On the Scalability of Building Binary Decision Diagrams for Current Feature Models

Tobias Heß
University of Ulm
Germany

Chico Sundermann
University of Ulm
Germany

Thomas Thüm
University of Ulm
Germany

ABSTRACT

Binary decision diagrams (BDD) have been proposed for numerous product-line analyses. These analyses typically exploit properties unique to decision diagrams, such as negation in constant time and space. Furthermore, the existence of a BDD representing the configuration space of a product line removes the need to employ SAT or #SAT solvers for their analysis. Recent work has shown that the performance of state-of-the-art BDD libraries is significantly lower than previously reported and hypothesized. In this work, we provide an assessment of the state-of-the-art of BDD scalability in this domain and explain why previous results on the scalability of BDDs do not apply to more recent product-line instances.

ACM Reference Format:
Tobias Heß, Chico Sundermann, and Thomas Thüm. 2021. On the Scalability of Building Binary Decision Diagrams for Current Feature Models. In *25th ACM International Systems and Software Product Line Conference - Volume A (SPLC '21), September 6–11, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3461001.3474452

1 INTRODUCTION

Feature models are commonly used to represent the variability in product lines [8, 9]. Manually analyzing feature models is infeasible, even for comparably small models. Automated analyses are typically realized by translating the feature model into propositional logic in conjunctive normal form (CNF) and expressing analysis queries in terms of satisfiability (SAT) problems [9, 58]. Subsequently, SAT solvers are used to decide the queries and #SAT solvers are employed to count the number of satisfying assignments of queries [54]. It is widely accepted that SAT solvers scale very well to instances induced by analyses of feature models [35, 41] and we recently extended this favorable assertion to #SAT and all but the most complex feature models [55]. As a consequence, analyses based on SAT [9] (e.g., verifying if a feature is dead) or on #SAT (e.g., how many valid configurations include a certain feature [9, 54]), scale reasonably well to be used in practice.

There are, however, analyses of crucial importance to practical product-line engineering that cannot be feasibly expressed in terms of conjunctive normal form and are therefore elusive to the approaches above [6, 57, 59]. A prime example is the computation of differences between feature model versions. Acher et al. argue that these differences should themselves be expressed in terms of a feature model [6]. Construction of the latter requires the negation of the propositional formula (typically in conjunctive normal form) representing one of the input feature models. It is well known [59] that this causes an exponential increase in the size of the formula and thus another encoding for the formulas is required. The authors therefore advocate for the use of binary decision diagrams (BDD) which allow for negation in constant time and space as well as composition and disjunction in polynomial time and space [15, 16].

These properties are unique to BDDs and closely related variants, for example zero-suppressed decision diagrams [43]. In particular, they cannot be found in data structures based on negation normal forms, such as deterministic, decomposable negation normal forms [19, 20]. Furthermore, the existence of a BDD representing a feature model removes the need to employ SAT or #SAT solvers, as all respective queries can be solved with BDDs in at most linear time, with respect to their number of nodes [11, 15].

As SAT and #SAT are well-known to be NP-complete [17] and #P-complete [13, 61], respectively, the complexity of solving these problems is shifted towards the construction of the BDD, which in turn is intractable in general. This replacement of multiple expensive computations by a single expensive computation yielding a readily evaluated data structure, is commonly referred to as knowledge compilation [21, 57]. Naturally, the question of scalability arises and one is interested in bounding it by means of easily calculable metrics, i.e., "technique x scales when metric $y \leq z$".

In the case of BDDs and feature models, many authors [3–6, 9, 44, 51, 56, 66] reference a result of Mendonça et al. from 2008 stating that BDDs can be constructed to feature models with up to 2,000 features [42]. Curiously, in a recent large-scale evaluation of the performance of #SAT solvers, we were unable to construct BDDs for a variety of real-world feature models, many of them with less than 1,250 features [55]. On the other hand, Fernández-Amorós et al. were able to construct BDDs for feature models with as many as 17,365 features [22]. Therefore, we argue that the result of Mendonça et al. does not transfer to current product lines and that it is therefore unknown when BDD-based analyses are feasible.

