Coq Bug Minimizer

Jason Gross

CoqPL 2015

Abstract

Sales Pitch

Are bugs the bane of your existence? Do you dread Coq upgrades, because they mean you'll have to spend days tracking down subtle failures deep in your developments? Have you ever hit an anomaly that just wouldn't go away, and wished you understood what triggered it? Have you ever been tormented by two blocks of code that looked identical, but behaved differently? Do you wish you could submit more helpful error reports, but don't want to put in the time to construct minimal examples? If you answered "yes" to any of these questions, then the Coq Bug Minimizer is for you! Clone your own copy at `https://github.com/JasonGross/coq-bug-finder`.

What does it do? How do I use it?

The Coq Bug Minimizer is an external script that will take any error-generating Coq file in the middle of a development, and after confirming that it has picked up the right error message, will automatically produce a small, stand-alone file which is guaranteed to generate the same error message.

The transformations automatically performed by the bug minimizer include:

- inlining all of the dependencies of a file
- removing unnecessary definitions, lemmas, hints, tactics, section variables, notations, and other lines
- (optionally) replacing unnecessary proofs and definition bodies with `admit`
- flattening module and section structure
- removing unnecessary imports

The bug minimizer explicitly does not perform the following transformations:

- removing unnecessary definition-specific variables
- removing constructors of inductive types nor fields of records
- simplifying the error-generating proof script itself

The first two are, unfortunately, changes that are often required to happen in multiple places at once; the bug minimizer, currently, only considers changes that can be made in isolation. The third of these is actually a feature; it allows you to check for a particular state and then `fail` with an appropriate message.[1]

Hence, typical interaction will consist of a few rounds of automatic minimization alternated with manual simplification of the final proof script and removal of record fields. In practice, each round of automatic minimization decreases the number of lines by a factor of 5–10, bottoming out at around 50–500 lines.

This sounds amazing! Is it fast, too?

Not yet, unfortunately. Because it re-runs Coq for every change it attempts, it tends to take a few minutes to minimize around a thousand lines of code, a few hours to handle a 5 000 line file, and a few weeks to handle a 40 000 line file.

However, by the time the workshop comes, the Coq Bug Minimizer will hopefully be 100% faster![2]

What's left to do?

Possible future avenues of development include:

- automatic anomaly reporting integrated into CoqIDE
- automatic bug minimization service, where an end user can upload a tarball of their development, and the Coq developers can get a standalone test-case
- smarter heuristics for minimizing test cases
- integration of multiple versions of Coq simultaneously
- using the new `-ideslave` XML protocol rather than the old and error-prone `-emacs` option
- translation from Python into a saner language (like OCaml or Haskell)

What's left to say?

The proposed presentation will consist of a demo of using the Coq Bug Minimizer and a brief explanation of how it works and what scenarios it can be used in. This will be followed by an audience driven presentation or discussion of other user-cases and details of its functioning, possibly including the warts of Coq that workarounds were required for. The presentation will close with a brief period for the audience to make feature requests.

[1] If you have case where `rewrite` used to progress but no longer does, you would want to be able to use `progress rewrite foo` to trigger the error. But if the minimizer was allowed to change anything, it could inform you that `progress idtac` produces the same error message. That is certainly not the test case you were looking for!

[2] How is this possible, you may ask? Currently, the bug minimizer takes time roughly $\mathcal{O}((\text{lines of code})^3)$. By taking advantage of `BackTo`, it should be possible to chop off at least a factor of the number of lines of code.