FaMa FW

Framework for automated analyses of feature models

User Guide

Content

1. Introduction

FAMA-FW is a Framework for automated analysis of feature models integrating some of the most commonly used logic representations and solvers proposed in the literature (BDD, SAT and CSP solvers are implemented). FAMA is the first tool integrating different solvers for the automated analyses of feature models.

Welcome to the FAMA Tool Suite user's guide. With this guide, you will learn how to use our framework, that it is intended to be a reference in Features Model Analysis.

2. FAMA Tool Suite files

FAMA.jar: This is the core application. It must be included in the build path.

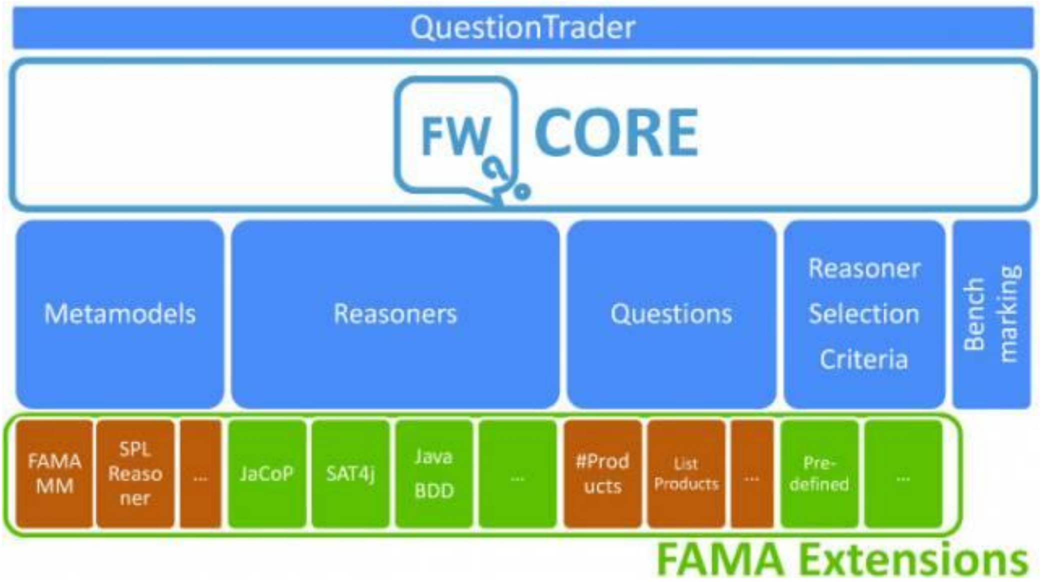
FAMAquestions.jar: This library includes the questions interfaces that are implemented in the reasoners

FAMAmodel.jar: This library includes the specification of the FAMA Feature Model. This is the variability model provided with FAMA Tool Suite by default.

JavaBDDReasoner.jar: The reasoners that FAMA Tool Suite uses to answer the questions. You can also use another one.

FAMAconfig.xml: This is the main configuration file where we could define the questions, reasoners and criteria we want FAMA Tool Suite to use.

3. FAMA architecture



We can structure FAMA files in three types:

- **FAMA core**: FAMA.jar & FAMAquestions.jar
- **FAMA metamodel**: FAMAmodel.jar
- **FAMA reasoners**: JavaBDDReasoner.jar

4. Quick Start

At first download your fama distribution that you can find at www.isa.us.es/fama in zip format.

Once you have downloaded it, the first thing you must do is unzip the file. Inside it, you will find the following folders and files:

- lib: with the jars
- docs: with the user guide
- fm-samples: with feature models samples on xml format
- files:
  - version.txt: with the information of the version of different jar files
  - license.txt: with the license terms
  - FAMAconfig.xml: it is the main configuration file. Please do not modify it, unless you are sure about your are doing
  - FAMA_logo.png: the FAMA logo
  - FAMAFirstTime.java: an example file

4.1 Creating and configuring a new empty project.

Now, you have uncompressed FaMaTS. The next step is to create a new empty java project in your IDE, and take a new class with a main method. Also, you should include in the project the FaMaTS libraries. For this, create a folder on the project called `lib`, copy inside all the jars that are on `lib` folder on FaMaTS.zip, and copy also FAMAconfig.xml and test.fama (or HIS.xml) files on the project’s root folder add FAMA.jar and FAMAmodel.jar to the build path of the project, and finally include the FAMAFirstTime.java file into your project.

