University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master degree in Computer Science

Abstract Hoare logic

Supervisor
Prof. Francesco Ranzato

Co. Supervisor
Prof. Paolo Baldan

Candidate
Alessio Ferrarini

Academic Year 2023–2024

ABSTRACT

In theoretical computer science, program logics are essential for verifying the correctness of software. Hoare logic, provides a systematic way of reasoning about program correctness using preconditions and postconditions. This thesis explores the development and application of an abstract Hoare logic framework that generalizes traditional Hoare logic by using arbitrary elements of complete lattices as the assertion language, extrapolating what makes Hoare logic sound sound and complete. We also demonstrate the practical application of this framework through by obtaining a program logic for hyperproperties, highlighting its versatility and efficacy. From the concept of Abstract Hoare logic Reverse abstract Hoare logic is developed and is used to develop a system to perform backward correctness reasoning.

ACKNOWLEDGMENTS

To . . .

> *"Progress is possible only if we train ourselves to think about programs without thinking of them as pieces of executable code."*
>
> Edsger W. Dijkstra

CONTENTS

INTRODUCTION

The verification of program correctness is a critical task in computer science. Ensuring that software behaves as expected under all possible conditions is fundamental in a society that increasingly relies on computer programs. Programmers often reason about the behavior of their programs at an intuitive level. While this is definitely better than not reasoning at all, intuition alone becomes insufficient as the size of programs grows.

Writing tests for programs is definitely a useful task, but at best, it can show the presence of bugs, not prove their absence. We cannot feasibly write a test for every possible input of the program. To offer a guarantee of the absence of undesired behavior, we need sound logical models rooted in logic. The field of formal methods in computer science aims to develop the logical tools necessary to prove properties of software systems.

Hoare logic, first popularized by Hoare in the late 60s, provides a set of logical rules to reason about the correctness of computer programs. Hoare logic formalizes, with axioms and inference rules, the relationship between the initial and final states after running a program.

Hoare logic, beyond being one of the first, is arguably also one of the most influential ideas in the field of software verification. It created the whole field of program logics—systems of logical rules aimed at proving properties of programs. Over the years, modifications of Hoare logic have been developed, sometimes to support new language features such as dynamic memory allocation and pointers, or to prove different properties such as equivalence between programs or properties of multiple executions. Every time Hoare logic is modified, it is necessary to prove again that the proof system indeed proves properties about the program (soundness) and that the proof system is powerful enough to prove the properties of interest (completeness).

Most modifications of Hoare logic usually do not alter the fundamental proof principles of the system. Instead, they often extend the assertion language to express new properties and add new commands to support new features in different programming languages.

We introduce Abstract Hoare Logic, which aims to be a framework general enough to serve as an extensible platform for constructing new Hoare-like logics without the burden of proving soundness and completeness anew. We demonstrate, by example, how some properties that are not expressible in standard Hoare logic can be simply instantiated within Abstract Hoare Logic, while keeping the proof system as simple as possible.

The theory of Abstract Hoare Logic is deeply connected to the theory of abstract interpretation. The semantics of the language is defined as an inductive abstract interpreter, and the validity of the Abstract Hoare triples depends on it. By not using the strongest postcondition directly, we are able to reason about properties that are not expressible in the powerset of the states, such as hyperproperties.

This thesis is subdivided as follows:

- In Chapter 1, we introduce the basic mathematical background of order theory and abstract interpretation.

- In Chapter 2, we introduce standard Hoare logic and the general framework of Abstract

Hoare Logic: the extensible $\mathbb{L}$ language, its syntax and semantics, the generalization of the strongest postcondition, and finally, the actual Abstract Hoare Logic and its proof system, proving the general results of soundness and relative completeness.

- In Chapter 3, we show some interesting instantiations of Abstract Hoare Logic: we demonstrate that it is possible to obtain program logic where the implication is decidable, thus making the goal of checking a derivation computable; we show how to obtain a proof system for hyperproperties (and introduce the concept of the strongest hyper postcondition); and we show that it is possible to obtain a proof system for partial incorrectness.
- In Chapter 4, we show how to enrich the barebones proof system of Abstract Hoare Logic by adding more restrictions on the assertion language or the semantics.
- In Chapter 5, we show how to reuse the idea of Abstract Hoare Logic to generalize proof systems for backward reasoning.
- In Chapter 6, we provide a brief recap of the most important points of the thesis. We discuss possible extensions to the framework of Abstract Hoare Logic and, finally, we examine the relationship of Abstract Hoare Logic with other similar work.

CHAPTER 1

BACKGROUND

1.1 Order theory

When defining the semantics of programming languages, the theory of *partially ordered sets* and *lattices* is fundamental. These concepts are at the core of denotational semantics [Sco70] and *Abstract Interpretation* [CC77], where the semantics of programming languages and abstract interpreters are defined as monotone functions over some complete lattice.

1.1.1 Partial Orders

Definition 1.1 (Partial order)**.** A partial order on a set X is a relation $\leq \subseteq X \times X$ such that the following properties hold:

- Reflexivity: $\forall x \in X,\ (x,x) \in \leq$
- Anti-symmetry: $\forall x, y \in X,\ (x,y) \in \leq \text{ and } (y,x) \in \leq \implies x = y$
- Transitivity: $\forall x, y, z \in X,\ (x,y) \in \leq \text{ and } (y,z) \in \leq \implies (x,z) \in \leq$

Given a partial order $\leq$, we will use $\geq$ to denote the converse relation $\{(y,x) \mid (x,y) \in \leq\}$ and $<$ to denote $\{(x,y) \mid (x,y) \in \leq \text{ and } x \neq y\}$.

From now on we will use the notation xRy to indicate $(x,y) \in R$.

Definition 1.2 (Partially ordered set)**.** A partially ordered set (or poset) is a pair $(X, \leq)$ in which $\leq$ is a partial order on X.

Definition 1.3 (Monotone function)**.** Given two ordered sets $(X, \leq)$ and $(Y, \sqsubseteq)$, a function $f : X \to Y$ is said to be monotone if $x \leq y \implies f(x) \sqsubseteq f(y)$.

Definition 1.4 (Galois connection)**.** Let $(C, \sqsubseteq)$ and $(A, \leq)$ be two partially ordered sets, a Galois connection written $\langle C, \sqsubseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \leq \rangle$, are a pair of functions: $\gamma : A \to D$ and $\alpha : D \to A$ such that:

- γ is monotone
- α is monotone
- $\forall c \in C\ c \sqsubseteq \gamma(\alpha(c))$
- $\forall a \in A\ a \leq \alpha(\gamma(a))$

Definition 1.5 (Galois Insertion). Let $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle A, \leq \rangle$, be a Galois connection, a Galois insertion written $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle A, \leq \rangle$ are a pair of functions: $\gamma : A \to D$ and $\alpha : D \to A$ such that:

- (γ, α) are a Galois connection
- $\alpha \circ \gamma = id$

Definition 1.6 (Fixpoint). Given a function $f : X \to X$, a fixpoint of f is an element $x \in X$ such that $x = f(x)$.

We denote the set of all fixpoints of a function as $\text{fix}(f) = \{x \mid x \in X \text{ and } x = f(x)\}$.

Definition 1.7 (Least and Greatest fixpoints). Given a function $f : X \to X$,

- We denote the *least fixpoint* as $\text{lfp}(f) = \min\ \text{fix}(f)$.
- We denote the *greatest fixpoint* as $\text{gfp}(f) = \max\ \text{fix}(f)$.

1.1.2 Lattices

Definition 1.8 (Meet-semilattice). A poset $(X, \leq)$ is a meet-semilattice if $\forall x, y \in X, \exists z \in X$ such that $z = \inf\{x, y\}$, called the *meet*.

Usually, the meet of two elements $x, y \in X$ is written as $x \wedge y$.

Definition 1.9 (Join-semilattice). A poset $(X, \leq)$ is a join-semilattice if $\forall x, y \in X, \exists z \in X$ such that $z = \sup\{x, y\}$, called the *join* or *least upper bound*.

Usually, the join of two elements $x, y \in X$ is written as $x \vee y$.

Observation 1.1. Both join and meet operations are idempotent, associative, and commutative.

Definition 1.10 (Lattice). A poset $(X, \leq)$ is a lattice if it is both a join-semilattice and a meet-semilattice.

Definition 1.11 (Complete lattice). A lattice $(X, \leq)$ is said to be complete if $\forall\, Y \subseteq X$:

- $\exists\, z \in X$ such that $z = \sup\ Y$
- $\exists\, z \in X$ such that $z = \inf\ Y$

We denote the *least element* or *bottom* as $\perp = \inf\ X$ and the *greatest element* or *top* as $\top = \sup\ X$.

Observation 1.2. A complete lattice cant be empty.

Definition 1.12 (Point-wise lift). Given a complete lattice L and a set A we call *point-wise* lift of L the set of all functions $A \to L$ ordered point-wise $f \leq g \iff \forall a \in A\ f(a) \leq f(g)$.

Theorem 1.1 (Point-wise fixpoint). *The leaft-fixpoint and greatest fixpoint on some point-wise lifted lattice on a monotone function defined point-wise is the point-wise lift of the function.*

$$lfp(\lambda p' a. f(p'(a))) = \lambda a. lfp(\lambda p'. f(a))$$
$$gfp(\lambda p' a. f(p'(a))) = \lambda a. gfp(\lambda p'. f(a))$$

Theorem 1.2 (Knaster-Tarski theorem). *Let $(L, \leq)$ be a complete lattice and let $f : L \to L$ be a monotone function. Then $(fix(f), \leq)$ is also a complete lattice.*

Two direct consequences that both the greatest and the least fixpoint of f exists and are respectively $\top$ and $\perp$ of $\text{fix}(f)$.

Theorem 1.3 (Post-fixpoint inequality). *Let f be a monotone function on a complete lattice then*

$$f(x) \leq x \implies lfp(f) \leq x$$

Proof. By theorem 1.2 $\text{lfp}(f) = \bigwedge\{y \mid y \geq f(y)\}$ thus $\text{lfp}(f) \leq x$ since $x \in \{y \mid y \geq f(y)\}$. □

Theorem 1.4 (lfp monotonicity)**.** *Let L be a complete lattice then if $P \leq Q$ and f is monotone*

$$lfp(\lambda X.P \vee f(X)) \leq lfp(\lambda X.Q \vee f(X))$$

Proof.

$$\begin{aligned} P \vee f(\text{lfp}(\lambda X.Q \vee f(X))) &\leq Q \vee f(\text{lfp}(\lambda X.Q \vee f(X))) && \text{Since } P \leq Q \\ &= \text{lfp}(\lambda X.Q \vee f(X)) && \text{By definition of fixpoint} \end{aligned}$$

Thus by theorem 1.3 pick $f = \lambda X.P \vee f(X)$ and $x = lfp(\lambda X.Q \vee f(X))$ it follows that $lfp(\lambda X.P \vee f(X)) \leq lfp(\lambda X.Q \vee f(X))$. □

1.2 Abstract Interpretation

Abstract interpretation [CC77] is the leading technique used for static program analysis. The specification of a program can be expressed as a pair of initial and final sets of states, $Init, Final \in \wp(\mathbb{S})$, and the task of verifying a program C involves checking if $[\![C]\!](Init) \subseteq Final$.

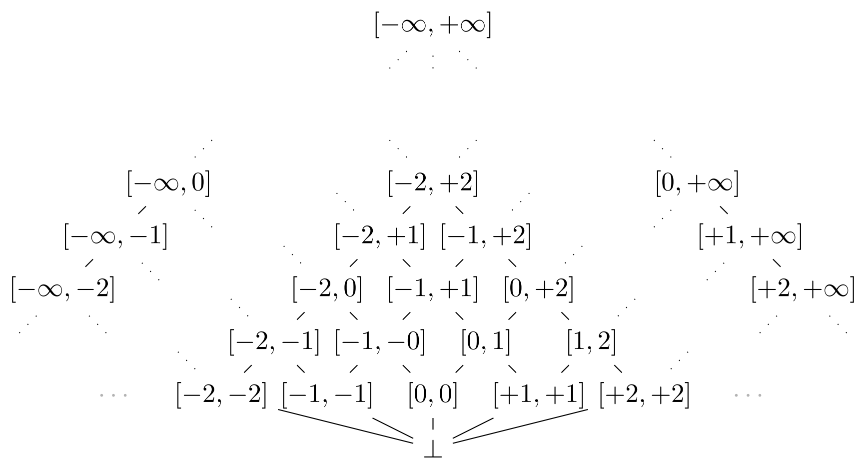
Clearly, this task cannot be performed programmatically in general. The solution proposed by the framework of abstract interpretation is to construct an approximation, usually denoted by $[\![\cdot]\!]^{\#}$, that is computable.

1.2.1 Abstract Domains

One of the techniques used by abstract interpretation to make the problem of verification tractable involves representing collections of states with a finite amount of memory.

Definition 1.13 (Abstract Domain)**.** A poset $(A, \leq)$ is an abstract domain if there exists a Galois insertion $\langle \wp(\mathbb{S}), \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \leq\rangle$.

Example 1.1 (Interval Domain)**.** Let $Int = \{[a, b] \mid a, b \in \mathbb{Z} \cup \{+\infty, -\infty\}, a \leq b\} \cup \{\bot\}$ be ordered by inclusion, each element $[a, b]$ represent the set $\{x \mid a \leq x \leq b\}$ and $\bot$ is used as a representation of $\emptyset$. The structure of the lattice can be summarized by the following Hasse diagram:



Then, there is a Galois insertion from Int to $\wp(\mathbb{Z})$ defined as:

$$\gamma(A) = \begin{cases} \{x \mid a \leq x \leq b\} & \text{if } A = [a, b] \\ \emptyset & \text{otherwise} \end{cases}$$

$$\alpha(C) = \begin{cases} [\min\ C, \max\ C] & \text{if } C \neq \emptyset \\ \bot & \text{otherwise} \end{cases}$$

Example 1.2 (Complete sign domain)**.** Let $Sign = \{\bot, < 0, > 0, = 0, \leq 0, \neq 0, \geq 0, \mathbb{Z}\}$ be ordered by inclusion, each element op 0 represent the set $\{x \mid x \ op \ 0\}$, $\mathbb{Z}$ represents the set $\mathbb{Z}$ and $\bot$ represents $\emptyset$. The structure of the lattice can be summarized by the following Hasse diagram:

$$\begin{tikzcd}
 & \mathbb{Z} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\leq 0 \arrow[d, dash] \arrow[dr, dash] & \neq 0 \arrow[dl, dash] \arrow[dr, dash] & \geq 0 \arrow[dl, dash] \arrow[d, dash] \\
< 0 \arrow[dr, dash] & = 0 \arrow[d, dash] & > 0 \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$

Then, there is a Galois insertion from $Sign$ to $\wp(\mathbb{Z})$ defined as:

$$\gamma(A) = \begin{cases} \{x \mid x \ op \ 0\} & \text{if } A = op \ 0 \\ \mathbb{Z} & \text{if } A = \mathbb{Z} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\alpha(C) = \begin{cases} \bot & \text{if } C = \emptyset \\ op \ 0 & \text{if } C \subseteq \{x \mid x \ op \ 0\} \text{ and } op \in \{<, >, =, \leq, \geq, \neq\} \\ \mathbb{Z} & \text{otherwise} \end{cases}$$

The fundamental goal of abstract interpretation is to provide an approximation of the non-computable aspects of program semantics. The core concept is captured by the definition of soundness:

Definition 1.14 (Soundness)**.** Given an abstract domain A, an abstract function $f^{\#} : A \to A$ is a sound approximation of a concrete function $f : \wp(\mathbb{S}) \to \wp(\mathbb{S})$ if

$$\alpha(f(P)) \leq f^{\#}(\alpha(P))$$

Hence, the goal of abstract interpretation is to construct a sound over-approximation of the program semantics that is computable (efficiently).

CHAPTER 2

THE ABSTRACT HOARE LOGIC FRAMEWORK

In this chapter, we will develop the minimal theory of *Abstract Hoare Logic.* We will formalize the extensible $\mathbb{L}$ language, a minimal imperative programming language that is parametric on a set of base commands to permit the definition of arbitrary program features, such as pointers, objects, etc. We will define the semantics of the language, provide a refresher on Hoare triples, and introduce the concept of abstract inductive semantics; a modular approach to express the strongest postcondition of a program, where the assertion language is a complete lattice. Additionally, we will present a sound and complete proof system to reason about these properties.

