Bachelor's Thesis

Incremental Construction of Modal Implication Graphs for Feature-Model Evolution

Author:

Rahel Arens

January 08, 2021

Advisors:

Prof. Dr.-Ing. Ina Schaefer

Michael Nieke, M.Sc.

Institute of Software Engineering and Automotive Informatics
TU Braunschweig

Prof. Dr.-Ing. Thomas Thüm

Institute of Software Engineering and Programming Languages
Ulm University

For FinancialServices01 we have ten FM versions with nine evolution steps. In Figure 5.19, we show the difference between the number of detected redundant clauses for each evolution step. As the diagram depicts, the incremental calculation finds more redundant clauses for each evolution step.

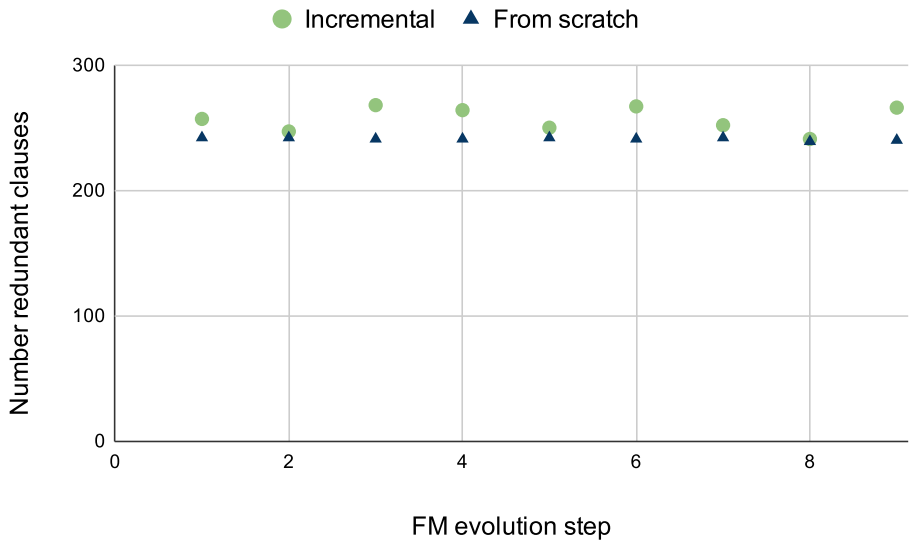

Figure 5.19: Number of found redundant clauses for all FinancialServices01 MIGs.

Figure 5.20 shows the redundant clauses that both kinds of calculation have detected. The diagram shows that the incremental construction and the calculation from scratch find around the same number of redundant clauses.

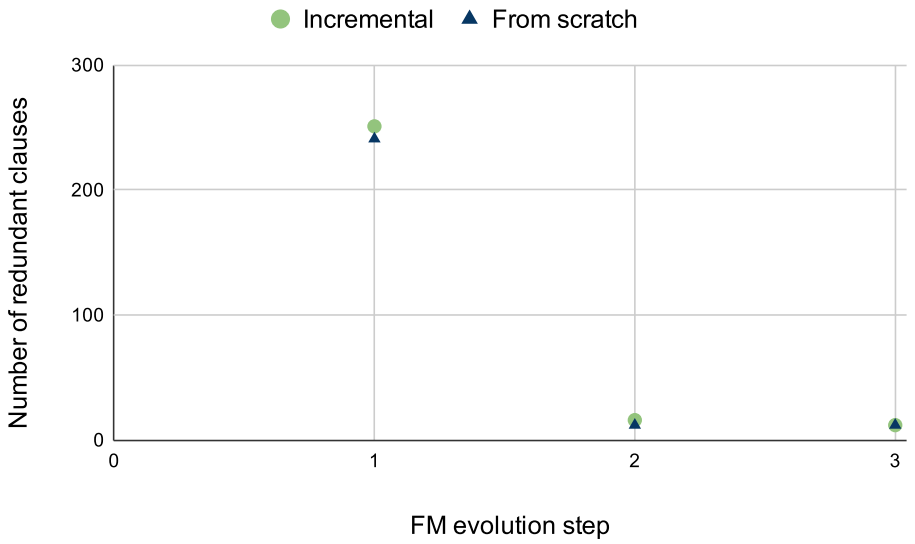

Figure 5.20: Number of found redundant clauses for all Automotive02 MIGs.

