Static analysis by abstract interpretation: a Mathematical Programming approach[1]

Eric Goubault[a,2] Stéphane Le Roux[b,3] Jeremy Leconte[c,4]
Leo Liberti[d,5] Fabrizio Marinelli[e,6]

[a] *CEA Saclay, France*

[b] *LIX, École Polytechnique, 91128 Palaiseau, France*

[c] *Dep. Info., ENS, 45 rue d'Ulm, 75005 Paris, France*

[d] *LIX, École Polytechnique, 91128 Palaiseau, France*

[e] *DIIGA, Univ. Politecnica delle Marche, Ancona, Italy*

Abstract

Static analysis of a computer program by abstract interpretation helps prove behavioural properties of the program. Programs are defined by means of a forward collecting semantics function relating the values of the program variables during the execution of the program. The least fixed point of the semantics function is a program invariants providing useful information about the program's behaviour. Mathematical Programming is a formal language for describing and solving optimization problems expressed in very general terms. This paper establishes a link between the two disciplines by providing a mathematical program that models the problem of finding the least fixed point of a semantics function. Although we limit the discussion to integer affine arithmetic semantics in the interval domain, the flexibility and power of mathematical programming tools have the potential for enriching static analysis considerably.

Keywords: Guaranteed smallest code invariant, constraints, bilinear MINLP, policy iteration, branch-and-bound.

1 Introduction

Static Analysis by Abstract Interpretation (SAAI) was introduced by Cousot and Cousot in [9] and [10], and further developed, *e.g.*, in [11]. It is widely used

[1] We thank David Monniaux and Nicolas Halbwachs for many enlightening discussions and precious suggestions. This work was partially supported by grants: Île-de-France research council (post-doctoral fellowship), System@tic consortium ("EDONA" project), ANR 07-JCJC-0151 "Ars", ANR 08-SEGI-023 "Asopt", Digiteo Emergence "Paso".

[2] Email: eric.goubault@cea.fr

[3] Email: leroux@lix.polytechnique.fr

[4] Email: jeremy.leconte1@ens.fr

[5] Email: liberti@lix.polytechnique.fr

[6] Email: marinelli@diiga.univpm.it

in static analysis of imperative programs to approximate the behaviour of a program, for instance in terms of its variable environments. Given a program, one builds a forward collecting semantics function expressing statically how the environments at a given control point depend dynamically on other control points. This function has a least fixed point (lfp), which is the "best" information that the function may give about the program. Usual methods to compute the lfp range from increasing sequences of under-approximations (relying on Kleene fixed point theorem), decreasing sequences of over-approximations (relying on Tarski fixed point theorem), or both methods combined (relying on widening). The Policy Iteration (PI) method was introduced on the interval domain in [7], further developed in [1] and extended to other (relational) domains in [12,2]. PI computes the lfp when the semantics function is non-expansive in the sup norm, and a fixed point otherwise. Another PI method on intervals was described in [14] and later generalized to relational domains in [15].

Computing the lfp of the semantics function is quite naturally an optimization problem. Mathematical Programming (MP) is a declarative language that describes the solution of very general optimization problems [26]. An MP consists of a set of parameters (encoding the problem input prior to the solution process), a set of decision variables $x \in \mathbb{R}^n$ (encoding the problem output after the solution process), an objective function $f : \mathbb{R}^n \to \mathbb{R}$, a set of equality and/or inequality constraints $g(x) \leq 0$ with $g : \mathbb{R}^n \to \mathbb{R}^m$, a set of variable bounds $x^L \leq x \leq x^U$ and a set of integrality constraints $\forall j \in Z\ x_j \in \mathbb{Z}$ [19]. MPs are categorised according to the nature of the solution as: Linear Programs (LPs), Nonlinear Programs (NLPs), Mixed-Integer Linear Programs (MILPs), Mixed-Integer Nonlinear Programs (MINLPs), each category having dedicated solution algorithms.

We study the following decision problem.

> Static Analysis by Abstract Interpretation Problem (SAAIP). Given a program written in the language $\mathbb{P}$ (defined in Sect. 2) does its semantics function (defined in Sect. 3.1) have a finite lfp?

SAAIP is actually a problem schema, because it can be parametrized by the type of abstraction used to overapproximate the concrete program semantics. This paper aims to establish a strong link between SAAI and MP by formalizing the search for the lfp by means of a MP formulation. When the semantics function only includes integer convex arithmetic, the MP turns out to be a MINLP with convex objective and constraints, which can always be solved to optimality in worst-case exponential time [4]. For semantics functions including continuous and/or nonconvex arithmetic, the resulting MINLP can be solved to ε-approximation using the spatial Branch-and-Bound (sBB) algorithm [3]. The MP standard toolbox also includes several practically ef-

ficient heuristic methods [5,21] which find non-optimal but feasible solutions: in the present setting, these correspond to fixed points without guarantee of minimality, which may provide useful information about the program. The flexibility of MP can hardly be underestimated: variable relations, for example, simply give rise to additional constraints which can just be adjoined to the current MP formulation.

We set the framework by exemplifying the use of MP in SAAI limited to a very classical setting: interval domains with integer affine arithmetic. Although a particular case of the corresponding SAAIP was recently shown to be in **P** [13], whereas our MP is solved by a worst-case exponential time Branch-and-Bound (BB) algorithm, one of the restrictions of the polynomial algorithm proposed in [13] is that all intersections must involve a constant interval, whereas our MP need not necessarily be restricted in this sense. In other words, the MP can naturally take into account variable relations arising from test conditions. We define an imperative programming language (Sect. 2) and its forward-collecting interval domain semantics function (Sect. 3) inductively. This enables the inductive definition of the MP (Sect. 4), and the proof that the semantics function has a finite lfp if and only if the MP has a solution (Sect. 5). We also test the practical applicability of the proposed methodology using the well-known CPLEX solver [17] (Sect. 7).

We remark that MP techniques were sometimes used in SAAI [23,8,6,14]; more precisely, LP technology was used as an operator within two extensions of PI-type algorithms to relational domains [12,15]. However, to the best of our knowledge, modelling fixed point equations by means of MP is new.