2.1 The $\mathbb{L}$ programming language

2.1.1 Syntax

The $\mathbb{L}$ language is inspired by Dijkstra's guarded command languages [Dij74] but with the goal of beeing as general as possible by beeing parametric on a set of *base commands.* The $\mathbb{L}$ language is general enough to describe any imperative non deterministic programming language.

Definition 2.1 ($\mathbb{L}$ language syntax)**.** Given a set *Base* of base commands, the set on valid $\mathbb{L}$ programs is defined by the following inductive definition:

$$
\begin{array}{rcll}
\mathbb{L} & ::= & \mathbb{1} & \text{Skip} \\
& | & b & \text{Base command} \\
& | & C_1 \fatsemi C_2 & \text{Program composition} \\
& | & C_1 + C_2 & \text{Non deterministic choice} \\
& | & C^{\text{fix}} & \text{Iteration}
\end{array}
$$

Where $C, C_1, C_2 \in \mathbb{L}$ and $b \in Base$.

Example 2.1. Usually the set of base commands contains a command $e?$ to discard execution that don't satisfy the predicate e and $x := y$ to assing the value y to the variable x.

2.1.2 Semantics

Fixed a set $\mathbb{S}$ of states (usually a collection of associations between variables names and values) and a family of partial functions $[\![\cdot]\!]_{base} : \mathbb{S} \hookrightarrow \mathbb{S}$ we can define the denotational semantics of programs in $\mathbb{L}$, the *collecting semantics* is a function $[\![\cdot]\!] : \mathbb{L} \to \wp(\mathbb{S}) \to \wp(\mathbb{S})$ that associates a program C

and set of initial states to the set of states reached after executing the program C from the initial states.

Definition 2.2 ($\mathbb{L}$ denotational semantics)**.** Given a set $\mathbb{S}$ of states and a family of partial functions $[\![b]\!]_{base} : \mathbb{S} \hookrightarrow \mathbb{S}\ \forall b \in Base$ the denotational semantics is defined as follows:

$$\begin{aligned}
[\![\cdot]\!] \ &:\ \mathbb{L} \to \wp(\mathbb{S}) \to \wp(\mathbb{S}) \\
[\![\mathbb{1}]\!] &\stackrel{\text{def}}{=} id \\
[\![b]\!] &\stackrel{\text{def}}{=} \lambda P.\{[\![b]\!]_{base}(p) \downarrow \mid p \in P\} \\
[\![C_1 \,⨟\, C_2]\!] &\stackrel{\text{def}}{=} [\![C_2]\!] \circ [\![C_1]\!] \\
[\![C_1 + C_2]\!] &\stackrel{\text{def}}{=} \lambda P.[\![C_1]\!]P \cup [\![C_2]\!]P \\
[\![C^{\text{fix}}]\!] &\stackrel{\text{def}}{=} \lambda P.\text{lfp}(\lambda P'.P \cup [\![C]\!]P')
\end{aligned}$$

Example 2.2. We can define the semantics of the base commands introduced in 2.1 as:

$$[\![e?]\!]_{base}(\sigma) \stackrel{\text{def}}{=} \begin{cases} \sigma & \sigma \models e \\ \uparrow & otherwise \end{cases}$$

$$[\![x := y]\!]_{base}(\sigma) \stackrel{\text{def}}{=} \sigma[x/eval(y,\sigma)]$$

Where *eval* is some evaluate function for the expressions on the left-hand side of assignments.

Theorem 2.1 (Complete lattice)**.** *$(\wp(\mathbb{S}), \subseteq)$ is a complete lattice.*

Proof. To prove that $(\wp(\mathbb{S}), \subseteq)$ is a complete lattice we exhibit: $\forall P \subseteq \wp(states)$

- $\inf P = \bigcap P$, it's clearly a lowerbound, and it's the greatest since any other set $Z \supsetneq \bigcap P$ contains some not in any of the elements in P.
- $\sup P = \bigcup P$, it's clearly an upper bound, and it's the smallest one since any other set $Z \subsetneq \bigcup P$ is missing some element that is in one of the elements of P.

□

Theorem 2.2 (Monotonicity)**.** *$\forall\, C \in \mathbb{L}\ [\![C]\!]$ is monotone.*

Proof. We want to prove that $\forall P, Q \in \wp(\mathbb{S})$ and $C \in \mathbb{L}$

$$P \subseteq Q \implies [\![C]\!](P) \subseteq [\![C]\!](Q)$$

By structural induction on C:

- $\mathbb{1}$:

$$\begin{aligned}
[\![\mathbb{1}]\!](P) &= P && \text{By definition of } [\![\mathbb{1}]\!] \\
&\subseteq Q \\
&= [\![\mathbb{1}]\!](Q) && \text{By definition of } [\![\mathbb{1}]\!]
\end{aligned}$$

- b:

$$\begin{aligned}
[\![b]\!](P) &= \{[\![b]\!]_{base}(x) \downarrow \mid x \in P\} && \text{By definition of } [\![b]\!] \\
&\subseteq \{[\![b]\!]_{base}(x) \downarrow \mid x \in Q\} && \text{Since } P \subseteq Q \\
&= [\![b]\!](Q) && \text{By definition of } [\![b]\!]
\end{aligned}$$

- $C_1 \mathbin{;} C_2$:

  By inductive hypothesis $[\![C_1]\!]$ is monotone hence $[\![C_1]\!](P) \subseteq [\![C_2]\!](Q)$

$$\begin{aligned} [\![C_1 \mathbin{;} C_2]\!](P) &= [\![C_2]\!]([\![C_1]\!](P)) && \text{By definition of } [\![C_1 \mathbin{;} C_2]\!] \\ &\subseteq [\![C_2]\!]([\![C_1]\!](Q)) && \text{By inductive hypothesis on } [\![C_2]\!] \end{aligned}$$

- $C_1 + C_2$:

$$\begin{aligned} [\![C_1 + C_2]\!](P) &= [\![C_1]\!](P) \cup [\![C_2]\!](P) && \text{By definition of } [\![C_1 + C_2]\!] \\ &\subseteq [\![C_1]\!](Q) \cup [\![C_2]\!](P) && \text{By inductive hypothesis on } [\![C_1]\!] \\ &\subseteq [\![C_1]\!](Q) \cup [\![C_2]\!](Q) && \text{By inductive hypothesis on } [\![C_2]\!] \\ &= [\![C_1 + C_2]\!](Q) && \text{By definition of } [\![C_1 + C_2]\!] \end{aligned}$$

- C^{fix}:

$$\begin{aligned} [\![C^{\text{fix}}]\!](P) &= lfp(\lambda P'.P \cup [\![C]\!](P')) && \text{By definition of } [\![C^{\text{fix}}]\!] \\ &\subseteq lfp(\lambda P'.Q \cup [\![C]\!](P')) && \text{By theorem 1.4} \\ &= [\![C^{\text{fix}}]\!](Q) && \text{By definition of } [\![C^{\text{fix}}]\!] \end{aligned}$$

□

Lemma 2.1 ($[\![\cdot]\!]$ well-defined)**.** *$\forall\, C \in \mathbb{L}$ $[\![C]\!]$ is well-defined.*

Proof. From theorems 2.1, 2.2 and 1.2 all the least fixpoints in the definition of $[\![C^{\text{fix}}]\!]$ exists; for all the other commands the semantics is trivially well-defined. □

Observation 2.1. As observed in [FL79] when the set of base commands contains a command to discard executions we can define the usual deterministic control flow commands as syntactic sugar.

$$if\ b\ then\ C_1\ else\ C_2 \stackrel{\text{def}}{=} (b? \mathbin{;} C_1) + (\neg b? \mathbin{;} C_2)$$

$$while\ b\ do\ C \stackrel{\text{def}}{=} (b? \mathbin{;} C)^{\text{fix}} \mathbin{;} \neg b?$$

Observation 2.2. Some other languages usually provide an iteration command usually denoted $C^\star$ whose semantics is $[\![C^\star]\!](P) \stackrel{\text{def}}{=} \bigcup_{n\in\mathbb{N}} [\![C]\!]^n(P)$, this is equivalent to C^{fix}, the reasoning on why a fixpoint formulation was chosen will become clear in 2.4.

2.2 Abstract inductive semantics

From the theory of abstract interpretation we know that the definition of the denotational semantics can be modified to work on any complete lattice as long that we can provide sensible function for the base commands. The rationale behind is the same as in the denotational semantics but instead representing collections of states with $\wp(\mathbb{S})$ now they are represented by an arbitrary complete lattice.

Definition 2.3 (Abstract inductive semantics)**.** Given a complete lattice A and a family of monotone functions $[\![b]\!]^A_{base} : A \to A$ $\forall b \in Base$ the abstract inductive semantics is defined as follows:

$$
\begin{aligned}
[\![\cdot]\!]^A_{ais} &: \mathbb{L} \to A \to A \\
[\![\mathbb{1}]\!]^A_{ais} &\stackrel{\text{def}}{=} id \\
[\![b]\!]^A_{ais} &\stackrel{\text{def}}{=} [\![b]\!]^A_{base} \\
[\![C_1 \mathbin{⨟} C_2]\!]^A_{ais} &\stackrel{\text{def}}{=} [\![C_2]\!]^A_{ais} \circ [\![C_1]\!]^A_{ais} \\
[\![C_1 + C_2]\!]^A_{ais} &\stackrel{\text{def}}{=} \lambda P.[\![C_1]\!]^A_{ais} P \vee_A [\![C_2]\!]^A_{ais} P \\
[\![C^{\text{fix}}]\!]^A_{ais} &\stackrel{\text{def}}{=} \lambda P.\text{lfp}(\lambda P'.P \vee_A [\![C]\!]^A_{ais} P')
\end{aligned}
$$

When developing abstract interpreters to perform abstract interpretation, iterative commands are usually not expressed directly as fixpoints but by some over-approximation, as is the case for the C^{fix} command. This is necessary since the scope of the abstract interpreter is to be run, and in general, if the lattice on which the interpretation is run has infinite ascending chains, the computation can diverge. In our case, the termination requirement is necessary since we aren't interested in running the abstract inductive semantics but using it as the semantics on which the definition of abstract Hoare logic is dependent.

As we did for the concrete collecting semantics, we need to prove that the semantics is well-defined. In general, if we drop the requirement for A to be a complete lattice or for $[\![b]\!]_{base}$ to be monotone, the least fixpoint could be undefined.

Theorem 2.3 (Monotonicity)**.** *$\forall\, C \in \mathbb{L}$ $[\![C]\!]^A_{ais}$ is monotone.*

Proof. We want to prove that $\forall P, Q \in A$ and $C \in \mathbb{L}$

$$P \leq_A Q \implies [\![C]\!]^A_{ais}(P) \leq_A [\![C]\!]^A_{ais}(Q)$$

By structural induction on C:

- $\mathbb{1}$:

$$
\begin{aligned}
[\![\mathbb{1}]\!](P) &= P && \text{By definition of } [\![\mathbb{1}]\!]^A_{ais} \\
&\leq Q \\
&= [\![\mathbb{1}]\!](Q) && \text{By definition of } [\![\mathbb{1}]\!]^A_{ais}
\end{aligned}
$$

- b:

$$
\begin{aligned}
[\![b]\!](P) &= [\![b]\!]^A_{base}(P) && \text{By definition of } [\![b]\!]^A_{ais} \\
&\leq [\![b]\!]^A_{base}(Q) && \text{By definition} \\
&= [\![b]\!](Q) && \text{By definition of } [\![b]\!]^A_{ais}
\end{aligned}
$$

- $C_1 \mathbin{⨟} C_2$:
  By inductive hypothesis $[\![C_1]\!]^A_{ais}$ is monotone hence $[\![C_1]\!]^A_{ais}(P) \leq_A [\![C_1]\!]^A_{ais}(Q)$

$$
\begin{aligned}
[\![C_1 \mathbin{⨟} C_2]\!]^A_{ais}(P) &= [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P)) && \text{By definition of } [\![C_1 \mathbin{⨟} C_2]\!]^A_{ais} \\
&\leq_A [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(Q)) && \text{By inductive hypothesis on } [\![C_2]\!]^A_{ais}
\end{aligned}
$$

- $C_1 + C_2$:

$$\begin{aligned}
[\![C_1 + C_2]\!]^A_{ais}(P) &= [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P) && \text{By definition of } [\![C_1 + C_2]\!]^A_{ais} \\
&\leq_A [\![C_1]\!]^A_{ais}(Q) \vee_A [\![C_2]\!]^A_{ais}(P) && \text{By inductive hypothesis on } [\![C_1]\!]^A_{ais} \\
&\leq_A [\![C_1]\!]^A_{ais}(Q) \vee_A [\![C_2]\!]^A_{ais}(Q) && \text{By inductive hypothesis on } [\![C_2]\!]^A_{ais} \\
&= [\![C_1 + C_2]\!]^A_{ais}(Q) && \text{By definition of } [\![C_1 + C_2]\!]^A_{ais}
\end{aligned}$$

- C^{fix}:

$$\begin{aligned}
[\![C^{\text{fix}}]\!]^A_{ais}(P) &= lfp(\lambda P'.P \vee_A [\![C]\!]^A_{ais}(P')) && \text{By definition of } [\![C^{\text{fix}}]\!]^A_{ais} \\
&\leq_A lfp(\lambda P'.Q \vee_A [\![C]\!]^A_{ais}(P')) && \text{By theorem 1.4} \\
&= [\![C^{\text{fix}}]\!]^A_{ais}(Q) && \text{By definition of } [\![C^{\text{fix}}]\!]^A_{ais}
\end{aligned}$$

□

Lemma 2.2 ($[\![\cdot]\!]$ well-defined)**.** *$\forall\, C \in \mathbb{L}$ $[\![C]\!]^A_{ais}$ is well-defined.*

Proof. From theorems 2.3 and 1.2 all the least fixpoints in the definition of $[\![C^{\text{fix}}]\!]^A_{ais}$ exists; for all the other commands the semantics is trivially well-defined. □

From now on we will refer to the complete lattice used to define the abstract inductive semantics as *domain* borrowing the convention from abstract interpretation.

Observation 2.3. When picking as a domain the lattice $\wp(\mathbb{S})$ and as base commands $[\![b]\!]^{\wp(\mathbb{S})}_{base}(P) = \{[\![b]\!]_{base}(\sigma) \downarrow \mid \sigma \in P\}$ will result in obtaining the denotational semantics from the abstract inductive semantics. $\forall\, C \in \mathbb{L}$ $\forall P \in \wp(\mathbb{S})$

$$[\![C]\!]^{\wp(\mathbb{S})}_{ais}(P) = [\![C]\!](P)$$

This can be easily assessed by comparing the two definitions.

From this observation, we can see that lemma 2.1 is just a special case of lemma 2.2 since, as shown in theorem 2.1, $\wp(\mathbb{S})$ is a complete lattice and the semantics of the base commands is monotone by construction.

2.2.1 Connection with Abstract Interpretation

As states before, the definition of abstract inductive semantics is closely related to the one of abstract semantics in [CC77].

In particular, the definition of abstract inductive semantics, when the semantics of the base commands is sound, is equivalent to an abstract semantics.

Theorem 2.4 (Abstract Interpretation Basis)**.** *If A is an abstract domain and $[\![\cdot]\!]^A_{base}$ is a sound over-approximation of $[\![\cdot]\!]_{base}$, then $[\![\cdot]\!]^A_{ais}$ is a sound over-approximation of $[\![\cdot]\!]$.*

This connection also allows us to obtain abstract inductive semantics through Galois insertion.

Definition 2.4 (Abstract Inductive Semantics by Galois Insertion)**.** Let $\langle C, \sqsubseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \leq\rangle$ be a Galois insertion, and let $[\![C]\!]^C_{ais}$ be some abstract inductive semantics defined on C. Then, the abstract inductive semantics defined on A with $[\![b]\!]^A_{base} \overset{\text{def}}{=} \alpha \circ [\![c]\!]^C_{base} \circ \gamma$ is the abstract inductive semantics obtained by the Galois insertion between C and A.

