Scientific Writing

An Introduction | Thomas Thüm | May 19, 2021

Software Engineering
Programming Languages

ulm university universität uulm

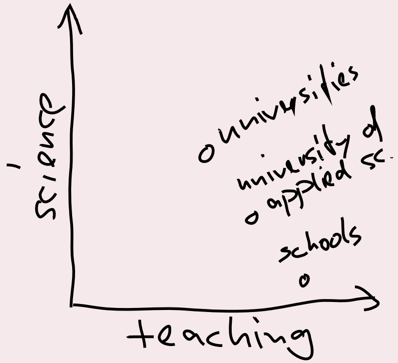
Lecture Overview

1. Scientific Writing and Peer Review
2. Structure in Writing
3. Clarity in Writing

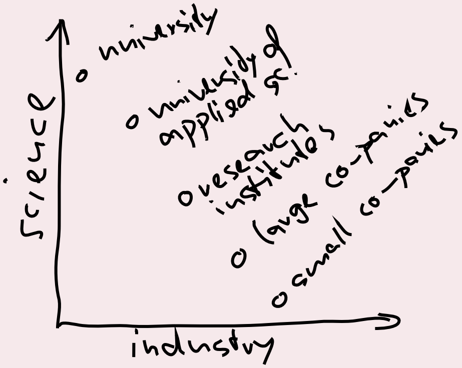
Scientific Writing and Peer Review

Scientific Writing: Why Science?

Science vs Teaching



Science vs Industry



Scientific Writing: Why Writing?

Research vs Writing

- research gives new insights
- insights need to be documented
- communication to achieve impact, get feedback, to initiate collaborations, get recognized
- research is never finished until it is published
- publication is the core process and ultimate result of scientific research

Does it Need to be Writing?

- Talking? Singing?
- Drawing? Showing?
- Presenting?
- Dancing?
- ...
- Influence of COVID-19?

Types of Scientific Contributions

Scientific Contributions

- theory: mathematical or machine-checked proofs
- conceptual contribution: algorithms, applications of algorithms, methodologies
- artifact: prototypical tool support, data/benchmark, challenges
- empirical evaluation: experiments, hypothesis testing
- survey: literature overview, identification of gaps, research roadmap

Kinds of Scientific Literature (Venues)

Literature without Peer Review

- books
- bachelor's, master's, Ph.D. theses
- technical report, magazine articles, blog post, tweet

Peer-Reviewed Literature

- journal article (typically no presentation)
- conference/symposium/workshop paper
- poster, extended abstract

In Ulm: at least three papers/articles at major conferences/journals for cumulative dissertation

Workshop vs Conference vs Journal

- workshop: work in progress, presentation with 10–20 participants, 4–8 pages, core community
- conference: validated research, presentation with 20–500 participants, 10–14 pages, wider community
- journal: results of 1-5 years, 15–50 pages, extensive review process, wide target audience, optional conference presentation in journal-first track

Watchout for Fake Conferences

German documentary on fake conferences

Criteria for Authorship

Association for Computing Machinery:

“Anyone listed as author on an ACM manuscript submission must meet all the following criteria:”

- “they have made **substantial intellectual contributions** to some components of the original work described in the manuscript; and
- they have **participated** in drafting and/or revision of the manuscript and
- they are **aware** the manuscript has been submitted for publication; and
- they agree to be held **accountable** for any issues relating to correctness or integrity of the work”

Association for Computing Machinery:

“ACM requires that all published papers include the names and affiliations of all authors listed on the paper, as well as provide accurate contact information to ACM as required in the ACM rights contract. ACM does not allow anonymous authors, and any papers published in the ACM Digital Library without author names and affiliations may be retracted by ACM. ACM authors have published under actual **pen names** using “independent consultant” as the author’s listed affiliation, but even under such circumstances ACM must be provided with contact information for such authors using a pen name, so that ACM will be able to reach all authors of published papers.”

https://www.acm.org/publications/policies/authorship

Scientific Peer Review

What is Peer Review?

- review: critical read and frank comments on your work
- peers: 3–4 external, independent researchers/experts (e.g., PhD students, PostDocs, professors)
- peer review: aim is objective evaluation and constructive feedback
- acceptance/rejection based on reviews
- central element in the process of research
- single-blind review: anonymous reviewers
- double-blind review: anonymous reviewers and authors

Why Peer Review?

- quality control: publishers, conferences, journals
- quality assessment: job offers, fund raising
- alternative: technical report without peer review + metrics on impact (only measurable after years)

Writing Peer Reviews

Structure of Reviews

- score (conferences): strong accept, (accept,) weak accept, (borderline,) weak reject, (reject,) strong reject
- score (journal): accept as is, minor revision, major revision, reject
- confidence: 5 (expert), 4 (high), 3 (medium), 2 (low), 1 (none)
- summary (1 paragraph), 4–7 keypoints in favor/against acceptance, major comments (3–5 paragraphs), minor comments (optional)

Exception: desk reject (score by program chairs, no detailed review)

Review Criteria

- significance: does the research address a relevant problem?
- clarity: is the paper well-written and structured?
- novelty: is the contribution over the state-of-the-art clear?
- correctness: are claims supported by proofs, tools, examples, experiments?
- reproducibility (same results with same data) and replicability (same res. with new data)
- more: soundness, illustration, presentation, self-containedness

Scientific Writing and Peer Review

Lessons Learned

- The need for scientific writing
- Types of scientific contributions and venues
- Authorship
- Peer review: goals, structure, criteria

Practice

- What is your motivation to learn about scientific writing?
- What are corner cases for authorship decisions?
- Why is peer review critical for research?
- Optional: watch German documentary on fake conferences

Lecture Contents

Structure in Writing

The Need for Structure

Motivation

How to encode a set of related thoughts and ideas as a linear stream of text?

