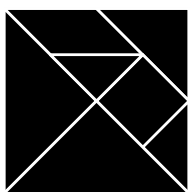
Reverse Engineering Feature Models.

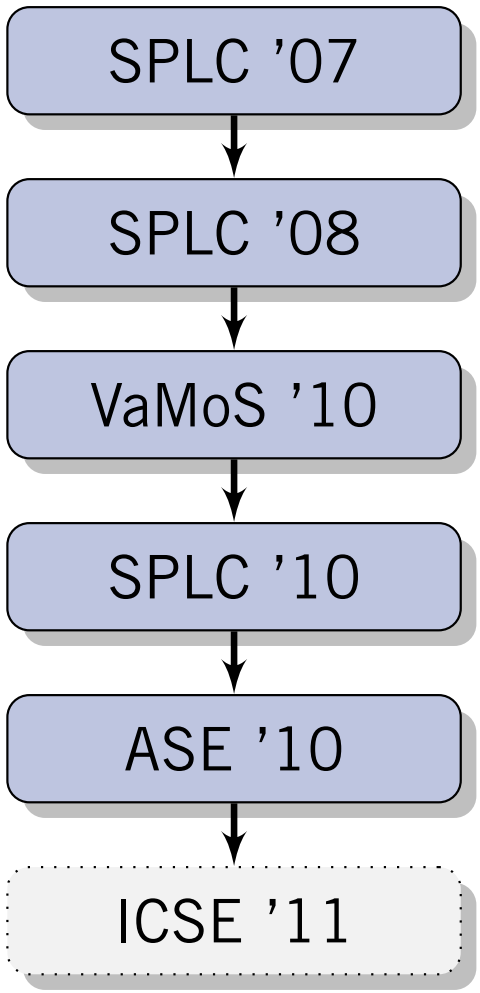


S. She, R. Lotufo, T. Berger, A. Wasowski, K. Czarnecki
Generative Software Development Lab
University of Waterloo

International Conference on Software Engineering,
Honolulu, Hawaii, USA.

Outline.

Timeline.



Propositional logic and FMs.

Sample spaces and FMs.

Initial study on the Linux FM.

Evolution of Linux feature models.

Linux and eCos feature models.

Rev. engineering Linux, eCos, and parts of the FreeBSD FMs.

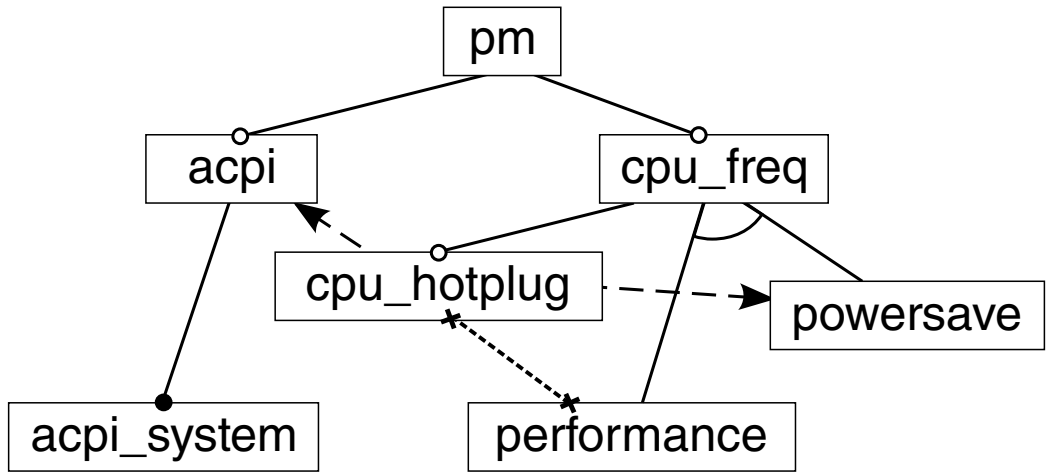
Motivation.

- Feature models are great for representing variability in software product lines (SPLs), e.g. Linux and eCos.
- Some SPLs don't have feature models, e.g. FreeBSD.
- Without a feature model, dependencies are hard to understand for users and developers.
- Solution: Reverse engineer a feature model from existing project artifacts.
- We present procedures to address this task.

Outline.

