Relational Hoare Logic for Sequential Program Verification

ANONYMOUS AUTHOR(S)

We propose an encoding theory to encode relational Hoare logic (∀∃) into traditional Hoare logic (∀). We encapsulate the ∀∃ pattern in assertions and prove that the validity of the encoded traditional Hoare triples ensures the validity of the original relational triples. Moerover, our encoding theory facilitates a hybrid reasoning approach with both relational and traditional Hoare logic.

Additional Key Words and Phrases: Program Verification, Relational Hoare Logic, Hoare Logic

1 Introduction

Relational Hoare logic is a valuable tool for reasoning about program equivalence (∀∀) and program refinement (∀∃). In this work, we focus on program refinement and employ program-as-resource assertions [5, 11, 12].

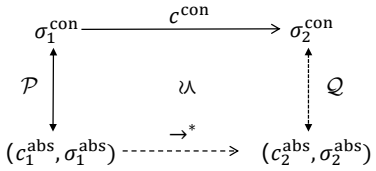

Fig. 1. Relational Hoare Triples

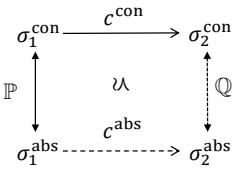

Fig. 2. Program Refinement

Definition 1.1 (Relational Hoare Triples with Program-as-resource Assertions). For ternary assertions $\mathcal{P}, \mathcal{Q} \subseteq \Sigma^{con} \times \Sigma^{abs} \times \text{Prog}^{abs}$, relational Hoare triple $\langle \mathcal{P} \rangle\ c^{con}\ \langle \mathcal{Q} \rangle$ is valid if and only if for any $(\sigma_1^{con}, \sigma_1^{abs}, c_1^{abs}) \in \mathcal{P}$ and any σ_2^{con} such that $(\sigma_1^{con}, \sigma_2^{con}) \in [\![c^{con}]\!]$, there exist σ_2^{abs} and c_2^{abs} such that $(\sigma_1^{abs}, c_1^{abs}) \hookrightarrow (\sigma_2^{abs}, c_2^{abs})$ and $(\sigma_2^{con}, \sigma_2^{abs}, c_2^{abs}) \in \mathcal{Q}$.

Here, Σ^{con} and Σ^{abs} denote the sets of concrete and abstract program states, respectively. $[\![c]\!]$ represents denotational semantics of program c and semantic reduction $(\sigma_1, c_1) \hookrightarrow (\sigma_2, c_2) \triangleq \forall \sigma_3.(\sigma_2, \sigma_3) \in [\![c_2]\!] \Rightarrow (\sigma_1, \sigma_3) \in [\![c_1]\!]$. In this way, the refinement relation between the concrete statement c^{con} and abstract statement c^{abs}, as shown in Fig. 2, can be described by the triple $\langle \mathbb{P} \wedge [c^{abs}] \rangle\ c^{con}\ \langle \mathbb{Q} \wedge [\text{skip}] \rangle$. Here binary assertions $\mathbb{P}$ and $\mathbb{Q}$ are lifted to ternary assertions, and we use $[-]$ to represent the abstract program. Based on relational Hoare triples, we identify inference rules similar to those in traditional Hoare logic, as shown in Fig. 3. For example, the rule Rel-Seq evaluates the currently focused statement in a manner similar to the reasoning employed by the rule Seq. Moreover, it is known that two refinement results can be vertically combined, and a refinement judgment can be vertically combined with a traditional Hoare triple.

Our research on relational Hoare logic has two contributions:

- We present a theory to encode ∀∃ relational Hoare logic into traditional Hoare logic. Consequently, our work facilitates the establishment of formalized relational proofs in existing traditional Hoare logic frameworks.
- We observe that while relational Hoare logic can be used in traditional Hoare logic proofs via vertical composition rules (appendix B), it is sometimes necessary to employ traditional Hoare logic in refinement proofs. Then we propose a hybrid reasoning approach that flexibly combines traditional Hoare logic with relational Hoare logic.

