Reasoning about Feature Models in Higher-Order Logic

Mikoláš Janota    Joseph Kiniry

Systems Research Group,
University College Dublin, Ireland

SPLC ’07

Mobius    lero

IST-15905
Feature Models

- capture variability and commonality of a product line
- features represent the building blocks

```
  securityProfile
     [0..*]
    /     |
passwordPolicy permissionSet(String)
```

Janota and Kiniry
Reasoning about Feature Models in Higher-Order Logic
Why Formalize?

Disambiguation

- informal explanation of the meaning might be ambiguous
- for example, absolute vs. relative meaning of *mandatory*
Why Formalize?

Disambiguation
- informal explanation of the meaning might be ambiguous
- for example, absolute vs. relative meaning of *mandatory*

Reasoning about Feature Models
**Why Formalize?**

**Disambiguation**
- informal explanation of the meaning might be ambiguous
- for example, absolute vs. relative meaning of *mandatory*

**Reasoning at the Meta Level**

```
Feature Diagram
Meta-modeling

Formalization

Semantics

Translation

Tool 1
Reasoning
Feedback

Translation

Tool n
Reasoning
```
# Mechanization of the Formalization

## PVS
- proof assistant widely used in computer science
- typed higher-order logic language

## Pros and Cons
- reason about feature-models that have infinite number of configurations (e.g., feature cloning, attributes)
- express and reason about constraints expressible in HOL
- high level of trustworthiness of the formalization as proofs are checked by a computer
- requires expertise in using a HOL proof-assistant
- some tasks might be tedious
Feature Models as Oracles

- the set of selected features and values of their attributes constitute a *configuration*
- a configuration either does or does not *conform* to the model
### Features and Attributes

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
</tr>
<tr>
<td>size</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Feature → \( \mathcal{P}(\text{AttributeIdentifier}) \)

\text{AttributeIdentifier} → Type

Janota and Kiniry
Reasoning about Feature Models in Higher-Order Logic
### Features and Attributes

<table>
<thead>
<tr>
<th>Feature</th>
<th>name : String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>size : Integer</td>
</tr>
</tbody>
</table>

Feature → \( \mathcal{P}(\text{AttributeIdentifier}) \)

AttributeIdentifier → Type

### Feature Configurations

<table>
<thead>
<tr>
<th>Feature</th>
<th>name : String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>name : String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>name : String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>memoryRequirement : Memory</td>
</tr>
</tbody>
</table>
Features and Attributes

<table>
<thead>
<tr>
<th>Feature</th>
<th>name : String</th>
<th>size : Integer</th>
</tr>
</thead>
</table>

Feature → \(\mathcal{P}(\text{AttributeIdentifier})\)

AttributeIdentifier → Type

Feature Configurations

<table>
<thead>
<tr>
<th>Feature</th>
<th>name = &quot;air-bag&quot;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>name = &quot;cruise-control&quot;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>name = &quot;crash-detection&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryRequirement = 100MB</td>
<td></td>
</tr>
</tbody>
</table>

- \textit{value assignment function} assigns values to attributes
  \(\bigwedge \equiv \text{Feature} \rightarrow (\text{AttributeIdentifier} \rightarrow \text{AttributeValues})\)
Features and Attributes

| Feature | name : String | size : Integer |

Feature → ℘(AttributeIdentifier)
AttributeIdentifier → Type

Feature Configurations

- **value assignment function** assigns values to attributes
  \[ \text{assign} \equiv \text{Feature} \rightarrow (\text{AttributeIdentifier} \rightarrow \text{AttributeValue}) \]
- **selection function** determines the selected features
  \[ \text{select} \equiv \text{Feature} \rightarrow \text{Boolean} \]
Feature Models as Restriction Functions

a restriction function determines whether the given feature selection and attributes' values conform to the model

\[ \text{restr} \equiv \text{select} \times A \rightarrow \text{Boolean} \]
Feature Models as Restriction Functions

A *restriction function* determines whether the given feature selection and attributes’ values conform to the model.

\[
\text{restr} \equiv \text{select} \times A \rightarrow \text{Boolean}
\]

Examples of Restriction Functions

- \(f_1\) requires \(f_2\):
  \[r_1(s : \text{select}, a : A) \equiv s(f_1) \Rightarrow s(f_2)\]

- \(f_2\) requires \(f_3\) with a specific version:
  \[r_2(s : \text{select}, a : A) \equiv s(f_2) \Rightarrow (s(f_3) \land a(f_3)(\text{version}) = 7)\]

- Restriction functions can be combined:
  \[r_3(s : \text{select}, a : A) \equiv r_1(s, a) \land r_2(s, a)\]
Feature Models as Restriction Functions

