A BDD for Linux?
The Knowledge Compilation Challenge for Variability

Thomas Thüm
University of Ulm, Germany
thomas.thuem@uni-ulm.de

ABSTRACT
What is the number of valid configurations for Linux? How to generate uniform random samples for Linux? Can we create a binary decision diagram for Linux? It seems that the product-line community tries hard to answer such questions for Linux and other configurable systems. However, attempts are often not published due to the publication bias (i.e., unsuccessful attempts are not published). As a consequence, researchers keep trying by potentially spending redundant effort. The goal of this challenge is to guide research on these computationally complex problems and to foster the exchange between researchers and practitioners.

CCS CONCEPTS
• Software and its engineering → Formal software verification; Software testing and debugging; Software verification; Automated static analysis; Consistency; Software configuration management and version control systems; Preprocessors; • Theory of computation → Program verification; Program analysis; Logic and verification

KEYWORDS
software product line, configurable system, software configuration, product configuration, feature models, decision models, artificial intelligence, satisfiability solving, knowledge compilation, binary decision diagrams

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 MOTIVATION
What is the holy grail of the product-line community? A binary decision diagram for Linux.

Linux is an operating system with thousands of configuration options (cf. Figure 1). These options cannot be arbitrarily combined, as every option typically comes with constraints with respect to several other options. Constraints are specified in KConfig [20], but can be translated into feature models or propositional logic [11, 31, 33, 35, 38, 47, 55, 66, 69, 75, 79–81]. Whenever we analyze the Linux kernel for errors, ignoring those constraints would lead to false positives. That is, tools would report errors for invalid configurations, which cannot be used to compile a kernel. Hence, these constraints are crucial for any kind of analysis of Linux.

A binary decision diagram (short BDD) is data structure representing a propositional formula. While there are multiple representations of propositional formulas, BDDs have the advantage of reducing NP-complete problems into more tractable problems (aka. knowledge compilation). For instance, checking whether a formula represented as a BDD is satisfiable is an operation with constant effort. While operations on BDDs often scale well, the downside of BDDs is that their construction can be intractable. The main reason for the scalability challenge is that the variable ordering heavily influences the size of the BDD and they tend to explode for most variable orderings.

Why do we argue that a BDD for Linux is the holy grail of the product-line community?

First, it seems to be a challenging task. To the best of our knowledge, no one has been able to create a BDD for Linux so far. In recent work, we tried to create a BDD for hundreds of large feature
models and failed for 98% of them (i.e., 100% of models having more than 550 features) [78]. Furthermore, even though other knowledge compilation techniques scaled to many large feature models, not a single knowledge compilation tool scaled to Linux when we tried to compute the number of valid configurations [78]. Similarly, t-wise sampling algorithms typically do not scale to Linux [67].

Second, there is a considerable amount of research that is based on the translation of the feature model into a BDD. In the past two decades, researchers proposed the use of BDDs to count the number of valid configurations [8, 10, 30, 59, 68, 78], to compute feature-model slices [1] and differences [2], for interactive product configuration [42], to check whether product-line artifacts are consistent [28, 77], or to simplify preprocessor-based product lines [39], to simplify preprocessor annotations [89, 91], and to lift test-suite generation [14], dataflow analyses [12, 13], or model checking [5, 7, 16, 17, 19, 21–24, 41, 88, 89] to product lines. If we aim to apply that research to Linux or similarly complex configuration spaces, it is an open question whether BDDs can be created for them.

Third, there are many advantages of having a BDD. While it is widely accepted that satisfiability solving scales well for product lines [60, 85], the potential advantage of a BDD is that one-time effort for the construction can pay-off when the BDD is later employed in follow-up analyses. The potential is amplified by several factors. First, a feature model is typically changed less frequently than implementation artifacts [40, 52, 67]. As a consequence, a new BDD only needs to be created when the feature model changes. Second, for every new revision of the product line there are several analyses needed of which each typically is reduced to numerous satisfiability problems. It is likely that a large portion of the satisfiability problems can be solved more efficiently using a BDD.

What is the goal of this challenge?

While our claim about the holy grail is focused on BDDs and Linux, there is a more general challenge behind this specific one. Linux is just one example of a large-scale configuration space. A BDD is just one example of a knowledge compilation technique [30]. The goal of this challenge is to promote the problem of knowledge compilation for large-scale configuration spaces. In particular, this challenge is not only focused on software configuration, but also configurable systems and product configuration. Furthermore, any representation of the configuration space that invests offline computations (i.e., compilation) in favor of faster online computations would fall into the scope of this challenge.

What is the problem of the current situation? Many product-line researchers seem to address very similar problems for various reasons, but largely without documenting their failed attempts. Other researchers are most likely repeating the same mistakes again. The goal of this challenge is to make this problem explicit, to give researchers a forum to discuss attempts, and to exchange ideas on possible solutions.

2 STATE-OF-THE-ART

Knowledge compilation is the process of translating a propositional formula into a target language offline, which is then used online to answer numerous queries more efficiently (i.e., in polytime) [30]. Ideally, such a target language fulfills three properties, namely the target representation is small, many classes of queries run efficiently, and it can be efficiently translated into other target representations. Besides BDDs, there are numerous other target languages of knowledge compilation which are typically variations of conjunctive normal form or disjunctive normal form [30].