The abstract inductive semantics obtained by Galois insertion between $\wp(\mathbb{S})$ and any domain A corresponds to the best abstract inductive interpreter on A.

Observation 2.4. There are some domains where $\exists\ C \in \mathbb{L}$ such that $\bigvee_{n\in\mathbb{N}}([\![C]\!]^A_{ais})^n(P) \neq \text{lfp}(\lambda P'.P \vee_A [\![C]\!]^A_{ais}(P'))$.

2.3 Abstract Hoare Logic

2.3.1 Hoare logic

Hoare logic [Hoa69; Flo93] was one of the first method designed for the verification of programs, is based on the core concept of partial correctness assertions. A triple is a formula $\{P\}\ C\ \{Q\}$ where P and Q are assertions on the initial and final states of running program C, respectively. These assertions are partial in the sense that Q is meaningful only when the execution of C on P terminates.

Hoare logic is organized as a proof system, where the syntax $\vdash \{P\}\ C\ \{Q\}$ indicates that the triple $\{P\}\ C\ \{Q\}$ is proved by some combination of rules of the proof system.

The original formulation of Hoare logic was given for an imperative language with deterministic constructs, but it can be easily translated for our language $\mathbb{L}$ following the work in [MOH21].

Definition 2.5 (Hoare triple)**.** Fixed the semantics of the base commands, an Hoare triple written $\{P\}\ C\ \{Q\}$ is valid if and only if $[\![C]\!](P) \subseteq Q$.

$$\models \{P\}\ C\ \{Q\} \iff [\![C]\!](P) \subseteq Q$$

We will use the syntax $\models \{P\}\ C\ \{Q\}$ to refer to valid triples, $\not\models \{P\}\ C\ \{Q\}$ to refer to invalid triples, and $\{P\}\ C\ \{Q\}$ when we are not asserting the validity or invalidity of a triple.

Example 2.3 (Some Hoare triples examples)**.** $\models \{x \in [1, 2]\}\ x := x + 1\ \{x \in [2, 3]\}$, for example, is a valid triple since from any state in which either $x = 1$ or $x = 2$, incrementing by one the value of x leads to states in which x is either 2 or 3. Specifically, starting from $x = 1$ leads us to $x = 2$ and starting from $x = 2$ leads us to $x = 3$.

Since the conclusion of Hoare triples must contain all the final states, the triple $\models \{P\}\ C\ \{\top\}$ is always valid since $\top$ contains all possible states.

An example of an invalid triple is $\not\models \{x \in [1, 2]\}\ x := x + 1\ \{x \in [1, 2]\}$ since the state $x = 2$ satisfies the precondition and running the program on it results in the state $x = 3$, which does not satisfy $x \in [1, 2]$.

Since Hoare logic is concerned only with termination, when the program is non-terminating, we can prove any property. For example, $\models \{x \in [0, 10]\}\ (x \leq 20? \fatsemi x := x - 1)^{\text{fix}} \fatsemi x \geq 20?\ \{Q\}$ is always a valid triple since the program is non-terminating for any $x \in [0, 10]$. The set of reachable states is empty, thus the postcondition is vacuously true.

This is the reason why Hoare logic is said to be a partial correctness logic, as it is partial in the sense that it can prove the adherence of a program to some specification only when it is terminating. The termination of the program must be assessed in some other way.

Definition 2.6 (Hoare logic)**.**

$$\frac{}{\vdash \{P\}\ \mathbb{1}\ \{P\}}\ (\mathbb{1})$$

$$\frac{}{\vdash \{P\}\ b\ \{[\![b]\!]_{base}(P)\}}\ (base)$$

$$\frac{\vdash \{P\}\ C_1\ \{Q\} \qquad \vdash \{Q\}\ C_2\ \{R\}}{\vdash \{P\}\ C_1 \fatsemi C_2\ \{R\}}\ (seq)$$

$$\frac{\vdash \{P\}\ C_1\ \{Q\} \qquad \vdash \{P\}\ C_2\ \{Q\}}{\vdash \{P\}\ C_1 + C_2\ \{Q\}}\ (disj)$$

$$\frac{\vdash \{P\}\ C\ \{P\}}{\vdash \{P\}\ C^{\text{fix}}\ \{P\}}\ (iterate)$$

$$\frac{P \subseteq P' \qquad \vdash \{P'\}\ C\ \{Q'\} \qquad Q' \subseteq Q}{\vdash \{P\}\ C\ \{Q\}}\ (consequence)$$

The proof system described in Definition 2.6 is logically sound, meaning that all the triples provable by it are valid with respect to the definition in 2.5. This result was already present in the original work [Hoa69].

Theorem 2.5 (Soundness)**.**

$$\vdash \{P\}\ C\ \{Q\} \implies \models \{P\}\ C\ \{Q\}$$

As observed by Cook in [Coo78], the reverse implication is not true in general, as a consequence of Gödel's incompleteness theorem. For this reason, Cook developed the concept of relative completeness, in which all instances of $\subseteq$ are provided by an oracle, proving that the incompleteness of the proof system is only caused by the incompleteness of the assertion language.

Theorem 2.6 (Relative completeness)**.**

$$\models \{P\}\ C\ \{Q\} \implies \vdash \{P\}\ C\ \{Q\}$$

2.3.2 Abstracting Hoare logic

The idea of developing a Hoare-like logic to reason about properties of programs expressible within the theory of lattices using concepts from abstract interpretation is not new. In fact, [Cou+12] already proposed a framework to perform this kind of reasoning. However, the validity of such triples is dependent on the standard definition of Hoare triples, and the proof system provided is incomplete if we ignore the rule to embed standard Hoare triples in the abstract ones.

Our approach will be different. In particular, the meaning of abstract Hoare triples will be dependent on the abstract inductive semantics, and we will provide a sound and (relatively) complete proof system that fully operates in the abstract.

Definition 2.7 (Abstract Hoare triple)**.** Given an abstract inductive semantics $[\![\cdot]\!]^A_{ais}$ on the complete lattice A, the abstract Hoare triple written $\langle P \rangle_A\ C\ \langle Q \rangle$ is valid if and only if $[\![C]\!]^A_{ais}(P) \leq_A Q$.

$$\models \langle P \rangle_A\ C\ \langle Q \rangle \iff [\![C]\!]^A_{ais}(P) \leq_A Q$$

The definition is equivalent as the one provided in definition 2.5 but here the abstract inductive semantics is used to procide the strongest postcondition of programs.

In Abstract Hoare logic some of the examples show in example 2.3 still hold, in particular:

Theorem 2.7.

$$\models \langle P \rangle_A\ C\ \langle \top \rangle$$

Proof.

$$\models \langle P \rangle_A\ C\ \langle \top \rangle \iff [\![C]\!]^A_{ais}(P) \leq \top \qquad \text{By definition of } \langle \cdot \rangle_A\ \cdot\ \langle \cdot \rangle$$

And since $\top$ is the top element of A $\top \geq [\![C]\!]^A_{ais}(P)$ □

Proof system

As per Hoare logic we will peovide a sound an relatively complete (in the sense of [Coo78]) proof system to derive valid abstract Hoare triples in a compositional manner.

Definition 2.8 (Abstract Hoare rules)**.**

$$\frac{}{\vdash \langle P \rangle_A\ \mathbb{1}\ \langle P \rangle}\ (\mathbb{1})$$

The identity command does not change the state, so if P holds before, it will hold after the execution.

$$\frac{}{\vdash \langle P \rangle_A \; b \; \langle [\![b]\!]^A_{base}(P) \rangle} \; (b)$$

For a basic command b, if P holds before the execution, then $[\![b]\!]^A_{base}(P)$ holds after the execution.

$$\frac{\vdash \langle P \rangle_A \; C_1 \; \langle Q \rangle \qquad \vdash \langle Q \rangle_A \; C_2 \; \langle R \rangle}{\vdash \langle P \rangle_A \; C_1 \fatsemi C_2 \; \langle R \rangle} \; (\fatsemi)$$

If executing C_1 from state P leads to state Q, and executing C_2 from state Q leads to state R, then executing C_1 followed by C_2 from state P leads to state R.

$$\frac{\vdash \langle P \rangle_A \; C_1 \; \langle Q \rangle \qquad \vdash \langle P \rangle_A \; C_2 \; \langle Q \rangle}{\vdash \langle P \rangle_A \; C_1 + C_2 \; \langle Q \rangle} \; (+)$$

If executing either C_1 or C_2 from state P leads to state Q, then executing the nondeterministic choice $C_1 + C_2$ from state P also leads to state Q.

$$\frac{\vdash \langle P \rangle_A \; C \; \langle P \rangle}{\vdash \langle P \rangle_A \; C^{\text{fix}} \; \langle P \rangle} \; (\text{fix})$$

If executing command C from state P leads back to state P, then executing C repeatedly (zero or more times) from state P also leads back to state P.

$$\frac{P \leq P' \qquad \vdash \langle P' \rangle_A \; C \; \langle Q' \rangle \qquad Q' \leq Q}{\vdash \langle P \rangle_A \; C \; \langle Q \rangle} \; (\leq)$$

If P is stronger than P' and Q' is stronger than Q, then we can derive $\langle P \rangle_A \; C \; \langle Q \rangle$ from $\langle P' \rangle_A \; C \; \langle Q' \rangle$.

The proofsystem in nonother than the proofsystem of definition 2.6 where the assertion are replaced by elements of the complete lattice A.

Note that we denonte abstract hoare triples as defined in defintion 2.7 with the notation $\langle P \rangle_A \; C \; \langle Q \rangle$ and intread we denote the triples obtained with the inference rules of definition 2.8 with $\vdash \langle P \rangle_A \; C \; \langle Q \rangle$.

The proofsystem for Abstract Hoare logic is still sound, as the original Hoare logic.

Theorem 2.8 (Soundness)**.**

$$\vdash \langle P \rangle_A \; C \; \langle Q \rangle \implies \models \langle P \rangle_A \; C \; \langle Q \rangle$$

Proof. By structural induction on the last rule applied in the derivation of $\vdash \langle P \rangle_A \; C \; \langle Q \rangle$:

- $(\mathbb{1})$: Then the last step in the derivation was:

$$\frac{}{\vdash \langle P \rangle_A \; \mathbb{1} \; \langle P \rangle} \; (\mathbb{1})$$

The triple is valid since:

$$[\![\mathbb{1}]\!]^A_{ais}(P) = P \qquad \text{By definition of } [\![\cdot]\!]^A_{ais}$$

- (b): Then the last step in the derivation was:

$$\frac{}{\vdash \langle P \rangle_A \; b \; \langle [\![b]\!]^A_{base}(P) \rangle} \; (b)$$

The triple is valid since:

$$[\![b]\!]^A_{ais}(P) = [\![b]\!]^A_{base}(P) \qquad \text{By definition of } [\![\cdot]\!]^A_{ais}$$

- ($\fatsemi$): Then the last step in the derivation was:

$$\frac{\vdash \langle P\rangle_A\ C_1\ \langle Q\rangle \qquad \vdash \langle Q\rangle_A\ C_2\ \langle R\rangle}{\vdash \langle P\rangle_A\ C_1 \fatsemi C_2\ \langle R\rangle}\ (\fatsemi)$$

By inductive hypothesis: $[\![C_1]\!]^A_{ais}(P) \leq_A Q$ and $[\![C_2]\!]^A_{ais}(Q) \leq_A R$.

The triple is valid since:

$$\begin{aligned} [\![C_1 \fatsemi C_2]\!]^A_{ais}(P) &= [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P)) && \text{By definition of } [\![\cdot]\!]^A_{ais} \\ &\leq_A [\![C_2]\!]^A_{ais}(Q) && \text{By monotonicity of } [\![\cdot]\!]^A_{ais} \\ &\leq_A R \end{aligned}$$

- (+): Then the last step in the derivation was:

$$\frac{\vdash \langle P\rangle_A\ C_1\ \langle Q\rangle \qquad \vdash \langle P\rangle_A\ C_2\ \langle Q\rangle}{\vdash \langle P\rangle_A\ C_1 + C_2\ \langle Q\rangle}\ (+)$$

By inductive hypothesis: $[\![C_1]\!]^A_{ais}(P) \leq Q$ and $[\![C_2]\!]^A_{ais}(P) \leq Q$.

The triple is valid since:

$$\begin{aligned} [\![C_1 + C_2]\!]^A_{ais}(P) &= [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P) && \text{By definition of } [\![\cdot]\!]^A_{ais} \\ &\leq_A Q \vee_A Q \\ &= Q \end{aligned}$$

- (fix): Then the last step in the derivation was:

$$\frac{\vdash \langle P\rangle_A\ C\ \langle P\rangle}{\vdash \langle P\rangle_A\ C^{\text{lfp}}\ \langle P\rangle}\ (\text{fix})$$

By inductive hypothesis: $[\![C]\!]^A_{ais}P \leq P$

$$[\![C^{\text{fix}}]\!]^A_{ais}(P) = \text{lfp}(\lambda P' \to P \vee_A [\![C]\!]^A_{ais}(P'))$$

We will show that P is a fixpoint of $\lambda P' \to P \vee_A [\![C]\!]^A_{ais}(P')$.

$$\begin{aligned} (\lambda P' \to P \vee_A [\![C]\!]^A_{ais}(P'))(P) &= P \vee_A [\![C]\!]^A_{ais}(P) && \text{since } [\![C]\!]^A_{ais}(P) \leq P \\ &= P \end{aligned}$$

Hence P is a fixpoint of $\lambda P' \to P \vee_A [\![C]\!]^A_{ais}(P')$, therefore it's at leas as big as the least one, $\text{lfp}(\lambda P' \to P \vee_A [\![C]\!]^A_{ais}(P')) \leq_A P$ thus making the triple valid.

- ($\leq$): Then the last step in the derivation was:

$$\frac{P \leq P' \qquad \vdash \langle P'\rangle_A\ C\ \langle Q'\rangle \qquad Q' \leq Q}{\vdash \langle P\rangle_A\ C\ \langle Q\rangle}\ (\leq)$$

By inductive hypothesis: $[\![C]\!]^A_{ais}(P') \leq_A Q'$.

The triple is valid since:

$$
\begin{aligned}
[\![C]\!]^A_{ais}(P)[\![C]\!]^A_{ais}(P') & & \text{By monotonicity of } [\![\cdot]\!]^A_{ais} \\
\leq_A Q' & & \text{By inductive hypothesis} \\
\leq_A Q & &
\end{aligned}
$$

□

And is also relatively complete, in the sense that the axioms are complete relative to what we can prove in the underlying assertion language, that in our case is described by the complete lattice.

We will start by proving a slightly weaker result, we will show that we are able to prove the strongest post-condition of every program.

Theorem 2.9 (Relative $[\![\cdot]\!]^A_{ais}$-completeness)**.**

$$\vdash \langle P \rangle_A \; C \; \langle [\![C]\!]^A_{ais}(P) \rangle$$

Proof. By structural induction on C:

- $\mathbb{1}$: By definition $[\![\mathbb{1}]\!]^A_{ais}(P) = P$

$$\frac{}{\vdash \langle P \rangle_A \; \mathbb{1} \; \langle P \rangle} \; (\mathbb{1})$$

- b: By definition $[\![b]\!]^A_{ais}(P) = [\![b]\!]^A_{base}(P)$

$$\frac{}{\vdash \langle P \rangle_A \; b \; \langle [\![b]\!]^A_{base}(P) \rangle} \; (b)$$

- $C_1 \fatsemi C_2$: By definition $[\![C_1 \fatsemi C_2]\!]^A_{ais}(P) = [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P))$

$$\frac{\begin{array}{cc} \text{(Inductive hypothesis)} & \text{(Inductive hypothesis)} \\ \vdash \langle P \rangle_A \; C_1 \; \langle [\![C_1]\!]^A_{ais}(P) \rangle & \vdash \langle [\![C_1]\!]^A_{ais}(P) \rangle_A \; C_2 \; \langle [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P)) \rangle \end{array}}{\vdash \langle P \rangle_A \; C_1 \fatsemi C_2 \; \langle [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P)) \rangle} \; (\fatsemi)$$

- $C_1 + C_2$: By definition $[\![C_1 + C_2]\!]_{base}(P) = [\![C_1]\!]_{base}(P) \vee_A [\![C_2]\!]_{base}(P)$

$$\frac{\pi_1 \qquad \pi_2}{\vdash \langle P \rangle_A \; C_1 + C_2 \; \langle [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P) \rangle} \; (+)$$

Where π_1:

$$\frac{P \leq_A P \qquad \begin{array}{c} \text{(Inductive hypothesis)} \\ \vdash \langle P \rangle_A \; C_1 \; \langle [\![C_1]\!]^A_{ais}(P) \rangle \end{array} \qquad [\![C_1]\!]^A_{ais}(P) \leq_A [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P)}{\vdash \langle P \rangle_A \; C_1 \; \langle [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P) \rangle} \; (\leq)$$

and π_2:

$$\frac{P \leq_A P \qquad \begin{array}{c} \text{(Inductive hypothesis)} \\ \vdash \langle P \rangle_A \; C_2 \; \langle [\![C_2]\!]^A_{ais}(P) \rangle \end{array} \qquad [\![C_2]\!]^A_{ais}(P) \leq_A [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P)}{\vdash \langle P \rangle_A \; C_2 \; \langle [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P) \rangle} \; (\leq)$$

- C^{fix}: By definition $[\![C^{\text{fix}}]\!]_{base}(P) = lfp(\lambda P' \to P \vee_A [\![C]\!]^A_{ais}(S')$.

