



University of Padova

DEPARTMENT OF MATHEMATICS "TULLIO LEVI-CIVITA"

MASTER DEGREE IN COMPUTER SCIENCE

Titolo della tesi



Supervisor

Prof. Prof 1

Co. Supervisor

Prof. Prof 2

Candidate

Luca Zaninotto

ACADEMIC YEAR 2023-2024

Abstract

Abstract

Acknowledgments

?

Contents

1	Framework	1
1.1	The Imp language	1
1.2	Semantics	1
1.2.1	Functions in Imp	4
1.3	Deciding invariant finiteness	6
2	Intervals	9
2.1	Interval Analysis	9
2.1.1	Computing the interval semantics	12
3	Non relational collecting	15

Chapter 1

Framework

1.1 The Imp language

We'll denote by \mathbb{Z} the set of integers with the usual order plus two bonus elements $-\infty$ and $+\infty$, s.t. $-\infty \leq z \leq +\infty \quad \forall z \in \mathbb{Z}$. We also extend addition and subtraction by letting, for $z \in \mathbb{Z}$ $+\infty + z = +\infty$ $-\infty - z = -\infty$ and $-\infty + z = -\infty$ $+\infty - z = +\infty$.

We'll focus on the following non-deterministic language.

$$\begin{aligned} \text{Exp} \ni e ::= & \quad x \in S \mid x \in [a, b] \mid x \leq k \mid x > k \mid \text{true} \mid \text{false} \mid \\ & \quad x := k \mid x := y + k \mid x := y - k \\ \text{Imp} \ni C ::= & \quad e \mid C + C \mid C; C \mid C^* \end{aligned}$$

where $x, y \in \text{Var}$ a finite set of variables of interest, i.e., the variables appearing in the considered program, $S \subseteq \mathbb{N}$ is (possibly empty) subset of numbers, $a \in \mathbb{Z} \cup \{-\infty\}$, $b \in \mathbb{Z} \cup \{+\infty\}$, $a \leq b$, $k \in \mathbb{Z}$ is any finite integer constant.

1.2 Semantics

The first building block is that of environments. We'll use environments to model the most precise invariant our semantic can describe for a program.

Definition 1.1 (Environments). Environments are (total) maps from variables to (numerical) values

$$\text{Env} \triangleq \{\rho \mid \rho : \text{Var} \rightarrow \mathbb{Z}\}$$

Note: the set of notable variables is assumed to be finite.

Definition 1.2 (Semantics of Basic Expressions). For basic expressions $e \in \text{Exp}$ the concrete

semantics $\llbracket \cdot \rrbracket : \text{Exp} \rightarrow \text{Env} \rightarrow \text{Env} \cup \{\perp\}$ is recursively defined as follows:

$$\begin{aligned}
\llbracket x \in S \rrbracket \rho &\triangleq \begin{cases} \rho & \rho(x) \in S \\ \perp & \text{otherwise} \end{cases} \\
\llbracket x \in [a, b] \rrbracket \rho &\triangleq \begin{cases} \rho & \rho(x) \in [a, b] \\ \perp & \text{otherwise} \end{cases} \\
\llbracket x \leq k \rrbracket \rho &\triangleq \begin{cases} \rho & \rho(x) \leq k \\ \perp & \text{otherwise} \end{cases} \\
\llbracket x > k \rrbracket \rho &\triangleq \begin{cases} \rho & \rho(x) > k \\ \perp & \text{otherwise} \end{cases} \\
\llbracket \text{true} \rrbracket \rho &\triangleq \rho \\
\llbracket \text{false} \rrbracket \rho &\triangleq \perp \\
\llbracket x := k \rrbracket \rho &\triangleq \rho[x \mapsto k] \\
\llbracket x := y + k \rrbracket \rho &\triangleq \rho[x \mapsto \rho(y) + k] \\
\llbracket x := y - k \rrbracket \rho &\triangleq \rho[x \mapsto \rho(y) - k]
\end{aligned}$$

The next building block is the concrete collecting semantics for the language, maps each program in Imp to a function on the \mathbb{C} complete lattice.

Definition 1.3 (Concrete collecting domain). The concrete collecting domain for the Imp language concrete collecting semantics is the complete lattice

$$\mathbb{C} \triangleq \langle 2^{\text{Env}}, \subseteq \rangle$$

We can therefore define the concrete collecting semantics for our language:

Definition 1.4 (Concrete collecting semantics). The concrete collecting semantics for Imp is given by the total mapping

$$\langle \cdot \rangle : \text{Imp} \rightarrow \mathbb{C} \rightarrow \mathbb{C}$$

which maps each program $C \in \text{Imp}$ to its total mapping

$$\langle C \rangle : \mathbb{C} \rightarrow \mathbb{C}$$

on the complete lattice \mathbb{C} . The semantics is recursively defined as follows: given $X \in 2^{\text{Env}}$

$$\begin{aligned}
\langle e \rangle X &\triangleq \{ \llbracket e \rrbracket \rho \mid \rho \in X, \llbracket e \rrbracket \rho \neq \perp \} \\
\langle C_1 + C_2 \rangle X &\triangleq \langle C_1 \rangle X \cup \langle C_2 \rangle X \\
\langle C_1; C_2 \rangle X &\triangleq \langle C_2 \rangle (\langle C_1 \rangle X) \\
\langle C^* \rangle X &\triangleq \bigcup_{i \in \mathbb{N}} \langle C \rangle^i X
\end{aligned}$$

Along with the collecting semantics we're also defining a one step transition relation.