Implicit Strong Edges Difference

For the BusyBox models, we evaluated all 3,712 FM versions as one. We detected differences in the implicit strong edges in only 2 of the 3,711 evolution steps. This

means, that the incremental construction found almost every implicit strong edge that the calculation from scratch detects.

Now, we show the difference in the implicit strong edges for the FinancialServices01 MIGs in Figure 5.21. As the diagram shows, the incremental calculation finds almost as many implicit strong edges as the calculation from scratch (i.e., the incremental calculation finds 359,946 and the calculation from scratch finds 363,333 in total).

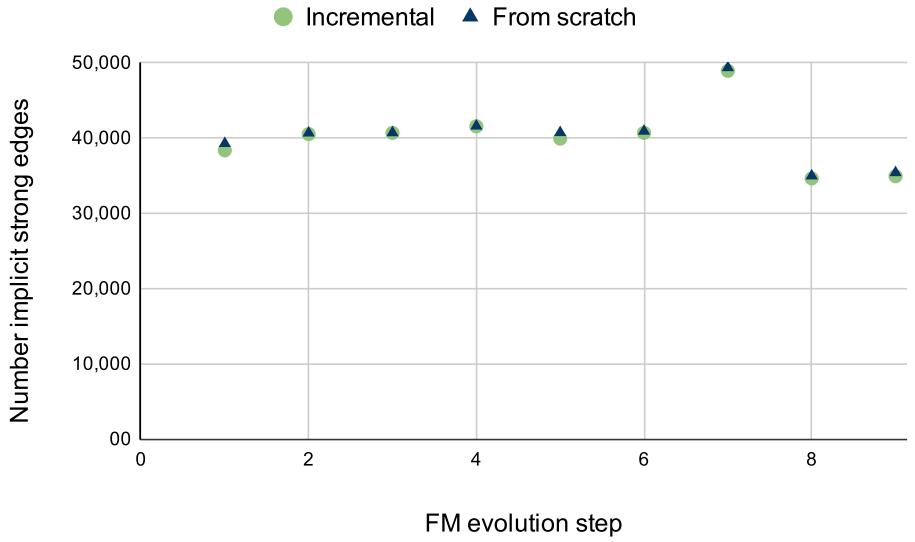

Figure 5.21: Number of found implicit strong edges for all FinancialServices01 MIGs.

For the Automotive02 MIGs, the difference regarding the implicit strong edges, depicted by the diagram in Figure 5.22, is zero for the first evolution step (both calculations found 166 implicit strong edges). In the second evolution step, the incremental calculation found 634 implicit strong edges while the calculation from scratch found 700. In the third evolution step, the incremental calculation found 690 implicit strong edges and the calculation from scratch 696.

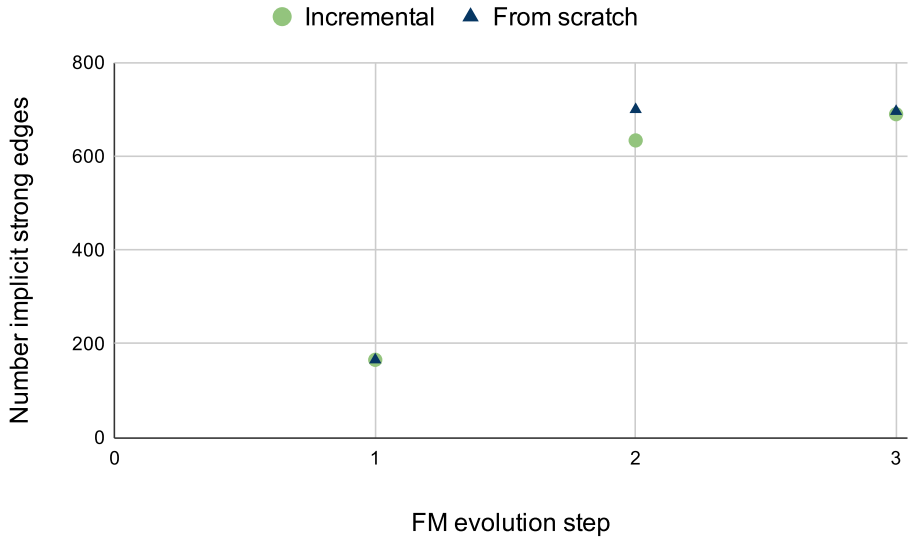

Figure 5.22: Number of found implicit strong edges for all Automotive02 MIGs.

