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

Chapter 1

Framework

1.1 The Imp language

We denote by $\mathbb{Z}$ the set of integers with the usual order, extended with bottom and top elements $-\infty$ and $+\infty$, s.t. $-\infty \leqslant z \leqslant +\infty$ $\forall z \in \mathbb{Z}$. We also extend addition and subtraction by letting, for $z \in \mathbb{Z}$ $+\infty + z = +\infty - z = +\infty$ and $-\infty + z = -\infty - z = -\infty$.

We focus on the following non-deterministic language.

$$
\begin{aligned}
\mathsf{Exp} \ni \mathsf{e} ::= \quad & \mathsf{x} \in S \mid \mathsf{x} \in [a,b] \mid \mathsf{x} \leqslant k \mid \mathsf{x} > k \mid \mathsf{true} \mid \mathsf{false} \mid \\
& \mathsf{x} := k \mid \mathsf{x} := \mathsf{y} + k \mid \mathsf{x} := \mathsf{y} - k \\
\mathrm{Imp}_{\neq \star} \ni \mathsf{C} ::= \quad & \mathsf{e} \mid \mathsf{C} + \mathsf{C} \mid \mathsf{C}; \mathsf{C} \\
\mathrm{Imp} \ni \mathsf{C} ::= \quad & \mathsf{e} \mid \mathsf{C} + \mathsf{C} \mid \mathsf{C}; \mathsf{C} \mid \mathsf{C}*
\end{aligned}
$$

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

1.2 Semantics

The first building block is that of environments, mapst from the set of variables to their value.

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

$$
\mathsf{Env} \triangleq \{\rho \mid \rho : Var \to \mathbb{Z}\}
$$

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

semantics $(\![\cdot]\!) : \mathsf{Exp} \to \mathsf{Env} \to \mathsf{Env} \cup \{\bot\}$ is inductively defined as follows:

$$(\![\mathsf{x} \in S]\!)\rho \triangleq \begin{cases} \rho & \rho(\mathsf{x}) \in S \\ \bot & \text{otherwise} \end{cases}$$

$$(\![\mathsf{x} \in [a, b]]\!)\rho \triangleq \begin{cases} \rho & \rho(\mathsf{x}) \in [a, b] \\ \bot & \text{otherwise} \end{cases}$$

$$(\![\mathsf{x} \leqslant k]\!)\rho \triangleq \begin{cases} \rho & \rho(\mathsf{x}) \leqslant k \\ \bot & \text{otherwise} \end{cases}$$

$$(\![\mathsf{x} > k]\!)\rho \triangleq \begin{cases} \rho & \rho(\mathsf{x}) > k \\ \bot & \text{otherwise} \end{cases}$$

$$(\![\mathsf{true}]\!)\rho \triangleq \rho$$

$$(\![\mathsf{false}]\!)\rho \triangleq \bot$$

$$(\![\mathsf{x} := k]\!)\rho \triangleq \rho[\mathsf{x} \mapsto k]$$

$$(\![\mathsf{x} := \mathsf{y} + k]\!)\rho \triangleq \rho[\mathsf{x} \mapsto \rho(\mathsf{y}) + k]$$

$$(\![\mathsf{x} := \mathsf{y} - k]\!)\rho \triangleq \rho[\mathsf{x} \mapsto \rho(\mathsf{y}) - k]$$

The next building block is the concrete collecting semantics for the language, it associates each program in Imp to a function which, given a set of initial environments X "collects" the set of terminal states produced by executing the program from X.

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

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

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

$$\langle \cdot \rangle : \mathrm{Imp} \to \mathbb{C} \to \mathbb{C}$$

which maps each program $C \in \mathrm{Imp}$ to a total mapping over the complete lattice $\mathbb{C}$,

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

inductively defined as follows: given $X \in 2^{\mathsf{Env}}$

$$\langle \mathsf{e} \rangle X \triangleq \{ (\![\mathsf{e}]\!)\rho \mid \rho \in X, (\![\mathsf{e}]\!)\rho \neq \bot \}$$

$$\langle \mathsf{C}_1 + \mathsf{C}_2 \rangle X \triangleq \langle \mathsf{C}_1 \rangle X \cup \langle \mathsf{C}_2 \rangle X$$

$$\langle \mathsf{C}_1 ; \mathsf{C}_2 \rangle X \triangleq \langle \mathsf{C}_2 \rangle (\langle \mathsf{C}_1 \rangle X)$$

$$\langle \mathsf{C}^* \rangle X \triangleq \bigcup_{i \in \mathbb{N}} \langle \mathsf{C} \rangle^i X$$

Along with the collecting semantics we also define a one step transition relation.