A Propositional Feature Model...

is a Tree + Cross-Tree Constraints.



$$\text{powersave} \wedge \text{acpi} \rightarrow \text{cpu\_hotplug}$$

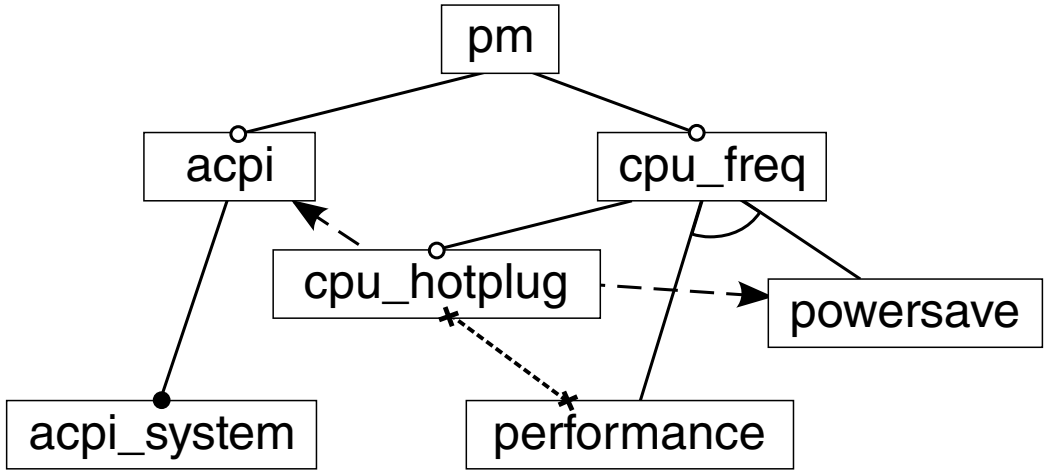
Configuration Semantics.

A feature model describes a set of legal configurations.

Legal configurations can be represented by a formula.

$$[\![FM]\!] \equiv \phi$$

Formula.



$$\text{powersave} \wedge \text{acpi} \rightarrow \text{cpu\_hotplug}$$

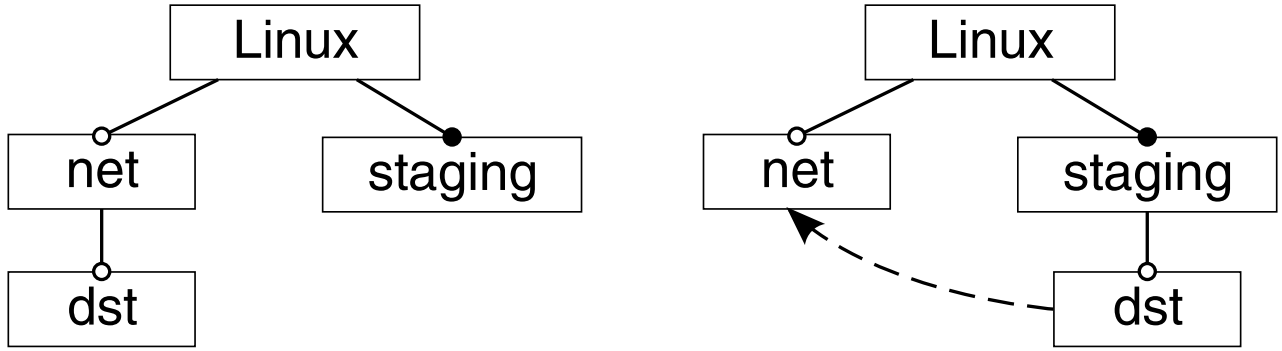
$$\begin{aligned}\phi = \quad & \text{child-parent impl.} \wedge \text{mandatory impl.} \\ \wedge \; & \text{implies edges} \wedge \text{excludes edges} \\ \wedge \; & \text{feature groups} \\ \wedge \; & \text{cross-tree constraints}\end{aligned}$$

Propositional Logic to Feature Models.

Given ϕ, there are **many** possible feature models that describe its legal configurations.

FM = tree + cross-tree constraints.

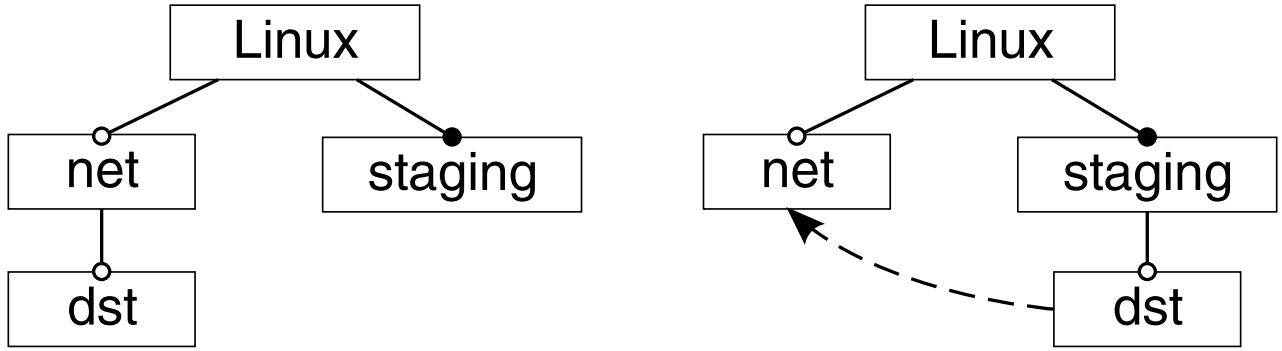
Ontological semantics.



dst is an experimental feature.