2 A basic programming language

The programming language $\mathbb{P}$ is defined inductively below. Its arithmetic expressions involve constants in $\mathcal{C}$ (including the integers and included in the real numbers), variables in $\mathcal{V} = \{v_1, \ldots, v_n\}$, multiplications of a constant by a variable, and additions of two variables. Programs in $\mathbb{P}$ and instructions in $\mathbb{I}$ are defined by (mutual) induction. There are only four basic instructions, namely an instruction that does nothing, the classic assignment, if-then-else branching, and the while loop.

$$
\begin{array}{rcl}
\mathbb{E} & ::= & \mathcal{C} \mid \mathcal{V} \mid \mathcal{C} * \mathbb{E} \mid \mathbb{E} + \mathbb{E} \\
\mathbb{T} & ::= & \mathcal{C} \leq \mathcal{V} \mid \mathcal{V} \leq \mathcal{C} \mid \mathcal{C} < \mathcal{V} \mid \mathcal{V} < \mathcal{C} \mid \mathcal{V} \leq \mathcal{V} \mid \mathcal{V} < \mathcal{V} \\
\mathbb{I} & ::= & \texttt{skip} \mid \mathcal{V} \leftarrow \mathbb{E} \mid \texttt{if}\ \mathbb{T}\ \{\mathbb{P}\}\ \{\mathbb{P}\} \mid \texttt{while}\ \mathbb{T}\ \{\mathbb{P}\} \\
\mathbb{P} & ::= & \mathbb{I} \mid \mathbb{I}\,\mathbb{P}
\end{array}
$$

We remark that $\mathbb{P}$-programs are structurally finite objects. The language $\mathbb{P}$ is not convenient for actual, large-scale programming but it allows simulating

other classical branching and looping, complex tests and so on. Furthermore, its simple definition keeps the proofs and explanations at a reasonable complexity level.

Control points. Although the notion of control point is not needed here, it may help intuition. In an alternative definition of the programming language above, the control points could lie in the places corresponding to the stars below.

$$\begin{array}{rcl} \mathbb{I} & ::= & \texttt{skip}\star \quad | \quad \mathcal{V} \leftarrow \mathbb{E}\star \quad | \\ & & \texttt{if}\ \mathbb{T}\,\{\star\mathbb{P}\}\,\{\star\mathbb{P}\}\star \quad | \quad \texttt{while}\ \star\ \mathbb{T}\,\{\star\mathbb{P}\}\star \end{array}$$

Program size. The notion of size will help define functions involving programs and instructions, *e.g.* semantics functions and corresponding MPs. The size of programs and instructions is defined below (definitions on the right). The size of a program corresponds to the number of control points that one may want to put in the program (definitions in the center).

$$\begin{array}{rcccl} |I\,P| & = & |I\,P| & \stackrel{\text{def}}{=} & |I| + |P| \\ |\texttt{skip}| & = & |\texttt{skip}\ \star| & \stackrel{\text{def}}{=} & 1 \\ |v_i \leftarrow \mathsf{expr}| & = & |v_i \leftarrow \mathsf{expr}\ \star| & \stackrel{\text{def}}{=} & 1 \\ |\texttt{if}\ \mathsf{test}\,\{P\}\,\{Q\}| & = & |\texttt{if}\ \mathsf{test}\,\{\star P\}\,\{\star Q\}\ \star| & \stackrel{\text{def}}{=} & |P| + |Q| + 3 \\ |\texttt{while}\ \mathsf{test}\,\{P\}| & = & |\texttt{while}\ \star\ \mathsf{test}\,\{\star P\}\ \star| & \stackrel{\text{def}}{=} & |P| + 3 \end{array}$$

3 Abstract interpretation

This paper uses a well-known lattice already used in [10] for abstract interpretation: a subset of $\mathcal{C}^n$ is abstracted into (i.e. approximated by) the smallest Cartesian box including the subset. When dealing with environments of program variables, let $\mathbb{B}$ be the set of the (environment) boxes: $\mathbb{B}$ contains the empty set and the Cartesian products $I_1 \times \cdots \times I_n$ where the I_i are non-empty intervals in $\mathcal{C}$. Moreover, let I_i^L (resp. I_i^U) be the lower (resp. upper) endpoint of interval I_i. We slightly abuse notation of Cartesian products and their elements for readability purposes. The following notation is used to modify a box along one given dimension.

Definition 3.1 *Let b be in $\mathbb{B}$ and z be an interval in $\mathcal{C}$. $b[i \leftarrow z]$ stands for $b_1 \times \cdots \times b_{i-1} \times z \times b_{i+1} \times \cdots \times b_n$ where b_i is the i-th projection of b.*

The following functions are shorthands that help relate box-based approximations of variable environments before and after branching according to a test.

Definition 3.2 *Below* T *and* F *are functions that are typed in* $\mathbb{B} \times \mathbb{T} \to \mathbb{B}$.

$$\begin{array}{llll}
\mathsf{T}(b, c \leq v_i) & \stackrel{\text{def}}{=} & b[i \leftarrow b_i \cap [c, +\infty)\,] & \mathsf{F}(b, v_i < c) \stackrel{\text{def}}{=} b[i \leftarrow b_i \cap [c, +\infty)\,] \\
\mathsf{T}(b, v_i \leq c) & \stackrel{\text{def}}{=} & b[i \leftarrow b_i \cap (-\infty, c]\,] & \mathsf{F}(b, c < v_i) \stackrel{\text{def}}{=} b[i \leftarrow b_i \cap (-\infty, c]\,] \\
\mathsf{T}(b, c < v_i) & \stackrel{\text{def}}{=} & b[i \leftarrow b_i \cap (c, +\infty)\,] & \mathsf{F}(b, v_i \leq c) \stackrel{\text{def}}{=} b[i \leftarrow b_i \cap (c, +\infty)\,] \\
\mathsf{T}(b, v_i < c) & \stackrel{\text{def}}{=} & b[i \leftarrow b_i \cap (-\infty, c)\,] & \mathsf{F}(b, c \leq v_i) \stackrel{\text{def}}{=} b[i \leftarrow b_i \cap (-\infty, c)\,]
\end{array}$$

$$\begin{array}{lll}
\mathsf{T}(b, v_j \leq v_i) & \stackrel{\text{def}}{=} & b[i \leftarrow b_i \cap [b_j^L, +\infty)\,][j \leftarrow b_j \cap (-\infty, b_i^U]\,] \\
\mathsf{F}(b, v_i < v_j) & \stackrel{\text{def}}{=} & b[i \leftarrow b_i \cap [b_j^L, +\infty)\,][j \leftarrow b_j \cap (-\infty, b_i^U]\,]
\end{array}$$

Given an expression expr in $\mathbb{E}$ and an environment box b in $\mathbb{B}$, the possible values that expr may take when the variable environment lies in b constitute an interval defined on expr below.