$$\textsc{Rel-Seq}\ \frac{\langle \mathcal{P}\rangle\ c_1^{\text{con}}\ \langle \mathcal{R}\rangle \qquad \langle \mathcal{R}\rangle\ c_2^{\text{con}}\ \langle \mathcal{Q}\rangle}{\langle \mathcal{P}\rangle\ c_1^{\text{con}}; c_2^{\text{con}}\ \langle \mathcal{Q}\rangle} \qquad \textsc{Seq}\ \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$

$$\textsc{Abs-Focus}\ \frac{\{P\}\ c_1^{\text{abs}}\ \{R\} \qquad \langle \lfloor F\rfloor \wedge \lceil R\rceil \wedge [c_2^{\text{abs}}]\rangle\ c^{\text{con}}\ \langle \mathcal{Q}\rangle}{\langle \lfloor F\rfloor \wedge \lceil P\rceil \wedge [c_1^{\text{abs}}; c_2^{\text{abs}}]\rangle\ c^{\text{con}}\ \langle \mathcal{Q}\rangle} \qquad \textsc{Conseq-Pre}\ \frac{P \Rightarrow R \qquad \{R\}\ c\ \{Q\}}{\{P\}\ c\ \{Q\}}$$

$$\textsc{Rht-ExIntro}\ \frac{\forall x.\ \langle \mathcal{P}(x)\rangle\ c^{\text{con}}\ \langle \mathcal{Q}\rangle}{\langle \exists x. \mathcal{P}(x)\rangle\ c^{\text{con}}\ \langle \mathcal{Q}\rangle} \qquad \textsc{ExIntro}\ \frac{\forall x.\ \{P(x)\}\ c\ \{Q\}}{\{\exists x. P(x)\}\ c\ \{Q\}}$$

Fig. 3. Similar Proof Rules in Relational and Traditional Hoare Logic

2 Encoding Theory

To illustrate program refinement, relational Hoare logic requires demonstrating: for any execution of the concrete program (formalized as $\forall$), there exists a corresponding execution of the abstract program that simulates it (formalized as $\exists$). In comparison, traditional Hoare triple $\{P\}\ c\ \{Q\}$ is defined to be valid if and only if, for any initial state satisfying the precondition P, any final state after executing program c will satisfy postcondition Q (formalized as $\forall$). It is known that relational logic for program equivalence ($\forall\forall$) can be systematically reduced to traditional Hoare logic through self-composition [1]. Moreover, in concurrent program verification, relational logic for program refinement can be encoded within concurrent separation logic with ghost variable ($\forall\exists$) [2, 9, 10], while its judgement is also defined in a $\forall\exists$ pattern. However, it remains unknown how the semantics of program refinement can be reduced to traditional Hoare logic.

In this work, we present a theory to encode relational Hoare triples into traditional Hoare triples. At the core of our encoding theory is to define the assertion encoding $(\!|-|\!)$, which transforms a ternary assertion $\mathcal{P} \subseteq \Sigma^{\text{con}} \times \Sigma^{\text{abs}} \times \text{Prog}^{\text{abs}}$ into a unary assertion $(\!|\mathcal{P}|\!) \subseteq \Sigma^{\text{con}}$. To define it, we introduce a terminal set of abstract states, denoted by X.

Definition 2.1 (Assertion Encoding). Given any abstract states set $X \subseteq \Sigma^{\text{abs}}$, for any ternary assertion $\mathcal{P} \subseteq \Sigma^{\text{con}} \times \Sigma^{\text{abs}} \times \text{Prog}^{\text{abs}}$, define its encoding as

$$(\!|\mathcal{P}|\!)_X \triangleq \lambda\sigma^{\text{con}}.\ \exists\sigma^{\text{abs}} c^{\text{abs}}.\ \sigma^{\text{abs}} \in \text{wp}(c^{\text{abs}}, X) \wedge (\sigma^{\text{con}}, \sigma^{\text{abs}}, c^{\text{abs}}) \in \mathcal{P}$$
$$\text{wp}(c^{\text{abs}}, X) \triangleq \{\sigma^{\text{abs}} \mid \forall\sigma_0^{\text{abs}}.\ (\sigma^{\text{abs}}, \sigma_0^{\text{abs}}) \in [\![c^{\text{abs}}]\!] \Rightarrow \sigma_0^{\text{abs}} \in X\}$$

Subsequently, we use the assertion encoding to encode relational Hoare triples and we prove that this encoding is correct through the following theorem.

