XV Jornadas de Ingeniería del Software y Bases de Datos
JISBD 2006
José Riquelme - Pere Botella (Eds)

A SURVEY ON THE AUTOMATED ANALYSES OF FEATURE MODELS

David Benavides, Antonio Ruiz–Cortés, Pablo Trinidad and Sergio Segura

Dpto. de Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Av. de la Reina Mercedes S/N, 41012 Seville, Spain
e-mail: {benavides, trinidad, aruiz, sergio}@tdg.lsi.us.es

Palabras clave: Software Product Lines, Domain Engineering, Feature Models

Resumen. *Feature models are one of the most important assets in software product line engineering when capturing variability. Although the analyses of feature models was stated as challenging problem since they were fist proposed, it has not been fully achieved yet. In this paper we survey the state of the art on the automated analyses of feature models outlining the main advances made until now. We conclude that our proposal based on constraint programming is one of the most promising ones since it supports all the operations identified in this paper over feature models.*

1 Introduction

The industrialization of the software engineering discipline is not achieved yet. We understand the industrialization of software as the capability of producing software systems that fulfill budget and quality in a prescribed way. In other industries this is achieved by producing product lines. Indeed, Boing, Dell or even Mc Donalds builds product lines of aircraft, computers or hamburgers. Why not *Software Product Lines* (SPLs)?

A key technical question that confronts software engineers is how to specify a particular product in an SPL. One of the possible solutions is to do it in terms of *features*, where a feature is an increment in product functionality. Designing a family of software systems in terms of features is more natural than doing it in terms of objects or classes because they are understood by all stakeholders [19]. While single software systems are specified in terms of features, software product lines are specified using feature models.

Feature models are recognized in the literature to be one of the most important contributions to SPL engineering [3]. A key task is to capture commonalities and variabilities among products and feature models are used to this end.

Although there has been advances in SPL engineering regarding feature models, a key technical aspect remains open: their automated analyses. In this paper we survey the state of the art on the automated analyses of feature models. First, we review the different

proposals of feature models in Section 2. Section 3 summarizes the operations that have been identified up to now. Section 4 lists the main contributions on the support of these operations and finally Section 5 presents our conclusions and future work.

2 Feature Models

A *Feature Model* (FM) represents all possible products of an SPL in a single model using features. A feature is an increment in product functionality [3, 17]. It can be used in different stages of development such as requirements engineering [16, 21] , architecture definition or code generation [1, 4] (feature oriented programming). A FM is a tree–like structure and consists of: *i*) relations between a parent feature and its child features. *ii*) cross–tree constraints that are typically inclusion or exclusion statements of the form "if feature F is included, then features X must also be included (or excluded)".

In 1990, Kang *et al.*[17] proposed the first FMs. However, despite years of research, there is no consensus and many extensions have been proposed since then. Most of the extensions are based on the relations allowed between parent and child features.

As an example, the automotive industry uses features to specify and build software for configurable cars. The software that goes into a car is determined by the car's features. To simplify we can use a simple example where we only consider the features of transmission type (automatic or manual), engine type (electric or gasoline), and the option of cruise control. See Figure 1 for this widely used example.

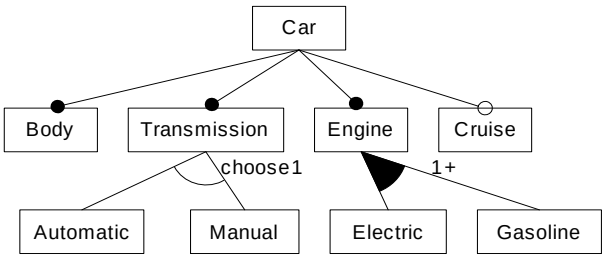

Figura 1: A sample feature model

First, the original FM notation, so–called FODA [17], was proposed. Later, Feature-RSEB [15] was presented as a FODA extension with an additional relation. Finally, Riebisch *et al.* [23] and Czarnecki *et al.* [12] proposed cardinality–based feature models where cardinalities (so–called multiplicities) were introduced.

