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 ...

Acknowledgments

To . . .

Contents

Chapter 1

Introduction and Background

In this chapter we give a brief introduction in the backround knowledge required to understand the rest of the thesis:

1.1 Order theory

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

1.1.1 Partial Orders

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

- Reflexivity: $\forall x \in X,\ (x, x) \in \leq$
- Anti-symmetry: $\forall x, y \in X,\ (x, y) \in \leq$ 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.

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

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

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

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

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

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

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

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

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

1.1.2 Lattices

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

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

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

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

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

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

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

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

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

Observation 1.2. A complete lattice cant be empty.

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

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

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

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

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

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

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

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

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

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

Proof.

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

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

1.2 Abstract Interpretation

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

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

1.2.1 Abstract Domains

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

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

Example 1.1 (Interval Domain)**.** Let $Int = \{[a, b] \mid a, b \in \mathbb{Z} \cup \{+\infty, -\infty\}, a \leq b\} \cup \{\bot\}$ be ordered by set inclusion. Then, there is a Galois insertion from Int to $\wp(\mathbb{Z})$ defined as:

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

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

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

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

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

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

Chapter 2

The abstract Hoare logic framework

In this chapter we will introduce the general framework of *Abstract Hoare logic*

- The $\mathbb{L}$ programming language
- *Abstract inductive semantics*
- *Abstract Hoare logic*

2.1 The $\mathbb{L}$ programming language

2.1.1 Syntax

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

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

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

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

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

2.1.2 Semantics

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

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

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

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

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

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

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

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

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

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

□

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

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

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

By structural induction on C:

- $\mathbb{1}$:

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

- b:

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

- $C_1 \fatsemi C_2$:

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

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

- $C_1 + C_2$:

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

- C^{fix}:

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

□

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

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

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

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

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

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

2.2 Abstract inductive semantics

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

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

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

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

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

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

By structural induction on C:

- $\mathbb{1}$:

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

- b:

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

- $C_1 ⨟ C_2$:

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

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

- $C_1 + C_2$:

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

- C^{fix}:

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

□

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

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

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

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

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

This can be easily assessed by comparing the two definitions.

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

2.2.1 Connection with Abstract Interpretation

It turns out that the definition of abstract inductive semantics is closely related to the one of abstract semantics in [CC77].

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

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

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

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

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

2.3 Abstract Hoare Logic

2.3.1 Hoare logic

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

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

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

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

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

Definition 2.6 (Hoare logic)**.**

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

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

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

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

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

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

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

Theorem 2.5 (Soundness)**.**

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

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

Theorem 2.6 (Relative completeness)**.**

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

2.3.2 Abstracting Hoare logic

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

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

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

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

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

Proof system

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The proofsystem is sound:

Theorem 2.7 (Soundness)**.**

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

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

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

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

The triple is valid since:

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

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

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

The triple is valid since:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

And clearly is bigger than the least one $\mathrm{lfp}(\lambda P' \to P \vee_A [\![C]\!]^A_{ais}(P')) \leq_A P$ thus making the triple valid.

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

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

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

The triple is valid since:

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

□

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

We will start by proving a slightly weaker result:

Theorem 2.8 (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 $[\![C_1 + C_2]\!]_{base}(P) = [\![C_1]\!]_{base}(P) \vee_A [\![C_2]\!]_{base}(P)$

$$\dfrac{\dfrac{P \leq_A P \qquad \overset{\text{(Inductive hypothesis)}}{\vdash \langle P\rangle_A\, C_1\, \langle [\![C_1]\!]^A_{ais}(P)\rangle} \qquad [\![C_1]\!]^A_{ais}(P) \leq_A [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P)}{\vdash \langle P\rangle_A\, C_1\, \langle [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P)\rangle}\,(\leq) \qquad \pi_1}{\vdash \langle P\rangle_A\, C_1 + C_2\, \langle [\![C_1]\!]^A_{ais}(P) \vee_A [\![C_2]\!]^A_{ais}(P)\rangle}\,(+)$$

Where π_1:

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

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

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

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

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

□

Now we can finally show the relative completeness:

Theorem 2.9 (Relative completeness)**.**

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

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

$$\dfrac{P \leq_A P \qquad \overset{\text{(By Theorem 2.8)}}{\vdash \langle P\rangle_A\, C\, \langle [\![C]\!]^A_{ais}(P)\rangle} \qquad Q \geq_A [\![C]\!]^A_{ais}(P)}{\vdash \langle P\rangle_A\, C\, \langle Q\rangle}\,(\leq)$$

