Automating Regression Verification

Dennis Felsing[1], Sarah Grebing[1], Vladimir Klebanov[1], Philipp Rümmer[2], Mattias Ulbrich[1,✉]

[1]Karlsruhe Institute of Technology, Germany
[2]Uppsala University, Sweden
✉ulbrich@kit.edu

Abstract: Regression verification is an approach complementing regression testing with formal verification. The goal is to formally prove that two versions of a program behave either equally or differently in a precisely specified way. We present a novel automatic approach for regression verification that reduces the equivalence of two related imperative integer programs to Horn constraints over uninterpreted predicates. Subsequently, state-of-the-art SMT solvers are used to solve the constraints. We have implemented the approach, and our experiments show non-trivial integer programs that can now be proved equivalent without further user input.

One of the main concerns during software evolution is to prevent *regressions*, i.e., to prevent breaking existing functionality when implementing new features, fixing defects, or during optimization. Undetected regressions can have severe consequences and incur high cost, in particular in late stages of development, or in software that is already deployed. Currently, the main quality assurance measure against regressions is *regression testing*. *Regression verification* is a complementary approach that attempts to achieve the same goal with techniques from formal verification. This means establishing a formal proof of equivalence of two program versions. In its basic form, we are trying to prove that the two versions produce the same output for all inputs. In more sophisticated scenarios, we want to verify that the two versions are equivalent only on some inputs (conditional equivalence) or differ in a formally specified way (relational equivalence). If successful, regression verification offers guaranteed coverage, while not requiring additional expenses to develop and maintain a test suite. Unlike for standard functional verification, one does not need to write and maintain complex specifications (which can be a significant bottleneck in the verification process).

A number of approaches and tools for regression verification exist already, but the majority of them are not automatic and require the user to supply inductive invariants. We present an approach and a tool for *automatic* regression verification of imperative programs with integer variables. We use automatic invariant generation techniques to infer sufficiently strong *coupling predicates* between programs—and thus prove behavior equivalence.

Our approach is targeted towards showing equivalence of programs with complex arithmetic and control flow, a kind of programs poorly supported by existing automatic approaches. It works well whenever sufficiently "simple" coupling predicates over linear arithmetic exist that prove program equivalence, which is often the case in practice. An example is given in Figure 1, which shows two versions of a recursive computation of the

```
int g(int n) {
  int r = 0;

  if (n <= 0) {
    r = 0;
  } else {
    r = g(n-1) + n;
  }
  return r;
}
```

(a) basic version P_1

```
int g(int n, int s) {
  int r = 0;

  if (n <= 0) {
    r = s;
  } else {
    r = g(n-1, n+s);
  }
  return r;
}
```

(b) optimized version P_2

Figure 1: Computing the n-th triangular number

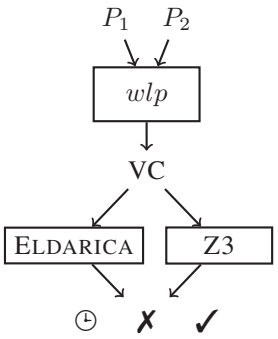

Figure 2: Architecture of our approach

n-th Gaussian triangle number. The original version is in (a), while the improved version in (b) is tail-recursive and can be executed without growing the callstack. To enable this optimization, an accumulator parameter `s` has been added to the signature of `g` for collecting and passing on intermediate results. As a consequence, `g` in P_1 performs summation from the end of the interval, while `g` in P_2 starts from the beginning. Using our method, P_1 and P_2 can be proved to compute the same result fully automatically.

Figure 2 sketches the workflow of our approach: a frontend translates the two programs into efficient logical verification conditions (VC) for program equivalence using a specially tailored weakest liberal precondition calculus (*wlp*). This translation is completely automatic; the user does not have to supply the coupling predicates, loop invariants, or function summaries. The produced VC are in Horn normal form and are passed to an SMT solver for Horn constraints (such as ELDARICA [RHK13] or Z3). If the solver succeeds in finding a solution, the programs are equivalent. Alternatively, the solver may show that no solution exists (i.e., disprove equivalence) or time out.

Our method for automatic regression verification has been implemented in a tool called RÊVE, accessible as a web service at `http://formal.iti.kit.edu/improve/`. The effectiveness of our technique has been proved by applying it to a collection of small but non-trivial benchmarks. A complete account of our method can be found in [FGK+14].

References

[FGK+14] Dennis Felsing, Sarah Grebing, Vladimir Klebanov, Philipp Rümmer, and Mattias Ulbrich. Automating Regression Verification. In *Automated Software Engineering (ASE 2014)*, pages 349–360. ACM, 2014.

[RHK13] Philipp Rümmer, Hossein Hojjat, and Viktor Kuncak. Disjunctive Interpolants for Horn-clause Verification. In *CAV'13*, pages 347–363. Springer-Verlag, 2013.