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 and complete. We also demonstrate the practical applications of this framework by obtaining a program logic for hyperproperties, highlighting its versatility and benefits. From the design of Abstract Hoare logic, we define Reverse Abstract Hoare logic, which is used to develop systems for backward correctness reasoning on programs.

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 and crucial 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. Software engineers 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 tests for every possible input of the program. To offer a guarantee of the absence of undesired behaviors, we need sound logical models rooted in logic. The field of formal methods in computer science aims at developing the logical tools necessary to prove properties of software systems.

Hoare logic, first introduced by Hoare in the late 60s [Hoa69], 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 executing a program.

Hoare logic, beyond being one of the first program logics, 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 ideally that the proof system is powerful enough to prove all 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.

In this work, 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, through examples, 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 [CC77]. The semantics of the language is defined as an inductive abstract interpreter, and the validity of the Abstract Hoare triples depends on it. Since we do not use the strongest postcondition directly, we are able to reason about properties that are not expressible in the powerset of the program states, such as hyperproperties.

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

- In Chapter 3, we show some notable instantiations of Abstract Hoare Logic: we demonstrate that it is possible to obtain program logics 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 we introduce the concept of the strongest hyper postcondition); Finally, 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 summary of the most important contributions of the thesis. We discuss possible extensions to the framework of Abstract Hoare Logic and, to conclude, 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 [Grä11; Bir40]. 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: $\forall x, y \in X,\ (x, y) \in \leq$ and $(y, x) \in \leq \implies x = y$
- Transitivity: $\forall x, y, z \in X,\ (x, y) \in \leq$ 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 $\{(x, y) \mid (x, y) \in \leq$ 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 $\text{lfp}(f)$ and is defined as $lfp(f) = a^* \in \text{fix}(f)$ and $\forall a \in \text{fix}(f)\ a^* \leq a$.
- We denote the *greatest fixpoint* as $\text{gfp}(f)$ and is defined as $gfp(f) = a^* \in \text{fix}(f)$ and $\forall a \in \text{fix}(f)\ a^* \geq a$.

1.1.2 Lattices

Definition 1.8 (Meet-semilattice). A meet-semilattice is a partially ordered set $(L, \leq)$ such that for every pair of elements $a, b \in L$, there exists an element $c \in L$ satisfying the following conditions:

1. $c \leq a$ and $c \leq b$
2. $\forall d \in L$, if $d \leq a$ and $d \leq b$, then $d \leq c$

The element c is called the *meet* of *greatest lower bound* of a and b, and is denoted by $a \wedge b$.

Definition 1.9 (Join-semilattice). A join-semilattice is a partially ordered set $(L, \leq)$ such that for every pair of elements $a, b \in L$, there exists an element $c \in L$ satisfying the following conditions:

1. $c \geq a$ and $c \geq b$
2. $\forall d \in L$, if $d \geq a$ and $d \geq b$, then $d \geq c$

The element c is called the *join* or *least upper bound* of a and b, and is denoted by $a \vee b$.

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

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

Definition 1.11 (Complete lattice). A partially ordered set $(L, \leq)$ is called a *complete lattice* if for every subset $S \subseteq L$, there exist elements $\sup S$ and $\inf S$ in L such that:

1. $\sup S$ (the supremum or least upper bound of S) is an element of L satisfying:
   - For all $s \in S$, $s \leq \sup S$.
   - For any $u \in L$, if $s \leq u$ for all $s \in S$, then $\sup S \leq u$.
2. $\inf S$ (the infimum or greatest lower bound of S) is an element of L satisfying:
   - For all $s \in S$, $\inf S \leq s$.
   - For any $l \in L$, if $l \leq s$ for all $s \in S$, then $l \leq \inf S$.

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

Observation 1.2. A complete lattice cannot be empty.

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

Observation 1.3 (Point-wise fixpoint). The least-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.

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

Theorem 1.1 (Knaster-Tarski theorem). *Let $(L, \leq)$ be a complete lattice and let $f : L \to L$ be a monotone function. Then $(\mathit{fix}(f), \leq)$ is also a complete lattice.*

We have two direct consequences: both the greatest and the least fixpoint of f exists as they are respectively top and bottom of $\text{fix}(f)$.

Theorem 1.2 (Post-fixpoint inequality). *Let f be a monotone function on a complete lattice then*

$$f(x) \leq x \implies \text{lfp}(f) \leq x$$

Proof. By theorem 1.1 $\text{lfp}(f) = \bigwedge\{y \mid y \geq f(y)\}$ thus $\text{lfp}(f) \leq x$ since $x \in \{y \mid y \geq f(y)\}$. □

Theorem 1.3 (lfp monotonicity). *Let L be a complete lattice, if $P \leq Q$ and f is monotone then*

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

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 1.2 pick $f = \lambda X.P \vee f(X)$ and $x = \text{lfp}(\lambda X.Q \vee f(X))$ it follows that $\text{lfp}(\lambda X.P \vee f(X)) \leq \text{lfp}(\lambda X.Q \vee f(X))$. □

1.2 Abstract Interpretation

Abstract interpretation [CC77; Cou21] is the de-facto standard approach for designing static program analysis. The specification of a program can be expressed as a pair of initial and final sets of states, $\mathit{Init}, \mathit{Final} \in \wp(\mathbb{S})$, and the task of verifying a program C boils down to checking if $[\![C]\!](\mathit{Init}) \subseteq \mathit{Final}$.

