Review of Ing. Jan Fiedor's PhD thesis

Judgment

Ing. Jan Fiedor's “Practical Methods of Automated Verification of Concurrent Programs”, submitted as PhD thesis, is a very good research work, well written and well organized, that provides novel contributions, both practical and theoretical, to the area of concurrency errors detection.

The general goals of the thesis are particularly interesting and worthy to be investigated. Targeting the efficiency and the usability of approaches for detecting concurrency errors is widely needed, as no approach will be ever used outside the research community if it is not usable. Also considering less studied kinds of error is a good contribution: I agree with the statement that the fact that such errors are not so investigated does not necessarily mean that they are not interesting.

The thesis provides different novel contributions.

- The main one is the *ANaConDA* framework that easies the development of dynamic analyzers for C/C++ programs; having such a framework can be particularly useful for those writing dynamic analyzers, who can exploit the capabilities of the tool to collect information from the running program and insert noise in particular places (in order to trigger concurrency errors).
- The new proposed noise heuristics are reasonable, and the wide set of performed experiments on existing and new heuristics constitute a good demonstration of the advantages of using noise injection to trigger concurrency errors, but they also show that finding the optimal noise setting is particularly tricky.
- The integration in ANaConDA of the support for *contracts for concurrency* and their extension with *data flow information* and *contextual information* is particularly valuable: it is a nice way to specify forbidden method interleavings and to detect atomicity violations.

All the approaches proposed in the thesis have been properly evaluated; the performed evaluations are always fair as they do not try to overestimate the approach: they show the benefits, but they also correctly identify the limits.

The only minor problem with the thesis is that, in few points, it contains some redundancies due to the fact that the thesis is an organized collection of published works and some common parts of the different papers are not always properly merged.

The thesis is based on a good number (7) of publications, mostly presented in good conferences/workshops (in particular RV 2012, RV 2013, and ICST 2017), and in the STVR journal. Moreover, all the publications are presented in conferences/journals that are well-targeted to the thesis topics (dynamic analysis, runtime verification, and testing): this means that the community working on these topics has recognized the research as valuable. Moreover, as a confirmation that the research reported in the thesis fulfilled its aim of practical usability, a paper on ANaConDA was awarded as best tool paper at RV 2012.

Based on my previous observations (and on the more detailed comments reported below), I think that the work done by Ing. Jan Fiedor is very valuable and well presented in his PhD thesis; I therefore think that the thesis meets all the requirements specified by the "Study and Examination Regulations of The Brno University of Technology", needed for the conferment of the PhD title.

In the following, I provide a more detailed evaluation of the different chapters of the thesis.

Detailed comments

Chapters 1-2

Chapter 1 provides a nice introduction to the need/advantages of multi-threading and of the concurrency problems it introduces. Then, it provides an overview of existing techniques for verifying/testing multi-threaded programs. It also points out the main weaknesses of the existing approaches. Chapter 2 overviews different concurrency errors and techniques proposed to detect them. Also less studied errors as order violations are devised.

As minor problem, I would have liked to have a common notation to describe multi-threaded programs (thread, read and write operations, locks, etc.); such notation could have been used throughout the thesis, making the content of the thesis more uniform.

Chapter 3

The chapter presents the DA-BMC tool that extrapolates suspicious traces by dynamic analysis and then reproduces them with bounded-model checking in order to confirm that they can actually bring to a concurrency error. The approach makes the use of model checking feasible, since the number of states to be explored is limited (those associated with traces identified by dynamic analysis).

Experiments show that the overhead depends on the analyzed program and that the recorded events should be chosen depending on the program (as said at page 34). Is it possible to somehow (by static analysis) automatically suggest to the user which events to record?

Chapter 4

The chapter introduces the ANaConDA framework that provides a platform over which dynamic analyzers for C/C++ programs can be easily built on. First of all, the framework provides a monitoring layer that records important events that can be checked by analyzers. Then, it also provides a noise injector (presented in chapter 5).