- Two FMs with the same configuration semantics, but different ontological semantics.
- We need a human to decide the placement of a feature.
- But, names and descriptions can be used to provide assistance.

Ontological semantics.



dst is an experimental feature.

- Two FMs with the same configuration semantics, but different ontological semantics.
- We need a human to decide the placement of a feature.
- But, names and descriptions can be used to provide assistance.

Outline.

Overview.

Key challenge.

- Building the feature hierarchy.
- Task—Select a parent for each feature.

Reverse-Engineering Procedures.

- Interactive—procedures provides assistance to the user for building hierarchy.
- Robust—we consider and evaluate on incomplete input.
- Mandatory features, feature groups, implies and excludes edges are automatically detected.

Overview.

Key challenge.

- Building the feature hierarchy.
- Task—Select a parent for each feature.

Reverse-Engineering Procedures.

- Interactive—procedures provides assistance to the user for building hierarchy.
- Robust—we consider and evaluate on incomplete input.
- Mandatory features, feature groups, implies and excludes edges are automatically detected.

Required Input.

$\mathcal{F}$

Feature Names

$\mathcal{D}$

Descriptions

ϕ

Dependencies

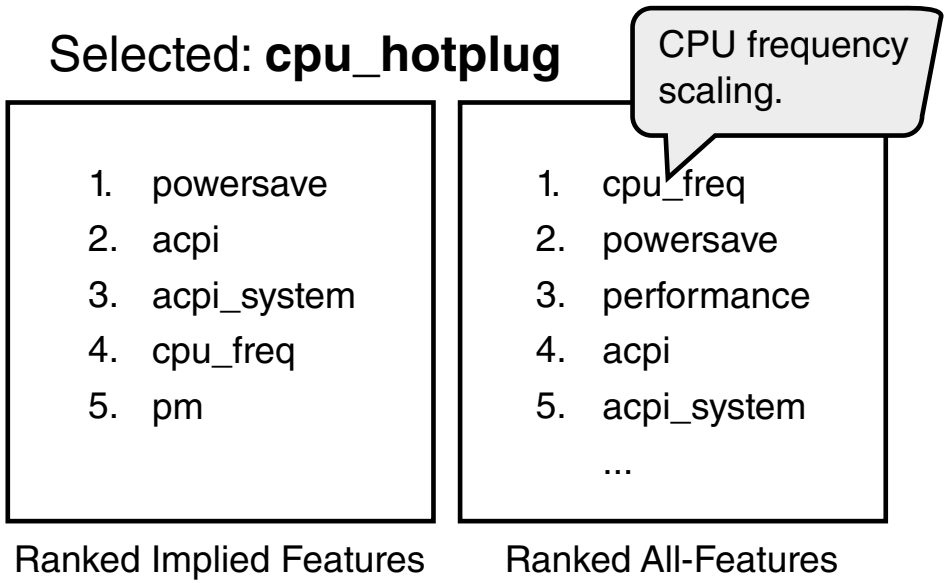
Names and Descriptions.

pm	Power management, CPU and ACPI options
acpi	Adv. Configuration and Power Interface support
acpi_system	Enable your system to shut down using ACPI
cpu_freq	CPU frequency scaling
cpu_hotplug	Allows turning CPU on and off
powersave	This CPU governor uses the lowest frequency
performance	This CPU governor uses the highest frequency

Dependencies.