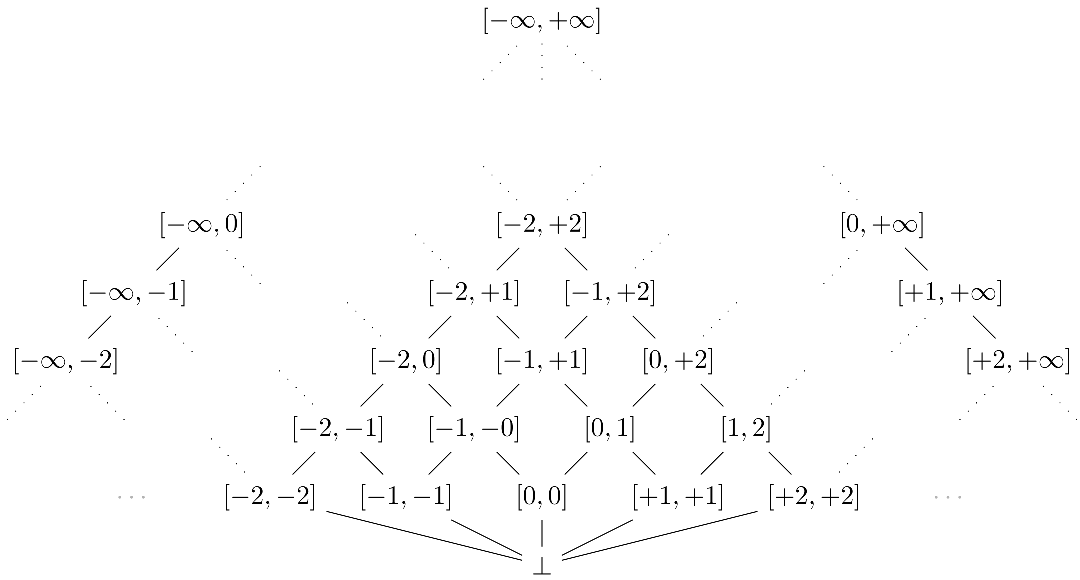
Clearly, this task cannot be performed in general. The solution proposed by the framework of abstract interpretation is to construct an approximation of $[\![\cdot]\!]$, usually denoted by $[\![\cdot]\!]^{\#}$, that is computable.

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 $\mathit{Int} = \{[a, b] \mid a, b \in \mathbb{Z} \cup \{+\infty, -\infty\}, a \leq b\} \cup \{\bot\}$ 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:

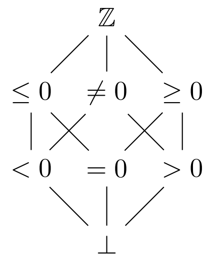


Then, there is a Galois insertion from *Int* to $\wp(\mathbb{Z})$ defined as:

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

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

Example 1.2 (Complete sign domain). Let $Sign = \{\perp, < 0, > 0, = 0, \leq 0, \neq 0, \geq 0, \mathbb{Z}\}$ be ordered by following the hasse diagram below.



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

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

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

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).

Example 1.3. We can use the sign domain to construct a sound approximation of the multiplication operation:

$\times^{\#}$	$\perp$	< 0	> 0	$= 0$	≤ 0	$\neq 0$	≥ 0	$\mathbb{Z}$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
< 0	$\perp$	> 0	< 0	$= 0$	< 0	$\neq 0$	≤ 0	$\mathbb{Z}$
> 0	$\perp$	< 0	> 0	$= 0$	≤ 0	$\neq 0$	≥ 0	$\mathbb{Z}$
$= 0$	$\perp$	$= 0$	$= 0$	$= 0$	$= 0$	$= 0$	$= 0$	$= 0$
≤ 0	$\perp$	< 0	≤ 0	$= 0$	≤ 0	$\neq 0$	≤ 0	$\mathbb{Z}$
$\neq 0$	$\perp$	$\neq 0$	$\neq 0$	$= 0$	$\neq 0$	$\neq 0$	$\neq 0$	$\mathbb{Z}$
≥ 0	$\perp$	≤ 0	≥ 0	$= 0$	≤ 0	$\neq 0$	≥ 0	$\mathbb{Z}$
$\mathbb{Z}$	$\perp$	$\mathbb{Z}$	$\mathbb{Z}$	$= 0$	$\mathbb{Z}$	$\mathbb{Z}$	$\mathbb{Z}$	$\mathbb{Z}$

Table 1.1: Multiplication table for *Sign* domain

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 language $\mathbb{L}$, a minimal imperative programming language that is parametric on a set of basic commands to permit the definition of arbitrary program features, such as pointers, objects, etc. We will define the semantics of the language, provide the standard definition of 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] with the goal of being as general as possible by being parametric on a set of *basic 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 $BCmd$ of basic commands, the set on valid $\mathbb{L}$ programs is defined by the following inductive definition:

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

Example 2.1. Usually the set of basic commands contains a command to perform tests $e?$ discarding executions that do not satisfy the predicate e, and $x := v$ to assign the value v 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 $[\![\cdot]\!]_{base} : BCmd \to \mathbb{S} \hookrightarrow \mathbb{S}$ we can define the denotational semantics of programs in $\mathbb{L}$. The *collecting semantics* is defined as a function $[\![\cdot]\!] : \mathbb{L} \to \wp(\mathbb{S}) \to \wp(\mathbb{S})$ that associates a program C and a set of initial states to the set of states reached after executing the program C from the initial states, this is also know as the predicate transformer semantics [Dij74].

Definition 2.2 (Denotational semantics). Given a set $\mathbb{S}$ of states and a family of partial functions $[\![\cdot]\!]_{base} : BCmd \to \mathbb{S} \hookrightarrow \mathbb{S}$ the denotational semantics is defined as follows:

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

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

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

Where $\sigma \models e$ means that the state σ satisfies the predicate e and $\uparrow$ is denoting that the function is diverging.

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

Where *eval* is some evaluate function for the expressions on the left-hand side of assignments and then is substitute in place of x in the state σ.