In this work, we quantify the performance of state-of-the-art BDD libraries for large real-world feature models and correlate our empirical results to commonly used feature-model metrics. Thereby, we explain why previous results on the scalability of BDDs do not hold for more recent real-world product-line instances. Contributing to the knowledge compilation challenge [57], it is our goal to establish a baseline for the performance of BDD libraries when constructing BDDs from feature models such that researchers and practitioners can better assess the challenges imposed by BDD-based analyses.

SPLC '21, September 6–11, 2021, Leicester, United Kingdom

ACM ISBN 978-1-4503-8469-8/21/09…$15.00
https://doi.org/10.1145/3461001.3474452

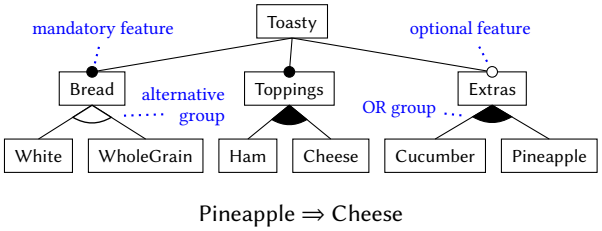

Figure 1: Running Example Feature Model

2 BACKGROUND

Feature Models. Figure 1 contains our running example, a toy product line for configuring tasty toasties. We use this feature model to introduce the mapping between feature model components and propositional logic. The selection of any feature mandates the selection of its parent feature (e.g., the selection of Cucumber implies the selection of Extras). Mandatory features must be selected when their parent feature is selected (e.g., every valid toasty has the feature Bread selected). Exactly one feature of an Alternative group must be selected when the parent feature is selected (e.g., every valid toasty has either White or WholeGrain). At least one feature of an Or group must be selected when the parent feature is selected (e.g., every valid toasty contains Ham or Cheese). Optional features induce no additional constraints on the configuration space, besides implying the selection of their parent feature. Cross-tree constraints impose additional constraints on the configuration space (in our example, the selection of Pineapple forces the selection of Cheese).

While a variety of different feature model notations has been described in the literature [8, 18, 32], we define a feature model as a tuple $\text{FM} = (\mathcal{F}, C = C_{\text{FD}} \cup C_{\text{CT}})$ over a set $\mathcal{F}$ of features and a set C of Boolean constraints, containing both the constraints C_{FD} induced by the feature diagram and the cross-tree constraints C_{CT}.

SAT and #SAT. Let $F : \{0,1\}^n \rightarrow \{0,1\}$ be a Boolean function. F is called *satisfiable* if $\exists v \in \{0,1\}^n$ with $F(v) = 1$. We write $|F| = |\{v \in \{0,1\}^n \mid F(v) = 1\}|$ to denote the number of variable assignments satisfying F. SAT refers to the problem of deciding satisfiability for a Boolean function, #SAT refers to the problem of calculating $|F|$.

Binary Decision Diagrams. Binary decision diagrams (BDD) represent Boolean functions as directed acyclic graphs with a single root node and two terminal nodes 0 and 1. Every non-terminal node u is associated with a variable x of the represented Boolean function and has precisely two outgoing edges, named *low* and *high*. The low edge represents the assignment $x = 0$ (i.e., false), the high edge the assignment $x = 1$ (i.e., true). A BDD is *ordered*, if nodes associated to a variable x_i always precede nodes associated to another variable x_j or vice versa [15]. A BDD is *reduced*, if it (1) does not contain nodes with their low and high edges incident with the same node and (2) no two nodes associated with the same variable have the same nodes incident to their low and high edges [15]. Figure 2 displays a BDD representing the configuration space of our running example (Figure 1).

