Automation of Hyper Hoare Logic

Master's Thesis Project Description

Anqi Li

Supervised by Thibault Dardinier and Prof. Dr. Peter Müller
Department of Computer Science, ETH Zürich

March 2023

1 Introduction

Program verification is the process of formally proving that a program behaves according to its specification. One proof system widely used in program verification is Hoare Logic (HL) [1]. It can prove properties of individual program executions, namely trace properties, by using Hoare triples of the form $\{P\}C\{Q\}$, where C is a program command and P, Q are predicates describing the initial and final program states respectively. A Hoare triple $\{P\}C\{Q\}$ holds if and only if executing C in an initial state satisfying P can only result in final states satisfying Q.

However, classical HL cannot prove hyperproperties [2], i.e. properties of multiple program executions, such as determinism. As a result, several extensions and adaptations of HL have been suggested to overcome this limitation. For instance, Sousa and Dillig [3] have proposed Cartesian Hoare Logic (CHL) that can reason over k executions of the same program, where k is some fixed integer ≥ 1. This is useful for verifying properties like associativity and monotonicity. Similar to classical HL, CHL can only prove the absence of bugs in computer programs because it overapproximates the possible program executions. To prove the existence of bugs in a program, O'Hearn [4] has proposed Incorrectness Logic (IL) which underapproximates the possible program executions [5]. Hoare-style logics combining overapproximation and underapproximation have also been proposed, such as Outcome Logic (OL) [6] and RHLE [7]. Nevertheless, each of the existing logics has its own limitations regarding the program properties that they can prove (including trace properties and hyperproperties) and the type of reasoning that they can perform (including overapproximation and underapproximation). For example, RHLE can prove both the existence and absence of bugs while relating multiple executions of the same program, and it can also prove $\forall\exists$-hyperproperties (i.e. the existence of an execution that satisfies some condition with respect to all possible executions), but, like any other existing logics, it cannot prove $\exists\forall$-hyperproperties (i.e.

the existence of executions with respect to all other executions) such as the existence of a minimum returned value. Consequently, verifying a program that involves various properties requires the application of multiple logics. This motivates the proposal of Hyper Hoare Logic (HHL) [8] – a logic that can prove and disprove all aforementioned program properties and more.

Unlike existing logics where assertions are predicates over a fixed number of program states, HHL assertions are predicates over arbitrary sets of states. Analogous to Hoare triples, HHL uses hyper-triples of the form $[P]C[Q]$, where C is a program command and P, Q are predicates describing the initial and final sets of program states respectively. A hyper-triple $[P]C[Q]$ is valid if and only if executing C in any initial set of states satisfying P leads to a final set of terminating states satisfying Q.

For the time being, HHL is a purely theoretical achievement. Its theoretical details have been comprehensively presented in the paper authored by Dardinier and Müller [8]. This project aims at implementing a prototype that automates HHL by developing an encoding into Viper [9] – a program verification infrastructure. This general goal is divided into multiple core goals and extension goals in the following sections.

2 Core Goals

2.1 Overapproximation Reasoning

The first step is to define an imperative programming language, similar to IMP, that includes at least the following commands. Note that `C` is a program command, `x` is a variable, `e` is an expression, and `b` is a predicate describing program states.

- `skip`
- `x := e` (assignment)
- `x := nonDet()` (non-deterministic assignment)
- `assume b`
- `assert b`
- `C; C` (sequential composition)
- `while(b){C}`
- `if(b){C} else{C}`

After the language is defined, the real challenge is to design an encoding into Viper to enable reasoning with ∀-HHL, which we define as the strict subset of HHL that involves overapproximation reasoning over multiple executions of the same program. The encoding should then be tested on small program verification examples retrieved from related literature. During this process, refinement should be applied to the encoding if necessary.

2.2 Verifier Implementation

Next, we need to build a verifier that translates a program written in our language into a Viper program to automatically generate the encoding from Section 2.1. The verifier should have scanning, parsing and code generation phases. A type checking phase is preferable but not mandatory. At this stage, we should also define the assertion

language for ∀-HHL and enable the verifier to translate any assertion written in this language.

2.3 Underapproximation Reasoning

Another important goal of the project is to automate reasoning with ∃-HHL, which we define as the strict subset of HHL that involves underapproximation reasoning over multiple executions of the same program. Similarly, an encoding from our language into Viper should also be designed for ∃-HHL, and the verifier from Section 2.2 should be updated to support ∃-HHL. Moreover, since underapproximation reasoning uses existential quantification, which is hard for SMT-based verifiers like Viper to automatically prove, we will explore the idea of enabling users to provide hints that can aid the verification process.

2.4 Evaluation

After ∀-HHL and/or ∃-HHL have been automated, we should evaluate our tool with respect to the tools implementing existing Hoare-style logics mentioned in Section 1, such as Descartes [3] and ORHLE [7]. The evaluation should show whether our tool can verify the programs verifiable by other tools and vice versa. Performance of the tools can be included in the evaluation but is not mandatory.

3 Extension Goals

3.1 ∀∃-HHL and ∃∀-HHL

Ideally, we would like the prototype to be able to establish ∀∃-hyperproperties and ∃∀-hyperproperties. To do so, we need to design an encoding into Viper, update the verifier, and also test the implementation on examples. This should build on the foundations of automation for ∀-HHL and ∃-HHL.

3.2 Method Calls

Method calls should be handled modularly. However, since assertions in HHL are predicates over sets of states, we need to explore how method specifications, i.e. preconditions and postconditions, should be written so that such modularity is achieved.

3.3 Comprehensions

There are hyperproperties that require comprehensions, such as counting and summation, rather than quantification over the program states. For example, one hyperproperty that requires counting is the existence of n different outputs for any given input. HHL is able to prove such properties theoretically, but the automation for proving such properties remains to be explored.

References

[1] Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969) https://doi.org/10.1145/363235.363259

[2] Clarkson, M.R., Schneider, F.B.: Hyperproperties. In: 2008 21st IEEE Computer Security Foundations Symposium, pp. 51–65 (2008). https://doi.org/10.1109/CSF.2008.7

[3] Sousa, M., Dillig, I.: Cartesian hoare logic for verifying k-safety properties. SIGPLAN Not. **51**(6), 57–69 (2016) https://doi.org/10.1145/2980983.2908092

[4] O'Hearn, P.W.: Incorrectness logic. Proc. ACM Program. Lang. **4**(POPL) (2019) https://doi.org/10.1145/3371078

[5] Vries, E., Koutavas, V.: Reverse hoare logic. In: Barthe, G., Pardo, A., Schneider, G. (eds.) Software Engineering and Formal Methods, pp. 155–171. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24690-6_12

[6] Zilberstein, N., Dreyer, D., Silva, A.: Outcome logic: A unifying foundation for correctness and incorrectness reasoning. ArXiv **abs/2303.03111** (2023) https://doi.org/10.48550/arXiv.2303.03111

[7] Dickerson, R., Ye, Q., Zhang, M.K., Delaware, B.: Rhle: Modular deductive verification of relational ∀∃ properties. In: Sergey, I. (ed.) Programming Languages and Systems, pp. 67–87. Springer, Cham (2022)

[8] Dardinier, T., Müller, P.: Hyper hoare logic: (dis-)proving program hyperproperties (extended version). ArXiv **abs/2301.10037** (2023) https://doi.org/10.48550/arXiv.2301.10037 arXiv:2301.10037

[9] Müller, P., Schwerhoff, M., Summers, A.J.: Viper: A verification infrastructure for permission-based reasoning. In: Verification, Model Checking, and Abstract Interpretation, pp. 41–62. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49122-5_2