Valid combinations of features in product lines or configuration options in configurable software are typically represented by feature models, decision models, or other kinds of variability models [4, 6, 25, 26, 32, 37, 43, 46, 62, 71]. To reason about these constraints, variability models are typically translated into propositional logic [6, 29, 44]. In the past three decades, a myriad of analyses have been proposed that require reasoning about constraints. These include automated analyses of feature models [9, 36] and analyses also incorporating other domain artifacts [84, 90]. In particular, a logical representation has been used for feature-model evolution [65, 85], feature-model interfaces and slicing [1, 72], computation of implicit constraints [3], product configuration [70] including staged configuration [27], parsing [48], dead-code analysis [81], code simplification [91], type checking [83], consistency checking [28], dataflow analyses [54], model checking [18], testing [15] including variability-aware execution [64] and sampling [56, 87], optimization of non-functional properties [76], and variant-preserving refactoring [34]. Each of these analyses may profit from knowledge compilation. As it is likely that numerous of such analyses are combined in practice, it is even more beneficial to invest in some offline computations if those help to speed-up several analyses later on.

Knowledge compilation has been used to reason about configuration spaces for more than one decade. Czarnecki and Pietroszek employed multiple solvers, such as SAT, CSP, and BDDs for the analysis of feature models in the FaMa framework [8, 10]. Mendonça investigated heuristics to scale BDDs to product lines in his dissertation [58, 61]. Besides randomly generated feature models with up-to 3,000 features, he exercised academic product lines from the SPLOT repository with up-to 200 features. Kübler et al. applied existing knowledge compilation techniques and a proprietary compilation technique to compute the number of valid configurations as well as the relative frequency of components [53]. They evaluated those techniques on automotive product lines from Mercedes-Benz with between 5,000 to 10,000 features. Only the proprietary technique was able to scale to 10,000 features with a runtime of about two hours. Pohl et al. investigated nine knowledge compilation techniques, including BDDs, CSP, and SAT solvers to compute the number of valid configurations [68]. While they only evaluated small academic product lines from the SPLOT repository, we recently extended their study with the largest known feature models from open-source projects and proprietary models [78]. While some knowledge compilation techniques scaled to all systems except Linux and one automotive product line, BDD did not scale for almost all models. In summary, when knowledge compilation is applied to product lines, often significantly smaller product lines than Linux are used. As far as we know, the largest known BDD for a real system has only hundreds of features, while recent Linux versions have more than 17,000 features.
We call for a community effort to advance the state-of-the-art on we call for related submissions:

| A BDD for Linux? The Knowledge Compilation Challenge for Variability SPLC’20, 19–23 October, 2020, Montreal, Canada |

We explicitly invite researchers and practitioners to document their efforts with knowledge compilation for product lines. Besides a BDD for Linux, we call for related submissions:

- BDDs for parts of Linux or older versions containing fewer features and BDDs for Linux ignoring some constraints.
- BDDs for real-world configuration spaces (cf. existing benchmarks [50, 67]) or randomly generated models with thousands of features (cf. existing generators [60, 73, 74, 85]).
- Application of existing knowledge compilation techniques beyond BDDs [30] to variability or development of new knowledge compilation techniques dedicated to variability.
- Strategies for incremental knowledge compilation to cope with the evolution of configuration spaces.
- Attempts to solve any of the above challenges, as this document can prevent others from redundant research.

For comparability of solutions, we recommend using one or several of the following three benchmarks. We have translated the KConfig model of Linux in more than 400 revisions between November 2013 and January 2018 [67]. That is, not every commit of Linux is considered, but only those that actually alter the KConfig model. The models have been translated into several formats using KConfigReader [47] and the FeatureIDE library [51, 57] and are available in an online repository. Nevertheless, the benchmark has two disadvantages. First, as the translation with KConfigReader uses the Tseytin transformation [86], the models contain more than 60,000 variables. Second, the feature models are flat and do not preserve the hierarchy of the features in KConfig. To the best of our knowledge, there is no better translation by now.

In a prior study [50], we translated more than 100 models from KConfig and the Component Definition Language (CDL) [2]. These models have been translated with an extension of the Linux Variability Analysis Tools (LVAT) [11]. The advantage over the previous benchmark is that the hierarchy of features is available and that models are available in a version without newly introduced variables. While the benchmark also contains Linux, the feature model represents version 2.6.33.3, which has been released in April 2010.

In our collaborations with industry, we have been able to publish feature models of commercial models [65, 67, 72]. The Automotive02 models [72] and FinancialServices01 models [65] originate from the automotive and financial services industry, respectively. These models are available as monthly snapshots and feature names are obfuscated. The obfuscation algorithm being used ensures that feature names are consistently replaced over the evolution history. In four revisions, Automotive02 grew from 14,010 features and 666 constraints to a version with 18,616 features and 1,339 constraints. Similarly, FinancialServices01 grew from 557 features and 1,001 constraints to a product line with 771 features and 1,080 constraints in ten revisions.

We ask submitters to accompany their solution with a number of metrics for later comparisons. Besides the target product line, we ask about the actual source and its version. If translations were used to translate variability models into another language or into a logical representation, solutions should describe the techniques or tools being used. For the target product lines, statistics should be reported, such as the number of features, constraints, and clauses (i.e., in conjunctive or disjunctive normal form). If only a subset of the feature model is compiled, the percentage of covered features, constraints, and clauses is to be specified. For the compilation, time and memory consumption with respect to the used hardware and software should be specified. In case of success, we also expect metrics on the result of knowledge compilation, such as the number of nodes for a BDD or the number of literals of a normal form.

Acknowledgments

In the past years, I have had numerous fruitful discussions on this topic with other researchers. In particular, I would like to thank Tobias Heß, Chico Sundermann, Sebastian Krieter, Tobias PItt, Ina Schaefer, Christian Kästner, Jeffrey Young, Eric Walkingshaw, Eric Boddin, Andrzej Wąsowski, and Leopoldo Teixeira.

References


https://github.com/PettTo/Feature-Model-History-of-Linux
https://github.com/AlexanderKnueppel/is-there-a-mismatch/tree/master/Data/LargeFeatureModels
https://code.google.com/archive/p/linux-variability-analysis-tools/


