Relational Hyper Hoare Logic

Master's Thesis Project Description

Matej Martinček

Supervisors: Dr. Thibault Dardinier, Hongyi Ling, Prof. Dr. Peter Müller

November 2025

1 Introduction

Traditional Hoare logic [1] reasons about the properties of individual program executions. This is insufficient in practice when trying to prove program properties such as determinism (any two executions of a program with the same initial state end up in the same final state) or non-interference [2] (any two executions of a program with the same initial low-sensitivity inputs end up with the same final low-sensitivity outputs). Thus, in recent years, there has been an effort to explore properties relating multiple executions of a program. The name for these kinds of properties was coined as *hyperproperties* [3]. In cases where the properties refer to the executions of potentially different programs, we refer to them as *relational* properties.

Logic for Hyper-triple Composition (LHC) [4] extends previous work [5], [6] concerned with proving hyperproperties relating potentially different program executions[1], where the hyperproperties are of the $\forall^*$ type[2]. The main achievement of LHC is the introduction of new compositionality rules that allow the logic to compose hyper-triples[3] in new ways, which facilitates simpler proofs.

Hyper Hoare Logic (or HHL) [7] allows for reasoning about arbitrarily quantified hyperproperties, including the $\forall^*\exists^*$, and $\exists^*\forall^*$ types. This allows for not only proving but also disproving hyperproperties (e.g., disproving the $\forall^*$ type of hyperproperties by proving the $\exists^*$ type of hyperproperties). The main limitation of HHL, however, is that it allows reasoning about (potentially multiple) executions of only a single program[4], which limits its usability. For example, it is not possible to express properties such as program refinement (to determine whether one program refines or preserves the behavior of another).

Not only can HHL not prove relational properties, but also HHL's hyperproperty proving has its limitations, which we explore through an example in section 2.

$$\frac{\vdash \{P\}\ C_1\ \{Q_1\} \qquad \vdash \{P\}\ C_2\ \{Q_2\}}{\vdash \{P\}\ \mathbf{if}\ (*)\ \{C_1\}\ \mathbf{else}\ \{C_2\}\ \{Q_1 \otimes Q_2\}}\ (\textsc{Choice})$$

Figure 1: HHL's Choice rule.

[1] So, the work is concerned with proving both hyperproperties and relational properties.

[2] These are hyperproperties that can be expressed using only universal quantification over program traces.

[3] These are Hoare triples in the context of hyperproperty proving.

[4] So, it cannot reason about relational properties.

2 From hyperproperties to relational properties

Consider proving that the program in Fig. 2 behaves as a monotonic function of the variable x. We start with a set of initial states that satisfy the precondition:
$\{\forall\langle\sigma_1\rangle\,\forall\langle\sigma_2\rangle.\ \sigma_1(t) = 1 \land \sigma_2(t) = 2 \Rightarrow \sigma_1(x) \geq \sigma_2(x)\}$. This is a hyper-assertion which, in the context of HHL, is a predicate over sets of states. Thus, σ_1 and σ_2 stand for the state variables whose values are states from the input set of states. By $\sigma(\cdot)$, we mean the value of a variable in the state σ. The variables in this hyper-assertion are x and t, and while x is a program variable (i.e., a variable usable by the program), t is a logical variable (i.e., a variable not usable by the program) used to separate states into categories. To prove the monotonicity, we would like to demonstrate that the post-states after the execution of the program satisfy the same hyper-assertion as a postcondition. For that purpose, we need to make use of HHL's Choice rule as defined in Fig. 1. However, to prove the postcondition, we need to be able to relate the post-states coming from the execution of branch C_1 with the post-states coming from the execution of branch C_2, as the postcondition relates all possible pairs of states. The limitation of the rule lies in the fact that none of its postconditions is able to relate the post-states coming from the execution of branch C_1 with the post-states coming from the execution of branch C_2. Postcondition Q_1 can only relate those states that come from the execution of branch C_1, and likewise, postcondition Q_2 can only relate those post-states coming from the execution of branch C_2. Most importantly, neither the postcondition $Q_1 \otimes Q_2$ defined as $Q_1 \otimes Q_2 \triangleq \exists\, S_1, S_2.\ S = S_1 \cup S_2 \ \land\ Q_1(S_1) \ \land\ Q_2(S_2)$ in the conclusion of the rule can relate them, since it only uses the postconditions Q_1 and Q_2 independently. This forces us to resort to the introduction of new logical variables that store the relationship of all pre-states and to frame it to the other side of the if statement while expressing and proving postconditions about exactly what branches C_1 and C_2 do[5], which makes the proof non-trivial, even-though the example is very simple.

```
if (∗) {
  assume x > 0;
  x := x + 2;
}
else {
  assume ¬(x > 0);
  x := x + 1;
}
```

Figure 2: A simple program to demonstrate HHL's Choice rule's limitations. The $*$ symbol denotes a non-deterministic choice.