5.3 Discussion

Regarding the RQ1, the evaluation shows that for most cases, the incremental calculation of the MIG takes less time than calculating the MIG from scratch. As Table 5.3 and Table 5.4 show, when constructing an incomplete MIG, the most time is saved during the calculation of transitive strong edges. Since this calculation is the same as for the from scratch calculated MIG except a few adaptions, it is only faster because of the way we implemented the method and not because of its incrementality. When disregarding the time for this step, Table 5.3 and Table 5.4 show that the incremental calculation still requires less computation time.

When constructing a complete MIG, depending on the size of the changes, the incremental calculation brings a large advantage. When inspecting the necessary calculation times for Automotive02, the first evolution step takes the incremental construction a lot of more time than for the other evolution steps. However, the first evolution step includes large changes in the CNF (i.e., more clauses have been added than clauses contained in the CNF before the evolution step). The second and third evolution steps also include relatively large changes to the FM, but the incremental construction is faster anyway. In summary we can say that the incremental construction brings a bigger advantage for large-scale feature models regarding the computation time. Also, the bigger the changes to the FM and, thus, to the CNF, the less advantage brings the incremental construction. This claim is also supported by Figure 5.9 where the first and the fifth evolution step are the only ones were the incremental construction takes more time than calculating from scratch. Table 5.2 shows, that these are by far the largest changes in the CNF for all nine evolution steps as there are over 2,000 added clauses whereas the other steps have less than 700 added clauses.

When evaluating RQ2 and, thus, the necessary SAT calls for every calculation, we can see that there is a connection between the calculation time and number of SAT calls in the calculation. This confirms our assumption that the reduction of SAT calls is an effective approach for reducing the calculation time which we achieved with our incremental calculations. For the vast majority of the evolution steps that we evaluated, the incremental construction requires fewer SAT calls than the calculation from scratch.

We detected that the incremental calculation of redundant clauses needs less time without a high loss of accuracy as we can see by the fact that it finds more redundant clauses in most of the cases we evaluated in RQ3. The reason is how the SAT solver works. Every clause that is already negatively checked for redundancy is added to the SAT solver. Clauses that were not checked yet are not part of the SAT solver. It might be that a clause that would be redundant is checked before the set of clauses causing the clause to be redundant is part of the SAT solver. Hence, the clause would not be declared redundant. In the incremental construction, the clauses that are not directly affected by one of the added clauses are added to the SAT solver before we inspect the potentially redundant clauses. Since it is more likely that clauses that share a literal with one of the the added clauses or one of the added clauses are also affected in their redundancy, the incremental construction enlarges the chance to find them due to the already existing set of remaining clauses in

the SAT solver. Consider the following example: we have the clauses $\{B,\ C\}$ and $\{A,\ \neg C\}$. These clauses express, that when deselecting C we have to select B and when selecting C we have to select A. Then, we add the clause $\{A,\ B\}$ which is redundant, because either A or B have to be selected due to the already existing clauses. When calculating from scratch, we might check the clause $\{A,\ B\}$ first and since the other clauses are not checked yet and, thus, are not part of the CNF in the SAT solver yet, we would not detect the redundancy of the clause $\{A,\ B\}$. In the incremental calculation, the clauses $\{B,\ C\}$ and $A,\ \neg C$ are part of the SAT solver before verifying $\{A,\ B\}$, since the added clauses are checked last with all the remaining clauses already in the CNF of the SAT solver. Thus, the incremental calculation would detect the redundancy of the specific clause.

