

[Herrenkrugsteg Magdeburg]

Advanced Topics in Feature-Model Analysis

Thesis Topics and Software Projects

August 10, 2026

Elias Kuiter[1]
University of Magdeburg[1]

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Advanced Topics in Feature-Model Analysis

1. Thesis Topics

Generalized Feature-Model Experiments for System Software (B/M)

CNF Transformation of Feature Models: A White-Box Analysis (B/M)

CNF Transformation of Feature Models: Miscellaneous Topics (B/M)

2. Software Projects

torte – Fully Automated Feature-Model Experiments in Bash (P)

clausy – Robust and Efficient CNF Transformation in Rust (P)

variED-NG – Collaborative Feature Modeling in TypeScript (P)

1. Introduction



Contact me:
kuiter@ovgu.de

About Me ...

- **until 2020**: M.Sc. Computer Science in Magdeburg
- **since 2021**: PhD student in Magdeburg supervised by Gunter Saake (Magdeburg) and Thomas Thüm (Braunschweig)

Research Interests

- generally: software variability, quality, and evolution
- specifically: feature-model and product-line analysis
- i.e., software engineering ∩ automated reasoning

You ...

- are interested in working on cutting-edge research tools and topics
- successfully completed our product-line course
- want to write your thesis in English or German

B: Bachelor thesis — **M**: Master thesis — **P**: software project

2. Background

Modeling Features and their Dependencies

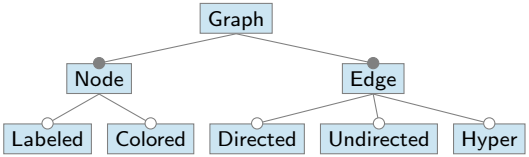
Feature Models

- tree models **features**
- cross-tree **constraints** model dependencies
- solver-based **analyses** on the configuration space

The Linux Kernel

- $\approx$ 20,000 features
- $> 10^{1,700}$ products
- 114 dead features [2013]
- 151 reverse dependency bugs [2019]

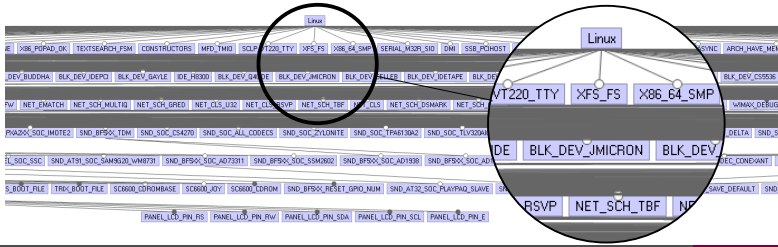
. . . and many other systems!



$\neg(\mathit{Directed} \wedge \mathit{Undirected})$

$\mathit{Hyper} \rightarrow \mathit{Undirected}$

$\mathit{Directed} \nleftrightarrow (\mathit{Undirected} \wedge \mathit{Hyper})$



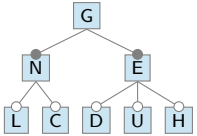
Analyzing Feature Models with SAT and #SAT Solvers

KConfig Specifications

```
config D
  bool "Directed"
  depends on !U
config U
  bool "Undirected"
  depends on !D
config H
  bool "Hyper"
  select U
```

→

A Feature Model *FM*



$\neg(D \wedge U)$
$H \rightarrow U$
$D \not\Leftrightarrow (U \wedge H)$

→

As a Formula $\Phi(FM)$

$$
\begin{aligned}
& G \\
\wedge\ & (N \leftrightarrow G) \wedge (E \leftrightarrow G) \\
\wedge\ & ((L \vee C) \rightarrow N) \\
\wedge\ & ((D \vee U \vee H) \rightarrow E) \\
\wedge\ & \neg(D \wedge U) \wedge (H \rightarrow U) \\
\wedge\ & (D \not\Leftrightarrow (U \wedge H))
\end{aligned}
$$

$\downarrow \Theta$

As a CNF $\Theta(\Phi(FM))$

$$
\begin{aligned}
&\{\{G\}, \{\neg N, G\}, \{N, \neg G\}, \\
&\ \{\neg E, G\}, \{E, \neg G\}, \{\neg L, N\}, \\
&\ \{\neg C, N\}, \{\neg D, E\}, \{\neg U, E\}, \\
&\ \{\neg H, E\}, \{\neg D, \neg U\}, \{\neg H, U\}, \\
&\ \{\{D, U\}, \{D, H\}, \{\neg D, \neg U, \neg H\}\}\}
\end{aligned}
$$

←

Core Feature *F*?