2.1 Basic feature models

Four different kinds of relations where defined:

Mandatory. A child feature is said to be mandatory when it is required to appear when the parent feature appears. For instance, it is mandatory to have the body of a car.

Optional. A child feature is said to be optional when it can or not appear when the parent features appears. For instance, it is optional to have cruise control in a car.

Alternative. A set of child features are said to be alternative when only one child feature can be selected when the parent feature appears. For instance the transmission can only be automatic or manual.

Or–relation. A set of child features are said to have an or–relation with their parent when one or more sub features can be selected when the parent feature appears. For instance, the engine of a car can be electric, gasoline or both at the same time.

In addition, compatibility rules are allowed. Compatibility rules refer to additional cross–tree constraints to restrict feature combinations:

Excludes. A feature X excludes Y means that if X appears then Y should not appear and backwards.

Requires. A feature X requires Y means that if X appears then Ys should appear but not backwards.

2.2 Multiplicities and cardinalities

Another set of FMs is based on *cardinalities* (so–called *multiplicities*). Their main motivation was driven by practical applications [10] and "conceptual completness". Some authors [12, 23] proposed extending feature models with multiplicities. Mandatory and optional relations were generalized as feature cardinality relations, meanwhile alternative and or–relations where generalized in such a way that only a group relation was considered:

Feature cardinality. Optional and mandatory relations are generalized in the sense that the times that a feature appears is determined by its cardinality. Thus, an optional relation is substituted by a [0..1] feature cardinality relation and a mandatory relation by a [1..1] feature cardinality relation.

Multiplicity relation. A set of child features is said to have a multiplicity relation if a number of features can be selected when a parent appears. This number depends on the cardinality. Thus, an alternative relation is equivalent to a multiplicity relation with $\langle 1-1 \rangle$ cardinality and an or–relation is equivalent to a multiplicity relation with $\langle 1-N \rangle$ cardinality, being N the number of features in the relation.

2.3 Extended feature models

Basic FMs are suitable to express commonality and variability among different products in an SPL. However, sometimes it is necessary to extend FMs to include more information about features: feature attributes. These type of models where additional information is included are called extended feature models [1, 6, 13].

In fact, FODA already contemplated the inclusion of some additional information in FMs. For instance, relations between features and feature attributes were introduced. Later, [18] makes an explicit reference to what they called 'non–functional' features related to feature attributes. In addition, other group of authors have also proposed the inclusion

of attributes in feature models [6, 7, 3, 12, 24].

3 Operations on the automated analyses of feature models

The automated analyses of FMs was already identified as a critical task in the original FODA report [17, pag. 70]. However, there is still a lack of an automated support for FM analyses. As a matter of fact, there is not even a consensus on which are the operations included in this task. Next, we summarize the different proposals on the identification of operations to be performed. However, it is important to underline that in some of the works, although the operations were identified, no automated support was necessarily provided.

Determining feature model satisfiability. FM satisfiability (so–called FM validation) has been identified as a basic operation. Most relevant proposals call attention to this operation [3, 13, 14, 21, 28, 31, 32]. A FM is satisfiable (or valid) if at least one product is represented by the FM. A basic FM without cross–tree constraints can not be unsatisfiable (as the one of Figure 1). As a consequence, FM unsatisfiability in basic FMs may arise from the inclusion of cross–tree constraints in the model.

Finding a product. This operation returns a possible product if any. When the number of products in the FM is greater than one, the product obtained by this operation will depend on how the operation is implemented. For instance, finding a product in the FM of Figure 1 could retrieve this product considering that a product in a feature model is specified with both, leaf and compound features, which is something that is not still clear in the community [2]: $\{Car, Body, Transmission, Automatic, Engine, Electric\}$.