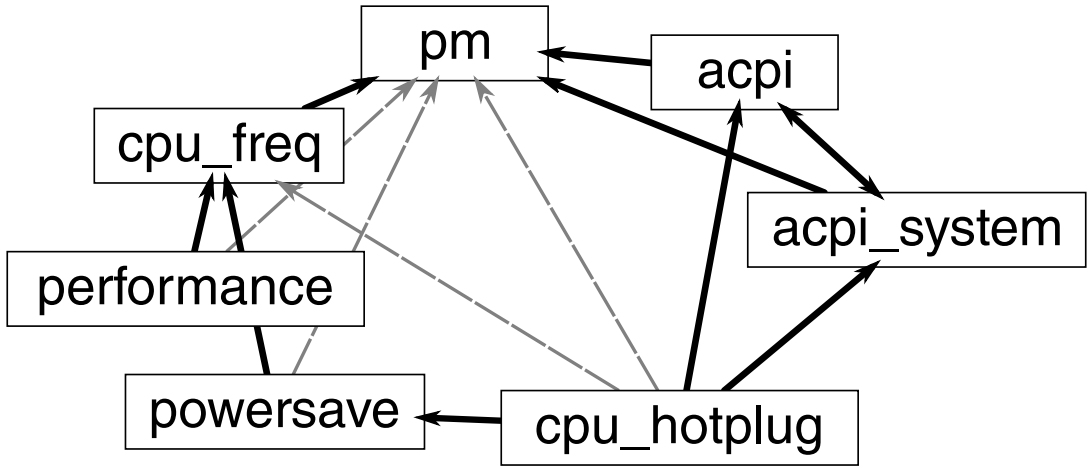
$$
\begin{aligned}
& (\text{acpi} \rightarrow \text{acpi\_system} \wedge \text{pm}) \\
\wedge\ & (\text{acpi\_system} \rightarrow \text{acpi}) \\
\wedge\ & (\text{cpu\_freq} \rightarrow \text{pm}) \\
\wedge\ & (\text{cpu\_freq} \rightarrow \text{powersave} \vee \text{performance}) \\
\wedge\ & (\text{cpu\_hotplug} \rightarrow \text{powersave}) \\
\wedge\ & (\text{cpu\_hotplug} \rightarrow \neg\text{performance}) \\
\wedge\ & (\text{cpu\_hotplug} \rightarrow \text{acpi} \wedge \text{cpu\_freq}) \\
\wedge\ & (\text{powersave} \rightarrow \neg\text{performance}) \\
\wedge\ & (\text{powersave} \rightarrow \text{cpu\_freq}) \\
\wedge\ & (\text{performance} \rightarrow \text{cpu\_freq}) \\
\wedge\ & (\text{powersave} \wedge \text{acpi} \rightarrow \text{cpu\_hotplug})
\end{aligned}
$$

Parent Candidates.



- **Ranked Implied Features (RIFs)**: set of features the selected feature implies sorted by similarity.
- **Ranked All-Features (RAFs)**: all features sorted by their similarity.

Implication Graph.



- An edge $v \rightarrow w$: implication from v to w exists in ϕ.
- **Direct implications** are the thick edges.
- Calculated using a SAT solver.

Similarity Heuristic.

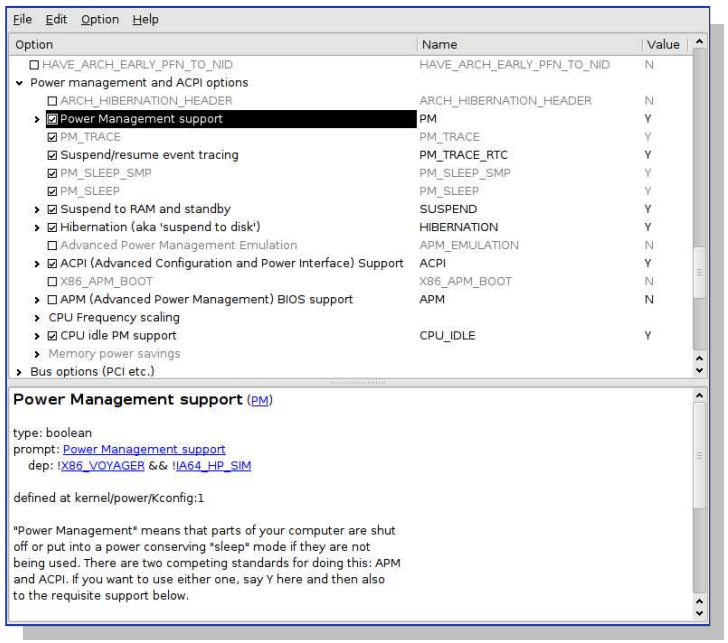
- Similarity function δ compares two features and measures how many words they share.
- Removes stop words, then calculates the sum of the **inverse document frequency** (IDF) between the two features.
- **RIFs**(f): Using the implication graph, prioritize direct implications then all remaining implications, sorted by δ.
- **RAFs**(f): All features, sorted by δ.