Theorem 2.1 (Monotonicity). *$\forall\, C \in \mathbb{L}$ $[\![C]\!]$ is well-defined and 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 $[\![C_1]\!](P) \subseteq [\![C_2]\!](Q)$

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

- $C_1 + C_2$:

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

- C^{fix}:

$$\begin{aligned}
[\![C^{\text{fix}}]\!](P) &= \text{lfp}(\lambda P'.P \cup [\![C]\!](P')) && \text{[By definition of } [\![C^{\text{fix}}]\!]\text{]} \\
&\subseteq \text{lfp}(\lambda P'.Q \cup [\![C]\!](P')) && \text{[By theorem 1.3]} \\
&= [\![C^{\text{fix}}]\!](Q) && \text{[By definition of } [\![C^{\text{fix}}]\!]\text{]}
\end{aligned}$$

Clearly all the lfp are well-defined since by inductive hypothesis $[\![C]\!]$ is monotone and $\wp(\mathbb{S})$ is a complete from 1.1 the least-fixpoint exists.

□

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

$$\textbf{\textit{if}}\ b\ \textbf{\textit{then}}\ C_1\ \textbf{\textit{else}}\ C_2 \stackrel{\text{def}}{=} (b? \fatsemi C_1) + (\neg b? \fatsemi C_2)$$

$$\textbf{\textit{while}}\ b\ \textbf{\textit{do}}\ C \stackrel{\text{def}}{=} (b? \fatsemi C)^{\text{fix}} \fatsemi \neg b?$$

Observation 2.2. Regular languages of Kleene algebras [Koz97] 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.

Example 2.3. Let $C \stackrel{\text{def}}{=} (x \le 10? \fatsemi x := x + 1)^{\text{fix}} + (x := 55)$ and $P = \{x = 1\}$ then we can compute $[\![C]\!](P)$ as:

$$\begin{aligned}
[\![C]\!](P) &= [\![(x \le 10? \fatsemi x := x + 1)^{\text{fix}}]\!](P) \cup [\![x := 55]\!](P) \\
&= \text{lfp}(\lambda P'.P \cup [\![x \le 10? \fatsemi x := x + 1]\!](P')) \cup \{x = 55\} \\
&= \{x \in \{1, ..., 10\}\} \cup \{x = 55\} \\
&= \{x \in \{1, ..., 10, 55\}\}
\end{aligned}$$

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 as we provide suitable function for the basic commands. The rationale behind is the same as in the denotational semantics but instead of representing collections of states with $\wp(\mathbb{S})$ now they are represented in an arbitrary complete lattice.

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

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

When designing 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 goal of the abstract interpreter is to be executed and, in general, if the lattice on which the interpretation executed run has infinite ascending chains, it's computation can diverge. In our case, the termination requirement isn't necessary since we are not interested in computing the abstract inductive semantics but using it as a reference 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 2.2 (Monotonicity). *$\forall\, C \in \mathbb{L}$ $[\![C]\!]^A_{ais}$ is well-defined and 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:

- $\mathbb{1}$:

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

- b:

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

- $C_1 \fatsemi C_2$:

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

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

- $C_1 + C_2$:

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

- C^{fix}:

$$\begin{aligned}
[\![C^{\text{fix}}]\!]^A_{ais}(P) &= \text{lfp}(\lambda P'.P \vee_A [\![C]\!]^A_{ais}(P')) && \text{[By definition of } [\![C^{\text{fix}}]\!]^A_{ais}\text{]}\\
&\leq_A \text{lfp}(\lambda P'.Q \vee_A [\![C]\!]^A_{ais}(P')) && \text{[By theorem 1.3]}\\
&= [\![C^{\text{fix}}]\!]^A_{ais}(Q) && \text{[By definition of } [\![C^{\text{fix}}]\!]^A_{ais}\text{]}
\end{aligned}$$

Clearly all the lfp are well-defined since by inductive hypothesis $[\![C]\!]$ is monotone and A is a complete from 1.1 the least-fixpoint exists. □

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

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

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

This can be easily checked by comparing the two definitions.

From this observation, we can see that theorem 2.1 is just an instance of theorem 2.2 since $\wp(\mathbb{S})$ is a complete lattice and the semantics of the basic commands is monotone by construction.

2.2.1 Connection with Abstract Interpretation

As stated above, the definition of abstract inductive semantics is closely related to the one of abstract semantics [CC77]. In particular, the definition of abstract inductive semantics, when the semantics of the basic commands is sound, is equivalent to an abstract semantics.

Theorem 2.3 (Abstract interpretation instance). *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 $[\![\cdot]\!]$.*

Proof. We prove $\alpha([\![C]\!](P)) \leq [\![C]\!]^A_{ais}(\alpha(P))$ by structural induction on C:

- $\mathbb{1}$:

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

- b:

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

- $C_1 \fatsemi C_2$:

$$
\begin{aligned}
\alpha(\llbracket C_1 \fatsemi C_2 \rrbracket(P)) &= \alpha(\llbracket C_2 \rrbracket(\llbracket C_1 \rrbracket(P))) && \text{[By definition of } \llbracket C_1 \fatsemi C_2 \rrbracket\text{]} \\
&\leq \llbracket C_2 \rrbracket^A_{ais}(\alpha(\llbracket C_1 \rrbracket(P))) && \text{[By inductive hypothesis on } C_2\text{]} \\
&\leq \llbracket C_2 \rrbracket^A_{ais}(\llbracket C_1 \rrbracket^A_{ais}(\alpha(P))) && \text{[By inductive hypothesis on } C_1 \\
& && \text{and } \llbracket C_2 \rrbracket^A_{ais} \text{ monotone]} \\
&= \llbracket C_1 \fatsemi C_2 \rrbracket^A_{ais}(\alpha(P)) && \text{[By definition of } \llbracket C_1 \fatsemi C_2 \rrbracket^A_{ais}\text{]}
\end{aligned}
$$