Definition 1.5 (Program State). Program states are tuples of programs and program environments:

$$\text{State} \triangleq \text{Imp} \times \text{Env}$$

Definition 1.6 (Small step semantics). The small step transition relation $\rightarrow: \text{State} \times (\text{State} \cup \text{Env})$ is a small step semantics for the Imp language. It is defined based on the following rules

$$\begin{array}{c} \frac{\langle e \rangle \rho \neq \perp}{\langle e, \rho \rangle \rightarrow \langle e \rangle \rho} \text{ expr} \\[10pt] \frac{}{\langle C_1 + C_2, \rho \rangle \rightarrow \langle C_1, \rho \rangle} \text{ sum}_1 \quad \frac{}{\langle C_1 + C_2, \rho \rangle \rightarrow \langle C_2, \rho \rangle} \text{ sum}_2 \\[10pt] \frac{\langle C_1, \rho \rangle \rightarrow \langle C'_1, \rho' \rangle}{\langle C_1; C_2, \rho \rangle \rightarrow \langle C'_1; C_2, \rho' \rangle} \text{ comp}_1 \quad \frac{\langle C_1, \rho \rangle \rightarrow \rho'}{\langle C_1; C_2, \rho \rangle \rightarrow \langle C_2, \rho' \rangle} \text{ comp}_2 \\[10pt] \frac{}{\langle C^*, \rho \rangle \rightarrow \langle C; C^*, \rho \rangle} \text{ star} \quad \frac{}{\langle C^*, \rho \rangle \rightarrow \rho} \text{ star}_{\text{fix}} \end{array}$$

Lemma 1.1 (Collecting and small step link). For any $C \in \text{Imp}$, $X \in 2^{\text{Env}}$

$$\langle C \rangle X = \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C, \rho \rangle \rightarrow^* \rho_t \}$$

Therefore $\langle C \rangle X = \emptyset \iff \forall \rho \in X \langle C, \rho \rangle$ does not reach a final environment ρ_t .

Proof. by induction on C :

Base case $C \equiv e$:

$\langle e \rangle X = \{ \langle e \rangle \rho \mid \rho \in X \wedge \langle e \rangle \rho \neq \perp \}$, $\forall \rho \in X. \langle e, \rho \rangle \rightarrow \langle e \rangle \rho$ if $\langle e \rangle \rho \neq \perp$ because of the expr rule

$$\langle e \rangle X = \{ \langle e \rangle \rho \mid \rho \in X \wedge \langle e \rangle \rho \neq \perp \} = \{ \rho_t \in \text{Env} \mid \rho \in X \langle e, \rho \rangle \rightarrow \rho_t \}$$

Inductive cases:

1. $C \equiv C_1 + C_2$: $\langle C_1 + C_2 \rangle X = \langle C_1 \rangle X \cup \langle C_2 \rangle X$, $\forall \rho \in X. \langle C_1 + C_2, \rho \rangle \rightarrow \langle C_1, \rho \rangle \vee \langle C_1 + C_2, \rho \rangle \rightarrow \langle C_2, \rho \rangle$ respectively according to rules sum_1 and sum_2 . By inductive hypothesis

$$\langle C_1 \rangle X = \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C_1, \rho \rangle \rightarrow^* \rho_t \} \quad \langle C_2 \rangle X = \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C_2, \rho \rangle \rightarrow^* \rho_t \}$$

Therefore

$$\begin{aligned} \langle C_1 + C_2 \rangle X &= \langle C_1 \rangle X \cup \langle C_2 \rangle X && \text{(by definition)} \\ &= \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C_1, \rho \rangle \rightarrow^* \rho_t \} \cup \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C_2, \rho \rangle \rightarrow^* \rho_t \} && \text{(by ind. hp)} \\ &= \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C_1, \rho \rangle \rightarrow^* \rho_t \vee \langle C_2, \rho \rangle \rightarrow^* \rho_t \} \\ &= \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C_1 + C_2, \rho \rangle \rightarrow^* \rho_t \} \end{aligned}$$

2. $C \equiv C_1; C_2$: $\langle C_1; C_2 \rangle X = \langle C_2 \rangle (\langle C_1 \rangle X)$. By inductive hp $\langle C_1 \rangle X = \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C_1, \rho \rangle \rightarrow^* \rho_t \} = Y$, by inductive hp again $\langle C_2 \rangle Y = \{ \rho_t \in \text{Env} \mid \rho \in Y, \langle C_2, \rho \rangle \rightarrow^* \rho_t \}$. Therefore

$$\begin{aligned} \langle C_1; C_2 \rangle X &= \langle C_2 \rangle (\langle C_1 \rangle X) && \text{(by definition)} \\ &= \{ \rho_t \in \text{Env} \mid \rho_x \in \{ \rho_x \mid \rho \in X, \langle C_1, \rho \rangle \rightarrow^* \rho_x \}, \langle C_2, \rho_x \rangle \rightarrow^* \rho_t \} && \text{(by ind. hp)} \\ &= \{ \rho_t \in \text{Env} \mid \rho \in X \langle C_1, \rho \rangle \rightarrow^* \rho_x \wedge \langle C_2, \rho_x \rangle \rightarrow^* \rho_t \} && \text{(by definition)} \\ &= \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C_1; C_2, \rho \rangle \rightarrow^* \rho_t \} \end{aligned}$$

3. $C \equiv C^*$: $\langle C^* \rangle X = \cup_{i \in \mathbb{N}} \langle C \rangle^i X$

$$\begin{aligned} \langle C^* \rangle X &= X \cup \langle C \rangle X \cup \langle C \rangle^2 X \cup \dots && \text{(by definition)} \\ &= X \cup \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C, \rho \rangle \rightarrow^* \rho_t \} \cup \{ \rho_t \in 2^{\text{Env}} \mid \rho \in X, \langle C; C, \rho \rangle \rightarrow^* \rho_t \} \cup \dots && \text{(by ind. hp)} \\ &= \cup_{i \in \mathbb{N}} \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C^i, \rho \rangle \rightarrow^* \rho_t \} \\ &= \{ \rho_t \in \text{Env} \mid \rho \in X, \forall i \in \mathbb{N} \langle C^i, \rho \rangle \rightarrow^* \rho_t \} \\ &= \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C^*, \rho \rangle \rightarrow^* \rho_t \} \end{aligned}$$

□

We can notice that $\langle C \rangle X = \emptyset \iff \nexists \rho_t \in \text{Env}, \rho \in X \mid \langle C, \rho \rangle \rightarrow^* \rho_t$.

