A TECHNICAL DEFINITIONS OMITTED FROM THE PAPER

Definition 12. ***Evaluation of syntactic hyper-expressions and satisfiability of hyper-assertions.***
Let Σ a mapping from variables (such as φ) to states, and Δ a mapping from variables (such as x) to values.[12] *The evaluation of hyper-expressions is defined as follows:*

$$\begin{aligned}
&[\![c]\!]^{\Sigma}_{\Delta} \triangleq c\\
&[\![y]\!]^{\Sigma}_{\Delta} \triangleq \Delta(y)\\
&[\![\varphi^{P}(x)]\!]^{\Sigma}_{\Delta} \triangleq (\Sigma(\varphi))^{P}(x)\\
&[\![\varphi^{L}(x)]\!]^{\Sigma}_{\Delta} \triangleq (\Sigma(\varphi))^{L}(x)\\
&[\![e_1 \oplus e_2]\!]^{\Sigma}_{\Delta} \triangleq [\![e_1]\!]^{\Sigma}_{\Delta} \oplus [\![e_2]\!]^{\Sigma}_{\Delta}\\
&[\![f(e)]\!]^{\Sigma}_{\Delta} \triangleq f([\![e]\!]^{\Sigma}_{\Delta})
\end{aligned}$$

Let S be a set of states. The satisfiability of hyper-assertions is defined as follows:

$$\begin{aligned}
&S, \Sigma, \Delta \models b \triangleq b\\
&S, \Sigma, \Delta \models e_1 \geq e_2 \triangleq \left([\![e_1]\!]^{\Sigma}_{\Delta} \geq [\![e_2]\!]^{\Sigma}_{\Delta}\right)\\
&S, \Sigma, \Delta \models A \land B \triangleq (S, \Sigma, \Delta \models A \land S, \Sigma, \Delta \models B)\\
&S, \Sigma, \Delta \models A \lor B \triangleq (S, \Sigma, \Delta \models A \lor S, \Sigma, \Delta \models B)\\
&S, \Sigma, \Delta \models \forall x.\, A \triangleq (\forall v.\, S, \Sigma, \Delta[x \mapsto v] \models A)\\
&S, \Sigma, \Delta \models \exists x.\, A \triangleq (\exists v.\, S, \Sigma, \Delta[x \mapsto v] \models A)\\
&S, \Sigma, \Delta \models \forall \varphi.\, A \triangleq (\forall \alpha.\, S, \Sigma[\varphi \mapsto \alpha], \Delta \models A)\\
&S, \Sigma, \Delta \models \exists \varphi.\, A \triangleq (\exists \alpha.\, S, \Sigma[\varphi \mapsto \alpha], \Delta \models A)
\end{aligned}$$

When interpreting hyper-assertions in hyper-triples, we start with Δ and Σ being the empty mappings, except when there is an explicit quantifier around the triple, such as in the premises for the rule While-$\exists$ from Fig. 6.

Definition 13. ***Syntactic transformation for deterministic assignments.***
$\mathcal{A}^{e}_{x}[A]$ yields the hyper-assertion A, where $\varphi(x)$ is syntactically substituted by $e(\varphi)$, for all (existentially or universally) quantified states φ:

$$\begin{aligned}
&\mathcal{A}^{e}_{x}[b] \triangleq b\\
&\mathcal{A}^{e}_{x}[e_1 \geq e_2] \triangleq e_1 \geq e_2\\
&\mathcal{A}^{e}_{x}[A \land B] \triangleq \mathcal{A}^{e}_{x}[A] \land \mathcal{A}^{e}_{x}[B]\\
&\mathcal{A}^{e}_{x}[A \lor B] \triangleq \mathcal{A}^{e}_{x}[A] \lor \mathcal{A}^{e}_{x}[B]\\
&\mathcal{A}^{e}_{x}[\forall x.\, A] \triangleq \forall x.\, \mathcal{A}^{e}_{x}[A]\\
&\mathcal{A}^{e}_{x}[\exists x.\, A] \triangleq \exists x.\, \mathcal{A}^{e}_{x}[A]\\
&\mathcal{A}^{e}_{x}[\forall \langle\varphi\rangle.\, A] \triangleq \left(\forall \langle\varphi\rangle.\, \mathcal{A}^{e}_{x}[A[e(\varphi)/\varphi(x)]]\right)\\
&\mathcal{A}^{e}_{x}[\exists \langle\varphi\rangle.\, A] \triangleq \left(\exists \langle\varphi\rangle.\, \mathcal{A}^{e}_{x}[A[e(\varphi)/\varphi(x)]]\right)
\end{aligned}$$

where $A[y/x]$ refers to the standard syntactic substitution of x by y.

[12]In our Isabelle formalization, these mappings are actually lists, since we use De Bruijn indices [de Bruijn 1972].

Definition 14. ***Syntactic transformation for non-deterministic assignments.*** *$\mathcal{H}_x\ [A]$ yields the hyper-assertion A where $\varphi(x)$ is syntactically substituted by a fresh quantified variable v, universally (resp. existentially) quantified for universally (resp. existentially) quantified states:*

$$
\begin{aligned}
&\mathcal{H}_x\ [b] \triangleq b\\
&\mathcal{H}_x\ [e_1 \geq e_2] \triangleq e_1 \geq e_2\\
&\mathcal{H}_x\ [A \wedge B] \triangleq \mathcal{H}_x\ [A] \wedge \mathcal{H}_x\ [B]\\
&\mathcal{H}_x\ [A \vee B] \triangleq \mathcal{H}_x\ [A] \vee \mathcal{H}_x\ [B]\\
&\mathcal{H}_x\ [\forall x.\ A] \triangleq \forall x.\ \mathcal{H}_x\ [A]\\
&\mathcal{H}_x\ [\exists x.\ A] \triangleq \exists x.\ \mathcal{H}_x\ [A]\\
&\mathcal{H}_x\ [\forall \langle \varphi \rangle.\ A] \triangleq (\forall \langle \varphi \rangle.\ \forall v.\ \mathcal{H}_x\ [A[v/\varphi(x)]])\\
&\mathcal{H}_x\ [\exists \langle \varphi \rangle.\ A] \triangleq (\exists \langle \varphi \rangle.\ \exists v.\ \mathcal{H}_x\ [A[v/\varphi(x)]])
\end{aligned}
$$

Definition 15. ***Syntactic transformation for assume statements.***

$$
\begin{aligned}
&\Pi_p\ [b] \triangleq b\\
&\Pi_p\ [e_1 \geq e_2] \triangleq e_1 \geq e_2\\
&\Pi_p\ [A \wedge B] \triangleq \Pi_p\ [A] \wedge \Pi_p\ [B]\\
&\Pi_p\ [A \vee B] \triangleq \Pi_p\ [A] \vee \Pi_p\ [B]\\
&\Pi_p\ [\forall x.\ A] \triangleq \forall x.\ \Pi_p\ [A]\\
&\Pi_p\ [\exists x.\ A] \triangleq \exists x.\ \Pi_p\ [A]\\
&\Pi_p\ [\forall \langle \varphi \rangle.\ A] \triangleq \forall \langle \varphi \rangle.\ p(\varphi) \Rightarrow \Pi_p\ [A]\\
&\Pi_p\ [\exists \langle \varphi \rangle.\ A] \triangleq \exists \langle \varphi \rangle.\ p(\varphi) \wedge \Pi_p\ [A]
\end{aligned}
$$

B EXAMPLE OF A PROGRAM HYPERPROPERTY RELATING AN UNBOUNDED NUMBER OF EXECUTIONS

Given a program with a low-sensitivity (*low* for short) input l, a high-sensitivity (*high for short*) input h, and output o, an interesting problem is to *quantify* how much information about h is leaked through o. This information flow can be quantified with *min-capacity* [Assaf et al. 2017; Smith 2009], which boils down to quantifying the number of different values that the output o can have, given that the initial value of l is fixed (but the initial value of h is not). The problem (1) of *upper-bounding* the number of possible values of o is hypersafety, but not k-safety for any $k > 0$ [Yasuoka and Terauchi 2010]. This problem requires the ability to reason about an *unbounded* number of executions, which is not possible in any existing Hoare logic, but is possible in Hyper Hoare Logic. The harder problem (2) of both *lower-bounding* (to show that there is some leakage) and upper-bounding this quantity is not hypersafety anymore, and thus requires to be able to reason directly about properties of sets, in this case cardinality.

```
$o := 0$;
$i := 0$;
**while** $(i < max(l, h))$ {
    $r := nonDet()$;
    **assume** $0 \leq r \leq 1$;
    $o := o + r$
    $i := i + 1$
}
```

Fig. 10. The program C_l that leaks information about the high input h via its output o.

As an example, consider the program C_l shown in Fig. 10. Assuming that we know $h \geq 0$, the output o of this program can at most be h, hence leaking information about h: We learn that $h \geq o$. With respect to problem (1), we can express that this program can have *at most* $v + 1$ output values, where v is the initial value of l, with the hyper-triple

$$\{\square(h \geq 0) \land low(l)\}\ C_l\ \{\lambda S.\, \exists v.\, (\forall \varphi \in S.\, \varphi(l) = v) \land |\{\varphi(o) \mid \varphi \in S\}| \leq v\}$$

Moreover, with respect to the harder problem (2), we can express that this program can have *exactly* $v + 1$ output values, with the hyper-triple

$$\{\square(h \geq 0) \land low(l)\}\ C_l\ \{\lambda S.\, \exists v.\, (\forall \varphi \in S.\, \varphi(l) = v) \land |\{\varphi(o) \mid \varphi \in S\}| = v\}$$

C EXPRESSING JUDGMENTS OF HOARE LOGICS AS HYPER-TRIPLES

In this section, we demonstrate the expressivity of the logic by showing that hyper-triples can express the judgments of existing over- and underapproximating Hoare logics (App. C.1 and App. C.2) and enable reasoning about useful properties that go beyond over- and underapproximation (App. C.3). All theorems and propositions in this section have been proved in Isabelle/HOL.

C.1 Overapproximate Hoare Logics

The vast majority of existing Hoare logics prove the absence of bad (combinations of) program executions. To achieve that, they prove properties *for all* (combinations of) executions, that is, they overapproximate the set of possible (combinations of) executions. In this subsection, we discuss overapproximate logics that prove properties of single executions or of k executions (for a fixed number k), and show that Hyper Hoare Logic goes beyond them by also supporting properties of unboundedly or infinitely many executions.

Single executions. Classical Hoare Logic [Hoare 1969] is an overapproximate logic for properties of single executions (trace properties). The meaning of triples can be defined as follows:

Definition 16. ***Hoare Logic (HL).*** *Let P and Q be sets of extended states. Then*

$$\models_{HL} \{P\}\ C\ \{Q\} \triangleq (\forall \varphi \in P. \forall \sigma'. \langle C, \varphi^P \rangle \to \sigma' \Rightarrow (\varphi^L, \sigma') \in Q)$$

This definition reflects the standard partial correctness meaning of Hoare triples: executing C in some initial state that satisfies P can only lead to final states that satisfy Q. This meaning can be expressed as a program hyperproperty as defined in Def. 8:

Proposition 1. ***HL triples express hyperproperties.*** *Given sets of extended states P and Q, there exists a hyperproperty $\mathcal{H}$ such that, for all commands C, $C \in \mathcal{H}$ iff $\models_{HL} \{P\}\ C\ \{Q\}$.*

Proof sketch. We define

$$\mathcal{H} \triangleq \{C \mid \forall \varphi \in P. \forall \sigma'. (\varphi^P, \sigma') \in \Sigma(C) \Rightarrow (\varphi^L, \sigma') \in Q\}$$

and prove $\forall C. C \in \mathcal{H} \iff \models_{HL} \{P\}\ C\ \{Q\}$. □

This proposition together with completeness of our logic implies the *existence* of a proof in Hyper Hoare Logic for every valid classical Hoare triple. But there is an even stronger connection: we can map any assertion in classical Hoare logic to a hyper-assertion in Hyper Hoare Logic, which suggests a direct translation from classical Hoare logic to our Hyper Hoare Logic.

