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-like logic framework that generalizes the traditional Hoare program 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 systematically deriving a program logic for hyperproperties, thus highlighting versatility and benefits of our general framework. From the design of Abstract Hoare logic, we then define Reverse Abstract Hoare logic, which is used to develop a proof system 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

1 Background **3**
- 1.1 Order theory . . . 3
  - 1.1.1 Partial Orders . . . 3
  - 1.1.2 Lattices . . . 4
  - 1.1.3 Fixpoints . . . 5
- 1.2 Abstract Interpretation . . . 6
  - 1.2.1 Abstract Domains . . . 6

2 The abstract Hoare logic framework **9**
- 2.1 The $\mathbb{L}$ programming language . . . 9
  - 2.1.1 Syntax . . . 9
  - 2.1.2 Semantics . . . 9
- 2.2 Abstract inductive semantics . . . 12
  - 2.2.1 Connection with Abstract Interpretation . . . 13
- 2.3 Abstract Hoare Logic . . . 15
  - 2.3.1 Hoare logic . . . 15
  - 2.3.2 Abstracting Hoare logic . . . 16

3 Instantiating Abstract Hoare Logic **21**
- 3.1 Hoare logic . . . 21
  - 3.1.1 Algebraic Hoare Logic . . . 21
  - 3.1.2 Abstract Interval Logic . . . 22
  - 3.1.3 Abstract vs Algebraic Hoare Logic . . . 23
- 3.2 Hoare logic for hyperproperties . . . 23
  - 3.2.1 Introduction to Hyperproperties . . . 23
  - 3.2.2 Inductive Definition of the Strongest Hyper Postcondition . . . 24
  - 3.2.3 Hyper Domains . . . 24
  - 3.2.4 Inductive Definition for Hyper Postconditions . . . 26
  - 3.2.5 Hyper Hoare Triples . . . 27
- 3.3 Partial Incorrectness . . . 28

4 Extending the proof system **29**
- 4.1 Merge rules . . . 29

5 Backward Abstract Hoare Logic **35**
- 5.1 Framework . . . 35
  - 5.1.1 Backward abstract inductive semantics . . . 35
  - 5.1.2 Backward Abstract Hoare Logic . . . 36
- 5.2 Instantiations . . . 37

5.2.1 Partial Incorrectness, Again 37
5.2.2 Hoare Logic, Again 38

6 Conclusions **39**
6.1 Future work 39
6.2 Related work 40

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 $\{(x, y) \mid (x, y) \in \leq \text{ and } x \neq y\}$.

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

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

We will use partially ordered sets to encode collections of program states.

Definition 1.3 (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
- $\forall c \in C\ c \sqsubseteq \gamma(\alpha(c))$
- $\forall a \in A\ a \leq \alpha(\gamma(a))$

In the context of program analysis, it is common to refer to C as the concrete domain and to A as the abstract domain, as the idea behind the use of Galois connections is to relate C to a simpler and approximate representation of itself.

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

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

Galois insertions are a refinement of Galois connections. By requiring $\alpha \circ \gamma = id$, we are ensuring that all the abstract elements $a \in A$ have distinct concrete representations. It is always possible to obtain a Galois insertion from a Galois connection by simply removing all the elements a such that $\alpha(\gamma(a)) \neq a$ from A.

1.1.2 Lattices

Definition 1.5 (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 1.6 (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 1.7 (Lattice). A poset $(L, \leq)$ is a lattice if it is both a join-semilattice and a meet-semilattice.

Definition 1.8 (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, since it must contain at least $\sup \emptyset$.

Definition 1.9 (Point-wise lifting). Given a complete lattice $(L, \leq)$ and a set A, the set of all functions from A to L, denoted L^A, is usually called the *point-wise lifting* of L. $(L^A, \sqsubseteq)$ is a complete lattice where $f \sqsubseteq g \iff \forall a \in A\ f(a) \leq g(a)$.

1.1.3 Fixpoints

Definition 1.10 (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 $\text{fix}(f) = \{x \mid x \in X \text{ and } x = f(x)\}$.

Fixpoints of function will be used to describe the semantics of programs therefore we are interested in which conditions they exists.

Definition 1.11 (Monotone function). Given two ordered sets $(X, \leq)$ and $(Y, \sqsubseteq)$, a function $f : X \to Y$ is said to be monotone if $x \leq y \implies f(x) \sqsubseteq f(y)$.

The representation of program semantics as functions falls naturally in the category of monotone functions, as different executions of the program are not supposed to influence each other. If the semantics is not monotone, that would mean that the program is able to distinguish between the set of executions x and y and perform different actions.

Definition 1.12 (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 $\forall a \in \text{fix}(f)\ a^* \geq a$.

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 $(fix(f), \leq)$ 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.

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

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. Fixed some reprentation of the state of the program usually denoted by $\mathbb{S}$, 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 $[\![C]\!](\mathit{Init}) \subseteq \mathit{Final}$.