- $C_1 + C_2$:

$$
\begin{aligned}
\alpha(\llbracket C_1 + C_2 \rrbracket(P)) &= \alpha(\llbracket C_1 \rrbracket(P) \cup \llbracket C_2 \rrbracket(P)) && \text{[By definition of } \llbracket C_1 + C_2 \rrbracket\text{]} \\
&\leq \alpha(\llbracket C_1 \rrbracket(P)) \vee \alpha(\llbracket C_2 \rrbracket^A_{ais}(P)) && \\
&\leq \llbracket C_1 \rrbracket^A_{ais}(\alpha(P)) \vee \llbracket C_2 \rrbracket^A_{ais}(\alpha(P)) && \text{[By inductive hypothesis on } C_1 \\
& && \text{and } C_2\text{]} \\
&= \llbracket C_1 + C_2 \rrbracket^A_{ais}(\alpha(P)) && \text{[By definition of } \llbracket C_1 + C_2 \rrbracket^A_{ais}\text{]}
\end{aligned}
$$

- C^{fix}:

$$
\begin{aligned}
\alpha(\llbracket C^{\text{fix}} \rrbracket(P)) &= \alpha(\text{lfp}(\lambda P'.P \cup \llbracket C \rrbracket(P'))) && \text{[By definition of } \llbracket C^{\text{fix}} \rrbracket\text{]} \\
&= \alpha(\bigcup_{n \in \mathbb{N}} \llbracket C \rrbracket^n(P)) && \\
&\leq \bigvee_{n \in \mathbb{N}} \alpha(\llbracket C \rrbracket^n(P))) && \\
&\leq \bigvee_{n \in \mathbb{N}} (\llbracket C \rrbracket^A_{ais})^n(\alpha(P)) && \text{[By inductive hypothesis on } C\text{]} \\
&\leq \text{lfp}(\lambda P'.\alpha(P) \vee \llbracket C \rrbracket^A_{ais}(P')) && \\
&= \llbracket C^{\text{fix}} \rrbracket^A_{ais}(\alpha(P)) && \text{[By definition of } \llbracket C^{\text{fix}} \rrbracket^A_{ais}\text{]}
\end{aligned}
$$

□

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

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 $\llbracket C \rrbracket^C_{ais}$ be some abstract inductive semantics defined on C. Then, the abstract inductive semantics defined on A with $\llbracket b \rrbracket^A_{base} \overset{\text{def}}{=} \alpha \circ \llbracket c \rrbracket^C_{base} \circ \gamma$ 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 $\bigvee_{n \in \mathbb{N}}(\llbracket C \rrbracket^A_{ais})^n(P) \neq \text{lfp}(\lambda P'.P \vee_A \llbracket C \rrbracket^A_{ais}(P'))$.

Example 2.4. Let $C \overset{\text{def}}{=} (x > 1? \fatsemi ((even(x)? \fatsemi X := x+3) + (\neg even(x)? \fatsemi x := x-2))^{\text{fix}}$ when performing the computation on the interval domain, if we compute C using the infinitary join:

$$\begin{aligned}\llbracket C \rrbracket^A_{ais}([5,5]) &= \bigvee_{n\in\mathbb{N}} (\llbracket x > 1? \fatsemi ((even(x)? \fatsemi x := x+3) + (\neg even(x)? \fatsemi x := x-2)) \rrbracket^A_{ais})^n([5,5]) \\ &= [5,5] \vee [3,3] \vee [1,1] \vee \bot \vee \bot ... \\ &= [1,5]\end{aligned}$$

The difference is caused by the fact that when we are computing the infinite join, all the joins happen after executing the semantics of the loop body.

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, It's core concept is that of partial correctness assertions. A Hoare triple is a formula $\{P\}\ C\ \{Q\}$ where P and Q are assertions on the initial and final states of a 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 designed as a proof system, where the syntax $\vdash \{P\}\ C\ \{Q\}$ indicates that the triple $\{P\}\ C\ \{Q\}$ is proved by applying the 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 defined for our language $\mathbb{L}$ following the work in [MOH21].

Definition 2.5 (Hoare triple). Fixed the semantics of the basic commands, an Hoare triple denoted by $\{P\}\ C\ \{Q\}$, is valid if and only if $\llbracket C \rrbracket(P) \subseteq Q$.

$$\models \{P\}\ C\ \{Q\} \iff \llbracket C \rrbracket(P) \subseteq Q$$

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 2.5 (Hoare triples). We have that $\{x \in [1,2]\}\ x := x+1\ \{x \in [2,3]\}$, 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 $\{P\}\ C\ \{\top\}$ is always valid since $\top$ contains all the possible states.

An example of an invalid triple is $\{x \in [1,2]\}\ x := x+1\ \{x \in [1,2]\}$ since the state $x = 2$ satisfies the precondition and executing 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, $\{x \in [0,10]\}\ (x \leq 20? \fatsemi x := x-1)^{\text{fix}} \fatsemi x \geq 20?\ \{Q\}$ 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 called a partial correctness logic, where partial means that it can prove the adherence of a program to some specification only when it is terminating. The termination of the program must be proved by resorting to some alternative method.

Definition 2.6 (Hoare logic).
The rules of Hoare logic are defined as follows:

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

$$\frac{}{\vdash \{P\}\ b\ \{\llbracket b \rrbracket_{base}(P)\}}\ (base)$$

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

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

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

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

The proof system described in Definition 2.6 is logically sound, meaning that all it's provable triples are valid with respect to definition 2.5.

Theorem 2.4 (Soundness).

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

As observed by Cook [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 the 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 2.5 (Relative completeness).

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

2.3.2 Abstracting Hoare logic

The idea of designing 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 the triples in [Cou+12] dependends on the standard definition of Hoare triples, and the proof system 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 without resorting to embedd Hoare logic in it's proof system as [Cou+12].

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 to the definition 2.5 but here the abstract inductive semantics is used to provide the strongest postcondition of programs.

In Abstract Hoare logic some of the examples shown in example 2.5 still hold, in particular we have that:

Example 2.6.

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

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 $\top \geq [\![C]\!]^A_{ais}(P)$ □

2.3.3 Proof system

As per Hoare logic we will provide a sound and relatively complete (in the sense of [Coo78]) proof system to derive abstract Hoare triples in a compositional fashion.