The assertions P and Q of a valid Hoare triple characterize *all* initial and *all* final states of executing a command C. Consequently, they represent *upper bounds* on the possible initial and final states. We can use this observation to map classical Hoare triples to hyper-triples by interpreting their pre- and postconditions as upper bounds on sets of states.

Proposition 2. ***Expressing HL in Hyper Hoare Logic.*** *Let $\overline{P} \triangleq (\lambda S. S \subseteq P)$. Then $\models_{HL} \{P\}\ C\ \{Q\}$ iff $\models \{\overline{P}\}\ C\ \{\overline{Q}\}$.*

Equivalently, $\models_{HL} \{P\}\ C\ \{Q\}$ iff $\models \{\forall \langle \varphi \rangle. \varphi \in P\}\ C\ \{\forall \langle \varphi \rangle. \varphi \in Q\}$.

This proposition implies that some rules of Hyper Hoare Logic have a direct correspondence in HL. For example, the rule *Seq* instantiated with $\overline{P}$, $\overline{R}$, and $\overline{Q}$ directly corresponds to the sequential composition rule from HL. Moreover, the upper-bound operator distributes over $\otimes$ and $\bigotimes$, since $\overline{A} \otimes \overline{B} = \overline{A \cup B}$, and $\bigotimes_i \overline{F_i} = \overline{\bigcup_i F(i)}$. Consequently, we can for example easily derive in Hyper Hoare Logic the classic while-rule from HL, using the rule *While* from Fig. 3.

k executions. Many extensions of HL have been proposed to deal with hyperproperties of k executions. As a representative of this class of logics, we relate Cartesian Hoare Logic [Sousa and Dillig 2016] to our Hyper Hoare Logic. To define the meaning of Cartesian Hoare Logic triples, we first lift our semantic relation $\rightarrow$ from one execution on states to k executions on extended states. Let $k \in \mathbb{N}^+$. We write $\overrightarrow{\varphi}$ to represent the k-tuple of extended states $(\varphi_1, \dots, \varphi_k)$, and $\forall \overrightarrow{\varphi}$ (resp. $\exists \overrightarrow{\varphi}$) as a shorthand for $\forall \varphi_1, \dots, \varphi_k$ (resp. $\exists \varphi_1, \dots, \varphi_k$). Moreover, we define the relation $\xrightarrow{k}$ as $\langle \overrightarrow{C}, \varphi \rangle \xrightarrow{k} \overrightarrow{\varphi'} \triangleq (\forall i \in [1,k].\, \langle C, {\varphi_i}^P \rangle \rightarrow {\varphi'}_i^P \;\wedge\; {\varphi_i}^L = {\varphi'}_i^L)$.

Definition 17. ***Cartesian Hoare Logic (CHL).*** *Let $k \in \mathbb{N}^+$, and let P and Q be sets of k-tuples of extended states. Then*

$$\models_{CHL(k)} \{P\}\ C\ \{Q\} \triangleq (\forall \overrightarrow{\varphi} \in P.\, \forall \overrightarrow{\varphi'}.\, \langle \overrightarrow{C}, \varphi \rangle \xrightarrow{k} \overrightarrow{\varphi'} \Rightarrow \overrightarrow{\varphi'} \in Q)$$

$\models_{CHL(k)} \{P\}\ C\ \{Q\}$ is valid iff executing C k times in k initial states that *together* satisfy P can only lead to k final states that together satisfy Q. This meaning can be expressed as a program hyperproperty:

Proposition 3. ***CHL triples express hyperproperties.*** *Given sets of k-tuples of extended states P and Q, there exists a hyperproperty $\mathcal{H}$ such that, for all commands C, $C \in \mathcal{H} \iff \models_{CHL(k)} \{P\}\ C\ \{Q\}$.*

Proof sketch. We define

$$\mathcal{H} \triangleq \{C \mid \forall \overrightarrow{\varphi} \in P.\, \forall \overrightarrow{\varphi'}.$$
$$(\forall i \in [1,k].\, \varphi_i^L = {\varphi'}_i^L \wedge (\varphi_i^P, {\varphi'}_i^P) \in \Sigma(C)) \Rightarrow \overrightarrow{\varphi'} \in Q\}$$

and prove $\forall C.\, C \in \mathcal{H} \iff \models_{CHL(k)} \{P\}\ C\ \{Q\}$. □

Like we did for Hoare Logic, we can provide a direct translation from CHL triples to hyper-triples in our logic. Similarly to HL, CHL assertions express upper bounds, here on sets of k-tuples. However, simply using upper bounds as in Prop. 2 does not capture the full expressiveness of CHL because executions in CHL are *distinguishable*. For example, one can express monotonicity from x to y as $\models_{CHL(k)} \{x(1) \geq x(2)\}\ y := x\ \{y(1) \geq y(2)\}$. When going from (ordered) tuples of states in CHL to (unordered) sets of states in Hyper Hoare Logic, we need to identify which state in the final set of states S corresponds to execution 1, and which state corresponds to execution 2. As we did in App. D.2 to express monotonicity, we use a logical variable t to tag a state with the number i of the execution it corresponds to.

Proposition 4. ***Expressing CHL in Hyper Hoare Logic.*** *Let*

$$P' \triangleq (\forall \overrightarrow{\varphi}.\, (\forall i \in [1,k].\, \langle \varphi_i \rangle \wedge \varphi_i^L(t) = i) \Rightarrow \overrightarrow{\varphi} \in P)$$
$$Q' \triangleq (\forall \overrightarrow{\varphi}.\, (\forall i \in [1,k].\, \langle \varphi_i \rangle \wedge \varphi_i^L(t) = i) \Rightarrow \overrightarrow{\varphi} \in Q)$$

where t does not occur free in P or Q. Then $\models_{CHL(k)} \{P\}\ C\ \{Q\} \iff\ \models \{P'\}\ C\ \{Q'\}$.

Recall that $\langle \varphi \rangle \triangleq (\lambda S.\, \varphi \in S)$. As an example, we can express the CHL assertion $y(1) \geq y(2)$ as the hyper-assertion $\forall \langle \varphi_1 \rangle, \langle \varphi_2 \rangle.\, \varphi_1^L(t) = 1 \wedge \varphi_2^L(t) = 2 \Rightarrow \varphi_1^P(y) \geq \varphi_2^P(y)$. Such translations provide a direct way of representing CHL proofs in Hyper Hoare Logic.

CHL, like Hyper Hoare Logic, can reason about multiple executions of a single command C, which is sufficient for many practically-relevant hyperproperties such as non-interference or determinism. Other logics, such as Relational Hoare Logic [Benton 2004], relate the executions of multiple (potentially different) commands, for instance, to prove program equivalence. In case these commands are all the same, triples of relational logics can be translated to Hyper Hoare Logic analogously to CHL. We explain how to encode relational properties relating different commands to Hyper Hoare Logic in App. C.3.

Unboundedly many executions. To the best of our knowledge, all existing overapproximate Hoare logics consider a fixed number k of executions. In contrast, Hyper Hoare Logic can reason about an unbounded number of executions, as we illustrate via the following example.

Consider a command C that encrypts a plaintext m using a secret key h and stores the result in an output variable x. We would like to prove that C is immune to known-plaintext attacks. That is, even though C leaks *some* information about the used key, it is not possible (assuming some computational limitations) to determine the key h from the plaintext m and the output x, no matter how often an attacker executes C.

In general, the more input-output pairs (m, x) an attacker observes, the more they learn about h, i.e., the fewer possibilities for h they have. We model this with a function f that takes the set of observed pairs (m, x) and returns the possibilities for h. We can then express that, for any number k of executions, an attacker cannot uniquely determine h:

$$\{\top\}\ C\ \{\lambda S.\, \forall k.\, \forall S' \subseteq S.\, |S'| \leq k \Rightarrow |f(\{(\varphi^P(m), \varphi^P(x)) | \varphi \in S'\})| > 1\}$$

This hyper-triple expresses a property over an unbounded number k of executions, which is not possible in existing Hoare logics. Since our hyper-assertions are functions of potentially-infinite sets of states, Hyper Hoare Logic can even express properties of infinitely-many executions, as we illustrate in App. C.3.

C.2 Underapproximate Hoare Logics

Several recent Hoare logics prove the *existence* of certain (combinations of) program executions, which is useful, for instance, to disprove a specification, that is, to demonstrate that a program definitely has a bug. These logics underapproximate the set of possible (combinations of) executions. In this subsection, we discuss two forms of underapproximate logics, *backward* and *forward*, and show that both can be expressed in Hyper Hoare Logic.

Backward underapproximation. Reverse Hoare Logic [de Vries and Koutavas 2011] and Incorrectness Logic [O'Hearn 2019] are both underapproximate logics. Reverse Hoare Logic is designed to reason about the reachability of good final states. Incorrectness Logic uses the same ideas to prove the presence of bugs in programs. We focus on Incorrectness Logic in the following, but our results also apply to Reverse Hoare Logic. Incorrectness Logic reasons about single program executions:

Definition 18. ***Incorrectness Logic (IL).*** *Let P and Q be sets of extended states. Then*

$$\models_{IL} \{P\}\ C\ \{Q\} \triangleq (\forall \varphi \in Q.\, \exists \sigma.\, (\varphi^L, \sigma) \in P \land \langle C, \sigma \rangle \rightarrow \varphi^P)$$

The meaning of IL triples is defined *backward* from the postcondition: any state that satisfies the postcondition Q can be reached by executing C in an initial state that satisfies the precondition P. This meaning can be expressed as a program hyperproperty:

Proposition 5. ***IL triples express hyperproperties.*** *Given sets of extended states P and Q, there exists a hyperproperty $\mathcal{H}$ such that, for all commands C, $C \in \mathcal{H}$ iff $\models_{IL} \{P\}\ C\ \{Q\}$.*

Proof sketch. We define

$$\mathcal{H} \triangleq \{C \mid \forall \varphi \in Q.\, \exists \sigma.\, (\varphi^L, \sigma) \in P \land (\sigma, \varphi^P) \in \Sigma(C)\}$$

and prove $\forall C.\, C \in \mathcal{H} \iff \models_{IL} \{P\}\ C\ \{Q\}$. □

Hoare Logic shows the absence of executions by overapproximating the set of possible executions, whereas Incorrectness Logic shows the existence of executions by underapproximating it. This duality also leads to an analogous translation of IL judgments into Hyper Hoare Logic, which uses lower bounds on the set of executions instead of the upper bounds used in Prop. 2.

Proposition 6. ***Expressing IL in Hyper Hoare Logic.*** *Let* $\underline{P} \triangleq (\lambda S.\, P \subseteq S)$*. Then* $\models_{IL} \{P\}\ C\ \{Q\}$ *iff* $\models \{\underline{P}\}\ C\ \{\underline{Q}\}$.
Equivalently, $\models_{IL} \{P\}\ C\ \{Q\}$ *iff* $\models \{\forall \varphi \in P.\, \langle\varphi\rangle\}\ C\ \{\forall \varphi \in Q.\, \langle\varphi\rangle\}$.

Analogous to the upper bounds for HL, the lower-bound operator distributes over $\otimes$ and $\bigotimes$: $\underline{A} \otimes \underline{B} = \underline{A \cup B}$ and $\bigotimes_i \underline{F_i} = \underline{\bigcup_i F(i)}$. Using the latter equality with the rules *While* and *Cons*, it is easy to derive the loop rules from both Incorrectness Logic and Reverse Hoare Logic.

