Feature Model Synthesis

Steven She

Generative Software Development Lab
What is Variability in Software?

Variability in a software system is its ability for a system to adapt and customize for a particular context.

—van Gurp et al., 2001
Why Variability Modeling?

Large software systems contain variability scattered over *documentation*, *design* and *implementation*.

e.g., FreeBSD

®
Documentation

STACK enables the stack(9) facility… stack(9) will also be compiled in automatically if DDB(4) is compiled into the kernel.

Source Code

```c
#ifdef DDB
#ifdef KDB
#error KDB must be enabled for DDB to work!
#endif
#endif
```
### Configuring FreeBSD

- FreeBSD is configured by setting values to config options.
- Features and dependencies are scattered over documentation and code.
- Difficult to get an **overview of the variability**.

```plaintext
options SCHED_ULE  #ULE scheduler
options PREEMPTION #Enable kernel thread preemption
options INET      #InterNETworking
options INET6     #IPv6 communications protocols
```
Variability Models

Explicit model of a system's variability.

Benefits include **Graphical Configurators** and **Automated Analysis**.
Feature Models

Feature models describe the common and variable characteristics of products in a product line.

- First introduced by Kang et al.
- Describe a set of legal configurations.
Feature Model Syntax

powersave \land \text{acpi} \rightarrow \text{cpu\_hotplug}
Configuration Semantics

Feature models describe a set of **legal configurations**.

Represented as a propositional formula, $\varphi$.

- Satisfying assignments are the legal configurations.
What is Feature Model Synthesis?

Feature model synthesis is the construction and design of a feature model given a set of features and legal combinations of features.
Applicable Synthesis Scenarios

1. Synthesis From Product Configurations
2. Tool-Assisted Reverse Engineering from Code
3. Feature Model Merge Operations
From Product Configurations

- Input consists of variants describing a product line.
- e.g., model variants, products developed by cloning code.
- Variants are compared and Variation Points (VPs) identified.
- VPs and VP configurations used as input for synthesis.
Tool-Assisted Reverse Engineering from Code

- Input consists of source code containing variability.
- e.g., FreeBSD with #ifdef annotated code.
- Static analysis of #ifdef statements identifies code fragments as VPs and dependencies between VPs.
Feature Model Operations

- Input consists of feature models.
- Feature models translated to a prop. formula by configuration semantics.
- Operation applied to formula then used as input to synthesis.
Requirements for FM Synthesis

Input
Support input as either Configurations or Dependencies.

Sound and Complete
Derive an exact feature model describing the input.

Scalable
Support 10 to 1000's of features (e.g., Linux, FreeBSD).

Hierarchy Selection
Use user input or heuristics to select a distinct feature hierarchy.
We efficiently synthesize large scale feature models with algorithms that use SAT-based reasoning on propositional formulas and that suggest a feature hierarchy with textual similarity heuristics.
Contributions

1. Feature Graph Extraction


Contributions (cont.)

2. **Feature Tree Synthesis**

3. **Kconfig & the Linux Variability Model**
How the Algorithms Relate
Feature Graph Extraction
Requirements for FM Synthesis

Input
Support input as either Configurations or Dependencies.

**Sound and Complete**
Derive an exact feature model describing the input.

Scalable
Support 10 to 1000's of features (e.g., Linux, FreeBSD).

Hierarchy Selection
Use user input or heuristics to select a distinct feature hierarchy.
Soundness and Completeness

\{ OS, staging \},
\{ OS, staging, net \},
\{ OS, staging, net, dst \}

Less configs (sound)        More configs (complete)        Arbitrary

- OS
  - net
  - staging
  - dst

- OS
  - net
  - dst
  - staging

- OS
  - net
  - staging
  - dst
Sound and Complete Synthesis

\{ \text{OS, staging} \},
\{ \text{OS, staging, net} \},
\{ \text{OS, staging, net, dst} \}

Complete FD

Sound and Complete FD
Maximal Feature Diagram

\[
\{ \text{OS, staging } \},
\{ \text{OS, staging, net} \},
\{ \text{OS, staging, net, dst} \}
\]

Non-maximal FD

\[
\text{OS} \rightarrow \text{net} \quad \text{dst} \quad \text{maximal FD}
\]

Maximal FD

\[
\text{OS} \rightarrow \text{net} \quad \text{staging}
\]
Same Configs, Diff. Hierarchies