1.2.1 Functions in Imp

Since we're usually dealing with a finite number of free variables in our programs, we can without loss of generality refer to (input) variables as x_n with $n \in \mathbb{N}$. Therefore the collections of states $X \in 2^{\text{Env}}$ will look like

$$[x_1 \mapsto v_1, x_2 \mapsto v_2, \dots, x_n \mapsto v_n, y \mapsto v_y, z \mapsto v_z, \dots]$$

(since we're interested in finite programs, we can have only a finite set of free variables per program).

Notation 1.1 (Program input). Let $C \in \text{Imp}$ be a program, $(a_1, \dots, a_k) \in \mathbb{N}^k$ be a sequence of natural numbers. We indicate the sequence of \rightarrow relations starting from the configuration $\langle C, [x_1 \mapsto a_1, \dots, x_k \mapsto a_k] \rangle$ as

$$C(a_1, \dots, a_k)$$

Notation 1.2 (Program output). We say

$$C(a_1, \dots, a_k) \downarrow b \iff \exists \langle C, [x_1 \mapsto a_1, \dots, x_k \mapsto a_k] \rangle \rightarrow^* \rho_t \text{ s.t. } \rho_t(y) = b$$

In this sense we're considering the variable y as an output register for the program.

Observation 1.1. notice that this means, by lemma 1.1 that

$$C(a_1, \dots, a_k) \downarrow b \iff \exists \rho_t \in \langle C \rangle \{ [x_1 \mapsto a_1, \dots, x_k \mapsto a_k] \} . \rho_t(y) = b$$

Notation 1.3 (Program termination). We'll also write

$$C(a_1, \dots, a_k) \downarrow \iff \langle C \rangle \{ [x_1 \mapsto a_1, \dots, x_k \mapsto a_k] \} \neq \emptyset$$

Definition 1.7 (Imp computability). let $f : \mathbb{N}^k \rightarrow \mathbb{N}$ be a function. f is Imp computable if

$$\begin{aligned} & \exists C \in \text{Imp} \mid \forall (a_1, \dots, a_k) \in \mathbb{N}^k \wedge b \in \mathbb{N} \\ & C(a_1, \dots, a_k) \downarrow b \iff (a_1, \dots, a_k) \in \text{dom}(f) \wedge f(a_1, \dots, a_k) = b \end{aligned}$$

We argue that the class of function computed by Imp is the same as the set of partially recursive functions $\mathbb{N} \xrightarrow{r} \mathbb{N}$ (as defined in [Cut80]). To do that we have to prove that it contains the zero, successor and projection functions and is closed under composition, primitive recursion and unbounded minimization.

Lemma 1.2 (Imp functions richness). *The class of Imp-computable function is rich.*

Proof. We'll proceed by proving that Imp has each and every one of the basic functions (zero, successor, projection).

- The zero function:

$$\begin{aligned} z : \mathbb{N}^k & \rightarrow \mathbb{N} \\ (x_1, \dots, x_k) & \mapsto 0 \end{aligned}$$

is Imp-computable:

$$z(a_1, \dots, a_k) \triangleq y := 0$$

- The successor function

$$\begin{aligned} s : \mathbb{N} & \rightarrow \mathbb{N} \\ x_1 & \mapsto x_1 + 1 \end{aligned}$$

is Imp-computable:

$$s(a_1) \triangleq y := x_1 + 1$$

- The projection function

$$U_i^k : \mathbb{N}^k \rightarrow \mathbb{N}$$

$$(x_1, \dots, x_k) \mapsto x_i$$

is Imp-computable:

$$U_i^k(a_1, \dots, a_k) \triangleq y := x_i + 0$$

We'll then prove that it is closed under composition, primitive recursion and unbounded minimization.

Lemma 1.3. *let $f : \mathbb{N}^k \rightarrow \mathbb{N}$, $g_1, \dots, g_k : \mathbb{N}^n \rightarrow \mathbb{N}$ and consider the composition*

$$h : \mathbb{N}^k \rightarrow \mathbb{N}$$

$$\vec{x} \mapsto f(g_1(\vec{x}), \dots, g_k(\vec{x}))$$

h is Imp-computable.

Proof. Since by hp $f, g_n \forall n \in [1, k]$ are computable, we'll consider their programs $F, G_n \forall n \in [1, k]$. Now consider the program

$$\begin{aligned} &G_1(\vec{x}); \\ &y_1 := y + 0; \\ &G_2(\vec{x}); \\ &y_2 := y + 0; \\ &\dots; \\ &G_k(\vec{x}); \\ &y_k := y + 0; \\ &F(y_1, y_2, \dots, y_k); \end{aligned}$$

Which is exactly h . Therefore Imp is closed under generalised composition. \square

Lemma 1.4. *Given $f : \mathbb{N}^k \rightarrow \mathbb{N}$ and $g : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$ Imp computable, we argue that $h : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$*

$$\begin{cases} h(\vec{x}, 0) = f(\vec{x}) \\ h(\vec{x}, y + 1) = g(\vec{x}, y, h(\vec{x}, y)) \end{cases}$$

defined through primitive recursion is Imp-computable.

Proof. We want a program to compute $h : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$. By hypothesis we have programs F, G to compute respectively $f : \mathbb{N}^k \rightarrow \mathbb{N}$ and $g : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$. Consider the program $H(\vec{x}, x_{k+1})$:

$$\begin{aligned} &s := 0; \\ &F(\vec{x}); \\ &(x_{k+1} \notin [0, 0]; G(\vec{x}, s, y); s := s + 1; x_{k+1} := x_{k+1} - 1)^*; \\ &x_{k+1} \in [0, 0]; \end{aligned}$$

which computes exactly h . Therefore Imp is closed under primitive recursion. \square

Lemma 1.5. *Let $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ be a Imp-computable function. Then the function $h : \mathbb{N}^k \rightarrow \mathbb{N}$ defined through unbounded minimization*

$$h(\vec{x}) = \mu y. f(\vec{x}, y) = \begin{cases} \text{least } z \text{ s.t.} & \begin{cases} f(\vec{x}, z) = 0 \\ f(\vec{x}, z) \downarrow & f(\vec{x}, z') \neq 0 \quad \forall z < z' \end{cases} \\ \uparrow & \text{otherwise} \end{cases} \quad (1.1)$$

is Imp-computable.