Murray [2020] has recently proposed an underapproximate logic based on IL that can reason about two executions of two (potentially different) programs, for instance, to prove that a program violates a hyperproperty such as non-interference. We use the name *k-Incorrectness Logic* for the restricted version of this logic where the two programs are the same (and discuss relational properties between different programs in App. C.3). The meaning of triples in k-Incorrectness Logic is also defined backward. They express that, for any pair of final states (φ_1', φ_2') that together satisfy a relational postcondition, there exist two initial states φ_1 and φ_2 that together satisfy the relational precondition, and executing command C in φ_1 (resp. φ_2) leads to φ_1' (resp. φ_2'). Our formalization lifts this meaning from 2 to k executions:

Definition 19. ***k-Incorrectness Logic (k-IL).*** *Let* $k \in \mathbb{N}^+$*, and* P *and* Q *be sets of* k*-tuples of extended states. Then* $\models_{k-IL} \{P\}\ C\ \{Q\} \triangleq (\forall \overrightarrow{\varphi'} \in Q.\, \exists \overrightarrow{\varphi} \in P.\, \langle \overrightarrow{C}, \varphi\rangle \xrightarrow{k} \overrightarrow{\varphi'})$.

Again, this meaning is a hyperproperty:

Proposition 7. ***k-IL triples express hyperproperties.*** *Given sets of* k*-tuples of extended states* P *and* Q*, there exists a hyperproperty* $\mathcal{H}$ *such that, for all commands* C*,* $C \in \mathcal{H} \iff \models_{k-IL} \{P\}\ C\ \{Q\}$.

Proof sketch. We define

$$\mathcal{H} \triangleq \{C \mid \forall \overrightarrow{\varphi'} \in Q.\, \exists \overrightarrow{\varphi'} \in P.\\ (\forall i \in [1,k].\, \varphi_i^L = {\varphi'}_i^L \wedge (\varphi_i^P, {\varphi'}_i^P) \in \Sigma(C))\}$$

and prove $\forall C.\, C \in \mathcal{H} \iff \models_{k-IL} \{P\}\ C\ \{Q\}$. □

Together with Thm. 3, this implies that we can express any k-IL triple as hyper-triple in Hyper Hoare Logic. However, defining a direct translation of k-IL triples to hyper-triples is surprisingly tricky. In particular, it is *not* sufficient to apply the transformation from Prop. 4, which uses a logical variable t to tag each state with the number of the execution it belongs to. This approach works for Cartesian Hoare Logic because CHL and Hyper Hoare Logic are both forward logics (see Def. 5 and Def. 17). Intuitively, this commonality allows us to identify corresponding tuples from the preconditions in the two logics and relate them to corresponding tuples in the postconditions.

However, since k-IL is a *backward* logic, the same approach is not sufficient to identify corresponding tuples. For two final states φ_1' and φ_2' from the same tuple in the final set of states, we know through the tag variable t to which execution they belong, but not whether they originated from one tuple $(\varphi_1, \varphi_2) \in P$, or from two *unrelated* tuples.

To solve this problem, we use another logical variable u, which records the "identity" of the initial k-tuple that satisfies P. To avoid cardinality issues, we define the encoding under the assumption that P depends only on program variables. Consequently, there are at most $|PStates^k|$ such k-tuples, which we can represent as logical values if the cardinality of *LVals* is at least the cardinality of $PStates^k$, as shown by the following result:

Proposition 8. ***Expressing k-IL in Hyper Hoare Logic.*** *Let t, u be distinct variables in LVars and*

$$P' \triangleq (\forall \overrightarrow{\varphi} \in P. (\forall i \in [1,k].\, \varphi_i^L(t) = i) \Rightarrow (\exists v. \forall i \in [1,k].\, \langle \varphi_i[u := v] \rangle))$$
$$Q' \triangleq (\forall \overrightarrow{\varphi'} \in Q. (\forall i \in [1,k].\, {\varphi'}_i^L(t) = i) \Rightarrow (\exists v. \forall i \in [1,k].\, \langle \varphi'_i[u := v] \rangle))$$

If (1) P depends only on program variables, (2) the cardinality of LVals is at least the cardinality of $PStates^k$, and (3) t, u do not occur free in P or Q, then $\models_{k-IL} \{P\}\ C\ \{Q\} \iff \models \{P'\}\ C\ \{Q'\}$.

This proposition provides a direct translation for some k-IL triples into hyper-triples. Those that cannot be translated directly can still be verified in Hyper Hoare Logic, according to Prop. 7.

Forward underapproximation. Underapproximate logics can also be formulated in a forward way: Executing command C in any state that satisfies the precondition reaches at least one final state that satisfies the postcondition. Forward underapproximation has recently been explored in Outcome Logic [Zilberstein et al. 2023], a Hoare logic whose goal is to unify correctness (in the sense of classical Hoare logic) and incorrectness reasoning (in the sense of forward underapproximation) for single program executions. We focus on the underapproximation aspect of Outcome Logic here; overapproximation can be handled analogously to Hoare Logic (see App. C.1). Moreover, we restrict the discussion to the programming language defined in Sect. 3.1; Outcome Logic also supports heap-manipulating and probabilistic programs, which we do not consider here.

Forward underapproximation for single executions can be formalized as follows:

Definition 20. ***Forward Underapproximation (FU).*** *Let P and Q be sets of extended states. Then* $\models_{FU} \{P\}\ C\ \{Q\} \triangleq \left(\forall \varphi \in P. \exists \sigma'. \langle C, \varphi^P \rangle \to \sigma' \wedge (\varphi^L, \sigma') \in Q\right)$

This meaning can be expressed in Hyper Hoare Logic as follows: If we execute C in an initial set of states that contains at least one state from P then the final set of states will contain at least one state in Q.

Proposition 9. ***Expressing FU in Hyper Hoare Logic.***

$$\models_{FU} \{P\}\ C\ \{Q\} \iff \models \{\lambda S.\, P \cap S \neq \emptyset\}\ C\ \{\lambda S.\, Q \cap S \neq \emptyset\}$$

Equivalently, $\models_{FU} \{P\}\ C\ \{Q\}$ *iff* $\models \{\exists \langle \varphi \rangle.\, \varphi \in P\}\ C\ \{\exists \langle \varphi \rangle.\, \varphi \in Q\}$.

The precondition (resp. postcondition) states that the intersection between S and P (resp. Q) is non-empty. If instead it required that S is a *non-empty subset* of P (resp. Q), it would express the meaning of Outcome Logic triples, i.e., the conjunction of classical Hoare Logic and forward underapproximation.

While Outcome Logic reasons about single executions only, it is possible to generalize it to multiple executions:

Definition 21. ***k-Forward Underapproximation (k-FU).*** *Let $k \in \mathbb{N}^+$, and let P and Q be sets of k-tuples of extended states. Then* $\models_{k-FU} \{P\}\ C\ \{Q\} \triangleq (\forall \overrightarrow{\varphi} \in P. \exists \overrightarrow{\varphi'} \in Q. \langle \overrightarrow{C}, \varphi \rangle \xrightarrow{k} \overrightarrow{\varphi'})$.

Again, this meaning can be expressed as a hyperproperty:

Proposition 10. ***k-FU triples express hyperproperties.*** *Given sets of k-tuples of extended states P and Q, there exists a hyperproperty $\mathcal{H}$ such that, for all commands C, $C \in \mathcal{H} \iff \models_{k-FU} \{P\}\ C\ \{Q\}$.*

Proof sketch. We define

$$\mathcal{H} \triangleq \{C \mid \forall \overrightarrow{\varphi} \in P. \exists \overrightarrow{\varphi'} \in Q. \\ (\forall i \in [1,k]. \varphi_i^L = {\varphi'}_i^L \land (\varphi_i^P, {\varphi'}_i^P) \in \Sigma(C))\}$$

and prove $\forall C. C \in \mathcal{H} \iff \models_{k-FU} \{P\}\ C\ \{Q\}$. □

Since FU corresponds exactly to k-FU for $k = 1$, this proposition applies also to FU.

Because k-FU is *forward* underapproximate, we can use the tagging from Prop. 4 to translate k-FU triples into hyper-triples. The following encoding intuitively corresponds to the precondition $(S_1 \times \ldots \times S_k) \cap P \neq \emptyset$ and the postcondition $(S_1 \times \ldots \times S_k) \cap Q \neq \emptyset$, where S_i corresponds to the set of states with $t = i$:

Proposition 11. ***Expressing k-FU in Hyper Hoare Logic.*** *Let $P' \triangleq (\exists \overrightarrow{\varphi} \in P. \forall i \in [1,k]. \langle \varphi_i \rangle \land \varphi_i^L(t) = i)$ and $Q' \triangleq (\exists \overrightarrow{\varphi'} \in Q. \forall i \in [1,k]. \langle \varphi'_i \rangle \land {\varphi'}_i^L(t) = i)$. If t does not occur free in P or Q, then $\models_{k-FU} \{P\}\ C\ \{Q\} \iff \models \{P'\}\ C\ \{Q'\}$.*

C.3 Beyond Over- and Underapproximation

In the previous subsections, we have discussed overapproximate logics, which reason about *all* executions, and underapproximate logics, which reason about the *existence* of executions. In this subsection, we explore program hyperproperties that combine universal and existential quantification, as well as properties that apply other comprehensions to the set of executions. We also discuss relational properties about multiple programs (such as program equivalence).

∀∃-hyperproperties. Generalized non-interference (see Sect. 2.3) intuitively expresses that for each execution that produces a given observable output, there exists another execution that produces the same output using any other secret. That is, observing the output does not reveal any information about the secret. GNI is a hyperproperty that cannot be expressed in existing over- or underapproximate Hoare logics. It mandates the existence of an execution *based on other possible executions*, whereas underapproximate logics can show only the existence of (combinations of) executions that satisfy some properties, *independently of the other possible executions*. Generalized non-interference belongs to a broader class of ∀∃-hyperproperties.

RHLE [Dickerson et al. 2022] is a Hoare-style relational logic that has been recently proposed to verify ∀∃-relational properties, such as program refinement [Abadi and Lamport 1991]. We call the special case of RHLE where triples specify properties of multiple executions of the same command *k-Universal Existential*; we can formalize its triples as follows:

Definition 22. ***k-Universal Existential (k-UE).*** *Let $k_1, k_2 \in \mathbb{N}^+$, and let P and Q be sets of $(k_1 + k_2)$-tuples of extended states. Then*

$$\models_{k-UE(k_1,k_2)} \{P\}\ C\ \{Q\} \triangleq (\forall(\overrightarrow{\varphi}, \overrightarrow{\gamma}) \in P. \forall \overrightarrow{\varphi'}. \langle \overrightarrow{C}, \varphi \rangle \xrightarrow{k_1} \overrightarrow{\varphi'} \Rightarrow (\exists \overrightarrow{\gamma'}. \langle \overrightarrow{C}, \gamma \rangle \xrightarrow{k_2} \overrightarrow{\gamma'} \land (\overrightarrow{\varphi'}, \overrightarrow{\gamma'}) \in Q))$$

Given $k_1 + k_2$ initial states $\varphi_1, \ldots, \varphi_{k_1}$ and $\gamma_1, \ldots, \gamma_{k_2}$ that together satisfy the precondition P, for any final states $\varphi'_1, \ldots, \varphi'_{k_1}$ that can be reached by executing C in the initial states $\varphi_1, \ldots, \varphi_{k_1}$, there exist k_2 final states $\gamma'_1, \ldots, \gamma'_{k_2}$ that can be reached by executing C in the initial states $\gamma_1, \ldots, \gamma_{k_2}$, such that $\varphi'_1, \ldots, \varphi'_{k_1}, \gamma'_1, \ldots, \gamma'_{k_2}$ together satisfy the postcondition Q.