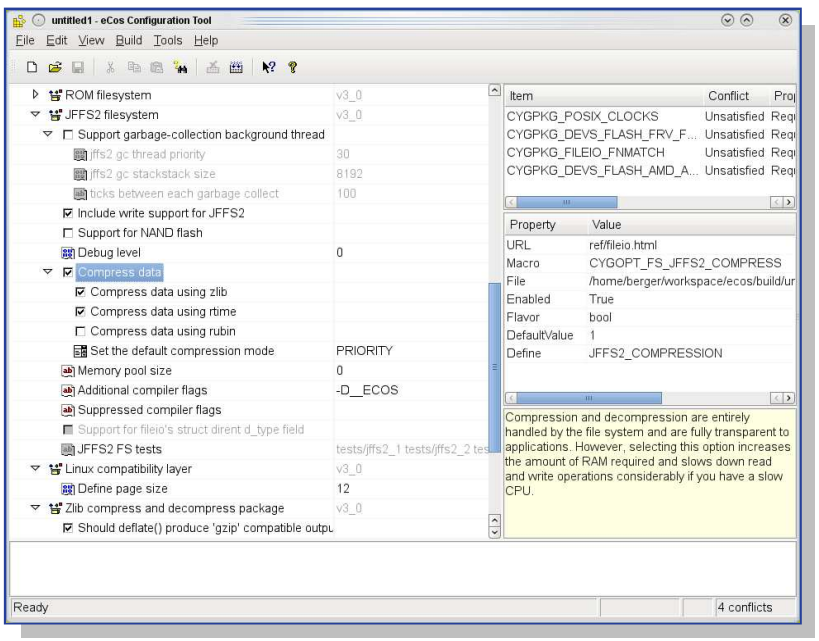
Groups and Cross-Tree Constraints.

- Construct a **Mutex Graph** to detect exclusions.
- Detect mutex and xor-groups.
- Implies and excludes edges are found using the hierarchy, groups, and the remaining edges in the implication and mutex graphs.

Outline.

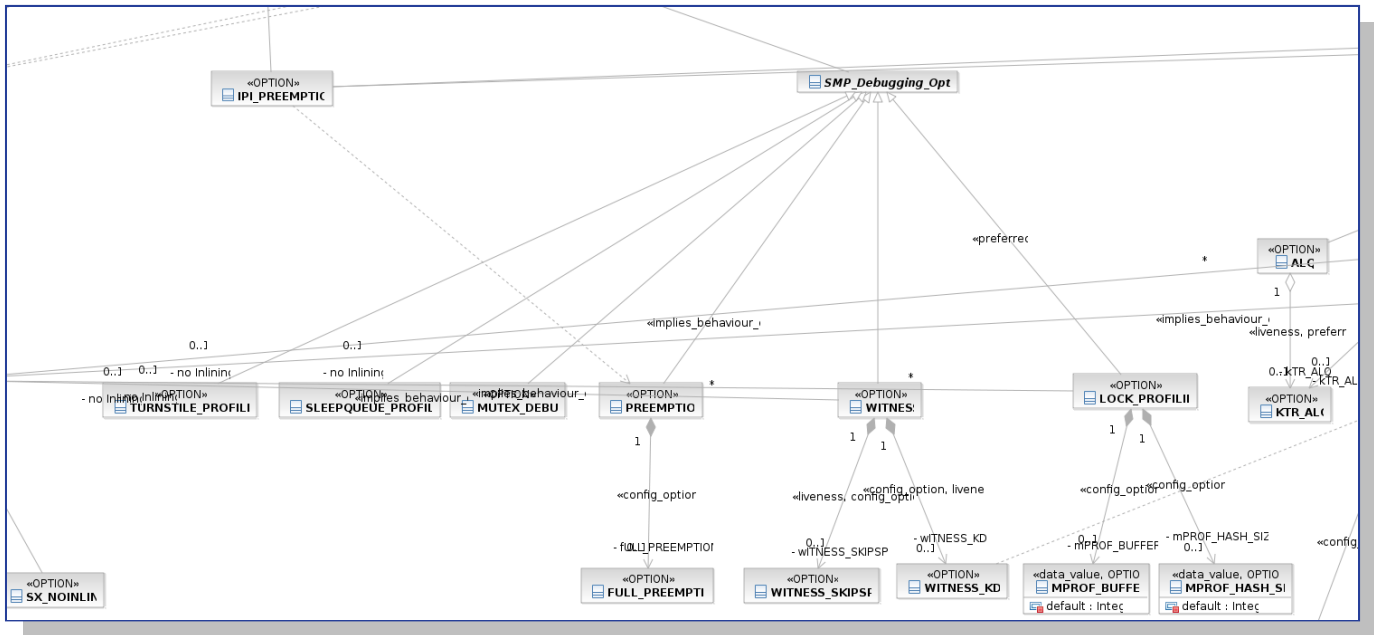
Linux and eCos Feature Models.