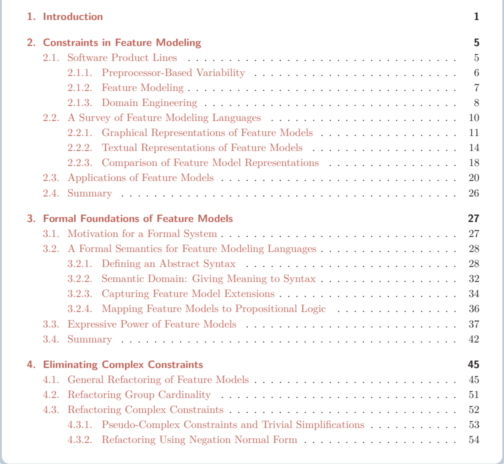
How to help readers to navigate in that text and read relevant parts?

Hint

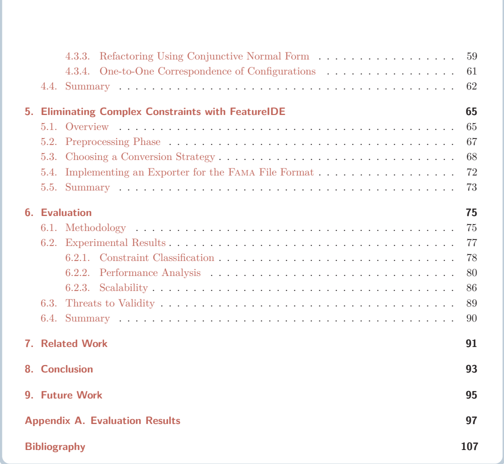
paper organization != research process

Top-Level Structure

A Master's Thesis [Knüppel 2016]



A Master's Thesis [Knüppel 2016]



Top-Level Structure

Typical Structure of a Paper

- Title and Authors
- Abstract
- Introduction
- Background
- (Problem Statement)
- Own Contribution
- (Implementation)
- Evaluation
- Related Work
- Conclusion
- (Acknowledgements)
- (Appendix)
- References

Comments

- adapt to your needs
- Background section can be replaced by Motivating Example section
- use problem statement if the problem is not well established
- concept may be split into separate sections
- in some communities: Related Work section after Background
- future work discussed in Conclusion section or in a dedicated section right afterwards called Future Work
- only theses and books: table of contents, list of figures / tables / code listings / abbreviations, ...
- extended abstracts have no structure

Title and Authors

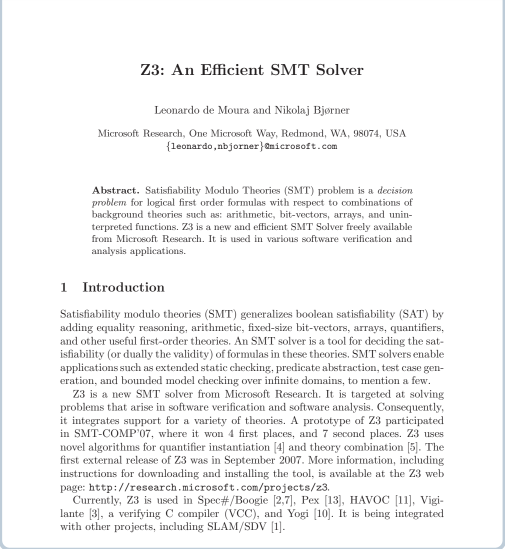
Title

- clear, informative, reflects aim and approach
- fewest possible words describing the contents
- prefer specific terms over general
- avoid abbreviations and jargon
- brainstorming or idea engineering for outstanding titles

Authors

- name: consistent writing, avoid umlauts
- affiliation, country
- optional: address, e-mail

Example from 2008 with 7000+ citations [Z3]



Z3: An Efficient SMT Solver

Leonardo de Moura and Nikolaj Bjørner

Microsoft Research, One Microsoft Way, Redmond, WA, 98074, USA
{leonardo,nbjorner}@microsoft.com

Abstract. Satisfiability Modulo Theories (SMT) problem is a *decision problem* for logical first order formulas with respect to combinations of background theories such as: arithmetic, bit-vectors, arrays, and uninterpreted functions. Z3 is a new and efficient SMT Solver freely available from Microsoft Research. It is used in various software verification and analysis applications.

1 Introduction

Satisfiability modulo theories (SMT) generalizes boolean satisfiability (SAT) by adding equality reasoning, arithmetic, fixed-size bit-vectors, arrays, quantifiers, and other useful first-order theories. An SMT solver is a tool for deciding the satisfiability (or dually the validity) of formulas in these theories. SMT solvers enable applications such as extended static checking, predicate abstraction, test case generation, and bounded model checking over infinite domains, to mention a few.

Z3 is a new SMT solver from Microsoft Research. It is targeted at solving problems that arise in software verification and software analysis. Consequently, it integrates support for a variety of theories. A prototype of Z3 participated in SMT-COMP'07, where it won 4 first places, and 7 second places. Z3 uses novel algorithms for quantifier instantiation [4] and theory combination [5]. The first external release of Z3 was in September 2007. More information, including instructions for downloading and installing the tool, is available at the Z3 web page: `http://research.microsoft.com/projects/z3`.

Currently, Z3 is used in Spec#/Boogie [2,7], Pex [13], HAVOC [11], Vigilante [3], a verifying C compiler (VCC), and Yogi [10]. It is being integrated with other projects, including SLAM/SDV [1].

