge (\neg\varphi_{100} \vee P_{100}). \tag{4}$$

When one draws the BDD for ψ with respect to the variable order

$$(CAM, NAVI, GER, USA, P_{70}, P_{80}, P_{90}, P_{100}).$$

the result is a regular BDD where the lower half (shown in Fig. 3) contains the new variables. To make the transition towards a multi terminal BDD easier to see, we removed the negative terminal in Fig. 4.

In Fig. 4 one can clearly see that a satisfying assignment on the leftmost path has to assign $P_{70} = true$ for example. On the rightmost path none of the P are relevant, which can only be the case if none of the φ in ψ are satisfied.

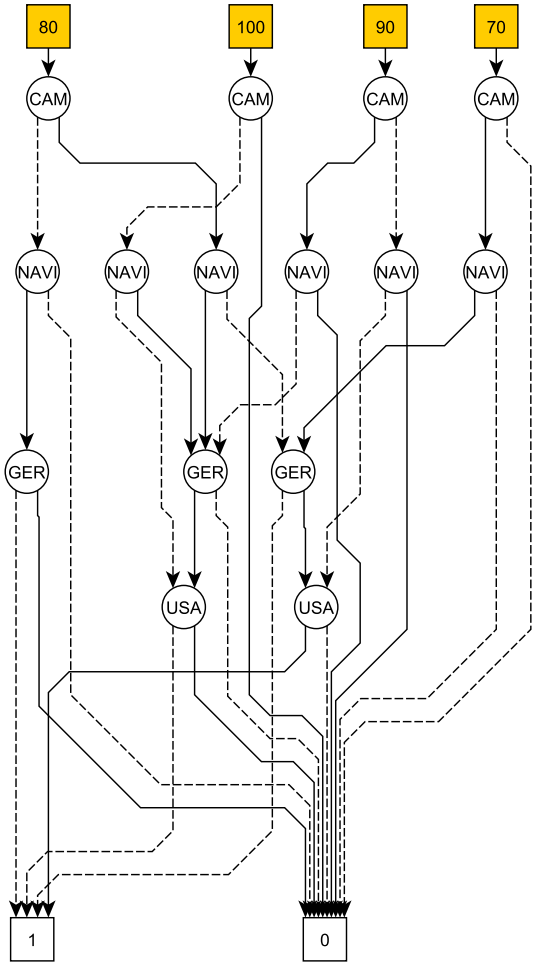

Fig. 2: A shared OBDD for all formulæ from Tab. 1. The numbered nodes at the top show where each formula's BDD starts.

To compile the multi terminal BDD we stop drawing after the feature level and form terminal nodes (cf. Fig. 5). Each terminal node represents a formula where the features $CAM, NAVI, GER, USA$ are already assigned a value and therefore this formula only consists of the variables $P_{70}, \ldots, P_{100}$. In each terminal node we write the variables which have to be set to true in order to satisfy the corresponding formula. This might be none[5],

[5] If none of the P_i have to be assigned a specific value in order to satisfy ψ, the formula is already *true* at this point. I.e. the node is the positive terminal of the BDD. This is the case only when all the φ_i are false.

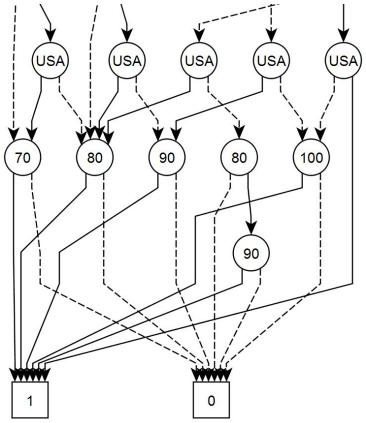

Fig. 3: The lower half of a BDD for ψ.

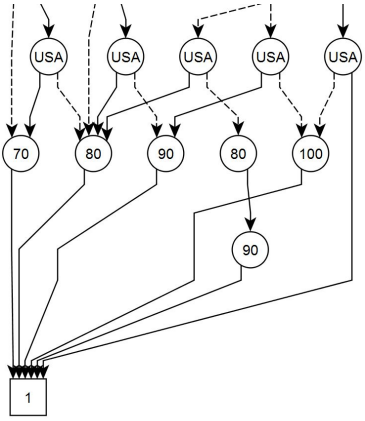

Fig. 4: The lower half of a BDD for ψ without the negative terminal.