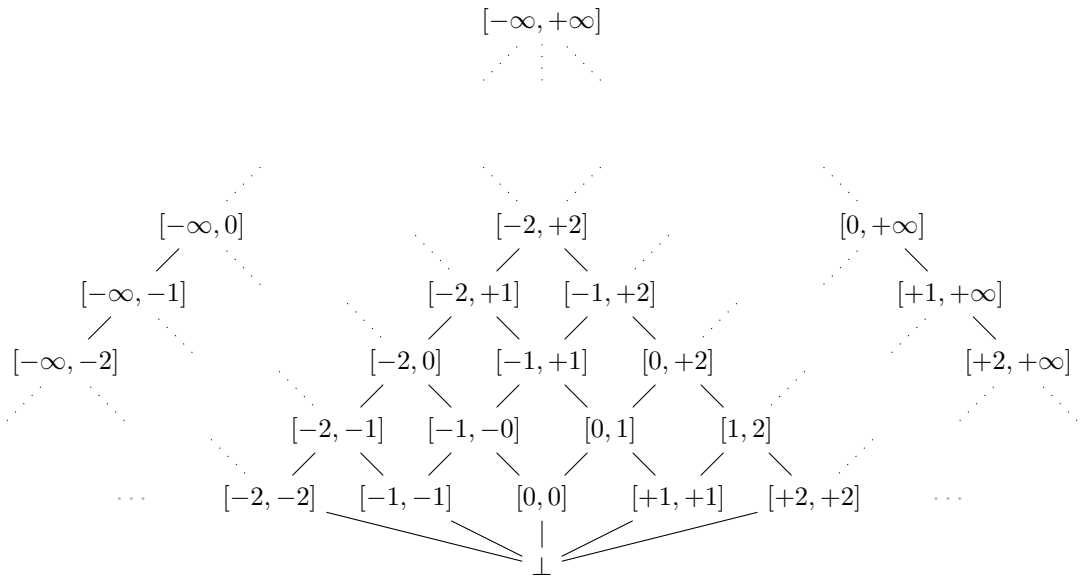
Clearly, this task cannot be performed in general. The solution proposed by the framework of abstract interpretation is to construct an approximation of $[\![\cdot]\!]$, usually denoted by $[\![\cdot]\!]^{\#}$, that is computable.

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 of $\mathbb{S}$ 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:

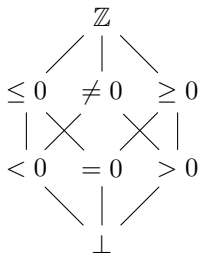


Then, there is a Galois insertion from Int to $\wp(\mathbb{Z})$ defined as:

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

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

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



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

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

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

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

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

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

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

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

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

Table 1.1: Multiplication table for $Sign$ domain

CHAPTER 2

THE ABSTRACT HOARE LOGIC FRAMEWORK

In this chapter, we will develop the basic 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:

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

Where the notation $[\![b]\!]_{base}(p)\downarrow$ is used to denote that $[\![b]\!]_{base}$ is defined on input p.

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.

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

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:

- $\mathbb{1}$:

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

- b:

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

- $C_1 \fatsemi C_2$:

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

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

- $C_1 + C_2$:

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

- C^{fix}:

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

  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.

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

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

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 reason why a fixpoint formulation was chosen will become clear in 2.4.

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

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

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, its computation can diverge. In our case, the termination requirement is not 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, For this we require for A to be a complete lattice or for $[\![b]\!]_{base}$ to be monotone, play an essential role as they guarantee the existance of the required least-fixpoint.

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:

- $\mathbb{1}$:

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

- b:

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

- $C_1 \fatsemi C_2$:

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

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

- $C_1 + C_2$:

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

- C^{fix}:

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

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:

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

- $C_1 \fatsemi C_2$:

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

- $C_1 + C_2$:

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

- C^{fix}:

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

□

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 $[\![C]\!]^C_{ais}$ be some abstract inductive semantics defined on C. Then, the abstract inductive semantics defined on A with $[\![b]\!]^A_{base} \overset{\text{def}}{=} \alpha \circ [\![b]\!]^C_{base} \circ \gamma$ 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 can be seen as the best abstract inductive interpreter on A.

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

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

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

Instead using the least-fixpoint:

$$\begin{aligned}[\![C^{\text{fix}}]\!]^A_{ais}([5,5]) &= \text{lfp}(\lambda P'.[5,5] \vee [\![x > 1?\ ;\ ((even(x)?\ ;\ x := x + 3) + (\neg even(x)?\ ;\ x := x - 2))]\!]^A_{ais}(P')) \\ &= [-\infty, +\infty]\end{aligned}$$

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, instead, when using the least-fixpoint formulation the join is performed before executing the the semantics of the body.

2.3 Abstract Hoare Logic

2.3.1 Hoare logic

Hoare logic [Hoa69; Flo93] was one of the first methods designed for the verification of programs, Its 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 $[\![C]\!](P) \subseteq Q$.

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

We will use the syntax $\models \{P\}\ C\ \{Q\}$ to refer to valid triples, $\not\models \{P\}\ C\ \{Q\}$ to refer to invalid triples.