Now imagine being able to reason about branches C_1 and C_2 as separate programs within a logic capable of expressing relational properties. This would allow us to create a choice rule capable of relating the post-states of the execution of program C_1 with the post-states of the execution of program C_2. Thus, the goal of this project is to extend HHL with ideas from LHC into a logic that supports expressing and reasoning about relational properties and simplifies the proofs of similar examples. We call this logic *Relational Hyper Hoare Logic* (RHHL).

3 Approach

Following LHC, we define hyper-programs as partial functions from natural numbers to programs and extend HHL's hyper-assertions from predicates over sets of states to predicates over functions from natural numbers to sets of states. For this purpose, we

[5] We express it by the Q_1 and Q_2 postconditions.

$$\frac{\vdash \{\mathbb{C}(P)\}\ \left[\begin{array}{l} 1:\ C_1 \\ 2:\ C_2 \end{array}\right]\ \{\mathbb{C}(Q)\}}{\vdash \{P\}\ [1:\ \textbf{if}\ (*)\ \{C_1\}\ \textbf{else}\ \{C_2\}]\ \{Q\}}\ (\textsc{Choice})$$

Figure 3: RHHL's Choice rule. Notice the hyper-program in the premise being a function from $\{1, 2\}$ to programs C_1 and C_2. This is in contrast with simple programs C_1 and C_2 from the premise of the HHL's version of the rule.

$$\frac{\vdash \left\{\begin{array}{l} (\forall_1\langle\sigma_1\rangle\,\forall_1\langle\sigma_2\rangle.\ P') \wedge \\ (\forall_1\langle\sigma_1\rangle\,\forall_2\langle\sigma_2\rangle.\ P') \wedge \\ (\forall_2\langle\sigma_1\rangle\,\forall_1\langle\sigma_2\rangle.\ P') \wedge \\ (\forall_2\langle\sigma_1\rangle\,\forall_2\langle\sigma_2\rangle.\ P') \end{array}\right\}\ \left[\begin{array}{l} 1:\ C_1' \\ 2:\ C_2' \end{array}\right]\ \left\{\begin{array}{l} (\forall_1\langle\sigma_1\rangle\,\forall_1\langle\sigma_2\rangle.\ P') \wedge \\ (\forall_1\langle\sigma_1\rangle\,\forall_2\langle\sigma_2\rangle.\ P') \wedge \\ (\forall_2\langle\sigma_1\rangle\,\forall_1\langle\sigma_2\rangle.\ P') \wedge \\ (\forall_2\langle\sigma_1\rangle\,\forall_2\langle\sigma_2\rangle.\ P') \end{array}\right\}}{\vdash \{\forall_1\langle\sigma_1\rangle\,\forall_1\langle\sigma_2\rangle.\ P'\}\ [1:\ \textbf{if}\ (*)\ \{C_1'\}\ \textbf{else}\ \{C_2'\}]\ \{\forall_1\langle\sigma_1\rangle\,\forall_1\langle\sigma_2\rangle.\ P'\}}\ (\textsc{Choice})$$

Figure 4: RHHL's Choice rule instantiated on the example from section 2.

need to extend HHL's syntax for hyper-assertions. Thus, when writing $\forall_i\langle\sigma\rangle$ or $\exists_i\langle\sigma\rangle$, we quantify over the states from the set of states at index i in the input function.

To illustrate what rules in RHHL might look like, we introduce RHHL's new Choice rule. In addition, this rule will also provide a simple way of proving the example from section 2, which posed problems for HHL. The general rule is defined as shown in Fig. 3. The substitution $\mathbb{C}$ replaces every occurrence of $\forall_1\langle\sigma\rangle.A$ with the conjunction of $\forall_1\langle\sigma\rangle.A$ and $\forall_2\langle\sigma\rangle.A$, and every occurrence of $\exists_1\langle\sigma\rangle.A$ with the disjunction of $\exists_1\langle\sigma\rangle.A$ and $\exists_2\langle\sigma\rangle.A$. Most importantly, unlike in HHL, the postconditions are now capable of relating the post-states of the two branches. This can be seen even better in the instantiation of the rule for our example. Let $C_1' \coloneqq \textbf{assume}\ x > 0;\ x := x + 2;$ and $C_2' \coloneqq \textbf{assume}\ \neg(x > 0);\ x := x + 1;$ be the branches of our example program, and $P' \coloneqq \sigma_1(t) = 1 \wedge \sigma_2(t) = 2 \Rightarrow \sigma_1(x) \geq \sigma_2(x)$ be the body of the hyper-assertion from our example. Then, the rule is instantiated as shown in Fig. 4. The $\mathbb{C}$ substitution divides our precondition and postcondition into four cases based on all the possible combinations of branches that the states related by the hyper-assertion can take. For example, the second conjunct corresponds to the case when the first state in the relationship takes the first branch, and the second state in the relationship takes the second branch. Crucially, the postconditions (especially the one in the rule's conclusion) are now capable of relating the post-states coming from the execution of branch C_1' with the post-states coming from the execution of branch C_2', which allows us to finish the proof of the example straightforwardly.