Definition 3.3 (Evaluation of expressions) *Let* b *be in* $\mathbb{B}$ *and* expr *in* $\mathbb{E}$. *If* b *is empty, so is* $[\![\mathsf{expr}]\!]b$. *Otherwise* $[\![\mathsf{expr}]\!]b$ *is defined by induction on* expr.

$$\begin{array}{rcll}
[\![c]\!]b & \stackrel{\text{def}}{=} & [c, c] \qquad [\![v_j]\!]b \stackrel{\text{def}}{=} b_j & \\
[\![c \ * \mathsf{expr}]\!]b & \stackrel{\text{def}}{=} & c * [\![\mathsf{expr}]\!]b = \{c * y \mid y \in [\![\mathsf{expr}]\!]b\} & \\
[\![\mathsf{expr}_1 + \mathsf{expr}_2]\!]b & \stackrel{\text{def}}{=} & [\![\mathsf{expr}_1]\!]b + [\![\mathsf{expr}_2]\!]b = \{y_1 + y_2 \mid y_i \in [\![\mathsf{expr}_i]\!]b\} &
\end{array}$$

3.1 Inductive definition of a forward collecting semantics function

Given a program P in $\mathbb{P}$ and an instruction I in $\mathbb{I}$, the semantic functions F_P of P and F_I of I are typed as $F_P : \mathbb{B} \to \mathbb{B}^{|P|} \to \mathbb{B}^{|P|}$ and $F_I : \mathbb{B} \to \mathbb{B}^{|I|} \to \mathbb{B}^{|I|}$ respectively. The right-hand part of the types, *i.e.* $\mathbb{B}^{|\cdot|} \to \mathbb{B}^{|\cdot|}$, corresponds to the usual definition of the semantics function. The left-hand part of the types, *i.e.* the stand-alone $\mathbb{B}$, corresponds to an environment box that the program's execution may start with. Using this box parameter allows programs and instructions with free variables to be meaningful and to have associated semantics functions. This is useful when defining these functions inductively on programs and instructions.

Definition 3.4 (Upper bound operators) *We define the upper bounding operator* $\cup^{\mathbb{B}}$ *in* $\mathbb{B}$ *and an asymmetric union* $\cup_w^{\mathbb{B}}$ *(which is useful for the* while *loop):*

$$x \cup^{\mathbb{B}} y \stackrel{\text{def}}{=} \bigcap_{x,y \subseteq z} z \qquad\qquad x \cup_w^{\mathbb{B}} y \stackrel{\text{def}}{=} \begin{cases} \emptyset & \text{if } x = \emptyset \\ x \cup^{\mathbb{B}} y & \text{otherwise.} \end{cases}$$

The semantics function is defined below by induction on programs and instructions. For the sake of readability, we may write $F_P(b)(X)$ instead of $F_P(b, X)$ for every P, b, and X in the definition and in the sequel. Also, in the definition expr and test are meta-variables for expressions in $\mathbb{E}$ and tests in $\mathbb{T}$. The while-loop invokes the asymmetric union $\cup_w^{\mathbb{B}}$. It is relevant since the execution of a program may exit a while-loop only after performing the entrance test.

Definition 3.5 (Forward collecting semantics function)

$$\forall b \in \mathbb{B}\ F_{\texttt{skip}}(b) : \mathbb{B} \to \mathbb{B} \quad \text{s.t. } x \mapsto b$$

$$\forall b \in \mathbb{B}\ F_{I\,P}(b) : \mathbb{B}^{|I|} \times \mathbb{B}^{|P|} \to \mathbb{B}^{|I|} \times \mathbb{B}^{|P|} \quad \text{s.t. } X, Y \mapsto F_I(b, X), F_P(X_{|I|}, Y)$$

$$\forall b \in \mathbb{B}\ F_{v_i \leftarrow \mathsf{expr}}(b) : \mathbb{B} \to \mathbb{B} \quad \text{s.t. } x \mapsto b[i \leftarrow [\![\mathsf{expr}]\!]b]$$

$$\forall b \in \mathbb{B}\ F_{\texttt{if test}\,\{R\}\,\{S\}}(b) : \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B} \times \mathbb{B}^{|S|} \times \mathbb{B} \to \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B} \times \mathbb{B}^{|S|} \times \mathbb{B}$$
$$x, X, y, Y, z \mapsto \mathsf{T}(b, \mathsf{test}), F_R(x, X), \mathsf{F}(b, \mathsf{test}), F_S(y, Y), X_{|R|} \cup^{\mathbb{B}} Y_{|S|}$$

$$\forall b \in \mathbb{B}\ F_{\texttt{while test}\,\{R\}}(b) : \mathbb{B} \times \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B} \to \mathbb{B} \times \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B}$$
$$x, y, Y, z \mapsto b \cup^{\mathbb{B}}_{w} Y_{|R|}, \mathsf{T}(x, \mathsf{test}), F_R(y, Y), \mathsf{F}(x, \mathsf{test}).$$

The function $F_P(b)$ is an increasing self-map in the complete lattice of environment boxes $\langle \mathbb{B}, \subseteq, \cup^{\mathbb{B}}, \cap \rangle^{|P|}$, so by Tarski's fixed point theorem [25] $F_P(b)$ has a lfp which is included in every post-fixed point of $F_P(b)$.