Now you can run your main file and verify that everything is working correctly. The output should look like this:

```
Your feature model is valid
The number of products is: 4
Commonality of the selected: 2
Number of products after applying the filter: 2
```

4.2 Main classes

Now, we are going to detail the classes that appear on the example.

-**QuestionTrader**: It is the main interface of the tool.

-**GenericFeatureModel**: It represents an abstraction of a feature model. It can be created using an XML file.

-**Question** (and subclasses): Question class (and its subclasses, detailed more ahead) represents the abstraction of the different operations of analysis that can be performed by the FAMA Framework.

4.3 Loading a model

The first step is to create a QuestionTrader object.

```
qt = new QuestionTrader();
```

Now, we need a model to work. There are two ways to give a model to FAMA Tool Suite. We can load the model from a file, or build it from scratch and save into a file. The common way is load the model from a file. Here there's a example (with a file called test.fama).

```
GenericFeatureModel fm = (GenericFeatureModel)qt.openFile("test.fama");
qt.setVariabilityModel(fm);
```

4.4 Creating and performing analysis operations

At this time, we can create questions, and ask them to FAMA. Calling the `createQuestion` method of the class QuestionTrader we get a `Question` object. Note that you need a question identifier that you can find in Figure 3.

```
Question q = qt.createQuestion("Products");
```

Once we have created the question , we could ask it to FAMA.

```
PerformanceResult pr = qt.ask(q);
```

Finally, we can process the answer to the question provided by FAMA using the corresponding method.

```
ProductsQuestion pq = (ProductsQuestion) q;
Integer np = pq.getNumberOfProduct();
```

In Figure 3, we detail the different questions available in the current version of FAMA.

Question	Id	Class
Number of products	#Products	NumberOfProductsQuestion
Products	Products	ProductsQuestion
Valid model	Valid	ValidQuestion
Commonality	Commonality	CommonalityQuestion
Filter	Filter	FilterQuestion
Set	Set	SetQuestion

Figure 3

Note: these identifiers are in FAMAconfig.xml

And these are the methods for each question, to retrieve information about the model.

Question	Methods
Number of products	getNumberOfProducts(): long
Products	getNumberOfProduct(): int getProduct(**int** index): Product
Valid model	isValid(): boolean
Commonality	setFeature(GenericFeature f): void getCommonality(): int
Filter	addFeature(GenericFeature f,**int** cardinality): void removeFeature(GenericFeature f): void
Set	addQuestion(Question q)

Figure 4

4.5 Questions

In this section, we will explain all the questions supported now. To create a question, you should ask to QuestionTrader. Above you can see the identifiers for that.

4.5.1 Products

This question allows you to access to the set of products represented by the feature model. Once we have asked the question to FAMA, we have two possible methods to use:

-`getNumberOfProduct()` returns the total number of products of the model.

-`getProduct(int index)` returns the product (class Product) associated with that index. A product have a list of features that we can consult (getFeature(int index) method, or getNumberOfFeatures()).

4.5.2 Number of products

Only have one method:

-`getNumberOfProducts()` returns the total number of products of the model.

4.5.3 Valid

This question determines if the model is valid, i.e. it represents at least one product, and also has one method only.

`-isValid()` returns a boolean that indicates if the model is valid, or not.

4.5.4 Commonality

The goal of this question is to determine the commonality of a feature, i.e. the number of products where a given feature appears.
Hence, before we call to the ask method, it is necessary to specify a feature. The methods are:

`-setFeature(Feature f)`: with this method, we specify the feature that we want to know his commonality

`-getCommonality()`: once we have specify the feature and we have ask to FAMA, it's the moment to use getCommonality, to obtain the result.

4.5.6 Filter

With FilterQuestion, we can specify a filter on our model, adding or removing features

`-addFeature(GenericFeature f, int cardinality)`

`-removeFeature(GenericFeature f)`

To use FilterQuestion, you should use SetQuestion also (see below).

4.5.7 Set

With SetQuestion, we can compose a set of questions, and ask them one after one.

`-addQuestion(Question q)`

First, we create some questions that we consider. Now, we create the SetQuestion, we add the questions with any order, and ask to the QuestionTrader for the SetQuestion.

5. Contact us

If you want contact us for any problem or question, send a mail to fama.support@gmail.com, or visit www.isa.us.es/fama.