The properties expressed by k-UE assertions are hyperproperties:

Proposition 12. ***k-UE triples express hyperproperties.*** *Given sets of (k_1+k_2)-tuples of extended states P and Q, there exists a hyperproperty $\mathcal{H}$ such that, for all commands C, $C \in \mathcal{H} \iff \models_{k-UE(k_1,k_2)} \{P\}\ C\ \{Q\}$.*

Proof sketch. We define

$$\mathcal{H} \triangleq \{C \mid \forall(\overrightarrow{\varphi}, \overrightarrow{\gamma}) \in P. \forall\overrightarrow{\varphi'}.$$
$$\left(\forall i \in [1, k_1]. (\varphi_i^P, \varphi'^P_i) \in \Sigma(C) \land \varphi_i^L = \varphi'^L_i\right) \Rightarrow \exists\overrightarrow{\gamma'}.$$
$$(\overrightarrow{\varphi'}, \overrightarrow{\gamma'}) \in Q \land (\forall i \in [1, k_2]. (\gamma_i^P, \gamma'^P_i) \in \Sigma(C) \land \gamma_i^L = \gamma'^L_i)\}$$

and prove $\forall C. C \in \mathcal{H} \iff \models_{k-UE(k_1,k_2)} \{P\}\ C\ \{Q\}$. □

They can be directly expressed in Hyper Hoare Logic, as follows:

Proposition 13. ***Expressing k-UE in Hyper Hoare Logic.*** *Let t, u be distinct variables in LVars, and*

$$T_n \triangleq (\lambda\overrightarrow{\varphi}. \forall i \in [1, k_n]. \langle\varphi_i\rangle \land \varphi_i(t) = i \land \varphi_i(u) = n)$$
$$P' \triangleq (\forall i. \exists\langle\varphi\rangle. \varphi^L(t) = i \land \varphi^L(u) = 2) \land (\forall\overrightarrow{\varphi}, \overrightarrow{\gamma}. T_1(\overrightarrow{\varphi}) \land T_2(\overrightarrow{\gamma}) \Rightarrow (\overrightarrow{\varphi}, \overrightarrow{\gamma}) \in P)$$
$$Q' \triangleq (\forall\overrightarrow{\varphi'}. T_1(\varphi') \Rightarrow (\exists\overrightarrow{\gamma'}. T_2(\overrightarrow{\gamma'}) \land (\overrightarrow{\varphi'}, \overrightarrow{\gamma'}) \in Q))$$

where t, u do not occur free in P or Q. Then $\models_{k-UE(k_1,k_2)} \{P\}\ C\ \{Q\} \iff \models \{P'\}\ C\ \{Q'\}$.

This proposition borrows ideas from the translations of other logics we saw earlier. In particular, we use a logical variable t to tag the executions, and an additional logical variable u that indicates whether a state is universally ($u = 1$) or existentially ($u = 2$) quantified.

∃∀-hyperproperties. To the best of our knowledge, no existing Hoare logic can express ∃∀-hyperproperties, i.e., the *existence* of executions in relation to *all* other executions. As shown by the example in Sect. 3, ∃∀-hyperproperties naturally arise when disproving a ∀∃-hyperproperty (such as GNI), where the existential part can be thought of as a counter-example, and the universal part as the proof that this is indeed a counter-example. The existence of a minimum for a function computed by a command C is another simple example of an ∃∀-property, as shown in App. D.2.1.

Properties using other comprehensions. Some interesting program hyperproperties cannot be expressed by quantifying over states, but require other comprehensions over the set of states, such as counting or summation. As an example, the hyperproperty "there are exactly n different possible outputs for any given input" cannot be expressed by quantifying over the states, but requires counting. Other examples of such hyperproperties include statistical properties about a program:

Example 2. ***Mean number of requests.*** *Consider a command C that, given some input x, retrieves and returns information from a database. At the end of the execution of C, variable n contains the number of database requests that were performed. If the distribution of the inputs is restricted by the precondition P (e.g., the inputs are uniformly distributed), then the following hyper-triple expresses that the average number of requests performed by C is at most 2:*

$$\{P\}\ C\ \{\lambda S.\ mean_n^x(\{\varphi^P \mid \varphi \in S\}) \leq 2\}$$

where $mean_n^x$ computes the average (using a suitable definition for the average if the set is infinite) of the value of n based on the distribution of inputs x.

To the best of our knowledge, Hyper Hoare Logic is the only Hoare logic that can prove this property; existing logics neither support reasoning about mean-comprehensions over multiple execution states nor reasoning about infinitely many executions *at the same time* (which is necessary if the domain of input x is infinite).

Relational program properties. Relational program properties typically relate executions of several *different* programs and, thus, do not correspond to program hyperproperties as defined in Def. 8. However, it is possible to construct a single program that encodes the executions of several given programs, such that relational properties can be expressed as hyperproperties of the constructed program and proved in Hyper Hoare Logic.

We illustrate this approach on program refinement [Abadi and Lamport 1991]. A command C_2 *refines* a command C_1 iff the set of pairs of pre- and post-states of C_2 is a subset of the corresponding set of C_1. Program refinement is a $\forall\exists$-property, where the $\forall$ and the $\exists$ apply to different programs. To encode refinement, we construct a new program that non-deterministically executes either C_1 or C_2, and we track in a logical variable t which command was executed. This encoding allows us to express and prove refinement in Hyper Hoare Logic (under the assumption that the constructed program correctly reflects the executions of C_1 and C_2):

Example 3. ***Expressing program refinement in Hyper Hoare Logic.***
Let $C \triangleq (t := 1; C_1) + (t := 2; C_2)$. If t does not occur free in C_1 or C_2 then C_2 refines C_1 iff

$$\models \{\top\}\ C\ \{\forall\langle\varphi\rangle.\ \varphi^P(t) = 2 \Rightarrow \langle(\varphi^L, \varphi^P[t := 1])\rangle\}$$

This example illustrates a general methodology to transform a relational property over different programs into an equivalent hyperproperty for a new program, and thus to reason about relational program properties in Hyper Hoare Logic. Relational logics typically provide rules that align and relate parts of the different program executions; we present such a rule for Hyper Hoare Logic in App. H.

This section demonstrated that Hyper Hoare Logic is sufficiently expressive to prove and disprove arbitrary hyperproperties as defined in Def. 8. Thereby, it captures and goes beyond the properties handled by existing Hoare logics.

D COMPOSITIONALITY

$$\dfrac{\forall \varphi_1, \varphi_2.\left(\varphi_1^L = \varphi_2^L \land \vdash \{\langle \varphi_1 \rangle\}\ C\ \{\langle \varphi_2 \rangle\} \Longrightarrow \vdash \{P_{\varphi_1}\}\ C\ \{Q_{\varphi_2}\}\right)}{\vdash \{\forall \langle \varphi \rangle.\, P_\varphi\}\ C\ \{\forall \langle \varphi \rangle.\, Q_\varphi\}}\ (\mathit{Linking})$$

$$\dfrac{\vdash \{P_1\}\ C\ \{Q_1\} \quad \vdash \{P_2\}\ C\ \{Q_2\}}{\vdash \{P_1 \land P_2\}\ C\ \{Q_1 \land Q_2\}}\ (\mathit{And}) \qquad \dfrac{\vdash \{P_1\}\ C\ \{Q_1\} \quad \vdash \{P_2\}\ C\ \{Q_2\}}{\vdash \{P_1 \lor P_2\}\ C\ \{Q_1 \lor Q_2\}}\ (\mathit{Or})$$

$$\dfrac{\vdash \{P\}\ C\ \{Q\} \quad \text{no } \exists\langle\_\rangle \text{ in } F \quad wr(C) \cap rd(F) = \varnothing}{\vdash \{P \land F\}\ C\ \{Q \land F\}}\ (\mathit{FrameSafe})$$

$$\dfrac{\forall x.\, (\vdash \{P_x\}\ C\ \{Q_x\})}{\vdash \{\forall x.\, P_x\}\ C\ \{\forall x.\, Q_x\}}\ (\mathit{Forall}) \qquad \dfrac{\forall x.\, (\vdash \{P_x\}\ C\ \{Q_x\})}{\vdash \{\bigotimes_{x \in X} P_x\}\ C\ \{\bigotimes_{x \in X} Q_x\}}\ (\mathit{IndexedUnion})$$

$$\dfrac{\vdash \{P_1\}\ C\ \{Q_1\} \quad \vdash \{P_2\}\ C\ \{Q_2\}}{\vdash \{P_1 \otimes P_2\}\ C\ \{Q_1 \otimes Q_2\}}\ (\mathit{Union}) \qquad \dfrac{\vdash \{P\}\ C\ \{Q\}}{\vdash \{\bigotimes P\}\ C\ \{\bigotimes Q\}}\ (\mathit{BigUnion})$$

$$\dfrac{\vdash \{P\}\ C\ \{Q\} \quad wr(C) \cap rd(b) = \varnothing}{\vdash \{\Pi_b\,[P]\}\ C\ \{\Pi_b\,[Q]\}}\ (\mathit{Specialize})$$

$$\dfrac{P \Rightarrow^V P' \quad \vdash \{P'\}\ C\ \{Q\} \quad inv^V(Q)}{\vdash \{P\}\ C\ \{Q\}}\ (\mathit{LUpdate})$$

$$\dfrac{\vdash \{P \land (\forall \langle \varphi \rangle.\, \varphi(t) = e(\varphi))\}\ C\ \{Q\} \quad t \notin rd(P) \cup rd(Q) \cup fv(e)}{\vdash \{P\}\ C\ \{Q\}}\ (\mathit{LUpdateS})$$

$$\dfrac{\vdash \{P\}\ C\ \{Q\}}{\vdash \{\sqsubseteq P\}\ C\ \{\sqsubseteq Q\}}\ (\mathit{AtMost}) \qquad \dfrac{\vdash \{P\}\ C\ \{Q\}}{\vdash \{\sqsupseteq P\}\ C\ \{\sqsupseteq Q\}}\ (\mathit{AtLeast})$$

$$\dfrac{}{\vdash \{P\}\ C\ \{\top\}}\ (\mathit{True}) \qquad \dfrac{}{\vdash \{\bot\}\ C\ \{Q\}}\ (\mathit{False}) \qquad \dfrac{}{\vdash \{emp\}\ C\ \{emp\}}\ (\mathit{Empty})$$

Fig. 11. Compositionality rules of Hyper Hoare Logic. All these rules have been proven sound in Isabelle/HOL. $wr(C)$ corresponds to the set of program variables that are potentially written by C (i.e., that appear on the left-hand side of an assignment), while $rd(F)$ corresponds to the set of program variables that appear in look-up expressions for quantified states. For example, $rd(\forall\langle\varphi\rangle.\, \exists n.\, \varphi^P(x) = n^2) = \{x\}$. The operators $\bigotimes$, $\sqsubseteq$, and $\sqsupseteq$ are defined as follows: $\bigotimes P \triangleq (\lambda S.\, \exists F.\, (S = \bigcup_{S' \in F} S') \land (\forall S' \in F.\, P(S')))$, $\sqsubseteq P \triangleq (\lambda S.\, \exists S'.\, S \subseteq S' \land P(S'))$, and $\sqsupseteq P \triangleq (\lambda S.\, \exists S'.\, S' \subseteq S \Rightarrow P(S'))$.

The core rules of Hyper Hoare Logic allow one to prove any valid hyper-triple, but not necessarily *compositionally*, as explained in Sect. 3.6. As an example, consider the sequential composition of a command C_1 that satisfies *generalized* non-interference (GNI) with a command C_2 that satisfies non-interference (NI). We would like to prove that C_1; C_2 satisfies GNI (the weaker property). As discussed in Sect. 2.3, a possible postcondition for C_1 is $GNI_l^h \triangleq (\forall\langle\varphi_1\rangle, \langle\varphi_2\rangle.\, \exists\langle\varphi\rangle.\, \varphi_1^L(h) = \varphi^L(h) \land \varphi^P(l) = \varphi_2^P(l))$, while a possible precondition for C_2 is $low(l) \triangleq (\forall\langle\varphi_1\rangle, \langle\varphi_2\rangle.\, \varphi_1(l) = \varphi_2(l))$.

The corresponding hyper-triples for C_1 and C_2 cannot be composed using the core rules. In particular, rule *Seq* cannot be applied (even in combination with *Cons*), since the postcondition of C_1 does not imply the precondition of C_2. Note that this observation does *not* contradict completeness: By Thm. 2, it is possible to prove *more precise* triples for C_1 and C_2, such that the postcondition of C_1 matches the precondition of C_2. However, to enable modular reasoning, our goal is to construct the proof by composing the given triples for the individual commands rather than deriving new ones.