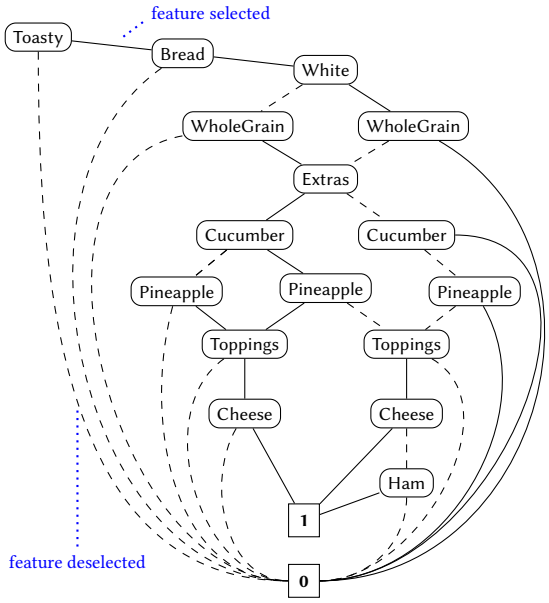

Figure 2: BDD for the Running Example

3 PERFORMANCE OF BDD LIBRARIES

In this section, we present the results of our performance evaluation of state-of-the-art BDD libraries in the context of feature-model analysis. We evaluate the performance of the two most-widely employed BDD libraries on a diverse set of real-world feature models. Subsequently, we observe correlations between feature model metrics and BDD library performance.

3.1 Evaluated Libraries and Subject Systems

Libraries. While we found over 100 BDD libraries when surveying the literature,[1] we restrict ourselves to the BDD libraries BuDDy [1] and CUDD [2] as they are the default libraries[2] for BDD-based analyses in the product-line domain [10, 11, 14, 22, 23, 25, 26, 42, 45, 46, 66] and in general [27–29, 31, 34, 47, 62, 68]. Despite their age of more than 25 years and the lack of maintenance or development they received in the past decade, many authors continue to view them as the state-of-the-art for BDD libraries [22, 23, 30, 31, 37, 38, 46, 62]. We also considered the BDD library Sylvan [63, 64]. However, while the authors of Sylvan report comparable performance for smaller problems [62], we were unable to achieve competitive performance to BuDDy or CUDD in a preliminary experiment for the feature models in our evaluation.

Neither BuDDy nor CUDD can be used as a black box, as they lack the functionality for parsing input files. Therefore, we use wrappers to interface with them. To reduce the influence of the wrapper on the performance of the library, we use three different wrappers. JavaBDD [67] has been used extensively by previous feature-model analysis research [10, 11, 14, 26, 42, 46, 49, 66] and we extended it to parse DIMACS. Regrettably, it only supports CUDD up to version 2.5.1 and not the latest release, 3.0.0. Logic2BDD [22]

[1] https://h3ssto.github.io/DDs/
[2] Sometimes via JavaBDD [67] using the Java Native Interface.

Table 1: Subject System Details

Name / Family	#Models	$\lvert\mathcal{F}\rvert$	$\lvert C_{\mathrm{CT}}\rvert$	$\lvert\mathcal{F}\rvert$ / $\lvert C_{\mathrm{CT}}\rvert$	ECR	#Clauses
Automotive01	1	2,513	2,833	0.89	50.9%	10,275
Automotive02v1-4	4	14,010–18,616	666–1,369	21.04–13.60	5.7–8.1%	237,702–350,187
BusyBox	1	854	123	0.14	26.3%	1,315
CDL	10	1,202–1,333	843–935	1.38–1.48	81.3–84.6%	2,973–3,270
FinancialServices01	1	771	1,080	0.71	89.9%	7,238
Linux	1	6,467	3,545	1.82	61.7%	42,264

is an extension of CUDD that reads the SXFM file format [40]. Before calling CUDD, it performs a simplification of the input, often reducing the number of clauses significantly, when compared to DIMACS. Furthermore, it directly constructs sub-BDDs for Alternative groups in the feature model, leading to smaller intermediate BDDs during BDD compilation. The third wrapper, ddueruem,[3] is our prototype and is being purpose-build to facilitate knowledge compilation for analyses and applications in the product-line domain. It currently interfaces to BuDDy and CUDD, is capable of parsing DIMACS or UVL [53], and contains an implementation of the FORCE pre-order heuristic [7].