$SAT(\Theta(\Phi(FM)) \wedge \neg F)$

←

Core Features

$\{G, N, E\}$

←

Products in *FM*?

$\#SAT(\Theta(\Phi(FM)))$

←

Feature Model Cardinality

8

3. Thesis Topics

Generalized Feature-Model Experiments for System Software (B/M)

Problem

- **torte** is a tool to perform feature-model experiments
- **ASE'22** (evaluation of CNF transformations) is not yet integrated and reproduced within torte [Kuiter et al. 2022]
- **TOSEM'25** and **ICSE'26** are currently only evaluated on Linux, not other system software [Kuiter et al. 2025, 2026]

⇒ this limits the generalizability and reproducibility of these studies' results

Goals

- reproduce **ASE'22** evaluation
- extend **TOSEM'25** and **ICSE'26** evaluations
- review system software and add it to torte
- improve code reuse across experiments in torte

Requirements

- comfortable with Bash programming (torte)
- proficient in working with a multitude of tools
- evaluation design, reading literature

CNF Transformation of Feature Models: A White-Box Analysis (B/M)

Problem

- **CNF transformation** tools have a measurable impact on solver performance [Kuiter et al. 2022]
- there are many shades of CNF transformations, only few of which have been evaluated on a large corpus of models (**black-box analysis**)
- a **white-box analysis** is missing, which considers CNF parametrization and solver internals
- thus, we may better understand the influence of CNF and **choose good parameters**



Goals

- extend our Rust tool `clausy` with a configurable CNF transformation (i.e., Tseitin threshold)
- instrument the C SAT solver `kissat` to monitor solver internals (i.e., pre-/inprocessing)
- evaluate performance of `clausy` + `kissat` for many parametrizations on a large corpus
- evaluate the minimum Tseitin threshold as a measure for feature-model complexity

Requirements

- considerable experience with Rust and C
- evaluation design, reading literature

CNF Transformation of Feature Models: Miscellaneous Topics (B/M)

On interest, we can develop these into more concrete topics.

Problems	Goals
are the differences between CNF transformations due to **algorithms** or **implementations**?	**cross-evaluate** influence of CNF transformation algorithms implemented in different tools
how do **equivalent phrasings** of a CNF formula differ in solving performance? [BH:POS19, L:APAL24]	evaluate the influence of **CNF scrambling** on solver performance, possibly establish a canonical CNF

Topic Sketches

On interest, we can develop these into concrete topics.

- extract solution-space models from KConfig, which encode program variants instead of configurations
- evaluate the influence of hybrid CNF transformations on feature-model differencing
- evaluate connection of CNF transformation with phase transition, community structure, and backdoors
- assess relevance of tristate vs. boolean feature-model encoding in the Linux kernel
- create a “genealogy” of KConfig implementations and it’s clone-and-own history
- symbolic execution of the Linux kernel configurator to improve the accuracy of feature model extraction
- reverse engineering feature models from command-line interfaces
- analyze what is needed to create proper UVL feature models for Linux, and lift some remaining limitations
- develop a specialized solver on Linux based on TiniSAT or BlackholeSAT

4. Software Projects

torte – Fully Automated Feature-Model Experiments in Bash (P)

What is torte? [/ekuiter/torte]

- a declarative workbench for **reproducible** feature-model analysis experiments
- can extract, transform, and analyze feature models in a **fully automated** fashion
- draft, execute distribute, and adapt experiments (without clone-and-own)

A Simple Experiment: Counting BusyBox

```
experiment-subjects() {
    add-busybox-kconfig-history --from 1_36_0 --to 1_36_1
}
experiment-stages() {
    clone-systems
    extract-kconfig-models
    transform-models-into-dimacs
    solve-model-count --timeout 10
}
```