As we also show when answering RQ3, the incremental construction does not find all implicit strong edges. The main cause of implicit strong edges are cross-tree constraints, since they cause relations between features that are neither expressed by the FM nor explicitly by a constraint. For the FinancialServices01 evolution steps, where the incremental construction misses the most implicit strong edges, we argue that the loss is relatively small compared to the time we can save. Comparing the results of FinancialServices01 and Automotive02 it might seem like an irregularity that the incremental construction misses for Automotive02, which has much more added and removed clauses, less implicit strong edges. This is probably due to the fact that FinancialServices01 has more cross-tree constraints in relative terms. While the Automotive02 versions have between 14,010 and 18,616 features and between 666 and 1,319 cross-tree constraints, FinancialServices01 has around 700 features but around 1,000 cross-tree constraints. That means, that FinancialServices01 has a lot of features that have a relation due to the cross-tree constraints and, hence, the MIGs of these FMs have a lot more implicit strong edges than the MIGs of the Automotive02 FM versions. Anyway, missing implicit strong edges has no impact on the correctness of the MIG but on the performance during the configuration construction. The implicit strong edges give a small benefit regarding the time needed to find the connections between features and, thus, the consequences of a (de-)selection of a feature.

The evaluation shows that the efficiency of the incremental calculation depends on multiple characteristics of the FM (i.e., the size, the number of changes, and the relative number of cross-tree constraints). The advantage of the incremental construction grows with the size of the feature models. The reason is, that the required time for the calculation from scratch typically grows with the size of a FM. When adapting the MIG to changes the size of the feature model has much less influence on the computation time than the number of changes that have to be adapted. The relative number of cross-tree constraints also influences the incremental adaption in a way that more cross-tree constraints lead to more implicit strong edges. Anyway, the goal of this thesis was to reduce the high number of SAT calls and the related high computation time when constructing a MIG and still have a resulting MIG that is efficient and correct. The evaluation strongly indicates that the goal is achieved. Even though we do not have a fully complete MIG resulting by the incremental construction, the loss is relatively small when comparing to the time we can save with the introduced concept.

5.4 Threats to Validity

In this section, we describe the threats to the validity of our evaluation.

We cannot guarantee the efficiency of the incremental construction of a MIG when using for FMs we did not evaluate in this thesis. To prevent that and give the best overview of the advantage of the incremental construction, we evaluated FMs from multiple domains with different sizes and FM evolution steps with a different number of changes in the FM. To present the best use cases for the incrementally calculated MIG we identified the characteristics of FMs that bring advantages for the incremental construction in Section 5.3.

We did not formally prove the correctness of the incrementally constructed MIG. Thus, we cannot completely guarantee the correctness of the incrementally constructed MIG. To this end, we made a lot of sanity checks. For example, we compared the resulting MIGs not only by time but also each variable to verify that the MIGs have the same strong edges, the same weak edges, and the same attributes (i.e., the same variables are core/dead). When detecting differences, like a clause that exists in the from scratch calculated MIG but not in the incrementally constructed MIG, we verified that it is a redundant clause and not a falsely ignored clause.

When using Java for the implementation, just-in-time compilation can have an impact on the required time for each calculation. To prevent the calculation times to be influenced by the JVM, we run a JVM warm up before each calculation.

We did not evaluate the additional memory that is required when incrementally calculating. Since we have to save a lot more information than necessary for the calculation from scratch, this might be an interesting part to evaluate in the future. However, for large-scale feature models this can be outsourced on servers. The money for the server can be retrieved by the time savings of the incremental construction.

6. Related Work

In this chapter, we describe work that is related to this thesis. We discuss feature model evolution and incremental SAT solvers.

Multiple researchers investigate feature model evolution on a feature level. Pleuss et al. [PBD+12] split FMs into *fragments* wherein changes in the fragments due to evolution are modeled by *EvoOperators*. Cordy et al. [CCS+12] analyze feature model changes whereas they reduce the model with knowledge about the edits for model checking after evolution steps. Thüm et al. [TBK09] divide edits to feature models into four classifications. Even though they also use the CNF of feature models to verify changes in the FM, that is done only for the classifications and not for finding the exact difference in the CNFs of the FMs as base for FM evolution analysis. Hoff et al. [HNS+20] save FM evolution operations and determine changes in the FM by inspecting the performed operations. The described information about FM evolution operations could be used for future work to incrementally adapt MIGs more efficiently. Neves et al. [NTS+11] introduce multiple templates for the analysis of different FM evolution scenarios. However, none of these approaches uses CNF differences for feature model evolution changes. Acher et al. [AHC+12] introduce a concept of composing and decomposing FMs. They determine the FM differences of two FMs by translating them into a CNF and into a binary directed graph and compute the FM difference with the CNF or with the graph.

