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

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

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

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, $a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{+\infty\}, a \leqslant b, k \in \mathbb{Z}$ 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

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

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 $(\!|\cdot|\!) : \text{Exp} \to \text{Env} \to \text{Env} \cup \{\bot\}$ is recursively defined as follows:

$$
\begin{aligned}
(\!|x \in S|\!)\rho &\triangleq \begin{cases} \rho & \rho(x) \in S \\ \bot & \text{otherwise} \end{cases} \\
(\!|x \in [a,b]|\!)\rho &\triangleq \begin{cases} \rho & \rho(x) \in [a,b] \\ \bot & \text{otherwise} \end{cases} \\
(\!|x \leqslant k|\!)\rho &\triangleq \begin{cases} \rho & \rho(x) \leqslant k \\ \bot & \text{otherwise} \end{cases} \\
(\!|x > k|\!)\rho &\triangleq \begin{cases} \rho & \rho(x) > k \\ \bot & \text{otherwise} \end{cases} \\
(\!|\texttt{true}|\!)\rho &\triangleq \rho \\
(\!|\texttt{false}|\!)\rho &\triangleq \bot \\
(\!|x := k|\!)\rho &\triangleq \rho[x \mapsto k] \\
(\!|x := y + k|\!)\rho &\triangleq \rho[x \mapsto \rho(y) + k] \\
(\!|x := y - k|\!)\rho &\triangleq \rho[x \mapsto \rho(y) - k]
\end{aligned}
$$

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

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

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

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

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

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

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 $\to$: State $\times$ (State$\cup$Env) is a small step semantics for the Imp language. It is defined based on the following rules

$$\frac{(\!|e|\!)\rho \neq \bot}{\langle e, \rho\rangle \to (\!|e|\!)\rho}\ \text{expr}$$

$$\frac{}{\langle C_1 + C_2, \rho\rangle \to \langle C_1, \rho\rangle}\ \text{sum}_1 \quad \frac{}{\langle C_1 + C_2, \rho\rangle \to \langle C_2, \rho\rangle}\ \text{sum}_2$$

$$\frac{\langle C_1, \rho\rangle \to \langle C_1', \rho'\rangle}{\langle C_1; C_2, \rho\rangle \to \langle C_1'; C_2, \rho'\rangle}\ \text{comp}_1 \quad \frac{\langle C_1, \rho\rangle \to \rho'}{\langle C_1; C_2, \rho\rangle \to \langle C_2, \rho'\rangle}\ \text{comp}_2$$

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

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

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

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

Proof. by induction on C:

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

$$\langle e\rangle X = \{(\!|e|\!)\rho \mid \rho \in X \wedge (\!|e|\!)\rho \neq \bot\} = \{\rho_t \in \text{Env} \mid \rho \in X \langle e, \rho\rangle \to \rho_t\}$$

Inductive cases:

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

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

   Therefore

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

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 $\langle C_1\rangle X = \{\rho_t \in \text{Env} \mid \rho \in X, \langle C_1, \rho\rangle \to^* \rho_t\} = Y$, by inductive hp again $\langle C_2\rangle Y = \{\rho_t \in \text{Env} \mid \rho \in Y, \langle C_2, \rho\rangle \to^* \rho_t\}$. Therefore

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

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

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

□

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

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

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

(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 $C \in \text{Imp}$ be a program, $(a_1, \ldots, a_k) \in \mathbb{N}^\omega$ be a sequence of natural numbers. We indicate the sequence of $\to$ 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

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

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

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

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

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

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

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

We argue that the class of function computed by Imp is the same as the set of partially recursive functions $\mathbb{N} \overset{r}{\hookrightarrow} \mathbb{N}$ (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 minimalization.

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:

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

  is Imp-computable:

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

- The successor function

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

  is Imp-computable:

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

- The projection function

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

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 minimalization.

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

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

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. □

Lemma 1.4. *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 $H(\vec{x}, x_{k+1})$:

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

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

Lemma 1.5. *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*

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

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 $H(\vec{x})$

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

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

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. □

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

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

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

- deciding weather $\langle C \rangle X \neq \varnothing$ (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 weather $\langle C \rangle X = \varnothing$ (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]).

1.3 Deciding invariant finiteness

Lemma 1.6. *Given $C \in$ Imp where the * operator does not appear, and a finite $X \in 2^{env}$, the predicate "$\langle C \rangle X$ is finite" is decidable.*

Proof. By induction on the program C:

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

Inductive cases:

1. $C \equiv C_1 + C_2$, therefore $\langle C_1 + C_2 \rangle X = \langle C_1 \rangle X \cup \langle C_2 \rangle X$. 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.

□

Lemma 1.7. *Given $C \in$ Imp where the * operator does not appear, and a finite $X \in 2^{env}$, 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 $C(a_1, \ldots, a_k) \downarrow$ or $C(a_1, \ldots, a_k) \uparrow$, 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 $\langle C^* \rangle X = \bigcup_{i \in \mathbb{N}} \langle C \rangle^i X$ and $\forall i \in \mathbb{N} \langle C \rangle^i X \equiv \langle \underbrace{C; C; \ldots; C}_{i \text{ times}} \rangle X$ 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^* does not reach an upper bound starting from X in a finite amount of steps.

- 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 finte amount of applications of C, therefore C halts.
  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 trough a finite amount of different collections of states $X_1, X_2, \ldots, X_k$:

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

     in this case the program does not terminate but cycles trough a finite amount of program environments.

  Either way, we can decide weather $C(a_1, \ldots, a_k) \downarrow$ or $C(a_1, \ldots, a_k) \uparrow$ which are two undecidable statements.

□

Chapter 2

Intervals

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.