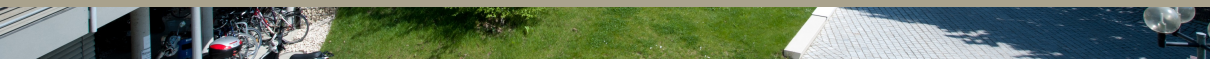
A Binary Decision Diagram for Linux?

The Knowledge Compilation Challenge for Variability | Thomas Thüm | October 22nd, 2020

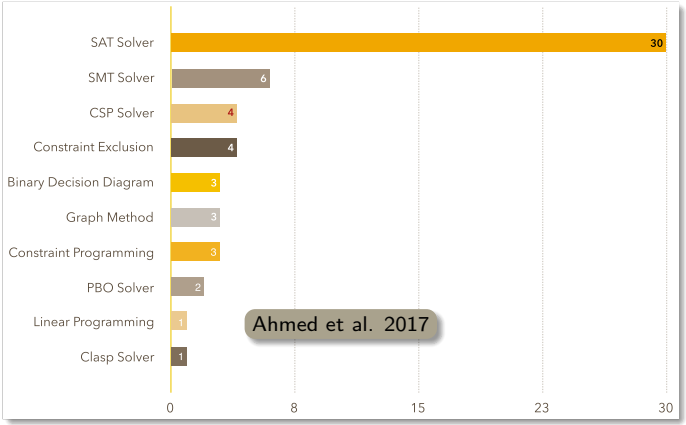


Software Engineering
Programming Languages

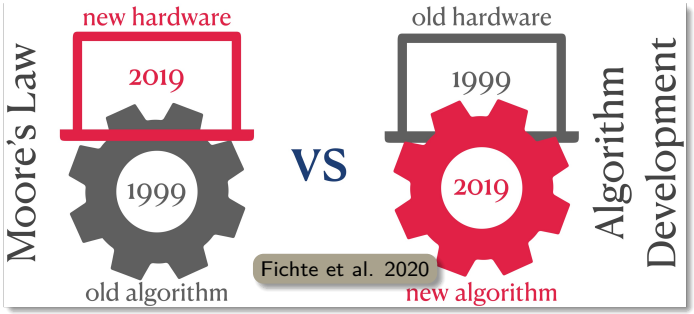
ulm university universität
uulm

Product Lines are Powered by SAT

Used Solvers (for Interaction Testing)



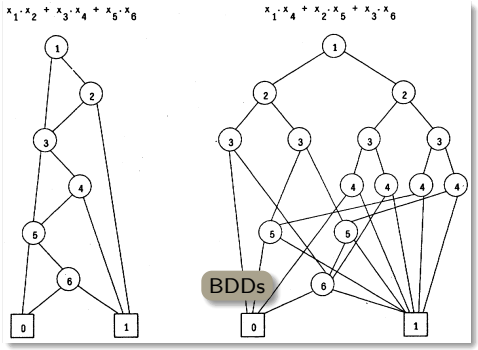
Time-Leap Challenge for SAT-Solving

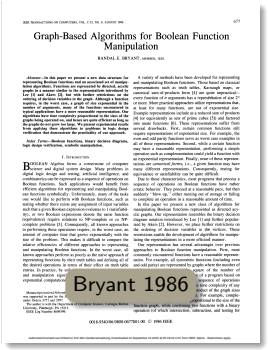


What is Our Holy Grail?

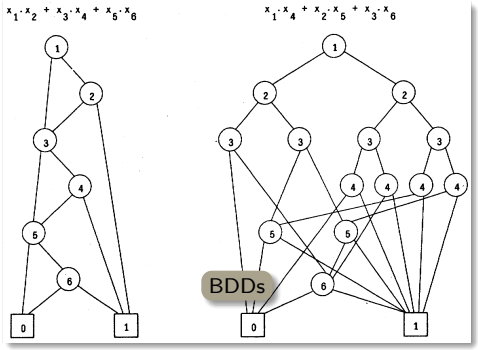
A BDD for Linux.

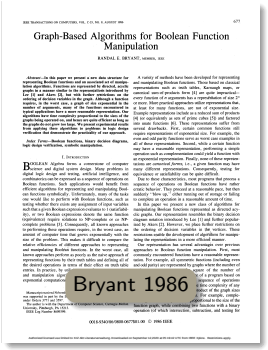
Why Binary Decision Diagrams?



