Autofunk: an inference-based formal model generation framework for production systems.

William Durand[1] and Sébastien Salva[2]

[1] Michelin, France - william.durand@fr.michelin.com
[2] Auvergne University, France - sebastien.salva@udamail.fr

Abstract. In this paper, we present *Autofunk*, a fast and scalable framework designed at Michelin to automatically build formal models (Symbolic Transition Systems) based on production messages gathered from production systems themselves. Our approach combines model-driven engineering with rule-based expert systems and human knowledge.

Keywords: Model inference, symbolic transition system, expert system, production system, regression testing

1 Introduction

Michelin is a worldwide tire manufacturer which designs all its factories, production systems, and software by itself. Like many other industrial companies, Michelin follows the Computer Integrated Manufacturing approach, using computers and software to control the entire manufacturing process. Michelin Level 2 applications are often deployed for 20 years, and are very important for its business. Maintaining these software is inevitable, but due to their importance, this is risky. That is why Michelin puts a lot of efforts in documenting how these applications behave. Unfortunately, keeping such knowledge up to date is difficult, and it often implies under-specified or not documented legacy systems that no one wants to maintain because of lack of understanding.

In this paper, we focus on this problem for legacy systems in an industrial context. Model inference is a recent research field that addresses this issue. Models are here built from execution traces (i.e. sequences of observed actions of an application). Several approaches have been proposed for different types of systems, usually GUI applications [4, 5, 3]. However, our experience shows that these approaches are not tailored to support running production systems that are complex and distributed over several devices. From the literature, we deduced the following key observations:

- Model inference approaches learn approximate models capturing the behaviours of a system and more. In our context, we want exact models that could be used for regression test case generation,
- Some approaches perform active testing on the systems to learn models. Applying active testing on running systems is not possible since these must not be disrupted,

– Production systems exchange thousands and thousands of messages a day. Most of the model inference approaches cannot take such a huge amount of information to build models.

That is why we have developed *Autofunk*, our fast and scalable framework to infer both exact and formal models from production messages, using expert systems and inference rules to emulate human knowledge, and transition systems to embrace formal tools.

2 Framework

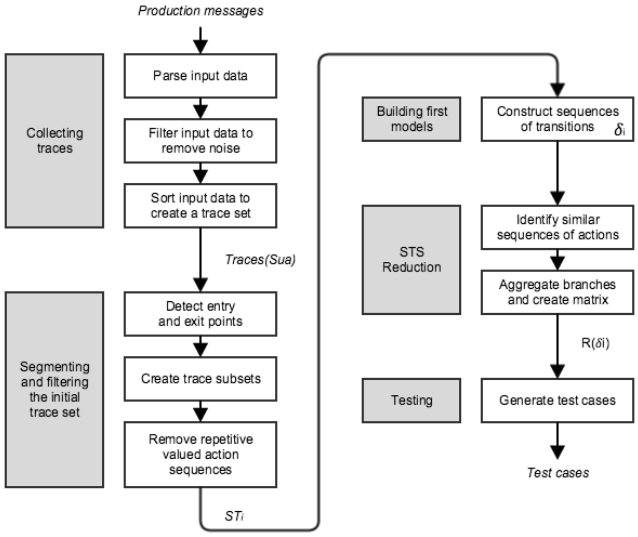

Fig. 1: Autofunk's architecture

Figure 1 depicts the architecture of our framework. It contains five modules (in grey in the figure), the first four modules aim at building models and the last one (which will not be described in the paper) is used to generate test cases. *Autofunk* is developed in Java and relies on *Drools* [3], a powerful Java rule-based expert system engine which supports knowledge bases with facts given as Java objects.

We consider Symbolic Transition Systems (STS) [2] as models for representing production system behaviours. STSs are state machines incorporating actions, labelled on transitions, that show what can be observed on a system. In addition, actions are tied to an explicit notion of data.

[3] http://www.drools.org/

2.1 Production messages and traces

Autofunk takes production messages as input from a (running) system under analysis Sua. A production message can be either a stimulus or a response, owns a timestamp defined by a global clock, and contains variable data. These messages are formatted no matter their initial source (e.g. a logging system in our case), so that it is possible to use messages from different providers. We call *valued actions* the resulting set of messages.