Subject Systems. For our evaluation, we use the most complex real-world feature models from the FeatureIDE [39] repository (see Table 1). Many of the feature models previously appeared in other benchmarks [33, 55]. `Automotive01` and `Automotive02` are feature models from the eponymous automotive domain [33]. BusyBox is a software suite of Unix utility tools. The CDL feature models originate form the eCos operating system and have been translated from the component definition language [12, 65]. As the CDL feature models are known to be very similar [55], we randomly selected only ten of the more than 120 models. FinancialServices01 is a feature model from the financial sector [24]. Finally, Linux is the feature model for the Linux kernel version 2.6.33.3, translated from KConfig [36, 50]. We refrained from considering small feature models, as the performance of BDD libraries for small feature models has been thoroughly investigated in the past [22, 42, 46].

3.2 Setup

Experiment Design. We performed five runs per model (18 models) and wrapper configuration (five configurations) (i.e., selected library), 450 in total. Every run was limited to 10 minutes of runtime. When a run encountered the time limit, we tracked the percentage of clauses that was successfully incorporated into the BDD. All experiments ran single-threaded on a machine with an Intel i7-10510U and 32 GB of RAM.

Library and Wrapper Configuration. ddueruem was configured to pre-order the variables and clauses of the input with the FORCE heuristic [7] for a maximum of 60 s. BuDDy and CUDD were configured according to their respective manuals and were set to use the converging variant of the SIFT heuristic [48] for dynamic variable ordering. We followed the examples of JavaBDD to implement the construction of the BDD and chose setting parameters as in the manual [67]. We used Logic2BDD as a black box, by using the scripts provided by its authors [22].

[3] https://github.com/h3ssto/ddueruem/releases/tag/v2021-02

3.3 Results

The results of our experiments are depicted in Table 2. For every feature model and experiment configuration, we state the median of the five runs for either the construction time (in the case of successful construction) or the percentage of successfully incorporated clauses (in the case of timeouts). For the grouped feature models (e.g., CDL), we specify the average and standard deviation of the median percentages. In total, we were only able to compute BDDs for three feature models within our time limit, namely Automotive02v1, BusyBox, and FinancialServices01. Logic2BDD significantly outperformed the other experiment configurations.

Metrics. The number of features does not appear to be a good indicator for the performance to expect from BDD libraries, as we were able to construct BDDs for feature models with 771 and 14,616 features, but failed to do so for all of the CDL feature models with 1,202–1,333 features. Similarly, the ratio between the number of features and the number of cross-tree constraints (i.e., $\lvert\mathcal{F}\rvert$ / $\lvert C_{\mathrm{CT}}\rvert$) does also not appear to be good indicator, as we compiled BDDs for feature models with ratios of 0.14, 0.71, and 21.04 but failed for feature models with ratios of, for example, 0.89, 1.82, or 13.6. Lastly, the ECR, the ratio of features appearing in cross-tree constraints [42], provides little insight as well, as we compiled BDDs for feature models with an ECR of 5.7%, 26.3%, and 89.9%, but failed for feature models with an ECR of, for example, 6.4%, 50.9%, or 83.2% (CDL average). We conclude that none of the previously employed metrics is well-suited to predict BDD compilation performance.

Compilation Strategy. The differences in performance between Logic2BDD and the other experiment configurations, suggest that it is beneficial to take the structure of the feature model into account. Similar observations were made by Mendonça et al. for artificial feature models [42] and by the authors of Logic2BDD for low-constrained real-world feature models [22]. However, the results on the more complex feature models, Automotive01 and Linux suggest that this mostly benefits the performance on low-constrained feature models. Nevertheless, the results suggest that CNF is not a desirable starting point when building BDDs.