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

$$\mathsf{State} \triangleq \mathrm{Imp} \times \mathsf{Env}$$

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

$$\frac{(\![e]\!)\rho \neq \bot}{\langle \mathsf{e}, \rho \rangle \rightarrow (\![e]\!)\rho} \ \text{expr}$$

$$\frac{}{\langle \mathsf{C}_1 + \mathsf{C}_2, \rho \rangle \rightarrow \langle \mathsf{C}_1, \rho \rangle} \ \text{sum}_1 \quad \frac{}{\langle \mathsf{C}_1 + \mathsf{C}_2, \rho \rangle \rightarrow \langle \mathsf{C}_2, \rho \rangle} \ \text{sum}_2$$

$$\frac{\langle \mathsf{C}_1, \rho \rangle \rightarrow \langle \mathsf{C}'_1, \rho' \rangle}{\langle \mathsf{C}_1; \mathsf{C}_2, \rho \rangle \rightarrow \langle \mathsf{C}'_1; \mathsf{C}_2, \rho' \rangle} \ \text{comp}_1 \quad \frac{\langle \mathsf{C}_1, \rho \rangle \rightarrow \rho'}{\langle \mathsf{C}_1; \mathsf{C}_2, \rho \rangle \rightarrow \langle \mathsf{C}_2, \rho' \rangle} \ \text{comp}_2$$

$$\frac{}{\langle \mathsf{C}^*, \rho \rangle \rightarrow \langle \mathsf{C}; \mathsf{C}^*, \rho \rangle} \ \text{star} \quad \frac{}{\langle \mathsf{C}^*, \rho \rangle \rightarrow \rho} \ \text{star}_{\text{fix}}$$

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

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

Therefore $\langle \mathsf{C} \rangle X = \varnothing \iff \nexists \rho_t \in \mathsf{Env}, \rho \in X$ s.t. $\langle \mathsf{C}, \rho \rangle \rightarrow^* \rho_t$.

Proof. by induction on C:

Base case $C \equiv e$**:**
$\langle \mathsf{e} \rangle X = \{ (\![e]\!)\rho \mid \rho \in X \wedge (\![e]\!)\rho \neq \bot \}, \forall \rho \in X. \langle \mathsf{e}, \rho \rangle \rightarrow (\![e]\!)\rho$ if $(\![e]\!)\rho \neq \bot$ because of the expr rule

$$\langle \mathsf{e} \rangle X = \{ (\![e]\!)\rho \mid \rho \in X \wedge (\![e]\!)\rho \neq \bot \} = \{ \rho_t \in \mathsf{Env} \mid \rho \in X \langle \mathsf{e}, \rho \rangle \rightarrow \rho_t \}$$

Inductive cases:

1. $C \equiv C_1 + C_2 : \langle \mathsf{C}_1 + \mathsf{C}_2 \rangle X = \langle \mathsf{C}_1 \rangle X \cup \langle \mathsf{C}_2 \rangle X, \forall \rho \in X. \langle \mathsf{C}_1 + \mathsf{C}_2, \rho \rangle \rightarrow \langle \mathsf{C}_1, \rho \rangle \vee \langle \mathsf{C}_1 + \mathsf{C}_2, \rho \rangle \rightarrow \langle \mathsf{C}_2, \rho \rangle$ respectively according to rules sum$_1$ and sum$_2$. By inductive hypothesis

   $$\langle \mathsf{C}_1 \rangle X = \{ \rho_t \in \mathsf{Env} \mid \rho \in X, \langle \mathsf{C}_1, \rho \rangle \rightarrow^* \rho_t \} \quad \langle \mathsf{C}_2 \rangle X = \{ \rho_t \in \mathsf{Env} \mid \rho \in X, \langle \mathsf{C}_2, \rho \rangle \rightarrow^* \rho_t \}$$