Theorem 2.2 (Encoding Relational Triples). *For any concrete statement c^{con}, and assertion $\mathcal{P}$, $\mathcal{Q} \subseteq \Sigma^{con} \times \Sigma^{\text{abs}} \times Prog^{\text{abs}}$, the relational Hoare triple*

$$\langle \mathcal{P}\rangle\ c^{\text{con}}\ \langle \mathcal{Q}\rangle \text{ is valid} \qquad \text{iff.} \qquad \forall(X \subseteq \Sigma^{\text{abs}}).\ \{(\!|\mathcal{P}|\!)_X\}\ c^{\text{con}}\ \{(\!|\mathcal{Q}|\!)_X\} \text{ is valid}$$

In real program verification, ternary assertions are are usually written in a decomposed form of components related to concrete states, abstract states, and abstract programs, using existential variables and pure facts. When encoding $\mathcal{P}$, the existential variables and pure facts are lifted out, and the remaining components are encoded as the conjunction of a pure fact and a unary assertion.

Proposition 2.3. $(\!|\exists a. \mathcal{P}(a)|\!)_X \iff \exists a.\ (\!|\mathcal{P}(a)|\!)$ *and* $(\!|B \wedge \mathcal{P}|\!)_X \iff B \wedge (\!|\mathcal{P}|\!)_X$.

Proposition 2.4. $(\!|\lfloor P^{\text{con}} \rfloor \wedge \lceil P^{\text{abs}} \rceil \wedge [c^{\text{abs}}]|\!)_X \iff \mathsf{Exec}_X(P^{\text{abs}}, c^{\text{abs}}) \wedge P^{\text{con}}$.

Here $\mathsf{Exec}_X(P^{\text{abs}}, c^{\text{abs}}) \triangleq \mathsf{wp}(c^{\text{abs}}, X) \subseteq P^{\text{abs}}$, indicating that assertion P^{abs} overestimates the weakest precondition. Then, we can use the following Hoare triple to describe program refinement.

$$\forall X.\, \{\mathsf{Exec}_X(P^{\text{abs}}, c^{\text{abs}}) \wedge P^{\text{con}}\}\ c^{\text{con}}\ \{\exists a.\, \mathsf{Exec}_X(Q^{\text{abs}}(a), \mathsf{skip}) \wedge Q^{\text{con}}(a)\}$$

Furthermore, traditional Hoare logic's inference rules can address relational verification problems. For example, rule Abs-Focus can be considered as a special case of the rule Conseq-Pre:

$$\frac{\mathsf{Exec}_X(P^{\text{abs}}, c_1^{\text{abs}}; c_2^{\text{abs}}) \Rightarrow \mathsf{Exec}_X(R^{\text{abs}}, c_2^{\text{abs}}) \qquad \{\mathsf{Exec}_X(R^{\text{abs}}, c_2^{\text{abs}}) \wedge F^{\text{con}}\}\ c^{\text{con}}\ \{(\!|Q|\!)\}}{\{\mathsf{Exec}_X(P^{\text{abs}}, c_1^{\text{abs}}; c_2^{\text{abs}}) \wedge F^{\text{con}}\}\ c^{\text{con}}\ \{(\!|Q|\!)\}}$$

3 Hybrid Reasoning with Relational and Traditional Hoare Logic

We consider a hybrid merge sorting algorithm, as shown in Fig. 4. The concrete program `mergesort`

concrete program

```
1 struct list * mergesort(struct list *x)
2 { int l = listlen(x);
3   if (l < 8)  return inssort(x);
4   struct list *p, *q;
5   p = q = NULL;
6   split(x, &p, &q);
7   p = mergesort(p);
8   q = mergesort(q);
9   return merge(p, q); }
```

abstract program

```
Def extsort_A (l: list Z) :=
  return r s.t. permutation(l,r) ∧ sorted(r)
Def mergesort_A (l: list Z) :=
choice (return extsort_A l ,
  test (length(l) ≥ 2 );
  (m, n) ← split_A(l,nil,nil);
  m ← mergesort_A(m);
  n ← mergesort_A(n);
  return merge_A(m,n))
```

Fig. 4. Hybrid Sort: Concrete vs. Abstract Implementations

uses insertion sort for lists with fewer than 8 elements, while the abstract program $\texttt{mergesort}_A$ nondeterministically selects between an external sort and merge sort. Then the functional correctness of `mergesort` can be divided into two parts: proving that `mergesort` refines $\texttt{mergesort}_A$ and verifying the algorithm correctness of $\texttt{mergesort}_A$. This proof decomposition enables verifiers to reuse the algorithm correctness proof for verifying various concrete implementations.

