Learning from Difference: An Automated Approach for Learning Family Models from Software Product Lines

(Artifact)

Carlos Diego N. Damasceno
damascenodiego@usp.br
University of Sao Paulo, BR and
University of Leicester, UK

Mohammad Reza Mousavi
mm789@leicester.ac.uk
University of Leicester
Leicester, UK

Adenilso Simao
adenilso@icmc.usp.br
University of Sao Paulo (ICMC-USP)
São Carlos, SP, BR

ABSTRACT

Family models are useful assets to represent variability and pave the way for efficient model-based testing and model checking for SPLs. Albeit reasonably efficient, their creation and maintenance tend to be time consuming and error-prone, especially if there are crosscutting features. To tackle this issue, we introduce $FFSM_{Diff}$, a fully automated technique to learn featured finite state machines (FFSM), a family-based formalism that unifies Mealy Machines from SPLs into a single representation. Our technique incorporates variability to match and merge Mealy machines into succinct FFSMs, especially if there is high feature reuse. This submission aims at the documentation of the artifacts created for the homonym paper accepted at the research track of the SPLC'19.

1 INTRODUCTION

In our homonym paper, we provide a description of $FFSM_{Diff}$, a technique to learn featured finite state machines (FFSM) [3], and an empirical analysis of (i) its effectiveness for learning succinct models and (ii) the correlation between the size of learnt FFSMs and amount of feature sharing. There we provide a summary of the experiment and a link to our lab package, as well as references to third-party tools.

In this paper, we provide more information about our repository at https://github.com/damascenodiego/learningFFSM that should be sufficient to repeat and reproduce our experiments. We hope our artifacts are useful to researchers and practitioners explore related work or replications using their particular setting.

2 THE REPOSITORY

In Figure 1, we depict our repository structure with links to folders. In the `FFSM_diff` repository, we have a Java project that can be opened using the Eclipse IDE [4] and JDK version 1.8. This project has three subfolders: `Benchmark_SPL`, `lib`, and `src`. Details about the subject systems, code artifacts and how to replicate our study are shown in Sections 3, 4 and 5, respectively.

In *Benchmark_SPL*, we have the subject systems , i.e., `agm`, `vm`, and `ws`; and their respective models (i.e., FSMs, FFSMs) as KISS files, feature models as XML files and test scripts. Images of the models are also available. In `learningFFSMs`, there is an RStudio [6] project for the analysis of result. The `learnFFSM.jar` is a compiled version of the $FFSM_{Diff}$ algorithm. To replicate our experiments, run the test scripts `run_‹ID›.py` and `run_‹ID›_pairs.py`.

SPLC '19, September 9–13, 2019, Paris, France
2019. ACM ISBN 978-1-4503-7138-4/19/09…$15.00
https://doi.org/10.1145/3336294.3336307

In folder *lib*, we have some of the libraries required in our project. These include `FeatureIDE` [7], and `Apache Commons Math` [1]. Other libraries (e.g., LearnLib and AutomataLib [5]) are imported using Apache Maven.

In folder `src`, we find the source-code of the $FFSM_{Diff}$ algorithm. Our project contains two packages: `br.usp.icmc` and `uk.le.ac`. The former has code artifacts developed by Fragal et al. [3] that we used in the first phase of our study to validate FFSMs and derive FSMs. The latter has code artifacts that we developed to (i) read/write FSMs; (ii) solve systems of linear equations; and (iii) merge FSM models into annotated FFSMs. Our code artifacts are discussed in Section 4.

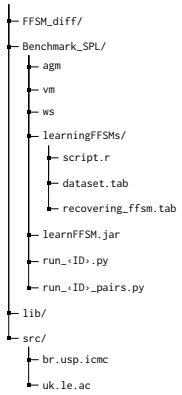

Figure 1: Project structure

3 THE SUBJECT SYSTEMS

The Arcade Game Maker (AGM) SPL is a well-known pedagogical example that describes arcade game machines with different game rules. In Figure 2, our version of the AGM SPL [3] supports three alternative games and one optional feature to *Save* the game. The FSMs derived from the AGM FFSM have at least three states for the game modes *start*, *running* and *paused*. If *Save* feature is included, a fourth state is added.

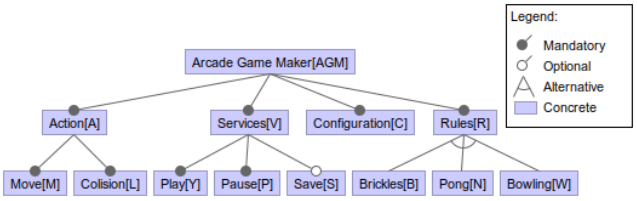

Figure 2: The AGM feature model

The Vending Machine (VM) is an SPL that we hand-crafted based on featured transition systems from a collection of illustrative SPLs [2]. In Figure 3, our VM SPL supports three beverages, one optional *RingTone* played when the beverage is completed, and two alternative currencies. These features composed interesting scenarios as they resulted on FSMs with distinct structures and languages. For the FSMs derived from the VM FFSM, we highlight main two changes: (i) the addition of states for each of beverage; (ii) changes in the initial state for each currency. The VM FFSM is our largest SPL in terms of number of products and states.

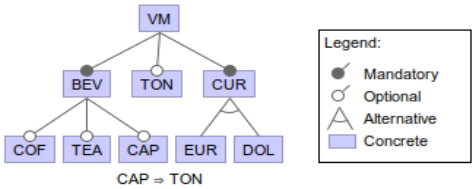

Figure 3: The VM feature model

The Wiper System (WS) is another SPL that we hand-crafted based on the same collection aforementioned [2]. In Figure 4, our WS has of two subsystems – a sensor to detect rain, and the wiper itself; that come in two qualities, i.e., *high* and *low*, and one optional feature for permanent movement. A high quality sensor can discriminate between heavy/little rain, whereas a low quality sensor can only distinguish between rain/no rain. Similarly, the high quality wipers can operate at two speeds, and the low quality wiper operates at one single speed. These features lead to significant changes in the structure and language of its derived FSMs.