Definition 2.8 (Abstract Hoare rules).

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

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

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

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

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

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

The rules can be summarized as:

- The identity command does not change the state, so if P holds before, it will hold after the execution.
- For a basic command b, if P holds before the execution, then $[\![b]\!]^A_{base}(P)$ holds after the execution.
- 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.
- 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.
- 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.
- 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 is nonother than definition 2.6, where the assertion are replaced by elements of the complete lattice A.

Note that we denote Abstract Hoare Triples as defined in defintion 2.7 with the notation $\langle P \rangle_A \ C \ \langle Q \rangle$ while we denote the triples obtained with the inference rules of definition 2.8 by $\vdash \langle P \rangle_A \ C \ \langle Q \rangle$.

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

Theorem 2.6 (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:

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

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

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

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, $\mathrm{lfp}(\lambda P' \to P \vee_A [\![C]\!]^A_{ais}(P')) \leq_A P$ 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:

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

□

The proof system turns out to be relatively complete aswell, 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 first prove a slightly weaker result, where we will show that we can prove the strongest post-condition of every program.

Theorem 2.7 (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:

$$\dfrac{P \leq_A P \qquad \overset{\text{(Inductive hypothesis)}}{\vdash \langle P\rangle_A\ C_2\ \langle [\![C_2]\!]^A_{ais}(P)\rangle} \qquad [\![C_2]\!]^A_{ais}(P) \leq_A [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P)}{\vdash \langle P\rangle_A\ C_2\ \langle [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P)\rangle}\ (\leq)$$

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

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

□

We can now show the relative completeness, by applying the rule $(\leq)$ to achieve the desired post-condition.

Theorem 2.8 (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 $\models \langle P\rangle_A\ C\ \langle Q\rangle \iff Q \geq_A [\![C]\!]^A_{ais}(P)$

$$\dfrac{P \leq_A P \qquad \overset{\text{(By Theorem 2.7)}}{\vdash \langle P\rangle_A\ C\ \langle [\![C]\!]^A_{ais}(P)\rangle} \qquad Q \geq_A [\![C]\!]^A_{ais}(P)}{\vdash \langle P\rangle_A\ C\ \langle Q\rangle}\ (\leq)$$

□

CHAPTER 3

INSTANTIATING ABSTRACT HOARE LOGIC

In this chapter, we will demonstrate how to instantiate abstract Hoare logic to create new proof systems. We will also illustrate that the framework of abstract Hoare logic is sufficiently general to reason about properties that cannot be expressed 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 basic command semantics, is equivalent to the denotational semantics given in Definition 2.2. As we can see from the definitions 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 in computer-representable abstract domains.

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

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

From this definition, we see that 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]).

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

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

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

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

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

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 based on 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 $C \overset{\text{def}}{=} ((x := 1) + (x := 3))\,;\,(x = 2?\,;\,x := 5) + (x \neq 2?\,;\,x := x - 1)$

Then the following derivation is valid:

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

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:

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

π_4:

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

π_5:

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

π_6 :

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

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

3.2.2.1 Applications

This framework, like Algebraic Hoare Logic, can be used to specify how a static analyzer for a given abstract domain should work. Since $[\![\cdot]\!]^A_{ais}$ is the best abstract analyzer on the 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 basic 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 2.3, $[\![\cdot]\!]^A_{ais}$ is a sound overapproximation of $[\![\cdot]\!]$.

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

Proof.

$$\begin{aligned}
\vdash \langle P\rangle_A\ C\ \langle Q\rangle &\implies [\![C]\!]^A_{ais}(P) \leq Q && \text{[From Theorem 2.6]}\\
&\implies [\![C]\!](\gamma(P)) \subseteq \gamma(Q) && \text{[From Theorem 2.3]}\\
&\implies \vdash \{\gamma(P)\}\ C\ \{\gamma(Q)\} && \text{[From Theorem 2.5]}\\
&\implies \vdash \overline{\{}P\overline{\}}\ C\ \overline{\{}Q\overline{\}} && \text{[From rule }(\overline{S})\text{]}
\end{aligned}$$

$\square$

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

Example 3.3 (Counterexample). 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 $[\![C]\!]\top = \underline{\{}0, 2\underline{\}}$. Via Theorem 2.5, 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 discrepancy arises because, via the (S) rule in Algebraic Hoare logic, we are able to prove the best correct approximation of any program C. However, 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 used in Hoare logic, belong to the set $\wp(\mathbb{S})$. In contrast, hyperproperties belong to $\wp(\wp(\mathbb{S}))$, as they encode relations between different executions. A common example is the property of a program being deterministic. For instance, if our programs have a single integer variable x, proving determinism involves an infinite number of Hoare triples of the form: for each $n \in \mathbb{N}$, there exists $m \in \mathbb{N}$ such that $\models \{\{x = n\}\}\ C\ \{\{x = m\}\}$. However, determinism can be succinctly encoded in a single hyper triple: $\models \{\{P \in \wp(\wp(\mathbb{S})) \mid |P| = 1\}\}\ C\ \{\{Q \in \wp(\wp(\mathbb{S})) \mid |Q| = 1\}\}$.

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

$$\{\llbracket C \rrbracket(P) \mid P \in \mathcal{X}\}$$

As we are interested in the strongest postcondition of every starting state in $\mathbb{X}$.

3.3.2 Inductive Definition of the Strongest Hyper Post Condition

To establish a sound and relatively complete logic for hyperproperties within our framework, it is crucial to develop an abstract inductive semantics that precisely computes the strongest hyper postcondition. This challenge has been explored in prior works [Ass+17; MP18], primarily in the context of abstract interpretation. However, existing approaches often provide an overapproximation of the strongest hyper postcondition, which, while suitable for abstract interpretation, falls short of maintaining relative completeness in our context.