Proof. Let F be the program for the computable function f with ariety $k + 1$, $\vec{x} = (x_1, x_2, \dots, x_k)$. Consider the program $H(\vec{x})$

$$\begin{aligned}
& z := 0; \\
& F(\vec{x}, z); \\
& (y \notin [0, 0]; z := z + 1; F(\vec{x}, z))^*; \\
& y \in [0, 0]; \\
& y := z + 0;
\end{aligned}$$

Which outputs the least z s.t. $F(\vec{x}, z) \downarrow 0$, and loops forever otherwise. Imp is therefore closed under bounded minimalization. \square

Since has the zero function, the successor function, the projections function and is closed under composition, primitive recursion and unbounded minimalization, the class of Imp-computable functions is rich. \square

Since it is rich and $\mathbb{N} \xrightarrow{r} \mathbb{N}$ is the least class of rich functions, $\mathbb{N} \xrightarrow{r} \mathbb{N} \subseteq \text{Imp}_f$ holds. Therefore we can say

$$f \in \mathbb{N}^k \xrightarrow{r} \mathbb{N} \Rightarrow \exists C \in \text{Imp} \mid C(a_1, \dots, a_k) \downarrow b \iff f(a_1, \dots, a_k) \downarrow b$$

From this we get a couple of facts that derive from well known computability results:

- deciding weather $\langle C \rangle X \neq \emptyset$ (i.e., $C(a_1, \dots, a_k) \downarrow$) is the same as deciding $x \in \text{dom}(f)$ for some $f \in \mathbb{N}^k \xrightarrow{r} \mathbb{N}$, which is undecidable (from the input problem in [Cut80, p. 104])
- dually, deciding weather $\langle C \rangle X = \emptyset$ (i.e., $C(a_1, \dots, a_k) \uparrow$) is also undecidable. The set of functions $f \in \mathbb{N}^k \xrightarrow{r} \mathbb{N}$ s.t. $f(x) \uparrow \forall x \in \mathbb{N}^k$ is not trivial and saturated, therefore it is not recursive (by Rice's theorem [Ric53]).

1.3 Deciding invariant finiteness

Lemma 1.6. *If $C \in \text{Imp}$ where the $*$ operator does not appear, and a finite $X \in 2^{\text{env}}$ then*

$$\langle C \rangle X \text{ is finite}$$

Proof. By induction on the program C :

Base case:

$C \equiv e$, therefore $\langle e \rangle X = \{\langle e \rangle \rho \mid \rho \in X, \langle e \rangle \rho \neq \perp\}$, which is finite, since X is finite.

Inductive cases:

1. $C \equiv C_1 + C_2$, therefore $\langle C_1 + C_2 \rangle X = \langle C_1 \rangle X \cup \langle C_2 \rangle X$. By inductive hypothesis, both $\langle C_1 \rangle X, \langle C_2 \rangle X$ are finite, as they're sub expressions of C . Since the union of finite sets is finite, $\langle C_1 + C_2 \rangle X$ is finite.
2. $C \equiv C_1; C_2$, therefore $\langle C_1; C_2 \rangle X = \langle C_2 \rangle (\langle C_1 \rangle X)$. By inductive hypothesis $\langle C_1 \rangle X = Y$ is finite. Again by inductive hypothesis $\langle C_2 \rangle Y$ is finite.

\square

Lemma 1.7. *Given $C \in \text{Imp}$ where the $*$ operator does not appear, and a finite $X \in 2^{\text{env}}$, the predicate " $\langle C^* \rangle X$ is finite" is undecidable.*

Proof. Suppose we can decide weather $\langle C^* \rangle X$ is finite or infinite. We'll show that in both cases we can decide weather $C^*(a_1, \dots, a_k) \downarrow$ or $C^*(a_1, \dots, a_k) \uparrow$, which we already show were undecidable statements.

- Suppose we can decide that $\langle C^* \rangle X$ is infinite for some $C \in \text{Imp}$ and $X \in 2^{\text{Env}}$. Since $\langle C^* \rangle X = \bigcup_{i \in \mathbb{N}} \langle C \rangle^i X$, $\forall i \in \mathbb{N} \langle C \rangle^i X \equiv \underbrace{\langle C; C; \dots; C \rangle}_{i \text{ times}} X$ is finite because of lemma 1.6. The

only way we could end up with an infinite amount of states is by resulting in an infinite amount of different collections of environments for each C application. In other words $\nexists i, j \in \mathbb{N} \mid \langle C \rangle^i X = \langle C \rangle^{i+j} X$ and therefore $\forall i, j \in \mathbb{N} \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C^i, \rho \rangle \rightarrow^* \rho_t \} \neq \{ \rho_t \in \text{Env} \mid \rho \in X, \langle C^{i+j}, \rho \rangle \rightarrow^* \rho_t \}$. Therefore $C^*(a_1, \dots, a_k) \uparrow$.

- If we know instead that $\langle C^* \rangle X$ is finite, we have 2 subcases:
 1. $\exists i \in \mathbb{N} \mid \langle C \rangle^i X = \langle C \rangle^{i+1} X$, which means that by kleene iteration we reach the lub after a finite amount of applications of C , therefore $C^*(a_1, \dots, a_k) \downarrow$.
 2. $\exists i, j \in \mathbb{N}, j > 1 \mid \langle C \rangle^i X = \langle C \rangle^{i+j} X$ which means that by kleene iteration we do not reach the lub after a finite amount of iterations, but we cycle through a finite amount of different collections of states X_1, X_2, \dots, X_k :

$$\langle C \rangle^i X \neq \langle C \rangle^{i+1} X \wedge \langle C \rangle^i X = \langle C \rangle^{i+j} X$$

in this case $C(a_1, \dots, a_k) \uparrow$.