\{ \{ \text{OS, staging} \}, \{ \text{OS, staging, net} \}, \{ \text{OS, staging, net, dst} \} \}
Feature Graph

{ OS, staging },
{ OS, staging, net },
{ OS, staging, net, dst }

- Encapsulates all feature diagrams that are complete.
- DAG as hierarchy, and overlapping feature groups.
Requirements for FM Synthesis

**Input**
Support input as either Configurations or Dependencies.

**Sound and Complete**
Derive an exact feature model describing the input.

**Scalable**
Support 10 to 1000's of features (e.g., Linux, FreeBSD).

**Hierarchy Selection**
Use user input or heuristics to select a distinct feature hierarchy.
Input as Configuration

\[ \{ \text{OS, staging } \}, \{ \text{OS, staging, net} \}, \{ \text{OS, staging, net, dst} \} \]

\[ \mapsto \quad (\text{OS} \land \text{staging} \land \neg \text{net} \land \neg \text{dst}) \lor \\
(\text{OS} \land \text{staging} \land \text{net} \land \neg \text{dst}) \lor \\
(\text{OS} \land \text{staging} \land \text{net} \land \text{dst}) \]

Configurations represented as DNF formula.

Input as Dependencies

\[ \begin{align*}
\text{staging} \lor \text{net} & \rightarrow \text{OS} \\
\text{dst} & \rightarrow \text{net} \\
\text{OS} & \rightarrow \text{staging}
\end{align*} \]

\[ \mapsto \quad (\neg \text{staging} \lor \text{OS}) \land (\neg \text{net} \lor \text{OS}) \land \\
(\neg \text{dst} \lor \text{net}) \land \\
(\neg \text{OS} \lor \text{staging}) \]

Dependencies represented as a CNF Formula.
Feature Graph Extraction (Fge)

\[ \text{FGE}(\varphi_{\text{CNF, DNF}}) \]

- **Fully automatic** algorithm for extracting feature graphs.
- Algorithm uses a SAT solver.
DAG Hierarchy Recovery

\[ \text{DAG}(\varphi) \mapsto \]

- Given a formula, \( \varphi \), build an Implication Graph.
- Each edge \((u, v)\) is an implication such that \( \varphi \land u \rightarrow v \)
- Describes all possible hierarchies as a DAG.
Group and CTC Recovery

Mutex Groups [0..1]

Find **maximal cliques** in the **mutex graph** where an edge \((u, v)\) exists if \(\varphi \land u \rightarrow \neg v\).

Or Groups [1..n]

Given a parent \(p\), find **prime implicates** of \(\varphi \land p\) with the form
\[f_1 \lor f_2 \lor \ldots \lor f_k\].

Xor Groups [1..1]

Groups that are both Mutex and Or groups.
Requirements for FM Synthesis

Input
Support input as either Configurations or Dependencies.

Sound and Complete
Derive an exact feature model describing the input.

Scalable
Support 10 to 1000's of features (e.g., Linux, FreeBSD).

Hierarchy Selection
Use user input or heuristics to select a distinct feature hierarchy.
Experimental Evaluation

Purpose
Evaluate performance of our algorithms by comparing to other algorithms that build a feature graph.

Dataset
Input representative of synthesis scenarios.
Derive input from FMs in a FM repository, generated FMs, and the Linux variability model.

Measure
Time needed to compute each part of a feature graph.
Quality does not need to evaluated.
Feature graph encapsulates all complete feature diagrams.
# Evaluation Algorithms

## Fge-CNF Evaluation

<table>
<thead>
<tr>
<th>Fge-CNF</th>
<th>BDD-Based [Czarnecki and Wąsowski]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>Dependencies</td>
</tr>
<tr>
<td><strong>Technique</strong></td>
<td>SAT Solver</td>
</tr>
</tbody>
</table>

## Fge-DNF Evaluation

<table>
<thead>
<tr>
<th>Fge-DNF</th>
<th>FCA-Based [Ryssel et al.]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>Configurations</td>
</tr>
<tr>
<td><strong>Technique</strong></td>
<td>SAT</td>
</tr>
</tbody>
</table>
Dataset Characteristics

**SPLLOT Model Repository**
- Largest, public repository of feature models.
- 267 FMs gathered from academic papers, experience reports, by volunteers.

