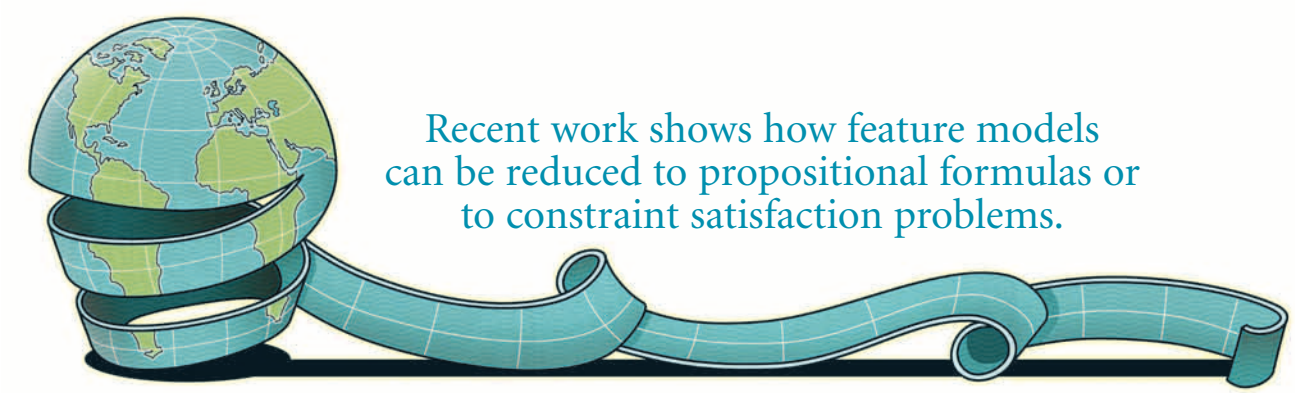
By Don Batory, David Benavides, and
Antonio Ruiz-Cortés

AUTOMATED ANALYSIS OF FEATURE MODELS: CHALLENGES AHEAD

A *feature* is an increment in product functionality. Features are commonly used to specify and distinguish products in product lines [8]. They communicate product functions in an easy-to-understand way, capture functionalities concisely, and help delineate the commonalities and variabilities of a domain. Features can have attributes (much like graphical user interface components can be customized by property lists), where the values of certain attributes are computed from the properties of other features (for example, the cost of a product is the sum of the costs of its constituent features). Features also often have constraints on their usage: the selection of one feature may preclude or require the selection of others.

Current tool support for feature models is ad hoc, offering little or no support for debugging feature models or optimizing feature selections. Recent work shows how feature models can be reduced to propositional formulas or to constraint satisfaction problems, for which off-the-shelf tools can validate properties of models (such as confirming that a given set of features is incompatible or compatible) or to optimize the selection of features (for example, performance) [1, 2, 4, 5, 10]. This opens up new possibilities for next-generation tools for specifying products in software product lines.



Recent work shows how feature models can be reduced to propositional formulas or to constraint satisfaction problems.

Of course, feature models are a front-end to a back-end synthesis technology that takes the output of a feature model (that is, a program specification) and converts it into the program itself. There are many technologies for doing this, and reviewing them is beyond the scope of this article. Our goal here is to alert readers to recent advances in formalizing feature models and to the challenges ahead in automating product specification and design.

Open Issues and a Research Agenda

Model Consistency. The automotive industry has feature models with up to 10,000 features. It is well known that these models are riddled with inconsistencies that are difficult to detect. As an elementary illustration, suppose a feature model requires that (1) if feature `A` is used then `B` must also be used `(A implies B)`, and (2) if feature `B` is used, feature `A` cannot be used `(B implies not A)`. Clearly there is an inconsistency: if `A` is true, we can conclude `A` is false. Such inconsistencies are rarely this simple in practice. The way they are discovered today is by accidentally stumbling over them: the correct set of features must be selected to expose the error. Unfortunately, the number of subformulas to examine is $O(2^n)$, where n is the number of variables in a formula. Are there automated ways to find model inconsistencies?

Explanations. Features can be automatically deselected by numerical constraints (such as performance). It is possible for users to specify constraints that are unsatisfiable (for example, the memory requirements of a program cannot exceed `x` and the program must have feature `Y`, where `memoryRequirements(Y)>x`). Explaining why there is no product for a given set of constraints, and perhaps more importantly, how the situation can be rectified is key. Finding a minimal number of violated constraints, which is vital to understandable explanations, is a difficult problem. Model diagnosis research may be relevant [12].