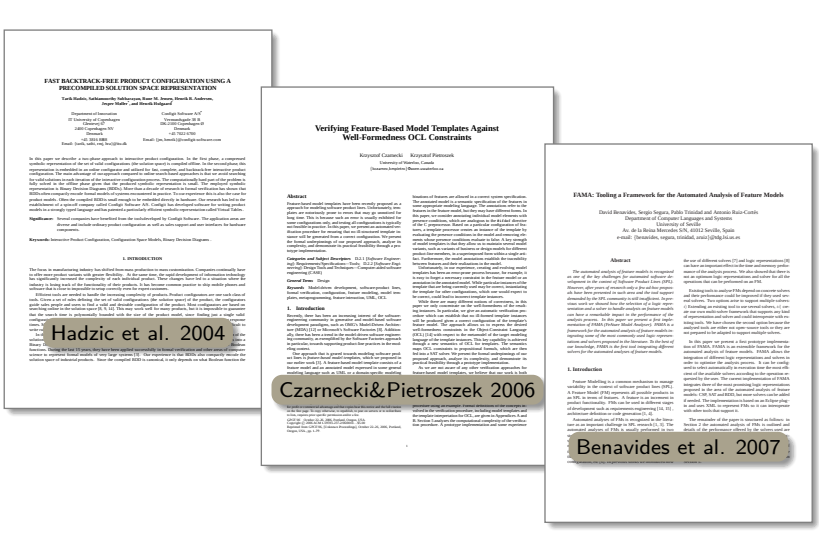


Many Queries on BDDs are fast.

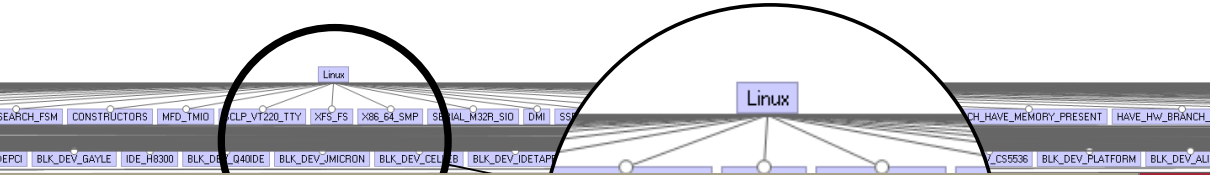




Use of BDDs for Product Lines



Why Linux?


Linux
Linux
CONSTRUCTORS
MFD_TMIO
XFS_FS
X86_64_SMP
DMI
BLK_DEV_GAYLE
IDE_H8300
BLK_DEV_JMICRON
BLK_DEV_IDETAPE
BLK_DEV_PLATFORM

Large Product-Lines Beyond Linux

Lazy Product Discovery in Huge Configuration Spaces

Michael Lienhardt
ONERA – The French Aerospace Lab
France
michael.lienhardt@onera.fr

Ferruccio Damiani
University of Turin
Italy
ferruccio.damiani@unito.it

Einar Broch Johnsen
University of Oslo
Norway
einarj@ifi.uio.no

Jacopo Mauro
University of Southern Denmark
Denmark
mauro@sdu.dk

ABSTRACT

Highly-configurable software systems can have thousands of interdependent configuration options across different subsystems. In the resulting configuration space, discovering a valid product configuration for some selected options can be complex and error prone. The configuration space can be organized using a feature model, fragmented into smaller interdependent feature models reflecting the configuration options of each subsystem.

We propose a method for lazy product discovery in large fragmented feature models with interdependent features. We formalize the method and prove its soundness and completeness. The evaluation explores an industrial-size configuration space. The results show that lazy product discovery has significant performance benefits compared to standard product discovery, which in contrast to our method requires all fragments to be composed to analyze the feature model. Furthermore, the method succeeds when more efficient, heuristics-based engines fail to find a valid configuration.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; *Feature interaction*; *Abstraction, modeling and modularity*; *Software libraries and repositories*; *Software creation and management*;

KEYWORDS

Software Product Lines, Configurable Software, Variability Modeling, Feature Models, Composition, Linux Distribution

ACM Reference Format:
Michael Lienhardt, Ferruccio Damiani, Einar Broch Johnsen, and Jacopo Mauro. 2020. Lazy Product Discovery in Huge Configuration Spaces. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10…

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380372

1 INTRODUCTION

Highly-configurable software systems can have thousands of interdependent configuration options across different subsystems. In the resulting configuration space, different software variants can be obtained by selecting among these configuration options. The interdependencies between options are typically caused by interaction in the resulting software system. Constructing a well-functioning software variant can be a complex and error-prone process [7].

Feature models [8] allow us to organize the configuration space and facilitate the construction of software variants by describing configuration options using interdependent *features* [32]: a feature is a name representing some functionality, a set of features is called a *configuration*, and each software variant is identified by a *valid configuration* (called a *product*, for short).