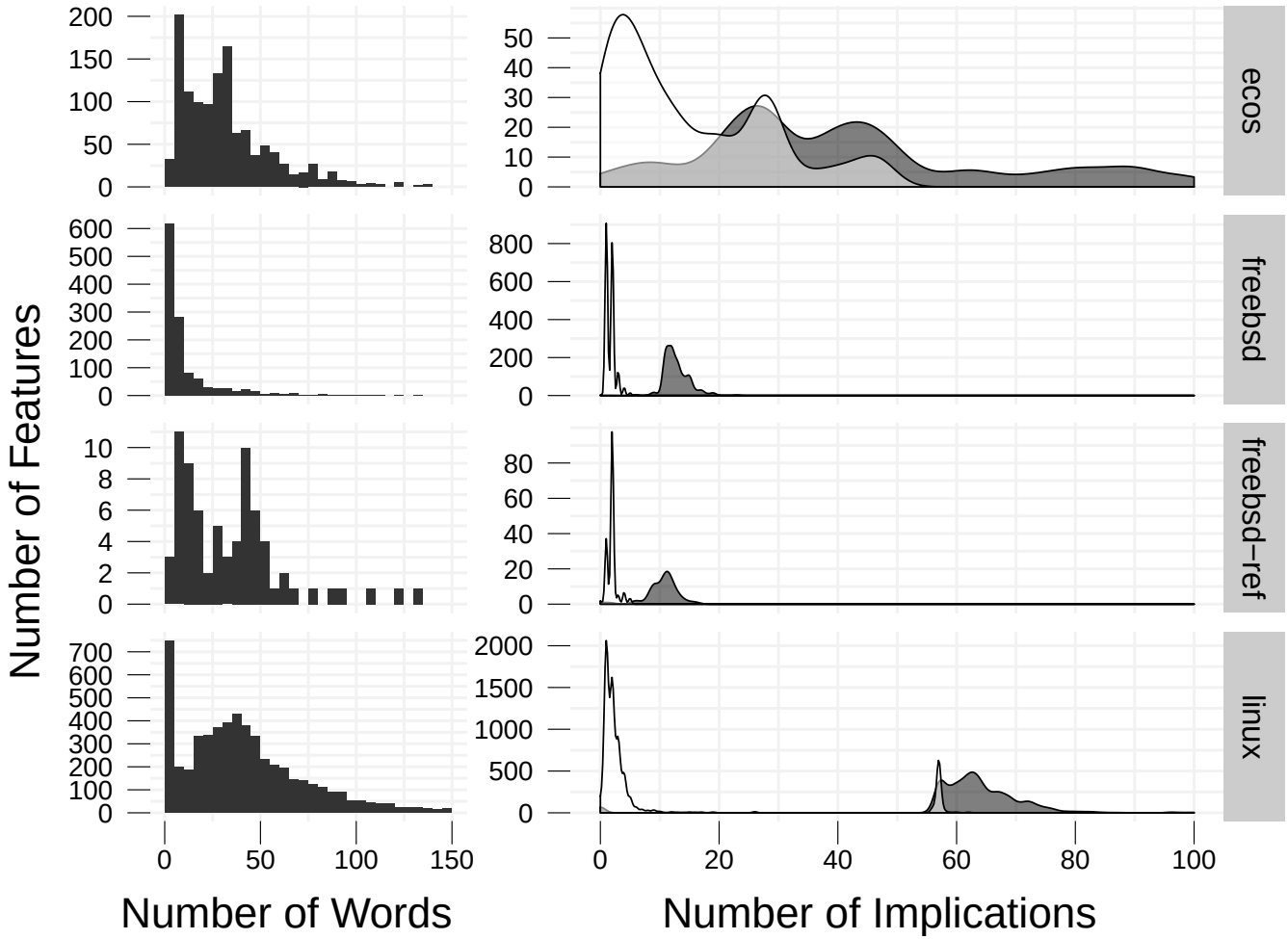
- Kconfig and CDL languages.
- Reference models to evaluate our procedures.

FreeBSD Reference Model.



- Dependencies and descriptions extracted from various artifacts in FreeBSD kernel.
- Manually created ontology of 192 features.
- Reference FM of 90 features extracted from ontology.
- Thorsten's area of expertise.

Characterization of Input Data.