Either way, we can decide whether $C^*(a_1, \dots, a_k) \downarrow$ or $C^*(a_1, \dots, a_k) \uparrow$ which are two undecidable statements.

□

Chapter 2

Intervals

2.1 Interval Analysis

We define *interval analysis* of the above language Imp in a standard way, taking the best correct approximations (bca) for the basic expressions in Exp .

Definition 2.1 (Integer intervals). We call

$$\text{Int} \triangleq \{[a, b] \mid a \in \mathbb{Z} \cup \{-\infty\} \wedge b \in \mathbb{Z} \cup \{+\infty\} \wedge a \leq b\} \cup \{\perp\}$$

set of integer intervals.

Definition 2.2 (Concretization map). We define the *concretization map* $\gamma : \text{Int} \rightarrow 2^{\mathbb{Z}}$ as

$$\begin{aligned} \gamma([a, b]) &\triangleq \{x \in \mathbb{Z} \mid a \leq x \leq b\} \\ \gamma(\perp) &\triangleq \emptyset \end{aligned}$$

Observation 2.1. $\langle \text{Int}, \sqsubseteq \rangle$ is a complete lattice where for all $I, J \in \text{Int}$, $I \sqsubseteq J$ iff $\gamma(I) \subseteq \gamma(J)$.

Definition 2.3 (Abstract integer domain). Let $\text{Int}_* \triangleq \text{Int} \setminus \{\perp\}$. The abstract domain \mathbb{A} for program analysis is the variable-wise lifting of Int_* :

$$\mathbb{A} \triangleq (\text{Var} \rightarrow \text{Int}_*) \cup \{\perp\}$$

where the intervals for a given variable are always nonempty, while \perp represents the empty set of environments. Thus, the corresponding concretization is defined as follows:

Definition 2.4 (Interval concretization). We define the *concretization map* for the abstract domain \mathbb{A} $\gamma_{\text{Int}} : \mathbb{A} \rightarrow 2^{\text{Env}}$ as

$$\begin{aligned} \gamma_{\text{Int}}(\perp) &\triangleq \emptyset \\ \forall \eta \neq \perp \quad \gamma_{\text{Int}}(\eta) &\triangleq \{\rho \in \text{Env} \mid \forall x \in \text{Var} \ \rho(x) \in \gamma(\eta(x))\} \end{aligned}$$

Observation 2.2. If we consider the ordering \sqsubseteq on \mathbb{A} s.t.

$$\forall \eta, \vartheta \in \mathbb{A} \quad \eta \sqsubseteq \vartheta \iff \gamma_{\text{Int}}(\eta) \subseteq \gamma_{\text{Int}}(\vartheta)$$

then $\langle \mathbb{A}, \sqsubseteq \rangle$ is a complete lattice.

Definition 2.5 (Interval sharpening). For a nonempty interval $[a, b] \in \text{Int}$ and $c \in \mathbb{Z}$, we define two operations raising \uparrow the lower bound to c and lowering \downarrow the upper bound to c , respectively:

$$\begin{aligned} [a, b] \uparrow c &\triangleq \begin{cases} [\max\{a, c\}, b] & \text{if } c \leq b \\ \perp & \text{if } c > b \end{cases} \\ [a, b] \downarrow c &\triangleq \begin{cases} [a, \min\{b, c\}] & \text{if } c \geq a \\ \perp & \text{if } c < a \end{cases} \end{aligned}$$

Observe that $\max([a, b] \downarrow c) \leq c$ always holds. □

Definition 2.6 (Interval addition and subtraction). For a nonempty interval $[a, b] \in \text{Int}$ and $c \in \mathbb{Z}$ define $[a, b] \pm c \triangleq [a \pm c, b \pm c]$ (recall that $\pm\infty + c = \pm\infty - c = \pm\infty$). \square

Observe that for every interval $[a, b] \in \text{Int}$ and $c \in \mathbb{Z}$

$$\max([a, b] \uparrow c) \leq b \quad \text{and} \quad \max([a, b] \downarrow c) \leq c$$

that trivially holds by defining $\max(\perp) \triangleq 0$ (i.e., 0 is the maximum of an empty interval).

The *interval semantics* of Imp is defined as the strict (i.e., preserving \perp) extension of the following function $\llbracket \cdot \rrbracket : \text{exp} \cup \text{Imp} \rightarrow \mathbb{A} \rightarrow \mathbb{A}$. For all $\eta : \text{Var} \rightarrow \text{Int}_*$,