Since the structures of `mergesort` and $\texttt{mergesort}_A$ are quite similar, the refinement proof can be reduced to smaller refinement statements, such as proving that `merge` refines $\texttt{merge}_A$.

$$\begin{aligned} &\forall X.\, \{\mathsf{Exec}_X(\mathsf{m} = l_x \wedge \mathsf{n} = l_y, \texttt{merge}_A(\mathsf{m}, \mathsf{n})) \wedge \mathsf{sll}(\mathsf{x}, l_x) * \mathsf{sll}(\mathsf{y}, l_y)\} \\ &\quad \texttt{merge}(\mathsf{x}, \mathsf{y}) \\ &\quad \{\exists l_r.\mathsf{Exec}_X(\mathsf{r}^{\text{abs}} = l_r, \mathsf{skip}) \wedge \mathsf{sll}(\mathsf{r}^{\text{con}}, l_r)\} \end{aligned} \tag{1}$$

Here, $\mathsf{sll}(\mathsf{x}, l)$ denotes that variable x stores a singly-linked list l, and variables r^{abs} and r^{con} store the return values in the abstract and concrete programs respectively. The most interesting part is, in order to prove that `inssort` refines $\texttt{extsort}_A$, it suffices to prove a traditional Hoare triple:

$$\{\mathsf{sll}(\mathsf{x}, l_x)\}\ \texttt{inssort}(\mathsf{x})\ \{\exists l_r.\mathsf{permutation}(l_x, l_r) \wedge \mathsf{sorted}(l_r) \wedge \mathsf{sll}(\mathsf{r}^{\text{con}}, l_r)\} \tag{2}$$

4 Related Work

In concurrent program verification, Liang and Feng [5] and Turon et al. [12] proposed treating abstract programs as resources. Subsequent works [2, 6–10] on program refinement all adopt a $\forall\exists$ pattern in their semantics. In contrast, we encapsulate $\forall\exists$ simulations using logical variables within unary assertions and encode relational Hoare triples into traditional Hoare triples.

References

[1] G. Barthe, P.R. D'Argenio, and T. Rezk. 2004. Secure information flow by self-composition. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.* 100–114. https://doi.org/10.1109/CSFW.2004.1310735

[2] Dan Frumin, Robbert Krebbers, and Lars Birkedal. 2020. ReLoC Reloaded: A Mechanized Relational Logic for Fine-Grained Concurrency and Logical Atomicity. *CoRR* abs/2006.13635 (2020). arXiv:2006.13635 https://arxiv.org/abs/2006.13635

[3] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.* 28 (2018), e20. https://doi.org/10.1017/S0956796818000151

[4] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, Sriram K. Rajamani and David Walker (Eds.). ACM, 637–650. https://doi.org/10.1145/2676726.2676980

[5] Hongjin Liang and Xinyu Feng. 2013. Modular verification of linearizability with non-fixed linearization points. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, Hans-Juergen Boehm and Cormac Flanagan (Eds.). ACM, 459–470. https://doi.org/10.1145/2491956.2462189

[6] Hongjin Liang and Xinyu Feng. 2016. A program logic for concurrent objects under fair scheduling. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 385–399. https://doi.org/10.1145/2837614.2837635

[7] Hongjin Liang and Xinyu Feng. 2018. Progress of concurrent objects with partial methods. *Proc. ACM Program. Lang.* 2, POPL (2018), 20:1–20:31. https://doi.org/10.1145/3158108

[8] Hongjin Liang, Xinyu Feng, and Zhong Shao. 2014. Compositional verification of termination-preserving refinement of concurrent programs. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, Thomas A. Henzinger and Dale Miller (Eds.). ACM, 65:1–65:10. https://doi.org/10.1145/2603088.2603123

[9] Simon Spies, Lennard Gäher, Daniel Gratzer, Joseph Tassarotti, Robbert Krebbers, Derek Dreyer, and Lars Birkedal. 2021. Transfinite Iris: resolving an existential dilemma of step-indexed separation logic. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 80–95. https://doi.org/10.1145/3453483.3454031

[10] Joseph Tassarotti, Ralf Jung, and Robert Harper. 2017. A Higher-Order Logic for Concurrent Termination-Preserving Refinement. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10201)*, Hongseok Yang (Ed.). Springer, 909–936. https://doi.org/10.1007/978-3-662-54434-1_34

