Ph.D. Proposal: Automatic Repair of Loops

Martin Monperrus       Daniel Le Berre

December 4, 2014

If you’d like to apply to this PhD position: please send an email to
martin.monperrus@univ-lille1.fr, leberre@cril.fr

Abstract

This PhD topic is about automatic software repair. Automatic software repair is the process of fixing software bugs automatically. Research on automatic software repair has recently started, esp. since the invention of GenProg, an automatic repair system for C code [3]. We have been successfully contributing to this field [4, 5]. The PhD student will explore how to automatically repair a class of bugs – buggy loops – which has never been considered for automatic repair. The envisioned techniques will use Satisfiability Modulo Theory (SMT) to drive the patch search.

1 Foundations

1.1 Motivation

Software bugs are ubiquitous. They are hidden in source code and show up at development time, testing time or worse, once deployed in production. Except for specific application domains where formal proofs are achievable, bugs can not be eradicated upfront. As an order of magnitude, on Dec 16 2011, the Eclipse bug repository contains 366922 bug reports. Software engineers and developers work on bug repair on a daily basis [7]. Not all developers spend the same time on bug repair. On large companies, it’s sometimes a full-time role to manage bugs (often referred to as Quality Assurance (QA) Software Engineers). Some bugs are analyzed and repaired within minutes, other may take months to be solved [8]. This costs a lot of time and money to all companies involved in the design, implementation and deployment of software.

The motivation of this thesis is to invent techniques for automatically repairing specific bugs so as to save time and money

There is no silver bullet for automatic software repair [6]. In this thesis, we will consider one specific and well-defined class of bugs: buggy loops. The PhD student will explore how to automatically repair a new class of bugs: buggy loops. Buggy loops form a common class of bugs. For instance, the commit 6625fd of the library Apache Commons Lang fixes such an infinite loop.

Our intuition is that we can build on our previous work [1] to build an automatic repair system for loops. NOPOL is able to repair buggy if-conditions using an SMT encoding of the repair problem. Since buggy loops are also related to buggy conditions, we see NOPOL as a good starting point for automatic repair of loops.

The objective of this thesis is to systematically characterize the problem of automatic repair of loops and come up with new SMT-based repair techniques for this class of bugs.
1.2 State-of-the-art

Automatic software repair is the process of fixing software bugs automatically. An automatic software repair system fixes software bugs with no human intervention. The goal of automatic software repair is to save maintenance costs and to enable systems to be more resilient in face of bugs and unexpected situations [6].

The repair may happen at the level of the program code [3] (for instance by changing the conditional expression of an if statement) or at the level of the program state [2] (for instance by replacing a stale unrecoverable object by a fresh one). An automatic software repair system focuses on one particular type of bug, in one particular technical space (e.g. initialization bugs in Java programs). It embeds algorithms and heuristics that leverage knowledge about this kind of bugs: on what the causes are, on how the symptoms manifest themselves, on what the likely fixes look like. The repair may be discovered offline by reproducing field failures, or online, at runtime, when it is not possible to stop the system or to wait for a fix to be found.

To the best of our knowledge, nobody has ever studied the automatic repair of loop bugs.

Our intuition is that we can build on our previous work [1] to build an automatic repair system for loops. NOPOL [1] is a test-suite based repair system. As input, it takes a program and its test suite, given that there is at least one failing test case embodying the bug to be repaired. Then, NOPOL analyzes one by one the statements that are executed by the failing test case in order to identify those for which there may exist a patch. This is what we call “fix point localization”: finding potential fixing locations (as opposed to fault localization which consists of finding potential root causes).

In the process of finding potential fixing locations, NOPOL identifies fix oracles. For example, one automatically identifies that skipping statement compute() would enable the test to pass. Of course, completely removing the statement is not an option, it would break the passing test cases. In other words, one has to synthesized precondition that must output “false” for the failing test case and “true” for the passing ones. The “false” is a fix oracle. Then, NOPOL collects information from test suite execution through code instrumentation. This information is basically the local program state at each fix point. This information contains both “primitive” values (integer, booleans) as well as object-oriented ones (nullness, object state). Next, those runtime traces are transformed into a Satisfiability Modulo Theory (SMT) problem. An SMT solver (Z3 in our case) says whether there exists a solution. If such a solution exists, it is translated back as source code, i.e. a source code patch is generated. We refer to [1] for a complete description of the algorithm.

2 PhD Work

2.1 Research Strategy

To tackle the problem of automatic repair of loops, the student will apply the following research strategy.