In [Ass+17], for instance, the hyper semantics of ***if*** b ***then*** C_1 ***else*** C_2 is given as $\{\llbracket b? \fatsemi C_1 \rrbracket T \cup \llbracket \neg b? \fatsemi C_2 \rrbracket \mid T \in \mathbb{T}\}$, thereby lacking inductiveness. This non-inductive definition allows for the analysis of ***if*** $1 = 1$ ***then*** C, for any program C, rendering the use of hyper semantics practically unnecessary for analyzing any program.

The fundamental issue lies in $\wp(\wp(\mathbb{S}))$ where, under the standard powerset ordering, the least upper bound fails to distinguish between different executions.

Example 3.4. Let $\mathcal{X} \stackrel{\text{def}}{=} \{\{1, 2, 3\}, \{5\}\}$. Clearly,

$$\llbracket (x := x + 1) + (x := x + 2) \rrbracket_{ais}^{\wp(\wp(\mathbb{S}))}(\mathcal{X}) = \{\{2, 3, 4\}, \{6\}, \{3, 4, 5\}, \{7\}\},$$

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

When applying the rule for non-deterministic choice, $\llbracket C_1 + C_2 \rrbracket_{ais}^{\wp(\wp(\mathbb{S}))}(\mathcal{P}) = \llbracket C_1 \rrbracket_{ais}^{\wp(\wp(\mathbb{S}))}(\mathcal{P}) \cup \llbracket C_2 \rrbracket_{ais}^{\wp(\wp(\mathbb{S}))}(\mathcal{P})$, the union of outer sets is performed instead of the inner ones that contain actual executions. Attempts to modify the ordering on $\wp(\wp(\mathbb{S}))$ prove futile as each set lacks information about the generating execution, leading to unavoidable precision loss in union construction.

To date, no literature addresses an abstract inductive semantics that exactly computes the strongest hyper postcondition; existing works settle for sound overapproximations. While adequate for abstract interpreters, such approximations prove insufficient for verifying certain hyperproperties within Abstract Hoare logic, especially where precision in abstract inductive semantics is compromised.

3.3.3 Hyper Domains

To address the limitations of $\wp(\wp(\mathbb{S}))$, we introduce a more sophisticated family of domains designed to keep track of the execution of interest across the different executions. We utilize a set K to index individual executions and define the join operation in a manner that preserves their distinctiveness.

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

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

The complete lattice structure of $H(B)_K$ is obtained by lifting the pointwise lattice of B+undef, where B + undef forms a complete lattice on B with *undef* as the new bottom element, meaning that $\uparrow < \bot_B$.

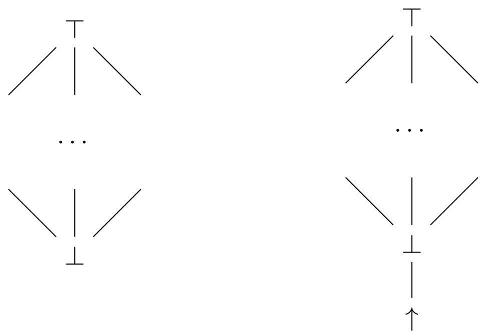


Figure 3.1: On the left the Hasse diagram of B, on the right the Hasse diagram of $B + undef$

Here, the role of K is solely to index different executions, with no specific requirements on its elements beyond ensuring enough distinct indices to account for all executions.

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

$$[\![b]\!]^{H(B)_K}_{base}(\mathbb{X}) \stackrel{\text{def}}{=} \lambda r. [\![b]\!]^{B}_{base}(\mathbb{X}(r))$$

The concept of hyper instantiation lifts the abstract inductive semantics from domain B to its "hyper" version, applying the semantics of basic commands from B to each execution.

Next, we establish that the abstract inductive semantics instantiated on a hyper-domain preserves non-interference, meaning that running the hyper inductive semantics yields the same results as running the original semantics on each execution.

Theorem 3.2 (Non-interference between executions).

$$[\![C]\!]^{H(B)_K}_{ais}(\mathbb{X}) = \lambda r. [\![C]\!]^{B}_{ais}(\mathbb{X}(r))$$

Proof. By structural induction on C:

- $\mathbb{1}$:

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

- b:

$$[\![b]\!]^{H(B)_K}_{ais}(\mathbb{X}) = \lambda r. [\![b]\!]^{B}_{ais}(\mathbb{X}(r))$$

- $C_1 \fatsemi C_2$:

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

- $C_1 + C_2$:

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

- C^{fix}:

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

□

3.3.4 Inductive Definition for Hyper Postconditions

Having introduced hyper domains to overcome the limitations of $\wp(\wp(\mathbb{S}))$, we now use a different domain in our abstract inductive semantics. To bridge this gap, we establish a method for converting standard hyperproperties to their hyper domain counterparts and vice versa. This involves defining a pair of functions, referred to as the conversion pair, which facilitates this operation. Since there are infinitely many functions converting a standard hyperproperty into a version using hyper domains (due to the infinite representations of the same property), we employ a single representative (an injective function) to encapsulate them all, ensuring that our results remain independent of the chosen indexing function.

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

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

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

By instantiating the hyper domain as $H(\wp(\mathbb{S}))_{\mathbb{R}}$, we demonstrate that our abstract inductive semantics computes the strongest hyper postcondition.

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

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

Proof.

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

□

3.3.5 Hyper Hoare Triples

The instantiation of hyper domains provides a sound and complete Hoare-like logic for hyperproperties, particularly when using α on pre- and postconditions.

Example 3.5 (Determinism in Abstract Hoare Logic). As discussed in Example 3.4, we express the determinism (up to termination) of a command by proving that the hyperproperty $\{P \mid |P| = 1\}$ serves as both precondition and postcondition for the command.

Assuming our language $\mathbb{L}$ uses single-variable assignments, allowing us to represent states with integers.

The property $\mathbb{P}$ we use as precondition is:

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

We prove that $\mathbb{1}$ (skip command) is deterministic:

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

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

Similarly, we demonstrate determinism for the increment function:

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

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

Clearly, $\alpha(\mathbb{Q}) = \{..., \{0\}, \{1\}, \{2\}, ...\}$, proving determinism.