   Therefore

   $$\begin{aligned} \langle \mathsf{C}_1 + \mathsf{C}_2 \rangle X &= \langle \mathsf{C}_1 \rangle X \cup \langle \mathsf{C}_2 \rangle X & \text{(by definition)} \\ &= \{ \rho_t \in \mathsf{Env} \mid \rho \in X, \langle \mathsf{C}_1, \rho \rangle \rightarrow^* \rho_t \} \cup \{ \rho_t \in \mathsf{Env} \mid \rho \in X, \langle \mathsf{C}_2, \rho \rangle \rightarrow^* \rho_t \} & \text{(by ind. hp)} \\ &= \{ \rho_t \in \mathsf{Env} \mid \rho \in X, \langle \mathsf{C}_1, \rho \rangle \rightarrow^* \rho_t \vee \langle \mathsf{C}_2, \rho \rangle \rightarrow^* \rho_t \} \\ &= \{ \rho_t \in \mathsf{Env} \mid \rho \in X, \langle \mathsf{C}_1 + \mathsf{C}_2, \rho \rangle \rightarrow^* \rho_t \} \end{aligned}$$

2. $C \equiv C_1; C_2 : \langle \mathsf{C}_1; \mathsf{C}_2 \rangle X = \langle \mathsf{C}_2 \rangle (\langle \mathsf{C}_1 \rangle X)$. By inductive hp $\langle \mathsf{C}_1 \rangle X = \{ \rho_t \in \mathsf{Env} \mid \rho \in X, \langle \mathsf{C}_1, \rho \rangle \rightarrow^* \rho_t \} = Y$, by inductive hp again $\langle \mathsf{C}_2 \rangle Y = \{ \rho_t \in \mathsf{Env} \mid \rho \in Y, \langle \mathsf{C}_2, \rho \rangle \rightarrow^* \rho_t \}$. Therefore

   $$\begin{aligned} \langle \mathsf{C}_1; \mathsf{C}_2 \rangle X &= \langle \mathsf{C}_2 \rangle (\langle \mathsf{C}_1 \rangle X) & \text{(by definition)} \\ &= \{ \rho_t \in \mathsf{Env} \mid \rho_x \in \{ \rho_x \mid \rho \in X, \langle \mathsf{C}_1, \rho \rangle \rightarrow^* \rho_x \}, \langle \mathsf{C}_2, \rho_x \rangle \rightarrow^* \rho_t \} & \text{(by ind. hp)} \\ &= \{ \rho_t \in \mathsf{Env} \mid \rho \in X \langle \mathsf{C}_1, \rho \rangle \rightarrow^* \rho_x \wedge \langle \mathsf{C}_2, \rho_x \rangle \rightarrow^* \rho_t \} & \text{(by definition)} \\ &= \{ \rho_t \in \mathsf{Env} \mid \rho \in X. \langle \mathsf{C}_1; \mathsf{C}_2, \rho \rangle \rightarrow^* \rho_t \} \end{aligned}$$

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

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

$\square$

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

1.3   Transition system

With the set of states State, the set of environments Env and the small operational semantics $\to$ we define a transition system:

Definition 1.7 (Transition system)**.**

$$\langle \mathsf{State} \cup \mathsf{Env}, \mathsf{Env}, \to \rangle$$

is a transition system for the language Imp

Definition 1.8 (Reductions)**.** Given a program $C \in \mathsf{Imp}$ we define its reductions $\mathsf{red}(C)$ as

$$\mathsf{red}(C) = \{C' \mid \langle C, \rho \rangle \to^* \langle C', \rho' \rangle \quad \text{for some } \rho\}$$

Lemma 1.2. *Given a program* $C \in Imp$, *for* $\mathsf{red}(C)$ *it holds that*

$$\mathsf{red}(e) = \varnothing$$
$$\mathsf{red}(C_1 + C_2) = \{C_1, C_2\} \cup \mathsf{red}(C_1) \cup \mathsf{red}(C_2)$$
$$\mathsf{red}(C_1; C_2) = \{C_2\} \cup \mathsf{red}(C_2) \cup \{C'_1; C_2 \mid C'_1 \in \mathsf{red}(C_1)\}$$
$$\mathsf{red}(C^*) = \{C; C^*\} \cup \{C'; C^* \mid C' \in \mathsf{red}(C)\}$$

Proof. By definition:

- $C \equiv e$, and $\mathsf{red}(e) = \{C' \mid \langle e, \rho \rangle \to \langle C', \rho' \rangle \quad \text{for some } \rho\}$. But because of expr rule, either $\langle e, \rho \rangle \to (\!| e |\!)\rho$ (if $(\!| e |\!)\rho \neq \bot$) or $\langle e, \rho \rangle \nrightarrow$ (if $(\!| e |\!)\rho = \bot$), therefore $\nexists C' \mid \langle e, \rho \rangle \to \langle C', \rho' \rangle$ for some ρ, therefore

$$\mathsf{red}(e) = \varnothing$$

- $C \equiv C_1 + C_2$ and $\mathsf{red}(C_1 + C_2) = \{C' \mid \langle C_1 + C_2, \rho \rangle \to \langle C', \rho' \rangle$ for some $\rho\}$. Because sum_1 and sum_2 $\langle C_1 + C_2, \rho \rangle \to \langle C_1, \rho \rangle$ and $\langle C_1 + C_2, \rho \rangle \to \langle C_2, \rho \rangle$ respectively. Therefore

$$\mathsf{red}(C_1 + C_2) = \{C_1, C_2\} \cup \mathsf{red}(C_1) \cup \mathsf{red}(C_2)$$