Building a Taxonomy of Loop Bugs The student will build a taxonomy of bugs reported in bug repositories that are related to incorrect loops. For this, the student will systematically browse and annotate bug reports. Systematically browsing means using stratified sampling and statistical analysis to extract some of the thousands available bug reports in open-source projects. The main outcome of the empirical study will be a deep understanding of loop bugs: How and why were they reported? How to reproduce them? How were they diagnosed? How were they fixed?

Devising techniques for Repairing Loop Bugs The student will systematically explore automatic repair of loops bugs. The student will focus on three preliminary classes of buggy loops:
• Buggy loops due to of an incorrect loop condition. This encompasses infinite loops and off-by-one loop errors;

• Infinite recursion. This is very common in functional programs and when recursion is used in imperative programming;

• Buggy loops due to of a wrong statement in the loop code. This happens when a statement inside the loop body fails to change the program state (for instance, a statement fails to increase a counter that is used in the loop condition);

Evaluation The proposed techniques will be evaluated on real infinite loops of open-source java programs. For instance, the bug described in commit 6625fd1dd48234ea8f7e2ce28d086d47b21b578 of open-source project Apache Commons Lang is meant to be an evaluation case study.

2.2 Expected Outcome

The first expected outcome is a set of publications in top-tiers conferences and journals in software engineering and programming language: ICSE, FSE, PLDI, etc;

Beyond academic publications, the results of this PhD thesis are directly applicable to help the diagnosis and repair of in-the-fields bugs. It is planned to apply for transfer funding at the end of the PhD in order to rewrite the research prototypes into industry-ready tools. If applicable, this tool can be made open-source to widen its impact.

3 Environment

3.1 Supervision

Martin Monperrus is the principal investigator of this proposal. He will co-advise the PhD student with Daniel Le Berre, expert in constraint solving. They are perfectly complementary for this topic: Martin Monperrus comes from the software engineering field and is expert in automatic software repair. Daniel Le Berre comes from the satisfiability field and is expert in boolean reasoning.

Martin Monperrus has been an associate professor at the University of Lille since 2011 and a researcher at Inria. He was previously with the Darmstadt University of Technology as a research associate. He received a Ph.D. from the University of Rennes in 2008 and a Master's degree from the Compiègne University of Technology in 2004. He publishes in top journals and conferences in software engineering (ICSE, ESEC/FSE, ISSTA, ACM TOSEM). His main research goal is to invent computer programs that repair themselves and is among the top researchers in this field. His main source of inspiration is biological diversity (in immune systems, ecosystems, etc.) which is the key to the astonishing healing and recovering capabilities of biological systems. Selected publications (all available at [http://www.monperrus.net/martin/publications](http://www.monperrus.net/martin/publications)):


Daniel Le Berre is professor of computer science at Artois University in the faculty of science. His main research interests are the formal and practical aspects of boolean reasoning, and its application in Artificial Intelligence and Software Engineering. Most of his work ends up in Sat4j, an open source Java library for solving boolean satisfaction or optimization problems.

Selected publications related to the proposal:

- The International SAT Solver Competitions (Matti Järvisalo, Daniel Le Berre, Olivier Roussel, Laurent Simon), AI Magazine 33(1), 2012
- The Sat4j library, release 2.2 (Daniel Le Berre, Anne Parrain) JSAT 7(2-3): 59-6 (2010)

3.2 Laboratory

The PhD student will be a full-time member in the Spirals research group (Lille, France). Spirals is a joint research group between the INRIA and the University of Lille. Spirals’ main research goal is to introduce more automation in the adaptation mechanisms of software systems, in particular, transitioning from adaptive systems to self-adaptive systems. Spirals especially targets two properties: self-healing and self-optimization. The PhD student will work in the seal-healing part of the group, lead by Martin Monperrus. Martin Monperrus is a full-time member of the Inria’s Spirals research group, and Daniel Le Berre, after having spent one year in the group is now an associate member.

Inria is France’s leading research institutions in computer science. It gathers the biggest number of European projects and ERC grants in that fields. The Spirals research group is a well established group with Inria Lille, with a strong publication record. The PhD student will work together with the ten other PhD student in the group.

3.3 Collaboration

Since Nopol and the envisioned techniques for repairing loops are based on SMT, this PhD thesis is meant to be part of a collaboration with Microsoft Research and the Z3 community.

Furthermore, this PhD is a collaboration between two institutions, since Daniel Le Berre is affiliated with the University of Artois / CRIL and Martin Monperrus with INRIA and the University of Lille. The co-advised PhD is the follow-up of the collaboration started with Nopol [1].

References