We can establish that a nondeterministic choice between two identical programs also remains deterministic:

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

However, different programs cannot be treated the same:

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

Where π:

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

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

Observation 3.1. Different elements within the hyper domain correspond to the same hyperproperty, reflecting that the nondeterministic choice does not always "preserve" hyperproperties. This approach parallels other logics that handle hyperproperties by introducing a new disjunction operator capable of distinguishing between different executions.

A related Hoare-like logic, Hyper Hoare Logic ([DM23]), offers a sound and relatively complete framework for hyperproperties. While it was specifically developed for this purpose, it is equivalent to the logic derived from the abstract Hoare logic framework. Notably, it departs from using the classical disjunction connective (equivalent to the least upper bound in $\wp(\wp(\mathbb{S}))$), opting instead for an exotic disjunction operator ($\otimes$) that can differentiate between executions, similarly as the least upper bound works in the hyper-domain.

3.4 Partial Incorrectness

Any instantiation of the abstract inductive semantics provides us with another instantiation for free, as the semantics is parameterized by a complete lattice A, and the dual of a complete lattice A is also complete. Therefore, we can derive the dual abstract inductive semantics on the complete lattice A^{op}.

Definition 3.7 (Dual Abstract Inductive Semantics). Given an abstract inductive semantics defined on a complete lattice A with basic commands $[\![\cdot]\!]^{A}_{base}$, the dual abstract inductive semantics is defined on the complete lattice A^{op} with basic command semantics $[\![\cdot]\!]^{A^{op}}_{base} = [\![\cdot]\!]^{A}_{base}$.

Since the dual abstract inductive semantics is itself an abstract inductive semantics, it naturally induces an Abstract Hoare Logic. In the dual lattice, where the partial order is inverted, operations such as joins and meets are reversed, leading to an inversion of lfp and gfp. Hence, the dual abstract inductive semantics, viewed from the dual lattice, can be formulated as follows:

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

Interpreting the dual abstract inductive semantics, we understand that in the dual lattice A^{op}, non-deterministic choices are handled by taking the meet of two branches, reflecting certainty rather than possibility. Instead of considering all reachable states (union of states reached by each branch), it considers the intersection of states guaranteed to be reached by both branches. This inversion similarly applies to the fix command.

Given the inverted order in the dual lattice, the validity of Abstract Hoare triples is reversed:

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

When deriving the dual abstract inductive semantics from the abstract inductive semantics on $\wp(\mathbb{S})$ (strongest postcondition), the dual semantics corresponds to the strongest liberal postcondition as introduced in [ZK22] (in the boolean case). These triples are termed "partial incorrectness," implying that if $\models \langle Q \rangle_{A^{op}}\ C\ \langle P \rangle$, then P over-approximates the states reaching Q, accounting for termination. This concept aligns with "Necessary Preconditions" explored in [Cou+13], and Abstract Hoare Logic provides a sound and complete proof system for this logic.

CHAPTER 4

EXTENDING THE PROOF SYSTEM

The proof system for Abstract Hoare logic (definition 2.8) is rather minimalistic. The objective of Abstract Hoare logic is to establish a comprehensive framework for developing Hoare-like logics, aiming to impose as few assumptions as possible on both the assertion language and the semantics of base commands. Throughout this chapter, we explore the potential to derive additional sound rules for the proof system by introducing more constraints either on the lattice of assertions or on the semantics of base commands.

4.1 Merge rules

When developing software verification tools, the capability to perform multiple local reasonings and subsequently merge their results proves particularly beneficial. An example of this is evident in the conjunction rule within concurrent separation logic [BO16].

In Hoare logic, the soundness of the following two rules is established:

Definition 4.1 (Merge rules in Hoare logic).

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

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

Although not essential for the completeness of the proof system, the practice of conducting two distinct analyses and subsequently merging their results can be highly advantageous. As noted in [Cou+12], the abstract versions of merge rules are generally unsound in Algebraic Hoare Logic, a fact that holds true for Abstract Hoare logic as well. We will present a counterexample for the ($\lor$) rule, which can be readily adapted to illustrate issues with the ($\land$) rule.

Definition 4.2 (Merge rules in Abstract Hoare logic).

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

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

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, Abstract Interval Logic, and let $C \stackrel{\text{def}}{=} (x = 4? \fatsemi x := 50) + (x \neq 4? \fatsemi x := x + 1)$. 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, one might think the issue is merely "local," as $\gamma([3]) \cup \gamma([5]) = \{3,5\} \neq \{3,4,5\} = \gamma([3] \vee [5])$. It might seem sufficient to require that $\gamma(P_1 \vee P_2) = \gamma(P_1) \cup \gamma(P_2)$, since the least upper bound adds new states in the precondition. However, this assumption is incorrect. Arbitrary programs can be constructed that exploit the fact that $\vee$ is generally a convex operation capable of introducing 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 $C \stackrel{\text{def}}{=} (x = 0? + x = 2?) \fatsemi x = 1?$

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 shows the actual root cause of the issue: the imprecision introduced by $\vee$, which is unrelated to the preconditions. Specifically, consider the program $C' \stackrel{\text{def}}{=} (x = 1? \fatsemi x = 0?) + (x = 2? \fatsemi x = 0?)$, where we do not encounter the same issue. Despite C and C' being equivalent programs in the concrete domain ($\wp(\wp(\mathbb{S}))$), they differ in the Int domain. Therefore, the equality $[\![(C_1 + C_2) \fatsemi C_3]\!]^A_{ais} = [\![(C_1 \fatsemi C_3) + (C_2 \fatsemi C_3)]\!]^A_{ais}$ does not generally hold.

In particular, we can easily demonstrate that for a subset of the preconditions (those admitting a program capable of having them as a postcondition), requiring the distributivity rule to hold is equivalent to demanding 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.