Time Limit. We also investigated the influence of the time limit on the outcome of our experiment. While the results seem to suggest that Logic2BDD was within seconds of completing the compilation for Automotive01, Automotive02v2, and the CDL feature models, this is not the case. For illustration, Figure 3 depicts the number of incorporated constraints (after simplification) over a time period of more than 24 hours for Automotive01. Note that 99% of the constraints have been incorporated into the BDD after merely 10 s (< 2% of allowed computation time) and the remaining time was spend

Table 2: Evaluation Results (< 100% indicates a timeout)

	JavaBDD				Logic2BDD		ddueruem			
	BuDDy (2.4)		CUDD (2.5.1)				BuDDy (2.4)		CUDD (3.0.0)	
Name / Family	Time	%Clauses	Time	%Clauses	Time	%Clauses	Time	%Clauses	Time	%Clauses
Automotive01		7.2%		8.2%		99.7%		76.4%		25.9%
Automotive02v1		1.2%		1.3%	13.3 s	100%		1.3%		5.4%
Automotive02v2-4		0.4%		0.7%		93.3 ± 5.7%		0.55 ± 0.05%		3.9 ± 1.0%
BusyBox		37.2%		39.8%	0.1 s	100%		48.3%	89.7 s	100%
CDL		18.5 ± 0.4%		21.5 ± 0.8%		97.7 ± 1.2%		88.9 ± 2.1%		70.1 ± 2.3%
FinancialServices01		3.5%		3.6%	160.5 s	100%		54.2%		61.4%
Linux		2.7%		2.9%		82.5%		17.2%		15.2%

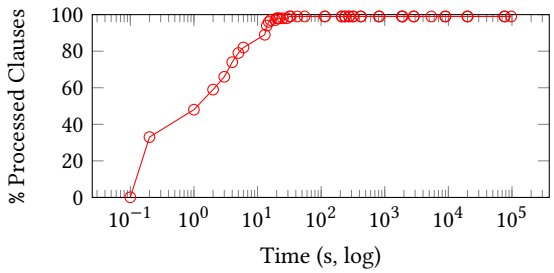

Figure 3: Insights Into the Compilation Of Automotive01

incorporating the remaining 1% of constraints, before terminating with seven not incorporated constraints remaining, 5767 of 5774. We observed comparable effects for the CDL feature models. However, we want to note that Logic2BDD was able to compile BDDs for Automotive2v2+ in about 30 minutes, respectively.

3.4 Threats to Validity

Translation from Subject System to Feature Model. The translation of the subject systems into feature models might be incorrect. However, the automotive and financial feature models have been verified by experts from our industrial partners and the threats originating from the CDL and KConfig feature models have been extensively analyzed by Knüppel et al. and were found to be insignificant [33].

Translation from Feature Model to Input Format. The translation of the feature models into the input formats might be incorrect. We used FeatureIDE [39] to translate the feature models into DIMACS and SXFM. We performed the following sanity checks to assess the correctness of the translation. First, we verified the correctness manually for some very small feature models. Second, we compiled BDDs for some small real-world feature models, were the compilation is performed effortlessly by all the experiment configurations. We verified the correctness of the BDDs by testing their output for a variety of known valid and invalid product configurations (generated and verified with FeatureIDE [39]) and computed the number of satisfying assignments. We cross-checked the results and compared them with known values and our previous work [55]. Third, we assured that the translation was performed without the introduction of additional variables, i.e., without invoking the Tseytin transformation [60]. All sanity checks concluded with success.

Wrapper Implementation. The implementation of the wrappers might be inefficient or incorrect. We verified their correctness by using the sanity checks from above. The interface of ddueruem is implemented in accordance with the respective manuals of BuDDy [1] and CUDD [52], as is our modification of JavaBDD [67]. JavaBDD was compiled and executed in Java 15.

External Validity. We cannot claim that our results can be extended to all real-world configurable systems, as we only evaluated a small number of models. However, we considered the most complex publicly available feature models from three domains and four sources. While significant statements on compilation strategies and metrics require a considerably larger-scaled evaluation, in both runtime and number of instances, we argue that our results provide good indicators.

4 CONCLUSION

Our results show that the performance of BDD libraries has been significantly overestimated by previous research. We were only able to build BDDs for three of the 18 considered real-world feature models, many of them below the threshold of 2,000 features, established by Mendonça et al. for artificially generated feature models [42]. Thereby, we raise the issue of how the performance of BDD libraries in particular and feature-model analysis tooling in general can be quantified, predicted, or learned.