  Let $K \stackrel{\text{def}}{=} lfp(\lambda P' \to P \vee_A [\![C]\!]^A_{ais}(S')$ hence $K = P \vee_A [\![C]\!]^A_{ais}(K)$ since it is a fixpoint, thus

  - α_1: $K \geq_A P$
  - α_2: $K \geq_A [\![C]\!]^A_{ais}(K)$

$$\dfrac{\alpha_1 \qquad \dfrac{\dfrac{K \leq_A K \qquad \overset{\text{(Inductive hypothesis)}}{\vdash \langle K \rangle_A\ C\ \langle [\![C]\!]^A_{ais}(K) \rangle} \qquad \alpha_2}{\vdash \langle K \rangle_A\ C\ \langle K \rangle}}{\vdash \langle K \rangle_A\ C^{\text{fix}}\ \langle K \rangle}\text{(fix)} \qquad K \leq_A K}{\vdash \langle P \rangle_A\ C^{\text{fix}}\ \langle K \rangle}(\leq)$$

□

Now we can finally show the relative completeness, by applying the $(\leq)$ to obtain the desired post-condition.

Theorem 2.10 (Relative completeness)**.**

$$\models \langle P \rangle_A\ C\ \langle Q \rangle \implies \vdash \langle P \rangle_A\ C\ \langle Q \rangle$$

Proof. By definition of $\models \langle P \rangle_A\ C\ \langle Q \rangle \iff Q \geq_A [\![C]\!]^A_{ais}(P)$

$$\dfrac{P \leq_A P \qquad \overset{\text{(By Theorem 2.9)}}{\vdash \langle P \rangle_A\ C\ \langle [\![C]\!]^A_{ais}(P) \rangle} \qquad Q \geq_A [\![C]\!]^A_{ais}(P)}{\vdash \langle P \rangle_A\ C\ \langle Q \rangle}(\leq)$$

□

CHAPTER 3

INSTANTIATING ABSTRACT HOARE LOGIC

In this chapter, we will show how to instantiate abstract Hoare logic to create new proof systems. We will also demonstrate that the framework of abstract Hoare logic is so general that, in some instantiations, it is able to reason about properties that are not expressible in standard Hoare logic.

3.1 Hoare logic

Following Observation 2.3, the abstract inductive semantics, when using $(\wp(\mathbb{S}), \subseteq)$ as the domain and $[\![b]\!]^{\wp(\mathbb{S})}_{base}(P) = \{[\![b]\!]_{base}(\sigma) \downarrow \mid \sigma \in P\}$ as the base command semantics, is equivalent to the denotational semantics given in Definition 2.2. As we can see from the definition of Hoare logic (Definition 2.5) and Abstract Hoare logic (Definition 2.7), they are equivalent. Hence, in this abstraction, Abstract Hoare Logic and Hoare Logic have the same formulation. Since both proof systems are sound and (relatively) complete, they are equivalent.

3.2 Absract Interval Logic and Algebraic Hoare Logic

3.2.1 Algebraic Hoare Logic

As explained in section 2.3, Abstract Hoare Logic was inspired by Algebraic Hoare Logic [Cou+12]. Both logics can be used to prove properties chosen in computer-representable abstract domains.

Definition 3.1 (Algebraic Hoare triple)**.** Given two Galois insertions $\langle \wp(\mathbb{S}), \subseteq \rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle A, \leq \rangle$ and $\langle \wp(\mathbb{S}), \subseteq \rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle B, \sqsubseteq \rangle$, an Algebraic Hoare triple written $\overline{\{}P\overline{\}}\ C\ \overline{\{}Q\overline{\}}$ is valid if and only if $\{\gamma_1(P)\}\ C\ \{\gamma_2(Q)\}$ is valid.

$$\models \overline{\{}P\overline{\}}\ C\ \overline{\{}Q\overline{\}} \iff \models \{\gamma_1(P)\}\ C\ \{\gamma_2(Q)\}$$

From this definition, we see that the definition of Algebraic Hoare Logic is deeply connected to standard Hoare Logic and thus to the strongest postcondition of the program in the concrete domain.

Definition 3.2 (Algebraic Hoare logic proof system[1])**.**

$$\frac{}{\vdash \overline{\{}\bot_1\overline{\}}\ C\ \overline{\{}Q\overline{\}}}\ (\overline{\bot})$$

$$\frac{}{\vdash \overline{\{}P\overline{\}}\ C\ \overline{\{}\top_2\overline{\}}}\ (\overline{\top})$$

[1] Rules $(\overline{\vee})$ and $(\overline{\wedge})$ are missing but will be discussed in section 4.1

$$\frac{\models \{\gamma_1(P)\}\ C\ \{\gamma_2(Q)\}}{\vdash \overline{\{}P\overline{\}}\ C\ \overline{\{}Q\overline{\}}}\ (\overline{S})$$

$$\frac{P \leq P' \qquad \vdash \overline{\{}P'\overline{\}}\ C\ \overline{\{}Q'\overline{\}} \qquad Q' \sqsubseteq Q}{\vdash \overline{\{}P\overline{\}}\ C\ \overline{\{}Q\overline{\}}}\ (\Rrightarrow)$$

This proof system highlights that most of the work is done by rule $(\overline{S})$, which embeds Hoare triples in Algebraic Hoare triples. One can easily prove that the proof system is relatively complete from the relative completeness of Hoare logic. In particular, only the $(\overline{S})$ rule is actually needed, since all the implications in the abstract must also hold in the concrete.

3.2.2 Abstract Interval Logic

As shown in Definition 2.4, via a Galois insertion we can obtain a similar family of triples as those in Algebraic Hoare Logic when the abstract domains used in the pre- and post-conditions are the same.

Example 3.1 (Interval logic)**.** Applying Definition 2.4 to the Galois insertion on the interval domain defined in Example 1.1, we obtain a sound and relatively complete logic to reason about properties of programs that are expressible as intervals.

Example 3.2 (Derivation in interval logic)**.** Let $C \overset{\text{def}}{=} ((x := 1) + (x := 3)) \fatsemi (x = 2? \fatsemi x := 5) + (x \neq 2? \fatsemi x := x - 1)$

Then the following derivation is valid:

$$\frac{\pi_1 \qquad \pi_3}{\vdash \langle \top \rangle_{Int}\ C\ \langle [0,5] \rangle}\ (\fatsemi)$$

π_1:

$$\frac{\dfrac{\top \leq \top \qquad \dfrac{}{\vdash \langle \top \rangle_{Int}\ x := 1\ \langle [1,1] \rangle}\,(b) \qquad [1,1] \leq [1,3]}{\vdash \langle \top \rangle_{Int}\ x := 1\ \langle [1,3] \rangle} \qquad \pi_2}{\vdash \langle \top \rangle_{Int}\ (x := 1) + (x := 3)\ \langle [1,3] \rangle}\ (+)$$

π_2:

$$\frac{\top \leq \top \qquad \dfrac{}{\vdash \langle \top \rangle_{Int}\ x := 3\ \langle [3,3] \rangle}\,(b) \qquad [3,3] \leq [1,3]}{\vdash \langle \top \rangle_{Int}\ x := 3\ \langle [1,3] \rangle}\ (\leq)$$

π_3:

$$\frac{\pi_4 \qquad \pi_5}{\vdash \langle [1,3] \rangle_{Int}\ (x = 2? \fatsemi x := 5) + (x \neq 2? \fatsemi x := x - 1)\ \langle [0,5] \rangle}\ (+)$$

π_4:

$$\frac{[1,3] \leq [1,3] \qquad \dfrac{\dfrac{}{\vdash \langle [1,3] \rangle_{Int}\ x = 2?\ \langle [2] \rangle}\,(b) \qquad \dfrac{}{\vdash \langle [2] \rangle_{Int}\ x := 5\ \langle [5] \rangle}\,(b)}{\vdash \langle [1,3] \rangle_{Int}\ x = 2? \fatsemi x := 5\ \langle [5] \rangle}\,(\fatsemi) \qquad [5,5] \leq [0,5]}{\vdash \langle [1,3] \rangle_{Int}\ x = 2? \fatsemi x := 5\ \langle [0,5] \rangle}\ (\leq)$$

π_5:

$$\frac{[1,3] \leq [1,3] \qquad \pi_6 \qquad [0,2] \leq [0,5]}{\vdash \langle [1,3] \rangle_{Int}\ x \neq 2? \fatsemi x := x - 1\ \langle [0,5] \rangle}$$

π_6 :

$$\frac{\dfrac{}{\vdash \langle [1,3] \rangle_{Int}\ x \neq 2?\ \langle [1,3] \rangle}\,(b) \qquad \dfrac{}{\vdash \langle [1,3] \rangle_{Int}\ x := x - 1\ \langle [0,2] \rangle}\,(b)}{\vdash \langle [1,3] \rangle_{Int}\ x \neq 2? \fatsemi x := x - 1\ \langle [0,2] \rangle}\ (\fatsemi)$$

This is also the best we can derive since $[\![C]\!]^{Int}_{ais}(\top) = [0,5]$.

Applications

This framework, like Algebraic Hoare Logic, can be used to specify how a static analyzer for a given abstract domain should work. Since $\llbracket\cdot\rrbracket^A_{ais}$ is the best abstract analyzer on abstract domain A when it is defined inductively, and since the whole proof system is in the abstract, we can check that a derivation is indeed correct algorithmically (as long as we can check implications and base commands). These are usually the standard requirements for an abstract domain to be useful. The same does not hold for Algebraic Hoare Logic since deciding the validity of arbitrary triples would require deciding the validity of standard Hoare logic triples, and in general, we cannot decide implications between arbitrary properties.

3.2.3 Relationship

Clearly, Algebraic Hoare Logic can derive the same triples that are derivable by Abstract Hoare Logic when instantiated through a Galois insertion from $\wp(\mathbb{S})$ as in Example 3.1. From Theorem 2.4, $\llbracket\cdot\rrbracket^A_{ais}$ is a sound overapproximation of $\llbracket\cdot\rrbracket$.

Theorem 3.1. $\vdash \langle P\rangle_A\ C\ \langle Q\rangle \implies \vdash \overline{\{}P\overline{\}}\ C\ \overline{\{}Q\overline{\}}$

Proof.

$$\begin{aligned}
\vdash \langle P\rangle_A\ C\ \langle Q\rangle &\implies \llbracket C\rrbracket^A_{ais}(P) \leq Q && \text{From Theorem 2.8}\\
&\implies \llbracket C\rrbracket(\gamma(P)) \subseteq \gamma(Q) && \text{From Theorem 2.4}\\
&\implies \vdash \{\gamma(P)\}\ C\ \{\gamma(Q)\} && \text{From Theorem 2.6}\\
&\implies \vdash \overline{\{}P\overline{\}}\ C\ \overline{\{}Q\overline{\}} && \text{From rule } (\overline{S})
\end{aligned}$$

□

However, the converse is not true. The relative completeness of Algebraic Hoare Logic is with respect to the best correct approximation of $\llbracket\cdot\rrbracket$ and not with respect to $\llbracket\cdot\rrbracket^A_{ais}$ as in Abstract Hoare Logic.

Example 3.3 (Counter example)**.** From Example 3.2, we know that $\vdash \langle\top\rangle_A\ C\ \langle[0,5]\rangle$ is the best Abstract Hoare triple that we can obtain, but $\llbracket C\rrbracket\top = \{0,2\}$. Via Theorem 2.6, we can obtain $\vdash \{\top\}\ C\ \{\{0,2\}\}$. Hence, via the $(\overline{S})$ rule, we can obtain $\vdash \overline{\{}\top\overline{\}}\ C\ \overline{\{}[0,2]\overline{\}}$, which is unobtainable in Abstract Hoare Logic.

This is clearly a consequence that via the (S) rule in Algebraic Hoare logic we are able to prove the best correct approximation of any program C, but the property of being a best correct approximation does not "compose", meaning that the function composition of two best correct approximations is not the best correct approximation of the composition of the functions. Since in the abstract semantics the program composition is done in "the abstract," it's impossible to expect to be able to obtain any possible best correct approximation except in trivial abstract domains like the concrete $\wp(\mathbb{S})$ or the one-element lattice.

3.3 Hoare logic for hyperproperties

3.3.1 Introduction to Hyperproperties

Hyperproperties, introduced in [CS08], extend traditional program properties by considering relationships between multiple executions of a program, rather than focusing on individual traces. This concept is essential for reasoning about security and correctness properties that involve comparisons across different executions, such as non-interference, information flow security, and program equivalence.

Standard properties, like those utilized in Hoare logic, are elements of the set $\wp(\mathbb{S})$. In contrast, hyperproperties are elements of the set $\wp(\wp(\mathbb{S}))$ since as said before they encode relation between different executions. A common example is the property of a program being deterministic. Suppose our programs have only one integer variable named x. To prove that a program C is deterministic, we would need to prove an infinite number of Hoare triples of the form: for each value of $n \in \mathbb{N}$, there must exist $m \in \mathbb{N}$ such that $\{\{x = n\}\}\ C\ \{\{x = m\}\}$ is valid. Instead, determinism can be easily encoded in a single hyper triple: $\{\{P \in \wp(\wp(\mathbb{S})) \mid |P| = 1\}\}\ C\ \{\{Q \in \wp(\wp(\mathbb{S})) \mid |Q| = 1\}\}$.

Definition 3.3 (Strongest Hyper Postcondition)**.** The strongest postcondition of a program C starting from a collection of states $\chi \in \wp(\wp(\mathbb{S}))$ is defined as:

$$\{[\![C]\!](P) \mid P \in \chi\}$$

3.3.2 Inductive Definition of the Strongest Hyper Post Condition

To obtain a sound and (relative) complete logic for hyperproperties using our framework, we need to construct an abstract semantics that computes exactly that property. This problem was already studied in [Ass+17; MP18] but in the context of abstract interpretation. In all of them, what was obtained was an overapproximation of the strongest hyper postcondition that in abstract interpretation is enough but in our context isn't if we want to keep the relative completeness. In particular in [Ass+17], the hyper semantics of `if b then` C_1 `else` C_2 is given as (translated in $\mathbb{L}$) $\{[\![b? \mathbin{⨟} C_1]\!]T \cup [\![\neg b? \mathbin{⨟} C_2]\!] \mid T \in \mathbb{T}\}$, thus making the definition non-inductive. In particular, given any program C, we can perform the analysis of `if true then` C and perform the analysis of any program without practically ever using the hyper semantics

The root of the problem is that in $\wp(\wp(\mathbb{S}))$ with the standard ordering on the powerset, the least upper bound is unable to distinguish between different executions.

Example 3.4. Let $\chi \overset{\text{def}}{=} \{\{1, 2, 3\}, \{5\}\}$. Clearly,

$$[\![(x := x + 1) + (x := x + 2)]\!]^{\wp(\wp(\mathbb{S}))}_{ais}(\chi) = \{\{2, 3, 4\}, \{6\}, \{3, 4, 5\}, \{7\}\},$$

which is totally different from the strongest hyper postcondition, which is $\{\{2, 3, 4, 5\}, \{6, 7\}\}$.