**Generated Models**
- 20 generated FMs with difficult cross-tree constraints.

**Linux Variability Model**
- 5426 features.

![Number of features vs Models (ordered by model size)]
Experiment Setup

Null Hypothesis

For each component of Fge, (i.e., implication graph, mutex graph, OR-groups) there is no difference in the mean computation times for Fge-CNF and Fge-BDD.
Fge-CNF vs. Fge-BDD Results

### SPLOT Dataset

<table>
<thead>
<tr>
<th>Component</th>
<th>Mean Difference (ms)</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implications</td>
<td>-16</td>
<td>0.63</td>
</tr>
<tr>
<td>Mutual Exclusions</td>
<td>-20</td>
<td>0.38</td>
</tr>
<tr>
<td>Or Groups</td>
<td>-10,854</td>
<td>1.13 x 10^{-9}</td>
</tr>
</tbody>
</table>

Fge-CNF is **significantly faster** than the BDD-based algorithm for computing OR-Groups on the SPLOT dataset.

**Linux**

Fge-CNF completed in 7 hours. The BDD-based algorithm **ran out of memory**.

**Generated Dataset**

Fge-CNF completed 12 models. The BDD-based algorithm **timed out on all models**.
Fge-DNF vs. FCA-Based Results

SPLOT Dataset

<table>
<thead>
<tr>
<th>Component</th>
<th>Mean Difference (ms)</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implications</td>
<td>320</td>
<td>0.0059</td>
</tr>
<tr>
<td>Mutual Exclusions</td>
<td>166</td>
<td>0.0012</td>
</tr>
<tr>
<td>Or Groups</td>
<td>-3,904</td>
<td>0.1214</td>
</tr>
</tbody>
</table>

Performance of Fge-DNF is similar to that of the FCA-based algorithm, except for 5 models where Fge-DNF was significantly faster.
Models had a large number of sibling features at the root. Large search space for groups for FCA-based algorithm.
Feature Graph Extraction

Summary

\[ \text{FGE}(\varphi_{\text{CNF, DNF}}) \]

- Fully automated algorithm.
- Feature graph describes **all possible feature diagrams** that are complete for a given input.
Feature Tree Synthesis
How the Algorithms Relate

Feature Graph Extraction

Feature Graph

Feature Tree Synthesis

Feature Model

User Input, Heuristics

Configurations or Dependencies

Features
Requirements for FM Synthesis

Input
Support input as either Configurations or Dependencies.

Sound and Complete
Derive an exact feature model describing the input.

Scalable
Support 10 to 1000's of features (e.g., Linux, FreeBSD).

Hierarchy Selection
Use user input or heuristics to select a distinct feature hierarchy.
Selecting a Hierarchy

How do we select a hierarchy out of all possible hierarchies?

Feature Tree Synthesis combines logical constraints with a textual similarity heuristic.
Two Lists of Potential Parents

Selected: `cpu_hotplug`

- 1. powersave
- 2. acpi
- 3. acpi_system
- 4. cpu_freq
- 5. pm

**Ranked Implied Features (RIFs)**

The implied features ranked by similarity to the selected feature.

1. `cpu_freq`
2. powersave
3. performance
4. acpi
5. acpi_system

**Ranked All Features (RAFs)**

All features in the input ranked by similarity. Handles incomplete input.
## Feature Similarity Measure Example

Selecting a parent for:  
**bluetooth** \( \rightarrow \) a **network driver**.

<table>
<thead>
<tr>
<th>os_kernel</th>
<th>Operating systems</th>
<th>1. networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduler</td>
<td>I/O scheduling</td>
<td>2. ethernet</td>
</tr>
<tr>
<td>networking</td>
<td>Network drivers</td>
<td>3. os_kernel</td>
</tr>
<tr>
<td>ethernet</td>
<td>Type of local area</td>
<td>4. scheduler</td>
</tr>
<tr>
<td>networking</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ranked Implied Features

- Children features must logically imply parents in FMs.
- Combines logical implications with feature similarity measure.
- Further prioritize **directly implied features**.

Ranked All features

- Rely purely on features similarity in case of incomplete input.
Feature Similarity Measure

Given a selected feature $s$ and another feature $p$, define similarity measure $\delta(s, p)$:

1. Take **common words** in the descriptions of $p$ and $s$,
2. **Sum up the occurrence** of each shared word in $p$'s description,
3. weigh each word by its *Inverse Document Frequency* (IDF).