Example 2.5 (Hoare triples). We have that $\{x \in [1,2]\}\ x := x + 1\ \{x \in [2,4]\}$, 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, $\{x \in [0,10]\}\ (x \leq 20?\ ;\ x := x-1)^{\text{fix}}\ ;\ x \geq 20?\ \{Q\}$ 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:

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

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

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

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

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

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

The proof system described in Definition 2.6 is logically sound, meaning that all its 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 its 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 a generic 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 we have $\top \geq [\![C]\!]^A_{ais}(P)$ □

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:

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

- (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:

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

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

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

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:

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

- $(+)$: 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:

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

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

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

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

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

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

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

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

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

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

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

The triple is valid since:

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

□

The proof system turns out to be relatively complete as well, 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:

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

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

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

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

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

□

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

$$\frac{P \leq_A P \qquad \begin{array}{c}\text{(By Theorem 2.7)}\\ \vdash \langle P\rangle_A\ C\ \langle [\![C]\!]^A_{ais}(P)\rangle\end{array} \qquad Q \geq_A [\![C]\!]^A_{ais}(P)}{\vdash \langle P\rangle_A\ C\ \langle Q\rangle}\ (\leq)$$

□

CHAPTER 3
INSTANTIATING ABSTRACT HOARE LOGIC

In this chapter, we will demonstrate how to instantiate abstract Hoare logic to systematically design novel program logics. We will also show that our abstract Hoare logic framework is sufficiently general to reason about properties that cannot be expressed in standard Hoare logic, notably hyperproperties.

3.1 Hoare logic

According to Observation 2.3, the abstract inductive semantics, when using $(\wp(\mathbb{S}), \subseteq)$ as domain and $[\![b]\!]^{\wp(\mathbb{S})}_{base}(P) = \{[\![b]\!]_{base}(\sigma) \mid \sigma \in P \text{ and } [\![b]\!]_{base}(\sigma)\downarrow\}$ as semantics of basic commands, turns out to be equivalent to the denotational semantics given in Definition 2.2. Therefore Abstract Hoare logic (Definition 2.7) in this instance coincides with Hoare logic (Definition 2.5). Hence we obtain soundess and relative completeness for Hoare logic directly form Theorems 2.6 and 2.8.

3.1.1 Algebraic Hoare Logic

As discussed 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 $\langle \wp(\mathbb{S}), \subseteq\rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows\!\!\!\!\twoheadrightarrow}} \langle A, \leq\rangle$ and $\langle \wp(\mathbb{S}), \subseteq\rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows\!\!\!\!\twoheadrightarrow}} \langle B, \sqsubseteq\rangle$, an Algebraic Hoare triple, denoted by $\overline{\{}P\overline{\}}\ C\ \overline{\{}Q\overline{\}}$, is valid if and only if the Hoare triple $\{\gamma_1(P)\}\ C\ \{\gamma_2(Q)\}$ is valid, namely:

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

In the definition above, Algebraic Hoare Logic is strongly related to standard Hoare Logic, and, therefore, to the strongest postcondition of the program in the concrete domain.

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

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

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

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

[1] Rules $(\overline{\vee})$ and $(\overline{\wedge})$ in [Cou+12] are missing but will be discussed in Section 4.1

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

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

3.1.2 Abstract Interval Logic

By recalling Definition 2.4 and the properties of Galois insertions, we can easily derive a similar family of triples as those in Algebraic Hoare Logic, when pre- and post-conditions range in the same abstract domain.

Example 3.1 (Interval logic). Applying Definition 2.4 to the Galois insertion on the interval domain defined in Example 1.1, we systematically 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 us consider the following program:

$$C \stackrel{\text{def}}{=} ((x := 1) + (x := 3)) \fatsemi ((x = 2? \fatsemi x := 5) + (x \neq 2? \fatsemi x := x - 1))\,.$$

Then the following derivation is valid:

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

π_1:

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

π_2:

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

π_3:

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

π_4:

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

π_5:

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

π_6 :

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

The abstract post-condition $[0,5]$ is the best possible in the interval abstraction, because $[\![C]\!]^{Int}_{ais}(\top) = [0,5]$ holds.

3.2.2.1 Applications

This framework, analogously to 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 *inductive* abstract analyzer on the abstract domain A, and the whole proof system is defined in the abstract domain A, we can check that a derivation is indeed correct algorithmically, as long as we assume that implications and basic commands can be algorithmically checked. These are usually the standard requirements for an abstract domain to be useful in practice, that is, subject of an implementation. 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, of course we cannot decide implications between arbitrary properties.

3.1.3 Abstract vs Algebraic Hoare Logic

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 we did in Example 3.1. From Theorem 2.3, it turns out that $[\![\cdot]\!]^A_{ais}$ is a sound overapproximation of $[\![\cdot]\!]$.

Theorem 3.1 (Abstract entails Algebraic). $\vdash \langle P\rangle_A\ C\ \langle Q\rangle \implies \vdash \overline{\{P\}}\ C\ \overline{\{Q\}}$

Proof.