Abstract and Keywords

Abstract (Zusammenfassung)

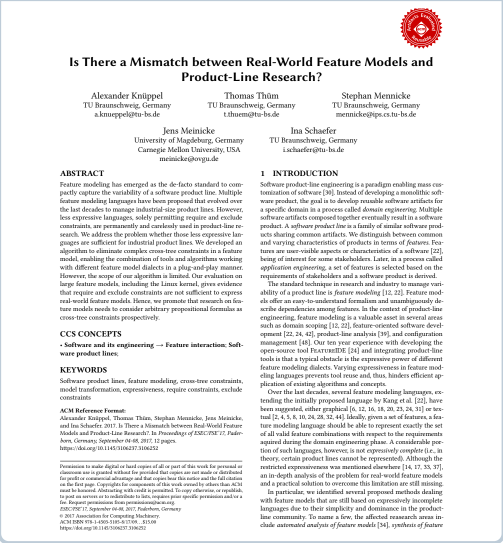
- very brief summary (often 1 paragraph)
- up-to one page for a thesis
- is this publication relevant to the reader?
- motivation: topic, general and specific problem, gap in the literature
- results: solution, main contributions, key findings, potential impact
- shorter is better (5–8 simple sentences)
- sometimes restricted to 200 or 250 words
- no references, no formulas

Terms, Keywords, . . .

- ACM Computing Classification System
- + freely chosen keywords
- + reference format, copyright or license

Example

[Knüppel 2017]



Is There a Mismatch between Real-World Feature Models and Product-Line Research?

Alexander Knüppel
TU Braunschweig, Germany
a.knueppel@tu-bs.de

Thomas Thüm
TU Braunschweig, Germany
t.thuem@tu-bs.de

Stephan Mennicke
TU Braunschweig, Germany
mennicke@ips.cs.tu-bs.de

Jens Meinicke
University of Magdeburg, Germany
Carnegie Mellon University, USA
meinicke@ovgu.de

Ina Schaefer
TU Braunschweig, Germany
i.schaefer@tu-bs.de

ABSTRACT

Feature modeling has emerged as the de-facto standard to compactly capture the variability of a software product line. Multiple feature modeling languages have been proposed that evolved over the last decades to manage industrial-size product lines. However, less expressive languages, solely permitting require and exclude constraints, are permanently and carelessly used in product-line research. We address the problem whether those less expressive languages are sufficient for industrial product lines. We developed an algorithm to eliminate complex cross-tree constraints in a feature model, enabling the combination of tools and algorithms working with different feature model dialects in a plug-and-play manner. However, the scope of our algorithm is limited. Our evaluation on large feature models, including the Linux kernel, gives evidence that require and exclude constraints are not sufficient to express real-world feature models. Hence, we promote that research on feature models needs to consider arbitrary propositional formulas as cross-tree constraints prospectively.

CCS CONCEPTS

• **Software and its engineering** → **Feature interaction**; **Software product lines**;

KEYWORDS

Software product lines, feature modeling, cross-tree constraints, model transformation, expressiveness, require constraints, exclude constraints

ACM Reference Format:
Alexander Knüppel, Thomas Thüm, Stephan Mennicke, Jens Meinicke, and Ina Schaefer. 2017. Is There a Mismatch between Real-World Feature Models and Product-Line Research?. In *Proceedings of ESEC/FSE'17, Paderborn, Germany, September 04-08, 2017*, 12 pages.
https://doi.org/10.1145/3106237.3106252

1 INTRODUCTION

Software product-line engineering is a paradigm enabling mass customization of software [30]. Instead of developing a monolithic software product, the goal is to develop reusable software artifacts for a specific domain in a process called *domain engineering*. Multiple software artifacts composed together eventually result in a software product. A *software product line* is a family of similar software products sharing common artifacts. We distinguish between common and varying characteristics of products in terms of *features*. Features are user-visible aspects or characteristics of a software [22], being of interest for some stakeholders. Later, in a process called *application engineering*, a set of features is selected based on the requirements of stakeholders and a software product is derived.

The standard technique in research and industry to manage variability of a product line is *feature modeling* [12, 22]. Feature models offer an easy-to-understand formalism and unambiguously describe dependencies among features. In the context of product-line engineering, feature modeling is a valuable asset in several areas such as domain scoping [12, 22], feature-oriented software development [22, 24, 42], product-line analysis [39], and configuration management [48]. Our ten year experience with developing the open-source tool FeatureIDE [24] and integrating product-line tools is that a typical obstacle is the expressive power of different feature modeling dialects. Varying expressiveness in feature modeling languages prevents tool reuse and, thus, hinders efficient application of existing algorithms and concepts.

Over the last decades, several feature modeling languages, extending the initially proposed language by Kang et al. [22], have been suggested, either graphical [6, 12, 16, 18, 20, 23, 24, 31] or textual [2, 4, 5, 8, 10, 24, 28, 32, 44]. Ideally, given a set of features, a feature modeling language should be able to represent exactly the set of all valid feature combinations with respect to the requirements aquired during the domain engineering phase. A considerable portion of such languages, however, is not *expressively complete* (i.e., in theory, certain product lines cannot be represented). Although the restricted expressiveness was mentioned elsewhere [14, 17, 33, 37], an in-depth analysis of the problem for real-world feature models and a practical solution to overcome this limitation are still missing.

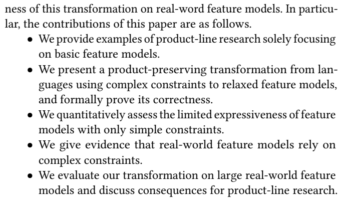
In particular, we identified several proposed methods dealing with feature models that are still based on expressively incomplete languages due to their simplicity and dominance in the product-line community. To name a few, the affected reasearch areas include *automated analysis of feature models* [34], *synthesis of feature*

Introduction

Introduction (Einleitung)

- more extensive version of the abstract
- one paragraph for each sentence in abstract?
- shorter is better (1–2 columns)
- motivating figure on the first page?
- **#ContributionUnclear**: end intro with list of 3–5 contributions (eigene, neue Beiträge)

Example for Contributions [Knüppel 2017]



ness of this transformation on real-word feature models. In particular, the contributions of this paper are as follows.