In this section, we present *compositionality rules* for hyper-triples (App. D.1). These rules are *admissible* in Hyper Hoare Logic, in the sense that they do not modify the set of valid hyper-triples that can be proved. Rather, these rules enable flexible compositions of hyper-triples (such as those discussed above). We illustrate these rules on two examples (App. D.2): Composing minimality with monotonicity, and GNI with NI. All technical results presented in this section (soundness of the rules shown in Fig. 11 and validity of the examples) have been formalized and proved in Isabelle/HOL.

D.1 Compositionality Rules

Fig. 11 shows a (selection of) compositionality rules for Hyper Hoare Logic, which we discuss below.

Linking. To prove hyper-triples of the form $\{\forall\langle\varphi_1\rangle.\, P_{\varphi_1}\}\ C\ \{\forall\langle\varphi_2\rangle.\, Q_{\varphi_2}\}$, the rule *Linking* considers each pair of pre-state φ_1 and post-state φ_2 separately, and lets one assume that φ_2 can be reached by executing C in the state φ_1, and that logical variables do not change during this execution.

Conjunctions and disjunctions. Hyper Hoare Logic admits the usual rules for conjunction (*And* and *Forall*) and disjunction (*Or* in Fig. 11, on top of the core rule *Exist* in Fig. 3).

Framing. Similarly to the frame rules in Hoare logic and separation logic [Reynolds 2002], Hyper Hoare Logic admits rules that allow us to frame information about states that is not affected by the execution of C. The rule *FrameSafe* allows us to frame any hyperassertion F if (1) it does not refer to variables that the program can modify, and (2) it does not existentially quantify over states. While (1) is standard, (2) is specific to hyper-assertions: Framing the existence of a state (e.g., with $F \triangleq \exists\langle\varphi\rangle.\, \top$) would be unsound if the execution of the program in the state φ does not terminate. We show in App. E that restriction (2) can be lifted if C terminates. We also show an example of how this rule is used in App. F.

Decompositions. As explained at the beginning of this section, the two triples $\{P\}\ C_1\ \{GNI^h_l\}$ and $\{low(l)\}\ C_2\ \{Q\}$ cannot be composed because GNI^h_l does not entail $low(l)$ (not all states in the set S of final states of C_1 need to have the same value for l). However, we can prove GNI for the composed commands by decomposing S into subsets that all satisfy $low(l)$ and considering each subset separately. The rule *BigUnion* allows us to perform this decomposition (formally expressed with the hyper-assertion $\bigotimes low(l)$), use the specification of C_2 on each of these subsets (since they all satisfy the precondition of C_2), and eventually recompose the final set of states (again with the operator $\bigotimes$) to prove our desired postcondition. Hyper Hoare Logic also admits rules for binary unions (rule *Union*) and indexed unions (rule *IndexedUnion*).

Note that unions ($\otimes$ and $\bigotimes$) and disjunctions in hyper-assertions are very *different*: $(P \otimes Q)(S)$ expresses that the set S can be decomposed into two sets S_P (satisfying P) and S_Q (satisfying Q), while $(P \vee Q)(S)$ expresses that the entire set S satisfies P or Q. Similarly, intersections and conjunctions are very different: While Hyper Hoare Logic admits conjunction rules, rules based on intersections would be unsound, as shown by the following example:

Example 4. *Let $P_1 \triangleq (\lambda S.\, \exists \varphi.\, S = \{\varphi\} \land \varphi(x) = 1)$, and $P_2 \triangleq (\lambda S.\, \exists \varphi.\, S = \{\varphi\} \land \varphi(x) = 2)$. Both triples $\{P_1\}$ $x := 1$ $\{P_1\}$ and $\{P_2\}$ $x := 1$ $\{P_1\}$ are valid, but the triple*

$$\{\lambda S.\, \exists S_1, S_2.\, S = S_1 \cap S_2 \land P_1(S_1) \land P_2(S_2)\}\ x := 1\ \{\lambda S.\, \exists S_1, S_2.\, S = S_1 \cap S_2 \land P_1(S_1) \land P_1(S_2)\}$$

is invalid, as the precondition is equivalent to emp, but the postcondition is satisfiable by a non-empty set (with states satisfying $x = 1$).

Specializing hyper-triples. By definition, a hyper-triple can only be applied to a set of states that satisfies its precondition, which can be restrictive. In cases where only a *subset* of the current set of states satisfies the precondition, one can obtain a *specialized* triple using the rule *Specialize*. This rule uses the syntactic transformation Π_b defined in Sect. 4.3 to weaken both the precondition and the postcondition of the triple, which is sound as long as the validity of b is not influenced by executing C. Intuitively, $\Pi_b\ [P]$ holds for a set S iff P holds for the subset of states from S that satisfy b. As an example, the triple $\{\Box(t{=}1 {\Rightarrow} x{\geq}0) {\land} \Box(t{=}2 {\Rightarrow} x{<}0)\}$ C $\{\forall \langle\varphi_1\rangle, \langle\varphi_2\rangle.\, \varphi_1(t){=}1 {\land} \varphi_2(t){=}2 {\Rightarrow} \varphi_1(y) {\geq} \varphi_2(y)\}$, whose postcondition corresponds to $mono^t_y$, can be derived from the two triples $\{\Box(x \geq 0)\}$ C $\{\Box(y \geq 0)\}$ and $\{\Box(x < 0)\}$ C $\{\Box(y < 0)\}$, by applying the rule *Specialize* twice, using $b \triangleq (t{=}1)$ and $b \triangleq (t{=}2)$ respectively, followed by the consequence rule.

Logical updates. Logical variables play an important role in the expressivity of the logic: As we have informally shown in Sect. 2.2, and as we formally show in App. C, relational specifications are typically expressed in Hyper Hoare Logic by using logical variables to formally link the pre-state of an execution with the corresponding post-states. Since logical variables cannot be modified by the execution, these tags are preserved.

To apply this proof strategy with existing triples, it is often necessary to update logical variables to introduce such tags. The rule *LUpdate* allows us to update the logical variables in a set V, provided that (1) from every set of states S that satisfies P, we can obtain a new set of states S' that satisfies P', by only updating (for each state) the logical variables in V, (2) we can prove the triple with the updated set of initial states, and (3) the postcondition Q cannot distinguish between two sets of states that are equivalent up to logical variables in V. We formalize this intuition in the following:

Definition 23. ***Logical updates.*** *Let V be a set of logical variable names. Two states φ_1 and φ_2 are equal up to logical variables V, written $\varphi_1 \overset{V}{=} \varphi_2$, iff $\forall i.\, i \notin V \Rightarrow \varphi_1^L(i) = \varphi_2^L(i)$ and $\varphi_1^P = \varphi_2^P$.*

Two sets of states S_1 and S_2 are equivalent up to logical variables V, written $S_1 \overset{V}{=} S_2$, iff every state $\varphi_1 \in S_1$ has a corresponding state $\varphi_2 \in S_2$ with the same values for all variables except those in V, and vice-versa:

$$(\forall \varphi_1 \in S_1.\, \exists \varphi_2 \in S_2.\, \varphi_1 \overset{V}{=} \varphi_2) \land (\forall \varphi_2 \in S_2.\, \exists \varphi_1 \in S_1.\, \varphi_1 \overset{V}{=} \varphi_2)$$

A hyper-assertion P entails a hyper-assertion P' modulo logical variables V, written $P \overset{V}{\Rightarrow} P'$, iff

$$\forall S.\, P(S) \Longrightarrow (\exists S'.\, P'(S') \land S \overset{V}{=} S')$$

Finally, a hyper-assertion P is invariant with respect to logical updates in V, written $inv^V(P)$, iff

$$\forall S_1, S_2.\, S_1 \overset{V}{=} S_2 \Longrightarrow (P(S_1) \Longleftrightarrow P(S_2))$$

Note that $inv^V(Q)$ means that Q cannot inspect the value of logical variables in V, but it usually also implies that Q cannot check for *equality* between states, and cannot inspect the cardinality of the set, since updating logical variables might collapse two states that were previously distinct (because of distinct values for logical variables in V).

$$\dfrac{\dfrac{\vdash \{\mathit{isSingleton}\}\ C_2\ \{\mathit{isSingleton}\}}{\vdash \{\Pi_{i=1}\,[\mathit{isSingleton}]\}\ C_2\ \{\Pi_{i=1}\,[\mathit{isSingleton}]\}}\,(\mathit{Specialize})}{\vdash \{\underbrace{\Pi_{i=1}\,[\mathit{isSingleton}] \land (\forall\langle\varphi\rangle.\ \varphi^L(i) \in \{1,2\})}_{P'}\}\ C_2\ \{\underbrace{\Pi_{i=1}\,[\mathit{isSingleton}] \land (\forall\langle\varphi\rangle.\ \varphi^L(i) \in \{1,2\})}_{Q'}\}}\,(\mathit{FrameSafe}) \qquad (1)$$

$$\dfrac{\vdash \{P\}\ C_1\ \{\mathit{hasMin}_x\} \qquad \dfrac{\mathit{hasMin}_x \overset{\{i\}}{\Rightarrow} \mathit{mono}^i_x \land P' \qquad \dfrac{\dfrac{\vdash \{\mathit{mono}^i_x\}\ C_2\ \{\mathit{mono}^i_y\} \qquad \dfrac{(1)}{\vdash \{P'\}\ C_2\ \{Q'\}}}{\vdash \{\mathit{mono}^i_x \land P'\}\ C_2\ \{\mathit{mono}^i_y \land Q'\}}\,(\mathit{And})}{\vdash \{\mathit{mono}^i_x \land P'\}\ C_2\ \{\mathit{hasMin}_y\}}\,(\mathit{Cons}) \qquad \mathit{inv}^{\{i\}}(\mathit{hasMin}_y)}{\vdash \{\mathit{hasMin}_x\}\ C_2\ \{\mathit{hasMin}_y\}}\,(\mathit{LUpdate})}{\vdash \{P\}\ C_1; C_2\ \{\mathit{hasMin}_y\}}\,(\mathit{Seq})$$

Fig. 12. A compositional proof that the sequential composition of a command that has a minimum and a monotonic, deterministic command in turn has a minimum. Recall that $\mathit{isSingleton} \triangleq (\exists\langle\varphi\rangle.\ \forall\langle\varphi'\rangle.\ \varphi = \varphi')$, and thus $\Pi_{i=1}\,[\mathit{isSingleton}] = (\exists\langle\varphi\rangle.\ \varphi(i) = 1 \land (\forall\langle\varphi'\rangle.\ \varphi'(i) = 1 \Rightarrow \varphi = \varphi'))$

Since this rule requires semantic reasoning, we also derive a weaker syntactic version of this rule, *LUpdateS*, which is easier to use. The rule *LUpdateS* allows us to strengthen a precondition P to $P \land (\forall\langle\varphi\rangle.\ \varphi(t) = e(\varphi))$, which corresponds to updating the logical variable t with the expression e, as long as the logical variable t does not appear *syntactically* in P, Q, and e (and thus does not influence their validity). For example, to connect the postcondition $\square(x = 0 \lor x = 1)$ to the precondition $\mathit{mono}^t_x = (\forall\langle\varphi_1\rangle, \langle\varphi_2\rangle.\ \varphi_1(t) = 1 \land \varphi_2(t) = 2 \Rightarrow \varphi_1(x) \geq \varphi_2(x))$ described in Sect. 2.2, one can use this rule to assign 1 to t if $x = 1$, and 2 otherwise. App. F shows a detailed example.

D.2 Examples

We now illustrate our compositionality rules on two examples: Composing minimality and monotonicity, and composing strong and generalized non-interference.

D.2.1 Composing Minimality and Monotonicity. Consider a command C_1 that computes a function that has a minimum for x, and a deterministic command C_2 that is monotonic from x to y. We want to prove *compositionally* that $C_1; C_2$ has a minimum for y.