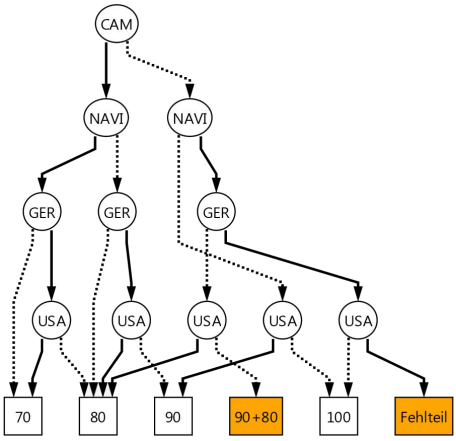

Fig. 5: A multi terminal BDD visualizing the formulæ from Tab. 1 (*Fehlteil* is German for missing part).

one or many of the P_i. Note that it is not possible to falsify ψ without assigning the P_i. The result for the formulæ from Tab. 1 is shown in Fig. 5.

In this visualization one can clearly see that there is an assignment for which two formulæ evaluate to true, hence the terminal labelled “90+80”. It is also obvious that there is an assignment for which none of the formulæ evaluate to true, hence the terminal that reads *Fehlteil* (missing part). One can also see which assignments lead to these situations.

In our opinion the visualization using the multi terminal approach is superior for the task at

hand compared to the standard BDD approaches. In the next section we will show how our approach deals with additional configuration constraints that limit the variability space of possible cars.

4 High Level Configuration

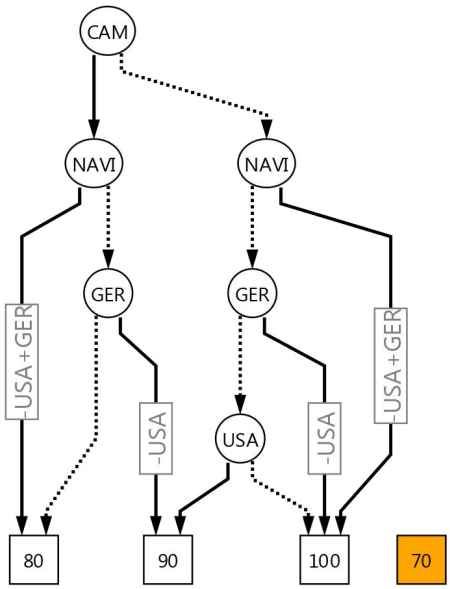

Fig. 6: A labeled edge multi terminal BDD visualizing the formulæ from Tab. 1 under the HLC.

Real world vehicle features are subject to configuration constraints. In the automotive industry, these are known e.g. as Model Description (VW), Vehicle Description Summary (General Motors), or Product Overview (Daimler: *Produktübersicht*). In line with the FODA approach, the term High Level Configuration (HLC) is used [Ha98] for the problem of configuring a vehicle from sales options, in contrast to the Low Level Configuration problem of configuring an individual vehicle from the set of all materials in the BoM. HLC dependencies are complicated and are organized as a set of Boolean constraints (several thousand in practice). Tab. 1 showed an example without an HLC, i.e. all assignments to the variables CAM, NAVI, GER, USA were possible without limitations. Let us assume an HLC where GER and USA exclude each other because we aim to produce cars for single countries only. Further we assume that NAVI implies GER because there are only road maps for Germany available. The HLC can then be written as follows:

$$(\neg USA \vee \neg GER) \wedge (\neg NAVI \vee GER). \tag{5}$$

We can now draw our BDD while respecting the HLC by conjugating this formula to ψ.

Since some feature combinations are forbidden by the HLC, there is no reason to draw them. Instead when the HLC forces a variable assignment, we label the corresponding edge with that assignment. In Fig. 6 an assignment of *true* to $NAVI$ forces the assignment of *true* to GER as well, which in turn forces the assignment of *false* to variable USA. Therefore the positive edge of the NAVI-nodes is labeled with $-USA + GER$. This decreases the overall size of the BDD significantly.

