Minimizing Stainless counter-examples

CS550 Formal Verification, Project Report

Matthieu Bovel (matthieu.bovel@epfl.ch)
Solal Pirelli (solal.pirelli@epfl.ch)

January 10, 2022

Abstract

Stainless is a tool for verifying Scala programs. It can prove that some invariants inside a function hold for all possible inputs, or show counter-examples when they exist.

These synthesized counter-examples are unfortunately often sub-optimal in terms of human-readability. For example, when asked to provide two Java integers whose sum is smaller than 10, Stainless outputs "2'146'959'355 and -2'146'959'347". While correct, the answer "0 and 0" would probably look more intuitive to most human brains.

We address this problem by extending Inox—the engine behind Stainless—to make it able to find models that are not only valid, but also minimal with respect to some sizing constraints. To this end, we leverage the optimizing capabilities of the Z3 backend.

1 Introduction

[Stainless translates to Pure Scala. Then Inox converts to constraints and communique with an SMT solver]

[Brief recall of SMT, DPLL.]

[Example 1: Scala Code, generated constraints, model found]

[Brief recall of Optimizing SMT. Soft constraints, minimize, maximize.] [1]

[Example 2: Optimization with Z3]

[Our work is to extend Inox so that it generates these minimize statements.]

[A note about multiple objectives.]

2 Implementation

[Inox supports multiple backends. Z3 with text or native. On top of that, optimizing versions.]

[Optimizer did not support maximize and minimize. We first added support for that (first PR).]

[Then we added a Minimizer on tp of Z3 text (second PR).]

2.1 Numbers

[What does it mean to minimize a number? Minimize absolute value.]

[Different types of numbers: bit vectors and integers.]

[We generate one expression to minimize per number]

Example:

```
def add(x: Int, y: Int): Int = { x + y } ensuring(res => res >= 10)
```

Generated constraints:

```
...
(declare-fun x!110 () (_ BitVec 32))
(minimize (ite (bvsge x!110 0) x!110 (bvneg x!110)))
(declare-fun y!21 () (_ BitVec 32))
(minimize (ite (bvsge y!21 0) y!21 (bvneg y!21)))
...
```

Result:

2.2 ADT

[Generate one constraint per field recursively.]

[Example]

2.3 Recursion

[Recursive functions and unrolling]

[Loops]

3 Benchmarks

[Quick benchmarks and a nice boxplot with our examples comparing run times between (–solvers=smt-z3-min and –solvers=smt-z3-opt)].

4 Conclusion

[Problems?]

[Next steps?]

[Summary]

References

[1] Robert Nieuwenhuis and Albert Oliveras. On SAT modulo Theories and Optimization Problems. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing*, SAT'06, page 156–169, Berlin, Heidelberg, 2006. Springer-Verlag.