Some of these actions are not part of the functioning of the system (logging information for instance), and thus must be removed. Filtering is achieved by Drools expert system and a few inference rules given by a domain expert. The remaining actions are sorted to produce an initial set of traces denoted $Traces(Sua)$.

2.2 Trace segmentation and STSs

We define a complete trace as a trace containing all actions expressing the path taken by a product in a production system, from one of its entry points to one of its exit points. In the trace set $Traces(Sua)$, we do not want to keep incomplete traces. *Autofunk* performs a statistical analysis on $Traces(Sua)$ and computes two ratios for the first and last valued actions of every trace in order to automatically find the entry and exit points of Sua.

$Traces(Sua)$ is then split into subsets ST_i, one for each entry point of Sua. Every trace set ST_i will give birth to one model, describing all possible behaviours starting from its corresponding entry point. Here we obtain the set $ST = \{ST_1, ..., ST_N\}$ such that each $ST_i \subseteq Traces(Sua)$.

Given a subset ST_i in ST, a first STS denoted $\mathcal{S}$ is built by relying on the LTS semantics [2] transformation applied in a backward manner. This model has a tree structure and its traces are equivalent to those of ST_i.

2.3 STS reduction

The previous model $\mathcal{S}$ is often too large, and thus cannot be beneficial as is. Using such a model for testing purpose would lead to too many test cases for instance. That is why *Autofunk* performs a reduction step, aiming at diminishing the first model into a second one, denoted $R(\mathcal{S})$ that will be more usable.

Most of the existing approaches propose two solutions. Models can directly be inferred with high levels of abstraction but it implies not exact models. The second solution is to apply a minimisation technique [1] which guarantees trace equivalence, but it is costly and highly time consuming on large models. As a result, we chose a simpler approach which consists in combining branches that have the same sequences of actions so that we still obtain a model having a tree structure. *Autofunk* generates a signature for each branch b, i.e. a hash (SHA1 algorithm) of the concatenation of the signatures of the actions of b. This gives good results in terms of STS reduction and requires low processing time, even with millions of actions.

3 Evaluation

We conducted several experiments with real sets of production messages, recorded in one of Michelin's factories. The most significant one ran with a month of data. *Autofunk* handled 10 million production messages in 5 minutes to build two models including around 1,600 branches (parsing step put aside). The framework revealed 120,000 complete traces, which represents 78% of the initial trace set, and reduced the models by 97%. More results can be found in [6].

4 Conclusion

We built a fast and scalable framework combining model inference, expert systems and statistical analyses to derive STSs models based on production traces, i.e. generating formal models from running production systems.

We focused on reducing the models while keeping them exact thanks to trace equivalence preservation, and also because we considered complete branches only. We would like to investigate partial branch concatenation to reduce generated models because we believe that models could be even more reduced. However, this would probably affect performance as partial branch concatenation is time consuming, and we don't want to sacrify speed.

This framework is part of a regression testing system we are working on. In the future, we plan to work on passive testing by applying our framework for different versions of a system and draw conclusions based on the generated models. This will be part of a newer testing module.

References

1. Abdulla, P.A., Kaati, L., Hgberg, J.: Bisimulation minimization of tree automata. Tech. rep., In Proc. 11th Int. Conf. Implementation and Application of Automata, volume 4094 of LNCS (2006)
2. Frantzen, L., Tretmans, J., Willemse, T.: Test Generation Based on Symbolic Specifications. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. pp. 1–15. No. 3395 in Lecture Notes in Computer Science, Springer (2005)
3. Mariani, L., Pezze, M.: Dynamic detection of cots component incompatibility. IEEE Software 24(5), 76–85 (2007)
4. Memon, A., Banerjee, I., Nagarajan, A.: Gui ripping: Reverse engineering of graphical user interfaces for testing. In: Proceedings of the 10th Working Conference on Reverse Engineering. pp. 260–. WCRE '03, IEEE Computer Society, Washington, DC, USA (2003), `http://dl.acm.org/citation.cfm?id=950792.951350`
5. Mesbah, A., van Deursen, A., Lenselink, S.: Crawling Ajax-based web applications through dynamic analysis of user interface state changes. ACM Transactions on the Web (TWEB) 6(1), 3:1–3:30 (2012)
6. Salva, S., Durand, W.: Inferring formal models from production systems. Tech. rep., LIMOS, http://sebastien.salva.free.fr/RR-15-02.pdf (2015), LIMOS Research Report RR-15-02