Requirements for Hoare Advanced Homework Assistant

Tadeusz Sznuk Aleksy Schubert Jacek Chrząszcz

Abstract

The paper collects requirements for HAHA.

Introduction

The current situation in teaching formal aspects of computer programming is notoriously difficult mainly because students perceive it as tiresome and obscure. In particular, a proof of program correctness with help of Hoare logic requires writing annotations that have to be carefully checked afterwards. This process is not very interesting and is perceived to be as error-prone as writing the program in the first place. That is why vast majority of students are not motivated to learn program correctness proving techniques and testing remains their only technique of quality assurance.

The goal of the HAHA tool that we want to create is to ease the task of verifying that the annotations in the program are sufficient to prove its correctness according to the Hoare logic rules. Therefore students will concentrate on writing the annotations and by having an immediate feedback that their work is not (yet) correct, possibly with some explanation, will be motivated to improve the formulae and try again until the annotated program passes the verification.

The consequence of having an automated teaching tool to develop proofs of programs is the possibility to do many more exercises in the same amount of time, with much better understanding and bigger satisfaction. Consequently, one may expect better student motivation and therefore better teaching results.

Due to tediousness of application of the Hoare logic rules, currently one never does a full Hoare logic derivation. Most rules, like assignment rule or conditional rules are applied mostly intuitively and only the most difficult ones, like loop rule, are applied more rigorously. Having an automated tool would bring back the rigour of all Hoare rules.

Use cases

There are two natural categories of users which might be interested in our tool: [#Student students] and [#Tutor tutors].

Student

Does her homework

Traditionally, homework assignments take the form of a simple program to which students must add assertions and invariants. Typing the solution in an IDE is more convenient than writing it by hand and the result is more readable. Besides, a successfully validated solution is certainly at least formally correct.

The main use case for working on a homework assignment is as follows:

1. The student loads the assignment into HAHA.
2. The student tests the program with example data to understand how it works.
3. The student enters / corrects formulae.
4. The student immediately sees syntax errors.
5. The student validates the annotated program.
6. The student sees the validation result, in particular is informed immediately in case of errors.
7. The student analyses the errors and adjusts the formulae (repeats steps 3-7).
8. The student creates a package with solution to send out to a tutor.

Optionally, fragments of lecture notes can be embedded in HAHA (e.g. explanation of rules) and can be a quick reference if the student gets stuck during a proof. The use case could be as follows:

1. The student right-clicks on an instruction in a program
2. The student selects "Show Hoare rule" from the context menu
3. The student can see the appropriate Hoare rule, possibly with its application to the particular instruction in the program.

Prepares for an exam

In preparation for an exam a student should be able to review the material in the most convenient way as well as practice her abilities to solve proving problems on excerpts from the course notes, examples given by the tutor, and her own ones. The latter might be result of (a sequence of) modifications of tutor's examples, giving the space for the exploration of the subject and genuine interest of a student in the studied matters. The expected use cases are variations of the homework use case:

- **The student experiments to improve one's understanding of Hoare logic:**
  - modifies some parts of the program and tries to adapt the logical formulae,
  - modifies (e.g. simplifies) logical formulae without breaking the proof of program correctness.
- **The student practices by solving exercises form course notes:**
  - completes the given examples where some of the formulae are already given

Tutor

Prepares exam/homework

The tutor should have a possibility to prepare an assignment in HAHA. Such assignments can be then used for exams or for homeworks. Preparing the assignment on paper requires special care in

- typesetting of the program to look in a readable way,
- checking all the details of instructions and formulae that they match ones with the others.

The tutor should be able to follow the scenario

1. The tutor loads an old assignment.
2. The tutor modifies the program in the assignment.
3. The tutor modifies the description of the assignment.
4. The tutor types in the assertions in HAHA and verifies if they are satisfied.
5. The tutor modifies the program and description of assignment to make it work.
6. The tutor works in the points 2-6 above until she/he is satisfied.
7. The tutor saves the assignment in the solved version and exports the unsolved version for students.

The unsolved version of the assignment available for students is given in a file of the special form which contains the necessary information. It should be, however, possible to generate PDF version of the assignment so that it can be handed out.

To achieve this the format of the files should make possible to represent programs, Hoare-like annotations, and descriptions. The Hoare-like annotations and descriptions should be located in special structured comments that will make possible to sort out which parts are Hoare assertions, which are assignment descriptions, and which are just local comments available only to the tutor.

The tutor should also have the possibility to mark which Hoare-like annotations are exported to the version available for students and which should be hidden from them.

Checks submissions

Verifying proof correctness by hand is tedious and error-prone. Helping the tutor in this task by a tool is very desirable. Automatic verification allows the tutor to focus on other aspects of students' solutions (e.g. style). The HAHA tool should therefore help in checking students' submissions in the following aspects:

- the submission is a solution of the original task,
- the submission is a program with correct annotations.