[11] Aaron Turon, Derek Dreyer, and Lars Birkedal. 2013. Unifying Refinement and Hoare-Style Reasoning in a Logic for Higher-Order Concurrency. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming* (Boston, Massachusetts, USA) *(POPL '13)*. Association for Computing Machinery, New York, NY, USA, 377–390. https://doi.org/10.1145/2500365.2500600

[12] Aaron Joseph Turon, Jacob Thamsborg, Amal Ahmed, Lars Birkedal, and Derek Dreyer. 2013. Logical relations for fine-grained concurrency. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, Roberto Giacobazzi and Radhia Cousot (Eds.). ACM, 343–356. https://doi.org/10.1145/2429069.2429111

A Notations, Definitions and Conclusions

The notations used in this work are summarized in the reference table, as shown in Table 1.

Notation	Description
Σ^{con}	The set of concrete program states
Σ^{abs}	The set of abstract program states
Prog^{con}	The set of concrete programs
Prog^{abs}	The set of abstract programs
$\sigma^{\text{con}}(\in \Sigma^{\text{con}})$	A concrete program state
$\sigma^{\text{abs}}(\in \Sigma^{\text{abs}})$	An abstract program state
$c^{\text{con}}(\in \text{Prog}^{\text{con}})$	A concrete program
$c^{\text{abs}}(\in \text{Prog}^{\text{abs}})$	An abstract program
$[\![c]\!](\subseteq \Sigma \times \Sigma)$	Denotational semantics of program c
$P^{\text{con}}(\subseteq \Sigma^{\text{con}})$	Unary assertion about concrete program states
$P^{\text{abs}}(\subseteq \Sigma^{\text{abs}})$	Unary assertion about abstract program states
$\mathbb{P}(\subseteq \Sigma^{\text{con}} \times \Sigma^{\text{abs}})$	Binary assertion about states of both programs
$\mathcal{P}(\subseteq \Sigma^{\text{con}} \times \Sigma^{\text{abs}} \times \text{Prog}^{\text{abs}})$	Ternary program-as-resource assertion
$\lfloor P^{\text{con}} \rfloor$	Lifted ternary assertion for P^{con}
$\lceil P^{\text{abs}} \rceil$	Lifted ternary assertion for P^{abs}
$[c^{\text{abs}}]$	Lifted ternary assertion for c^{abs}
$(\!\lvert \mathcal{P} \rvert\!)_X$	Encoded unary assertion based on state set X
$\langle \mathcal{P} \rangle\ c^{\text{con}}\ \langle \mathcal{Q} \rangle$	Relational Hoare triple
$\{P\}\ c\ \{Q\}$	Traditional Hoare triple
$\text{wp}(c, Q)$	Weakest precondition
$\text{Exec}_X(P^{\text{abs}}, c^{\text{abs}})$	Encoded pure fact about abstract program execution
r^{abs}	Return variable in the abstract program
r^{con}	Return variable in the concrete program

Table 1. Notation table for key symbols and definitions.

We formally list all definitions and conclusions in our work.

Definition A.1 (Denotational Semantics). for any program $c \in \text{Prog}$, its denotation $[\![c]\!] \subseteq \Sigma \times \Sigma$ is defined as a nondeterministic state transition. For any state pair $(\sigma_1, \sigma_2) \in [\![c]\!]$, program c executes from initial state σ_1 can terminate at state σ_2.

Proposition A.2. $(\sigma_1, \sigma_2) \in [\![\text{skip}]\!]$ *iff* $\sigma_1 = \sigma_2$

Definition A.3 (Semantic Reduction). For any programs $c_1, c_2 \in \mathrm{Prog}$ and programs states $\sigma_1, \sigma_2 \in \Sigma$, we define semantic reduction based on denotational semantics as

$$(\sigma_1, c_1) \hookrightarrow (\sigma_2, c_2) \triangleq \forall \sigma_3. (\sigma_2, \sigma_3) \in [\![c_2]\!] \Rightarrow (\sigma_1, \sigma_3) \in [\![c_1]\!]$$

Proposition A.4. $(\sigma_1, c) \hookrightarrow (\sigma_2, \mathsf{skip})$ *is equivalent to* $(\sigma_1, \sigma_2) \in [\![c]\!]$

Definition A.5 (Traditional Hoare Triples). For unary assertions $P, Q \subseteq \Sigma$, triple $\{P\}\ c\ \{Q\}$ is valid if for any $\sigma_1 \in P$ and any σ_2 such that $(\sigma_1, \sigma_2) \in [\![c]\!]$, we have $\sigma_2 \in Q$.