- We provide examples of product-line research solely focusing on basic feature models.
- We present a product-preserving transformation from languages using complex constraints to relaxed feature models, and formally prove its correctness.
- We quantitatively assess the limited expressiveness of feature models with only simple constraints.
- We give evidence that real-world feature models rely on complex constraints.
- We evaluate our transformation on large real-world feature models and discuss consequences for product-line research.

Hints for the Intro: DONTs

Avoid . . .

- repeating sentences from the abstract
- unnecessary information
- **#Overclaim**: overselling your work
- **#NoveltyUnclear**: claiming novelty without proper reference to the literature
- blank piece of paper problem

Hints for the Intro: DOs

- start writing down the contributions
- what concepts need to be introduced?
- what is the most related work?
- write outline first, then collect references
- allocate large chunks of time for writing

Background and Own Contribution

Background (Grundlagen)

- what is necessary to understand this work?
- know your audience
- **#NotSelfContained**: missing background
- **#LateContribution**: unnecessary or commonly known (i.e., by $>$ 90% readers)
- try motivating and running example

A Running Example [Knüppel 2017]

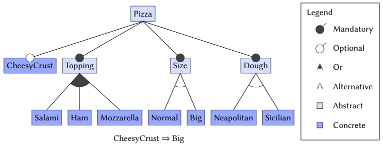


Figure 1: Basic feature model representing a product line for pizzas in FeatureIDE notation.

Own Contribution (Eigener Beitrag)

- main part of the publication ($>$ 50%)
- depends on type of contribution (cf. Slide 5)
- theory: formalization, proofs
- conceptual: new algorithms, techniques
- tool support: programs, scripts, data
- empirical evaluation: experiment or case study design, research questions, hypothesis (testing), threats to validity, discussion
- survey: methodology, results
- what is the innovation? references, examples where necessary
- **#ArgumentationMissing**: why over what, justify method / parameters / subjects
- **#NotReproducible**: results not reproducible

Evaluation and Discussion

Evaluation (Evaluierung)

- empirical evaluation: experiment, case study
- (theoretical evaluation: proof)
- start with research questions: is it better? does it scale (to industrial systems)?
- define and justify hypothesis (exception: explorative study)
- experiment design: subject (e.g., participants or software systems), threatment and baseline (e.g., algorithms), criteria, hardware
- experiment results: separate data from interpretation
- threats to validity: internal validity (what may have biased the results?), external validity (can results be generalized?)
- discussion: consequences of the results

A Table with Statistics and Results [Knüppel 2017]

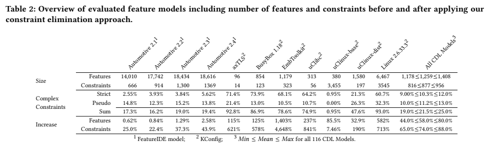


Table 2: Overview of evaluated feature models including number of features and constraints before and after applying our constraint elimination approach.

		Automotive 2.1[1]	Automotive 2.2[1]	Automotive 2.3[1]	Automotive 2.4[1]	axTLS[2]	BusyBox 1.18[2]	EmbToolkit[2]	uClibc[2]	uClinux-base[2]	uClinux-dist[2]	Linux 2.6.33.3[2]	All CDL Models[3]
Size	Features	14,010	17,742	18,434	18,616	96	854	1,179	313	380	1,580	6,467	1,178 ≤ 1,259 ≤ 1,408
	Constraints	666	914	1,300	1369	14	123	323	56	3,455	197	3545	816 ≤ 877 ≤ 956
Complex Constraints	Strict	2.55%	3.93%	3.84%	5.62%	71.4%	73.9%	68.1%	64.2%	0.95%	21.3%	60.7%	9.00% ≤ 10.3% ≤ 12.0%
	Pseudo	14.8%	12.3%	15.2%	13.8%	21.4%	13.0%	10.5%	10.7%	0.00%	26.3%	32.3%	10.0% ≤ 11.2% ≤ 13.0%
	Sum	17.3%	16.2%	19.0%	19.4%	92.8%	86.9%	78.6%	74.9%	0.95%	47.6%	93.0%	19.0% ≤ 21.5% ≤ 25.0%
Increase	Features	0.62%	0.84%	1.29%	2.58%	115%	125%	1,403%	237%	85.5%	32.9%	582%	44.0% ≤ 58.0% ≤ 80.0%
	Constraints	25.0%	22.4%	37.3%	43.9%	621%	578%	4,648%	841%	7.46%	190%	713%	65.0% ≤ 74.0% ≤ 88.0%

[1] FeatureIDE model; [2] KConfig; [3] *Min* ≤ *Mean* ≤ *Max* for all 116 CDL Models.

A Boxplot Diagram [Knüppel 2017]

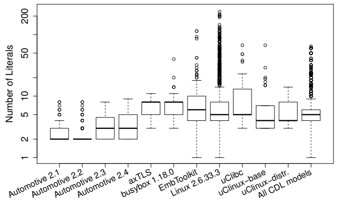


Figure 8: Number of literals of unprocessed complex constraints per evaluated feature model in logarithmic scale.