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

□

However, the converse of Theorem 3.1 does not hold. The relative completeness of Algebraic Hoare Logic is stated with respect to the best correct approximation of $[\![\cdot]\!]$, differently from Abstract Hoare Logic which considers $[\![\cdot]\!]^A_{ais}$.

Example 3.3 (Counterexample to the converse of Theorem 3.1). 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 derive. However, we have that $[\![C]\!]\top = \{0,2\}$. By Theorem 2.5, we can infer $\vdash \{\top\}\ C\ \{\{0,2\}\}$. Hence, by rule $(\overline{S})$, we can obtain $\vdash \overline{\{\top\}}\ C\ \overline{\{[0,2]\}}$, which cannot be derived in Abstract Hoare Logic.

This divergence between abstract and algebraic Hoare logics arises because, through the rule (S) rule of Algebraic Hoare logic, we are always able to prove the best correct approximation of any program C. However, the property of being a best correct approximation is not compositional, meaning that the function composition of two best correct approximations is not the best correct approximation of the composition of these functions. Since in the abstract semantics the program composition is done in "the abstract", it is impossible to expect to be able to derive any possible best correct approximation, except in trivial abstract domains such as the concrete domain $\wp(\mathbb{S})$ or the one-element abstraction $\{\top\}$.

3.2 Hoare logic for hyperproperties

3.2.1 Introduction to Hyperproperties

Program hyperproperties [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 comparing different executions, such as non-interference and information flow security [GM82].

Standard program properties, such as those of Hoare logic, range into the set $\wp(\mathbb{S})$. By contrast, hyperproperties range in $\wp(\wp(\mathbb{S}))$, as they encode relations between different executions. A typical example is the property of a program of being deterministic. For instance, if our programs involve

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 $\models \{\{x = n\}\}\ C\ \{\{x = m\}\}$ holds. However, determinism can be succinctly encoded by a single hyper triple as follows:

$$\models \{\{P \in \wp(\wp(\mathbb{S})) \mid |P| = 1\}\}\ C\ \{\{Q \in \wp(\wp(\mathbb{S})) \mid |Q| = 1\}\}\,.$$

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

$$\{[\![C]\!](P) \mid P \in \mathbb{X}\}\,. \qquad \square$$

Note that Definition 3.3 is justified by the requirement that we are interested in modeling the strongest postcondition of every initial state ranging in $\mathbb{X}$.

3.2.2 Inductive Definition of the Strongest Hyper Postcondition

To design a sound and relatively complete logic for hyperproperties within our framework, it is crucial to define an abstract inductive semantics that precisely computes the strongest hyper postcondition. This objective has been investigated in prior works [Ass+17; MP18], mostly in an abstract interpretation-based scenario. However, existing approaches often provide an over-approximation 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 the branching command **if** b **then** C_1 **else** C_2 from a starting hyper-state $\mathbb{T}$ is defined to be $\{[\![b?]\!] \fatsemi C_1[\![]\!]T \cup [\![\neg b?]\!] \fatsemi C_2[\![]\!] \mid T \in \mathbb{T}\}$, thereby lacking inductiveness. It is worth remarking that with this noninductive definition, we have that, for any program C, the hypersemantics of **if** $1 = 1$ **then** C coincides with that of C, thus making this hyper semantics practically meaningless for program analysis.

The fundamental issue lies in the fact that in the domain $\wp(\wp(\mathbb{S}))$, ordered w.r.t. the usual subset inclusion, the least upper bound, namely set union, fails to distinguish between different executions, as shown by the following example.

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

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

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

When applying the rule for non-deterministic choice,

$$[\![C_1 + C_2]\!]_{ais}^{\wp(\wp(\mathbb{S}))}(\mathcal{P}) = [\![C_1]\!]_{ais}^{\wp(\wp(\mathbb{S}))}(\mathcal{P}) \ \cup\ [\![C_2]\!]_{ais}^{\wp(\wp(\mathbb{S}))}(\mathcal{P})\,,$$

the union of outermost sets is considered rather than of the innermost sets that include actual executions. Attempts to alter the ordering on the domain $\wp(\wp(\mathbb{S}))$ turned out to be unsuccessful as each set lacks information about the generating execution, thus leading to an unavoidable loss of precision in the definition of the union.

To the best of our knowledge, no approach has been put forward for defining an abstract inductive semantics that exactly computes the strongest hyper postcondition. Existing works just provide sound overapproximations, which are adequate for abstract interpreters, but are not precise enough for verifying certain hyperproperties within Abstract Hoare logic, especially where precision in abstract inductive semantics is compromised.

3.2.3 Hyper Domains

To address the limitations of $\wp(\wp(\mathbb{S}))$ discussed above, we introduce a family of domains designed to keep track of the execution of interest across different executions. Our definition leverages an index set K (K stands for "keys") to enumerate individual executions, and, accordingly, define the join operation in a manner that allows us to distinguish them.