Definition A.6 (Relational Hoare Triples with Program-as-resource Assertions). For ternary assertions $\mathcal{P}, \mathcal{Q} \subseteq \Sigma^{\mathrm{con}} \times \Sigma^{\mathrm{abs}} \times \mathrm{Prog}^{\mathrm{abs}}$, relational Hoare triple $\langle \mathcal{P} \rangle\ c^{\mathrm{con}}\ \langle \mathcal{Q} \rangle$ is valid if and only if for any $(\sigma_1^{\mathrm{con}}, \sigma_1^{\mathrm{abs}}, c_1^{\mathrm{abs}}) \in \mathcal{P}$ and any σ_2^{con} such that $(\sigma_1^{\mathrm{con}}, \sigma_2^{\mathrm{con}}) \in [\![c^{\mathrm{con}}]\!]$, there exist σ_2^{abs} and c_2^{abs} such that $(\sigma_1^{\mathrm{abs}}, c_1^{\mathrm{abs}}) \hookrightarrow (\sigma_2^{\mathrm{abs}}, c_2^{\mathrm{abs}})$ and $(\sigma_2^{\mathrm{con}}, \sigma_2^{\mathrm{abs}}, c_2^{\mathrm{abs}}) \in \mathcal{Q}$.

Definition A.7 (Abstract Program Lifting). For any abstract program $c_0^{\mathrm{abs}} \in \mathrm{Prog}^{\mathrm{abs}}$, its lifting to a ternary assertion is defined as $[c_0^{\mathrm{abs}}] \triangleq \{(\sigma^{\mathrm{con}}, \sigma^{\mathrm{abs}}, c^{\mathrm{abs}}) \mid c^{\mathrm{abs}} = c_0^{\mathrm{abs}}\}$

Definition A.8 (Concrete Assertion Lifting). For any concrete program assertion $P^{\mathrm{con}} \subseteq \Sigma^{\mathrm{con}}$, its lifting to a ternary assertion is defined as $\lfloor P^{\mathrm{con}} \rfloor \triangleq \{(\sigma^{\mathrm{con}}, \sigma^{\mathrm{abs}}, c^{\mathrm{abs}}) \mid \sigma^{\mathrm{con}} \in P^{\mathrm{con}}\}$

Definition A.9 (Abstract Assertion Lifting). For any abstract program assertion $P^{\mathrm{abs}} \subseteq \Sigma^{\mathrm{abs}}$, its lifting to a ternary assertion is defined as $\lceil P^{\mathrm{abs}} \rceil \triangleq \{(\sigma^{\mathrm{con}}, \sigma^{\mathrm{abs}}, c^{\mathrm{abs}}) \mid \sigma^{\mathrm{abs}} \in P^{\mathrm{abs}}\}$

Proposition A.10. $(\sigma^{\mathrm{con}}, \sigma^{\mathrm{abs}}, c^{\mathrm{abs}}) \in \mathbb{P} \wedge [c_0^{\mathrm{abs}}]$ *iff* $(\sigma^{\mathrm{con}}, \sigma^{\mathrm{abs}}) \in \mathbb{P}$ *and* $c^{\mathrm{abs}} = c_0^{\mathrm{abs}}$

Proposition A.11. $(\sigma^{\mathrm{con}}, \sigma^{\mathrm{abs}}, c^{\mathrm{abs}}) \in \lfloor P^{\mathrm{con}} \rfloor \wedge \lceil P^{\mathrm{abs}} \rceil \wedge [c_0^{\mathrm{abs}}]$ *iff* $\sigma^{\mathrm{con}} \in P^{\mathrm{con}}$ *and* $\sigma^{\mathrm{abs}} \in P^{\mathrm{abs}}$ *and* $c^{\mathrm{abs}} = c_0^{\mathrm{abs}}$