Related Work

Related Work (Verwandte Arbeiten)

- quality of research depends knowing the state-of-the-art
- what related work exists and what is the relation to your work?
- evidence for claims on novelty, contributions
- #NoClassificationOfRW: paragraph for each paper ⇒ paragraph for each topic classifying several papers
- #MissingRW: paper omits related work
- #NoComparisonWithRW: missing relation to your work ⇒ end each paragraph with 1–3 sentences describing pros, cons, trade-offs, synergies, and unique characteristics
- #NoveltyOverPriorWorkUnclear: own prior work not cited or not discussed
- do not write ~~related works~~

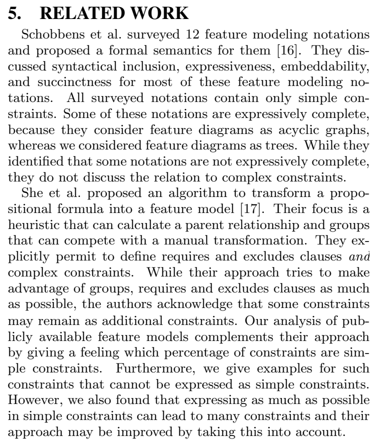
Example Paragraphs [Knüppel 2017]

Formal Semantics of Feature Models. The idea of defining a general formal semantics to catch a variety of feature modeling dialects, and, thus, enhancing applicability of algorithms and research in general, is not new. Czarnecki et al. [13] proposed a cardinality-based notation to support their introduced concept of staged configurations. They proposed a similar formal semantics to the one we defined in this paper. Schobbens et al. [33] surveyed 12 feature modeling languages and based upon them proposed a general formal semantics called *Free Feature Model* [33]. They discussed properties such as expressive power, embeddability, and succinctness. All surveyed languages contain only simple constraints and the role of complex constraints is not discussed. Our work extends their prior theory and highlights the differences in expressive power with real numbers between basic feature models and feature models with complex constraints.

Eliminating Cross-Tree Constraints. Only little emphasis in the product-line community was put on the elimination of cross-tree constraints. Broek and Galvao [43] discussed the elimination of simple constraints by translating a feature model to a generalized feature tree, a structure allowing features to occur multiple times in different places including a potential negation. Our approach uses abstract features and simple constraints. Therefore, it is applicable for most tools requiring basic feature models as input. Gil et al. [17] aim to prove that all cross-tree constraints can be eliminated for the price of introducing a new set of features. Their approach is, however, only addressed from a theoretical perspective. Both approaches are beyond the goals we intend to accomplish, since some sort of constraints are typically supported in product-line research.

Related Work

#NoClassificationOfRW

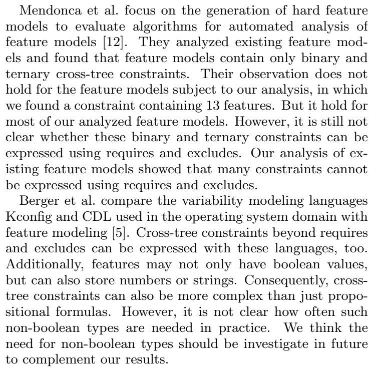


5. RELATED WORK

Schobbens et al. surveyed 12 feature modeling notations and proposed a formal semantics for them [16]. They discussed syntactical inclusion, expressiveness, embeddability, and succinctness for most of these feature modeling notations. All surveyed notations contain only simple constraints. Some of these notations are expressively complete, because they consider feature diagrams as acyclic graphs, whereas we considered feature diagrams as trees. While they identified that some notations are not expressively complete, they do not discuss the relation to complex constraints.

She et al. proposed an algorithm to transform a propositional formula into a feature model [17]. Their focus is a heuristic that can calculate a parent relationship and groups that can compete with a manual transformation. They explicitly permit to define requires and excludes clauses *and* complex constraints. While their approach tries to make advantage of groups, requires and excludes clauses as much as possible, the authors acknowledge that some constraints may remain as additional constraints. Our analysis of publicly available feature models complements their approach by giving a feeling which percentage of constraints are simple constraints. Furthermore, we give examples for such constraints that cannot be expressed as simple constraints. However, we also found that expressing as much as possible in simple constraints can lead to many constraints and their approach may be improved by taking this into account.

#MissingRW



Mendonca et al. focus on the generation of hard feature models to evaluate algorithms for automated analysis of feature models [12]. They analyzed existing feature models and found that feature models contain only binary and ternary cross-tree constraints. Their observation does not hold for the feature models subject to our analysis, in which we found a constraint containing 13 features. But it hold for most of our analyzed feature models. However, it is still not clear whether these binary and ternary constraints can be expressed using requires and excludes. Our analysis of existing feature models showed that many constraints cannot be expressed using requires and excludes.

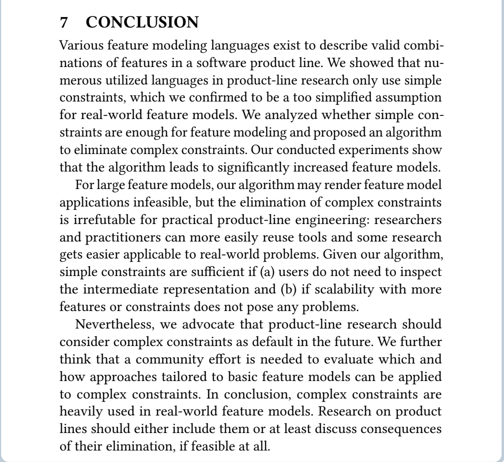
Berger et al. compare the variability modeling languages Kconfig and CDL used in the operating system domain with feature modeling [5]. Cross-tree constraints beyond requires and excludes can be expressed with these languages, too. Additionally, features may not only have boolean values, but can also store numbers or strings. Consequently, cross-tree constraints can also be more complex than just propositional formulas. However, it is not clear how often such non-boolean types are needed in practice. We think the need for non-boolean types should be investigate in future to complement our results.