4 Mathematical programming

We assume in the following that the distance between two different elements in $\mathcal{C}$ is greater than a given $\epsilon > 0$. So any $v_j < v_i$ (resp. $v_i < v_j$) can be replaced with $v_j + \epsilon \leq v_i$ (resp. $v_i \leq v_j + \epsilon$) in $\mathbb{P}$-programs. To each box $b \in \mathbb{B}$ we associate a triplet $(e, \ell, u) \in \{0, 1\} \times \mathbb{R}^n \times \mathbb{R}^n$, where $b = [\ell(b)_1, u(b)_1] \times \cdots \times [\ell(b)_n, u(b)_n]$ and $e = 0$ iff b is empty.

The constraints $\mathsf{SB}(b, e)$ force the bounds of b to 0 when the binary decision variable e is 0.

$$\mathsf{SB}(b, e) \quad \stackrel{\text{def}}{=} \quad \bigwedge_{1 \leq i \leq n} \quad (1 - e)u_i = 0 \quad \wedge \quad (1 - e)\ell_i = 0.$$

The constraint $\mathsf{Incl}(b, b', e)$, active only when $e = 1$, compares the bounds of b' and the bounds of b.

$$\mathsf{Incl}(b, b', e) \quad \stackrel{\text{def}}{=} \quad \bigwedge_{1 \leq i \leq n} \quad e\,u_i \leq e\,u'_i \quad \wedge \quad e\,\ell'_i \leq e\,\ell_i.$$

The constraints $\mathsf{UI}(b^1, b^2, b^3)$ model the statement $\texttt{if} \quad v_j \leq v_i \, \{b^1 \ldots\} \, \{b^2 \ldots\} \, b^3$, where the b^j are environment boxes at key control points. Intuitively, they say that b^3 is empty iff both b^1 and b^2 are empty, otherwise b^3 includes the box-wise union $b^1 \cup^{\mathbb{B}} b^2$; e^1, e^2 deactivate the constraints when b^1, b^2 are empty, so that the bounds of the empty box (i.e. 0) should not interfere with meaningful bounds in further constraints. To compute the exact box-wise union, equality must hold instead of loose inclusion. This will however be enforced by the objective function direction.

$$\mathsf{UI}(b^1, b^2, b^3) \quad \stackrel{\text{def}}{=} \quad e^3 = e^1 + e^2 - e^1 e^2 \wedge \mathsf{SB}(b^3, e^3) \wedge$$
$$\mathsf{Incl}(b^1, b^3, e^1) \wedge \mathsf{Incl}(b^2, b^3, e^2).$$

The constraints $\mathsf{UW}(b^1, b^2, b^3)$ model the statement $\texttt{while}\ b^3\ v_j \leq v_i \, \{b^2 \ldots\} b^1$. Intuitively, they say that b^1 is empty if b^3 is empty, in which case b^3 is set to empty; otherwise, if b^2 is non-empty, its bounds are taken into account; the bounds of b^1 may always be taken into account since b^1 is empty if b^3 is empty.

$$\mathsf{UW}(b^1, b^2, b^3) \quad \stackrel{\text{def}}{=} \quad e^3 \geq e^1 \wedge \mathsf{SB}(b^3, e^3) \wedge$$
$$\mathsf{Incl}(b^1, b^3, 1) \wedge \mathsf{Incl}(b^2, b^3, e^2).$$

The constraints $\mathsf{Id}(b, b', \mathcal{I})$ enforce $b' = b$ (when b' is non-empty) on all

components not included in $\mathcal{I}$.

$$\mathsf{Id}(b,b',\mathcal{I}) \stackrel{\text{def}}{=} \bigwedge_{1\le j\le n}^{j\notin\mathcal{I}} u'_j = e'u_j \wedge \ell'_j = e'\ell_j.$$

The constraints $\mathsf{Empty}(e,e',x,y)$ set e' to 0 if $e=0$ or $x-y<0$ and otherwise to 1 (used to define intersection).

$$\begin{aligned}\mathsf{Empty}(e,e',x,y) \stackrel{\text{def}}{=}\ & e'\le e \ \wedge\ 0\le e'(x-y) \ \wedge\\ & 0\le e(1-e')(y-\epsilon-x).\end{aligned}$$

The constraints $\mathsf{ConstInter}(b,b^s,b^f,c,i)$ disjoin the i-th interval component of b into b^s,b^f, assigned respectively to the success (s) and failure (f) of the test $c\le v_i$; Id constraints manage the other components, the Empty constraints manage emptiness, and the last constraints set the bounds in a correlated manner, i.e. the bounds of b^s involve b^f and conversely.

$$\begin{aligned}&\mathsf{ConstInter}(b,b^s,b^f,c,i) \stackrel{\text{def}}{=}\\ &\mathsf{Id}(b,b^s,\{i\}) \ \wedge\ \mathsf{Id}(b,b^f,\{i\}) \ \wedge\\ &\mathsf{Empty}(e,e^s,u_i,c) \ \wedge\ \mathsf{Empty}(e,e^f,c-\epsilon,\ell_i) \ \wedge\\ &u^s_i = e^s u_i \ \wedge\ \ell^s_i = (1-e^f)\ell_i + e^s e^f c \ \wedge\\ &\ell^f_i = e^f\ell_i \ \wedge\ u^f_i = (1-e^s)u_i + e^s e^f(c-\epsilon).\end{aligned}$$

Similarly, the constraints $\mathsf{Inter}(b,b^s,b^f,j,i)$ disjoins the i-th and j-th interval components of b into b^s,b^f, assigned respectively to the success (s) and failure (f) of the test $v_j\le v_i$.