Definition A.12 (Assertion Encoding). Given any abstract states set $X \subseteq \Sigma^{\mathrm{abs}}$, for any ternary assertion $\mathcal{P} \subseteq \Sigma^{\mathrm{con}} \times \Sigma^{\mathrm{abs}} \times \mathrm{Prog}^{\mathrm{abs}}$, define its encoding as

$$(\!|\mathcal{P}|\!)_X \triangleq \lambda \sigma^{\mathrm{con}}.\ \exists \sigma^{\mathrm{abs}} c^{\mathrm{abs}}.\ \sigma^{\mathrm{abs}} \in \mathrm{wp}(c^{\mathrm{abs}}, X) \wedge (\sigma^{\mathrm{con}}, \sigma^{\mathrm{abs}}, c^{\mathrm{abs}}) \in \mathcal{P}$$

$$\mathrm{wp}(c^{\mathrm{abs}}, X) \triangleq \{\sigma^{\mathrm{abs}} \mid \forall \sigma_0^{\mathrm{abs}}.\ (\sigma^{\mathrm{abs}}, \sigma_0^{\mathrm{abs}}) \in [\![c^{\mathrm{abs}}]\!] \Rightarrow \sigma_0^{\mathrm{abs}} \in X\}$$

Theorem A.13 (Encoding Relational Triples). *For any concrete statement* c^{con}*, and assertion* $\mathcal{P}, \mathcal{Q} \subseteq \Sigma^{con} \times \Sigma^{\mathrm{abs}} \times Prog^{\mathrm{abs}}$*, the relational Hoare triple*

$$\langle \mathcal{P} \rangle\ c^{\mathrm{con}}\ \langle \mathcal{Q} \rangle \text{ is valid} \quad \text{iff.} \quad \forall (X \subseteq \Sigma^{\mathrm{abs}}).\ \{(\!|\mathcal{P}|\!)_X\}\ c^{\mathrm{con}}\ \{(\!|\mathcal{Q}|\!)_X\} \text{ is valid}$$

Definition A.14 (Abstract Program Execution Predicate). for any abstract program $c^{\mathrm{abs}} \in \mathrm{Prog}^{\mathrm{abs}}$, starting states set P^{abs}, and terminal states set X, define

$$\mathsf{Exec}_X(P^{\mathrm{abs}}, c^{\mathrm{abs}}) \triangleq \mathrm{wp}(c^{\mathrm{abs}}, X) \subseteq P^{\mathrm{abs}}$$

which represents that from some state in P^{abs}, any final state belongs to X.

Proposition A.15. $(\!|\exists a. \mathcal{P}(a)|\!)_X \iff \exists a.\ (\!|\mathcal{P}(a)|\!)$ *and* $(\!|B \wedge \mathcal{P}|\!)_X \iff B \wedge (\!|\mathcal{P}|\!)_X$.

Proposition A.16. $(\!|\lfloor P^{\mathrm{con}} \rfloor \wedge \lceil P^{\mathrm{abs}} \rceil \wedge [c^{\mathrm{abs}}]|\!)_X \iff \mathsf{Exec}_X(P^{\mathrm{abs}}, c^{\mathrm{abs}}) \wedge P^{\mathrm{con}}$.

Corollary A.17 (Encoding Relational Triples with Decomposed Assertions). *Relational triple* $\langle \lfloor P^{\mathrm{con}} \rfloor \wedge \lceil P^{\mathrm{abs}} \rceil \wedge [c_1^{\mathrm{abs}}] \rangle\ c^{\mathrm{con}}\ \langle \exists a. \lfloor Q^{\mathrm{con}}(a) \rfloor \wedge \lceil Q^{\mathrm{abs}}(a) \rceil \wedge [c_2^{\mathrm{abs}}] \rangle$ *is valid if and only if for any* X*, traditional triple* $\{\mathsf{Exec}_X(P^{\mathrm{abs}}, c_1^{\mathrm{abs}}) \wedge P^{\mathrm{con}}\}\ c^{\mathrm{con}}\ \{\exists a. \mathsf{Exec}_X(Q^{\mathrm{abs}}(a), c_2^{\mathrm{abs}}) \wedge Q^{\mathrm{con}}(a)\}$ *is valid*

B Vertical Composition

In relational frameworks, when a concrete program is proved to refine an abstract program, then all the properties that hold in the abstract also hold in the concrete. This concept can be captured by the rules Vertical-Comp and Refine-Comp.