When applying the rule for the non-deterministic choice, $[\![C_1{+}C_2]\!]^{\wp(\wp(\mathbb{S}))}_{ais}(P) = [\![C_1]\!]^{\wp(\wp(\mathbb{S}))}_{ais}(P) \cup [\![C_2]\!]^{\wp(\wp(\mathbb{S}))}_{ais}(P)$, we are performing the union of the outer sets instead of the inner ones that contain the actual executions. We might think that we could solve this issue by modifying the order defined on $\wp(\wp(\mathbb{S}))$, but each set does not carry any information on which execution actually generated it. Therefore, it is impossible to construct a union that does not lose precision.

To our knowledge, there is no literature on an abstract inductive semantics that exactly computes the strongest hyper postcondition, but only some sound over-approximation. This is sufficient when trying to develop an abstract interpreter, as clearly it will also be lost when switching from the concrete $\wp(\wp(\mathbb{S}))$ to an abstract domain that is actually machine-representable will be used to run the interpreter. But in our case we are interest to being able to obtain the strongest postcondition possible as having some overapproximation will make this particular instance of Abstract Hoare logic unable to prove some hyperproperties, namely the one on with the abstract inductive semantics is imprecise.

3.3.3 Hyper Domains

To address the issues caused by $\wp(\wp(\mathbb{S}))$, we will define a more complex family of domains whose semantics satisfy the distributive property of different executions. We will use a set K to keep track of each execution and define the join operation in such a way that it does not confuse different executions together.

Definition 3.4 (Hyper Domain)**.** Given a complete lattice B and a set K, the hyper domain $H(B)_K$ is defined as:

$$H(B)_K \overset{\text{def}}{=} K \to B + \mathit{undef}.$$

The complete lattice of $H(B)_K$ is the pointwise lift of the one defined on B + undef, where B + undef is the complete lattice defined on B with *undef* added as a new bottom element.

Since the role of K, as stated before, is only to index the different executions, what actually it's used it's not important, the only property of interest is having at least as many elements as the number of executions that we want to keep track of.

Definition 3.5 (Hyper Instantiation)**.** Given an instantiation of the abstract inductive semantics on a domain B with semantics of the base commands $[\![\cdot]\!]^B_{base}$, we can instantiate the abstract inductive semantics for the hyper domain $H(B)_K$ with base commands defined as follows:

$$[\![b]\!]^{H(B)_K}_{base}(\chi) \stackrel{\text{def}}{=} \lambda r \to [\![b]\!]^B_{base}(\chi(r))$$

The idea of defining hyper instantiation it to lift the abstract inductive semantics on some domain B in it's "hyper" version, the definition of the base commands applies the semantics of the base commands on B to each execution.

Now we prove that the abstract inductive semantics when instantiated on an hyper-domain is non interfering, in the sense that running the hyper inductive semantics is the same as running the original semantics on each execution.

Theorem 3.2 (Non interference between executions)**.**

$$[\![C]\!]^{H(B)_K}_{ais}(\chi) = \lambda r \to [\![C]\!]^B_{ais}(\chi(r))$$

Proof. By structural induction on C:

- $\mathbb{1}$:

$$\begin{aligned}
[\![\mathbb{1}]\!]^{H(B)_K}_{ais}(\chi) &= \chi && \text{By definition of } [\![\cdot]\!]^{H(B)_K}_{ais} \\
&= \lambda r \to \chi(r) && \text{By extensionality} \\
&= \lambda r \to [\![\mathbb{1}]\!]^B_{ais}(\chi(r)) && \text{By definition of } [\![\cdot]\!]^B_{ais}
\end{aligned}$$

- b:

$$[\![b]\!]^{H(B)_K}_{ais}(\chi) = \lambda r \to [\![b]\!]^B_{ais}(\chi(r))$$

- $C_1 \fatsemi C_2$:

$$\begin{aligned}
[\![C_1 \fatsemi C_2]\!]^{H(B)_K}_{ais}(\chi) &= [\![C_2]\!]^{H(B)_K}_{ais}([\![C_1]\!]^{H(B)_K}_{ais}(\chi)) && \text{By definition of } [\![\cdot]\!]^{H(B)_K}_{ais} \\
&= [\![C_2]\!]^{H(B)_K}_{ais}(\lambda r_1 \to [\![C_1]\!]^B_{ais}(\chi(r_1)) && \text{By inductive hypothesis} \\
&= \lambda r_2 \to [\![C_2]\!]^B_{ais}(\lambda r_1 \to [\![C_1]\!]^B_{ais}(\chi(r_1))(r_2)) && \text{By inductive hypothesis} \\
&= \lambda r_2 \to [\![C_2]\!]^B_{ais}([\![C_1]\!]^B_{ais}(\chi(r_2))) \\
&= \lambda r_2 \to [\![C_1 \fatsemi C_2]\!]^B_{ais}(\chi(r_2)) && \text{By definition of } [\![\cdot]\!]^B_{ais}
\end{aligned}$$

- $C_1 + C_2$:

$$\begin{aligned}
[\![C_1 + C_2]\!]^{H(B)_K}_{ais}(\chi) = {} & [\![C_1]\!]^{H(B)_K}_{ais}(\chi) \vee [\![C_2]\!]^{H(B)_K}_{ais}(\chi) \\
& \text{By definition of } [\![\cdot]\!]^{H(B)_K}_{ais} \\
& = (\lambda r_1 \to [\![C_1]\!]^B_{ais}(\chi(r_1))) \vee (\lambda r_2 \to [\![C_1]\!]^B_{ais}(\chi(r_2))) \\
& \text{By inductive hypothesis} \\
& = \lambda r \to [\![C_1]\!]^B_{ais}(\chi(r)) \vee [\![C_2]\!]^B_{ais}(\chi(r)) \\
& = \lambda r \to [\![C_1 + C_2]\!]^B_{ais}(\chi(r)) \\
& \text{By definition of } [\![\cdot]\!]^B_{ais}
\end{aligned}$$

- C^{fix}:

$$\begin{aligned}
[\![C^{\text{fix}}]\!]_{ais}^{H(B)_K}(\chi) &= \text{lfp}(\lambda\psi \to \chi \vee [\![C]\!]_{ais}^{H(B)_K}(\psi)) && \text{By definition of } [\![\cdot]\!]_{ais}^{H(B)_K} \\
&= \text{lfp}(\lambda\psi \to \chi \vee \lambda r \to [\![C]\!]_{ais}^{B}(\psi(r))) && \text{By inductive hypothesis} \\
&= \text{lfp}(\lambda\psi \to \lambda r \to \chi(r) \vee [\![C]\!]_{ais}^{B}(\psi(r))) && \text{By theorem 1.1} \\
&= \lambda r \to \text{lfp}(\lambda P \to \chi(r) \vee [\![C]\!]_{ais}^{B} P) && \text{By definition of } [\![\cdot]\!]_{ais}^{B} \\
&= \lambda r \to [\![C^{\text{fix}}]\!]_{ais}^{B}(\chi(r))
\end{aligned}$$

□

3.3.4 Inductive definition for Hyper postconditions

Our goal with the hyper domains was to address the issue caused by taking $\wp(\wp(\mathbb{S}))$ as the domain. However, our abstract inductive semantics now uses a different domain. To handle this, we need a way to convert the standard representation of hyperproperties to the new one using hyper domains and vice versa. To achieve this, we define a pair of functions called the conversion pair to perform the operation. Since there could be infinite functions converting a standard hyperproperty into the version using hyper domains (since we have infinite representation for the same property), we can use a single abstraction (an injective function) to represent them all. All the results are independent of the chosen indexing function.

Definition 3.6 (Conversion Pair)**.** Given an injective function $idx : B \to K$, we can define the conversion pair as follows:

$$\begin{aligned}
\alpha &: H(B)_K \to \wp(B) \\
\alpha(\chi) &\stackrel{\text{def}}{=} \{\chi(r)\downarrow \mid r \in K\}
\end{aligned}$$

$$\begin{aligned}
\beta &: \wp(B) \to H(B)_K \\
\beta(\mathcal{X}) &\stackrel{\text{def}}{=} \lambda r \to \begin{cases} P & \exists P \in \mathcal{X} \text{ such that } idx(P) = r \\ undef & \text{otherwise} \end{cases}
\end{aligned}$$

By instantiating the hyper domain as $H(\wp(\mathbb{S}))_{\mathbb{R}}$, we will be able to prove that the abstract inductive semantics of $H(\wp(\mathbb{S}))_{\mathbb{R}}$ computes the strongest hyper postcondition.

We have an infinite amount of injective functions $\wp(\mathbb{S}) \to \mathbb{R}$ since if $\mathbb{S}$ is countable then $|\wp(\mathbb{S})| = |\wp(\mathbb{N})| = |\mathbb{R}|$ thus at least one conversion pair exists, and since all the results are independent of witch one we choose we won't specify one.

Now show that by composing the conversion pair with the abstract inductive semantics, we compute exactly the strongest hyper postcondition.

Theorem 3.3 (Abstract Inductive Semantics as Strongest Hyper Postcondition)**.**

$$\alpha([\![C]\!]_{ais}^{H(\wp(\mathbb{S}))_{\mathbb{R}}}(\gamma(\mathcal{X}))) = \{[\![C]\!]_{ais}^{\wp(\mathbb{S})}(P) \mid P \in \mathcal{X}\}$$

Proof.

$$\begin{aligned}
\alpha([\![C]\!]_{ais}^{H(\wp(\mathbb{S}))_{\mathbb{R}}}(\beta(\mathcal{X}))) &= \alpha(\lambda r \to [\![C]\!]_{ais}^{\wp(\mathbb{S})}(\beta(\mathcal{X})(r))) && \text{By Theorem 3.2} \\
&= \{[\![C]\!]_{ais}^{\wp(\mathbb{S})}(\beta(\mathcal{X})(r))\downarrow \mid r \in \mathbb{R}\} && \text{By the definition of } \alpha \\
&= \{[\![C]\!]_{ais}^{\wp(\mathbb{S})}(P) \mid P \in \mathcal{X}\} && \text{By the definition of } \beta \text{ and injectivity}
\end{aligned}$$

□

3.3.5 Hyper Hoare triples

The instantiation provides us with a sound and complete Hoare-like logic for hyperproperties when we apply α on the pre and post conditions.

Example 3.5 (Determinism in Abstract Hoare Logic)**.** As explained in Example 3.4, we can express that a command is deterministic (up to termination) by proving that the hyperproperty $\{P \mid |P| = 1\}$ is both a precondition and a postcondition of the command.

Assume that we are working with $\mathbb{L}$ where assignment involves only one variable, so that we can represent states with a single integer.

The encoding of the property that we want to use as a precondition is:

$$\mathcal{P} = \lambda r \to \begin{cases} \{x\} & \exists x \in \wp(\mathbb{S}) \text{ such that } idx(P) = r \\ undef & \text{otherwise} \end{cases}$$

We can prove that the program $\mathbb{1}$ is deterministic:

$$\cfrac{}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ \mathbb{1}\ \langle P \rangle}\ (\mathbb{1})$$

Since $\alpha(P) = \{..., \{-1\}, \{0\}, \{1\}, ...\}$, we have proven that the command is deterministic.

The same can be done with the increment function:

$$\cfrac{}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ x := x + 1\ \langle Q \rangle}\ (:=)$$

Where $Q = \lambda r \to \begin{cases} \{x+1\} & \exists \{x\} \in \wp(\mathbb{S}) \text{ such that } idx(P) = r \\ undef & \text{otherwise} \end{cases}$

And clearly $\alpha(Q) = \{..., \{0\}, \{1\}, \{2\}, ...\}$, hence proving that the command is deterministic.

We can prove that a non-deterministic choice between two identical programs is also deterministic:

$$\cfrac{\cfrac{}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ x := x + 1\ \langle Q \rangle}\ (:=) \qquad \cfrac{}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ x := x + 1\ \langle Q \rangle}\ (:=)}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ (x := x + 1) + (x := x + 1)\ \langle Q \rangle}\ (+)$$

But obviously we cannot do the same with two different programs:

$$\cfrac{\cfrac{P \leq P \qquad \cfrac{}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ \mathbb{1}\ \langle P \rangle}\ (\mathbb{1}) \qquad P \leq P \vee Q}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ \mathbb{1}\ \langle P \vee Q \rangle}\ (\leq) \qquad \pi}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ \mathbb{1} + (x := x + 1)\ \langle P \vee Q \rangle}\ (+)$$

Where π:

$$\cfrac{P \leq P \qquad \cfrac{}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ x := x + 1\ \langle Q \rangle}\ (:=) \qquad Q \leq P \vee Q}{\vdash \langle P \rangle_{H(\wp(\mathbb{S}))_{\mathbb{R}}}\ x := x + 1\ \langle P \vee Q \rangle}\ (\leq)$$

And clearly $\alpha(P \vee Q) = \{..., \{-1, 0\}, \{0, 1\}, \{1, 2\}, ...\}$.

Observation 3.1. We can clearly see that different elements in the hyper domain correspond to the same hyperproperty. This is an expected behavior since the non-deterministic choice does not, in general, "preserve" hyperproperties. The same trick is performed in other logics that can express hyperproperties by adding a new disjunction operator that splits the condition.

There is already a sound and (relative) complete Hoare-like logic, Hyper Hoare Logic ([DM23]). While arguably more usable since it was developed specifically for this goal, it is equivalent to the logic obtained via the abstract Hoare logic framework. We can observe that they also had to diverge from the usage of the classical disjunction connective (which is equivalent to the least upper bound in $\wp(\wp(\mathbb{S}))$) and had to define an exotic version of disjunction ($\otimes$) that is able to distinguish between different executions. The resemblance to the least upper bound for the hyperdomains is striking.

3.4 Partial Incorrectness

Any instantiation of the abstract inductive semantics gives us another instantiation for free, since the semantics is parametric on a complete lattice A, and the dual of a complete lattice is also a complete lattice. We can obtain the dual abstract inductive semantics for free on the complete lattice A^{op}.

Definition 3.7 (Dual Abstract Inductive Semantics)**.** Given an abstract inductive semantics defined on some complete lattice A and base commands $[\![\cdot]\!]^{A}_{base}$, we can define the dual abstract inductive semantics as the abstract inductive semantics instantiated on the complete lattice A^{op} with base command semantics $[\![\cdot]\!]^{A^{op}}_{base} = [\![\cdot]\!]^{A}_{base}$.

Since the dual abstract inductive semantics is an abstract inductive semantics, it also induces an Abstract Hoare Logic. In the dual lattice, since the partial order is inverted, the roles of joins and meets are inverted, and thus also lfp and gfp are inverted. Hence, the dual abstract inductive semantics, when seen from the dual lattice, can be expressed as:

$$
\begin{aligned}
[\![\mathbb{1}]\!]^{A^{op}}_{ais} &= id & &= id \\
[\![b]\!]^{A^{op}}_{ais} &= [\![b]\!]^{A^{op}}_{base} & &= [\![b]\!]^{A}_{base} \\
[\![C_1 \mathbin{⨟} C_2]\!]^{A^{op}}_{ais} &= [\![C_2]\!]^{A^{op}}_{ais} \circ [\![C_1]\!]^{A^{op}}_{ais} & &= [\![C_2]\!]^{A^{op}}_{ais} \circ [\![C_1]\!]^{A^{op}}_{ais} \\
[\![C_1 + C_2]\!]^{A^{op}}_{ais} &= \lambda P. [\![C_1]\!]^{A^{op}}_{ais} P \vee_{A^{op}} [\![C_2]\!]^{A^{op}}_{ais} P & &= \lambda P. [\![C_1]\!]^{A^{op}}_{ais} P \wedge_{A} [\![C_2]\!]^{A^{op}}_{ais} P \\
[\![C^{\text{fix}}]\!]^{A^{op}}_{ais} &= \lambda P. \mathrm{lfp}_{A^{op}}(\lambda P'. P \vee_{A^{op}} [\![C]\!]^{A^{op}}_{ais} P') & &= \lambda P. \mathrm{gfp}_{A}(\lambda P'. P \wedge_{A} [\![C]\!]^{A^{op}}_{ais} P')
\end{aligned}
$$

Following the intuition that the abstract inductive semantics is some abstract version of the strongest postcondition, what interpretation can we give for the dual abstract inductive semantics? While on the lattice A, for the non-deterministic choice, we take the join of the two branches. In the opposite lattice, we take their meet. Following this intuition, instead of taking all the reachable states (the union of the states reached by the two branches), we take the states that we are sure we are reaching, i.e., the intersection of the states reached by the two branches. The same reasoning also holds for the fix command.