- $C \equiv C_1; C_2$

- $C \equiv C^*$

$\square$

Lemma 1.3. $\forall C \in Imp$ $\mathsf{red}(C)$ *is finite.*

Proof. We work by induction on C:

Base case:
$C \equiv e$. Because of lemma 1.2 $\mathsf{red}(e) = \varnothing$ which is finite.

Inductive cases:

- $C \equiv C_1 + C_2$. By lemma 1.2 $\mathsf{red}(C_1 + C_2) = \{C_1, C_2\} \cup \mathsf{red}(C_1) \cup \mathsf{red}(C_2)$, which is finite as it is union of finite sets.

- $C \equiv C_1; C_2$ By lemma 1.2 $\mathsf{red}(C_1; C_2) = \{C_2\} \cup \mathsf{red}(C_2) \cup \{C'_1; C_2 \mid C'_1 \in \mathsf{red}(C_1)\}$. Both $\mathsf{red}(C_1)$ and $\mathsf{red}(C_2)$ are finite by inductive hypothesis, and therefore the union of finite sets is finite..

- $C \equiv C^*$. By lemma 1.2 $\mathsf{red}(C^*) = \{C; C^*\} \cup \{C'; C^* \mid C' \in \mathsf{red}(C)\}$. $\mathsf{red}(C)$ is finite by inductive hypothesis, and therefore the union is finite.

$\square$

1.4 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 $X \in 2^{\mathsf{Env}}$ will look like

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

(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 $C \in \mathsf{Imp}$ be a program, $(a_1, \ldots, a_k) \in \mathbb{N}^\omega$ be a sequence of natural numbers. We indicate the sequence of $\rightarrow$ relations starting from the configuration $\langle C, [x_1 \mapsto a_1, \ldots, x_k \mapsto a_k] \rangle$ as

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

Notation 1.2 (Program output). We say

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

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

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

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

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

Definition 1.9 (Imp computability). let $f : \mathbb{N}^k \to \mathbb{N}$ be a function. f is Imp computable if

$$\exists C \in \mathsf{Imp} \mid \forall (a_1, \ldots, a_k) \in \mathbb{N}^k \wedge b \in \mathbb{N}$$

$$C(a_1, \ldots, a_k) \downarrow b \iff (a_1, \ldots, a_k) \in dom(f) \wedge f(a_1, \ldots, a_k) = b$$

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 it is closed under composition, primitive recursion and unbounded minimalization.

Lemma 1.4 (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:

$$z : \mathbb{N}^k \to \mathbb{N}$$
$$(x_1, \ldots, x_k) \mapsto 0$$

  is Imp-computable:

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

- The successor function

$$s : \mathbb{N} \to \mathbb{N}$$
$$x_1 \mapsto x_1 + 1$$

  is Imp-computable:

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

- The projection function

$$U_i^k : \mathbb{N}^k \to \mathbb{N}$$
$$(x_1, \ldots, x_k) \mapsto x_i$$

  is Imp-computable:

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

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

Lemma 1.5. *let* $f : \mathbb{N}^k \to \mathbb{N}$, $g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N}$ *and consider the composition*

$$h : \mathbb{N}^k \to \mathbb{N}$$
$$\vec{x} \mapsto f(g_1(\vec{x}), \ldots, g_k(\vec{x}))$$

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

$$G_1(\vec{x});$$
$$y_1 := y + 0;$$
$$G_2(\vec{x});$$
$$y_2 := y + 0;$$
$$\ldots;$$
$$G_k(\vec{x});$$
$$y_k := y + 0;$$
$$F(y_1, y_2, \ldots, y_k);$$

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

Lemma 1.6. *Given* $f : \mathbb{N}^k \to \mathbb{N}$ *and* $g : \mathbb{N}^{k+2} \to \mathbb{N}$ *Imp computable, we argue that* $h : \mathbb{N}^{k+1} \to \mathbb{N}$

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

defined trough primitive recursion is Imp-computable.