$$\frac{\langle \mathbb{P} \wedge [c^{\text{abs}}] \rangle\ c^{\text{con}}\ \langle \mathbb{Q} \wedge [\texttt{skip}] \rangle \qquad \{P^{\text{abs}}\}\ c^{\text{abs}}\ \{Q^{\text{abs}}\}}{\{\mathbb{P} \circ P^{\text{abs}}\}\ c^{\text{con}}\ \{\mathbb{Q} \circ Q^{\text{abs}}\}}\ \textsc{Vertical-Comp}$$

$$\frac{\langle \mathbb{P}_1 \wedge [c_2] \rangle\ c_1\ \langle \mathbb{Q}_1 \wedge [\texttt{skip}] \rangle \qquad \langle \mathbb{P}_2 \wedge [c_3] \rangle\ c_2\ \langle \mathbb{Q}_2 \wedge [\texttt{skip}] \rangle}{\langle \mathbb{P}_1 \circ \mathbb{P}_2 \wedge [c_3] \rangle\ c_1\ \langle \mathbb{Q}_1 \circ \mathbb{Q}_2 \wedge [\texttt{skip}] \rangle}\ \textsc{Refine-Comp}$$

Here, the operator $\circ$ is overloaded to denote the concatenation of two sets:

$$\mathbb{P} \circ P^{\text{abs}} \triangleq \{\sigma^{\text{con}} \mid \exists \sigma^{\text{abs}}.(\sigma^{\text{con}}, \sigma^{\text{abs}}) \in \mathbb{P} \wedge \sigma^{\text{abs}} \in P^{\text{abs}}\}$$
$$\mathbb{P}_1 \circ \mathbb{P}_2 \triangleq \{(\sigma_1, \sigma_2) \mid \exists \sigma_0.(\sigma_1, \sigma_0) \in \mathbb{P}_1 \wedge (\sigma_0, \sigma_2) \in \mathbb{P}_2\}$$

C the Relational Nature of Traditional Hoare Logic Proofs

We discover that some functional correctness proofs based on traditional Hoare logic inherently possesses relational proof characteristics. Typically, the verification can be divided into two parts: implementation correctness, ensuring that the concrete program accurately implements an algorithm description, and algorithm correctness, validating the correctness of the algorithm description itself. Such proof decomposition allows verifiers to reuse the algorithm correctness proof for verifying different implementations. Considering the insertion function for binary search trees, the Hoare triple below demonstrates the implementation correctness:

$$\{\mathsf{storetree}(\mathsf{b}, t)\}\ \texttt{insert}(\mathsf{b}, k, v)\ \{\mathsf{storetree}(\mathsf{b}, \texttt{ins}(t, k, v))\} \tag{3}$$

where $\mathsf{storetree}(\mathsf{b}, t)$ signifies that a tree t is stored at address b, and `ins` is a functional program in Coq. Then, another proof ensures that $\texttt{ins}(t, k, v)$ satisfies algorithm correctness:

$$\forall t\ k\ v\ m.\ \mathsf{bst}(t) \wedge t \sim m \Rightarrow \mathsf{bst}(\mathsf{ins}(t, k, v)) \wedge \texttt{ins}(t, k, v) \sim m[k \mapsto v] \tag{4}$$

where $\mathsf{bst}(t)$ asserts that tree t is a binary search tree, and $t \sim m$ indicates that the data of tree t corresponds to a mapping m. By combining these two proofs, the functional correctness is derived:

$$\{\mathsf{storemap}(\mathsf{b}, m)\}\ \texttt{insert}(\mathsf{b}, k, v)\ \{\mathsf{storemap}(\mathsf{b}, m[k \mapsto v])\} \tag{5}$$

where $\mathsf{storemap}(\mathsf{b}, m) \triangleq \exists t.\ t \sim m \wedge \mathsf{storetree}(\mathsf{b}, t)$.

In fact, the traditional Hoare triple (3) implicitly establishes a relationship between the states of the concrete program $\texttt{insert}(\mathsf{b}, k, v)$ and the abstract program $\texttt{ins}(t, k, v)$. Then we can restructure it into a relational triple:

$$\langle \lfloor \mathsf{storetree}(\mathsf{b}, t) \rfloor \wedge [t := \texttt{ins}(t, k, v)] \rangle\ \texttt{insert}(\mathsf{b}, k, v)\ \langle \lfloor \mathsf{storetree}(\mathsf{b}, t) \rfloor \wedge [\texttt{skip}] \rangle \tag{6}$$

Besides, if we represent the algorithm correctness (4) as a traditional Hoare triple:

$$\{\mathsf{bst}(t) \wedge t \sim m\}\ t := \texttt{ins}(t, k, v)\ \{\mathsf{bst}(t) \wedge t \sim m[k \mapsto v]\}$$

then we can vertically compose it with the quadruple to derive the functional correctness (5) using the rule Vertical-Comp.