Since the order on the dual lattice is inverted, inversion holds also for the validity of the Abstract Hoare triples:

$$\models \langle P \rangle_{A^{op}}\ C\ \langle Q \rangle \iff [\![C]\!]^{A}_{ais}(P) \leq_{A^{op}} Q \iff [\![C]\!]^{A^{op}}_{ais}(P) \geq_{A} Q$$

When the dual abstract inductive semantics is obtained from the abstract inductive semantics on $\wp(\mathbb{S})$ (the strongest postcondition), the dual semantics becomes the strongest liberal post-condition introduced in [ZK22] (in the boolean case). The triples are given the name "partial incorrectness," as if $\models \langle Q \rangle_{A^{op}}\ C\ \langle P \rangle$, P is an over-approximation of the states that end up in Q modulo termination. The same concept is studied under the name of "Necessary Preconditions" in [Cou+13], and thanks to Abstract Hoare Logic, we obtained a sound and complete proof system for the logic.

CHAPTER 4

EXTENDING THE PROOF SYSTEM

The proof system for Abstract Hoare logic (definition 2.8) is pretty barebone. The goal of Abstract Hoare logic is to define a general theory for constructing Hoare-like logics, so the idea was to have the least possible assumptions on the assertion language and on the semantics of the base commands. In this chapter, we will see how by adding more requirements on the lattice of the assertions and/or the semantics of the base commands, we can obtain new sound rules for the proof system.

4.1 Merge rules

When developing software verification tools, the ability to perform multiple local reasoning and then merge the results is particularly useful. One example of this is the conjunction rule in concurrent separation logic [BO16].

In Hoare logic, the following two rules are sound:

Definition 4.1 (Merge rules in Hoare logic)**.**

$$\frac{\vdash \{P_1\}\ C\ \{Q_1\} \qquad \vdash \{P_2\}\ C\ \{Q_2\}}{\vdash \{P_1 \vee P_2\}\ C\ \{Q_1 \vee Q_2\}}\ (\vee)$$

$$\frac{\vdash \{P_1\}\ C\ \{Q_1\} \qquad \vdash \{P_2\}\ C\ \{Q_2\}}{\vdash \{P_1 \wedge P_2\}\ C\ \{Q_1 \wedge Q_2\}}\ (\wedge)$$

Even though they aren't needed for the completeness of the proof system, performing two different analyses and then merging the results of them can be actually useful. Already in [Cou+12], it was noted that the abstract version of the merge rules are, in general, unsound in Algebraic Hoare Logic. The same is also true for Abstract Hoare logic. We will show a counterexample for the ($\vee$) rule, but the example can be easily modified for the ($\wedge$) rule.

Definition 4.2 (Merge rules in Abstract Hoare logic)**.**

$$\frac{\vdash \langle P_1\rangle_A\ C\ \langle Q_1\rangle \qquad \vdash \langle P_2\rangle_A\ C\ \langle Q_2\rangle}{\vdash \langle P_1 \vee P_2\rangle_A\ C\ \langle Q_1 \vee Q_2\rangle}\ (\vee)$$

$$\frac{\vdash \langle P_1\rangle_A\ C\ \langle Q_1\rangle \qquad \vdash \langle P_2\rangle_A\ C\ \langle Q_2\rangle}{\vdash \langle P_1 \wedge P_2\rangle_A\ C\ \langle Q_1 \wedge Q_2\rangle}\ (\wedge)$$

Example 4.1 (Counterexample for the $(\vee)$ rule)**.** Let $\langle\cdot\rangle_{Int} \;\cdot\; \langle\cdot\rangle$ be the Abstract Hoare logic instantiation of example 3.1, Interval Logic, and let $C \overset{\text{def}}{=} (x = 4? \fatsemi x := 50) + (x \neq 4? \fatsemi x := x+1)$. Then we can perform the following two derivations:

$$\dfrac{\pi_1 \qquad \pi_2}{\vdash \langle [3,3] \rangle_{Int}\; C\; \langle [4,4] \rangle}\,(+)$$

Where π_1:

$$\dfrac{[3,3] \leq [3,3] \qquad \dfrac{\dfrac{}{\vdash \langle [3,3] \rangle_{Int}\; x = 4?\; \langle \bot \rangle}\,(b) \qquad \dfrac{}{\vdash \langle \bot \rangle_{Int}\; x := 50\; \langle \bot \rangle}\,(b)}{\vdash \langle [3,3] \rangle_{Int}\; x = 4? \fatsemi x := 50\; \langle \bot \rangle}\,(\fatsemi) \qquad \bot \leq [4,4]}{\vdash \langle [3,3] \rangle_{Int}\; x = 4? \fatsemi x := 50\; \langle [4,4] \rangle}\,(\leq)$$

And π_2:

$$\dfrac{\dfrac{}{\vdash \langle [3,3] \rangle_{Int}\; x \neq 4?\; \langle [3,3] \rangle}\,(b) \qquad \dfrac{}{\vdash \langle [3,3] \rangle_{Int}\; x := x+1\; \langle [4,4] \rangle}\,(b)}{\vdash \langle [3,3] \rangle_{Int}\; x \neq 4? \fatsemi x := x+1\; \langle [4,4] \rangle}\,(\fatsemi)$$

And

$$\dfrac{\pi_3 \qquad \pi_4}{\vdash \langle [5,5] \rangle_{Int}\; C\; \langle [6,6] \rangle}\,(+)$$

Where π_3:

$$\dfrac{[5,5] \leq [5,5] \qquad \dfrac{\dfrac{}{\vdash \langle [5,5] \rangle_{Int}\; x = 4?\; \langle \bot \rangle}\,(b) \qquad \dfrac{}{\vdash \langle \bot \rangle_{Int}\; x := 50\; \langle \bot \rangle}\,(b)}{\vdash \langle [5,5] \rangle_{Int}\; x = 4? \fatsemi x := 50\; \langle \bot \rangle}\,(\fatsemi) \qquad \bot \leq [6,6]}{\vdash \langle [5,5] \rangle_{Int}\; x = 4? \fatsemi x := 50\; \langle [6,6] \rangle}\,(\leq)$$

and π_4:

$$\dfrac{\dfrac{}{\vdash \langle [5,5] \rangle_{Int}\; x \neq 4?\; \langle [6,6] \rangle}\,(b) \qquad \dfrac{}{\vdash \langle [5,5] \rangle_{Int}\; x := x+1\; \langle [6,6] \rangle}\,(b)}{\vdash \langle [5,5] \rangle_{Int}\; x \neq 4? \fatsemi x := x+1\; \langle [6,6] \rangle}\,(\fatsemi)$$

Thus we can construct the following proof tree:

$$\dfrac{\vdash \langle [5,5] \rangle_{Int}\; C\; \langle [6,6] \rangle \qquad \vdash \langle [3,3] \rangle_{Int}\; C\; \langle [4,4] \rangle}{\vdash \langle [3,5] \rangle_{Int}\; C\; \langle [4,6] \rangle}$$

But clearly, it's unsound as:

$$\begin{aligned}
[\![C]\!]^{Int}_{ais}([3,5]) &= [\![x = 4? \fatsemi x := 50]\!]^{Int}_{ais}([3,5]) \vee [\![x \neq 4? \fatsemi x := x+1]\!]^{Int}_{ais}([3,5]) \\
&= [\![x := 50]\!]^{Int}_{base}([\![x = 4?]\!]^{Int}_{base}([3,5])) \vee [\![x := x+1]\!]^{Int}_{base}([\![x \neq 4?]\!]^{Int}_{base}([3,5])) \\
&= [50,50] \vee [4,6] \\
&= [4,50]
\end{aligned}$$

And $[4,50] \not\leq [4,6]$.

Naively, we could think that the issue is only "local" as $\gamma([3]) \cup \gamma([5]) = \{3,5\} \neq \{3,4,5\} = \gamma([3] \vee [5])$. Since the least upper bound is adding new states in the precondition, requiring that $\gamma(P_1 \vee P_2) = \gamma(P_1) \cup \gamma(P_2)$ might seem enough, but actually, it is not true. We can construct arbitrary programs that are able to exploit the fact that $\vee$ is generally a convex operation that can add new elements.

Definition 4.3 (Local $\vee$ rule for abstract Hoare logic)**.**

$$\dfrac{\gamma(P_1 \vee P_2) = \gamma(P_1) \cup \gamma(P_2) \qquad \vdash \langle P_1 \rangle_A\; C\; \langle Q_1 \rangle \qquad \vdash \langle P_2 \rangle_A\; C\; \langle Q_2 \rangle}{\vdash \langle P_1 \vee P_2 \rangle_A\; C\; \langle Q_1 \vee Q_2 \rangle}\,(\vee - local)$$

Example 4.2 (Counterexample for the ($\vee - local$) rule)**.** Let $\langle\cdot\rangle_{Int}\ \cdot\ \langle\cdot\rangle$ be the Abstract Hoare logic instantiation of example 3.1, Interval Logic, and let $C \overset{\text{def}}{=} (x = 0? + x = 2?) \fatsemi x = 1?$

Then we can perform the following two derivations:

$$\dfrac{\pi_1 \qquad \dfrac{}{\vdash \langle [0,0]\rangle_{Int}\ x = 1?\ \langle \bot \rangle}\,(b)}{\vdash \langle [0,1]\rangle_{Int}\ C\ \langle \bot \rangle}\,(\fatsemi)$$

Where π_1:

$$\dfrac{\dfrac{}{\vdash \langle [0,1]\rangle_{Int}\ x = 0?\ \langle [0,0]\rangle}\,(b) \qquad \dfrac{\dfrac{}{\vdash \langle [0,1]\rangle_{Int}\ x = 2?\ \langle [\bot]\rangle}\,(b) \qquad \bot \leq [0,0]}{\vdash \langle [0,1]\rangle_{Int}\ x = 2?\ \langle [0,0]\rangle}\,(\leq)}{\vdash \langle [0,1]\rangle_{Int}\ (x = 0?) + (x = 2?)\ \langle [0,0]\rangle}\,(+)$$

And

$$\dfrac{\pi_2 \qquad \dfrac{}{\vdash \langle [2,2]\rangle_{Int}\ x = 1?\ \langle \bot \rangle}\,(b)}{\vdash \langle [2,2]\rangle_{Int}\ C\ \langle \bot \rangle}\,(\fatsemi)$$

Where π_2:

$$\dfrac{\dfrac{\dfrac{}{\vdash \langle [2,2]\rangle_{Int}\ x = 0?\ \langle [\bot]\rangle}\,(b) \qquad \bot \leq [2,2]}{\vdash \langle [2,2]\rangle_{Int}\ x = 0?\ \langle [2,2]\rangle}\,(\leq) \qquad \dfrac{}{\vdash \langle [2,2]\rangle_{Int}\ x = 2?\ \langle [2,2]\rangle}\,(b)}{\vdash \langle [2,2]\rangle_{Int}\ (x = 0?) + (x = 2?)\ \langle [2,2]\rangle}\,(+)$$

Thus we can construct the following proof tree:

$$\dfrac{\vdash \langle [2,2]\rangle_{Int}\ C\ \langle \bot \rangle \qquad \vdash \langle [0,1]\rangle_{Int}\ C\ \langle \bot \rangle}{\vdash \langle 0,2\rangle_{Int}\ C\ \langle \bot \rangle}$$

But clearly is unsound as:

$$\begin{aligned}
[\![C]\!]^{Int}_{ais}([0,2]) &= [\![x = 1?]\!]^{Int}_{base}([\![x = 0?]\!]^{Int}_{base}([0,2]) \vee [\![x = 2]\!]^{Int}_{base}([0,2])) \\
&= [\![x = 1?]\!]^{Int}_{base}([0,0] \vee [2,2]) \\
&= [\![x = 1?]\!]^{Int}_{base}([0,2]) \\
&= [1,1]
\end{aligned}$$

And clearly $[1,1] \not\leq \bot$

This example highlights the actual root cause of the issue: the imprecision introduced by $\vee$, and it has nothing to do with the preconditions. In particular, we can note that with the program $C' \overset{\text{def}}{=} (x = 1? \fatsemi x = 0?) + (x = 2? \fatsemi x = 0?)$, we don't have the same issue. Even if C and C' are equivalent programs in the concrete domain ($\wp(\wp(\mathbb{S}))$), they aren't in the Int domain. Hence, $[\![(C_1 + C_2) \fatsemi C_3]\!]^A_{ais} = [\![(C_1 \fatsemi C_3) + (C_2 \fatsemi C_3)]\!]^A_{ais}$ is not true in general.

In particular, we can easily show that for a subset of the precondition (the precondition that admit a program that can generate them), requiring the distributivity rule to hold is equivalent to requiring the semantics to be additive.

Theorem 4.1 (Equivalence between additivity and distributivity)**.**
$\forall\, i \in [1,3]\ \ \exists\, C_{P_i}\ \ s.t.\ \ \forall\, Q\ \ [\![C_{P_i}]\!]^A_{ais}(Q) = P_i$

$$[\![(C_1 + C_2) \fatsemi C_3]\!]^A_{ais}(P_1) = [\![(C_1 \fatsemi C_3) + (C_2 \fatsemi C_3)]\!]^A_{ais}(P_1)$$
$$\Longleftrightarrow$$
$$[\![C']\!]^A_{ais}(P_2 \vee P_3) = [\![C']\!]^A_{ais}(P_2) \vee [\![C']\!]^A_{ais}(P_3)$$

Proof.

- ($\Longleftarrow$):

$$\begin{aligned}
[\![(C_1 + C_2) ⨾ C_3]\!]^A_{ais}(P_1) &= [\![C_3]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P_1) \vee [\![C_2]\!]^A_{ais}(P_1)) \\
&= [\![C_3]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P_1)) \vee [\![C_3]\!]^A_{ais}([\![C_2]\!]^A_{ais}(P_1)) \\
&= [\![(C_1 ⨾ C_3) + (C_2 ⨾ C_3)]\!]^A_{ais}(P_1)
\end{aligned}$$

- ($\Longrightarrow$):

$$\begin{aligned}
[\![C']\!]^A_{ais}(P_1 \vee P_2) &= [\![C']\!]^A_{ais}([\![C_{P_2}]\!]^A_{ais}(Q) \vee [\![C_{P_3}]\!]^A_{ais}(Q)) \\
&= [\![(C_{P_2} + C_{P_3}) ⨾ C']\!]^A_{ais}(Q) \\
&= [\![(C_{P_2} ⨾ C) + (C_{P_3} ⨾ C')]\!]^A_{ais}(Q) \\
&= [\![C]\!]^A_{ais}([\![C_{P_2}]\!]^A_{ais}(Q)) \vee [\![C]\!]^A_{ais}([\![C_{P_3}]\!]^A_{ais}(Q)) \\
&= [\![C]\!]^A_{ais}(P_2) \vee [\![C]\!]^A_{ais}(P_3)
\end{aligned}$$

□

Hence, we can gain intuition that the $\vee$ rule is not working because, in general, the abstract inductive semantics is not additive, and the root of non additivity is the non additivity of the base commands.

