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

FaMaSDK.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.

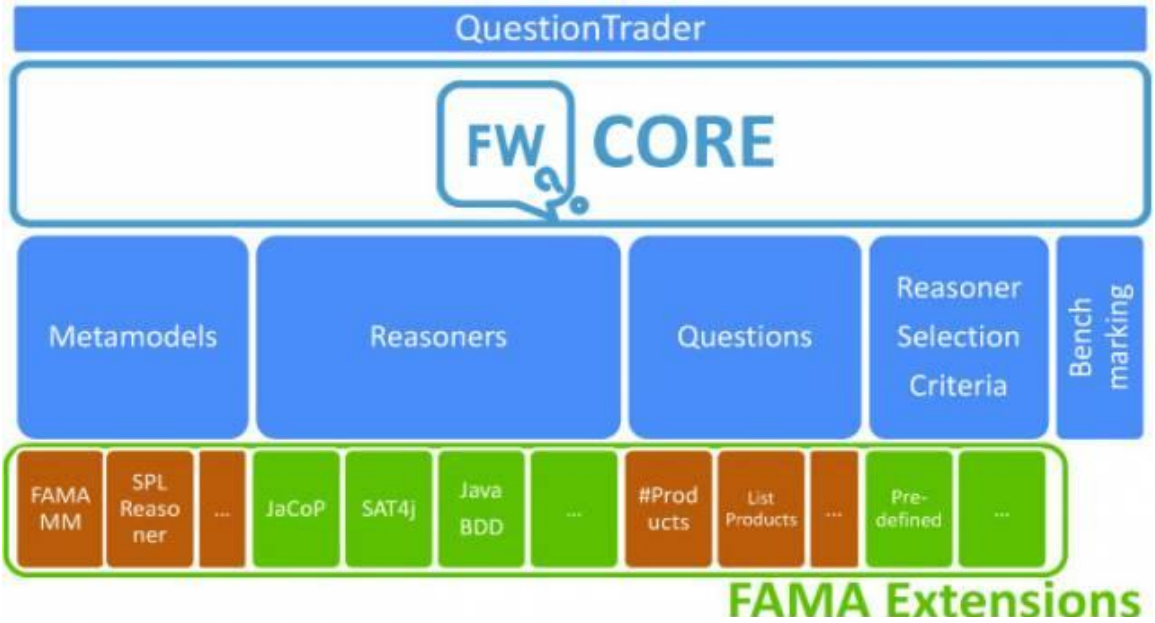
JavaBDDReasoner.jar: One of the reasoners that FaMa Tool Suite uses to answer the questions, trough Binary Decision Diagram (BDD). You can also use another one.

Sat4jReasoner.jar: Another reasoner for FaMa. This one creates and solves a Boolean Satisfiability Problem (the shorthand is SAT).

ChocoReasoner.jar: A CSP (Constraint Satisfaction Problem) reasoner for FaMa. You will find also ChocoReasoner4Explanations.jar. It is the same reasoned, but it works only with a special question of FaMa.

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**: FaMaSDK.jar & FaMaQuestions.jar
- **FAMA metamodel**: FaMaModel.jar
- **FAMA reasoners**: JavaBDDReasoner.jar, Sat4jReasoner.jar, ChocoReasoner.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
- fama-samples: FAMAFirstTime.java, DetectErrorsExample.java, ProductsAndValidProductsExample.java. Some example java classes.
- 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
    - Feature-model-schema.xsd : this is the pattern to create our xml for FAMA

4.1 Creating and configuring a new empty project.

Once you have uncompressed FaMaTS, follow these steps to run the first time FaMaTS.

1) Make sure that you have, at least, jre 5.0 (or higher) installed
2) Create a new java empty project (with, at least, jre 5.0)
3) Make a new dir at the root folder, called “lib” (if you are on NetBeans, make sure that you are on “files” view)
4) Copy to this folder all the jars that are on FaMaTS.
5) Add to the project’s build path some files previously copied: FaMaSDK.jar, FaMaQuestions.jar and FaMaModel.jar
6) Copy FaMaConfig.xml to the project’s root folder.
7) Copy the folder “fm-samples” to the project’s root folder
8) Create a new package on the project’s source folder (with the name that you want)
9) Copy the java sources that are on “FaMa-samples” on the new package
10) Run the classes