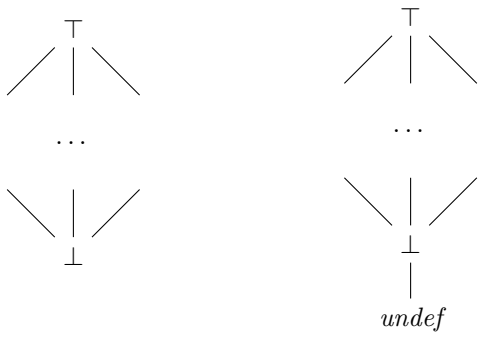


Figure 3.1: On the left the Hasse diagram of B, on the right the Hasse diagram of $B + undef$

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

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

The structure of complete lattice of $H(B)_K$ is defined by lifting the pointwise lattice of $B + \text{undef}$, where $B + \text{undef}$ forms a complete lattice on B where *undef* is the new bottom element, meaning that $undef < \bot_B$ (see Figure 3.2.3). □

Let us point out that the role played by the index set K in Definition 3.4 is merely that of encoding different executions, where no specific requirements on its elements are assumed, while K should simply have enough distinct indices to account for all the executions of interest.

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

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

Hence, this notion of hyper instantiation lifts the abstract inductive semantics from the domain B to its "hype" version, by applying the semantics of basic commands from B to each execution. Next, we show that the abstract inductive semantics instantiated on a hyper-domain preserves non-interference, meaning that the hyper inductive semantics yields the same results as computing the original semantics on each execution.

Theorem 3.2 (Non-interference between executions). *For all programs C, we have that*

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

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:

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

- $C_1 \fatsemi C_2$:

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

- $C_1 + C_2$:

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

- C^{fix}:

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

□

3.2.4 Inductive Definition for Hyper Postconditions

Our definition of hyper domains allows us to overcome the limitations of $\wp(\wp(\mathbb{S}))$ found in literature. On the other hand, 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 *conversion pair*, to make this conversion simpler. Of course, there exist infinitely many functions to convert a standard hyperproperty into a version using hyper domains—due to the infinite representations of the same property—and we exploit a single representative (which is an 1-1 function) to encapsulate all these representations, so that our results will remain independent of the chosen indexing function.

Definition 3.6 (Conversion Pair). Given a 1-1 function $idx : B \to K$, the conversion pair $\langle \alpha, \beta \rangle$ is defined as follows:

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

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

By instantiating the hyper domain to $H(\wp(\mathbb{S}))_{\mathbb{R}}$, we show that our abstract inductive semantics actually defines 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.

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

□

3.2.5 Hyper Hoare Triples

The instantiation of hyper domains provides a sound and complete Hoare-like logic for hyperproperties, particularly when using the function α of Definition 3.6 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.

We assume that the language $\mathbb{L}$ uses single-variable assignments only, so that program states are represented simply by integers.

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

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

We prove that $\mathbb{1}$ (i.e., the 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 $\alpha(\mathbb{P}) = \{\ldots, \{-1\}, \{0\}, \{1\}, \ldots\}$, we can therefore infer that the command is deterministic.

Similarly, we can prove that the increment function is deterministic as follows:

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

where $\mathbb{Q} \stackrel{\text{def}}{=} \lambda r. \begin{cases} \{x+1\} & \exists \{x\} \in \wp(\mathbb{S}) \text{ such that } idx(P) = r \\ \mathit{undef} & \text{otherwise} \end{cases}$ Clearly, we have that $\alpha(\mathbb{Q}) = \{..., \{0\}, \{1\}, \{2\}, ...\}$, thus proving determinism.

Finally, we can establish that a nondeterministic choice between two identical programs remains deterministic by the following proof:

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

However, different programs cannot be handled in the same way:

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

where the proof tree π is the following:

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

Clearly, we have that $\alpha(\mathbb{P} \vee \mathbb{Q}) = \{..., \{-1, 0\}, \{0, 1\}, \{1, 2\}, ...\}$. Let us observe that different elements within the hyper domain may correspond to the same hyperproperty, therefore 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. □

Hyper Hoare Logic [DM23] is a related Hoare-like logic that provides a sound and relatively complete program logic for hyperproperties. While Hyper Hoare Logic was specifically designed for this purpose, it turns out to be equivalent to the logic derived from our abstract Hoare logic framework. Notably, it departs from using the classical disjunction connective—equivalent to the least upper bound in $\wp(\wp(\mathbb{S}))$—adopting instead a peculiar disjunction operator $\otimes$ which is able to distinguish different executions, similarly to the least upper bound in our hyper domain.

3.3 Partial Incorrectness

Any instantiation of the abstract inductive semantics provides a corresponding program logic, as the semantics is parameterized by the complete lattice A, and the dual A^{op} of a complete lattice A is a complete lattice as well. 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 $[\![\cdot]\!]^{A^{\mathrm{op}}}_{base} = [\![\cdot]\!]^{A}_{base}$. □

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 reversed, operations such as joins and meets are swapped, leading to an inversion of lfp and gfp. Hence, the dual abstract inductive semantics, as given in the dual lattice, can be formulated as follows:

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

In this dual abstract inductive semantics, we observe 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 (i.e., 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.