Conclusion and Future Work

Conclusion (Fazit)

- ~~take~~ stay home messages
- conclusion ≠ summary/abstract
  ⇒ focus on findings, implications, limitations, future perspectives
- long-term vision, abstract from details, remove clutter
- try one paragraph each for summary, conclusion, and future work
- summary: introductory sentence(s), then a sentence on each contribution
- conclusion: consequences of the results
- future work: your plans, challenges for the community, be explicit and honest
- dedicated chapter on future work in theses
- do not use ~~Conclusions~~ as title

Summary, Conclusion, Future Work [Knüppel 2017]



7 CONCLUSION

Various feature modeling languages exist to describe valid combinations of features in a software product line. We showed that numerous utilized languages in product-line research only use simple constraints, which we confirmed to be a too simplified assumption for real-world feature models. We analyzed whether simple constraints are enough for feature modeling and proposed an algorithm to eliminate complex constraints. Our conducted experiments show that the algorithm leads to significantly increased feature models.

For large feature models, our algorithm may render feature model applications infeasible, but the elimination of complex constraints is irrefutable for practical product-line engineering: researchers and practitioners can more easily reuse tools and some research gets easier applicable to real-world problems. Given our algorithm, simple constraints are sufficient if (a) users do not need to inspect the intermediate representation and (b) if scalability with more features or constraints does not pose any problems.

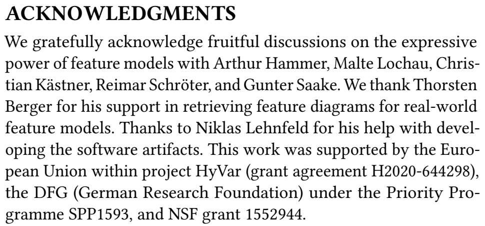
Nevertheless, we advocate that product-line research should consider complex constraints as default in the future. We further think that a community effort is needed to evaluate which and how approaches tailored to basic feature models can be applied to complex constraints. In conclusion, complex constraints are heavily used in real-world feature models. Research on product lines should either include them or at least discuss consequences of their elimination, if feasible at all.

Acknowledgments and Appendix

Acknowledgments (Danksagung)

- thank people that contributed (but not enough to be an author)
- give credits to source of funding
- British English: Acknowledgements

Example [Knüppel 2017]



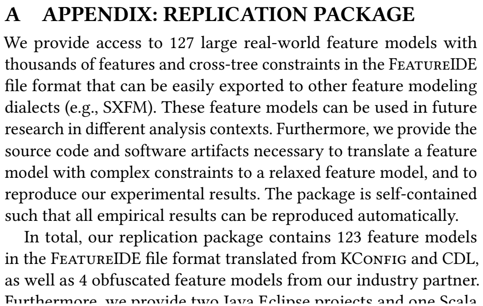
ACKNOWLEDGMENTS

We gratefully acknowledge fruitful discussions on the expressive power of feature models with Arthur Hammer, Malte Lochau, Christian Kästner, Reimar Schröter, and Gunter Saake. We thank Thorsten Berger for his support in retrieving feature diagrams for real-world feature models. Thanks to Niklas Lehnfeld for his help with developing the software artifacts. This work was supported by the European Union within project HyVar (grant agreement H2020-644298), the DFG (German Research Foundation) under the Priority Programme SPP1593, and NSF grant 1552944.

Appendix (Anhang)

is optional! tables, diagrams, formalisms, details, tools that are only relevant to some readers

Example [Knüppel 2017]



A APPENDIX: REPLICATION PACKAGE

We provide access to 127 large real-world feature models with thousands of features and cross-tree constraints in the FeatureIDE file format that can be easily exported to other feature modeling dialects (e.g., SXFM). These feature models can be used in future research in different analysis contexts. Furthermore, we provide the source code and software artifacts necessary to translate a feature model with complex constraints to a relaxed feature model, and to reproduce our experimental results. The package is self-contained such that all empirical results can be reproduced automatically.

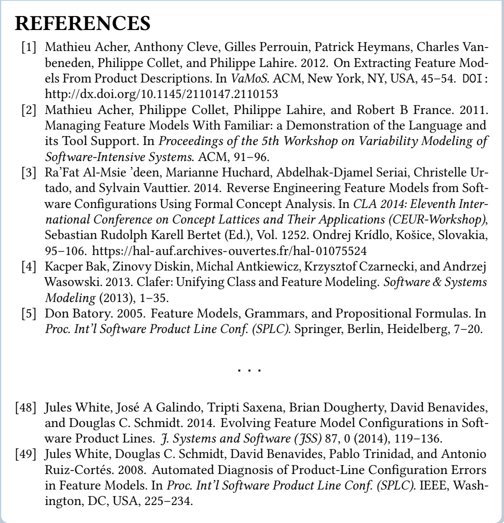
In total, our replication package contains 123 feature models in the FeatureIDE file format translated from KConfig and CDL, as well as 4 obfuscated feature models from our industry partner. Furthermore, we provide two Java Eclipse projects and one Scala

References

References (Literaturverzeichnis)

- (scientific) **writing requires reading**
- give credits to previous and contextual work
- reference claims, quotes, prior results
- avoid irrelevant citations
- prefer primary over secondary literature
- prefer journal to conference to workshop to technical report to web pages
- prefer BibTeX entries from ACM over DBLP over Springer/IEEE/... over Google Scholar/Mendeley/Research Gate
- clean-up, complete, and make consistent
- use `note = {To appear.}` if accepted but not yet published
- never use `\nocite`

Example [Knüppel 2017]



REFERENCES

[1] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. 2012. On Extracting Feature Models From Product Descriptions. In *VaMoS*. ACM, New York, NY, USA, 45–54. DOI: http://dx.doi.org/10.1145/2110147.2110153

[2] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B France. 2011. Managing Feature Models With Familiar: a Demonstration of the Language and its Tool Support. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*. ACM, 91–96.