More precisely, we assume that C_1 satisfies the specification $\{P\}\ C_1\ \{\mathit{hasMin}_x\}$, where $\mathit{hasMin}_x \triangleq (\exists\langle\varphi\rangle.\ \forall\langle\varphi'\rangle.\ \varphi^P(x) \leq \varphi'^P(x))$, and C_2 satisfies the two specifications $\{\mathit{mono}^i_x\}\ C_2\ \{\mathit{mono}^i_y\}$ (monotonicity) and $\{\mathit{isSingleton}\}\ C_2\ \{\mathit{isSingleton}\}$ (determinism[13]), where $\mathit{mono}^i_x \triangleq (\forall\langle\varphi_1\rangle, \langle\varphi_2\rangle.\ \varphi^L_1(i) = 1 \land \varphi^L_2(i) = 2 \Rightarrow \varphi^P_1(x) \leq \varphi^P_2(x))$, and $\mathit{isSingleton} \triangleq (\exists\langle\varphi\rangle.\ \forall\langle\varphi'\rangle.\ \varphi = \varphi')$. With the core rules alone, we cannot compose the two triples to prove that $C_1; C_2$ has a minimum for y since the postcondition of C_1 does not imply the precondition of C_2.

Fig. 12 shows a valid derivation in Hyper Hoare Logic of $\vdash \{P\}\ C_1; C_2\ \{\mathit{hasMin}_y\}$ (which we have proved in Isabelle/HOL). The key idea is to use the rule *LUpdate* to mark the minimal state with $i = 1$, and all the other states with $i = 2$, in order to match C_1's postcondition with C_2's precondition. Note that we *had to* use the consequence rule to turn C_2's postcondition $\mathit{mono}^i_y \land Q'$ into hasMin_y *before* applying the rule *LUpdate*, because the latter hyper-assertion is invariant w.r.t. logical updates in $\{i\}$ (as required by the rule *LUpdate*), whereas the former is not.

[13]This triple ensures that C_2 does not map the initial state with the minimum value for x to potentially different states with incomparable values for y (the order $\leq$ on values might be partial). Moreover, it ensures that C_2 does not drop any initial states because of an assume command or a non-terminating loop.

$$\dfrac{\dfrac{\vdash \{low(l)\}\ C_2\ \{low(l)\} \qquad \dfrac{\dfrac{\vdash \{\neg emp\}\ C_2\ \{\neg emp\}}{\vdash \{\Pi_{h=\varphi_1^L(h)}\ [\neg emp]\}\ C_2\ \{\Pi_{h=\varphi_1^L(h)}\ [\neg emp]\}}\ (\mathit{Specialize})}{\vdash \{\exists\langle\varphi\rangle.\ \varphi^L(h) = \varphi_1^L(h)\}\ C_2\ \{\exists\langle\varphi\rangle.\ \varphi^L(h) = \varphi_1^L(h)\}}\ (\mathit{Cons})}{\dfrac{\vdash \{low(l) \land (\exists\langle\varphi\rangle.\ \varphi^L(h) = \varphi_1^L(h))\}\ C_2\ \{low(l) \land (\exists\langle\varphi\rangle.\ \varphi^L(h) = \varphi_1^L(h))\}}{\vdash \{\bigotimes \left(low(l) \land (\exists\langle\varphi\rangle.\ \varphi^L(h) = \varphi_1^L(h))\right)\}\ C_2\ \{\bigotimes \left(low(l) \land (\exists\langle\varphi\rangle.\ \varphi^L(h) = \varphi_1^L(h))\right)\}}\ (\mathit{BigUnion})}\ (\mathit{And})}{\vdash \{\underbrace{\forall\langle\varphi_2\rangle.\ \exists\langle\varphi\rangle.\ \varphi_1^L(h) = \varphi^L(h) \land \varphi_2^P(l) = \varphi^P(l)}_{P'_{\varphi_1}}\}\ C_2\ \{\underbrace{\forall\langle\varphi_2\rangle.\ \exists\langle\varphi\rangle.\ \varphi_1^L(h) = \varphi^L(h) \land \varphi_2^P(l) = \varphi^P(l)}_{Q'_{\varphi_1}}\}}\ (\mathit{Cons}) \tag{2}$$

$$\dfrac{\vdash \{low(l)\}\ C_1\ \{GNI_l^h\} \qquad \dfrac{\dfrac{\text{using } (2) \text{ and } \varphi_1^L = \varphi_1'^L \Longrightarrow Q'_{\varphi_1} = Q'_{\varphi_1'}}{\forall\varphi_1, \varphi_1'.\ (\varphi_1^L = \varphi_1'^L \land \vdash \{\langle\varphi_1\rangle\}\ C\ \{\langle\varphi_1'\rangle\} \Longrightarrow (\vdash \{P'_{\varphi_1}\}\ C_2\ \{Q'_{\varphi_1'}\})}}{\vdash \{GNI_l^h\}\ C_2\ \{GNI_l^h\}}\ (\mathit{Linking})}{\vdash \{low(l)\}\ C_1; C_2\ \{GNI_l^h\}}\ (\mathit{Seq})$$

Fig. 13. A compositional proof that the sequential composition of a command that satisfies GNI and a command that satisfies NI in turn satisfies GNI.

The upper part of Fig. 12 shows the derivation of $\vdash \{P'\}\ C_2\ \{Q'\}$, which uses *Specialize* to restrict the triple $\{isSingleton\}\ C_2\ \{isSingleton\}$ to the subset of states where $i = 1$, ensuring the existence of a unique state (the minimum) where $i = 1$ after executing C_2. We also use the rule *FrameSafe* to ensure that our set only contains states with $i = 1$ or $i = 2$.

D.2.2 Composing Generalized and Strong Non-Interference. To illustrate additional compositionality rules, we re-visit the example introduced at the beginning of this section. Consider a command C_1 that satisfies GNI (for a public variable l and a secret variable h) and a command C_2 that satisfies NI (for the public variable l). We want to prove that $C_1; C_2$ satisfies GNI (for l and h).

More precisely, we assume that C_1 satisfies the hyper-triple $\vdash \{low(l)\}\ C_1\ \{GNI_l^h\}$, where $GNI_l^h \triangleq (\forall\langle\varphi_1\rangle, \langle\varphi_2\rangle.\ \exists\langle\varphi\rangle.\ \varphi_1^L(h) = \varphi^L(h) \land \varphi^P(l) = \varphi_2^P(l))$. Moreover, we assume that C_2 satisfies the triples $\vdash \{low(l)\}\ C_2\ \{low(l)\}$, where $low(l) \triangleq (\forall\langle\varphi_1\rangle, \langle\varphi_2\rangle.\ \varphi_1^P(l) = \varphi_2^P(l))$, and $\vdash \{\neg emp\}\ C_2\ \{\neg emp\}$. The second triple is needed to ensure that C_2 does not drop executions depending on some values for h (e.g., because of secret-dependent non-termination), which might cause $C_1; C_2$ to violate GNI.

Fig. 13 shows a valid derivation of the triple $\vdash \{low(l)\}\ C_1; C_2\ \{GNI_l^h\}$ (which we have proved in Isabelle/HOL). The first key idea of this derivation is to use the rule *Linking* to eliminate the $\forall\langle\varphi_1\rangle$ in the pre- and postcondition of the triple $\{GNI_l^h\}\ C_2\ \{GNI_l^h\}$, while assuming that they have the same value for the logical variable h (implied by the assumption $\varphi_1^L = \varphi_1'^L$). The second key idea is to decompose any set of states S that satisfies P'_{φ_1} (defined as $\forall\langle\varphi_2\rangle.\ \exists\langle\varphi\rangle.\ \varphi_1^L(h) = \varphi^L(h) \land \varphi_2^P(l) = \varphi^P(l)$) into a union of smaller sets that all satisfy $low(l) \land (\exists\langle\varphi\rangle.\ \varphi_1^L(h) = \varphi^L(h))$. More precisely, we rewrite S as the union of all sets $\{\varphi, \varphi_2\}$ for all $\varphi, \varphi_2 \in S$ such that $\varphi_1^L(h) = \varphi^L(h) \land \varphi_2^P(l) = \varphi^P(l)$, using the rule *Cons*. Unlike S, these smaller sets all satisfy the precondition $low(l)$ of C_2, which allows us to leverage the triple $\vdash \{low(l)\}\ C_2\ \{low(l)\}$. Finally, we use the rule *Specialize* to prove that, after executing C_2 in each of the smaller sets $\{\varphi, \varphi_2\}$, there will exist at least one state φ' with $\varphi'^L(h) = \varphi_1^L(h)$.

E TERMINATION-BASED REASONING

E.1 Termination-Based Rules

In App. D, we have introduced the rule *FrameSafe* (Fig. 4), which is sound only for hyper-assertions that do not contain any $\exists\langle\_\rangle$, because the program C around which we want to frame some hyper-assertion might not terminate. Moreover, in Sect. 5.1, we have introduced the synchronized while rule *WhileSync* (Fig. 6), which contains a *emp* disjunct in the postcondition of the conclusion, which prevents this rule from being useful to prove hyperproperties of the form $\exists^+\forall^*$, i.e., with a top-level existential quantifier over state. This *emp* disjunct corresponds to the case where the loop terminates.

In this section, we show that we can overcome those two limitations by introducing *total* hyper-triples, which are stronger than normal hyper-triples, in that they also ensure the existence of at least one terminating execution for any initial state:

Definition 24. ***Total hyper-triples.***

$$\models_{\Downarrow} \{P\}\ C\ \{Q\} \triangleq \Big(\forall S.\, P(S) \Rightarrow (Q(sem(C,S)) \land (\forall \varphi \in S.\, \exists \sigma'.\, \langle C, \varphi^P\rangle \rightarrow \sigma'))\Big)$$

For any program statement C that does not contain any **assume** statement, both triples are equivalent: $\models_{\Downarrow} \{P\}\ C\ \{Q\} \Longleftrightarrow \models \{P\}\ C\ \{Q\}$.

Using total hyper-triples, we can now express and prove sound (which we have done in Isabelle) the following rules, which solve the aforementioned limitations:

$$\frac{wr(C) \cap fv(F) = \varnothing \qquad \vdash_{\Downarrow} \{P\}\ C\ \{Q\} \qquad F \text{ is a syntactic hyper-assertion}}{\vdash_{\Downarrow} \{P \land F\}\ C\ \{Q \land F\}}\ (\mathit{Frame})$$

$$\frac{\vdash_{\Downarrow} \{I \land \Box(b \land e = t^L)\}\ C\ \{I \land low(b) \land \Box(e \prec t^L)\} \qquad \prec \text{well-founded} \qquad t^L \notin rd(I)}{\vdash_{\Downarrow} \{I \land low(b)\}\ \textbf{while}\ (b)\ \{C\}\ \{I \land \Box(\neg b)\}}\ (\mathit{WhileSyncTot})$$

As can be seen, the rule *Frame* can be used for *any* hyper-assertion expressed in the syntax defined in Sect. 4.1. Unlike the rule *WhileSync*, the rule *WhileSyncTot* does not have the *emp* disjunct in the postcondition of its conclusion anymore, and thus can be used to prove hyperproperties of the form $\exists^+\forall^*$! It achieves this by requiring that (1) the loop body C terminates (in the sense of Def. 24), and (2) that the loop itself terminates, by requiring that a variant e decreases in all executions. The initial value of the variant e is stored in the logical variable t^L, such that it can be referred to in the postcondition. Note that we can prove a total variant of each loop rule presented in Sect. 5, by doing something similar as point (2) here, in order to obtain a complete proof system for total hyper-triples.

E.2 (Dis-)Proving Termination

Hyper Hoare Logic in its current version is a "partial correctness" logic, in the sense that it proves (hyper)properties about the set of *terminating* executions. By slightly strengthening the definition of total hyper-triples (Def. 24) such that *all* executions are required to terminate, we could obtain a "total correctness" version of Hyper Hoare Logic, with which we can prove that all considered executions terminate. Note that, even with this stronger definition, the rules *Frame* and *WhileSyncTot* would stay the same.