$$\begin{aligned}&\mathsf{Inter}(b,b^s,b^f,j,i) \stackrel{\text{def}}{=}\\ &\mathsf{Id}(b,b^s,\{i,j\}) \ \wedge\ \mathsf{Id}(b,b^f,\{i,j\}) \ \wedge\\ &\mathsf{Empty}(e,e^{si},u_i,\ell_j) \ \wedge\ \mathsf{Empty}(e,e^{fi},\ell_j,\ell_i) \ \wedge\\ &\mathsf{Empty}(e,e^{sj},u_i,\ell_j) \ \wedge\ \mathsf{Empty}(e,e^{fj},u_j-\epsilon,u_i) \ \wedge\\ &u^s_i = e^{si}u_i \ \wedge\ \ell^s_i = (1-e^{fi})\ell_i + e^{si}e^{fi}\ell_j \ \wedge\\ &\ell^f_i = e^{fi}\ell_i \ \wedge\ u^f_i = (1-e^{si})u_i + e^{fi}e^{si}(\ell_j-\epsilon) \ \wedge\\ &\ell^s_j = e^{sj}\ell_j \ \wedge\ u^s_j = (1-e^{fj})u_j + e^{sj}e^{fj}u_i \ \wedge\\ &u^f_j = e^{fj}u_j \ \wedge\ \ell^f_j = (1-e^{sj})\ell_j + e^{fj}e^{sj}(u_i+\epsilon).\end{aligned}$$

The following express how environment boxes are transformed by assignment statements. Let b be in $\mathbb{B}$ and expr in $\mathbb{E}$; $\mathsf{L}(b,\mathsf{expr})$ and $\mathsf{U}(b,\mathsf{expr})$ are defined by induction on expr.

$$\begin{aligned}
\mathsf{L}(b,c),\quad \mathsf{U}(b,c) &\stackrel{\text{def}}{=} e(b)c,\quad e(b)c\\
\mathsf{L}(b,v_j),\quad \mathsf{U}(b,v_j) &\stackrel{\text{def}}{=} \ell(b)_j,\quad u(b)_j\\
\mathsf{L}(b,c*\mathsf{expr}),\ \mathsf{U}(b,c*\mathsf{expr}) &\stackrel{\text{def}}{=} c*\mathsf{L}(b,\mathsf{expr}),\ c*\mathsf{U}(b,\mathsf{expr}) \quad \text{if} \quad 0\le c\\
\mathsf{L}(b,c*\mathsf{expr}),\ \mathsf{U}(b,c*\mathsf{expr}) &\stackrel{\text{def}}{=} c*\mathsf{U}(b,\mathsf{expr}),\ c*\mathsf{L}(b,\mathsf{expr}) \quad \text{if} \quad c<0\\
\mathsf{L}(b,\mathsf{expr}_1+\mathsf{expr}_2) &\stackrel{\text{def}}{=} \mathsf{L}(b,\mathsf{expr}_1)+\mathsf{L}(b,\mathsf{expr}_2)\\
\mathsf{U}(b,\mathsf{expr}_1+\mathsf{expr}_2) &\stackrel{\text{def}}{=} \mathsf{U}(b,\mathsf{expr}_1)+\mathsf{U}(b,\mathsf{expr}_2)\\
\mathsf{Assign}(b,b',i,\mathsf{expr}) &\stackrel{\text{def}}{=} e'=e \wedge u'_i = \mathsf{U}(b,\mathsf{expr}) \wedge \ell'_i = \mathsf{L}(b,\mathsf{expr}).
\end{aligned}$$

4.1 Inductive definition of a mathematical program

For all programs or instructions P/I (*i.e.* P in $\mathbb{P}$ or I in $\mathbb{I}$), for all environment boxes b, the constraints $C_{P/I}(b)$ and the objective function $O_{P/I}$ are defined inductively below.

Definition 4.1 (Objective and constraints)

$$O_{\texttt{skip}},\, O_{v_i \leftarrow \texttt{expr}} : \quad \mathbb{B} \to \mathcal{C} \quad \text{s.t. } x \mapsto 0$$

$$\text{For all } P, I, R, S: \quad O_{I\,P} : \quad \mathbb{B}^{|I|} \times \mathbb{B}^{|P|} \to \mathcal{C} \quad \text{s.t. } (X, Y) \mapsto O_I(X) + O_P(Y)$$

$$O_{\texttt{if test}\,\{R\}\,\{S\}} : \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B} \times \mathbb{B}^{|S|} \times \mathbb{B} \to \mathcal{C} \quad \text{s.t.}$$
$$(x, X, y, Y, z) \mapsto O_R(X) \; + \; O_S(Y) + \sum_{1 \le i \le n} (u(z)_i - \ell(z)_i)$$

$$O_{\texttt{while test}\,\{R\}} : \quad \mathbb{B} \times \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B} \to \mathcal{C} \quad \text{s.t.}$$
$$(x, y, Y, z) \mapsto O_R(Y) + \sum_{1 \le i \le n} (u(x)_i - \ell(x)_i)$$

$$C_{\texttt{skip}}(b) : \quad \mathbb{B} \to \{0,1\} \quad \text{s.t. } x \mapsto e(x) = e(b) \wedge \left(\textstyle\bigwedge_{1 \le i \le n} u(x)_i = u(b)_i \wedge \ell(x)_i = \ell(b)_i\right)$$

$$C_{v_i \leftarrow \texttt{expr}}(b) : \quad \mathbb{B} \to \{0,1\} \quad \text{s.t. } x \mapsto \mathsf{Id}(b, x, \{i\}) \wedge \mathsf{Assign}(b, x, i, \texttt{expr})$$

$$\text{For all } P, I: \quad C_{I\,P} : \mathbb{B}^{|I|} \times \mathbb{B}^{|P|} \to \{0,1\} \quad \text{s.t. } (X, Y) \mapsto C_I(X) \wedge C_P(X_{|I|}, Y)$$

If $I = \texttt{if}\ c \le v_i\ \{R\}\,\{S\}$ (resp. $v_i \le c\ \{R\}\,\{S\}$):

$$C_I(b) : \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B} \times \mathbb{B}^{|S|} \times \mathbb{B} \to \{0,1\} \quad \text{s.t.}$$
$$(x, X, y, Y, z) \mapsto C_R(x, X) \wedge C_S(y, Y) \wedge \mathsf{UI}(X_{|R|}, Y_{|S|}, z) \wedge \mathsf{ConstInter}(b, x, y, c, i) \ (\textit{resp. } \mathsf{ConstInter}(b, y, x, c + \epsilon, i))$$

