Feature Modeling and Sampling

Thesis Topics | Sebastian Krieter | June 29, 2022

Software Engineering
Programming Languages

universität ulm

About Me

- **Short CV**
  - **2015** Master in Computer Science (Magdeburg)
  - **2022** Defended PhD Thesis (Magdeburg)
  - **Since 2022** Researcher at Uni Ulm
- **Research interests**
  - Feature modeling and analysis
  - Configuration sampling and testing
- **Email**
  - sebastian.krieter@uni-ulm.de

About Elias

- **Short CV**
  - **2020** Master in Computer Science in Magdeburg
  - **Since 2021** PhD student in Magdeburg
- **Research interests**
  - Product-Line Analysis
  - Feature-Model Interfaces and Transformations
- **Email**
  - elias.kuiter@ovgu.de

Counting Slices with Projected Model Counters (M)

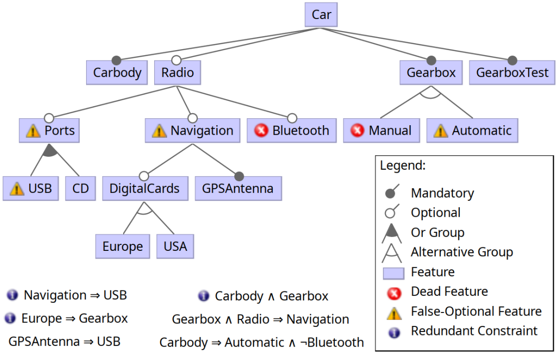
- Sometimes, we only want to count part of a feature model (= a slice)
- Currently, we have to compute the actual slice, then run a standard model counter (#SAT)
- Projected model counters (PMC) are new solvers that can directly count a slice
- **How does PMC compare to #SAT for feature-model slices?**

- **Goals:**
  1. Identify use cases where only the slice count is needed
  2. Evaluate and compare the performance of PMC and #SAT solvers on fragmented feature models

Cleaning Feature Models (B/P)

- Feature models may contains anomalies (e.g., dead features, false-optional features, redundant constraints…)
- Detecting them can be automatized
- Fixing them currently requires user decisions and manual effort
- **To which degree can this be automatized?**

- **Goals:**
  1. Compare and discuss suitable strategies for fixing (e.g., which redundant constraints to remove)
  2. Implement promising strategies in FeatureIDE



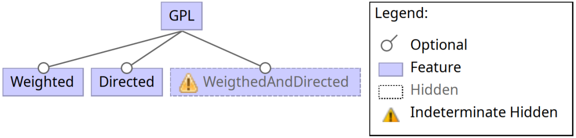
Efficient Analyses for Indeterminate Hidden Features (M)

- Hidden features cannot be configured directly

⇒ Whether a hidden feature is selected must **always** be determined by other features

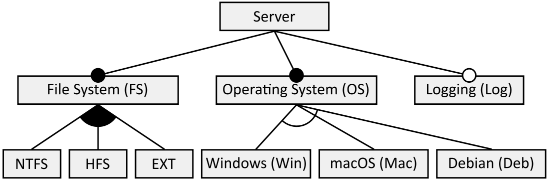
- Otherwise it is **indeterminate**
- Current analyses require much computational effort
- **Are there more efficient analyses for detecting indeterminate hidden features?**

- **Goals:**
  1. Improve the current analysis for finding indeterminate hidden features in FeatureIDE
  2. Evaluate the new analysis



Weighted ∧ Directed ⇒ WeigthedAndDirected

Sampling (Problem Space)



NTFS ↔ Windows (Win)   HFS ↔ macOS (Mac)

$\Rightarrow$

$\{Server, FS, OS, HFS, Mac\}$

$\{Server, FS, OS, NTFS, EXT, Win\}$

$\{Server, FS, OS, EXT, Deb, Log\}$

...

- Create a representative list of configurations (e.g., for testing)
  - Random
  - Coverage criteria
  - ...

Sampling (Solution Space)

Presence Conditions	Source File
Log	`//#if Logging`
Log	`public class Logger {`
Log	`  public void log(String message) {`
$(Log \wedge EXT)$	`    //#if EXT`
$(Log \wedge EXT)$	`    print("FS␣is␣EXT");`
$(Log \wedge EXT)$	`    //#endif`
$(Log \wedge Mac) \vee (Log \wedge Win)$	`    //#if macOS || Windows`
$(Log \wedge Mac) \vee (Log \wedge Win)$	`    print("OS␣is␣not␣Linux");`
$(Log \wedge Mac) \vee (Log \wedge Win)$	`    //#endif`
Log	`  }`
Log	`}`
Log	`//#endif`

⇒ Use presence conditions instead of features for sampling

Repairing Samples after Feature Model Evolution (M)

- A configuration sample is a representative list of valid configurations for a feature model
- Generating a sample can be computationally expensive
- After changes to the feature model (evolution), samples must be recomputed
- **Can samples be updated (repaired) instead?**

- **Goals:**
  1. Create a concept for efficiently updating a sample after feature model evolution
  2. Evaluate the proposed approach

Sampling with Evolving Presence Conditions (M)

- Samples can be based on presence conditions of implementation artifacts
- Presence conditions may also change when a product line evolves
- **How often and to what degree do presence conditions change on average?**

- **Goals:**
  1. Measure the rate at which presence conditions change during the history of multiple product lines
  2. Compare the results and try to find correlations and trends