Due to the reverse order in the dual lattice, the validity of Abstract Hoare triples is accordingly changed as follows:

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

When deriving the dual abstract inductive semantics from the abstract inductive semantics on $\wp(\mathbb{S})$ (i.e., strongest postcondition), the dual semantics corresponds to the strongest liberal postcondition as introduced in [ZK22] (notably in the Boolean case). These triples are referred to as "partial incorrectness" triples, entailing that if $\models \langle Q \rangle_{A^{\mathrm{op}}}\ C\ \langle P \rangle$, then P over-approximates the states reaching Q, accounting for termination. This notion aligns with the "necessary preconditions" investigated in [Cou+13], where the 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 given in Definition 2.8 is rather minimalistic. The overall objective of Abstract Hoare logic is to establish a comprehensive framework for designing Hoare-like logics, aiming to require as few assumptions as possible on both the assertion language and the semantics of base commands. Throughout this chapter, we explore its potential to derive additional sound rules for the proof system by introducing more constraints either on the complete lattice of assertions or on the semantics of base commands.

4.1 Merge rules

When designing a software verification tool, the capability to perform multiple local reasonings and, subsequently, merge their results provides clear benefits. An example of this arises for the conjunction rule in concurrent separation logic [BO16].

In Hoare logic, the following two merge rules turn out to be sound:

Definition 4.1 (Merge rules in Hoare logic).

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

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

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

Definition 4.2 (Merge rules in Abstract Hoare logic).

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

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

Example 4.1 (Counterexample for the rule $(\vee)$). Let $\langle\cdot\rangle_{Int} \cdot \langle\cdot\rangle$ be the Abstract Hoare logic instantiation of Example 3.1 for the Abstract Interval Logic, and let

$$C \stackrel{\text{def}}{=} (x = 4?\fatsemi x := 50) + (x \neq 4?\fatsemi x := x + 1)\,.$$

We have the following two derivations:

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

where π_1 is:

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

and π_2 is:

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

Moreover, we have:

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

where π_3 is:

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

and π_4 is:

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

Thus we can derive the following proof tree:

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

However, this is clearly unsound because:

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

and we have that $[4,50] \not\leq [4,6]$. □

One might claim that the issue is merely "local", because $\gamma([3,3]) \cup \gamma([5,5]) = \{3,5\} \neq \{3,4,5\} = \gamma([3,3] \vee [5,5])$, so that one could guess that requiring a local disjunctive condition such as $\gamma(P_1 \vee P_2) = \gamma(P_1) \cup \gamma(P_2)$ could fix the unsoundness, since this least upper bound adds new states in the precondition. However, this guess turns out to be incorrect. In fact, we can construct arbitrary programs that exploit the fact that $\vee$ is, in general, a convex operation capable of introducing new elements in its over-approximation.

Example 4.2 (Counterexample for a local disjunctive rule). Consider the following rule

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

and let $\langle \cdot \rangle_{Int} \cdot \langle \cdot \rangle$ be the Abstract Hoare logic instantiation to Interval Logic described in Example 3.1. Let us consider the program

$$C \stackrel{\text{def}}{=} (x = 0? + x = 2?) \fatsemi x = 1? \, .$$

The following derivation can be inferred:

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

where π_1 is:

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

Also, we have that:

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

where π_2 is:

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

Thus, we have in turn the following proof tree:

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

However, its conclusion is clearly unsound because:

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

and, obviously, $[1,1] \not\leq \bot$. □

Example 4.2 shows the actual root cause of the issue of the rule for disjunction, namely the overapproximation introduced by the abstract join $\vee$, which is unrelated to the preconditions. More precisely, consider the program

$$C' \stackrel{\text{def}}{=} (x = 1? \fatsemi x = 0?) + (x = 2? \fatsemi x = 0?) \, ,$$

where the issue does not appear. Despite the fact that C and C' are equivalent programs in the concrete domain $\wp(\wp(\mathbb{S}))$, they differ in the *Int* abstract domain. Therefore, the equality $[\![(C_1 + C_2) \fatsemi C_3]\!]^A_{ais} = [\![(C_1 \fatsemi C_3) + (C_2 \fatsemi C_3)]\!]^A_{ais}$, in general, does not hold. In particular, we can easily demonstrate that for a subset of the preconditions—namely, those admitting a program capable of having them as a postcondition—requiring the validity of the distributivity rule is equivalent to assuming that the additivity of the semantics.

Theorem 4.1 (Equivalence between additivity and distributivity). *Let C_1, C_2 and C' be programs. Assume for all $i \in [1,3]$, there exists a corresponding program C_{P_i} such that $\forall\, Q.\ [\![C_{P_i}]\!]^A_{ais}(Q) = P_i$ holds. Then,*

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

Proof.

($\Longleftarrow$)

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

($\Longrightarrow$)

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

□

Theorem 4.1 provides the intuition explaining why the above rule $\vee$ fails: in general, the abstract inductive semantics lacks additivity, stemming from the non-additivity of the base commands.