Goal

fix problems and implement new features from roadmap (issue #1)
⇒ enabling new use cases for torte

Requirements

- experience with Bash programming
- some experience with Docker
- willing to write clean code in Bash :-)



clausy – Robust and Efficient CNF Transformation in Rust (P)

What is clausy? [/ekuiter/clausy]

- transforms feature-model formulas into **conjunctive normal form (CNF)** for subsequent analysis
- improves comparability of transformation algorithms, competes with state-of-the-art tools
- implements algorithms for advanced use cases

Some Examples: Clausify Feature-Model Formulas

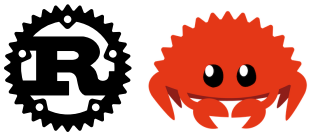
```
# simplify a given CNF
bin/clausy model.dimacs

# read from standard input and count solutions
cat model.uvl | bin/clausy --uvl to_cnf_tseitin count

# prove model equivalence
! bin/clausy a.model b.model '+(*(-1 2) *(1 -2))'
      to_cnf_tseitin satisfy &>/dev/null
```

Goal

fix problems and implement new features from roadmap (issue #1)
⇒ enabling new use cases for clausy

Requirements

- programming experience in Rust
- interested in algorithm engineering



variED-NG – Collaborative Feature Modeling in TypeScript (P)

What is variED? [/ekuiter/variED]

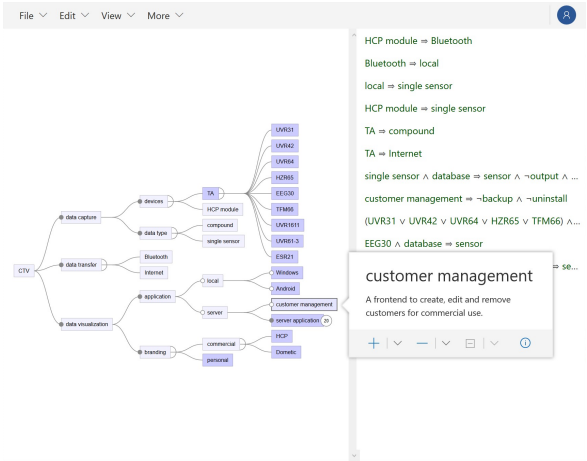
- a research prototype for viewing and editing feature models in a **collaborative, real-time** fashion
- vision: “Google Docs” for feature modeling
- easily share models, teach modeling, . . .

Goal

- prototype is complex and currently defunct
- continue work on simplified fork variED-NG
- fix and improve the editor

Requirements

- experience with TypeScript + React/Redux
- interested in modern web development



Interested?



Contact me: kuiter@ovgu.de

/ekuiter/

/ekuiter/

5. Archived Topics

These topics have already been assigned to or completed by another student.

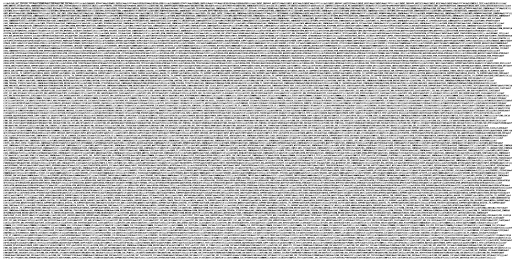
They can give you an impression of what we have done in the past.

Some of these topics might also be suited for follow-up work.

How Accurate Are Feature Models Extracted From KConfig Specifications? (B/M)

Problem

- KConfig extractors are used to extract feature models from KConfig specifications
- it is currently mostly unknown how accurately extractors construct configurations
- preliminary results are inconclusive [OGB+:TR19]



Goals

- evaluate construct validity of KConfig extractors
- is a satisfiable assignment really a valid KConfig configuration, and vice versa?
- is there a loss when doing a roundtrip transformation (to KConfig and back + vice versa)?
- how does KConfig behave when faced ...
  - ... with a partial configuration?
  - ... with an invalid configuration?
- how are non-Boolean features handled?
- does `y`/`m` make a difference for `tristate`?