$$\begin{aligned} \llbracket \mathbf{x} \in S \rrbracket \eta &\triangleq \begin{cases} \eta[\mathbf{x} \mapsto \eta(\mathbf{x}) \cap \text{Int}(S)] & \text{if } \eta(\mathbf{x}) \cap \text{Int}(S) \neq \perp \\ \perp & \text{otherwise} \end{cases} \\ \llbracket \mathbf{x} \in [a, b] \rrbracket \eta &\triangleq \begin{cases} \eta[\mathbf{x} \mapsto \eta(\mathbf{x}) \cap [a, b]] & \text{if } \eta(\mathbf{x}) \cap [a, b] \neq \perp \\ \perp & \text{otherwise} \end{cases} \\ \llbracket \mathbf{x} \leq k \rrbracket \eta &\triangleq \begin{cases} \eta[\mathbf{x} \mapsto \eta(\mathbf{x}) \downarrow k] & \text{if } \eta(\mathbf{x}) \downarrow k \neq \perp \\ \perp & \text{otherwise} \end{cases} \\ \llbracket \mathbf{x} > k \rrbracket \eta &\triangleq \begin{cases} \eta[\mathbf{x} \mapsto \eta(\mathbf{x}) \uparrow k + 1] & \text{if } \eta(\mathbf{x}) \downarrow k \neq \perp \\ \perp & \text{otherwise} \end{cases} \\ \llbracket \text{true} \rrbracket \eta &\triangleq \eta \\ \llbracket \text{false} \rrbracket \eta &\triangleq \perp \\ \llbracket \mathbf{x} := k \rrbracket \eta &\triangleq \eta[\mathbf{x} \mapsto [k, k]] \\ \llbracket \mathbf{x} := \mathbf{y} + k \rrbracket \eta &\triangleq \eta[\mathbf{x} \mapsto \eta(\mathbf{y}) + k] \\ \llbracket \mathbf{x} := \mathbf{x} - k \rrbracket \eta &\triangleq \eta[\mathbf{x} \mapsto \eta(\mathbf{y}) - k] \\ \llbracket \mathbf{C}_1 + \mathbf{C}_2 \rrbracket \eta &\triangleq \llbracket \mathbf{C}_1 \rrbracket \eta \sqcup \llbracket \mathbf{C}_2 \rrbracket \eta \\ \llbracket \mathbf{C}_1; \mathbf{C}_2 \rrbracket \eta &\triangleq \llbracket \mathbf{C}_2 \rrbracket (\llbracket \mathbf{C}_1 \rrbracket \eta) \\ \llbracket \mathbf{C}^* \rrbracket \eta &\triangleq \bigsqcup_{i \in \mathbb{N}} \llbracket \mathbf{C} \rrbracket^i(\eta) \\ \llbracket \text{fix}(\mathbf{C}) \rrbracket \eta &\triangleq \text{lfp} \lambda \mu. (\eta \sqcup \llbracket \mathbf{C} \rrbracket \mu) \end{aligned}$$

The semantics is well-defined, because of the following lemma:

Lemma 2.1. *for all $\mathbf{C} \in \text{Imp}$,*

$$\llbracket \mathbf{C} \rrbracket : \mathbb{A} \rightarrow \mathbb{A}$$

is monotone.

Proof. What we have to prove is that given $\eta, \vartheta \in \mathbb{A}$, with $\eta \sqsubseteq \vartheta$ then $\forall \mathbf{C} \in \text{Imp} \llbracket \mathbf{C} \rrbracket \eta \sqsubseteq \llbracket \mathbf{C} \rrbracket \vartheta$. We'll work by induction on the grammar of \mathbf{C} :

Base cases:

We avoid cases where $\eta = \perp$ and $\llbracket \mathbf{C} \rrbracket \eta = \perp$ as $\forall \vartheta \in \mathbb{A} \perp \sqsubseteq \vartheta$ and it becomes trivially true.

- $\mathbf{C} \equiv \mathbf{x} \in S$. Then

$$\begin{aligned} \llbracket \mathbf{x} \in S \rrbracket \eta &= \eta[\mathbf{x} \mapsto \eta(\mathbf{x}) \cap \text{Int}(S)] \\ \llbracket \mathbf{x} \in S \rrbracket \vartheta &= \vartheta[\mathbf{x} \mapsto \vartheta(\mathbf{x}) \cap \text{Int}(S)] \end{aligned}$$

Since $\eta(x) \sqcap \text{Int}(S) \neq \perp$ and $\eta \sqsubseteq \vartheta$, then $\vartheta(x) \sqcap \text{Int}(S) \neq \perp$. We can see that

$$\begin{aligned}
\eta \sqsubseteq \vartheta &\iff \gamma(\eta) \subseteq \gamma(\vartheta) \\
&\iff \{x \in \mathbb{Z} \mid x \in \eta(\mathbf{x})\} \subseteq \{x \in \mathbb{Z} \mid x \in \vartheta(\mathbf{x})\} \\
&\iff \{x \in \mathbb{Z} \mid x \in \eta(\mathbf{x})\} \cap \{x \in \mathbb{Z} \mid x \in \text{Int}(S)\} \subseteq \{x \in \mathbb{Z} \mid x \in \vartheta(\mathbf{x})\} \cap \{x \in \mathbb{Z} \mid x \in \text{Int}(S)\} \\
&\iff \{x \in \mathbb{Z} \mid x \in \eta(\mathbf{x}) \wedge x \in \text{Int}(S)\} \subseteq \{x \in \mathbb{Z} \mid x \in \vartheta(\mathbf{x}) \wedge x \in \text{Int}(S)\} \\
&\iff \{x \in \mathbb{Z} \mid x \in \eta(\mathbf{x}) \sqcap \text{Int}(S)\} \subseteq \{x \in \mathbb{Z} \mid x \in \vartheta(\mathbf{x}) \sqcap \text{Int}(S)\} \\
&\iff \gamma_{\text{Int}}(\eta[x \mapsto \eta(\mathbf{x}) \sqcap \text{Int}(S)](\mathbf{x})) \subseteq \gamma_{\text{Int}}(\vartheta[x \mapsto \vartheta(\mathbf{x}) \sqcap \text{Int}(S)](\mathbf{x})) \\
&\iff \llbracket \mathbf{x} \in S \rrbracket \eta \sqsubseteq \llbracket \mathbf{x} \in S \rrbracket \vartheta
\end{aligned}$$

