Problem Sheet 2: AutoProof

Chris Poskitt and Julian Tschannen
ETH Zürich

"Beware of bugs in the above code; I have only proved it correct, not tried it."
– Donald E. Knuth

Starred exercises (∗) are more challenging than the others.

1 Background

This exercise class is concerned with the AutoProof tool [2, 3], a static verifier for programs written in (a subset of) the object-oriented language Eiffel. The tool takes an Eiffel program—annotated with *contracts* (i.e. executable pre-/postconditions, class invariants, intermediate assertions)—and *automatically* attempts to verify the correctness of the program with respect to its contracts.

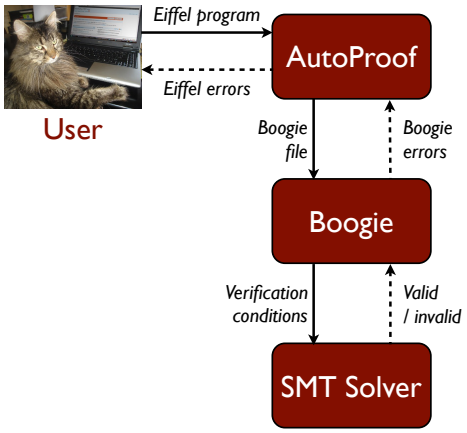

Figure 1: The AutoProof workflow

The tool is built on top of Boogie [1], an automatic verification framework developed by Microsoft Research. AutoProof translates Eiffel programs and their contracts (i.e. their proof obligations) into the front-end language of Boogie—an *intermediate verification language* encoding the semantics of the source program in terms of primitive constructs, and prescribing what it means for the source program to be correct. The Boogie tool then translates this intermediate program into a set of *verification conditions*; logical formulae which if valid, indicate the correctness of the source program. The validity of these verification conditions is checked automatically by an SMT solver (currently Z3).

This workflow is summarised in Figure 1. We will only be interacting with AutoProof itself in this exercise class, but it is helpful to be roughly aware of how it works and what translations it is performing (in a later class, we will look at the Boogie framework directly).

2 Exercises

The Eiffel programs for the following exercises are all available online in a web-based interface to AutoProof. Simply follow the given links for each exercise, edit the code and contracts in your browser, then hit the "Verify" button to run the tool.

Important: the web interface to AutoProof does not permanently save your changes, so please make sure to save local copies of your solutions.

i. Consider the class `WRAPPING_COUNTER` in:

http://cloudstudio.ethz.ch/e4pubs/#sv-task1

The method `increment` increases its integer input by one, *except* if the input is 59, in which case it wraps it round to 0. Verify the class in AutoProof *without* changing the implementation, i.e. adding only the necessary preconditions. Strengthen the postcondition further as suggested in the comments, and check that the proof still goes through.

ii. In the axiomatic semantics problem sheet, we encountered several simple program specifications expressed as Hoare triples. Using the class `AXIOMATIC_SEMANTICS` in:

http://cloudstudio.ethz.ch/e4pubs/#sv-task2

write some simple contract-equipped methods and show the following in AutoProof:

(A) $\models \{x = 21 \wedge y = 5\}$ `skip` $\{y = 5\}$
(B) $\models \{x > 10\}$ `x` $:=$ `2` $*$ `x` $\{x > 21\}$
(C) $\models \{x \geq 0 \wedge y > 1\}$ `while x < y do x :=` `x` $*$ `x` $\{x \geq y\}$
(D) $\models \{x = 5\}$ `while x > 0 do x :=` `x` $+$ `1` $\{x < 0\}$
(E) $\models \{x = a \wedge y = b\}$ `t := x; x :=` `x` $+$ `y; y := t` $\{x = a + b \wedge y = a\}$
(F) $\models \{in + m = 250\}$ `while (i > 0) do m :=` `m` $+$ `n; i :=` `i` $-$ `1` $\{in + m = 250\}$

Hint: Eiffel does not offer a while construct. Try experimenting with from-until-loop instead, as well as if-then-else with recursion (note that recursive calls should be surrounded by `wrap` and `unwrap` so that the verifier checks the class invariant—see the code comments).

iii. Consider the class `MAX_IN_ARRAY` in:

http://cloudstudio.ethz.ch/e4pubs/#sv-task3

What does the `max_in_array` method do? Prove the class correct in AutoProof by determining a suitable precondition and loop invariant.

Hint: you might find Eiffel's across-as-all loop construct[1] helpful for expressing loop invariants.

[1] See: http://bertrandmeyer.com/2010/01/26/more-expressive-loops-for-eiffel/

iv. (∗) Consider the class `SUM_AND_MAX` in:

`http://cloudstudio.ethz.ch/e4pubs/#sv-task4`

What does the method `sum_and_max` do? What can you prove about it using AutoProof?

v. (∗∗) Consider the class `LCP` in:

`http://cloudstudio.ethz.ch/e4pubs/#sv-task5`

The method `lcp` implements a Longest Common Prefix (LCP) algorithm[2] with input and output as follows:

> *Input:* an integer array a, and two indices x and y into this array.
>
> *Output:* length of the longest common prefix of the subarrays of a starting at x and y respectively.

What can you prove about the class in AutoProof?

References

[1] K. Rustan M. Leino. This is Boogie 2. Technical report, 2008. http://research.microsoft.com/en-us/um/people/leino/papers/krml178.pdf.

[2] Julian Tschannen, Carlo A. Furia, Martin Nordio, and Bertrand Meyer. Automatic verification of advanced object-oriented features: The AutoProof approach. In *Tools for Practical Software Verification - LASER 2011, International Summer School*, volume 7682 of *LNCS*, pages 134–156. Springer, 2012. http://se.inf.ethz.ch/people/tschannen/publications/TschannenLASER11.pdf.

[3] Julian Tschannen, Carlo A. Furia, Martin Nordio, and Bertrand Meyer. Program checking with less hassle. In *Proc. Verified Software: Theories, Tools, Experiments (VSTTE 2013)*, volume 8164 of *LNCS*, pages 149–169. Springer, 2014. http://se.inf.ethz.ch/people/tschannen/publications/tfnm-vstte13.pdf.

[2] From the FM 2012 verification challenge.