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

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

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

Lemma 1.7. *Let* $f : \mathbb{N}^{k+1} \to \mathbb{N}$ *be a Imp-computable function. Then the function* $h : \mathbb{N}^k \to \mathbb{N}$ *defined trough unbounded minimalization*

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

is Imp-computable.

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

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

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 $\mathbb{N} \overset{r}{\hookrightarrow} \mathbb{N}$ is the least class of rich functions, $\mathbb{N} \overset{r}{\hookrightarrow} \mathbb{N} \subseteq \mathrm{Imp}_f$ holds. Therefore we can say

$$f \in \mathbb{N}^k \overset{r}{\hookrightarrow} \mathbb{N} \Rightarrow \exists C \in \mathrm{Imp} \mid C(a_1, \ldots, a_k) \downarrow b \iff f(a_1, \ldots, a_k) \downarrow b$$

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

- deciding wether $\langle \mathsf{C} \rangle X \neq \varnothing$ (i.e., $C(a_1, \ldots, a_k) \downarrow$) is the same as deciding $x \in dom(f)$ for some $f \in \mathbb{N}^k \overset{r}{\hookrightarrow} \mathbb{N}$, which is undecidable (from the input problem in [Cut80, p. 104])

- dually, deciding wether $\langle \mathsf{C} \rangle X = \varnothing$ (i.e., $C(a_1, \ldots, a_k) \uparrow$) is also undecidable. The set of functions $f \in \mathbb{N}^k \overset{r}{\hookrightarrow} \mathbb{N}$ 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]).

1.5 Deciding invariant finiteness

Lemma 1.8. *If $C \in Imp_{\neq\star}$, and a finite $X \in 2^{env}$ then*

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

Proof. By induction on the program C:

Base case:
$C \equiv e$, therefore $\langle \mathsf{e} \rangle X = \{ (\!|e|\!)\rho \mid \rho \in X, (\!|e|\!)\rho \neq \bot \}$, which is finite, since X is finite.

Inductive cases:

1. $C \equiv C_1 + C_2$, therefore $\langle \mathsf{C_1} + \mathsf{C_2} \rangle X = \langle \mathsf{C_1} \rangle X \cup \langle \mathsf{C_2} \rangle X$. By inductive hypothesis, both $\langle \mathsf{C_1} \rangle X, \langle \mathsf{C_2} \rangle X$ are finite, as they're sub expressions of C. Since the union of finite sets is finite, $\langle \mathsf{C_1} + \mathsf{C_2} \rangle X$ is finite.

2. $C \equiv C_1; C_2$, therefore $\langle \mathsf{C_1}; \mathsf{C_2} \rangle X = \langle \mathsf{C_2} \rangle (\langle \mathsf{C_1} \rangle X)$. By inductive hypothesis $\langle \mathsf{C_1} \rangle X = Y$ is finite. Again by inductive hypothesis $\langle \mathsf{C_2} \rangle Y$ is finite.

$\square$

Lemma 1.9. *Given $C \in Imp_{\neq\star}$, and a finite $X \in 2^{env}$, the predicate "$\langle \mathsf{C}^* \rangle X$ is finite" is undecidable.*

Proof. Suppose we can decide wether $\langle \mathsf{C}^* \rangle X$ is finite. We'll show that in both cases we can decide wether $C^*(a_1, \ldots, a_k) \downarrow$, which we already show to be undecidable.

- Suppose we can decide wether $\langle \mathsf{C}^* \rangle X$ is infinite for $C \in \mathrm{Imp}$ and $X \in 2^{\mathsf{Env}}$. Since $\langle \mathsf{C}^* \rangle X = \bigcup_{i \in \mathbb{N}} \langle \mathsf{C} \rangle^i X$, $\forall i \in \mathbb{N} \langle \mathsf{C} \rangle^i X \equiv \underbrace{\langle C; C; \ldots; C \rangle}_{i \text{ times}} X$ is finite because of lemma 1.8. 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 $\nexists i, j \in \mathbb{N} \mid \langle \mathsf{C} \rangle^i X = \langle \mathsf{C} \rangle^{i+j} X$ and therefore $\forall i, j \in \mathbb{N} \{ \rho_t \in \mathsf{Env} \mid \rho \in X, \langle C^i, \rho \rangle \to^* \rho_t \} \neq \{ \rho_t \in \mathsf{Env} \mid \rho \in X, \langle C^{i+j}, \rho \rangle \to^* \rho_t \}$. Therefore $C^*(a_1, \ldots, a_k) \uparrow$.