- ($\Longleftarrow$):

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

- ($\Longrightarrow$):

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

□

Therefore, we can intuit that the $\vee$ rule fails because, in general, the of the basic commands. abstract inductive semantics lacks additivity, stemming from the non-additivity

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:

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

- $C_1 \fatsemi C_2$:

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

- $C_1 + C_2$:

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

- C^{fix}:

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

Let $F_i \stackrel{\text{def}}{=} [\![C^{\text{fix}}]\!]_{base}(P_i) = \text{lfp}(\lambda P' \to P_i \vee [\![C]\!]^A_{ais}(P'))$

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

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

Now we show that it's also the least one, let P be any fixpoint, $P = P_1 \vee P_2 \vee [\![C]\!]^A_{ais}(P)$.

Then by definition of $\vee$ it follows that $P_i \vee [\![C]\!]^A_{ais}(P) \leq P_1 \vee P_2 \vee [\![C]\!]^A_{ais}(P)$ but by F_i beeing a least fixpoint $F_i \leq P_i \vee [\![C]\!]^A_{ais}(P)$ thus $F_1 \vee F_2 \leq P_1 \vee [\![C]\!]^A_{ais}(P) \vee P_2 \vee [\![C]\!]^A_{ais}(P) = P_1 \vee P_2 \vee [\![C]\!]^A_{ais}(P) = P$ hence $F_1 \vee F_2$ it's the leas fixpoint.

$\square$

We can give a sufficient condition for the addivitivity of the abstract inductive semantics obtained trough a galois insertion:

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

Proof.

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

Then by theorem 4.2 $[\![\cdot]\!]^A_{ais}$ is additive. $\square$

Lastly, we can demonstrate that the additivity of the abstract inductive semantics is sufficient to ensure the soundness of the $(\vee)$ rule.

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.

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

□

Theorems 4.4 and 4.3 correspond to the result in Algebraic Hoare logic where the $(\overline{\vee})$ rule is sound under the condition that γ is additive.

Similar reasoning applies to ensure the soundness of the $(\wedge)$ rule by requiring the semantics to be co-additive.

Abstract domains that are both additive and co-additive are extremely rare, especially for additivity alone, although they do exist. For instance, the complete sign domain illustrated in Example 1.2 is one such domain, ensuring the soundness of both merge rules.

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 $[\![C_1 \fatsemi C_2]\!]^A_{ais} \stackrel{\text{def}}{=} [\![C_2]\!]^A_{ais} \circ [\![C_1]\!]^A_{ais}$. However, except for the rule ($\fatsemi$), 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 $[\![\cdot]\!]^A_{base} : BCmd \to A \to A$, the abstract inductive semantics is defined as follows:

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

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

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 well-defined and monotone.*

Proof. We modify the inductive case of the proof of theorem 2.2 by providing only the case for $[\![C_1 \fatsemi C_2]\!]^A_{bais}$ as all the other cases are identical.

- $C_1 \fatsemi C_2$:

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

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

□

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.

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

The rule can be read as: 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 2.6 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 2.7 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 $[\![C_1 \fatsemi C_2]\!]^{A}_{bais}(P) = [\![C_1]\!]^{A}_{bais}([\![C_2]\!]^{A}_{bais}(P))$

$$\frac{\overset{\text{(Inductive hypothesis)}}{\vdash \langle P\rangle^{B}_{A}\ C_2\ \langle [\![C_2]\!]^{A}_{bais}(P)\rangle} \qquad \overset{\text{(Inductive hypothesis)}}{\vdash \langle [\![C_2]\!]^{A}_{bais}(P)\rangle^{B}_{A}\ C_1\ \langle [\![C_1]\!]^{A}_{bais}([\![C_2]\!]^{A}_{bais}(P))\rangle}}{\vdash \langle P\rangle^{B}_{A}\ C_1 \fatsemi C_2\ \langle [\![C_1 \fatsemi C_2]\!]^{A}_{bais}(P)\rangle}\ (\fatsemi)$$

□

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 $\models \langle P\rangle^{B}_{A}\ C\ \langle Q\rangle \iff Q \geq_A [\![C]\!]^{A}_{bais}(P)$

$$\frac{P \leq_A P \qquad \overset{\text{(By Theorem 5.3)}}{\vdash \langle P\rangle^{B}_{A}\ C\ \langle [\![C]\!]^{A}_{bais}(P)\rangle} \qquad Q \geq_A [\![C]\!]^{A}_{bais}(P)}{\vdash \langle P\rangle^{B}_{A}\ C\ \langle Q\rangle}\ (\leq)$$

□

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 basic commands is inverted:

Definition 5.4 (Reverse Abstract Inductive Semantics). Given an abstract inductive semantics defined on some complete lattice A with basic 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 basic 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:

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

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 $[\![C_1 \fatsemi C_2]\!]^A_{ais} \overset{\text{def}}{=} [\![C_1]\!]^A_{ais} \star [\![C_2]\!]^A_{ais}$ 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. 37).

[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. 24).

[Bir40] G. Birkhoff. *Lattice Theory*. American Mathematical Society colloquium publications v. 25,pt. 2. American Mathematical Society, 1940. ISBN: 9780821810255 (cit. on p. 3).

[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. 29).

[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. 1, 3, 5, 13).

[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 pp. 16, 17).

[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. 16, 21, 29).

[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. 28).

[Cou21] P. Cousot. *Principles of Abstract Interpretation*. MIT Press, 2021. ISBN: 9780262044905 (cit. on p. 5).

[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. 23).

[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 pp. 9, 10).

[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. 27, 40).

[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. 11).

[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. 15).

[Grä11] G. Grätzer. *Lattice Theory: Foundation*. SpringerLink : Bücher. Springer Basel, 2011. ISBN: 9783034800181 (cit. on p. 3).

[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. 1, 15).

[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 pp. 11, 40).

[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. 40).

[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. 15, 39).

[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. 24).

[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. 40).

[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. 28, 39).