The literature contains multiple approaches for incremental SAT solvers [ES03, FBS19, NRS14]. Besides, Een et al. [EB05] introduce techniques to reduce a CNF to speed up the usage of a SAT solver. However, since the goal of this thesis is to analyze the advantage of the incremental MIG compared to the MIG calculated from scratch, we did not consider improvements regarding the SAT solver and instead adapted the SAT solver that already exists in the implementation of a MIG. Investigating the usage of incremental SAT solvers for improvements of the incremental calculation of a MIG is an interesting factor for future work.

7. Conclusion and Future Work

In this thesis, we introduced all concepts that are necessary for the incremental calculation of a MIG and provided an implementation realizing our method. We also presented an evaluation of our work for multiple industrial feature models with different evolution steps. For several steps of the incremental construction, we evaluated multiple algorithms to examine trade-offs between accuracy and runtime. In summary, we conclude that on the one hand the incrementally constructed MIG is more efficient regarding the computation time and the reduction of number of necessary SAT calls. On the other hand, it has a small loss of accuracy regarding the redundant clauses and the implicit strong edges. We argue that this loss is relatively small compared to the possible reduction of computation time and that the incremental construction is especially worth using when calculating a complete MIG. Our empirical evaluation indicates that the advantage grows for larger feature models when incrementally adapting the MIG after every evolution step in the FM. However, even for small feature models the incremental calculation is typically more efficient. Thus, we argue that applying the incremental construction yields benefits for feature models of various sizes. In general, the improvement regarding efficiency is larger for the calculation of a complete MIG since we save the most time when detecting implicit strong edges.

We identified multiple possible improvements for the incremental construction as future work. First, we can analyze whether we can use the knowledge of the previously dead and core features for an incremental algorithm of the calculation of the new dead and core features. Second, we can investigate changes on the level of FM edit operations (e.g., move feature) to detect affected features and, thus affected clauses more efficiently. This way, we might ignore clauses that are unnecessarily observed. Third, we can improve the search for implicit strong edges to save even more SAT calls. The MIG calculated from scratch uses a depth first search and a method for a preselection involved in the calculation of implicit strong edges which we could adapt for the incremental calculation. Thereupon, we can analyze whether the knowledge of changes in the MIG can be used to fix existing configurations that are invalid after the evolution of its FM.

A. Appendix

Figure A.1 shows the calculation for the FinancialServices01 models for complete graphs without redundant clauses.

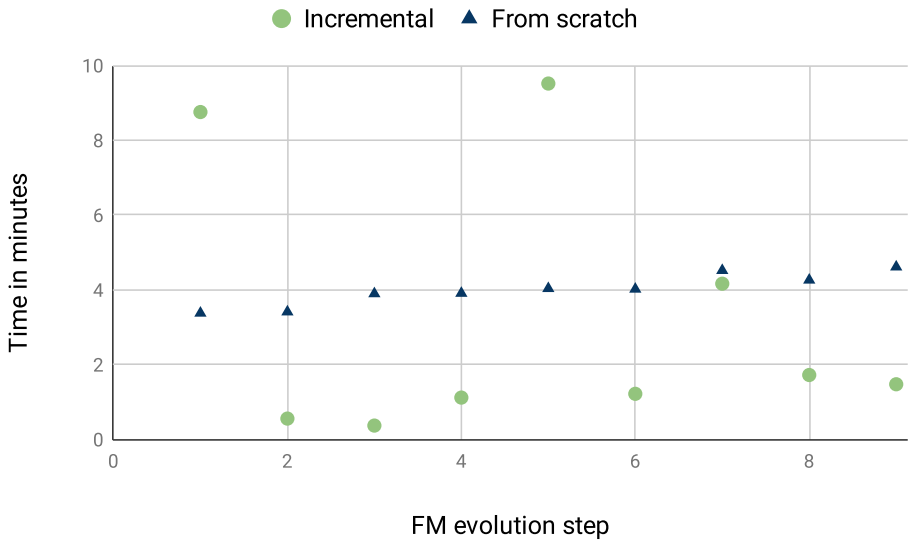

Figure A.1: Calculation times for all FinancialServices01 evolution steps for a complete MIG without redundant clauses.