Theorem 4.2 (Additivity of the abstract inductive semantics).
If, for all base commands b, $[\![b]\!]^A_{base}(P_1 \vee P_2) = [\![b]\!]^A_{base}(P_1) \vee [\![b]\!]^A_{base}(P_2)$ then, for all programs C, $[\![C]\!]^A_{ais}(P_1 \vee P_2) = [\![C]\!]^A_{ais}(P_1) \vee [\![C]\!]^A_{ais}(P_2)$.

Proof. By structural induction on C:

- $\mathbb{1}$:

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

- b:

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

- $C_1 \fatsemi C_2$:

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

- $C_1 + C_2$:

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

- C^{fix}:

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

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.

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

Now we show that this fixpoint is indeed the least fixpoint. Let P be any fixpoint, i.e., $P = P_1 \vee P_2 \vee [\![C]\!]^A_{ais}(P)$. Then, by definition of $\vee$, we have that $P_i \vee [\![C]\!]^A_{ais}(P) \leq P_1 \vee P_2 \vee [\![C]\!]^A_{ais}(P)$. Since F_i is the least fixpoint, we have that $F_i \leq P_i \vee [\![C]\!]^A_{ais}(P)$, thus, in turn, $F_1 \vee F_2 \leq P_1 \vee [\![C]\!]^A_{ais}(P) \vee P_2 \vee [\![C]\!]^A_{ais}(P) = P_1 \vee P_2 \vee [\![C]\!]^A_{ais}(P) = P$. Hence, $F_1 \vee F_2$ is the least fixpoint.

□

We can provide a sufficient condition for the additivity of the abstract inductive semantics defined through a Galois insertion:

Theorem 4.3. *Let $\langle C, \sqsubseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\!\!\rightarrow}} \langle A, \leq\rangle$ be a Galois insertion. If $[\![\cdot]\!]^C_{ais}$ and γ are additive functions then the abstract inductive semantics $[\![\cdot]\!]^A_{ais}$ induced by the Galois insertion is additive as well.*

Proof.

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

Then, by Theorem 4.2, we conclude that $[\![\cdot]\!]^A_{ais}$ is additive. □

We can also prove that the additivity of the abstract inductive semantics is sufficient to ensure the soundness of the rule $(\vee)$.

Theorem 4.4 (Soundness of the rule $(\vee)$). *If $[\![\cdot]\!]^A_{ais}$ 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.

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

□

Theorems 4.4 and 4.3 correspond to the result for Algebraic Hoare logic [Cou+12, Theorem 6] showing that the rule $(\overline{\vee})$ is sound under the condition that γ is additive. A similar argument can be applied to ensure the soundness of the rule $(\wedge)$ rule when the semantics is assumed 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 sign abstraction depicted in Example 1.2 is one such domain, guaranteeing 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 defined in a forward fashion, as we defined $[\![C_1 \fatsemi C_2]\!]^A_{ais} \stackrel{\text{def}}{=} [\![C_2]\!]^A_{ais} \circ [\![C_1]\!]^A_{ais}$. However, except for the rule $(\fatsemi)$, we never explicitly used this assumption. Thus, we can apply the theory of Abstract Hoare logic to define a slight variation thereof, called Backward Abstract Hoare logic, describing Hoare logics where the semantics is defined in a backward fashion.

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 underlying 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 $[\![\cdot]\!]^A_{bais} : \mathbb{L} \to A \to A$ is defined as follows:

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

Let us remark that the only difference with the abstract inductive semantics given in Definition 2.3 concerns the sequential composition $C_1 \fatsemi C_2$. 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 $[\![C_2]\!]^A_{bais}(P) \leq_A [\![C_2]\!]^A_{bais}(Q)$.

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

□

5.1.2 Backward Abstract Hoare Logic

We can give a corresponding definition of backward abstract Hoare triple, which is the same as for abstract Hoare triples except for the fact that the backward abstract inductive semantics is used.

Definition 5.2 (Backward Abstract Hoare triple). Given an abstract inductive semantics $[\![\cdot]\!]^A_{bais}$ on the complete lattice A, the backward abstract Hoare triple denoted by $\langle P \rangle^B_A\ C\ \langle Q \rangle$ is valid when $[\![C]\!]^A_{bais}(P) \leq_A Q$ holds, namely,

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

Intuitively now the roles of P and Q are reversed, in Hoare logic and Abstract Hoare logic P was a precondition and Q was a postcondition, in Backward Abstract Hoare logic instead P is a postcondition and Q a precondition.

Clearly, the proof system only needs to be modified to accommodate the new semantics for program composition, while the other rules are unchanged.

Definition 5.3 (Backward Abstract Hoare rules).
We provide the rule for sequential composition only—all the other rules are given 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 above composition rule can be intuitively read as follows: If executing backward C_2 from state P leads to a state Q, and executing backward C_1 from state Q leads to a state R, then executing backward C_2 followed by C_1 from state P leads to the state R. We can prove soundness and completeness of this backward proof system.

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. The last step in the derivation is as follows:

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

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

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

□

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

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

□

Theorem 5.4 (Relative completeness).

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

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

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

□

5.2 Instantiations

5.2.1 Partial Incorrectness, Again

An abstract inductive semantics induces systematically 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 with the semantics of basic commands defined by: $([\![\cdot]\!]^A_{base})^{-1}$.