- for the base cases $\mathbf{x} \in [a, b]$, $\mathbf{x} \leq k$, $\mathbf{x} > k$ we can use the same proceedings;
- $C \equiv \text{true}$. Then $\llbracket \text{true} \rrbracket \eta = \eta \sqsubseteq \vartheta = \llbracket \text{true} \rrbracket \vartheta$;
- $C \equiv \text{false}$. Then $\llbracket \text{false} \rrbracket \eta = \perp \sqsubseteq \perp = \llbracket \text{false} \rrbracket \vartheta$;
- $C \equiv \mathbf{x} := k$. Then

$$\begin{aligned}
\eta \sqsubseteq \vartheta &\iff \gamma_{\text{Int}}(\eta) \subseteq \gamma_{\text{Int}}(\vartheta) \\
&\iff \{\rho \in \text{Env} \mid \forall \mathbf{x} \in \text{Var} \rho(\mathbf{x}) \in \gamma(\eta(\mathbf{x}))\} \subseteq \{\rho \in \text{Env} \mid \forall \mathbf{x} \in \text{Var} \rho(\mathbf{x}) \in \gamma(\vartheta(\mathbf{x}))\} \\
&\iff \forall \mathbf{x} \in \text{Var}, \rho \in \text{Env} \quad \rho(\mathbf{x}) \in \gamma(\eta(\mathbf{x})) \Rightarrow \rho(\mathbf{x}) \in \gamma(\vartheta(\mathbf{x}))
\end{aligned} \tag{2.1}$$

Notice that

$$\begin{aligned}
\llbracket \mathbf{x} := k \rrbracket \eta &= \eta[\mathbf{x} \mapsto [k, k]] \\
\llbracket \mathbf{x} := k \rrbracket \vartheta &= \vartheta[\mathbf{x} \mapsto [k, k]]
\end{aligned}$$

because of equation 2.1 in this case we know that $\forall \mathbf{y} \in \text{Var}, \mathbf{y} \neq \mathbf{x} \rho(\mathbf{y}) \in \gamma(\eta(\mathbf{y})) \Rightarrow \rho(\mathbf{y}) \in \gamma(\vartheta(\mathbf{y}))$. For \mathbf{x} it holds that $\rho(\mathbf{x}) \in \gamma([k, k]) \Rightarrow \rho(\mathbf{x}) \in \gamma([k, k])$ and therefore

$$\begin{aligned}
\forall \mathbf{y} \in \text{Var}, \rho \in \text{Env} \quad \rho(\mathbf{y}) \in \gamma(\eta[\mathbf{x} \mapsto [k, k]](\mathbf{y})) &\Rightarrow \rho(\mathbf{y}) \in \gamma(\vartheta[\mathbf{x} \mapsto [k, k]](\mathbf{y})) \\
&\iff \gamma_{\text{Int}}(\llbracket \mathbf{x} := k \rrbracket \eta) \subseteq \gamma_{\text{Int}}(\llbracket \mathbf{x} := k \rrbracket \vartheta) \\
&\iff \llbracket \mathbf{x} := k \rrbracket \eta \sqsubseteq \llbracket \mathbf{x} := k \rrbracket \vartheta
\end{aligned}$$

- For $C \equiv \mathbf{x} := \mathbf{y} + k$, $\mathbf{x} := \mathbf{y} - k$ the procedure is the same.

Recursive cases:

- $C \equiv C_1 + C_2$. Then

$$\begin{aligned}
\llbracket C_1 + C_2 \rrbracket \eta &= \llbracket C_1 \rrbracket \eta \sqcup \llbracket C_2 \rrbracket \eta \\
&\sqsubseteq \llbracket C_1 \rrbracket \vartheta \sqcup \llbracket C_2 \rrbracket \vartheta && \text{by inductive hp.} \\
&= \llbracket C_1 + C_2 \rrbracket \vartheta
\end{aligned}$$

- $C \equiv C_1; C_2$. Then

$$\begin{aligned}
\llbracket C_1; C_2 \rrbracket \eta &= \llbracket C_2 \rrbracket (\llbracket C_1 \rrbracket \eta) \\
\alpha = \llbracket C_1 \rrbracket \eta \sqsubseteq \llbracket C_1 \rrbracket \vartheta &= \beta && \text{by inductive hp.} \\
\llbracket C_2 \rrbracket \alpha \sqsubseteq \llbracket C_2 \rrbracket \beta &&& \text{by inductive hp.} \\
\llbracket C_2 \rrbracket (\llbracket C_1 \rrbracket \eta) \sqsubseteq \llbracket C_2 \rrbracket (\llbracket C_1 \rrbracket \vartheta) &&& \text{by substitution}
\end{aligned}$$

- C^* . Then by inductive hypothesis $\forall i \in \mathbb{N}. \llbracket C \rrbracket^i \eta \sqsubseteq \llbracket C \rrbracket^i \vartheta$, which means

$$\llbracket C^* \rrbracket \eta = \bigsqcup_{i \in \mathbb{N}} \llbracket C \rrbracket^i \eta \sqsubseteq \bigsqcup_{i \in \mathbb{N}} \vartheta = \llbracket C^* \rrbracket \vartheta.$$

□

Theorem 1 (Correctness). *For all $C \in \text{Imp}$ and $\eta \in \mathbb{A}$, $\langle C \rangle \gamma(\eta) \subseteq \gamma(\llbracket C \rrbracket \eta)$ holds.*

Proof.

□

induzione?

Remark 2.1. Let us remark that in case we were interested in studying termination of the abstract interpreter, we could assume that the input of a program will always be a finite interval in such a way that nontermination can be identified with the impossibility of converging to a finite interval for some variable. In fact, starting from an environment η which maps each variable to a finite interval, $\llbracket C \rrbracket \eta$ might be infinite on some variable when C includes a either Kleene or fix iteration which does not converge in finitely many steps.

2.1.1 Computing the interval semantics

In this section we argue that for the language `Imp` the interval abstract semantics is computable in finite time without widening.

Observe that the exact computation provides, already for our simple language, a precision which is not obtainable with (basic) widening and narrowing. In the example below the semantics maps x and y to $[0, 2]$ and $[6, 8]$ resp., while widening/narrowing to $[0, \infty]$ and $[6, \infty]$

```
x:=0;
y:=0;
while (x<=5) do
  if (y=0) then
    y=y+1;
  endif;
  if (x==0) then
    x:=y+7;
  endif;
done;
end
```

Of course, for the collecting semantics this is not the case. Already computing a finite upper bound for loop invariants when they are finite is impossible as this would allow to decide termination. In fact, if the invariant is infinite then the loop diverges. If it is finite, then the number of possible states is finite and this termination can be decided as non-termination reduces to the presence of a cycle, as shown with lemma 1.7.