Theorem 4.2 (Additivity of the abstract inductive semantics).
If $[\![b]\!]^A_{base}(P_1 \vee P_2) = [\![b]\!]^A_{base}(P_1) \vee [\![b]\!]^A_{base}(P_2)$

$$[\![C]\!]^A_{ais}(P_1 \vee P_2) = [\![C]\!]^A_{ais}(P_1) \vee [\![C]\!]^A_{ais}(P_2)$$

Proof. By structural induction on C:

- $\mathbb{1}$:

$$\begin{aligned}
[\![\mathbb{1}]\!]^A_{ais}(P_1 \vee P2) &= P_1 \vee P_2 && \text{By definition of } [\![\cdot]\!]^A_{ais} \\
&= [\![\mathbb{1}]\!]^A_{ais}(P_1) \vee [\![\mathbb{1}]\!]^A_{ais}(P_2) && \text{By definition of } [\![\cdot]\!]^A_{ais}
\end{aligned}$$

- b:

$$\begin{aligned}
[\![b]\!]^A_{ais}(P_1 \vee P2) &= [\![b]\!]^A_{base}(P_1 \vee P_2) && \text{By definition of } [\![\cdot]\!]^A_{ais} \\
&= [\![b]\!]^A_{base}(P_1) \vee [\![b]\!]^A_{base}(P_2) \\
&= [\![b]\!]^A_{ais}(P_1) \vee [\![b]\!]^A_{ais}(P_2) && \text{By definition of } [\![\cdot]\!]^A_{ais}
\end{aligned}$$

- $C_1 ⨾ C_2$:

$$\begin{aligned}
[\![C_1 ⨾ C_2]\!]^A_{ais}(P_1 \vee P2) &= [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P_1 \vee P_2)) \\
&\text{By definition of } [\![\cdot]\!]^A_{ais} \\
&= [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P_1) \vee [\![C_1]\!]^A_{ais}(P_2)) \\
&\text{By inductive hypothesis} \\
&= [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P_1)) \vee [\![C_2]\!]^A_{ais}([\![C_1]\!]^A_{ais}(P_2)) \\
&\text{By inductive hypothesis} \\
&= [\![C_1 ⨾ C_2]\!]^A_{ais}(P_1) \vee [\![C_1 ⨾ C_2]\!]^A_{ais}(P_2) \\
&\text{By definition of } [\![\cdot]\!]^A_{ais}
\end{aligned}$$

- $C_1 + C_2$:

$$\begin{aligned}
\llbracket C_1 \fatsemi C_2 \rrbracket^A_{ais}(P_1 \vee P2) &= \llbracket C_1 \rrbracket^A_{ais}(P_1 \vee P_2) \vee \llbracket C_2 \rrbracket^A_{ais}(P_1 \vee P_2) \\
&\text{By definition of } \llbracket \cdot \rrbracket^A_{ais} \\
&= \llbracket C_1 \rrbracket^A_{ais}(P_1) \vee \llbracket C_1 \rrbracket^A_{ais}(P_2) \vee \llbracket C_2 \rrbracket^A_{ais}(P_1) \vee \llbracket C_2 \rrbracket^A_{ais}(P_2) \\
&\text{By inductive hypothesis} \\
&= \llbracket C_1 \rrbracket^A_{ais}(P_1) \vee \llbracket C_2 \rrbracket^A_{ais}(P_1) \vee \llbracket C_1 \rrbracket^A_{ais}(P_2) \vee \llbracket C_2 \rrbracket^A_{ais}(P_2) \\
&= \llbracket C_1 + C_2 \rrbracket^A_{ais}(P_1) \vee \llbracket C_1 + C_2 \rrbracket^A_{ais}(P_2) \\
&\text{By definition of } \llbracket \cdot \rrbracket^A_{ais}
\end{aligned}$$

- C^{fix}:

$$\llbracket C^{\text{fix}} \rrbracket^A_{ais}(P_1 \vee P2) = \text{lfp}(\lambda P' \to P_1 \vee P_2 \vee \llbracket C \rrbracket^A_{ais}(P')) \qquad \text{By definition of } \llbracket \cdot \rrbracket^A_{ais}$$

Let $F_i \overset{\text{def}}{=} \llbracket C^{\text{fix}} \rrbracket_{base}(P_i) = \text{lfp}(\lambda P' \to P_i \vee \llbracket C \rrbracket^A_{ais}(P'))$

We will show that $F_1 \vee F_2$ is the lfp of the first equation.

$$\begin{aligned}
(\lambda P' \to P_1 \vee P_2 \vee \llbracket C \rrbracket^A_{ais}(P'))(F_1 \vee F_2) &= P_1 \vee P_2 \vee \llbracket C \rrbracket^A_{ais}(F_1 \vee F_2) \\
&= P_1 \vee P_2 \vee \llbracket C \rrbracket^A_{ais}(F_1) \vee \llbracket C \rrbracket^A_{ais}(F_2) \\
&\text{By inductive hypothesis} \\
&= P_1 \vee \llbracket C \rrbracket^A_{ais}(F_1) \vee P_2 \vee \llbracket C \rrbracket^A_{ais}(F_2) \\
&= F_1 \vee F_2 \\
&\text{By definition of } F_i \\
&= \llbracket C^{\text{fix}} \rrbracket^A_{ais}(P_1) \vee \llbracket C^{\text{fix}} \rrbracket^A_{ais}(P_2) \\
&\text{By definition of } F_i
\end{aligned}$$

Now we show that it's also the least one, let P be any fixpoint, $P = P_1 \vee P_2 \vee \llbracket C \rrbracket^A_{ais}(P)$. Then by definition of $\vee$ it follows that $P_i \vee \llbracket C \rrbracket^A_{ais}(P) \leq P_1 \vee P_2 \vee \llbracket C \rrbracket^A_{ais}(P)$ but by F_i beeing a least fixpoint $F_i \leq P_i \vee \llbracket C \rrbracket^A_{ais}(P)$ thus $F_1 \vee F_2 \leq P_1 \vee \llbracket C \rrbracket^A_{ais}(P) \vee P_2 \vee \llbracket C \rrbracket^A_{ais}(P) = P_1 \vee P_2 \vee \llbracket C \rrbracket^A_{ais}(P) = P$ hence $F_1 \vee F_2$ it's the leas fixpoint.

□

We can give a sufficient condition for the addivitivity of the abstract inductive semantics:

Theorem 4.3. *Let $\langle C, \sqsubseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \leq \rangle$ be a Galois insertion, if $\llbracket \cdot \rrbracket^C_{ais}$ and γ are additive then the abstract inductive semantics $\llbracket \cdot \rrbracket^A_{ais}$ obtained via the galois insertion is also additive.*

Proof.

$$\begin{aligned}
\llbracket b \rrbracket^A_{base}(P_1 \vee P_2) &= \alpha(\llbracket b \rrbracket^C_{base}(\gamma(P_1 \vee P_2))) \\
&= \alpha(\llbracket b \rrbracket^C_{base}(\gamma(P_1))) \vee \alpha(\llbracket b \rrbracket^C_{base}(\gamma(P_1))) \\
&\text{By the additivity of } \gamma,\ \llbracket \cdot \rrbracket^C_{ais} \text{ and } \alpha \\
&= \llbracket b \rrbracket^A_{base}(P_1) \vee \llbracket b \rrbracket^A_{base}(P_2)
\end{aligned}$$

Then by theorem 4.2 $\llbracket \cdot \rrbracket^A_{ais}$ is additive. □

Last we can show that the abstract inductive semantics beeing additive is enough to make the $(\vee)$ rule sound.

Theorem 4.4 (Soundness of the ($\vee$) rule)**.** *Let $[\![\cdot]\!]^A_{ais}$ be additive then:*

$$[\![C]\!]^A_{ais}(P_1) \leq Q_1 \text{ and } [\![C]\!]^A_{ais}(P_2) \leq Q_2 \implies [\![C]\!]^A_{ais}(P_1 \vee P_2) \leq Q_1 \vee Q_2$$

Proof.

$$\begin{aligned} [\![C]\!]^A_{ais}(P_1 \vee P_2) &= [\![C]\!]^A_{ais}(P_1) \vee [\![C]\!]^A_{ais}(P_2) && \text{By additivity of } [\![\cdot]\!]^A_{ais} \\ &\leq Q_1 \vee Q_2 \end{aligned}$$

□

The two theorems 4.4 and 4.3 correspond to the result for Algebraic Hoare logic in which the rule ($\overline{\vee}$) is sound if γ is additive.

The same reasoning can be applied for the soundness of the rule ($\wedge$) but by requiring the semantics to be co-additive.

Abstract domain that are both additive and co-additive are extremley rare (expecially for the additivity as co-additivity is far more common) but they exists, for exampe the complete sign domain of example 1.2 is one of them, making both the merge rules sound.

CHAPTER 5

BACKWARD ABSTRACT HOARE LOGIC

When defining the semantics for $\mathbb{L}$, we implicitly assumed that the abstract inductive semantics is forward by giving the definition $[\![C_1 \mathbin{;} C_2]\!]^A_{ais} \stackrel{\text{def}}{=} [\![C_2]\!]^A_{ais} \circ [\![C_1]\!]^A_{ais}$. However, except for the rule ($;$), we never explicitly used this fact. We can reuse the theory of Abstract Hoare logic to construct a slight variation called Backward Abstract Hoare logic, which describes Hoare logics where the semantics is defined backward.

5.1 Framework

5.1.1 Backward abstract inductive semantics

To define the backward version of Abstract Hoare logic, we first need a backward version of the abstract inductive semantics:

Definition 5.1 (Backward abstract inductive semantics)**.** Given a complete lattice A and a family of monotone functions $[\![b]\!]^A_{base} : A \to A$ for all $b \in Base$, the abstract inductive semantics is defined as follows:

$$\begin{aligned}
[\![\cdot]\!]^A_{bais} &: \mathbb{L} \to A \to A \\
[\![\mathbb{1}]\!]^A_{bais} &\stackrel{\text{def}}{=} id \\
[\![b]\!]^A_{bais} &\stackrel{\text{def}}{=} [\![b]\!]^A_{base} \\
[\![C_1 \mathbin{;} C_2]\!]^A_{bais} &\stackrel{\text{def}}{=} [\![C_1]\!]^A_{bais} \circ [\![C_2]\!]^A_{bais} \\
[\![C_1 + C_2]\!]^A_{bais} &\stackrel{\text{def}}{=} \lambda P.[\![C_1]\!]^A_{bais} P \vee_A [\![C_2]\!]^A_{bais} P \\
[\![C^{\text{fix}}]\!]^A_{bais} &\stackrel{\text{def}}{=} \lambda P.\text{lfp}(\lambda P'.P \vee_A [\![C]\!]^A_{bais} P')
\end{aligned}$$

The only difference from the abstract inductive semantics provided in definition 2.3 is the case for $C_1 \mathbin{;} C_2$.

We can prove that the backward abstract inductive semantics is still monotone.

Theorem 5.1 (Monotonicity)**.** *For all $C \in \mathbb{L}$, $[\![C]\!]^A_{bais}$ is monotone.*

Proof. We modify the inductive case of the proof of theorem 2.3 by providing only the case for $[\![C_1 \mathbin{;} C_2]\!]^A_{bais}$ as all the other cases are identical.

- $C_1 \mathbin{;} C_2$:

  By inductive hypothesis, $[\![C_2]\!]^A_{bais}$ is monotone, hence $[\![C_2]\!]^A_{bais}(P) \leq_A [\![C_2]\!]^A_{bais}(Q)$.

$$\begin{aligned}
[\![C_1 \fatsemi C_2]\!]^A_{bais}(P) &= [\![C_1]\!]^A_{bais}([\![C_2]\!]^A_{bais}(P)) && \text{By definition of } [\![C_1 \fatsemi C_2]\!]^A_{bais} \\
&\leq_A [\![C_1]\!]^A_{bais}([\![C_2]\!]^A_{bais}(Q)) && \text{By inductive hypothesis on } [\![C_1]\!]^A_{bais}
\end{aligned}$$

□

Lemma 5.1 ($[\![\cdot]\!]^A_{bais}$ well-defined)**.** *For all $C \in \mathbb{L}$, $[\![C]\!]^A_{bais}$ is well-defined.*

Proof. From theorems 5.1 and 1.2, all the least fixpoints in the definition of $[\![C^{\text{fix}}]\!]^A_{bais}$ exist. For all the other commands, the semantics is trivially well-defined. □

5.1.2 Backward Abstract Hoare Logic

Now we can provide the definition for the backward abstract Hoare triples, which is the same as for abstract Hoare triples, only with the backward abstract inductive semantics instead of the usual abstract inductive semantics.

Definition 5.2 (Backward Abstract Hoare triple)**.** Given an abstract inductive semantics $[\![\cdot]\!]^A_{bais}$ on the complete lattice A, the abstract Hoare triple written $\langle P\rangle^B_A\ C\ \langle Q\rangle$ is valid if and only if $[\![C]\!]^A_{bais}(P) \leq_A Q$.

$$\models \langle P\rangle^B_A\ C\ \langle Q\rangle \iff [\![C]\!]^A_{bais}(P) \leq_A Q$$

Clearly, the proof system only needs to be modified to accommodate the new semantics for program composition, and all the other rules remain the same.

Definition 5.3 (Backward Abstract Hoare rules)**.**
We only provide the rule for program composition; all the other rules are identical to those provided in definition 2.8.

$$\frac{\vdash \langle P\rangle^B_A\ C_2\ \langle Q\rangle \qquad \vdash \langle Q\rangle^B_A\ C_1\ \langle R\rangle}{\vdash \langle P\rangle^B_A\ C_1 \fatsemi C_2\ \langle R\rangle}\ (\fatsemi)$$

If executing C_2 from state P leads to state Q, and executing C_1 from state Q leads to state R, then executing C_2 followed by C_1 from state P leads to state R.

We can prove again that the proof system is still sound and complete.

Theorem 5.2 (Soundness)**.**

$$\vdash \langle P\rangle^B_A\ C\ \langle Q\rangle \implies \models \langle P\rangle^B_A\ C\ \langle Q\rangle$$

Proof. We modify the inductive case of the proof of theorem 2.8 by providing only the case for rule ($\fatsemi$) as all the other cases are identical.

- ($\fatsemi$): Then the last step in the derivation was:

  $$\frac{\vdash \langle P\rangle^B_A\ C_2\ \langle Q\rangle \qquad \vdash \langle Q\rangle^B_A\ C_1\ \langle R\rangle}{\vdash \langle P\rangle^B_A\ C_1 \fatsemi C_2\ \langle R\rangle}\ (\fatsemi)$$

  By inductive hypothesis: $[\![C_2]\!]^A_{bais}(P) \leq_A Q$ and $[\![C_1]\!]^A_{bais}(Q) \leq_A R$.

  The triple is valid since:

  $$\begin{aligned}
  [\![C_1 \fatsemi C_2]\!]^A_{bais}(P) &= [\![C_1]\!]^A_{bais}([\![C_2]\!]^A_{bais}(P)) && \text{By definition of } [\![\cdot]\!]^A_{bais} \\
  &\leq_A [\![C_1]\!]^A_{bais}(Q) && \text{By monotonicity of } [\![\cdot]\!]^A_{bais} \\
  &\leq_A R
  \end{aligned}$$

□

Theorem 5.3 (Relative $[\![\cdot]\!]^A_{bais}$-completeness)**.**

$$\vdash \langle P\rangle^B_A\; C\; \langle [\![C]\!]^A_{bais}(P)\rangle$$

Proof. We modify the inductive case of the proof of theorem 2.9 by providing only the case for $C_1 \fatsemi C_2$ as all the other cases are identical.