[3] Ra'Fat Al-Msie 'deen, Marianne Huchard, Abdelhak-Djamel Seriai, Christelle Urtado, and Sylvain Vauttier. 2014. Reverse Engineering Feature Models from Software Configurations Using Formal Concept Analysis. In *CLA 2014: Eleventh International Conference on Concept Lattices and Their Applications (CEUR-Workshop)*, Sebastian Rudolph Karell Bertet (Ed.), Vol. 1252. Ondrej Krídlo, Košice, Slovakia, 95–106. https://hal-auf.archives-ouvertes.fr/hal-01075524

[4] Kacper Bak, Zinovy Diskin, Michal Antkiewicz, Krzysztof Czarnecki, and Andrzej Wasowski. 2013. Clafer: Unifying Class and Feature Modeling. *Software & Systems Modeling* (2013), 1–35.

[5] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proc. Int'l Software Product Line Conf. (SPLC)*. Springer, Berlin, Heidelberg, 7–20.

...

[48] Jules White, José A Galindo, Tripti Saxena, Brian Dougherty, David Benavides, and Douglas C. Schmidt. 2014. Evolving Feature Model Configurations in Software Product Lines. *J. Systems and Software (JSS)* 87, 0 (2014), 119–136.

[49] Jules White, Douglas C. Schmidt, David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2008. Automated Diagnosis of Product-Line Configuration Errors in Feature Models. In *Proc. Int'l Software Product Line Conf. (SPLC)*. IEEE, Washington, DC, USA, 225–234.

Substructure (Unterstruktur)

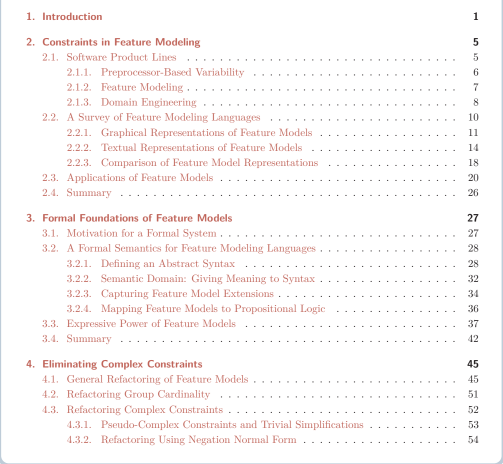
Chapters, Sections, Subsections, ...

- in LaTeX: `\chapter` (only in theses), `\section`, `\subsection`, `\subsubsection`, ..., `\paragraph`
- avoid too many/few siblings: 2–6 sections
- structural depth should correlate with length
- end with summary for sections or chapters describing your own contribution (concept, implementation, evaluation)

Paragraphs and Sentences

- start with topic sentence (connection to previous paragraph)
- end with summarizing sentence
- avoid too short/long paragraphs

Recap: A Master's Thesis [Knüppel 2016]



1. Introduction 1

2. Constraints in Feature Modeling 5
- 2.1. Software Product Lines . . . 5
  - 2.1.1. Preprocessor-Based Variability . . . 6
  - 2.1.2. Feature Modeling . . . 7
  - 2.1.3. Domain Engineering . . . 8
- 2.2. A Survey of Feature Modeling Languages . . . 10
  - 2.2.1. Graphical Representations of Feature Models . . . 11
  - 2.2.2. Textual Representations of Feature Models . . . 14
  - 2.2.3. Comparison of Feature Model Representations . . . 18
- 2.3. Applications of Feature Models . . . 20
- 2.4. Summary . . . 26

3. Formal Foundations of Feature Models 27
- 3.1. Motivation for a Formal System . . . 27
- 3.2. A Formal Semantics for Feature Modeling Languages . . . 28
  - 3.2.1. Defining an Abstract Syntax . . . 28
  - 3.2.2. Semantic Domain: Giving Meaning to Syntax . . . 32
  - 3.2.3. Capturing Feature Model Extensions . . . 34
  - 3.2.4. Mapping Feature Models to Propositional Logic . . . 36
- 3.3. Expressive Power of Feature Models . . . 37
- 3.4. Summary . . . 42

4. Eliminating Complex Constraints 45
- 4.1. General Refactoring of Feature Models . . . 45
- 4.2. Refactoring Group Cardinality . . . 51
- 4.3. Refactoring Complex Constraints . . . 52
  - 4.3.1. Pseudo-Complex Constraints and Trivial Simplifications . . . 53
  - 4.3.2. Refactoring Using Negation Normal Form . . . 54

Structure in Writing

Lessons Learned

- Motivation for structure
- Typical structure of papers and theses:

  title, authors, abstract, keywords,
  introduction, background,
  own contribution, evaluation,
  related work, conclusion, future work,
  acknowledgments, appendix, references
- Substructure and paragraphs

Practice

- Read abstract, introduction, and conclusion of Knüppel 2017
- What is the relation between them?
- Compare the structure of an A*-level conference or journal paper (e.g., Knüppel 2017) with that of a very good master's thesis (e.g., Knüppel 2016)
- Which similarities and differences are there?

Lecture Contents

Clarity in Writing

Why Care About Writing Quality?

Consequences of Poor Writing

- ambiguity leads to misunderstanding
- obscurity increases effort for readers
- poor reviews, bad grade or rejections
- frustration and little impact
- even if for world's greatest contribution
- sloppy presentation suggests irrelevant work

Why Students Learn About Scientific Writing?