Figure A.2 shows the calculation for the Automotive02 models for complete graphs without redundant clauses.

Table A.1 shows the exact times for each calculation for the evolution step from Automotive02_V2 to Automotive02_V3 for a complete graph without redundant clauses.

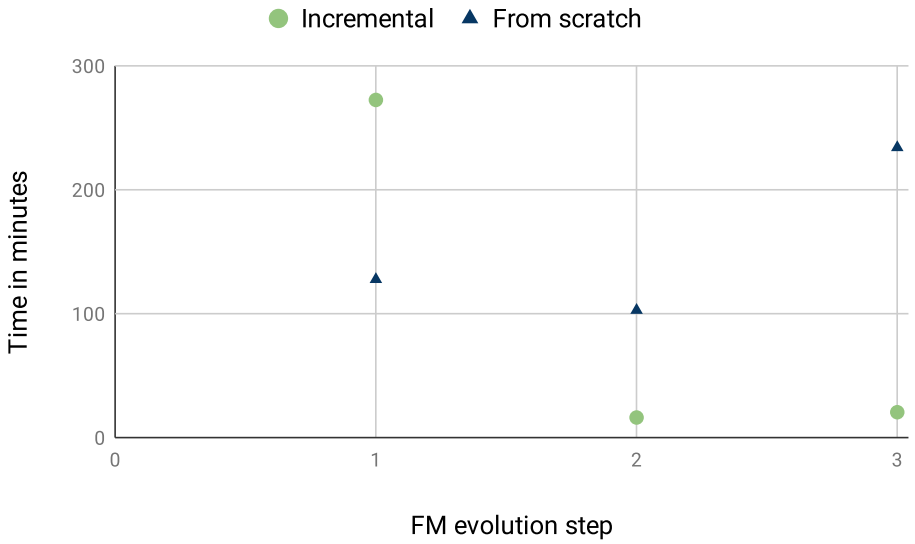

Figure A.2: Calculation times for all Automotive02 evolution steps for a complete MIG without redundant clauses.

Calculation Step	Calculated MIG (sec)	Adapted MIG (sec)	Time Difference Adapted MIG - Calculated MIG(sec)
Remove old transitive strong edges	0	0.09	+0.09
Calculate core and dead feature	8.92	8.94	+0.02
Clean CNF	0.18	0.1	-0.08
Calculate CNF difference	0	0.48	+0.48
Removed clauses	0	0.01	+0.01
Added clauses	0	2.05	+2.05
Find transitive strong edges	33.11	5.54	-27.57
Find transitive weak edges	12.65	0.8	-11.85
Find implicit strong edges	5,958.84	974.57	-4,984.27
Create MIG	1.86	0	-1.86
Calculate redundant clauses	0.02	0	-0.02
Find transitive strong edges	32.95	0.04	-32.91
Overall time	6,048.71	992.83	-5,055.88

Table A.1: Calculation times for each step for the complete MIG without redundant clauses in the evolution step from Automotive02_02 to Automotive02_03.

Bibliography

[AHC+12] Mathieu Acher, Patrick Heymans, Philippe Collet, Clément Quinton, Philippe Lahire, and Philippe Merle. Feature model differences. In *International Conference on Advanced Information Systems Engineering*, pages 629–645. Springer, 2012. (cited on Page 49)

[Bat05] Don Batory. Feature models, grammars, and propositional formulas. In *International Conference on Software Product Lines*, pages 7–20. Springer, 2005. (cited on Page 1, 3, and 4)

[BSRC10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information systems*, 35(6):615–636, 2010. (cited on Page 1 and 3)

[CCS+12] Maxime Cordy, Andreas Classen, Pierre-Yves Schobbens, Patrick Heymans, and Axel Legay. Managing evolution in software product lines: A model-checking perspective. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, pages 183–191, 2012. (cited on Page 49)