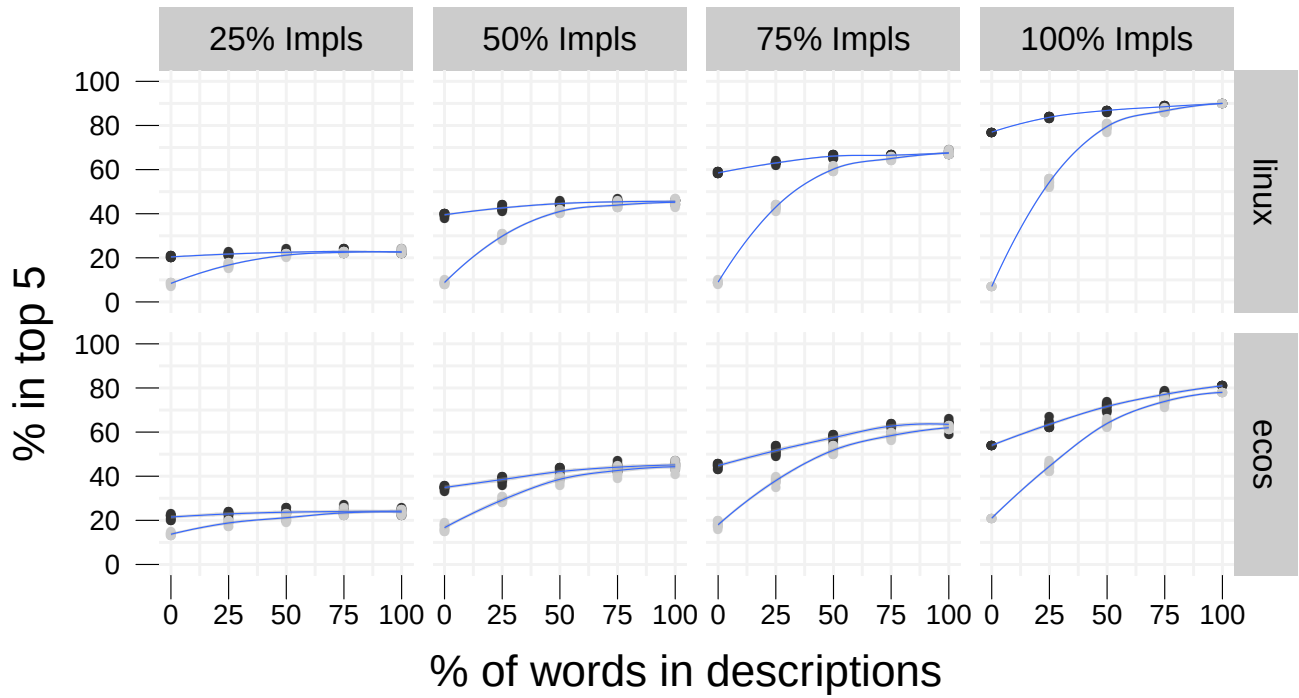


Measuring Effectiveness.

RIFs	RAFs
Is the reference parent in the **Top 5** results?	How many features do we have to examine to have a 75% chance of finding the reference parent?

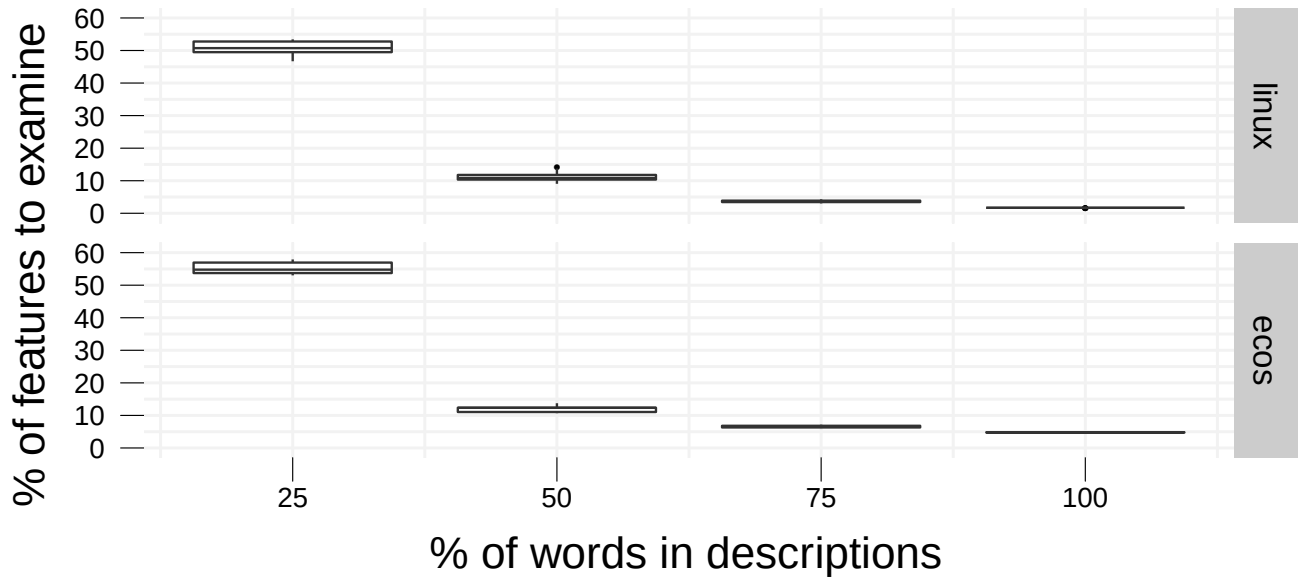
- Applied measures to complete and incomplete data sets.
- Generated sets of incomplete data for Linux and eCos by randomly removing implications and words.