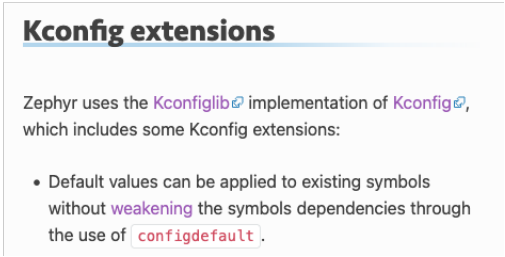
Requirements

- working with C- and Bash-based KConfig tooling
- evaluation design, reading literature

Extracting Feature Models From Extended KConfig Specifications (B/M)

Problem

- currently, feature models cannot be extracted for every project that uses `KConfig` specifications
- i.e., some projects use the **Python-based parser** `KConfigLib` instead of a C implementation
- e.g., the feature model of the operating system **Zephyr** can currently not be analyzed



Kconfig extensions

Zephyr uses the Kconfiglib implementation of Kconfig, which includes some Kconfig extensions:

- Default values can be applied to existing symbols without weakening the symbols dependencies through the use of `configdefault`.

Goals

- review usage of `KConfigLib` in Zephyr and other projects in the system software domain
- investigate **differences between** `KConfigLib` **and the C parser** used in Linux
- conceptualize and implement a solution for **extracting feature models from** `KConfigLib` (e.g., transform into Linux-compatible `KConfig`, or adapt `KClause` to `KConfigLib` input)
- evaluate feasibility and performance + accuracy of extracted models by comparing with `KClause`

Requirements

- reading literature and `KConfig` code
- experience with Python and C
- wrangle with subtle `KConfig` syntax/semantics

The Impact of Clone-and-Own on Parsing KConfig Specifications (B/M)

Problem

- the C `KConfig` parser of Linux **evolves over time**, slowly adding new syntax and semantics and deprecating or fixing old constructs
- moreover, other projects tend to **clone-and-own** the parser from Linux, leading to a variety of implementations
- it is unknown which (distinct) `KConfig` implementations exist and how they came to be
- it is also unclear to which degree these implementations are **compatible** with each other, and whether using an incorrect parser can threaten the validity of research evaluations

Goals

- trace the **genealogy of KConfig parsers** (who forked it when/why? with which adaptations?)
- evaluate **backwards compatibility** of the parser in a given project (can a new parser extract an old feature model? is the result accurate?)
- evaluate **cross-compatibility** of parsers

Requirements

- reading literature, code, and commit histories
- experience with C to compare implementations
- wrangle with subtle `KConfig` syntax/semantics



Minimizing CNFs to Isolate Solver Bugs (B)

Problem

- CNFs of real-world feature models sometimes uncover **bugs** even in production-grade (#)SAT and SMT solvers
- e.g., in countAntom, sharpSAT/dSharp, Z3, clausy, FeatJAR
- during development and maintenance of such solvers, reducing problematic CNFs to a **minimum non-working example** can facilitate finding the causes of bugs, reporting them, and preventing future regressions
- however, this process is currently a manual task and time-consuming



Goals

- identify fault oracles (e.g., solver crashes), review reduction strategies (e.g., removing clauses one-by-one, bisection, backtracking to avoid a local minimum)
- implement a (semi-)automatic tool that repeatedly reduces clauses and literals in a faulty CNF until it is minimal
- evaluate performance and compare with global minimum (e.g., obtained manually)

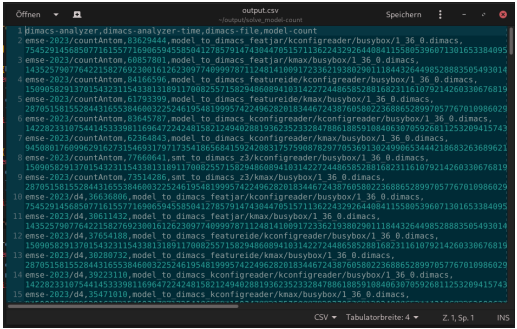
Requirements

- algorithm design, reading literature
- potential challenges: generative effects, local minima, choosing bugs to evaluate

A Dashboard for Evolving Variability in Open-Source Systems (P)