Model-Driven Development (MDD). Mapping feature selections in a feature model into other development artifacts (requirements, architecture, code modules, test cases, documentation) is fundamental to MDD. As an example of model and code integration, suppose the implementation of feature `F` makes a reference to a variable or method that is part of feature `G`. This means that if `F` is selected, then `G` must also be selected. It should not be possible to specify a product `P` where `F` is selected and `G` is not. That a feature model satisfies this constraint can be verified by a SAT solver. More generally, verifying that other program representations are consistent with their feature model is a significant research challenge [3, 7, 11].

Artificial Intelligence (AI) Configurators. Consider a product line of aircraft carriers. Each carrier may contain several different kinds of aircraft (short and long-range fighters), and each plane may be a member of a product line. The planes on a carrier impose constraints on the carrier's design. A web of customizable objects would be needed to describe a carrier (or other complex products) [6]. Feature models must be generalized to describe these "mega" products, and so too must tools that analyze and visualize these models. AI configurators, tools that configure constellations of objects, may be important for the analysis task [1, 9].

Performance Scalability. How well do SAT solvers, BDD tools, CSP solvers, and AI configurators perform with large models? (We can even imagine description logic-based reasoners being used to analyze feature models.) Even though there has been an enormous increase in computing power in the last decade, the problems of feature combinatorics remain NP-hard. Not all tools and approaches will perform equally well. Which tools should be used and when? Can the choice of which tools to use be made automatically to minimize the time to analyze feature models? Will it be necessary to integrate different solvers?

Conclusion

Validating and analyzing product specifications will have significant

practical payoffs. The benefits are tools that propagate constraints (so that incorrect specifications can be automatically detected), that provide explanations when design dead ends are reached (and how to fix such designs), and that automatically optimize configurations for specific needs (to simplify program designs). Exposing the theory that underlines feature models is central to this goal. Answering these challenges will require close cooperation between product line engineers and researchers. C

References

1. Asikainen, T., Männistö, T., and Soininen, T. Using a configurator for modelling and configuring software product lines based on feature models. In *Proceedings of the Workshop on Software Variability Management for Product Derivation, Software Product Line Conference (SPLC3)*, 2004.
2. Batory, D. Feature models, grammars, and propositional formulas. In *Proceedings of the Software Product Line Conference*, 2005.
3. Batory, D. and Thaker, S. Towards safe composition of product lines. Technical Report, Dept. Computer Sciences, University of Texas, TR-06-33, 2006.
4. Benavides, D., Trinidad, P., and Ruiz-Cortés, A. Automated reasoning on feature models. In *Proceedings of the Conference on Advanced Information Systems Engineering (CAISE)*. LNCS 3520, July 2005.
5. Benavides, D., Segura, S., Trinidad, P., and Ruiz-Cortés, A. Using Java CSP solvers in the automated analyses of feature models. In *Post-Proceedings of The Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE)*. LNCS 4143, 2006.
6. Czarnecki, K. and Kim, C.H.P. Cardinality-based feature modeling and constraints: A progress report. In *Proceedings of the OOPSLA Workshop on Software Factories*, 2005.
7. Czarnecki, K. and Pietroszek, K. Verifying feature-based model templates against well-formed OCL constraints. *Generative Programming and Component Engineering*, 2006.
8. Kang, K., Cohen, S., Hess, J., Nowak, W., and Peterson, S. Feature-oriented domain analysis (FODA) feasibility study. Technical Report, CMU/SEI-90TR-21, Nov. 1990.
9. Mittal, S. and Frayman, F. Towards a generic model of configuration tasks. In *Proceedings of the 11th International Conference on Artificial Intelligence*, 1989, 1391–1401.
10. Neema, S., Sztipanovits, J., and Karsai, G. Constraint-based design space exploration and model synthesis. In *Proceedings of EMSOFT 2003*, LNCS 2855, 290–305.
11. Pohl, K., Bockle, G., and Linden, F. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.
12. Reiter, R. A theory of diagnosis from first principles. *Artificial Intelligence 32*, 1 (1987), 57–96.

Don Batory (batory@cs.utexas.edu) is a professor in the Department of Computer Science at the University of Texas at Austin.

David Benavides (benavides@tdg.lsi.us.es) is a Ph.D. student in the Department of Computer Science Languages and Systems at the University of Seville in Spain.

Antonio Ruiz-Cortés (aruiz@tdg.lsi.us.es) is an associate professor in the Department of Computer Science Languages and Systems at the University of Seville in Spain.

This work was supported in part by NSF's Science of Design Project #CCF-0438786 and the Spanish Ministry of Science and Technology under grants TIC2003-02737-C02-01 and TIN2006-00472.