Accordingly, the reverse abstract inductive semantics is defined as follows:

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

According to the intuition that the abstract inductive semantics is an abstract version of the strongest postcondition, which intuitive interpretation should we give to reverse abstract inductive semantics? This 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 $\wp(\mathbb{S})$ (i.e., the strongest postcondition), the reverse semantics becomes the weakest precondition. Hence, from the validity of the corresponding triples, we have that:

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

This program logic has been studied in [Asc+24] under the name of NC, and it is indeed equivalent to the logic described in Section 3.3.

5.2.2 Hoare Logic, Again

Following Section 3.3, we can first obtain the reverse semantics, and next the dual of the reverse semantics. This latter semantics is therefore defined as follows:

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

This definition corresponds to the reverse inductive semantics obtained from the abstract inductive semantics, which is the abstract version of the weakest liberal precondition. In fact, when the dual is applied to $\wp(\mathbb{S})$ (the strongest postcondition), the reverse semantics becomes the weakest liberal precondition. This can be easily seen by looking at the definition for the non-deterministic choice and the loop commands: if we interpret the input of the semantics as the final state, we are computing all the states that must reach (up to termination) the final states.

Hence, from the validity of the triples we obtain:

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

We also have that $wlp(C, P) \supseteq Q \iff [\![C]\!](Q) \subseteq P$, hence this is equivalent to Hoare logic.

CHAPTER 6

CONCLUSIONS

In this thesis, we extended the ubiquitous traditional Hoare logic by transforming it into an abstract and versatile framework to be instantiated to design novel program logics. By following well-known abstract interpretation principles, we developed a general methodology for reasoning about a broader range of program properties. Our study demonstrated that multiple program logics known in literature can be viewed as instantiations of Abstract Hoare Logic. Notably, while constructing a program logic for hyperproperties in our general 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. In particular, this highlighted some fundamental differences with the Incorrectness Logic proof system: the infinitary loop rule required for the relative completeness of Incorrectness Logic:

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

is not due to the fact that Incorrectness Logic aims at proving an underapproximation but rather because Incorrectness Logic is a "total correctness logic", meaning that it inherently carries a proof of termination. We also studied 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

As stimulating directions of future work, we plan the investigate the following questions.

Total correctness/Incorrectness logics. We have shown 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 their relationship is analogous to that between partial correctness and incorrectness logics, 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 applied here to Abstract Hoare Logic to invert the semantics and the lattice, we can obtain all the four program logics that are missing, therefore completing the whole spectrum of possible logics.

Hyper domains. We investigated hyper domains to encode the strongest hyper postcondition, and, correspondingly, we 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 apply the same technique to the abstract inductive semantics instantiated with a hyper domain of $\wp(\mathbb{S})$, to study whether this would lead to some interesting novel logics or if they are all equivalent (this could happen because hyperproperties can disprove themselves, meaning that if a triple in hyper hoare logic is false we can prove its negation).

An interesting feature 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 given in [DM23] is actually quite similar to the one obtained with the hyper domains, but the Exist rule is missing. If the goal is that of proving the completeness of Hyper Hoare Logic, then the Exist rule must be embedded somewhere in the rules of Abstract Hoare Logic.

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 is inverted. A potential solution would be to make the semantics parametric on the composition $[\![C_1 \fatsemi C_2]\!]^A_{ais} \stackrel{\text{def}}{=} [\![C_1]\!]^A_{ais} \star [\![C_2]\!]^A_{ais}$ and let $P \star Q = Q \circ P$ for the forward semantics and $P \star Q = P \circ Q$ for the backward semantics. However, this approach would not be uniform when defining the command composition rule for the proof system.

6.2 Related work

The idea of systematically constructing program logics, of course, 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 (i.e., $[\![(C_1 + C_2) \fatsemi C_3]\!]^A_{ais} \neq [\![(C_1 \fatsemi C_3) + (C_2 \fatsemi C_3)]\!]^A_{ais}$), thus violating one of the axioms of Kleene algebras. A similar alternative was investigated 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 pivotal 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 these partial (in)-correctness Hoare-like logics.

The fundamental approach of Outcome Logic [ZDS23] is similar to that of Abstract Hoare Logic. Like Abstract Hoare Logic, the semantics of the language in Outcome Logic is parametric on the domain of execution, although 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, and has not be conceived to be a minimal theory for sound and complete Hoare-like logics. In fact, Outcome Logic does not bring a result of (relative) completeness. As discussed in [DM23], Outcome Logic with the powerset instantiation is actually a proof system for 2-hyperproperties (namely, hyperproperties regarding at most two executions). Thus, Outcome triples can be proved in the instantiation of Abstract Hoare Logic provided in section 3.2.1, 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, 6, 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, 34).

[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. 6).

[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. 28, 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).

[GM82] J. A. Goguen and J. Meseguer. "Security Policies and Security Models". In: *1982 IEEE Symposium on Security and Privacy*. 1982, pp. 11–11. DOI: 10.1109/SP.1982.10014 (cit. on p. 23).

[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).