- If we know instead that $\langle \mathsf{C}^* \rangle X$ is finite, we know that $\mathsf{red}(\mathsf{C}^*)$ is finite (lemma 1.3), but we can observe that the set of states $\{ \langle \mathsf{C}', \rho' \rangle \mid \langle \mathsf{C}, \rho \rangle \to^* \langle \mathsf{C}', \rho' \rangle \}$ is

$$\mathsf{C}'; \mathsf{C}^* \times X$$

with $\mathsf{C}' \in \mathsf{red}(\mathsf{C})$ and $X \subseteq \bigcup_{\mathsf{C}'' \in \mathsf{red}(\mathsf{C})} \langle \mathsf{C}^* \rangle \langle \mathsf{C}'' \rangle \{ \rho \}$.

$\square$

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

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

set of integer intervals.

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

$$\gamma([a,b]) \triangleq \{x \in \mathbb{Z} \mid a \leqslant x \leqslant b\}$$
$$\gamma(\bot) \triangleq \varnothing$$

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

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

$$\mathbb{A} \triangleq (Var \to Int_*) \cup \{\bot\}$$

where the intervals for a given variable are always nonempty, while $\bot$ 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 $\mathbb{A}$ $\gamma_{Int} : \mathbb{A} \to 2^{\mathsf{Env}}$ as

$$\gamma_{Int}(\bot) \triangleq \varnothing$$
$$\forall \eta \neq \bot \quad \gamma_{Int}(\eta) \triangleq \{\rho \in \mathsf{Env} \mid \forall x \in Var \; \rho(x) \in \gamma(\eta(x))\}$$

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

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

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

Definition 2.5 (Interval sharpening)**.** For a nonempty interval $[a,b] \in Int$ 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:

$$[a,b] \uparrow c \triangleq \begin{cases} [\max\{a,c\},b] & \text{if } c \leqslant b \\ \bot & \text{if } c > b \end{cases}$$

$$[a,b] \downarrow c \triangleq \begin{cases} [a,\min\{b,c\}] & \text{if } c \geq a \\ \bot & \text{if } c < a \end{cases}$$

Observe that $\max([a,b] \downarrow c) \leqslant c$ always holds. $\qquad\qquad\square$

Definition 2.6 (Interval addition and subtraction). For a nonempty interval $[a, b] \in Int$ 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 $[a, b] \in Int$ and $c \in \mathbb{Z}$

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

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

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

$$\llbracket \mathsf{x} \in S \rrbracket \eta \triangleq \begin{cases} \eta[\mathsf{x} \mapsto \eta(\mathsf{x}) \sqcap Int(S)] & \text{if } \eta(\mathsf{x}) \sqcap Int(S) \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

$$\llbracket \mathsf{x} \in [a, b] \rrbracket \eta \triangleq \begin{cases} \eta[\mathsf{x} \mapsto \eta(\mathsf{x}) \sqcap [a, b]] & \text{if } \eta(\mathsf{x}) \sqcap [a, b] \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

$$\llbracket \mathsf{x} \leqslant k \rrbracket \eta \triangleq \begin{cases} \eta[\mathsf{x} \mapsto \eta(\mathsf{x}) \downarrow k] & \text{if } \eta(\mathsf{x}) \downarrow k \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

$$\llbracket \mathsf{x} > k \rrbracket \eta \triangleq \begin{cases} \eta[\mathsf{x} \mapsto \eta(\mathsf{x}) \uparrow k + 1] & \text{if } \eta(\mathsf{x}) \downarrow k \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

$$\llbracket \mathsf{true} \rrbracket \eta \triangleq \eta$$

$$\llbracket \mathsf{false} \rrbracket \eta \triangleq \bot$$

$$\llbracket \mathsf{x} := k \rrbracket \eta \triangleq \eta[\mathsf{x} \mapsto [k, k]]$$

$$\llbracket \mathsf{x} := \mathsf{y} + k \rrbracket \eta \triangleq \eta[\mathsf{x} \mapsto \eta(\mathsf{y}) + k]$$

$$\llbracket \mathsf{x} := \mathsf{x} - k \rrbracket \eta \triangleq \eta[\mathsf{x} \mapsto \eta(\mathsf{y}) - k]$$

$$\llbracket \mathsf{C}_1 + \mathsf{C}_2 \rrbracket \eta \triangleq \llbracket \mathsf{C}_1 \rrbracket \eta \sqcup \llbracket \mathsf{C}_2 \rrbracket \eta$$