If $I = \texttt{if}\ v_j \le v_i\ \{R\}\,\{S\}$:

$$C_I(b) : \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B} \times \mathbb{B}^{|S|} \times \mathbb{B} \to \{0,1\} \quad \text{s.t. } (x, X, y, Y, z) \mapsto C_R(x, X) \wedge C_S(y, Y) \wedge \mathsf{UI}(X_{|R|}, Y_{|S|}, z) \wedge \mathsf{Inter}(b, x, y, j, i)$$

If $I = \texttt{while}\ c \le v_i\ \{R\}$ (resp. $\texttt{while}\ v_i \le c\ \{R\}$):

$$C_I(b) : \mathbb{B} \times \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B} \to \{0,1\} \quad \text{s.t.} (x, y, Y, z) \mapsto C_R(y, Y) \wedge \mathsf{UW}(b, Y_{|R|}, x) \wedge \mathsf{ConstInter}(x, y, z, c, i) (\textit{resp. } \mathsf{ConstInter}(x, z, y, c + \epsilon, i))$$

If $I = \texttt{while}\ v_j \le v_i\ \{R\}$:

$$C_I(b) : \mathbb{B} \times \mathbb{B} \times \mathbb{B}^{|R|} \times \mathbb{B} \to \{0,1\} \text{ s.t. } (x, y, Y, z) \mapsto C_R(y, Y) \wedge \mathsf{UW}(b, Y_{|R|}, x) \wedge \mathsf{Inter}(x, y, z, j, i).$$

Let $P \in \mathbb{P} \cup \mathbb{I}$ and $b \in \mathbb{B}$. A vector $X \in \mathbb{B}^{|P|}$ is a unique solution of the MP $M_P(O_P, C^0, C_P(b))$, where the constraints $C^0 : \mathbb{B} \to \{0,1\}$ are s.t. $x \mapsto \bigwedge_{1 \le i \le n} \ell(x)_i \le u(x)_i$, if it is the only vector satisfying the constraints and minimizing the objective: $M_P(b, X) \stackrel{\text{def}}{=} C_P(b, X) \wedge \forall Y, C_P(b, Y) \Rightarrow O_P(Y) \le O_P(X) \Rightarrow Y = X$.

Example 4.2 Let P be the program `int x=1; while (x<100) x=x+1;`. Then the corresponding MP is as follows (in this case we need only employ two binary variables e^3, e^5 controlling emptiness on the different test outputs, the others being fixed to 1). We minimize $\sum_{k=1}^{5}(u^k - \ell^k)$ such that: $\forall k \le 5\ u^k \ge \ell^k$ (bound consistency constraints C^0), $\ell^1 = 1 \wedge u^1 = 1$ (Assign for `x=1`), $\ell^2 \le \ell^1 \wedge u^2 \ge u^1 \wedge \ell^2 \le \ell^4 \wedge u^2 \ge u^4$ (Incl constraints), $\ell^4 = \ell^3 + 1 \wedge u^4 = u^3 + 1$ (Assign for `x=x+1`), $(1 - e^3)(\ell^2 - 100) \ge 0 \wedge e^3(99 - \ell^2) \ge 0 \wedge e^5(u^2 - 100) \ge 0 \wedge (1 - e^5)(99 - u^2) \ge 0$ (Empty constraints), $u^5 = e^5 u^2 \wedge \ell^5 = (1 - e^3)\ell^2 + 100e^3e^5 \wedge \ell^3 = e^3\ell^2 \wedge u^3 = (1 - e^5)u^2 + 99e^5e^3$ (ConstInter for `x<100`). This nonlinear MINLP was solved using Couenne [3] to find the (guaranteed) lfp $([1,1],[1,100],[1,99],[2,100],[100,100])$. Notice no widening operator was ever used, and no variable was artificially bounded to arbitrary large constants.

5 Correspondence between the semantics function and the mathematical program

We list some lemmata relating semantics function and MP, which prove that the MP characterizes the lfp.

Lemma 5.1 $b \neq \emptyset \quad \Rightarrow \quad [\![\mathsf{expr}]\!]b = [\mathsf{L}(b, \mathsf{expr}), \mathsf{U}(b, \mathsf{expr})]$

Proof by induction on expressions.

Lemma 5.2 $C_{v_i \leftarrow \mathsf{expr}}(b, x) \quad \Leftrightarrow \quad x = b[i \leftarrow [\![\mathsf{expr}]\!]b]$

Proof by double implication. For each implication consider cases on emptyness of b and use Lemma 5.1.

Lemma 5.3 $\mathsf{UI}(x, y, z) \quad \Leftrightarrow \quad x \cup^{\mathbb{B}} y \subseteq z \ \wedge \ (x = \emptyset \wedge y = \emptyset \ \Rightarrow \ z = \emptyset).$

Lemma 5.4 $\mathsf{UW}(x, y, z) \quad \Leftrightarrow \quad x \cup^{\mathbb{B}}_{w} y \subseteq z \ \wedge \ (x = \emptyset \wedge y = \emptyset \ \Rightarrow \ z = \emptyset).$

Proofs of the two above lemmata by case splitting.

Lemma 5.5 $\mathsf{ConstInter}(b, x, y, c, i) \Leftrightarrow x = \mathsf{T}(b, c \leq v_i) \ \wedge \ y = \mathsf{F}(b, c \leq v_i).$

Lemma 5.6 $\mathsf{Inter}(b, x, y, j, i) \Leftrightarrow x = \mathsf{T}(b, v_j \leq v_i) \ \wedge \ y = \mathsf{F}(b, v_j \leq v_i).$

Again, proofs of the two above lemmata by double implication and case split on emptyness of b. For the non-empty case, one may case split along $u(b)_i < c$, $c \leq \ell(b)_i$, and $\ell(b)_i < c \leq u(b)_i$.
Every fixed point of a semantics function complies with the corresponding constraints.

Lemma 5.7 $\forall P/I, b, X \quad F_{P/I}(b, X) = X \ \Rightarrow \ C_{P/I}(b, X).$

We define vector inclusion as follows:

$$\begin{aligned} X \subseteq Y &\stackrel{\text{def}}{=} \forall i, \, X_i \subseteq Y_i \\ X \subset Y &\stackrel{\text{def}}{=} X \subseteq Y \wedge X \neq Y \end{aligned}$$

Every vector satisfying the constraints of the MP is a post-fixed point of the corresponding semantics function.