Obtaining all the products. This operation retrieves all possible products of a FM. Although this operation is sometimes practical when the number of products is low, it is often unfeasible to perform this operation for FMs with a big number of potential products. [1]

Calculating the number of products. This operation returns the number of products of a FM (12 in Figure 1). It reveals information about the flexibility and complexity of the SPL [6, 13, 14, 30]. A big number of potential products can reveal a more flexible and a more complex SPL.

This operation as well as the two previous ones can be used to check the satisfiability of a FM. A FM can be defined to be satisfiable iff the number of products of the FM is grater than zero.

Calculating variability. This operation returns the ratio between the number of products of a FM and 2^n being n the number of leaf features [6, 7]. Some proposals also identified a similar concept [6, 7, 13, 14, 30]. This operation can help in the analyses of a FMs since a big factor would represent a flexible product line meanwhile a low factor would represent a more compact product line. For instance, in Figure 1 $VF = 12/64 = 0.1875$.

Calculating Commonality. This operation is applied to a feature within a FM and

[1] This can be tested on http://www.tdg-seville.info/topics/spl

returns a value that represents the percentage of products where the feature appears. This calculation could help in deciding the order in which feature are going to be developed and can also be used to detect dead features [27]. For instance in Figure 1 $CF(Manual) = 50\%$. This operation is being used to identify the core components in a multiagents system architecture [22].

Optimizing. This operation returns the best products according to a given criterion[6] and it is chiefly useful when dealing with extended FMs where attributes are added to features. For instance, imagine that the cost of each feature is an attribute of the FM of Figure 1, we could be interested for the products with a minimum price.

Reducing the number of possible products. This operation reduces the possible products of a feature model by the selection (or deselection) of features. All previous operations are based on what we call static FMs, that is, FMs that do not change over time. However, in practice, it is recognized that FMs evolve over time and this change has to be contemplated. This type of evolution is defined in the sense that the number of potential product can be reduced in different stages. This is known in the literature as staged configurations [11] and it is also known as interactive configuration. During the selection process, a FM evolves according to a selection (or deselection) of features. The final state of this process would be a FM that represents only one product, or a FM that represents no products (an unsatisfiable FM). For instance, if we want a car with automatic and manual transmission at the same time (selection of features) the model become unsatisfiable. If we select cruise control in the feature model and deselect electric engine, the number of potential products is reduced.

Decision Propagation. This operation returns a FM where some features are automatically selected (or deselected). This operation makes sense during a staged configuration process when the user can select (or deselect) features and the platform should automatically propagate the decision all along the FM. For instance, in Figure 1, if automatic transmission is selected then manual transmission should be automatically deselected.

Dead features detection. A satisfiable FM can have internal inconsistencies which leads to the apparition of dead features. A dead feature is a feature that never appears in any legal configuration of a FM. Detecting those dead features is yet another challenge to be solved by automated FMs analyses platforms [27].

Providing explanations. This operation takes as input a feature model and returns an explanation when the FM is unsatisfiable or a dead feature is detected within the feature model. Debugging feature models is an error–prone task if it is performed manually. Usually, the automated tools will be able to answer that the feature model is unsatisfiable, however, finding the source of unsatisfiability is a key challenge for automated tools. Providing explanations in basic feature models is already a challenging problem and more it is doing so in extended feature models because attributes and relations among attributes appear.

Unfortunately, providing explanations is not a trivial problem because rather than telling where is the source of the unsatisfiability, it is desirable to know the minimal

source of unsatisfiability. For instance, if a feature model is unsatisfiable a valid explanation would be "the feature model is the source of unsatisfiability", however, the valid information is to point the relation that makes the feature model to be unsatisfiable. Furthermore, when the user faces an unsatisfiable feature model, it is desirable for the automated platform to generate a corrective explanation that, rather than focusing on what has caused the problem, explains what can be done to overcome it.

Simplification. This operation returns a normalized FM. One of the drawbacks of FMs is that the same set of products can be specified using different FMs with different relations among features. This leads to the need of a simplification (so–called normalization) process where any FM can be translated to a canonical representation [29, 14, 32]

4 Automated support on the automated analyses of feature models

The operations identified in previous Section can be performed using different approaches. Although automated analyses of FMs is still a recent research area and the connection between automated platforms and FMs has not been fully appreciated yet [1, 3], there are already some proposals to support automated analyses of FMs. These proposals can be divided in four main groups.

