State Machine Mining from Embedded Software: An interactive approach

Wasim Said, Jochen Quante

Robert Bosch GmbH, Corporate Research
Renningen, Germany

{Wasim.Said, Jochen.Quante}@de.bosch.com

Abstract

Understanding embedded legacy software is one of the major challenges that software developers have to deal with. The extraction of high level and understandable models from the code, such as state machines, is therefore highly desired. The main drawback of fully-automatic state machine mining approaches is that the mined models are too detailed and not understandable. An effective way to make theses models understandable is the interaction with the experts.

In our recent work [3], we presented an approach for interactive state machine mining from embedded software written in C or modelled in ASCET. In this paper, we present an overview of this approach, and we present the basics that we found to be necessary for extracting state machines interactively from embedded software. This work is based on our first experiments with Bosch experts about the efficiency of the approach and their experience with state machines in embedded software.

1 Introduction

Embedded legacy software contains lots of expert knowledge that has been cumulated over many years. The software usually provides highly valuable and indispensable functionality. At the same time, it becomes more and more complex to understand and maintain. Mining of understandable models, such as state machines, from legacy software can greatly support developers in maintenance tasks. Developers need to understand the software to be able to evolve it. Fully-automatic model mining approaches consider all the details in the code, which makes the mined models too detailed and not understandable. Therefore, the mined models are not suitable for human comprehension. On the other side, manual model extraction provides high quality models regarding the understandability aspect. However, it is a very laborious and error-prone task. Combining automatic model mining with user interaction can overcome the aforementioned drawbacks: when the user determines the relevant details and abstracts away everything irrelevant, the models become much more understandable and can also be extracted faster.

2 State machine extraction steps

In this section, we introduce the necessary steps to extract state machines from embedded software and describe how we implement them in our approach.

1. **Static analysis:** The code of embedded systems and especially real time systems cannot easily be instrumented for dynamic analysis [4]. Also, it is very difficult to find good and representative test cases. Therefore, static analysis is the best technique for this use case. Static analysis collects information about all variables, paths and other details purely based on the source code. We use concolic testing [1] in path enumeration because of its advantages over symbolic execution, especially with respect to constraint solving.

2. **Identification of state variables:** The most of state machine mining approaches deals with object oriented systems, where state variables are simply the member variables of a class. These approaches cannot be applied to the procedural code of embedded systems. Therefore, we have to come up with additional heuristics to determine the relevant state candidate variables: For example, the variable must be global or static and it has to influence a control decision.

3. **Extraction of states:** After the determination of state variables, the possible states are extracted from code. A state is characterized by an invariant that holds as long as the system does not leave the state. In our work, we use the techniques presented in the work of Sen and Mall [5] – which is based on the work of Kung et al. [2] – to define states. Kung determines the possible values/ranges of a state variable, depending on the control decisions in code that contain these variables. Sen uses an SMT solver to check the satisfiability of the states that are derived from control decisions.

4. **Extraction of transitions:** This step checks for each pair of extracted states if there is a transition between the two states. We also build upon the approaches of Kung et al. [2] and Sen and Mall [5] in this step. Transitions are extracted with the

help of an SMT solver, which checks if there is a path condition that can be satisfied along with the state conditions of the considered transition.

5. **Extraction of transition conditions:** To extract transition conditions, it is necessary to collect the conditions of all possible paths between two states and connect them with OR operator.

6. **Reducing transition conditions:** Transitions of an extracted state machine can have very complex conditions in the form of boolean expressions. The reduction of these conditions to an understandable level is then essential.

3 Interactive exploration

In this section, we introduce different user interaction scenarios, which are implemented in our approach [3] and combined with the steps in Section 2 to make the mined models understandable.

1. **Selecting state variables subset:** Instead of extracting state machines from all state variables in code, the user can select the relevant variables and get models of only these variables.

2. **Determining the importance of state variables:** In complex software, the number of identified state variables can be very large. In addition, the user may have no idea about the system. Therefore, he/she may not be able to determine the relevant variables. In this case, we give the user information about the effect size of the state variables on the whole function, which can be used as an indication of relevance of each variable.

3. **Selecting available states or defining new states:** This scenario enables the user to select only the relevant states of a state variable and not all of them. For example, if the state variable a has the states $a \leq 0$, $a == 0$, $a > 0$: When the user is only interested in the states $a \leq 0$ and $a == 0$. Then he/she can select these states and gets the model that describes only the transitions between these two states. Furthermore, the user is also able to define his/her own states, such as $a == 0$ and $a \neq 0$, and then he/she will also become the corresponding model of these two states.

4. **Adding constraints:** The user can also add constraints on non-state variables in code. For example, setting some input variables or parameters to specific value ranges describes how the transitions between specific states can change according to these constraints. Our tool provides information about the values/ranges of all variables in code, and the user can select the relevant choices from the available information or define new constraints in the same way.

These interaction scenarios can be applied separately or in any combination, which helps the user get the desired understandable models quickly.

4 First Feedback from experts

We have conducted an experiment with Bosch experts to assess the efficiency of the interactive state machine mining approach. We have presented different models extracted from real-world automotive functions to the experts, who assessed the understandability of the mined models and to which extent are these models useful for them. The feedback was very promising: The experts reported that the interactively extracted state machines can be very helpful for program understanding, software maintenance, debugging, validation, verification and other reverse and forward engineering activities. Furthermore, they express a big interest to use the tool. Because they find, that with the help of our approach, the time and effort spent on program understanding can be drastically reduced. In addition, the interactive approach can be useful for users with different levels of system knowledge: When the user knows the system, he/she can quickly determine the relevant variables, states and constraints. A user who has no idea about the system can explore the system step by step using the information presented by the tool about the importance of the variables.

5 Stage of the research and outlook

We have implemented the steps of the state machine mining approach and the interactive measures as described in Section 2 and 3. Reduction of transition conditions is our current work. Furthermore, we plan to optimize the required runtime to extract the state machines and to add more interaction scenarios to the state machine mining process.

References

[1] P. Godefroid. Compositional dynamic test generation. In *Proc. of 34th POPL*, pages 47–54, 2007.

[2] D. C. Kung, N. Suchak, J. Z. Gao, P. Hsia, Y. Toyoshima, and C. Chen. On object state testing. In *Proc. of 18th Int'l Computer Software and Applications Conference (COMPSAC)*, pages 222–227, 1994.

[3] W. Said, J. Quante, and R. Koschke. Towards interactive mining of understandable state machine models from embedded software. In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD*, pages 117–128, 2018.

[4] V. Schulte-Coerne, A. Thums, and J. Quante. Challenges in reengineering automotive software. In *Proc. of 13th European Conf. on Software Maintenance and Reengineering*, pages 315–316, March 2009.

[5] T. Sen and R. Mall. Extracting finite state representation of Java programs. *Software & Systems Modeling*, 15(2):497–511, 2016.