Moreover, checking of a number of solutions should be done as a batch job resulting in a summary report. As soon as the batch is automatically checked, the tutor should be able to open each submission individually and evaluate and comment it for the student. The sequence of actions could be as follows:

1. The tutor prepares the directory with the original task and students' solutions.
2. The tutor executes the checking batch job.
3. The tutor reads the report.
4. The tutor opens one submission in the IDE.
5. The tutor evaluates the submission, places some comments in a dedicated location in GUI.
6. The tutor gives the mark to the submission.
7. The tutor marks the evaluation as finished.
8. The tutor repeats steps 3 to 7 until all students' solutions are evaluated.
9. The tutor transmits the comments and marks to the students.

The steps 4 and 5 are essential and do not exceed the requirements resulting from the usual student use case.

Some management of the group of solutions - which one is already evaluated and which one is not - is optional (steps 6 and 7).

Launching the batch checking job and presenting the summary report is another optional requirement (steps 1 and 2), which could be integrated with steps 6 and 7 above.

Transmitting the evaluation results to students (either through email or through systems such as moodle or USOS) is another optional requirement.

Creates slides or similar content

Tutor or any other person who works with formal methods should be able to generate some material to present her ideas associated with program verification. Modern editors support code highlighting. HAHA should support the making of presentations with these capabilities and provide means to edit content that is not directly interpreted as a program, but can be exported to some format in which program presentation can be done. Therefore, the tool

- should make possible to edit non-code parts of the file,
- should make possible to distinguish code parts of the file,
- should make possible to mark which is the expected export format (HTML and LaTeX should be supported).

The HTML export should be possible in two ways: as a whole webpage (with all styles and auxiliary files) and as a fragment to be included on a webpage which already has all necessary styles and files. The sequence of steps could be as follows:

1. The tutor opens a file with some proof of a program.
2. The tutor selects some part of the annotated program.
3. The tutor chooses "export" form the context menu (or main menu).
4. The tutor chooses the export format: whole web-page, HTML fragment, TeX fragment, etc.

5. The tutor chooses a filename for the exported elements.
6. The export operation is performed.

An important feature of the export operation is that it should be possible to generate snippets of exported files which can be included in other documents created by the tutor. This means that the exported code should be readable for humans so that it is possible to easily adapt its form.

Customizes the tool

We assume that the tool can be used in many academic environments. Each of the environments will surely have its own minor requirements for the presentation, for the syntax or for the semantics of the programs that are available in HAHA. The tutor should be able to adapt with lesser or greater effort the tool to her own needs. Therefore it should be relatively easy to adapt the tool to many requirements. In particular, it should be possible to work in a different variant of Hoare logic or to specify a new one.

The adaptability should concern:

- the syntax of the programming language and annotations,
- the visibility of identifiers,
- the way procedure parameters are handled,
- the way semantics of the language is defined.

The adaptation process in most cases will require the tutor to program parts of the tool. However, it should be clear enough where the programming should take place to obtain a particular result. This should rather be handled by a tutorial which shows the tutor which programming task should be accomplished to obtain a particular effect.

Requirements

This section groups requirements that result from the analysis of the use cases presented in the previous section. The requirements are divided according to their importance into obligatory, optional, and visionary ones. The first ones are considered to be essential for the tool to be useful. Implementation of the other ones will make the tool more convenient to use.

Obligatory requirements

This subsection presents the minimal set of requirements that are necessary to make HAHA a convenient tool to work with while-programs, their specifications and verification using Hoare logic.

Programming language support

The programming language that should be taken into consideration is a classic language of while-programs over integers. The programs are built from integer constants and variables using assignment, conditional, while loop and sequence. For convenience the code snippets of the program are grouped in procedures.

The requirements are listed below:

- Classical while-programs.
- Syntax similar to Pascal.
- Grouping construct to form procedures.
- **Data structures:**
  - exact integers,
  - 32-bit signed integers with two's complement arithmetic,
  - arrays of base types (no aliasing).

Logical language support

The logical language must be rich enough to express most of the interesting properties that one would like to say about a piece of program. The basic blocks are first order formulae using variables and constants, they constitute elements of annotations. The requirements are listed below:

- Assertions (including postconditions).
- Code preconditions.
- Loop invariants.
- Loop decreases formulae.
- First order formulae in a format similar to boolean expressions in the language.

Development support

The goal of the HAHA tool is to help users with writing programs, annotating and verifying them. The first step to do so is to present the written program, as well as responses from the system in the clearest possible way. The list of specific requirements follows:

- Syntax highlighting for program and logic part.
- **Clear reporting of syntax errors:**
  - Error markers,
  - Clear error messages.
- Completion of identifiers and keywords (Autocompletion / IntelliSense).
- Possibility of exporting coloured code to HTML or LaTeX.
- Integration with help for Hoare logic.

Interpreter

As HAHA is primarily targeted at students, it is important to let them test their code before or during the verification by Hoare logic. In order to do so an interpreter is needed that is able to run the program and display the result. The program operates on states so the environment should make available to a user the possibility:

- to define an initial state,
- to inspect the final state.

