Reasoning about Feature Models in Higher-Order Logic

Mikoláš Janota    Joseph Kiniry

Systems Research Group,
University College Dublin, Ireland

SPLC ’07

IST-15905
Feature Models

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

```
securityProfile
  \[0..*]\n  \[
  \text{passwordPolicy}
  \text{permissionSet(String)}
```
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

Formalization
- Feature Modeling
- Feature and Component Relations
- Architecture Modeling

Semantics
- Translation
- Tool 1
- Tool n

Reasoning
- Feedback for the User
Why Formalize?

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

Reasoning at the Meta Level
### 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

Janota and Kiniry

Reasoning about Feature Models in Higher-Order Logic
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

```
+-----------------------------+
| Configuration               |
+-----------------------------+
| Query                      |
+-----------------------------+
| Feature Model               |
+-----------------------------+
| Answer Conforms             |
+-----------------------------+
| Answer Does not conform     |
```
# Features and Configurations

## Features and Attributes

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

- **Feature** → $\mathcal{P}$(*AttributeIdentifier*)
- **AttributeIdentifier** → Type
Features and Attributes

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

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

AttributeIdentifier \(\rightarrow\) Type

Feature Configurations

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

name : String
memoryRequirement : Memory
Features and Configurations

Features and Attributes

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

Feature → ℘(Attributeldentifier)
Attributeldentifier → Type

Feature Configurations

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = &quot;air-bag&quot;</td>
</tr>
<tr>
<td>name = &quot;cruise-control&quot;</td>
</tr>
</tbody>
</table>
| name = "crash-detection"
memoryRequirement = 100MB |

• value assignment function assigns values to attributes

\[ \mathbb{A} \equiv \text{Feature} \rightarrow (\text{Attributeldentifier} \rightarrow \text{AttributeValues}) \]
### 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>
<tbody>
<tr>
<td>Feature</td>
<td>name = &quot;cruise-control&quot;</td>
</tr>
<tr>
<td>Feature</td>
<td>name = &quot;crash-detection&quot; memoryRequirement = 100MB</td>
</tr>
</tbody>
</table>

- **value assignment function** assigns values to attributes
  \[ \mathcal{A} \equiv \text{Feature} \rightarrow (\text{AttributeIdentifier} \rightarrow \text{AttributeValueValues}) \]

- **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.

$$\text{restr} \equiv \text{select} \times \mathbb{A} \rightarrow \text{Boolean}$$
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:
  \[r_3(s : \text{select}, a : A) \equiv r_1(s, a) \land r_2(s, a)\]
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:

\[
\text{require}(\text{requiree}, \text{required}: \text{FEATURE}) : \text{RESTRICTION} = \\
\text{LAMBDA (select: SELECT, da: DOMAIN_ASSIGNMENT)}: \\
(\text{select}(\text{requiree}) \text{ IMPLIES select}(\text{required}))
\]
Feature Models as Restriction Functions

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

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

More Examples in PVS Notation

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

- Combine two given restriction functions:
  \[
  \text{intersect}(r1, r2: \text{RESTRICITION}) : \text{RESTRICITION} = \\
  \text{LAMBDA} (\text{select}: \text{SELECT}, \text{da}: \text{DOMAIN_ASSIGNMENT}): \\
  r1(\text{select}, \text{da}) \text{ AND } r2(\text{select}, \text{da})
  \]
**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

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

Higher-Order Functions on Restriction Functions

assignment to an attribute value:

\[ \text{assign-value}(r : \text{restr}) \equiv \]
\[ \lambda s : \text{select}, a : A \bullet r(s, a) \land (a(f_1)(\text{version}) = 3) \]
Specialization of a Feature Model via Restriction Functions

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

Higher-Order Functions on Restriction Functions

assignment to an attribute value:

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

Reasoning

the function \text{assign-value} returns a specialization:

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

Schematically

Feature Diagram

securityProfile

passwordPolicy

permissionSet(String)

[0..*]

Formalization

mathematical object, type TREE

Semantics Formalization

restriction function

A Function From Diagram to Restriction Function

getRestriction : TREE → (select × A → Boolean)

Janota and Kiniry
Reasoning about Feature Models in Higher-Order Logic
From Feature Diagrams to Restriction Functions

Schematically

Feature Diagram

securityProfile

passwordPolicy

permissionSet(String)

[0..*]

Formalization

mathematical object, type TREE

Semantics Formalization

restriction function

A Function From Diagram to Restriction Function

getRestriction : TREE \rightarrow (\text{select} \times A \rightarrow \text{Boolean})
Modeling Gradual Specialization of Restriction Function

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

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

Janota and Kiniry

Reasoning about Feature Models in Higher-Order Logic