Improving a Deductive Program Verifier for Hyperproperties

Paul Winkler

Supervisors: Anqi Li, Thibault Dardinier and Prof. Dr. Peter Müller
Group: Programming Methodology Group, ETH Zurich

1 Introduction

Defined as design defects in an engineering system that cause unwanted behavior, bugs are a major problem in engineering disciplines. Their occurrence can be prevented through formal verification, the process of establishing mathematical guarantees about programs under certain specifications [5].

In verification, one is often interested in properties that relate multiple executions of a program, the so-called hyperproperties [1]. They can be used to express a wide variety of useful properties, for example, functional and security properties. In practice, there are four different types: $\forall^*$-, $\exists^*$-, $\forall^*\exists^*$- and $\exists^*\forall^*$-hyperproperties. The $\forall^*$-properties relate all combinations of executions of the same program and *overapproximate* the set of possible executions. Further, $\exists^*$-hyperproperties require proving the existence of executions, therefore *underapproximating* the set [2]. The remaining are combinations of previous properties. In order to prove them, many powerful logics have been developed, including RHLE and Hyper Hoare logic (HHL) [4, 3]. However, RHLE is limited because it cannot prove $\exists^*\forall^*$-hyperproperties, and many logics are alike. Such properties are quite relevant, as the example of the existence of a minimum (i.e. there exists an execution such that its output is smaller than or equal to the outputs of all other executions) shows.

Recently developed HHL is able to prove and disprove arbitrary hyperproperties, including all aforementioned types, and is the basis of our work. It is a generalization of Hoare logic which lifts assertions to properties of arbitrary *sets* of states. Similarly to its predecessor, HHL verifies a program C with respect to its precondition P and postcondition Q by proving the validity of the corresponding hyper-triple *[P] C [Q]*. The triple is defined to be valid if and only if executing C in any set of initial states satisfying P leads to a set of final states satisfying Q.

Automation is crucial for logic to be usable in a general setting, which is usually achieved using a deductive program verifier (verifier). In their recent paper, Dardinier et al. [2] present a new verifier called Hypra, which is based on HHL and extends it. It is the first deductive verifier to support all the aforementioned types of hyperproperties and to mix different types of hyperproperties in proofs. Additionally, it is capable of reasoning about runtime errors, allowing us to prove the correctness and incorrectness of a program. The key idea is to encode hyperproperties and the corresponding proof rules in Viper, a standard intermediate verification language. This encoding explicitly represents the sets of states of the input program in the sets of states in Viper. It then tracks simultaneously a lower and upper bound of the sets of states that can be reached by executing the input program in an arbitrary initial state.

However, Hypra has certain limitations. One major drawback is the generation of precise error messages, as they are automatically generated by the intermediate verifier and are not designed to provide feedback about the original program. Another limitation is the availability of various types. The current tool only supports integer-typed variables, but in order to prove a greater variety of programs, more must be supported. Advancements in these areas would increase the usability of the tool. Additionally, alternative encodings of states in Viper should be considered to improve Hypra's performance. This will be investigated within the scope of this thesis.

2 Core Goals

These core goals will be addressed as part of this thesis. They consist of a short description and motivation.

2.1 Improvement of error messages

As mentioned above, error handling is currently done by forwarding feedback from the intermediate verifier, namely Viper. Due to the increased complexity of the code in the translation process, the generated error messages often seem obscure and disconnected from the original program. Tracing back the issue to a certain element of the verification is difficult for the user. The goal is to implement the generation of error messages that contain information about the cause, location in the original program.

2.2 Support of new data types

Currently, Hypra's input language only supports integer-typed variables and the usual arithmetic operations, which practically restricts the number of possible programs it can verify. Therefore, the following types will be added in the scope of this section:

- Primitive Types: Boolean
- Type Combinators: finite sequences, sets and maps

Our current intermediate verifier, Viper, offers support for these types. The main challenge is to adapt the encoding of states to accommodate novel types. After determining this theoretically, Hypra will then be modified to support these types.

2.3 Evaluation of newly supported benchmarks

The introduction of new types will allow Hypra to support a larger number of programs. Therefore, its performance in handling these new cases must be assessed. In the scope of the original paper, a number of different benchmarks which were used to evaluate alternative verifiers, such as Descartes or ORHLE, were translated and used to evaluate Hypra. Due to the limitations mentioned above, not all available benchmarks could be translated. In this section, we will adapt these for execution in Hypra and extend the evaluation from the original paper.

3 Extension Goals

The extension goals offer possible topics for further investigation. Depending on the complexity of the previous core topics, some or all of them will be explored.

3.1 Support of custom types

In addition to the aforementioned types, Hypra should also be able to handle custom types. They would enable the definition of new user-defined types, which can be axiomatized, further broadening Hypra's use cases. This will be investigated. If the thesis analysis concludes that it can be realized with an appropriate amount of effort, it will also be implemented in Hypra.

3.2 Exploration of alternative encodings of states

As defined in the current encoding, the program states are represented by a custom axiomatized interface, a so-called Viper domain. The used axioms are quite complicated and alternatives, for example encoding states using Viper's map type, may improve Hypra's performance. Therefore, alternative finite representations of states will be investigated. If a suitable alternative is found, the performance using the modified encodings will be evaluated.

3.3 Development of a code editor extension

Currently, Hypra files lack syntax highlighting and other features that modern IDEs can provide. Verifying a file is done via the command line. This could be improved by developing an extension for common code editors, as has been done for Viper.

References

[1] Michael R Clarkson and Fred B Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[2] Thibault Dardinier, Anqi Li, and Peter Müller. Hypra: A deductive program verifier for hyper hoare logic. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2):1279–1308, 2024.

[3] Thibault Dardinier and Peter Müller. Hyper hoare logic: (dis-)proving program hyperproperties. *Proc. ACM Program. Lang.*, 8(PLDI):1485–1509, 2024.

[4] Robert Dickerson, Qianchuan Ye, Michael K Zhang, and Benjamin Delaware. Rhle: modular deductive verification of relational properties. In *Asian Symposium on Programming Languages and Systems*, pages 67–87. Springer, 2022.

[5] Osman Hasan and Sofiene Tahar. Formal verification methods. In *Encyclopedia of Information Science and Technology, Third Edition*, pages 7162–7170. IGI global, 2015.