[CHE05] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software process: Improvement and practice*, 10(1):7–29, 2005. (cited on Page 1 and 3)

[CW07] Krzysztof Czarnecki and Andrzej Wasowski. Feature diagrams and logics: There and back again. In *11th International Software Product Line Conference (SPLC 2007)*, pages 23–34. IEEE, 2007. (cited on Page 1 and 3)

[EB05] Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In *International conference on theory and applications of satisfiability testing*, pages 61–75. Springer, 2005. (cited on Page 49)

[ES03] Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003. (cited on Page 49)

[FBS19] Katalin Fazekas, Armin Biere, and Christoph Scholl. Incremental inprocessing in sat solving. In *International Conference on Theory and Appli-*

cations of Satisfiability Testing, pages 136–154. Springer, 2019. (cited on Page 49)

[HNS+20] Adrian Hoff, Michael Nieke, Christoph Seidl, Eirik Halvard Sæther, Ida Sandberg Motzfeldt, Crystal Chang Din, Ingrid Chieh Yu, and Ina Schaefer. Consistency-preserving evolution planning on feature models. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*, pages 1–12, 2020. (cited on Page 49)

[IF10] Ayelet Israeli and Dror G. Feitelson. The Linux kernel as a case study in software evolution. 83(3):485–501, 2010. (cited on Page 1)

[Jan08] Mikolas Janota. Do sat solvers make good configurators? In *SPLC (2)*, pages 191–195, 2008. (cited on Page 5)

[Kri15] Sebastian Krieter. *Efficient Configuration of Large-Scale Feature Models Using Extended Implication Graphs*. PhD thesis, Citeseer, 2015. (cited on Page 5)

[KTS+09] Christian Kastner, Thomas Thum, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. Featureide: A tool framework for feature-oriented software development. In *2009 IEEE 31st International Conference on Software Engineering*, pages 611–614. IEEE, 2009. (cited on Page 21)

[KTS+18] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Reimar Schröter, and Gunter Saake. Propagating configuration decisions with modal implication graphs. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 898–909. IEEE, 2018. (cited on Page 1, 2, 5, 9, and 18)

[KZK10] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin. Model counting in product configuration. *arXiv preprint arXiv:1007.1024*, 2010. (cited on Page 1 and 4)

[MTS+17] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. *Mastering software variability with FeatureIDE*. Springer, 2017. (cited on Page 21)

[NRS14] Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Ultimately incremental sat. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 206–218. Springer, 2014. (cited on Page 49)

[NTS+11] Laís Neves, Leopoldo Teixeira, Demóstenes Sena, Vander Alves, Uirá Kulesza, and Paulo Borba. Investigating the safe evolution of software product lines. In *Proceedings of the 10th ACM international conference on Generative programming and component engineering*, pages 33–42, 2011. (cited on Page 49)

[PBD+12] Andreas Pleuss, Goetz Botterweck, Deepak Dhungana, Andreas Polzer, and Stefan Kowalewski. Model-driven support for product line evolution on feature level. *Journal of Systems and Software*, 85(10):2261–2274, 2012. (cited on Page 49)

[SSK+20] Joshua Sprey, Chico Sundermann, Sebastian Krieter, Michael Nieke, Jacopo Mauro, Thomas Thüm, and Ina Schaefer. SMT-Based Variability Analyses in FeatureIDE. February 2020. (cited on Page 1)

[STS20] Chico Sundermann, Thomas Thüm, and Ina Schaefer. Evaluating #sat solvers on industrial feature models. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*, pages 1–9, 2020. (cited on Page 1 and 4)

[TBK09] Thomas Thum, Don Batory, and Christian Kastner. Reasoning about edits to feature models. In *2009 IEEE 31st International Conference on Software Engineering*, pages 254–264. IEEE, 2009. (cited on Page 49)

[TKB+14] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 79:70–85, 2014. (cited on Page 2 and 21)