BDDs are the most versatile option in the knowledge compilation arsenal [21]. They are without alternative for some feature-model analyses (e.g., computing the difference of two feature-models [6]) and an alternative to SAT and #SAT solvers [9, 15, 55] when already computed for other uses. Therefore, we argue that it is crucial to improve their performance in the future.

REFERENCES

[1] 2007. http://buddy.sourceforge.net/manual/main.html. Accessed: 2020-03-02.
[2] 2016. https://github.com/vscosta/cudd. Accessed: 2020-06-13.
[3] 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, 45–54.
[4] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2011. Slicing Feature Models. In *ASE*. IEEE, 424–427.
[5] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2013. Familiar: A Domain-Specific Language for Large Scale Management of Feature Models. *SCP* 78, 6 (2013), 657–681.
[6] Mathieu Acher, Patrick Heymans, Philippe Collet, Clément Quinton, Philippe Lahire, and Philippe Merle. 2012. Feature Model Differences. In *CAiSE*. Springer, 629–645.

[7] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. 2003. FORCE: A Fast and Easy-to-Implement Variable-Ordering Heuristic. In *GLSVLSI*. ACM, 116–119.

[8] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *SPLC*. Springer, 7–20.

[9] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems* 35, 6 (2010), 615–708.

[10] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. 2006. A First Step Towards a Framework for the Automated Analysis of Feature Models. In *SPLC*. IEEE, 39–47.

[11] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. 2007. FAMA: Tooling a Framework for the Automated Analysis of Feature Models. In *VaMoS*. Technical Report 2007-01, Lero, 129–134.

[12] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. 2010. Variability Modeling in the Real: A Perspective from the Operating Systems Domain. In *ASE*. ACM, 73–82.

[13] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. 2009. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press.

[14] Eric Bodden, Társis Tolêdo, Márcio Ribeiro, Claus Brabrand, Paulo Borba, and Mira Mezini. 2013. SPLLIFT: Statically Analyzing Software Product Lines in Minutes Instead of Years. In *PLDI*. ACM, 355–364.

[15] Randal E. Bryant. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *TC* C-35, 8 (1986), 677–691.

[16] Randal E. Bryant. 2018. Binary Decision Diagrams. In *Handbook of Model Checking*. Springer, 191–217.

[17] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *STOC*. ACM.

[18] Krzysztof Czarnecki and Ulrich Eisenecker. 2000. *Generative Programming: Methods, Tools, and Applications*. ACM/Addison-Wesley.

[19] Adnan Darwiche. 2001. Decomposable Negation Normal Form. *J. ACM* 48, 4 (2001), 608–647.

[20] Adnan Darwiche. 2002. A Compiler for Deterministic, Decomposable Negation Normal Form. In *AAAI*. AAAI Press, 627–634.

[21] Adnan Darwiche and Pierre Marquis. 2002. A Knowledge Compilation Map. *JAIR* 17, 1 (2002), 229–264.

[22] David Fernández-Amorós, Sergio Bra, Ernesto Aranda-Escolástico, and Ruben Heradio. 2020. Using Extended Logical Primitives for Efficient BDD Building. *Mathematics* 8, 8 (2020), 1253:1–1253:17.

[23] David Fernandez-Amoros, Ruben Heradio, Christoph Mayr-Dorn, and Alexander Egyed. 2019. A KConfig Translation to Logic with One-Way Validation System. In *SPLC*. ACM, 303–308.

[24] Claudia Fritsch, Richard Abt, and Burkhardt Renz. 2020. The Benefits of a Feature Model in Banking. In *SPLC*. ACM, Article 9, 11 pages.

[25] Paul Gazzillo. 2017. Kmax: Finding All Configurations of Kbuild Makefiles Statically. In *ESEC/FSE*. ACM, 279–290.