Notably, HHL could also be extended to disprove termination. To prove *non*-termination of a loop **while** (b) $\{C\}$, one can express and prove that a set of states R, in which all states satisfy the loop guard b, is a *recurrent set* [Gupta et al. 2008]. R is a recurrent set iff executing C in any state from R leads to at least another state in R, which can easily be expressed as a hyper-triple:

$$\{\exists\langle\varphi\rangle.\, \varphi \in R\}\ C\ \{\exists\langle\varphi\rangle.\, \varphi \in R\}$$

Thus, if one state from R reaches **while** (b) $\{C\}$, we know that there is at least one non-terminating execution.

Note that both extensions of Hyper Hoare Logic (to prove and disprove termination) would require modifying the underlying semantic model of the logic; in particular, the extended semantics in Def. 4 should be modified to also capture non-terminating executions. We do not expect such a modification to pose any significant challenge.

F FIBONACCI EXAMPLE

In this section, we show the proof that the program C_{fib} from Fig. 8 is monotonic. Precisely, we prove the triple

$$\vdash \{\forall\langle\varphi_1\rangle,\langle\varphi_2\rangle.\ \varphi_1(t)=1\wedge\varphi_2(t)=2\Longrightarrow\varphi_1(n)\geq\varphi_2(n)\}\ C_{fib}\ \{\forall\langle\varphi_1\rangle,\langle\varphi_2\rangle.\ \varphi_1(t)=1\wedge\varphi_2(t)=2\Longrightarrow\varphi_1(a)\geq\varphi_2(a)\}$$

using the rule *While*-$\forall^*\exists^*$ with the loop invariant $I \triangleq ((\forall\langle\varphi_1\rangle,\langle\varphi_2\rangle.\ \varphi_1(t)=1\wedge\varphi_2(t)=2 \Rightarrow (\varphi_1(n)-\varphi_1(i) \geq \varphi_2(n)-\varphi_2(i) \wedge \varphi_1(a) \geq \varphi_2(a) \wedge \varphi_1(b) \geq \varphi_2(b))) \wedge \Box(b \geq a \geq 0))$.

$\{\forall\langle\varphi_1\rangle,\langle\varphi_2\rangle.\ \varphi_1(t)=1\wedge\varphi_2(t)=2\Longrightarrow\varphi_1(n)\geq\varphi_2(n)\}$

$\{(\forall\langle\varphi_1\rangle,\langle\varphi_2\rangle.\ \varphi_1(t)=1\wedge\varphi_2(t)=2 \Rightarrow (\varphi_1(n)-0 \geq \varphi_2(n)-0 \wedge 0 \geq 0 \wedge 1 \geq 1)) \wedge \Box(1 \geq a \geq 0)\}$ (Cons)

$a := 0;$

$\{(\forall\langle\varphi_1\rangle,\langle\varphi_2\rangle.\ \varphi_1(t)=1\wedge\varphi_2(t)=2 \Rightarrow (\varphi_1(n)-0 \geq \varphi_2(n)-0 \wedge \varphi_1(a) \geq \varphi_2(a) \wedge 1 \geq 1)) \wedge \Box(1 \geq a \geq 0)\}$ (AssignS)

$b := 1;$

$\{(\forall\langle\varphi_1\rangle,\langle\varphi_2\rangle.\ \varphi_1(t)=1\wedge\varphi_2(t)=2 \Rightarrow (\varphi_1(n)-0 \geq \varphi_2(n)-0 \wedge \varphi_1(a) \geq \varphi_2(a) \wedge \varphi_1(b) \geq \varphi_2(b))) \wedge \Box(b \geq a \geq 0)\}$ (AssignS)

$i := 0;$

$\{(\forall\langle\varphi_1\rangle,\langle\varphi_2\rangle.\ \varphi_1(t)=1\wedge\varphi_2(t)=2 \Rightarrow (\varphi_1(n)-\varphi_1(i) \geq \varphi_2(n)-\varphi_2(i) \wedge \varphi_1(a) \geq \varphi_2(a) \wedge \varphi_1(b) \geq \varphi_2(b))) \wedge \Box(b \geq a \geq 0)\}$ (AssignS)

Fig. 14. First part of the proof, which proves that the loop invariant I holds before the loop.

Fig. 14 shows the (trivial) first part of the proof, which proves that the loop invariant I holds before the loop, and Fig. 15 shows the proof of $\vdash \{I\}$ **if** $(i < n)$ $\{C_{body}\}$ $\{I\}$, the fist premise of the rule *While*-$\forall^*\exists$ (the second premise is trivial). In Fig. 15, we first record the initial values of a, b, and i in the logical variables v_a, v_b, and v_i, respectively, using the rule *LUpdateS* presented in App. D. We then split our new hyper-assertion into a simple part, $\forall\langle\varphi\rangle.\ \varphi(i) = \varphi(v_i) \wedge \varphi(a) = \varphi(v_a) \wedge \varphi(b) = \varphi(v_b)$, and a frame F which stores the relevant information from the invariant I with the initial values. This frame is then framed around the if-statement, using the rule *FrameSafe* from App. D. The proof of the branches is straightforward; the postconditions of the two branches are combined via the rule *Choice*.

We finally conclude with the consequence rule. This last entailment is justified by a case distinction. Let φ_1, φ_2 be two states such that $\varphi_1(t) = 1$, $\varphi_2(t) = 2$, and $\langle\varphi_1\rangle$ and $\langle\varphi_2\rangle$ hold. From the frame F, we know that $\varphi_1(v_a) \geq \varphi_2(v_a)$, and $\varphi_1(v_b) \geq \varphi_2(v_b)$. We conclude the proof by distinguishing the following three cases (the proof for each case is straightforward): (1) Both φ_1 and φ_2 took the then branch of the if statement, i.e., $\varphi_1(v_i) < \varphi_1(n)$ and $\varphi_2(v_i) < \varphi_2(n)$, and thus both are in the set characterized by Q_1. (2) Both φ_1 and φ_2 took the else branch, i.e., $\varphi_1(v_i) \geq \varphi_1(n)$ and $\varphi_2(v_i) \geq \varphi_2(n)$. and thus both are in the set characterized by Q_2. (3) φ_1 took the then branch and φ_2 took the else branch, i.e., $\varphi_1(v_i) < \varphi_1(n)$ and $\varphi_2(v_i) \geq \varphi_2(n)$, and thus φ_1 is in the set characterized by Q_1 and φ_2 is in the set characterized by Q_2.

Importantly, the fourth case is not possible, because this would imply $\varphi_2(n) - \varphi_2(v_i) > 0 \geq \varphi_1(n) - \varphi_1(v_i)$, which contradicts the inequality $\varphi_1(n) - \varphi_1(v_i) \geq \varphi_2(n) - \varphi_2(v_i)$ from the frame F.

```
$\{\forall \langle\varphi_1\rangle, \langle\varphi_2\rangle.\ \varphi_1(t)=1 \land \varphi_2(t)=2 \Rightarrow (\varphi_1(n)-\varphi_1(i) \geq \varphi_2(n)-\varphi_2(i) \land \varphi_1(a) \geq \varphi_2(a) \land \varphi_1(b) \geq \varphi_2(b)) \land \Box(b \geq a \geq 0)\}$
$\{\forall \langle\varphi_1\rangle, \langle\varphi_2\rangle.\ \varphi_1(t)=1 \land \varphi_2(t)=2 \Rightarrow (\varphi_1(n)-\varphi_1(i) \geq \varphi_2(n)-\varphi_2(i) \land \varphi_1(a) \geq \varphi_2(a) \land \varphi_1(b) \geq \varphi_2(b)) \land \Box(b \geq a \geq 0) \land \Box(v_a = a \land v_b = b \land v_i = i)\}$
                                                                                        (LUpdateS)
$\{(\forall \langle\varphi\rangle.\ \varphi(i) = \varphi(v_i) \land \varphi(a) = \varphi(v_a) \land \varphi(b) = \varphi(v_b))$
$\underbrace{\land\ (\forall \langle\varphi_1\rangle, \langle\varphi_2\rangle.\ \varphi_1(t) = 1 \land \varphi_2(t) = 2 \Rightarrow \varphi_1(v_a) \geq \varphi_2(v_a) \geq 0 \land \varphi_1(v_b) \geq \varphi_2(v_b) \geq 0 \land \varphi_1(n) - \varphi_1(v_i) \geq \varphi_2(n) - \varphi_2(v_i))\}}_{F}$   (Cons)
$\{\forall \langle\varphi\rangle.\ \varphi(i) = \varphi(v_i) \land \varphi(a) = \varphi(v_a) \land \varphi(b) = \varphi(v_b)\}$
  if $(*)$ {
    $\{\forall \langle\varphi\rangle.\ \varphi(i) = \varphi(v_i) \land \varphi(a) = \varphi(v_a) \land \varphi(b) = \varphi(v_b)\}$
    $\{\forall \langle\varphi\rangle.\ \varphi(i) < \varphi(n) \Rightarrow \varphi(v_i) < \varphi(n) \land \varphi(i) + 1 = \varphi(v_i) + 1 \land \varphi(b) = \varphi(v_b) \land \varphi(a) + \varphi(b) = \varphi(v_a) + \varphi(v_b)\}$   (Cons)
    assume $i < n$;
    $\{\forall \langle\varphi\rangle.\ \varphi(v_i) < \varphi(n) \land \varphi(i) + 1 = \varphi(v_i) + 1 \land \varphi(b) = \varphi(v_b) \land \varphi(a) + \varphi(b) = \varphi(v_a) + \varphi(v_b)\}$   (AssumeS)
    $tmp := b$;
    $b := a + b$;
    $a := tmp$;
    $i := i + 1$
    $\underbrace{\{\forall \langle\varphi\rangle.\ \varphi(v_i) < \varphi(n) \land \varphi(i) = \varphi(v_i) + 1 \land \varphi(a) = \varphi(v_b) \land \varphi(b) = \varphi(v_a) + \varphi(v_b)\}}_{Q_1}$   (AssignS)
  }
  else {
    $\{\forall \langle\varphi\rangle.\ \varphi(i) = \varphi(v_i) \land \varphi(a) = \varphi(v_a) \land \varphi(b) = \varphi(v_b)\}$
    $\{\forall \langle\varphi\rangle.\ \varphi(i) \geq \varphi(n) \Rightarrow \varphi(v_i) \geq \varphi(n) \land \varphi(i) = \varphi(v_i) \land \varphi(a) = \varphi(v_a) \land \varphi(b) = \varphi(v_b)\}$   (Cons)
    assume $\neg(i < n)$
    $\underbrace{\{\forall \langle\varphi\rangle.\ \varphi(v_i) \geq \varphi(n) \land \varphi(i) = \varphi(v_i) \land \varphi(a) = \varphi(v_a) \land \varphi(b) = \varphi(v_b)\}}_{Q_2}$   (AssumeS)
  }
  $\{Q_1 \otimes Q_2\}$   (Choice)
$\{(Q_1 \otimes Q_2) \land F\}$   (FrameSafe)
$\{\forall \langle\varphi_1\rangle, \langle\varphi_2\rangle.\ \varphi_1(t)=1 \land \varphi_2(t)=2 \Rightarrow (\varphi_1(n)-\varphi_1(i) \geq \varphi_2(n)-\varphi_2(i) \land \varphi_1(a) \geq \varphi_2(a) \land \varphi_1(b) \geq \varphi_2(b)) \land \Box(b \geq a \geq 0)\}$   (Cons)
```

Fig. 15. Second part of the proof. This proof outline shows $\vdash \{I\}$ **if** $(i < n)$ $\{C_{body}\}$ $\{I\}$, the first premise of the rule *While*-$\forall^*\exists$, where C_{body} refers to the body of the loop.

G MINIMUM EXAMPLE

This section contains the proof, using the rule *While-exists*, that the program C_m from Fig. 9 satisfies the triple

$$\{\neg emp \land \Box(k \geq 0)\}\ C_m\ \{\exists\langle\varphi\rangle.\,\forall\langle\alpha\rangle.\,\varphi(x) \leq \alpha(x) \land \varphi(y) \leq \alpha(y)\}$$

Fig. 16 contains the (trivial) first part of the proof, which justifies that the hyper-assertion $\exists\langle\varphi\rangle.\,P_\varphi$, where $P_\varphi \triangleq (\forall\langle\alpha\rangle.\,0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y) \land \varphi(k) \leq \alpha(k) \land \varphi(i) = \alpha(i))$, holds before the loop, as required by the precondition of the conclusion of the rule *While*-$\exists$.