Task

Many software systems nowadays have a large number of features with a much larger number of different variants, e.g., the operating system of a mobile phone. A software product line (SPL) is used to manage the features and to provide reusability of their functionalities. For a better overview on the features, a feature model (FM) organizes them hierarchically in a tree structure that describes the dependencies between the features. With an FM, all valid combinations of features, called configurations, are defined, and a configuration's validity can be tested using the FM. The logic of an FM can be represented by a Boolean formula.
However, when selecting specific features while constructing an individual configuration, it is possible that other features become either implicitly selected or deselected due to constraints in the FM. To prevent the user from selecting an invalid combination of features, the configuration validity needs to be tested during this process, which is called the *online phase*. Validating the configuration during the online phase to inform the user about resulting consequences when selecting or deselecting features is called *decision propagation*. The validity check can be done by a SAT solver, that checks whether a Boolean term is satisfiable. Calling a SAT solver each time a feature gets selected or deselected to check which other features are still valid to be selected costs a lot of time in the decision propagation. To improve this situation, Modal Implication Graphs (MIGs) have been devised that can be used to identify implicit selection during the configuration process. The MIG is computed only once in the *offline phase*, to be used afterwards in the online phase.
A conjunctive normal form (CNF) as a logical representation of the FM serves as base for the computation of a MIG. A CNF describes a FM as a logical formula and contains a number of clauses that consist of literals. A clause represents the dependency between features by connecting the literals with a logical OR. Each literal represents the selection or deselection of a feature (e.g., having two features A and B with A$\leftrightarrow$B, the clause would be ($\neg$A, B)). All clauses are connected with a logical AND. A MIG consists of nodes, where each node represents a literal from the CNF. The nodes are connected by strong and weak edges that represent the connection between the features. Therefore, clauses with only one literal are ignored, a clause with two literals is represented by a strong edge and a clause with more than two literals by a weak edge. Despite the fact that a lot of time is saved during the configuration process, constructing the MIG in the first place takes a huge amount of time. Changing the FM in any way results in a changed CNF and, thus, requires a recalculation of the respective MIG. This is even necessary if the changes to the FM are small. Hence, a technique to incrementally compute a MIG based on FM changes could make the use of MIGs more efficient.

Introducing an algorithm for the incremental computation of a MIG

In this thesis, I will provide a method to construct an incremental MIG for an evolving FM. I will use the changes in the resulting CNF and compute their impact on the respective MIG. As basis for the incremental algorithm, I calculate the difference between the CNF before and after the changes. During the construction of an algorithm for the computation, I differentiate between complete and incomplete MIGs. An incomplete MIG does not contain all possible strong edges but is correct anyway. A complete MIG cannot be completely derived by the CNF, but requires additional computations by a SAT solver. Therefore, I will focus on incomplete MIGs in the first place. In the end, I will analyze the impact of the feature-model evolution operations (e.g., create or move a feature) on the CNF and investigate on how to improve the computation time of my algorithm.
To evaluate my method, I will test the correctness of the resulting graph. To do so, I will use small samples, generate the MIG, and evaluate the result. In addition, I will compare the time my algorithm needs to adapt the MIG with the time the original algorithm needs to recalculate the entire MIG. To this end, I will use different sizes of FMs as well as different numbers of changes that have been made since the last calculation. I will also compare the resulting MIGs by their size to inspect that the adapted MIG is not significantly larger than the newly calculated one.

For the development process I consider the following work packages:
Work package 1: Literature Study. I will study research on informations about *"Software Product Lines"*, *"Modal Implication Graphs"*, *"Differencing Algorithms"*, *"SAT Solver"*
Work package 2: Concept development. I subdivide the second work package in several smaller tasks:

- The first package is to find a concept for the incomplete graph to adapt. Therefore, I will develop an algorithm that detects the difference between the CNF before and after changes in the FM and adapts the MIG. I will do so for incomplete MIGs.
- In the second package, I need to analyze the impact of changes in the FM on the CNF and, accordingly, on the MIG.
- In the third package, I will investigate the possibility to extend the concept for complete graphs.
- The fourth package is to investigate whether knowing what operations have been made can simplify the process in some cases.

Work package 3: Implementation. This includes implementing the aforementioned algorithm in FeatureIDE, where it will be used when a MIG needs to be recalculated.
Work package 4: Evaluation. I will examine the correctness of my method as well as its computation time and the size of the resulting MIG.
Work package 5: Writing of the thesis.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Braunschweig, den 08. Januar 2021