5 An Editor for Mappings

Based on the visualization techniques presented in Sect. 3 and Sect. 4 we implemented a tool to draw diagrams based on configuration formulæ. The resulting graph is interactive and can be manipulated in order to fix problems like the ones depicted in Fig. 5 and Fig. 6. The corrected graph can be translated back into Boolean formulæ. For example the formula φ_{80} can be extracted from Fig. 5 as follows:

1. For each terminal containing "80" find all paths to the root. Each path represents an assignment. In Fig. 5 there are five such paths. They are denoted by $\psi_1, \psi_2, \psi_3, \psi_4, \psi_5$. For example the leftmost path is $\psi_1 = CAM \wedge NAVI \wedge GER \wedge \neg USA$.
2. The formula φ_{80} is the disjunction of these assignments.
   $\varphi_{80} = \psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4 \vee \psi_5 = (CAM \wedge NAVI \wedge GER \wedge \neg USA) \vee \psi_2 \vee \psi_3 \vee \psi_4 \vee \psi_5$

With this technique all underlying formulæ can be computed from a BDD and drawing a BDD based on the extracted formulæ reproduces the original BDD since BDDs are a canonical representation of a formula.

6 Conclusion

We have shown how Boolean formulæ are used in automotive product documentation. In Chapter 3 we demonstrated how traditional visualization methods fail to make error checking easy for human documentarists. We presented our approach to visualize multiple related formulæ in a single diagram and showed how the visualization helps to make parts selection rules including their bugs transparent and understandable for a human. In Chapter 4 we also showed how our approach deals with restrictions to the space of variations and how they even help to make diagrams smaller and more readable.

7 Future Work

We plan further research along the following lines:

- Test the existing method with real data in a larger case study.
- Develop algorithms to deal with complex high level configurations efficiently[6].
- Allow the introduction of self defined features[7] (shortcuts), which are expressed in terms of already existing features. Can they help to further decrease the size of the diagrams and increase readability?
- Additionally we are planning to evaluate the proposed method with practitioners. Our goal is to learn from the evaluation and to further improve our method, while already improving the data quality of the product configuration data.

References

[Ak78] Akers, S. B.: Binary decision diagrams. IEEE Trans. Computers 27/6, pp. 509–516, 1978.

[Ap13] Apel, S.; Batory, D.; Kästner, C.; Saake, G.: Feature-Oriented Software Product Lines. Springer, 2013.

[Ba05] Batory, D.: Feature models, grammars, and propositional formulas. In: International Conference on Software Product Lines. Springer, pp. 7–20, 2005.

[Ba93] Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; Somenzi, F.: Algebraic decision diagrams and their applications. In: Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on. IEEE, pp. 188–191, 1993.

[Br92] Bryant, R. E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys (CSUR) 24/3, pp. 293–318, 1992.

[Cl93] Clarke, E. M.; McMillan, K. L.; Zhao, X.; Fujita, M.; Yang, J.: Spectral transforms for large boolean functions with applications to technology mapping. In: Proceedings of the 30th international Design Automation Conference. ACM, pp. 54–60, 1993.

[Ha98] Haag, A.: Sales configuration in business processes. IEEE Intelligent Systems and their Applications 13/4, pp. 78–85, 1998, ISSN: 1094-7167.

[HB05] Hami-Nobari, S.; Blessing, L.: Effect-oriented Description of Variant Rich Products. In: DS 35: Proceedings ICED 05, the 15th International Conference on Engineering Design, Melbourne, Australia, 15.-18.08. 2005. 2005.

[6] In this paper it was not yet discussed how to compute the labels shown in Fig. 6 due to the complexity of the matter.

[7] in contrary to the features defined by the HLC

[Kn09] Knuth, D.: The Art of Computer Programming: Bitwise Tricks & Techniques; Binary Decision Diagrams, volume 4, fascicle 1, 2009.

[KS00] Küchlin, W.; Sinz, C.: Proving consistency assertions for automotive product data management. Journal of Automated Reasoning 24/1, pp. 145–163, 2000.

[MC03] Mannion, M.; Camara, J.: Theorem proving for product line model verification. In: International Workshop on Software Product-Family Engineering. Springer, pp. 211–224, 2003.

[SKK03] Sinz, C.; Kaiser, A.; Küchlin, W.: Formal methods for the validation of automotive product configuration data. AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing 17/01, pp. 75–97, 2003.