A restriction function determines whether the given feature selection and attributes’ values conform to the model:

\[
\text{restr} \equiv \text{select} \times \mathbb{A} \rightarrow \text{Boolean}
\]

More Examples in PVS Notation

- A restriction function that corresponds to a requires relation:
  \[
  \text{require(requiree, required: FEATURE) : RESTRICTION = LAMBDA (select: SELECT, da: DOMAIN_ASSIGNMENT): (select(requiree) IMPLIES select(required))}
  \]
Feature Models as Restriction Functions

A restriction function determines whether the given feature selection and attributes’ values conform to the model.

\[
\text{restr} \equiv \text{select} \times \mathbb{A} \rightarrow \text{Boolean}
\]

More Examples in PVS Notation

- A restriction function that corresponds to a requires relation:
  \[
  \text{require}(\text{requireee}, \text{required}: \text{FEATURE}) : \text{RESTRICTION} = \\
  \lambda (\text{select}: \text{SELECT}, \text{da}: \text{DOMAIN_ASSIGNMENT}):
  \left(\text{select}(\text{requireee}) \implies \text{select}(\text{required})\right)
  \]

- Combine two given restriction functions:
  \[
  \text{intersect}(r1, r2: \text{RESTRICTION}) : \text{RESTRICTION} = \\
  \lambda (\text{select}: \text{SELECT}, \text{da}: \text{DOMAIN_ASSIGNMENT}):
  r1(\text{select}, \text{da}) \land r2(\text{select}, \text{da})
  \]
Meta-Level Property Example

Specialization of a Feature Model via Restriction Functions

\[ \text{specialization?}(\text{restr}_1, \text{restr}_2 : \text{restr}) \equiv \]
\[ \forall s : \text{select}; a : A \bullet \text{restr}_1(s, a) \Rightarrow \text{restr}_2(s, a) \]
Specialization of a Feature Model via Restriction Functions

\[
specialization? (\text{restr}_1, \text{restr}_2 : \text{restr}) \equiv \forall s : \text{select}; a : \mathbb{A} \bullet \text{restr}_1(s, a) \Rightarrow \text{restr}_2(s, a)
\]

Higher-Order Functions on Restriction Functions

assignment to an attribute value:

\[
assign-value(r : \text{restr}) \equiv \lambda s : \text{select}, a : \mathbb{A} \bullet r(s, a) \land (a(f_1)(\text{version}) = 3)
\]
Meta-Level Property Example

Specialization of a Feature Model via Restriction Functions

\[ specialization?(r_1, r_2 : restr) \equiv \forall s : select; a : A \bullet r_1(s, a) \Rightarrow r_2(s, a) \]

Higher-Order Functions on Restriction Functions

assignment to an attribute value:

\[ assign-value(r : restr) \equiv \lambda s : select, a : A \bullet r(s, a) \land (a(f_1)(version) = 3) \]

Reasoning

the function \( assign-value \) returns a specialization:

\[ \forall r \bullet specialized?(assign-value(r), r) \]
From Feature Diagrams to Restriction Functions

Schematically

Feature Diagram

- securityProfile
- passwordPolicy
- permissionSet(String)
- [0..*]

Formalization

- mathematical object, type TREE

Semantics Formalization

- restriction function
From Feature Diagrams to Restriction Functions

Schematically

A Function From Diagram to Restriction Function

\[
get\text{Restriction} : \text{TREE} \rightarrow (\text{select} \times \mathbb{A} \rightarrow \text{Boolean})
\]
obtain a restriction function, e.g., from a feature diagram

\[ r_0 \equiv \text{getRestriction}(tree) \]
obtain a restriction function, e.g., from a feature diagram

\[ r_0 \equiv \text{getRestriction}(\text{tree}) \]

compose the functions defining each specialization:

\[ r_1 \equiv \text{spec}_1(r_0) \]
\[ r_2 \equiv \text{spec}_2(r_1) \]
\[ \ldots \]
\[ r_n \equiv \text{spec}_n(r_{n-1}) \]
Baking Restriction Functions

Modeling Gradual Specialization of Restriction Function

- obtain a restriction function, e.g., from a feature diagram

\[ r_0 \equiv \text{getRestriction}(tree) \]

- compose the functions defining each specialization:

\[ r_1 \equiv \text{spec}_1(r_0) \]
\[ r_2 \equiv \text{spec}_2(r_1) \]
\[ \ldots \]
\[ r_n \equiv \text{spec}_n(r_{n-1}) \]

Bringing Specializations Together

\[ r_n = \text{spec}_n(\ldots(\text{spec}_1(\text{getRestriction}(tree))))\ldots) \]
Feature Models as Oracles

- the oracle is an important characteristic of the feature model
- enables unified mathematical approach
  - meta-model level, e.g., what is specialization
  - model level, e.g., record constraints in mathematical notation
- oracles are compositional