Verification

The main requirement for HAHA is the integration of a verification machinery with the program writing process. It consists of a verification conditions generator (VCgen) and an automatic solver to discharge the generated conditions. To enhance usability, a support should be provided to deal with unsuccessful verification runs. In particular location of errors should be easy to find, unproved verification conditions should be marked and made available for view and it should be relatively easy to understand where each part of a formula came from in the code. The list of specific requirements follows:

- Integrated verification conditions generator.
- Verification conditions are discharged automatically with an SMT solver.
- Verification errors are reported in a manner similar to syntax errors.
- Generated verification conditions are viewable (e.g. can be saved to a file).
- It should easy to trace back the verification conditions to their original location.

Customisation

The tool will make available a basic customisation support. However most of the customisation tasks will require a code development. A manual describing the ways of modifying and extending the tool should be created and maintained.

Additional requirements

This subsection describes features which are not essential for usability of the tool. Some of these might not be implemented in initial releases of HAHA.

Development support

In program development many tasks may and should be automated. In HAHA this concerns not only traditional development activities such as variable renaming, but also concerns activities pertinent to specifications.

- **Refactoring:**
  - Rename variable in code and in specifications.
  - Rename procedure name in code and in specifications.
  - Add a parameter to a procedure.
  - Remove a parameter from a procedure.
  - Reorder parameters in the procedure.
- Generate weakest precondition/strongest postcondition based upon the formulae that are already typed in.
- Warn the user about suspicious programming constructs, e.g. such as the use of "=" in expressions of languages such as Java or C.
- Help system with Hoare logic rule presentation together with the way the rule is applied in a particular situation.

Tutor support

The extensive automation of the tutor's work needs support in many activities. The main situations that require additional functionality are:

- Extracting a program without (some) annotations.
- Batch operation for a number of programs to verify.
- Management of a group of programs (marking them with tags such as todo / done).
- Comments and marks for programs.
- Sending out comments.

Command-line interface

Some of the tool functionality can be made available with lightweight command-line tools. The automated tasks should include:

- Syntactic checking the correctness of annotations in program.
- Checking of the assertions with help of Hoare logic.

Internationalisation

We intend HAHA to be usable in different academic institutions around the world. While a tool with English-only user interface would probably be acceptable in most cases, we believe that providing students with support for their native languages would be beneficial.

- Support for easy adding new translations.
- Internationalisation mechanism should support advanced features, such as pluralisation.

Debugging

Ability to pause a program and inspect its state during execution is very useful in code development. This is particularly true when one is attempting to solve an assignment and needs to gain an understanding of the inner workings of an algorithm provided by a tutor. Some form of debugging support, similar to that provided by modern development environments for other languages, should therefore be implemented in HAHA. Said support should consist of the following features:

- Ability to set breakpoints.
- Single-stepping.
- Visualisation of local values.

Additional programming language support

The language of while-programs for Hoare logic is far from the languages that are used in real world. Still one may want to teach specification and verification of programs with particular features. Therefore we may desire to extend the basic format of programs with additional features. The desirable features to begin with are:

- Global variables.
- Recursive calls.
- Different modes of procedure calling (call-by-variable, call-by-reference etc.)
- Exceptions.
- Memory allocation and deallocation constructs.
- Pointer arithmetic.

Additional logical language constructs

The language of specifications can be extended to make possible to express more complicated properties of programs or to express some of the properties in a more convenient way. The desirable additional constructs to make available in HAHA are:

- Ghost variables.
- The possibility to refer to a initial state (similar to old JML construct).
- Sigma-notation for sums of numbers.
- The formulae to express if a particular piece of code is terminating.
- The formulae to express the termination measure for the particular code.
- The formulae to express which procedures may be called from a particular procedure.

The extended programming language support requires more refined ways to specify program properties. The most desirable kinds of formulae associated with additional language constructs are:

- The formulae to express which data is modified in a procedure.
- Exceptional procedure postconditions.
- The formulae to express which data is read in a procedure.
- The formulae to express which references can be captured in the method.

Visionary requirements

In this section we group additional requirements that need some research to be effectively implemented.

Rule definitions

It is desirable to have a single place which defines both Hoare logic procedures and the way the program is interpreted. Originally this is defined in programming language semantics, but it is not obvious which is the relevant way to represent programming language semantics so that it makes possible to define all the bits of HAHA environment.

Web interface

The tool can be made available not only as a standalone application, but also as an application through web interface. Even a partial web interface would be very useful for educational websites (e.g. http://wazniak.mimuw.edu.pl). However, this requires at least good Java support for the necessary GUI operations (e.g. AJAX-based one).

Choice of Hoare logic variants

Once a few variants of Hoare logic are available, a tutor may want to easily switch from one variant to another. The possible sequence of steps leading to a change in the logic and programming language is as follows:

1. The tutor opens HAHA preferences dialog.
2. The tutor can browse through available variants of Hoare logic.
3. The tutor selects the needed variant.
4. Consecutive verifications are performed using the selected variant.