Lemma 5.8 $\forall P/I, b, X \quad C_{P/I}(b, X) \ \Rightarrow \ F_{P/I}(b, X) \subseteq X.$

The objective $O_{P/I}$ is (weakly) increasing.

Lemma 5.9 $X \subseteq X' \quad \Rightarrow \quad O_{P/I}(X) \leq O_{P/I}(X').$

Proof by induction on programs and instructions, notice that $O_{P/I}$ only involves sums of upper bounds minus lower bounds.
The objective is strongly increasing on vectors satisfying the same constraints.

Lemma 5.10 $\forall P/I, b, X \quad C_{P/I}(b, X) \ \wedge \ C_{P/I}(b, X') \ \wedge \ X \subset X' \Rightarrow$
$O_{P/I}(X) < O_{P/I}(X').$

The lfp of the forward-collecting semantics function of a program is the unique solution of the MP associated with the program.

Theorem 5.11 $F_P(b,X) = X \wedge (F_P(b,Y) = Y \Rightarrow X \subseteq Y) \;\Rightarrow\; M_P(b,X)$.

6 Policy Iteration algorithm in the MP setting

The products $e^i e^j$ between binary variables appearing for some i,j in the MP of Sect. 4 can all be reformulated exactly as follows: replace $e^i e^j$ by an added binary variable e^{ij}, and adjoin the constraints $e^{ij} \leq e^i$, $e^{ij} \leq e^j$, $e^{ij} \geq e^i + e^j - 1$. This is also called "Fortet's reformulation", see [20], p. 178. After this reformulation, the MP has the following bilinear structure:

$$\min\{w^\top(x,e) \,|\, h(x) \leq 0 \;\wedge\; e^\top(g(x) - x) \leq 0 \;\wedge\; (1-e)^\top(g'(x) - x) \leq 0\}, \quad (1)$$

where $x = (\ell, u)$ is a vector of continuous decision variables, w is a constant vector encoding the (linear) interval width objective, h, g, g' are affine forms, and $e \in \{0,1\}^p$ are binary decision variables. A *policy* in this setting is an assignment of binary values to the binary variable vector e. It appears clear from (1) that policies determine whether $g(x) \leq x$ or $g'(x) \leq x$, where both g, g' are affine forms. The PI algorithm can be re-cast in the MP setting as follows.

(i) Let e^* be an initial (feasible) policy
(ii) Let $e \leftarrow e^*$ in (1), yielding a LP
(iii) *Value determination*: solve the LP to obtain a solution x^*
(iv) Let $\bar{e} \leftarrow e^*$
(v) *Policy improvement*:
  (a) $\forall i \leq p \; (e^*_i = 1 \wedge g_i(x^*) > g'_i(x^*) \Rightarrow \bar{e}_i \leftarrow 0)$
  (b) $\forall i \leq p \; (e^*_i = 0 \wedge g_i(x^*) < g'_i(x^*) \Rightarrow \bar{e}_i \leftarrow 1)$
(vi) If $\bar{e} = e^*$ then terminate with fixed point x^*
(vii) Set $e^* \leftarrow \bar{e}$ and repeat from Step ii.

Thus, the PI algorithm performs a local search on the e-space of (1), whereas the algorithms mentioned in Sect. 7 explore the entirety of the e-space, thereby *always* finding the guaranteed lfp. By comparison, a known sufficient condition for PI methods to find a guaranteed lfp is that the semantics function should be non-expansive in the sup norm [7].

7 Solving the mathematical program

Given a program P, the constraints C_P of the associated MP are generated in linear time w.r.t. the size $|P|$ of the program. These constraints involve $O(|P|)$ binary variables. There are $2^{O(|P|)}$ possible assignments for these variables. Fixing the binary variables to one of these assignments yields an LP, which can be solved in polynomial time in the size of the instance [18] (LP methods can also certify infeasibility and unboundedness). If for all possible assignments the LP has no solution, it means that the lfp of F_P is not finite. Otherwise,

any finite solution is a post-fixed point of F_P by Lemma 5.8. The lfp is one of the post-fixed points according to Theorem 5.11, and it is the smallest of them according to Tarski [25]. The practical complexity of the proposed algorithm for solving SAAIP is likely to be close to its worst-case complexity bound (exponential), and is thus only useful to improve the best known complexity bound so far for SAAIP (Kleene's iteration with no widening takes infinite time; the method proposed in [22] runs in doubly exponential time).

We can use MP methodology to derive a practically applicable algorithm for solving a slightly modified SAAIP. By assuming an arbitrary large bound on the variable values (this is akin to imposing that all boxes are in a large pre-determined box, similarly to what is done in widening), we are able to reformulate exactly ([20], p. 179) all products between decision variables occurring in the MP to a linear form, yielding a MILP which we solve using the BB based solver CPLEX 11 [17] on a 2.4GHz Intel Xeon CPU with 8GB RAM. Notice that for most practical cases, the large bound need not be arbitrary, as automatic range reduction techniques for MILP can help considerably [24].

Based on the above analysis, we implemented a C parser (recognizing a subset of C which is sufficiently rich to be Turing-equivalent) that outputs the corresponding MP. Our testbed consists of several (small) C programs [7] with integer affine arithmetic: some minimal ones for validation purposes (`short`), some longer ones (`long`) generated randomly, three instances using arrays and functions and the `subway` code from [16] with the `random()` call commented out and `nbtrains` set to 10. We compared our results to those obtained by a prototype implementation of the PI algorithm [7]; in both approaches, $\top$ was set to the interval $[-5000, 5000]$. In Table 1 we report: instance name, lines of code, total number of variables (arrays of length n counting as n variables), seconds of user CPU, lfp statistics (sum of the widths of all intervals $|\cdot|$, number of $\top$ intervals $|\top|$, sum of widths of non-$\top$ intervals $|\neg\top|$). In all tests we obtained fixed points of width equal to or smaller than those obtained by PI, thus validating the approach.