IDF used to discount common words in the domain e.g., choose, select, Linux.
Evaluation

Purpose
Validate that the lists *reduces the choices for a user to consider* when building the feature hierarchy *under complete and incomplete data*.

Complete Data
Input and reference models from Linux, and eCos.

Incomplete Data
Extracted features and dependencies from FreeBSD codebase. Manually created a reference model for part of FreeBSD.
Linux and eCos Variability Models

- Linux: almost 6000 features
- eCos: 1200 features

- Derive input based on language semantics.
- Use the variability models as the reference. i.e., the “correct” parent for each feature.
Evaluating RIFs

How many features have their reference parents in the Top 5 entries of our RIFs?

**All Features**

76% of features in Linux. 79% of features in eCos.
Prioritizing Direct Implications

![Graphs comparing features in top 5 results for Linux and eCos, showing RIFs with prioritizing order in black, without in gray.](image-url)
Evaluating RAFs

How many features need to be examined to find the reference parent for 75% of features using the RAFs?

at most 6% of all features in Linux.

at most 3% of all features in eCos.

at most 3% of all features in FreeBSD.
RAFs under Incomplete Input

Randomly removed 25%, 50%, and 75% of words from all descriptions.
Tree Synthesis Summary

\{ \{ \text{OS, staging} \}, \\
\{ \text{OS, staging, net} \}, \\
\{ \text{OS, staging, net, dst} \} \}\}

Present the potential parents of a feature to a user.

- Two lists that combines logical dependencies with a textual similarity measure.

Selected: `cpu_hotplug`

- `powersave`
- `acpi`
- `acpi_system`
- `cpu_freq`
- `pm`

Ranked Implied Features

CPU frequency scaling.

- `cpu_freq`
- `powersave`
- `performance`
- `acpi`
- `acpi_system`
- ...

Ranked All-Features
Kconfig Language and Models
The Kconfig Variability Modeling Language

Motivation
Variability models available to researchers were small and not used in industry or real world projects.

Contribution
Analyzed Kconfig—variability modeling in Linux kernel.
Reverse engineered semantics of the Kconfig language and analyzed models from 12 open source projects.
Largest variability models available to researchers. Used the Linux model for evaluating our tools.
Conclusions
Requirements for FM Synthesis

Input
Support input as either Configurations or Dependencies.

Sound and Complete
Derive an exact feature model describing the input.

Scalable
Support 10 to 1000's of features (e.g., Linux, FreeBSD).

Hierarchy Selection
Use user input or heuristics to select a distinct feature hierarchy.
Thesis Statement

We efficiently synthesize large scale feature models with algorithms that use SAT-based reasoning on propositional formulas and that suggest a feature hierarchy with textual similarity heuristics.
Contributions

1. Feature Model Synthesis Scenarios
2. Feature Graph Extraction
   - Builds a feature graph on DNF and CNF input with a SAT solver.
   - Feature graph is maximal and complete, with cross-tree constraints describes exactly the input.
3. Feature Tree Synthesis
   - Semi-automated technique for building the feature hierarchy using logical constraints and a textual similarity measure.
4. Kconfig and the Linux Variability Model
   - Largest variability models available to researchers.
Links: Publications, Scenarios, Feature Graph Extraction, Feature Tree Synthesis.
Extra Slides
Feature Similarity Measure (cont.)

Given a selected feature $s$ and another feature $p$:

$$\delta(p, s) = \sum_{w \in \{\text{desc}(p) \cap \text{desc}(s)\}} \text{count}(w, \text{desc}(p)) \times \text{idf}(w)$$

where $\text{idf}(w) = \log \frac{|\text{features}|}{|\{f \in \text{features} \mid \text{desc}(f) \text{ contains } w\}|}$
FreeBSD Variability Model

- Manually constructed a reference model for a subset of FreeBSD.

- First constructed an ontology, then traversed generalization and composition relations to create feature hierarchy.

- Resulting model had 192 features describing tracing, monitoring, and debugging features.

- Incomplete input (i.e., dependencies) was extracted by applying a fuzzy parser on documentation, and identifying dependencies in #ifdef code.