- one goal of study programs is to be able to work scientifically
- includes critical review of research
- scientific research requires scientific writing
- writing can be learned by practicing it (in contrast to inspiration and talent)
- skills in scientific writing improve your thinking and talking

Process of Scientific Writing

The Process

- first research, then writing? start writing as early as possible
- start creating an outline, discuss with colleagues or co-authors
- collect references (not during writing phase)
- write draft, revise later
- a high-quality writing often takes more than 10 revisions
- ~~everything~~ writing requires a schedule
- stick to your schedule, no excuses
- welcome feedback and criticism, improve your writing with every part

Research Takes Time [youtube.com]

- August 2011: idea for a paper
- December 2012: first reject (+1.25y)
- June 2014: begin master's thesis (+1.5y)
- November 2015: new student (+1.5y)
- August 2016: thesis published (+0.75y)
- June 2017: paper accepted (+0.75y)
- September 2017: paper presented (+0.25y)
- March 2018: paper presented again (+0.5y)
- 6.5 years of research

DOs and DONTs in Writing

DOs

- **#ActiveVoice**: use active (e.g., "we contribute") more than passive voice (e.g., "is discussed elsewhere")
- **#We**: use "we" even for single author papers and theses
- exception: "I" is fine in acknowledgments
- **#InconsistentTense**: consistent tense (mostly present tense)
- **#UseSingular**: prefer singular over plural (in definitions)
- **#Simplify**: support non-native speakers: use simple, familiar words and simple sentences
- **#IntroductoryPhrases**: "In summary, we . . . ", "Hence, we . . . ", "First, . . . Second, . . . Finally, . . . "

DONTs

- **#Informal**: avoid jargon
- **#Clutter**: cut unnecessary words
- **#Monotone**: avoid monotone sentence structures
- **#Ambiguous**: avoid ambiguities
- **#AvoidSynonyms**: it is fine to repeat subjects, important terms (we . . . the authors . . . the ones who performed this research)
- **#AvoidAbbreviations**: unless very common (e.g., RAM, DVD, URL, UML, ASCII)
- **#NoStartWithBut**: do not start sentences with "but"

#CohesionMissing

Cohesion (Roter Faden)

- split text into coherent paragraphs; one topic per paragraph
- every paragraph answers questions and poses new questions answered by next paragraph(s)
- **#TopicSentences**: say what you say before you say it – first sentence of a paragraph connects known concepts with the topic of the paragraph
- **#Resurface**: the last sentence of a paragraph summarizes the paragraph
- **#ParagraphTooShort**, **#ParagraphTooLong**
- try to re-order phrases within a sentence
- **#ForwardReferences**: support linear reading
- introduce terms before using them and only introduce those that you need

No Cohesion – The Ingredients

- Collect 5+ random co-authors – the more, the better
- Ideal: co-authors that have never worked together
- Meet for 10 minutes to discuss the paper (e.g., during lunch or part of an unrelated meeting)
- Everyone writes a different part of the paper
- Avoid wasting time with further meetings
- Avoid proof reading to avoid bias
- Submit as-is

Connectives (Konnektive)

Motivation

- #MonotoneConnectives: avoid word repetitions for connectives (unlike concepts)
- #TooConnected: do not use them in every sentence, use order of sentences instead

Connectives

- adding: furthermore, moreover, in addition, and, also, too
- sequencing: first, second, third, . . . , finally, next, then, before, after, meanwhile, eventually
- exemplifying: for example, for instance, such as (“colors, such as green, are”) – (avoid et cetera / etc.)

More Connectives

- causality: hence, thus, therefore, consequently, so, caused by, as, because, because of – (use “since” only for temporal connection)
- qualifying: if, however, despite, although, yet, unless, except, apart from, as long as
- emphasizing: in particular, especially, significantly, indeed, notably, above all, most of all
- comparing: in comparison, compared with, similarly, likewise, like, equally, as with
- contrasting: in contrast, otherwise, alternatively, whereas, instead of, unlike, on the one hand . . . on the other hand

Common Mistakes in Writing

Common Mistakes

- see hashtags above
- **#MissingMotivation**: why is it relevant?
- **#BigPicture**: overview missing, just details
- **#Why**: do not only report what you did but why you did it
- **#Only**, **#Also**: "I only compiled my code" (not tested or documented it) vs "I compiled only my code" (not yours)
- **#Allows**: verb is not reflexive – "allows developers to"
- **#Misspelled**: names are important, get them right – avoid autocorrection: Ina Schaefer (not Ina Schäfer) – get all symbols right: Andrzej Wąsowski, Ștefan Stănciulescu

Further Common Mistakes

- **#AEBE**: use American English (common) or British English, but do not mix
- **#OxfordComma**: "red, green, or blue"
- **#TooManyBrackets**: avoid brackets (for different meanings)
- **#cfieeg**: confer (compare), id est (that is), exempli gratia (for example) only in brackets: "(cf. bla)" but "(i.e., bar)" and "(e.g., foo)"
- **#etal**: "et al." is a shorthand for et alii (und andere)
- **#dot**: LaTeX creates a longer white space at the end of sentences ⇒ abbreviations need special care: "Fig.~1", "Knüppel~et~al.\ contribute", "Knüppel~et~al.~\cite{key}"

Clarity in Writing

Lessons Learned

- Scientific writing improves your thinking
- Process of writing
- DOs and DONTs
- Cohesion, connectives, common mistakes

Practice

- Rewrite a well-written paragraph by introducing at least one problem into each sentence
- Ask a colleague to add hashtags for all found problems and discuss those
- Optional: watch my experience report about 6.5 years of research

Lecture Contents