Now you can run your FAMAFirstTime 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
```

You can run the other examples included too.

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. You can change the identifiers writing on FAMAConfig.xml

Question	Id	Class
Number of products	#Products	NumberOfProductsQuestion
Products	Products	ProductsQuestion
Valid model	Valid	ValidQuestion
Commonality	Commonality	CommonalityQuestion
Filter	Filter	FilterQuestion
Set	Set	SetQuestion
Detect errors	DetectErrors	DetectErrorsQuestion
Explain errors	Explanations	ExplainErrorsQuestion
Valid product	ValidProduct	ValidProductQuestion
Valid configuration	ValidConfiguration	ValidConfigurationQuestion
Variability	Variability	VariabilityQuestion
ValidConfigurationErrors	ValidConfigurationErrors	ValidConfigurationErrorQuestion

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	`long getNumberOfProducts()`
Products	`long getNumberOfProducts()` `Collection<Product> getAllProducts()`
Valid model	`boolean isValid()`
Commonality	`void setFeature(GenericFeature f)` `int getCommonality()`
Filter	`void addValue(VariabilityElement ve, int value)` `void removeValue(VariabilityElement ve)`
Set	`void addQuestion(Question q)`
Detect errors	`void setObservations(Collection<Observation> c)` `Collection<Error> getErrors()`
Explain errors	`void setErrors (Collection<Error> colErrors)` `Collection<Error> getErrors()`
Valid product	`void setProduct(Product p)` `boolean isValid()`
Valid configuration	`void setProduct(Product p)` `boolean isValid()`
Variability	`float getVariability()`
Valid configuration errors	`Void setproduct(product p)`

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:

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

-`getAllProducts()` returns a collection with all the possible products of the feature model. 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.

4.5.5 Filter

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

```
-addValue(VariabilityElement ve, int value)
```

```
-removeValue(VariabilityElement ve)
```

To use FilterQuestion, you should use SetQuestion too.

4.5.6 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.

4.5.7 Valid product

The purpose of this question is to determine if a product is valid on a specified feature model. Before calling ask method of QuestionTrader, you should call to setProduct.

```
-void setProduct(Product p)

-boolean isValid()
```

4.5.8 Valid configuration

This question is very similar to Valid Product Question. The difference is that Valid Configuration evaluates a not finished product, only a temporally configuration before the product is finished. The methods are the same than Valid Product.

```
-void setProduct(Product p)

-boolean isValid()
```

4.5.9 Variability

Variability question is very simple. It returns the variability factor of the model.

```
-float getVariability()
```

4.5.10 Detect errors

This question is one of the most useful questions of FaMa. You can give to FaMa a feature model, and FaMa will tell you if the model has errors (and which are the errors)

```
-void setObservations(Collection<Observation> observations)

-Collection<Error> getErrors()
```

4.5.11 Explain Errors question

This question will tell you how to correct an error, previously detected with Detect errors question.

```
-void setErrors (Collection<Error> colErrors)

-Collection<Error> getErrors()
```

4.5.12 Valid Configuration Errors question

This question will tell you how to correct an error, previously detected with valid product question.

```
-void setProduct (Product p)
```

5. Metamodel (XML Schema)

The feature model metamodel is written on XML Schema. You can find here at the distribution package, or at the website of FaMa on googlecode (http://famats.googlecode.com/files/feature-model-schema.xsd)

5.1. XML edition utilities.

To edit the feature models on XML format, there are some utilities to create XML files based on a XML Schema (for us, the feature model schema above).

You can use the Eclipse XML Editor if you are an eclipse user. Try to create a new XML file on Eclipse. For this, click File → New → Other, and expand XML option (new XML). If your Eclipse version does not have this plugin, go to Software Updates → Available Software, and look for Web and Java EE Development →Eclipse XML Editors and Tools. Finally, select and install it.

Now, you have installed the XML Editor plugin, create the XML file inside your project home folder, and click next. Mark the option “Create XML file from a XML schema file”, and click next again. Select file from workspace, importing it (you must download the schema file). At last, select feature-model as root element, and finish.

Open the design perspective on XML file, and edit it with context menu.

6. References

FaMa’s site: http://www.isa.us.es/fama

Project site on google code: http://code.google.com/p/famats

ISA research group: http://www.isa.us.es

Sat4j: http://www.sat4j.org

JavaBDD: http://javabdd.sourceforge.net

Choco-Solver: http://choco-solver.net

7. 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.