Highly-configurable software systems can consist of thousands of features and combine several subsystems [12, 13, 37, 56], each with different features. The construction and maintenance of feature models with thousands of features for such highly-configurable systems, can be simplified by representing large feature models as sets of smaller interdependent feature models [12, 49] which we call *fragments*. However, the analysis of such fragmented feature models usually requires the fragments to be composed, to enable the application of existing analysis techniques [9, 10, 43, 53, 58, 59]. To this aim, many approaches for composing feature models from fragments have been investigated [3, 6, 14, 16, 48, 52].

The analysis of fragmented feature models can be simplified if suitable abstractions can safely replace some of the feature model fragments in the analysis. This simplification can be realized by means of *feature-model interfaces* [51]. A feature-model interface is a feature model that hides some of the features and dependencies of another feature model (thus, interfaces are closely related to *feature-model slicing* [4]). An interface can be used instead of a feature … del. For certain … duces results … any feature … t compatible

This paper addresses automated *product discovery* in large configuration spaces represented as sets of interdependent feature models. Product discovery (sometimes called product configuration) is a particular analysis for finding a product which includes a desired set of features [26]. We aim at automatically discovering a product that contains a given set of features from the feature model fragments,

Lienhardt et al. 2020

Gentoo: 670k features

Knowledge Compilation Beyond BDDs

Journal of Artificial Intelligence Research 17 (2002) 229-264 Submitted 12/01; published 9/02

A Knowledge Compilation Map

Adnan Darwiche DARWICHE@CS.UCLA.EDU
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095, USA

Pierre Marquis MARQUIS@CRIL.UNIV-ARTOIS.FR
Université d'Artois
F-62307, Lens Cedex, France

Abstract

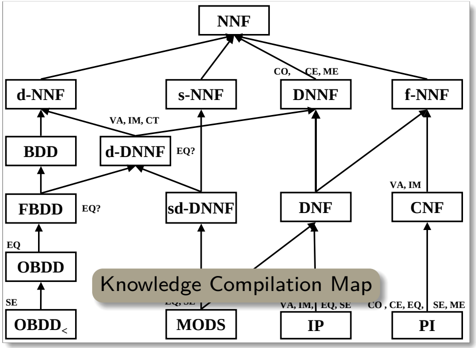
We propose a perspective on knowledge compilation which calls for analyzing different compilation approaches according to two key dimensions: the succinctness of the target compilation language, and the class of queries and transformations that the language supports in polytime. We then provide a knowledge compilation map, which analyzes a large number of existing target compilation languages according to their succinctness and their polytime transformations and queries. We argue that such analysis is necessary for placing new compilation approaches within the context of existing ones. We also go beyond classical, flat target compilation languages based on CNF and DNF, and consider a richer, nested class based on directed acyclic graphs (such as OBDDs), which we show to include a relatively large number of target compilation languages.

1. Introduction

Knowledge compilation has emerged recently as a key direction of research for dealing with the computational intractability of general propositional reasoning (Darwiche, 1999; Cadoli & Donini, 1997; Boufkhad, Grégoire, Marquis, Mazure, & Saïs, 1997; Khardon & Roth, 1997; Selman & Kautz, 1996; Schrag, 1996; Marquis, 1995; del Val, 1994; Dechter & Rish, 1994; Reiter & de Kleer, 1987). According to this direction, a propositional theory is compiled off-line into a target language, which is then used on-line to answer a large number of queries in polytime. The key motivation behind knowledge compilation is to push as much of the computational overhead into the off-line phase, which is amortized over all on-line queries. But knowledge compilation can serve other important purposes as well. For example, target compilation languages and their associated algorithms can be very simple, allowing one to develop on-line reasoning systems for simple software and hardware platforms. Moreover, the simplicity of algorithms that operate on compiled languages help in streamlining the effort of algorithmic design into a single task: that of generating the smallest compiled representations possible, as that turns out to be the main computational bottleneck in compilation approaches.

There are three key aspects of any knowledge compilation approach: the succinctness of the target language into which the propositional theory is compiled; the class of queries that can be answered in polytime based on the compiled representation; and the class of transformations that can be applied to the representation in polytime. The AI literature has thus far focused mostly on target compilation languages which are variations on DNF and CNF formulas, such as Horn theories and prime implicates. Moreover, it has focused mostly on clausal entailment queries, with very little discussion of tractable transformations on compiled theories.

Darwiche&Marquis 2002



Knowledge Compilation Map

What is the Goal of this Challenge?

Make Aware of Scalability Problems

Avoid Redundant Effort (cf. Publication Bias)

Foster Exchange on Solutions

Join Forces

Promote Knowledge Compilation

Advance State-of-the-Art

What is the Scope of this Challenge?

Knowledge Compilation (e.g., BDDs)

Large-Scale Configuration Spaces (e.g., Linux)

Partial Models and Older (i.e., Smaller) Versions

Incremental Knowledge Compilation (for Evolution)

Under- or Over-Approximations

(Un)successful Attempts

Knowledge Compilation Challenge for Variability