Problem

- torte fully automates feature-model analysis
- can be used to analyze latest Linux kernel
- **but**: no user-friendly frontend exists yet



Goal

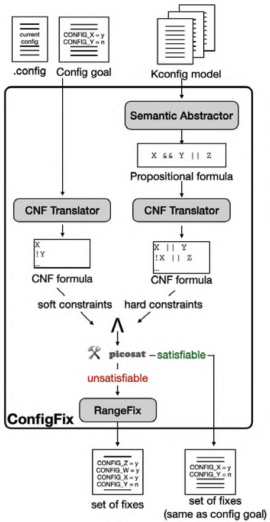
- develop a web frontend for torte
- find appropriate visualizations

⇒ quick visualization of current state of variability
⇒ extension of previous research evaluation

Requirements

- experience with frontend development (e.g., HTML/CSS, React/Vue/Dash, . . .)
- backend experience not necessarily needed (e.g., assume a static CSV file over AJAX)

Evaluating an Extractor for KConfig-Based Feature Models (B)



Problem

- a new feature-model extractor, ConfigFix, has recently been published
- however, it has not been compared to existing extractors yet

Goal

- implement extraction of feature models with ConfigFix in torte
- evaluate efficiency and accuracy on a large corpus on feature models

Requirements

- good in bash programming
- work with ConfigFix, which is implemented in C

Extracting Feature Hierarchies for KConfig-Based Feature Models (B)

Problem

- feature-model extractors for KConfig mostly ignore the **feature hierarchy**
- tooling for extracting hierarchies is now defunct, identification of **feature parents** in Kconfig is yet under-researched

```
1  namespace Root
2
3  features
4      Root
5          optional
6              UNWINDER_ORC
7              UNWINDER_FRAME_POINTER
8              UNWINDER_GUESS
9              X86_64
10             IO_DELAY_0X80
11             IO_DELAY_0XED
12             IO_DELAY_UDELAY
13             IO_DELAY_NONE
14             BRANCH_PROFILE_NONE
15             PROFILE_ANNOTATED_BRANCHES
```

Goal

- extract a feature hierarchy from KConfig specifications + evaluate accuracy
- and/or: reverse-engineer hierarchy from formula + compare with KConfig hierarchy

Requirements

- adjusting KConfig parser written in C
- adjust or implement a tool for reverse-engineering
- c.f. Yaman 2023, Yaman et al. 2024

Feature-Model Analysis with SAT Solvers: A Journey Through Time (B)

Problem

- feature models grow more complex over time
- automated reasoning tools (e.g., SAT solvers) get more efficient over time
- **but**: which development is faster? can SAT solvers actually keep up?



[Photo: Laurent Simon]

Goal

- collect best SAT solvers of the last 20 years
- collect feature models from the last 20 years
- run selected feature-model analyses with solver from year **X** on model of year **X**
- evaluate evolution of SAT solving performance (cf. Moore's law)
- see time leap challenge

Requirements

- methodology design, reading literature
- challenges: data availability and formats

Previously Supervised Theses [full list available here]

- **Efficiency of Projected Model Counting for Feature-Model Analysis** (B)

  Excerpts of feature models (so-called slices) allow for modularization and separation of concerns for software variability. However, calculating slices is complex. For some applications, it is sufficient to know the size (i.e., the number of valid products) of a slice. Projected Model Counting (PMC) is a new solver class that can directly determine the size of a slice without calculating it. The aim of this work is to evaluate the applicability of PMC for feature model analysis. Tasks include to research PMCs and their functionality, apply PMC to feature models of various sizes, and compare with slicing and subsequent #SAT calls.

- **Evaluating the Efficiency of Hybrid CNF Transformations for Feature-Model Formulas** (M)

  For automated feature-model analysis, SAT and #SAT solvers are used. These expect the input formula to be in CNF (conjunctive normal form). There are various methods to convert a formula into CNF. The aim of this work is to continue our existing research and conduct a detailed comparison of the distributive transformation with the Tseitin and Plaisted-Greenbaum transformations, thereby drawing conclusions about when to prefer which CNF transformation. Some aspects to consider are parameters for hybrid transformation, preprocessing in SAT solvers, and multiple implementations for external validity.

- ...