Fig. 17 shows the proof of the first premise of the rule *While*-$\exists$, namely

$$\forall v.\,\exists\langle\varphi\rangle.\vdash \{P_\varphi \land \varphi(i) < \varphi(k) \land v = \varphi(k) - \varphi(i)\}\ \mathbf{if}\ (i < k)\ \{C_{body}\}\ \{\exists\langle\varphi\rangle.\,P_\varphi \land \varphi(k) - \varphi(i) < v\}$$

where C_{body} is the body of the loop.

Finally, Fig. 18 shows the proof of the second premise of the rule *While*-$\exists$. More precisely, it shows

$$\forall\varphi.\vdash \{Q_\varphi\}\ \mathbf{if}\ (i < k)\ \{C_{body}\}\ \{Q_\varphi\}$$

where $Q_\varphi \triangleq \forall\langle\alpha\rangle.\,0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y)$, from which we easily derive the second premise of the rule *While*-$\exists$, using the consequence rule (since P_φ clearly entails Q_φ), and the rule *While*-$\forall^*\exists^*$ rule.

```
{¬emp ∧ □(k ≥ 0)}
{∃⟨φ⟩. ∀⟨α⟩. 0 ≤ 0 ≤ 0 ∧ 0 ≤ 0 ≤ 0 ∧ φ(k) ≤ α(k) ∧ 0 = 0}                        (Cons)
x := 0;
{∃⟨φ⟩. ∀⟨α⟩. 0 ≤ φ(x) ≤ α(x) ∧ 0 ≤ 0 ≤ 0 ∧ φ(k) ≤ α(k) ∧ 0 = 0}                  (AssignS)
y := 0;
{∃⟨φ⟩. ∀⟨α⟩. 0 ≤ φ(x) ≤ α(x) ∧ 0 ≤ φ(y) ≤ α(y) ∧ φ(k) ≤ α(k) ∧ 0 = 0}            (AssignS)
i := 0;
{∃⟨φ⟩. ∀⟨α⟩. 0 ≤ φ(x) ≤ α(x) ∧ 0 ≤ φ(y) ≤ α(y) ∧ φ(k) ≤ α(k) ∧ φ(i) = α(i)}      (AssignS)
```

Fig. 16. First part of the proof: Establishing the first loop invariant $\exists\langle\varphi\rangle.\,P_\varphi$.

```
$\{\exists\langle\varphi\rangle.\,(\forall\langle\alpha\rangle.\,0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y) \land \varphi(k) \leq \alpha(k) \land \varphi(i) = \alpha(i)) \land \varphi(i) < \varphi(k) \land v = \varphi(k) - \varphi(i)\}$
if $(i < k)$ {
  $\{\exists\langle\varphi\rangle.\,(\forall\langle\alpha\rangle.\,0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y) \land \varphi(k) \leq \alpha(k) \land \varphi(i) = \alpha(i)) \land \varphi(i) < \varphi(k) \land v = \varphi(k) - \varphi(i) \land \square(i < k)\}$
  $\{\exists\langle\varphi\rangle.\,\exists u.\,u \geq 2 \land (\forall\langle\alpha\rangle.\,\forall v.\,v \geq 2 \Rightarrow 0 \leq 2 * \varphi(x) + u \leq 2 * \alpha(x) + v \land 0 \leq \varphi(y) + \varphi(x) * u \leq \alpha(y) + \alpha(x) * v$
  $\land \varphi(k) \leq \alpha(k) \land \varphi(i) + 1 = \alpha(i) + 1) \land \varphi(k) - \varphi(i) \prec v\}$          (Cons (1))
  $r := nonDet()$;
  assume $r \geq 2$;
  $\{\exists\langle\varphi\rangle.\,(\forall\langle\alpha\rangle.\,0 \leq 2 * \varphi(x) + \varphi(r) \leq 2 * \alpha(x) + \alpha(r) \land 0 \leq \varphi(y) + \varphi(x) * \varphi(r) \leq \alpha(y) + \alpha(x) * \alpha(r) \land \varphi(k) \leq \alpha(k) \land \varphi(i) + 1 = \alpha(i) + 1)$
  $\land \varphi(k) - \varphi(i) \prec v\}$          (HavocS, AssumeS)
  $t := x$;
  $x := 2 * x + r$;
  $y := y + t * r$;
  $i := i + 1$
  $\{\exists\langle\varphi\rangle.\,(\forall\langle\alpha\rangle.\,0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y) \land \varphi(k) \leq \alpha(k) \land \varphi(i) = \alpha(i)) \land \varphi(k) - \varphi(i) \prec v\}$          (AssignS)
}
else {
  $\{\exists\langle\varphi\rangle.\,(\forall\langle\alpha\rangle.\,0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y) \land \varphi(k) \leq \alpha(k) \land \varphi(i) = \alpha(i)) \land \varphi(i) < \varphi(k) \land v = \varphi(k) - \varphi(i) \land \square(i \geq k)\}$
  $\{\exists\langle\varphi\rangle.\,(\forall\langle\alpha\rangle.\,0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y) \land \varphi(k) \leq \alpha(k) \land \varphi(i) = \alpha(i)) \land \varphi(k) - \varphi(i) \prec v\}$          (Cons (2))
  skip
  $\{\exists\langle\varphi\rangle.\,(\forall\langle\alpha\rangle.\,0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y) \land \varphi(k) \leq \alpha(k) \land \varphi(i) = \alpha(i)) \land \varphi(k) - \varphi(i) \prec v\}$          (Skip)
}
$\{\exists\langle\varphi\rangle.\,(\forall\langle\alpha\rangle.\,0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y) \land \varphi(k) \leq \alpha(k) \land \varphi(i) = \alpha(i)) \land \varphi(k) - \varphi(i) \prec v\}$          (IfSync)
```

Fig. 17. Second part of the proof. Establishing the first premise of the rule *While-*$\exists$, $\forall v.\,\exists\langle\varphi\rangle.\vdash \{P_\varphi \land \varphi(i) < \varphi(k) \land v = \varphi(k) - \varphi(i)\}$ **if** $(i < k)$ $\{C_{body}\}$ $\{\exists\langle\varphi\rangle.\,P_\varphi \land \varphi(k) - \varphi(i) \prec v\}$.
For Cons (1), we simply choose $u = 2$. For Cons (2), we notice that $\varphi(i) < \varphi(k)$ and $\square(i \geq k)$ are inconsistent (this branch is not taken at this stage), and thus the entailment trivially holds.

```
{$\forall\langle\alpha\rangle.\, 0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y)$}
if (*) {
    {$\forall\langle\alpha\rangle.\, 0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y)$}
    {$\forall\langle\alpha\rangle.\, \alpha(i) < \alpha(k) \Rightarrow \forall v.\, v \geq 2 \Rightarrow 0 \leq \varphi(x) \leq 2 * \alpha(x) + v \land 0 \leq \varphi(y) \leq \alpha(y) + \alpha(x) * v$}        (Cons)
    assume $i < k$;
    {$\forall\langle\alpha\rangle.\, \forall v.\, v \geq 2 \Rightarrow 0 \leq \varphi(x) \leq 2 * \alpha(x) + v \land 0 \leq \varphi(y) \leq \alpha(y) + \alpha(x) * v$}        (AssumeS)
    $r := nonDet()$;
    {$\forall\langle\alpha\rangle.\, \alpha(r) \geq 2 \Rightarrow 0 \leq \varphi(x) \leq 2 * \alpha(x) + \alpha(r) \land 0 \leq \varphi(y) \leq \alpha(y) + \alpha(x) * \alpha(r)$}        (HavocS)
    assume $r \geq 2$;
    {$\forall\langle\alpha\rangle.\, 0 \leq \varphi(x) \leq 2 * \alpha(x) + \alpha(r) \land 0 \leq \varphi(y) \leq \alpha(y) + \alpha(x) * \alpha(r)$}        (AssumeS)
    $t := x$;
    $x := 2 * x + r$;
    $y := y + t * r$;
    $i := i + 1$
    {$\forall\langle\alpha\rangle.\, 0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y)$}        (AssignS)
}
else {
    {$\forall\langle\alpha\rangle.\, 0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y)$}
    {$\forall\langle\alpha\rangle.\, \alpha(i) \geq \alpha(k) \Rightarrow 0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y)$}        (Cons)
    assume $i \geq k$;
    skip
    {$\forall\langle\alpha\rangle.\, 0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y)$}        (AssumeS, Skip)
}
{$(\forall\langle\alpha\rangle.\, 0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y)) \otimes (\forall\langle\alpha\rangle.\, 0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y))$}        (Choice)
{$\forall\langle\alpha\rangle.\, 0 \leq \varphi(x) \leq \alpha(x) \land 0 \leq \varphi(y) \leq \alpha(y)$}        (Cons)
```

Fig. 18. Third part of the proof. This proof outline shows $\forall\varphi. \vdash \{Q_\varphi\}$ **if** $(i < k)$ $\{C_{body}\}$ $\{Q_\varphi\}$.

H SYNCHRONOUS REASONING OVER DIFFERENT BRANCHES

The central thesis of this paper is that reasoning about how sets of states are affected by *one* program command is powerful enough to reason about any program hyperproperty, which is supported by our completeness result (Thm. 2).

However, reasoning about (for example) two executions of the same program sometimes boils down to reasoning about two executions of two *different* but similar programs, because of branching. One *a priori* appeal of relational program logics over Hyper Hoare Logic is thus the ability to reason about two different branches *synchronously*.

As an example, imagine that we want to reason about $C' \triangleq (x := x * 2; C) + C$. Except for the assignment that happens only in one branch, the two branches are extremely similar. In a relational program logic, we can exploit this similarity by first reasoning about the assignment on its own, and then reasoning about the two remaining branches C and C synchronously, since they are the same.

On the other hand, with the rule *If* from Fig. 3, we would have to reason about the two branches $x := x * 2; C$ and C separately, even though they are closely related.

This is not a fundamental limitation of Hyper Hoare Logic. We can indeed enable this kind of synchronous reasoning in Hyper Hoare Logic, by adding specialized rules, as illustrated by Prop. 14 below.

Let us first define the following notation:

Notation 1.

$$(A \otimes_{x=1,2} B)(S) \triangleq (A(\{(l,\sigma) \mid (l,\sigma) \in S \land l(x) = 1\}) \land B(\{(l,\sigma) \mid (l,\sigma) \in S \land l(x) = 2\}))$$

The assertion $A \otimes_{x=1,2}$ holds in a set S iff the subset of all states in S such that $l(x) = 1$ satisfies A, and the subset of all states in S such that $l(x) = 2$ must satisfy B.

Proposition 14. ***Synchronized* if *rule.*** *If*

(1) $\models \{P\}\ C_1\ \{P_1\}$
(2) $\models \{P\}\ C_2\ \{P_2\}$
(3) $\models \{P_1 \otimes_{x=1,2} P_2\}\ C\ \{R_1 \otimes_{x=1,2} R_2\}$
(4) $\models \{R_1\}\ C_1'\ \{Q_1\}$
(5) $\models \{R_2\}\ C_2'\ \{Q_2\}$
(6) $x \notin rd(P_1) \cup rd(P_2) \cup rd(R_1) \cup rd(R_2)$

Then $\models \{P\}\ (C_1; C; C_1') + (C_2; C; C_2')\ \{Q_1 \otimes Q_2\}$.

This proposition shows how to reason synchronously about the program command $(C_1; C; C_1') + (C_2; C; C_2')$. Points 1) and 2) show that we can reason independently about the different parts of the branches C_1 and C_2. Point 3) then shows how we can reason synchronously about the execution of C in both branches. Finally, points 4) and 5) show how to go back to reasoning independently about each branch.