[26] Paul Gazzillo and Robert Grimm. 2012. SuperC: Parsing All of C by Taming the Preprocessor. In *PLDI*. ACM, 323–334.

[27] Rajeev Goré and Jimmy Thomson. 2013. An Improved BDD Method for Intuitionistic Propositional Logic: BDDIntKt System Description. In *CADE*, Maria Paola Bonacina (Ed.), Vol. 7898. Springer, Berlin, Heidelberg, 275–281.

[28] Daniel F Gudbjartsson, Thorvaldur Thorvaldsson, Augustine Kong, Gunnar Gunnarsson, and Anna Ingolfsdottir. 2005. Allegro Version 2. *Nature Genetics* 37, 10 (2005), 1015–1016.

[29] Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. 2013. SPUDD: Stochastic Planning Using Decision Diagrams. *CoRR* (2013).

[30] Martin Jonáš and Jan Strejček. 2019. Q3B: An Efficient BDD-based SMT Solver for Quantified Bit-Vectors. In *CAV*. Springer, 64–73.

[31] Andrew V Jones and Alessio Lomuscio. 2010. Distributed BDD-Based BMC for the Verification of Multi-Agent Systems. In *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 675–682.

[32] Chang Hwan Peter Kim and Krzysztof Czarnecki. 2005. Synchronizing Cardinality-Based Feature Models and Their Specializations. In *ECMDA*. Springer, 331–348.

[33] 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 *ESEC/FSE*. ACM, 291–302.

[34] Ondřej Lhoták and Laurie Hendren. 2004. Jedd: A BDD-Based Relational Extension of Java. In *PLDI*. ACM, 158–169.

[35] Jia Hui Liang, Vijay Ganesh, Krzysztof Czarnecki, and Venkatesh Raman. 2015. SAT-Based Analysis of Large Real-World Feature Models Is Easy. In *SPLC*. Springer, 91–100.

[36] Rafael Lotufo, Steven She, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wąsowski. 2010. Evolution of the Linux Kernel Variability Model. In *SPLC*. Springer, 136–150.

[37] Guanfeng Lv, Yao Chen, Yachao Feng, Qingliang Chen, and Kaile Su. 2012. A Succinct and Efficient Implementation of a 2^ 32 BDD Package. In *Proc. Int'l Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE, 241–244.

[38] Guanfeng Lv, Kaile Su, and Yanyan Xu. 2013. CacBDD: A BDD Package With Dynamic Cache Management. In *CAV*. Springer, 229–234.

[39] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer.

[40] Marcílio Mendonça, Moises Branco, and Donald Cowan. 2009. S.P.L.O.T.: Software Product Lines Online Tools. In *OOPSLA*. ACM, 761–762.

[41] Marcílio Mendonça, Andrzej Wąsowski, and Krzysztof Czarnecki. 2009. SAT-Based Analysis of Feature Models is Easy. In *SPLC*. Software Engineering Institute, 231–240.

[42] Marcílio Mendonça, Andrzej Wąsowski, Krzysztof Czarnecki, and Donald Cowan. 2008. Efficient Compilation Techniques for Large Scale Feature Models. In *GPCE*. ACM, 13–22.

[43] Shin-ichi Minato. 1993. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *DAC*. ACM, 272–277.

[44] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding Near-Optimal Configurations in Product Lines by Random Sampling. In *ESEC/FSE*. 61–71.

[45] Héctor José Pérez Morago. 2016. *BDD Algorithms to Perform Hard Analysis Operations on Variability Models*. Ph.D. Dissertation. Universidad Nacional de Educación a Distancia.

[46] Richard Pohl, Kim Lauenroth, and Klaus Pohl. 2011. A Performance Comparison of Contemporary Algorithmic Approaches for Automated Analysis Operations on Feature Models. In *ASE*. IEEE, 313–322.

[47] Andrei Rimsa, Luis Enrique Zárate, and Mark A. J. Song. 2009. Evaluation of Different BDD Libraries to Extract Concepts in FCA - Perspectives and Limitations. In *ICCS (LNCS, Vol. 5544)*. Springer, 367–376.