RIFs.



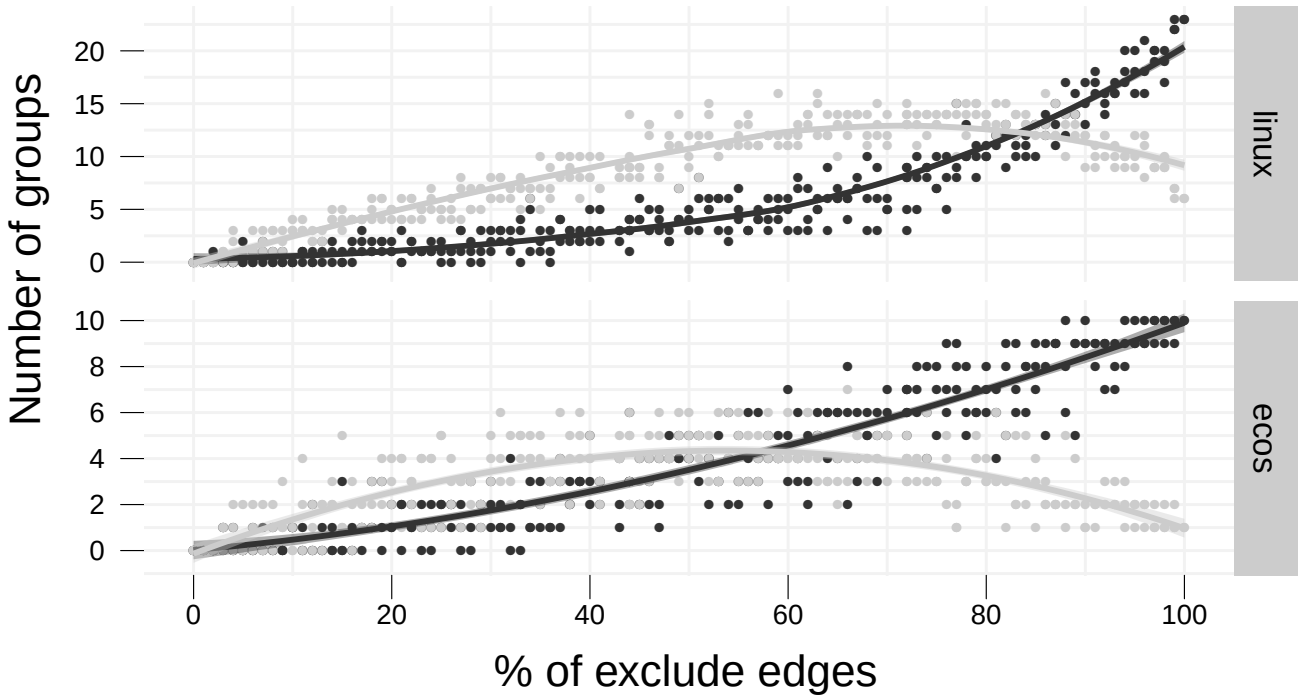
- With all implications, 76% for Linux and 79% for eCos.
- If we don't consider features at root, then 90% for Linux and 81% for eCos.

RAFs.



- With all descriptions, 3% for Linux, and 6% for eCos.

Feature Groups.



- xor-groups are detected as mutex-groups if the dependencies are incomplete.

Outline.

Conclusions.

Future Work.

- Develop a tool that incorporates these procedures.
- A study on how people create feature models; usability.

Conclusions.

- Key challenge: decide a parent for each feature.
- Ranked implied features (RIFs) and ranked all-features (RAFs) lists.
- Automatically add mandatory features, implies and excludes edges and feature groups.
- Reverse-engineer FM from project that didn't have one.