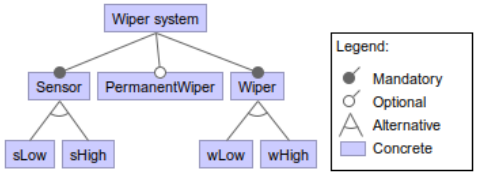

Figure 4: The WS feature model

4 THE CODE ARTIFACTS

In folder `src`, we find the source-code of the $FFSM_{Diff}$ project. The code artifacts we developed are found in the package `uk.le.ac`. These include classes to (i) read/write FSMs; (ii) solve systems of linear equations; and (iii) merge FSM models into FFSMs.

For reading and writing FSMs, we designed the `ProductMealy` class using the `CompactMealy` class from LearnLib [5]. This class extends basic operations over FSMs (e.g., reset, transition/output functions) with methods to maintain product configurations, as specified by our `IConfigurableFSM` interface.

For solving the systems of linear equations, we used the `Apache Commons Math` library [1]. This open-source library supported us to construct and solve the systems of linear equations to find the state pairs most likely to be equivalent.

To represent FFSMs, we designed the `FeaturedMealy` class using the `FastNFA` class, one of the LearnLib building blocks to represent non-deterministic models. We opted for a non-deterministic representation because, if we ignore presence conditions, FFSMs can be seen as a non-deterministic FSM. To represent conditional states/transitions, we designed the classes `ConditionalState` and `ConditionalTransition` with collections of `Node` objects. The `Node` class is a `FeatureIDE` building block to represent feature constraints.

5 REPLICATING OUR EXPERIMENT

To replicate our experiment, first download our GitHub repository. In Linux, run the `git clone` command as in Listing 1.

```
git clone https :// github .com/damascenodiego/learningFFSM.git
```

Listing 1: Cloning the $FFSM_{Diff}$ repository

After cloning our repository, the files required to replicate our experiments are available in the `Benchmark_SPL` folder. The jar file `learnFFSM.jar` is a compiled version of our $FFSM_{Diff}$ algorithm. To display the help menu, run `java -jar learnFFSM.jar -h`.

```
−clean <arg>   Simplify FFSM labels
−fm <arg>      Feature model
−fref <arg>    FFSM reference
−h             Help menu
−mref <arg>    Mealy reference
−out <arg>     Output file
−updt <arg>    Mealy update
```

Listing 2: The $FFSM_{Diff}$ help menu

To run our algorithm, we require a feature model in XML format compatible with `FeatureIDE` (to `-fm`), a reference FSM or FFSM model (to `-mref` or `-fref`) and an updated FSM version (to `-updt`). The `-out` parameter sets the path to the output file and the `-clean` parameter cleans FFSMs by removing self-loop transitions.

The FFSMs models are represented in the KISS format, where transitions are denoted as in Listing 3.

```
1@[(TRUE)] −− a@[(TRUE)]/_1() −> 4@[(B)]
1@[(TRUE)] −− b@[(W)]/_1() −> 5@[(TRUE)]
1@[(TRUE)] −− c@[(W and (not S))]/_1() −> 1@[(TRUE)]
1@[(TRUE)] −− c@[((not W) or S)]/_0() −> 1@[(TRUE)]
1@[(TRUE)] −− d@[(S)]/_0() −> 1@[(TRUE)]
```

Listing 3: Example of FFSM in KISS format

The FSMs are represented in a similar fashion, except for the product configuration in the first line of the file, as in Listing 4.

```
AGM A M L C R V Y P B (not S)
1 −− a/_1() −> 4
1 −− c/_0() −> 1
1 −− b/_0() −> 1
```

Listing 4: Example of FSM in KISS format

To run our experiments, we designed python scripts to call our tool to include all products into a single FFSM and learn fresh FFSMs from all pairs of products. These are `run_‹ID›.py` and `run_‹ID›_pairs.py`, respectively. The learnt FFSMs are saved in the folders `learnt` and `learnt_pairs` and measures are printed in the standard output. For the statistical analysis, we used the R script `learningFFSMs/script.r` to load the measures organized in tabular format, as in files `learningFFSMs/dataset.tab` and `learningFFSMs/recovering_ffsm.tab`, and plot our statistics.

REFERENCES

[1] Apache. 2016. Commons Math: The Apache Commons Mathematics Library. http://commons.apache.org/. [Online; accessed 28-Mar-19].
[2] Andreas Classen. 2010. Modelling with FTS: a Collection of Illustrative Examples. https://researchportal.unamur.be/files/1051983/69416.pdf
[3] Vanderson Hafemann Fragal, Adenilso Simao, and Mohammad Reza Mousavi. 2017. *Validated Test Models for Software Product Lines: Featured Finite State Machines.* Springer International Publishing, Cham, 210–227.
[4] Eclipse IDE. 2019. Eclipse desktop and Web IDEs. https://www.eclipse.org/ide/. [Online; accessed 19-May-19].
[5] Harald Raffelt and Bernhard Steffen. 2006. *LearnLib: A Library for Automata Learning and Experimentation.* Springer, Berlin, Heidelberg, 377–380.
[6] RStudio. 2019. RStudio: Open source and enterprise-ready professional software for data science. https://www.rstudio.com/. [Online; accessed 19-May-19].
[7] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming* 79 (2014), 70–85.