$$\llbracket \mathsf{C}_1 ; \mathsf{C}_2 \rrbracket \eta \triangleq \llbracket \mathsf{C}_2 \rrbracket (\llbracket \mathsf{C}_1 \rrbracket \eta)$$

$$\llbracket \mathsf{C}^* \rrbracket \eta \triangleq \bigsqcup_{i \in \mathbb{N}} \llbracket \mathsf{C} \rrbracket^i (\eta)$$

$$\llbracket \mathsf{fix}(\mathsf{C}) \rrbracket \eta \triangleq \mathrm{lfp} \lambda \mu . (\eta \sqcup \llbracket \mathsf{C} \rrbracket \mu)$$

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

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

$$\llbracket \mathsf{C} \rrbracket : \mathbb{A} \to \mathbb{A}$$

is monotone.

Proof. What we have to proof is that given $\eta, \vartheta \in \mathbb{A}$, with $\eta \sqsubseteq \vartheta$ then $\forall C \in \mathrm{Imp} \ \llbracket C \rrbracket \eta \sqsubseteq \llbracket C \rrbracket \vartheta$. We'll work by induction on the grammar of C:
Base cases:
We avoid cases where $\eta = \bot$ and $\llbracket C \rrbracket \eta = \bot$ as $\forall \vartheta \in \mathbb{A} \ \bot \sqsubseteq \vartheta$ and it becomes trivially true.

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

$$\llbracket \mathsf{x} \in S \rrbracket \eta = \eta[x \mapsto \eta(x) \sqcap Int(S)]$$
$$\llbracket \mathsf{x} \in S \rrbracket \vartheta = \vartheta[x \mapsto \vartheta(x) \sqcap Int(S)]$$

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

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

- for the base cases $\mathbf{x} \in [a, b], \mathbf{x} \leqslant k, \mathbf{x} > k$ we can use the same proceedings;

- $\mathsf{C} \equiv \mathsf{true}$. Then $[\![\mathsf{true}]\!]\eta = \eta \sqsubseteq \vartheta = [\![\mathsf{true}]\!]\vartheta$;

- $\mathsf{C} \equiv \mathsf{false}$. Then $[\![\mathsf{false}]\!]\eta = \bot \sqsubseteq \bot = [\![\mathsf{false}]\!]\vartheta$;

- $\mathsf{C} \equiv \mathbf{x} := k$. Then

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

$$(2.1)$$

Notice that

$$
[\![\mathbf{x} := k]\!]\eta = \eta[\mathbf{x} \mapsto [k, k]]
$$
$$
[\![\mathbf{x} := k]\!]\vartheta = \vartheta[\mathbf{x} \mapsto [k, k]]
$$

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

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

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

Recusrsive cases:

- $C \equiv C_1 + C_2$. Then

$$
\begin{aligned}
[\![C_1 + C_2]\!]\eta &= [\![C_1]\!]\eta \sqcup [\![C_2]\!]\eta \\
&\sqsubseteq [\![C_1]\!]\vartheta \sqcup [\![C_2]\!]\vartheta \qquad\qquad \text{by inductive hp.} \\
&= [\![C_1 + C_2]\!]\vartheta
\end{aligned}
$$

- $C \equiv C_1; C_2$. Then

$$
\begin{aligned}
[\![C_1; C_2]\!]\eta &= [\![C_2]\!]([\![C_1]\!]\eta) \\
\alpha = [\![C_1]\!]\eta &\sqsubseteq [\![C_1]\!]\vartheta = \beta \qquad\qquad \text{by inductive hp.} \\
[\![C_2]\!]\alpha &\sqsubseteq [\![C_2]\!]\beta \qquad\qquad \text{by inductive hp.} \\
[\![C_2]\!]([\![C_1]\!]\eta) &\sqsubseteq [\![C_2]\!]([\![C_1]\!]\vartheta) \qquad\quad \text{by substitution}
\end{aligned}
$$

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

$$
[\![C^*]\!]\eta = \bigsqcup_{i \in \mathbb{N}} [\![C]\!]^i\eta \sqsubseteq \bigsqcup_{i \in \mathbb{N}} \vartheta = [\![C^*]\!]\vartheta.
$$

$\square$

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

> Sempre per induzione?

Proof. $\square$

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, $\llbracket\mathsf{C}\rrbracket\eta$ might be infinite on some variable when C includes a either Kleene or fix iteration which does not convernge 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 1.9.