4.1 Propositional logic based analyses

There are some works in the literature that propose the translation of basic FMs into propositional formulas. Recognizing that connection has brought useful benefits since there are many off–the–shelf solvers that automatically analyze propositional formulas, therefore they can be used for the automated analyses of FMs.

Mannion [21] was the first to connect propositional formulas to FMs. In his work, FMs were used as requirements models for SPLs. Rules for translating such models into propositional formulas were provided and some operations were identified on the automated analyses of FMs.

Zhang *et. al* [32] took Mannion's proposal as base and suggested the use of an automated tool support based on the SVM System [2]. One of the main contributions of this paper is the definition of a process to be followed on the automated analyses of FMs that include the simplification of FMs. Moreover, a systematic way to detect dead features was provided as well.

Sun *et al.* [25] proposed a formalization of FMs using Z and the use of Alloy Analyzer [3] for the automated support of the analyses of FMs. Specially relevant is the identification and treatment of explanations when a FM is inconsistent.

Batory [1] summarized some of the advances up to date on the automated analyses of FMs. A coherent connection between FMs, grammars and propositional formulas was established. Basic FMs can be represented as context–free grammars plus propositional

[2] http://www.cs.cmu.edu/~ modelcheck/smv.html

[3] http://alloy.mit.edu

formulas what can be the base for the constructions of FMs compilers and domain specific languages. Grammars are a compact representation of propositional formulas and rules for translating grammars representing FMs into propositional formulas was provided. Furthermore, Logic Truth Maintenance Systems (a system that maintains the consequences of a propositional formula) was proposed to be built for the automated analyses of FMs. Such a system is built using a SAT solver and known boolean constraint propagation algorithms.

4.2 Description logic based analyses

To the best of our knowledge, there is only a work in the literature that propose the use of description logic reasoners for the automated analyses of FMs [31]. This proposal is based on the translation of FMs into an OWL DL ontology. OWL DL is an expressive, yet decidable sublanguage of OWL (Ontology Web Language). In that connection, it is possible to use automated tools such as RACER (Renamed ABox and Concept Expression Reasoner [4]) for the automated analyses of FMs.

4.3 Constraint programming based analyses

We have suggested the use of constraint programming for the automated analyses of FMs [5, 6, 7, 8, 27, 26]. A FM can be translated into a Constraint Satisfaction Problem (CSP) and using constraint programming techniques it is possible to leverage the automated analyses of FMs. Up to know, this is the only proposal that support the analyses of both cardinality–based FMs and extended FMs and therefore support the optimization operation.

4.4 Ad–hoc based analyses

There are some proposals in the literature where the conceptual underpinnings are not clearly exposed. We have decided to group such proposals as ad–hoc solutions.

Van Deursern *et. al* [14] proposed a feature description language to describe FMs. From this language, a FM algebra is described based on rules over the ASF+SDF Meta–Environment [20]. Using their system some operations such as, calculating the number of products, find inconsistencies and simplification of FMs are automatically performed. After the FODA report in 1990 where FMs where first presented, this is the first paper we have found that explicitly give a method for the automated analyses of FMs.

Cao *et al.* [9] presented ad–hoc algorithms for the automated analyses of FMs. Their algorithms are based on the translation of basic FMs into data structures that claim to be enough to obtain all the products as the base for some other operations. They also present a tool prototype based on their algorithm.

A summary of this survey is depicted in Table 1. There are four groups: Propositional logic (PL), Description Logic (DL), ad–hoc and Constraint Programing(CP) based analy-

[4] http://www.racer-systems.com/

ses. The first three rows of the table refer the support for basic, cardinality and extended feature models. Up to know, our proposal is the only one supporting both cardinality and extended feature models. Second group of rows refer to the identification (I) and Automated Support (AS) for the operations in the different proposals. Once again, we can see that our proposal has been the one that support more operations and plays a key role in the analyses of extended feature models [1, 2]. There are two operations which support have not been proposed yet but we are currently working on that and are having promising results [26, 27].