Problem 2.1 (Termination of interval analysis). Given $C \in \text{Imp}$, $\eta \in \mathbb{A}$, decide: $\langle C \rangle \eta =? \top$

First, given a program, we associate each variable with a *single bound*, which captures both both an *upper bound*, for which the rough idea is that, whenever a variable is beyond that bound, the behaviour of the program with respect to that variable becomes stable and an *increment bound* which captures the largest increment or decrement that can affect a variable.

Definition 2.7 (Program bound). The *bound* associated with a command $C \in \text{Imp}$ is a natural

number, denoted $(C)^b \in \mathbb{N}$, defined inductively as follows:

$$\begin{aligned}
(x \in S)^b &\triangleq \begin{cases} \min(S) & \text{if } \max(S) = \infty \\ \max(S) & \text{if } \max(S) \in \mathbb{N} \end{cases} \\
(x \in [a, b])^b &\triangleq \begin{cases} a & \text{if } b = \infty \\ b & \text{if } b \in \mathbb{N} \end{cases} \\
(x \leq k)^b &\triangleq k \\
(x > k)^b &\triangleq k \\
(\text{true})^b &\triangleq 0 \\
(\text{false})^b &\triangleq 0 \\
(x := k)^b &\triangleq k \\
(x := y + k)^b &\triangleq k \\
(x := y - k)^b &\triangleq k \\
(C_1 + C_2)^b &\triangleq (C_1)^b + (C_2)^b \\
(C_1; C_2)^b &\triangleq (C_1)^b + (C_2)^b \\
(C^*)^b &\triangleq (|\text{vars}(C)| + 1)(C)^b
\end{aligned}$$

where $\text{vars}(C)$ denotes the set of variables occurring in C .

Definition 2.8 (Bound Environment). A bound environment (benv for short) is a total function $b : \text{Var} \rightarrow \mathbb{N}$. We define $\mathbf{bEnv} \triangleq \{b \mid b : \text{Var} \rightarrow \mathbb{N}\}$. Each command $C \in \text{Imp}$ induces a benv transformer $[C]^b : \mathbf{bEnv} \rightarrow \mathbf{bEnv}$, which is defined inductively as follows:

$$\begin{aligned}
[x \in S]^b b &\triangleq \begin{cases} b[x \mapsto b(x) + \min(S)] & \text{if } \max(S) = \infty \\ b[x \mapsto b(x) + \max(S)] & \text{if } \max(S) \in \mathbb{N} \end{cases} \\
[x := k]^b b &\triangleq b[x \mapsto b(x) + k] \\
[x := y + k]^b b &\triangleq b[x \mapsto b(x) + b(y) + k] \\
[x := y - k]^b b &\triangleq b[x \mapsto b(x) + b(y) + k] \\
[C_1 + C_2]^b b &\triangleq \lambda x. ([C_1]^b b)(x) + ([C_2]^b b)(x) \\
[C_1; C_2]^b b &\triangleq \lambda x. ([C_1]^b b)(x) + ([C_2]^b b)(x) \\
[\text{fix}(C)]^b b &\triangleq \lambda x. (|\text{vars}(C)| + 1)([C]^b b)(x)
\end{aligned}$$

where $\text{vars}(C)$ denotes the set of variables occurring in C .

Lemma 2.2. For all $C \in \text{Imp}$, $(C)^b = \sum_{x \in \text{vars}(C)} ([C]^b b_0)(x)$, with $b_0 \triangleq \lambda x. 0$.

Proof. By induction on $C \in \text{Imp}$. The base cases are clear.

$(C_1 + C_2)$:

$$\begin{aligned}
(C_1 + C_2)^b &= \\
(C_1)^b + (C_2)^b &= \quad [\text{by inductive hypothesis}] \\
\sum_{\mathbf{x} \in \text{vars}(C_1)} ([C]^b b_0)(\mathbf{x}) + \sum_{\mathbf{x} \in \text{vars}(C_2)} ([C]^b b_0)(\mathbf{x}) &= \\
\sum_{\mathbf{x} \in \text{vars}(C_1) \cap \text{vars}(C_2)} ([C_1]^b b_0)(\mathbf{x}) + ([C_2]^b b_0)(\mathbf{x}) + \\
\sum_{\mathbf{x} \in \text{vars}(C_1) \setminus \text{vars}(C_2)} ([C_1]^b b_0)(\mathbf{x}) + \\
\sum_{\mathbf{x} \in \text{vars}(C_2) \setminus \text{vars}(C_1)} ([C_2]^b b_0)(\mathbf{x}) &= \\
[C_1 + C_2]^b b_0
\end{aligned}$$

$(C_1; C_2)$: identical to $(C_1 + C_2)$.

$(\text{fix}(C))$:

$$\begin{aligned}
(\text{fix}(C))^b &= \\
|\text{vars}(C) + 1| (C)^b &= \quad [\text{by inductive hypothesis}] \\
|\text{vars}(C) + 1| \sum_{\mathbf{x} \in \text{vars}(C)} ([C]^b b_0)(\mathbf{x}) &= \\
\sum_{\mathbf{x} \in \text{vars}(C)} |\text{vars}(C) + 1| ([C]^b b_0)(\mathbf{x}) &= \\
[\text{fix}(C)]^b b_0
\end{aligned}$$

□

Chapter 3

Non relational collecting

Bibliography

- [Cut80] Nigel Cutland. *Computability: An introduction to recursive function theory*. Cambridge university press, 1980.
- [Ric53] Henry Gordon Rice. “Classes of recursively enumerable sets and their decision problems”. In: *Transactions of the American Mathematical society* 74.2 (1953), pp. 358–366.