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

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 $\mathsf{C} \in \mathrm{Imp}$ is a natural

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

$$(\mathsf{x} \in S)^{\mathsf{b}} \triangleq \begin{cases} \min(S) & \text{if } \max(S) = \infty \\ \max(S) & \text{if } \max(S) \in \mathbb{N} \end{cases}$$

$$(\mathsf{x} \in [a, b])^{\mathsf{b}} \triangleq \begin{cases} a & \text{if } b = \infty \\ b & \text{if } b \in \mathbb{N} \end{cases}$$

$$(\mathsf{x} \leqslant k)^{\mathsf{b}} \triangleq k$$

$$(\mathsf{x} > k)^{\mathsf{b}} \triangleq k$$

$$(\mathsf{true})^{\mathsf{b}} \triangleq 0$$

$$(\mathsf{false})^{\mathsf{b}} \triangleq 0$$

$$(\mathsf{x} := k)^{\mathsf{b}} \triangleq k$$

$$(\mathsf{x} := \mathsf{y} + k)^{\mathsf{b}} \triangleq k$$

$$(\mathsf{x} := \mathsf{y} - k)^{\mathsf{b}} \triangleq k$$

$$(\mathsf{C}_1 + \mathsf{C}_2)^{\mathsf{b}} \triangleq (\mathsf{C}_1)^{\mathsf{b}} + (\mathsf{C}_2)^{\mathsf{b}}$$

$$(\mathsf{C}_1; \mathsf{C}_2)^{\mathsf{b}} \triangleq (\mathsf{C}_1)^{\mathsf{b}} + (\mathsf{C}_2)^{\mathsf{b}}$$

$$(\mathsf{C}^*)^{\mathsf{b}} \triangleq (|vars(\mathsf{C})| + 1)(\mathsf{C})^{\mathsf{b}}$$

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

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

$$[\mathsf{x} \in S]^{\mathsf{b}}b \triangleq \begin{cases} b[\mathsf{x} \mapsto b(\mathsf{x}) + \min(S)] & \text{if } \max(S) = \infty \\ b[\mathsf{x} \mapsto b(\mathsf{x}) + \max(S)] & \text{if } \max(S) \in \mathbb{N} \end{cases}$$

$$[\mathsf{x} := k]^{\mathsf{b}}b \triangleq b[\mathsf{x} \mapsto b(\mathsf{x}) + k]$$

$$[\mathsf{x} := \mathsf{y} + k]^{\mathsf{b}}b \triangleq b[\mathsf{x} \mapsto b(\mathsf{x}) + b(\mathsf{y}) + k]$$

$$(\mathsf{x} := \mathsf{y} - k)^{\mathsf{b}}b \triangleq b[\mathsf{x} \mapsto b(\mathsf{x}) + b(\mathsf{y}) + k]$$

$$[\mathsf{C}_1 + \mathsf{C}_2]^{\mathsf{b}}b \triangleq \lambda \mathsf{x}.([\mathsf{C}_1]^{\mathsf{b}}b)(\mathsf{x}) + ([\mathsf{C}_2]^{\mathsf{b}}b)(\mathsf{x})$$

$$[\mathsf{C}_1; \mathsf{C}_2]^{\mathsf{b}}b \triangleq \lambda \mathsf{x}.([\mathsf{C}_1]^{\mathsf{b}}b)(\mathsf{x}) + ([\mathsf{C}_2]^{\mathsf{b}}b)(\mathsf{x})$$

$$[\mathsf{fix}(\mathsf{C})]^{\mathsf{b}}b \triangleq \lambda \mathsf{x}.(|vars(\mathsf{C})| + 1)([\mathsf{C}]^{\mathsf{b}}b)(\mathsf{x})$$

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

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

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

$(C_1 + C_2)$:

$$(C_1 + C_2)^b =$$
$$(C_1)^b + (C_2)^b = \quad \text{[by inductive hypothesis]}$$
$$\sum_{x \in vars(C_1)} ([C]^b b_0)(x) + \sum_{x \in vars(C_2)} ([C]^b b_0)(x) =$$
$$\sum_{x \in vars(C_1) \cap vars(C_2)} ([C_1]^b b_0)(x) + ([C_2]^b b_0)(x) +$$
$$\sum_{x \in vars(C_1) \smallsetminus vars(C_2)} ([C_1]^b b_0)(x) +$$
$$\sum_{x \in vars(C_2) \smallsetminus vars(C_1)} ([C_2]^b b_0)(x) =$$
$$[C_1 + C_2]^b b_0$$

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

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

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

$\square$

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.