	PL		DL		ad–hoc		CP	
basic	+		+		+		+	
cardinality	-		-		-		+	
extended	-		-		-		+	
	I	AS	I	AS	I	AS	I	AS
consistency	[1, 21, 25, 32]	[1, 21, 25, 32]	[31]	[31]	[14, 9]	[14, 9]	[6]	[6]
finding	[1, 32]	[1, 32]	[31]	[31]	[14]	[14]	[6]	[6]
all products	[1, 21, 25, 32]	[1, 21, 25, 32]	-	-	[14, 9]	[14, 9]	[6]	[6]
#products	[21]	[21]	-	-	[14, 9]	[14, 9]	[6]	[6]
commonality	-	-	-	-	-	-	[6]	[6]
variability	-	-	-	-	-	-	[6]	[6]
optimization	-	-	-	-	-	-	[6]	[6]
reducing	[1, 21, 25, 32]	[1, 21, 25, 32]	[31]	[31]	[14, 9]	[14]	[6]	[6]
propagation	[1, 32]	[1, 32]	-	-	-	-	[6]	[6]
dead features	[32]	[32]	-	-	-	-	[26]	[26]
explanations	[1, 21, 25]	[1, 25]	[31]	[31]	-	-	[26]	-
simplification	[1, 25, 32]	[32]	-	-	[14]	[9]	-	-

Tabla 1: Summary of the proposals

5 Conclusions

The automated analyses of FMs is an ongoing research area, which involves much more operations that it could be initially hoped. Automating the analysis implies endows to every operation with a formal semantics. In this sense, constraint programming is a promising candidate. On the one hand, the declarativity of constraint programming eases to give a formal semantics to every analysis operation. On the other hand, current constraint solvers allow to build handy, efficient tools which offer support for every operation identified in this survey. Although other formalism and techniques could be used to implement a tool, the operations which involve reasoning on numerical attributes can be always interpreted as a constraint satisfaction problem. In fact, description logics reasoners and programming logic environments use algorithms from the constraint programming

area when they have to deal with numerical constraints. Thus, we conclude that our approach is a solid seed that can set up the basis for the construction of automated tool that leverage automated analyses on feature models.

Although we devise constraint programming as the most powerful tool for specification, we recognize that some operations can be computationally difficult to perform when constraint programming solvers are used [5, 8]. We are developing a complete reasoning framework to support other tools such as SAT or BDD solvers (solvers that work well with boolean variables) that in some specific operations can scale better than constraint programming solvers.

Acknowledgments

This work was partially supported by the Spanish Science and Education Ministry (MEC) under contract TIC2003-02737-C02-01 (AgilWeb). We would like to thank Don Batory and Salvador Trujillo for their helpful comments on an earlier draft of this paper.

REFERENCIAS

[1] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines Conference, LNCS 3714*, pages 7–20, 2005.

[2] D. Batory. Private correspondence. 2005–2006.

[3] D. Batory, D. Benavides, and A. Ruiz-Cortés. Automated analysis of feature models: Challenges ahead. *Communications of the ACM*, Conditionally accepted, 2006.

[4] D. Batory, J. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. *IEEE Trans. Software Eng.*, 30(6):355–371, 2004.

[5] D. Benavides, A. Ruiz-Cortés, B. Smith, Barry O'Sullivan, and P. Trinidad. Computational issues on the automated analyses of feature models using constraint programming. *International Journal of Software Engineering and Knowledge Engineering*, in preparation, 2006.

[6] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.

[7] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Using constraint programming to reason on feature models. In *The Seventeenth International Conference on Software Engineering and Knowledge Engineering, SEKE 2005*, 2005.