□

Chapter 3

Instantiating Abstract Hoare Logic

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

3.1 Hoare logic

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

3.2 Interval logic

3.3 Hoare logic for hyperproperties

3.3.1 Introduction to Hyperproperties

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

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

Definition 3.1 (Strongest Hyper Postcondition)**.** The strongest postcondition of a program C starting from a collection of states $\chi \in \wp(\wp(\mathbb{S}))$ is defined as:

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

3.3.2 Inductive Definition of the Strongest Hyper Post Condition

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

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

Example 3.1. Let $\chi \overset{\text{def}}{=} \{\{1, 2, 3\}, \{5\}\}$. Clearly,

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

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

To our knowledge, there is no literature on an abstract inductive semantics that exactly computes the strongest hyper postcondition.

3.3.3 Hyper Domains

To address this, we will define a more complex family of domains whose semantics satisfy the distributive property of different executions. We will use a set K to keep track of each execution and define the join operation in such a way that it does not confuse different executions together.

Definition 3.2 (Hyper Domain)**.** Given a complete lattice B and a set K, the hyper domain $H(B)_K$ is defined as:

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

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

Definition 3.3 (Hyper Instantiation)**.** Given an instantiation of the abstract inductive semantics on a domain B with semantics of the base commands $[\![\cdot]\!]^B_{base}$, we can instantiate the abstract inductive semantics for the hyper domain $H(B)_K$ with base commands defined as follows:

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

Now we prove that the hyper instantiate is dristibutive:

Theorem 3.1 (Distributivity)**.**

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

Proof. By structural induction on C:

- $\mathbb{1}$:

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

- b:

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

- $C_1 \fatsemi C_2$:

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

- $C_1 + C_2$:

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

- $C^\star$:

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

□

3.3.4 Inductive definition for Hyper postconditions

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

Definition 3.4 (Conversion Pair)**.** Given an injective function $idx : B \to K$, we can define the conversion pair as follows:

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

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

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

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

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

Theorem 3.2 (Abstract Inductive Semantics as Strongest Hyper Postcondition)**.**

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

Proof.

$$\begin{aligned}
\alpha(\llbracket C \rrbracket_{ais}^{H(\wp(\mathbb{S}))_\mathbb{R}}(\beta(\mathcal{X}))) &= \alpha(\lambda r \to \llbracket C \rrbracket_{ais}^{\wp(\mathbb{S})}(\beta(\mathcal{X})(r))) && \text{By Theorem 3.1} \\
&= \{\llbracket C \rrbracket_{ais}^{\wp(\mathbb{S})}(\beta(\mathcal{X})(r)) \downarrow \mid r \in \mathbb{R}\} && \text{By the definition of } \alpha \\
&= \{\llbracket C \rrbracket_{ais}^{\wp(\mathbb{S})}(P) \mid P \in \mathcal{X}\} && \text{By the definition of } \beta \text{ and injectivity}
\end{aligned}$$

□

3.3.5 Hyper Hoare triples

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

Example 3.2 (Determinism in Abstract Hoare Logic)**.** As explained in Example 3.1, we can express that a command is deterministic (up to termination) by proving that the hyperproperty $\{P \mid |P| = 1\}$ is both a precondition and a postcondition of the command.

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

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

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

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

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

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

The same can be done with the increment function:

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

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

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

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

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

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

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

Where π:

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

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

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

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

Bibliography

[Ass+17] Mounir Assaf et al. "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. 16).

[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, 9).

[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. 10, 11).

[Cou+12] Patrick Cousot et al. "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 p. 10).

[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. 15).

[Dij74] Edsger W. Dijkstra. "Guarded commands, non-determinacy and a calculus for the derivation of programs". circulated privately. June 1974. URL: http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD418.PDF (cit. on p. 5).

[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 p. 19).

[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. 7).

[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. 9).

[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. 9, 10).

[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 et al. Cham: Springer International Publishing, 2021, pp. 325–343. ISBN: 978-3-030-88701-8 (cit. on p. 10).

[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. 16).

[Sco70] Dana Scott. *OUTLINE OF A MATHEMATICAL THEORY OF COMPUTATION*. Tech. rep. PRG02. OUCL, Nov. 1970, p. 30 (cit. on p. 1).