- $C_1 \fatsemi C_2$: By definition $[\![C_1 \fatsemi C_2]\!]^A_{bais}(P) = [\![C_1]\!]^A_{bais}([\![C_2]\!]^A_{bais}(P))$

$$\dfrac{\begin{array}{cc}\text{(Inductive hypothesis)} & \text{(Inductive hypothesis)}\\ \vdash \langle P\rangle^B_A\; C_2\; \langle [\![C_2]\!]^A_{bais}(P)\rangle & \vdash \langle [\![C_2]\!]^A_{bais}(P)\rangle^B_A\; C_1\; \langle [\![C_1]\!]^A_{bais}([\![C_2]\!]^A_{bais}(P))\rangle\end{array}}{\vdash \langle P\rangle^B_A\; C_1 \fatsemi C_2\; \langle [\![C_1 \fatsemi C_2]\!]^A_{bais}(P)\rangle}\;(\fatsemi)$$

□

Theorem 5.4 (Relative completeness)**.**

$$\models \langle P\rangle^B_A\; C\; \langle C\rangle \implies \vdash \langle P\rangle^B_A\; C\; \langle Q\rangle$$

Proof. By definition of $\models \langle P\rangle^B_A\; C\; \langle Q\rangle \iff Q \geq_A [\![C]\!]^A_{bais}(P)$

$$\dfrac{P \leq_A P \qquad \begin{array}{c}\text{(By Theorem 5.3)}\\ \vdash \langle P\rangle^B_A\; C\; \langle [\![C]\!]^A_{bais}(P)\rangle\end{array} \qquad Q \geq_A [\![C]\!]^A_{bais}(P)}{\vdash \langle P\rangle^B_A\; C\; \langle Q\rangle}\;(\leq)$$

□

5.2 Instantiations

5.2.1 Partial Incorrectness, Again

An abstract inductive semantics induces automatically a backward abstract inductive semantics where the semantics of the base commands is inverted:

Definition 5.4 (Reverse Abstract Inductive Semantics)**.** Given an abstract inductive semantics defined on some complete lattice A with base command semantics $[\![\cdot]\!]^A_{base}$, we can define the reverse backward abstract inductive semantics as the backward inductive semantics instantiated on the complete lattice A and base command semantics $([\![\cdot]\!]^A_{base})^{-1}$.

Hence, the reverse abstract inductive semantics can be expressed as:

$$\begin{aligned}
[\![\mathbb{1}]\!]^A_{bais} &= id\\
[\![b]\!]^A_{bais} &= ([\![b]\!]^A_{base})^{-1}\\
[\![C_1 \fatsemi C_2]\!]^A_{bais} &= [\![C_1]\!]^A_{bais} \circ [\![C_2]\!]^A_{bais}\\
[\![C_1 + C_2]\!]^A_{bais} &= \lambda P.[\![C_1]\!]^A_{bais}P \vee_A [\![C_2]\!]^A_{bais}P\\
[\![C^{\text{fix}}]\!]^A_{bais} &= \lambda P.\text{lfp}(\lambda P'.P \vee_A [\![C]\!]^A_{bais}P')
\end{aligned}$$

Following the intuition that the abstract inductive semantics is some abstract version of the strongest postcondition, what interpretation can we give for the reverse abstract inductive semantics? The construction corresponds to the abstract version of the weakest precondition. In fact, when the dual reverse inductive semantics is obtained from the abstract inductive semantics on $\wp(\mathbb{S})$ (the strongest postcondition), the reverse semantics becomes the weakest precondition.

Hence, from the validity of the triples:

$$\models \langle Q\rangle^B_{\wp(\mathbb{S})}\; C\; \langle P\rangle \iff [\![C]\!]^{\wp(\mathbb{S})}_{bais}(Q) \subseteq P \iff wp(C,Q) \subseteq P$$

This program logic is introduced in [Asc+24] under the name of NC, and the program logic is actually equivalent to the one in section 3.4.

5.2.2 Hoare Logic, Again

Following what we did in section 3.4, we can first obtain the reverse semantics but then also obtain the dual of the reverse semantics. It can be expressed as:

$$\begin{aligned}
[\![\mathbb{1}]\!]^{A^{op}}_{bais} &= id \\
[\![b]\!]^{A^{op}}_{bais} &= ([\![b]\!]^{A}_{base})^{-1} \\
[\![C_1 \mathbin{;} C_2]\!]^{A^{op}}_{bais} &= [\![C_1]\!]^{A}_{bais} \circ [\![C_2]\!]^{A}_{bais} \\
[\![C_1 + C_2]\!]^{A^{op}}_{bais} &= \lambda P.[\![C_1]\!]^{A^{op}}_{bais} P \wedge_A [\![C_2]\!]^{A^{op}}_{bais} P \\
[\![C^{\text{fix}}]\!]^{A^{op}}_{bais} &= \lambda P.\mathrm{gfp}_A(\lambda P'.P \wedge_A [\![C]\!]^{A^{op}}_{bais} P')
\end{aligned}$$

Following the intuition that the abstract inductive semantics is some abstract version of the strongest postcondition, what interpretation can we give for the dual reverse abstract inductive semantics? The construction corresponds to the reverse inductive semantics is obtained from the abstract inductive semantics abstract version of the weakest liberal precondition. In fact, when the dual on $\wp(\mathbb{S})$ (the strongest postcondition), the reverse semantics becomes the weakest liberal precondition.

Hence, from the validity of the triples:

$$\models \langle Q \rangle^{B}_{\wp(\mathbb{S})^{op}} \; C \; \langle P \rangle \iff [\![C]\!]^{\wp(\mathbb{S})}_{bais}(Q) \supseteq P \iff wlp(C, Q) \supseteq P$$

But also $wlp(C, Q) \supseteq P \iff [\![C]\!](P) \subseteq Q$, hence it is equivalent to Hoare logic.

CHAPTER 6

CONCLUSIONS

We extended traditional Hoare logic by transforming it into a more abstract and versatile framework. By incorporating principles from abstract interpretation, we developed a method for reasoning about a broader range of properties.

Through our discussion, we demonstrated that multiple program logics known in literature are actually special cases of Abstract Hoare Logic. And notably, while constructing a program logic for hyperproperties within this framework, we provided a novel compositional definition of the strongest hyper postcondition.

Furthermore, we showed how the core proof principles of Hoare logic can be applied to proving an underapproximation of program properties, highlighting that the main differences of Incorrectness Logic core proof system, specifically suggesting that the infinitary loop rule required for relative completeness:

$$\frac{[p(n)]\ C\ [p(n+1)]}{[p(0)]\ C^{\star}\ [\exists n.p(n)]}$$

Is not due to the logic trying to prove an underapproximation but rather because it is a total correctness logic, inherently carrying a proof of termination:

We also discussed the requirements to introduce frame-like rules in Abstract Hoare Logic and how to obtain a backward variant of this framework.

6.1 Future work

The following are only preliminary results that we weren't able to explore because of time constraints.

6.1.1 Total correctness/Incorrectness logics

We have seen how all the partial correctness/incorrectness triples are instances of (backward) Abstract Hoare Logic and use a very similar proof system. To complete the picture presented in [ZK22], we are missing the total correctness and incorrectness logics, ??? and '¿¿?. Since they are related in the same way as the partial correctness/incorrectness logics are related, the same abstraction used to transform Hoare Logic into Abstract Hoare Logic could be used to transform Incorrectness Logic [MOH21] into Abstract Incorrectness Logic. By abstracting the proof system, we can obtain a sound and relatively complete proof system for Abstract Incorrectness Logic. Then, by reusing the same technique that we did for Abstract Hoare Logic by inverting the semantics and the lattice we can obtain all the four program logics that are missing and complete the whole picture.

6.1.2 Hyper domains

We used hyper domains to encode the strongest hyper postcondition and obtained a Hoare-like logic for hyperproperties. We have shown how we can use the abstract inductive semantics to model the strongest liberal postcondition, weakest precondition, and weakest liberal precondition. We could perform the same trick with the abstract inductive semantics instantiated with a hyper domain of $\wp(\mathbb{S})$ and see if it leads to some interesting logics or if they are all equivalent (since hyperproperties can negate themselves).

Another pain point of the hyper Hoare logic obtained via Abstract Hoare Logic is that the assertion language is relatively low-level, making it cumbersome to use for proving actual hyperproperties. The proof system in [DM23] is actually quite similar to the one obtained with the hyper domains, but the Exist rule is missing. Since it is necessary to prove the completeness of Hyper Hoare Logic, it must be embedded somewhere in the rules of Abstract Hoare Logic.

6.1.3 Unifying Forward and Backward Reasoning

The only difference between Abstract Hoare Logic and Backward Abstract Hoare Logic lies in the abstract inductive semantics, where the semantics of program composition ($\fatsemi$) is inverted. A potential solution would be to make the semantics parametric on the composition $[\![C_1 \fatsemi C_2]\!]^A_{ais} \stackrel{\text{def}}{=} [\![C_1]\!]^A_{ais} \star [\![C_2]\!]^A_{ais}$ and let $\star = \circ^{-1}$ for the forward semantics and $\star = \circ$ for the backward semantics. However, this approach is somewhat inelegant when defining the command composition rule for the proof system.

6.2 Related work

The idea of systematically constructing program logics is not new. Kleene Algebra with Tests (KAT) [Koz97] was one of the first works of this kind. In section 4.1, we discussed how, in general, we cannot distribute the non-deterministic choice ($[\![(C_1 + C_2) \fatsemi C_3]\!]^A_{ais} \neq [\![(C_1 \fatsemi C_3) + (C_2 \fatsemi C_3)]\!]^A_{ais}$), thus violating one of the axioms of Kleene algebras. Another similar alternative was proposed in [MMO06], using traced monoidal categories to encode properties of the program. For example, the monoidal structure is used to model non-deterministic choice but imposes the same distributivity requirements as Kleene Algebras (this is caused by $\oplus$ being a bifunctor). However, disregarding expressivity, the main difference lies in the philosophy behind the approach. Abstract Hoare Logic is a more semantics-centered approach instead of being an "equational" theory like KAT. This semantics-centered approach was also vital in providing the idea that abstract inductive semantics could be used not only to encode the strongest postcondition but also the strongest liberal postcondition, weakest precondition, and weakest liberal precondition, thereby unifying all partial Hoare-like logics.

A more similar approach to that of Abstract Hoare Logic is Outcome Logic [ZDS23]. Like Abstract Hoare Logic, the semantics of the language in Outcome Logic is parametric on the domain of execution, but the assertion language is fixed if we ignore the basic assertions on program states. Outcome Logic originally aimed to unify correctness and incorrectness reasoning with the powerset instantiation, not to be a minimal theory for sound and complete Hoare-like logics. In fact, the relative completeness proof is missing. As discussed in [DM23], Outcome Logic with the powerset instantiation is actually a proof system for 2-hyperproperties (hyperproperties regarding at most two executions). Thus, Outcome triples can be proved in the instantiation of Abstract Hoare Logic provided in section 3.3.1, even though it would be interesting to find a direct encoding of Outcome Logic in terms of Abstract Hoare Logic.

BIBLIOGRAPHY

[Asc+24] Flavio Ascari, Roberto Bruni, Roberta Gori, and Francesco Logozzo. *Sufficient Incorrectness Logic: SIL and Separation SIL*. 2024. arXiv: 2310.18156 (cit. on p. 35).

[Ass+17] Mounir Assaf, David A. Naumann, Julien Signoles, Éric Totel, and Frédéric Tronel. "Hypercollecting semantics and its application to static analysis of information flow". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL '17. Paris, France: Association for Computing Machinery, 2017, pp. 874–887. ISBN: 9781450346603. DOI: 10.1145/3009837.3009889. URL: https://doi.org/10.1145/3009837.3009889 (cit. on p. 22).

[BO16] Stephen Brookes and Peter W. O'Hearn. "Concurrent separation logic". In: *ACM SIGLOG News* 3.3 (Aug. 2016), pp. 47–65. DOI: 10.1145/2984450.2984457. URL: https://doi.org/10.1145/2984450.2984457 (cit. on p. 27).

[CC77] Patrick Cousot and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints". In: *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '77. Los Angeles, California: Association for Computing Machinery, 1977, pp. 238–252. ISBN: 9781450373500. DOI: 10.1145/512950.512973. URL: https://doi.org/10.1145/512950.512973 (cit. on pp. 3, 5, 11).

[Coo78] Stephen A. Cook. "Soundness and Completeness of an Axiom System for Program Verification". In: *SIAM Journal on Computing* 7.1 (1978), pp. 70–90. DOI: 10.1137/0207005. eprint: https://doi.org/10.1137/0207005. URL: https://doi.org/10.1137/0207005 (cit. on p. 13).

[Cou+12] Patrick Cousot, Radhia Cousot, Francesco Logozzo, and Michael Barnett. "An Abstract Interpretation Framework for Refactoring with Application to Extract Methods with Contracts". In: *ACM SIGPLAN Notices* 47 (Oct. 2012). DOI: 10.1145/2384616.2384633 (cit. on pp. 13, 19, 27).

[Cou+13] Patrick Cousot, Radhia Cousot, Manuel Fähndrich, and Francesco Logozzo. "Automatic Inference of Necessary Preconditions". In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 128–148. ISBN: 978-3-642-35873-9 (cit. on p. 26).

[CS08] Michael R. Clarkson and Fred B. Schneider. "Hyperproperties". In: *2008 21st IEEE Computer Security Foundations Symposium*. 2008, pp. 51–65. DOI: 10.1109/CSF.2008.7 (cit. on p. 21).

[Dij74] Edsger W. Dijkstra. "Guarded commands, non-determinacy and a calculus for the derivation of programs". circulated privately. June 1974. URL: http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD418.PDF (cit. on p. 7).

[DM23] Thibault Dardinier and Peter Müller. *Hyper Hoare Logic: (Dis-)Proving Program Hyperproperties (extended version)*. Jan. 2023. DOI: 10.48550/arXiv.2301.10037 (cit. on pp. 25, 38).

[FL79] Michael J. Fischer and Richard E. Ladner. "Propositional dynamic logic of regular programs". In: *Journal of Computer and System Sciences* 18.2 (1979), pp. 194–211. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(79)90046-1. URL: https://www.sciencedirect.com/science/article/pii/0022000079900461 (cit. on p. 9).

[Flo93] Robert W. Floyd. "Assigning Meanings to Programs". In: *Program Verification: Fundamental Issues in Computer Science*. Ed. by Timothy R. Colburn, James H. Fetzer, and Terry L. Rankin. Dordrecht: Springer Netherlands, 1993, pp. 65–81. ISBN: 978-94-011-1793-7. DOI: 10.1007/978-94-011-1793-7_4. URL: https://doi.org/10.1007/978-94-011-1793-7_4 (cit. on p. 12).

[Hoa69] C. A. R. Hoare. "An axiomatic basis for computer programming". In: *Commun. ACM* 12.10 (Oct. 1969), pp. 576–580. ISSN: 0001-0782. DOI: 10.1145/363235.363259. URL: https://doi.org/10.1145/363235.363259 (cit. on pp. 12, 13).

[Koz97] Dexter Kozen. "Kleene algebra with tests". In: *ACM Trans. Program. Lang. Syst.* 19.3 (May 1997), pp. 427–443. ISSN: 0164-0925. DOI: 10.1145/256167.256195. URL: https://doi.org/10.1145/256167.256195 (cit. on p. 38).

[MMO06] Ursula Martin, Erik Mathiesen, and Paulo Oliva. "Hoare Logic in the Abstract". In: vol. 4207. Jan. 2006, pp. 501–515. ISBN: 978-3-540-45458-8. DOI: 10.1007/11874683_33 (cit. on p. 38).

[MOH21] Bernhard Möller, Peter O'Hearn, and Tony Hoare. "On Algebra of Program Correctness and Incorrectness". In: *Relational and Algebraic Methods in Computer Science*. Ed. by Uli Fahrenberg, Mai Gehrke, Luigi Santocanale, and Michael Winter. Cham: Springer International Publishing, 2021, pp. 325–343. ISBN: 978-3-030-88701-8 (cit. on pp. 12, 37).

[MP18] Isabella Mastroeni and Michele Pasqua. "Verifying Bounded Subset-Closed Hyperproperties". In: *Static Analysis*. Ed. by Andreas Podelski. Cham: Springer International Publishing, 2018, pp. 263–283. ISBN: 978-3-319-99725-4 (cit. on p. 22).

[Sco70] Dana Scott. *OUTLINE OF A MATHEMATICAL THEORY OF COMPUTATION*. Tech. rep. PRG02. OUCL, Nov. 1970, p. 30 (cit. on p. 3).

[ZDS23] Noam Zilberstein, Derek Dreyer, and Alexandra Silva. "Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning". In: *Proc. ACM Program. Lang.* 7.OOPSLA1 (Apr. 2023). DOI: 10.1145/3586045. URL: https://doi.org/10.1145/3586045 (cit. on p. 38).

[ZK22] Linpeng Zhang and Benjamin Lucien Kaminski. "Quantitative strongest post: a calculus for reasoning about the flow of quantitative information". In: *Proc. ACM Program. Lang.* 6.OOPSLA1 (Apr. 2022). DOI: 10.1145/3527331. URL: https://doi.org/10.1145/3527331 (cit. on pp. 26, 37).