[8] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Using java csp solvers in the automated analyses of feature models. *LNCS*, to be asigned, 2006.

[9] F. Cao, B. Bryant, C. Burt, Z. Huang, R. Raje, A. Olson, and M. Auguston. Automating feature-oriented domain analysis. In *International Conference on Software Engineering Research and Practice (SERP'03)*, pages 944–949, June 2003.

[10] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker. Generative programming for embedded software: An industrial experience report. In *Generative Programming and Component Engineering, ACM SIGPLAN/SIGSOFT Conference, GPCE 2002*, pages 156–172, 2002.

[11] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *Proceedings of the Third Software Product Line Conference 2004*, pages 266–282. Springer, LNCS 3154, 2004.

[12] K. Czarnecki, S. Helsen, and U.W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[13] K. Czarnecki and P. Kim. Cardinality-based feature modeling and constraints: A progress report. In *Proceedings of the International Workshop on Software Factories At OOPSLA 2005*, 2005.

[14] A. van Deursen and P. Klint. Domain–specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002.

[15] M. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In *Proceedings of theFifthInternational Conference on Software Reuse*, pages 76–85, Canada, 1998.

[16] S. Jarzabek, Wai Chun Ong, and Hongyu Zhang. Handling variant requirements in domain modeling. *The Journal of Systems and Software*, 68(3):171–182, 2003.

[17] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[18] K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A feature–oriented reuse method with domain–specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[19] K.C. Kang, J. Lee, and P. Donohoe. Feature–Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, July/ August 2002.

[20] P. Klint. A meta-environment for generating programming environments. *ACM Trans. Softw. Eng. Methodol.*, 2(2):176–201, April 1993.

[21] M. Mannion. Using First-Order Logic for Product Line Model Validation. In *Proceedings of the Second Software Product Line Conference (SPLC2)*, LNCS 2379, pages 176–187, San Diego, CA, 2002. Springer.

[22] J. Peña, M. Hinchey, A. Ruiz-Cortés, and P. Trinidad. Building the core architecture of a multiagent system product line: With an example from a future nasa mission. In *7th International Workshop on Agent Oriented Software Engineering (AOSE'06)*, LNCS. Springer Verlag, 2006.

[23] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending feature diagrams with uml multiplicities. In *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, June 2002.

[24] D. Streitferdt, M. Riebisch, and I. Philippow. Details of formalized relations in feature models using ocl. In *Proceedings of 10th IEEE International Conference on Engineering of Computer–Based Systems (ECBS 2003), Huntsville, USA. IEEE Computer Society*, pages 45–54, 2003.

[25] J. Sun, H. Zhang, Y.F. Li, and H. Wang. Formal semantics and verification for feature modeling. In *Proceedings of the ICECSS05*, 2005.

[26] P. Trinidad, D. Benavides, and A. Ruiz-Cortés. Explanations for Agile Feature Modeling. In *Proceedings of the First International Workshop on Agile Product Line Engineering (APLE'06)*, Baltimore, MD, USA, 2006.

[27] P. Trinidad, D. Benavides, and A. Ruiz-Cortés. Isolated features detection in feature models. In *CAiSE Short Paper Proceedings*, 2006.

[28] T. von der Massen and H. Lichter. Requiline: A requirements engineering tool for software product lines. In F. van der Linden, editor, *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5)*, LNCS 3014, Siena, Italy, 2003. Springer Verlag.

[29] T. von der Massen and H. Lichter. Deficiencies in feature models. In Tomi Mannisto and Jan Bosch, editors, *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, 2004.

[30] T. von der Massen and H. Litcher. Determining the variation degree of feature models. In *Software Product Lines Conference, LNCS 3714*, pages 82–88, 2005.

[31] H. Wang, Y. Li, J. Sun, H. Zhang, and J. Pan. A semantic web approach to feature modeling and verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, November 2005.

[32] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. In J. Davies, editor, *ICFEM 2004*, volume 3308, pages 115–130. Springer–Verlag, 2004.