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

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

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

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

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 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 $X \in 2^{\mathrm{Env}}$

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

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:

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

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

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

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

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

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

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

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

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

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

□

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

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 $X \in 2^{\mathrm{Env}}$ 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 $C \in \mathrm{Imp}$ be a program, $(a_1, \dots, a_k) \in \mathbb{N}^\omega$ be a sequence of natural numbers. We indicate the sequence of $\to$ relations starting from the configuration $\langle C, [x_1 \mapsto a_1, \dots, x_k \mapsto a_k]\rangle$ as

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

Notation 1.2 (Program output)**.** We say

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

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

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

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

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

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

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

We argue that the class of function computed by Imp is the same as the set of partially recursive functions $\mathbb{N} \xhookrightarrow{r} \mathbb{N}$ (as defined in [Cut80]).

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

- deciding weather $\langle C\rangle X \neq \varnothing$ is the same as deciding $x \in dom(f)$ for some $f \in \mathbb{N}^k \xhookrightarrow{r} \mathbb{N}$, which is undecidable (from the input problem in [Cut80, p. 104])

1.3 Deciding invariant finiteness

Lemma 1.2. *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.

$\square$

Lemma 1.3. *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. $\square$

Chapter 2

Intervals

Chapter 3

Non relational collecting

Bibliography

[Cut80] Nigel Cutland. *Computability: An introduction to recursive function theory.* Cambridge university press, 1980.