The chapter presents interesting insights regarding the problems one faces when developing monitoring applications. I think that the right level of abstraction has been chosen: being more technical would have made the chapter less readable.

The chapter does not report results on the overhead introduced by the monitoring. It briefly says in Sect. 4.5 that the monitored program is 100 time slower, but no results are reported in Sect. 4.6 where some experiments are reported. The conclusions say that ANaConDA "can handle even large real-life programs". My doubt is about the relation between the program size and the introduced overhead: how much does the overhead grow with the program size? Is the relation always linear?

Chapter 5

The chapter describes the noise injection support of ANaConDA. Different noise placement and seeding heuristics are evaluated; moreover, new noise heuristics (as read/write) are proposed and evaluated. A wide set of experiments is performed on 116 C programs implementing a ticket algorithm. Similar experiments are also done on Java programs. Overall the chapter does a very good evaluation of the different heuristics and precisely analyzes when and why one heuristic is better than another one.

I only have some remarks on the content/structure of the chapter.

At page 71, it seems that the evaluation of Java programs is different from the evaluation of C programs. In C programs (Fig. 5.1), the y axis represents the number of test cases where the configurations was among the best 30% of configurations; I guess that "best" is related to the percentage of runs in which the error was detected. In Java (Fig. 5.2), instead, it is sufficient that the configuration is able to detect the error (so, is a run detecting the error sufficient to be reported in the plot?).

The chapter is based on the PADPAT 2012 paper and the STVR journal paper. The contents of the two papers are not properly merged; some concepts are described twice, as the presentation of ANaConDA, presentation of student's programs, etc. Also some experiments are reported twice, as the results reported in Sect. 5.3.2 and Sect. 5.4.1: data of Tables 5.5 and 5.6 are already reported in Table 5.1.

Chapter 6

The chapter compares different approaches (based on the kind of collected information) for monitoring transactional memory programs; such approaches are implemented through a lightweight and a heavyweight manner (the latter in ANaConDA).

I only have some minor remarks on the content of the chapter.

In section 6.3, the results of Table 6.1 are shown in Table 6.2 without ten biggest outliers. In order to better motivate this, I would have reported the variance before and after the removal of outliers.

The text says (page 84) that the decreasing tendency of lightweight monitoring shown in Table 6.2 is also found in Table 6.3; actually, for some programs, as ssca2, kmeans-low, and vacation-low, this tendency is not really visible in Table 6.3 (scca2 even increases) as it is in Table 6.2.

Chapter 7

The chapter presents a basic approach for verifying basic contracts for concurrency using a lockset-based approach. The main contribution of the chapter are the extension of basic contracts with parameters and spoilers, and the technique developed in ANaConDA to verify these contracts. The chapter shows which is the maximum number of events that must be kept in a trace window in order to perform the verification; the conditions that allow the removal of spoilers and targets from the window are correctly proved.

I have only some minor comments.

In Sect. 7.6, examples are still done on Java code (as in the previous sections where experiments were done on Java programs), but then the approach is implemented in ANaConDA for C/C++ programs. I agree that it makes sense to use the same Java code to be more uniform with the presentation of basic contracts; however, I would explicitly say that the contracts do not only apply to Java code.

At page 100, I am not sure whether ρ_3' is correct. If I try to remove X, I would access outside the ArrayList. I was expecting something as X=size() (remove(Y) | set (Y,_) | get(Y)) with Y < X. Is it possible to detect the use of a Y constrained by X?

Finally, I was wondering whether the approach could be adapted/extended to check more general temporal properties for runtime verification. The presented approach checks those traces that match the given contract for concurrency and looks for atomicity violations. I wonder whether it could be possible to reuse parts of the theoretical framework for a different kind of verification, in which general temporal properties are verified (similarly to what is done in JavaMOP), e.g., properties like “if method *next* has been called, method *hasNext* have been called before”.

Prague, March 8th, 2017 Paolo Arcaini