[48] Richard Rudell. 1993. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *ICCAD*. IEEE, 42–47.

[49] Sergio Segura. 2008. Automated Analysis of Feature Models Using Atomic Sets. In *SPLC*, Vol. 2. IEEE, 201–207.

[50] Steven She and Thorsten Berger. 2010. *Formal Semantics of the Kconfig Language*. Technical Report. University of Waterloo.

[51] Steven She, Uwe Ryssel, Nele Andersen, Andrzej Wasowski, and Krzysztof Czarnecki. 2014. Efficient Synthesis of Feature Models. *IST* 56, 9 (2014), 1122–1143.

[52] Fabio Somenzi. 2005. *CUDD: User's Manual*.

[53] Chico Sundermann, Kevin Feichtinger, Dominik Engelhardt, Rick Rabiser, and Thomas Thüm. 2021. Yet Another Textual Variability Language? A Community Effort Towards a Unified Language. In *SPLC*. ACM. To appear.

[54] Chico Sundermann, Michael Nieke, Paul Maximilian Bittner, Tobias Heß, Thomas Thüm, and Ina Schaefer. 2021. Applications of #SAT Solvers on Feature Models. In *VaMoS*. ACM, Article 12, 10 pages.

[55] Chico Sundermann, Thomas Thüm, and Ina Schaefer. 2020. Evaluating #SAT Solvers on Industrial Feature Models. In *VaMoS*. ACM, Article 3, 9 pages.

[56] Reinhard Tartler, Christian Dietrich, Julio Sincero, Wolfgang Schröder-Preikschat, and Daniel Lohmann. 2014. Static Analysis of Variability in System Software: The 90,000 #Ifdefs Issue. In *ATC*. USENIX Association, 421–432.

[57] Thomas Thüm. 2020. A BDD for Linux? The Knowledge Compilation Challenge for Variability. In *SPLC*. ACM, Article 16, 6 pages.

[58] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *CSUR* 47, 1 (2014), 6:1–6:45.

[59] Thomas Thüm, Don Batory, and Christian Kästner. 2009. Reasoning About Edits to Feature Models. In *ICSE*. IEEE, 254–264.

[60] Grigori S. Tseytin. 1983. *On the Complexity of Derivation in Propositional Calculus*. Springer, 466–483.

[61] Leslie G Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SICOMP* 8, 3 (1979), 410–421.

[62] Tom van Dijk, Ernst Moritz Hahn, David N. Jansen, Yong Li, Thomas Neele, Mariëlle Stoelinga, Andrea Turrini, and Lijun Zhang. 2015. A Comparative Study of BDD Packages for Probabilistic Symbolic Model Checking. In *SETTA (LNCS, Vol. 9409)*. Springer, 35–51.

[63] Tom van Dijk, Jeroen Meijer, Ioannis Filippidis, Alfons Laarman, and Matthias Volk. 2020. https://github.com/trolando/sylvan. Accessed: 2021-03-01.

[64] Tom van Dijk and Jaco van de Pol. 2015. Sylvan: Multi-Core Decision Diagrams. In *TACAS*. Springer, 677–691.

[65] Bart Veer and John Dallaway. 2017. The eCos Component Writer's Guide. Manual. Available online at http://ecos.sourceware.org/ecos/docs-3.0/pdf/ecos-3.0-cdl-guide-a4.pdf; visited on May 10th, 2017.

[66] Alexander von Rhein, Alexander Grebhahn, Sven Apel, Norbert Siegmund, Dirk Beyer, and Thorsten Berger. 2015. Presence-Condition Simplification in Highly Configurable Systems. In *ICSE*. IEEE, 178–188.

[67] John Whaley. 2007. JavaBDD. http://javabdd.sourceforge.net/. Accessed: 2021-04-21.

[68] Tianhua Xu, Haifeng Wang, Tangming Yuan, and MengChu Zhou. 2016. BDD-Based Synthesis of Fail-Safe Supervisory Controllers for Safety-Critical Discrete Event Systems. 17, 9 (2016), 2385–2394.