Additionally, we would like to devise new and useful relational loop rules. Consider the two programs in Fig. 5. We would like to prove that if the programs have the same values of n and x before their execution, then their value of x will be the same after their execution. This is intuitive, as we can see that, under such preconditions, the loop of program 5b will perform twice as many iterations as the loop of program 5a. At the same time, the value of x quadruples after every two iterations of program 5b and also quadruples after every iteration of program 5a. Thus, after the execution of the programs, variable x of program 5b will be multiplied by 2^{2*n}, and variable x

```
i := n
while (i > 0) {
  x := 4 * x;
  i := i − 1;
}
```

(a)

```
i := 2 * n
while (i > 0) {
  x := 2 * x;
  i := i − 1;
}
```

(b)

Figure 5: Two programs for illustrating synchronous loop rules with fixed alignment.

of program 5a will be multiplied by 4^n, which is the same value.[6] Devising a *fixed* alignment loop rule that enables us to prove the invariant that the values of x are equal after every two iterations of the loop of program 5b and after every iteration of the loop of program 5a would allow for a straightforward proof of the example without even proving what exactly the individual programs do. This rule is also called *synchronous* as if we allow program 5b to perform two iterations for every iteration of program 5a, the programs will finish at the same time. We would like to focus on the exploration of similar fixed alignment synchronous loop rules and incorporate them into our logic.

4 Goals

4.1 Core Goals

- Devise a core set of rules for RHHL and prove it sound and complete. RHHL will support the programming language used by HHL; most importantly, this includes loops and non-determinism.
- Facilitate simpler proofs of programs that contain assume statements and (deterministic or non-deterministic) assignments by devising rules based on transformations of syntactic hyper-assertions and proving the rules sound[7]. With regard to syntactic hyper-assertions, RHHL introduces quantification over the set of states corresponding to any input program, with quantifiers of type $\forall_i\langle\sigma\rangle$ and $\exists_i\langle\sigma\rangle$, and the index i specifying the program. The rest of the hyper-assertion syntax will likely remain the same as in HHL.
- Devise useful synchronous loop rules with fixed alignment.
- Explore the potential subsumption of LHC by RHHL. This, in particular, includes whether LHC's hyper-triples can be expressed in RHHL and whether all LHC's rules can be derived using RHHL's rules.
- Formalize the results in the Isabelle/HOL theorem prover [8].
- Perform case studies.

[6] Note that the value of n is not modified by the programs, so it does not matter whether we consider the value of n before or after the execution of the programs.

[7] Syntactic hyper-assertions are a restricted set of hyper-assertions that are defined syntactically.

4.2 Extensions

- Explore the potential for *alignment* completeness [9].
- Explore more loop alignment principles, such as asynchronous alignments and non-fixed alignments.

References

[1] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, Oct. 1969, ISSN: 0001-0782. DOI: 10.1145/363235.363259.

[2] D. Volpano, C. Irvine, and G. Smith, "A sound type system for secure flow analysis," *J. Comput. Secur.*, vol. 4, no. 2–3, pp. 167–187, Jan. 1996, ISSN: 0926-227X.

[3] M. R. Clarkson and F. B. Schneider, "Hyperproperties," in *2008 21st IEEE Computer Security Foundations Symposium*, 2008, pp. 51–65. DOI: 10.1109/CSF.2008.7.

[4] E. D'Osualdo, A. Farzan, and D. Dreyer, "Proving hypersafety compositionally," *Proc. ACM Program. Lang.*, vol. 6, no. OOPSLA2, Oct. 2022. DOI: 10.1145/3563298.

[5] N. Benton, "Simple relational correctness proofs for static analyses and program transformations," *SIGPLAN Not.*, vol. 39, no. 1, pp. 14–25, Jan. 2004, ISSN: 0362-1340. DOI: 10.1145/982962.964003.

[6] M. Sousa and I. Dillig, "Cartesian hoare logic for verifying k-safety properties," *SIGPLAN Not.*, vol. 51, no. 6, pp. 57–69, Jun. 2016, ISSN: 0362-1340. DOI: 10.1145/2980983.2908092.

[7] T. Dardinier and P. Müller, "Hyper hoare logic: (dis-)proving program hyperproperties," *Proc. ACM Program. Lang.*, vol. 8, no. PLDI, Jun. 2024. DOI: 10.1145/3656437.

[8] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL : a proof assistant for higher-order logic* (Lecture notes in computer science 2283), eng. Berlin: Springer, 2002, ISBN: 3540433767. [Online]. Available: http://www.loc.gov/catdir/enhancements/fy0817/2002020906-d.html.

[9] R. Nagasamudram and D. A. Naumann, "Alignment completeness for relational hoare logics," in *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470690.