8 Conclusion

We exhibited a mathematical program modelling the problem of finding the lfp of the semantic function of a program with integer affine arithmetic, and proceeded to show that this yields a practically viable method for computing lfps of programs. By this example we wish to emphasize the usefulness that the standard mathematical programming toolbox has in the field of static analysis by abstract interpretation. Future work will extend the MP approach to work with different domains (specifically, relational domains); we shall also employ

[7] http://www.lix.polytechnique.fr/~liberti/nsad10-instances.zip

Instance			MP				PI			
Name	*Lines*	*Vars*	CPU	$\|\cdot\|$	$\|\top\|$	$\|\neg\top\|$	CPU	$\|\cdot\|$	$\|\top\|$	$\|\neg\top\|$
`short_31`	32	3	0.008	250002	**25**	**2**	0	250023	25	23
`short_32`	20	3	0.02	270077	27	77	0	270077	27	77
`short_35`	22	3	0.02	32028	3	2028	0	32028	3	2028
`short_37`	25	3	0.008	420000	**42**	**0**	0	470000	47	0
`short_38`	35	3	0.12	34501	3	4501	0	34501	3	4501
`long_1`	213	4	0.768	90000	**9**	**0**	0.004	90052	9	52
`long_2`	217	4	0.916	80000	**8**	**0**	0.008	90002	9	2
`long_3`	130	4	0.64	120426	**12**	**426**	0.06	4.36e+06	436	246
`long_4`	195	4	0.412	120000	**12**	**0**	0.008	120002	12	2
`long_5`	216	4	0.772	110000	**11**	**0**	0.004	120010	12	10
`arrays`	22	6	0.04	300139	30	139	-	-	-	-
`fun_arrays`	53	6	0.016	30000	3	0	-	-	-	-
`functions`	62	7	0.112	101190	10	1190	-	-	-	-
`subway`	62	34	9.25258	1.77e+07	1766	675	-	-	-	-

Table 1
Comparison of MP and PI methods. Instances are marked '-' whenever the program could not be analyzed because of parsing limitations (arrays, functions) in our prototype PI implementation.

other methods, such as policy iteration, as upper bounding procedures within the standard Branch-and-Bound approach used to solve Problem (1).

References

[1] Adje, A., S. Gaubert and E. Goubault, *Computing the smallest fixed point of nonexpansive mappings arising in game theory and static analysis of programs* (2008), presented at MTNS 2008, also arXiv.org:0806.1160.

[2] Adjé, A., S. Gaubert and E. Goubault, *Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis*, in: A. Gordon, editor, *European Symposium on Programming* (2010), pp. 23–42.

[3] Belotti, P., J. Lee, L. Liberti, F. Margot and A. Wächter, *Branching and bounds tightening techniques for non-convex MINLP*, Optimization Methods and Software **24** (2009), pp. 597–634.

[4] Bonami, P., L. Biegler, A. Conn, G. Cornuéjols, I. Grossmann, C. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya and A. Wächter, *An algorithmic framework for convex mixed integer nonlinear programs*, Technical Report RC23771, IBM Corporation (2005).

[5] Bonami, P., G. Cornuéjols, A. Lodi and F. Margot, *A feasibility pump for mixed integer nonlinear programs*, Mathematical Programming **119** (2009).

[6] Colón, M., S. Sankaranarayanan and H. Sipma, *Linear invariant generation using non-linear constraint solving*, in: W. Hunt, editor, *Computer Aided Verification*, LNCS **2725** (2003), pp. 420–432.

[7] Costan, A., S. Gaubert, E. Goubault, M. Martel and S. Putot, *A policy iteration algorithm for computing fixed points in static analysis of programs*, in: K. Etessami and S. Rajamani, editors, *Computer Aided Verification*, LNCS **3576** (2005), pp. 462–475.

[8] Cousot, P., *Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming*, in: R. Cousot, editor, *Verification, Model Checking and Abstract Interpretation*, LNCS **3385** (2005), pp. 17–19.

[9] Cousot, P. and R. Cousot, *Static determination of dynamic properties of programs*, in: *Proceedings of the Second International Symposium on Programming* (1976), pp. 106–130.

[10] Cousot, P. and R. Cousot, *Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixed points*, Principles of Programming Languages 4 (1977), pp. 238–252.

[11] Cousot, P. and R. Cousot, *Abstract interpretation frameworks*, Journal of Logic and Computation **2** (1992), pp. 511–547.

[12] Gaubert, S., E. Goubault, A. Taly and S. Zennou, *Static analysis by policy iteration on relational domains*, in: R. D. Nicola, editor, *European Symposium on Programming*, LNCS **4421** (2007), pp. 237–252.

[13] Gawlitza, T., J. Leroux, J. Reineke, H. Seidl, G. Sutre and R. Wilhelm, *Polynomial precise interval analysis revisited*, in: S. Albers, H. Alt and S. Näher, editors, *Festschrift Mehlhorn*, Lecture Notes in Computer Science **5760** (2009), pp. 422–437.

[14] Gawlitza, T. and H. Seidl, *Precise fixpoint computation through strategy iteration*, in: R. D. Nicola, editor, *European Symposium on Programming*, LNCS **4421** (2007), pp. 300–315.

[15] Gawlitza, T. and H. Seidl, *Precise relational invariants through strategy iteration*, in: J. Duparc and T. Henzinger, editors, *Computer Science Logic*, 2007, pp. 23–40.

[16] Halbwachs, N., Y.-E. Proy and P. Roumanoff, *Verification of real-time systems using linear relation analysis*, Formal Methods in System Design **11** (1997), pp. 157–185.

[17] ILOG, "ILOG CPLEX 11.0 User's Manual," ILOG S.A., Gentilly, France (2008).

[18] Karmarkar, N., *A new polynomial time algorithm for linear programming*, Combinatorica **4** (1984), pp. 373–395.

[19] Liberti, L., *Reformulations in mathematical programming: Definitions and systematics*, RAIRO-RO **43** (2009), pp. 55–86.

[20] Liberti, L., S. Cafieri and F. Tarissan, *Reformulations in mathematical programming: A computational approach*, in: A. Abraham, A.-E. Hassanien, P. Siarry and A. Engelbrecht, editors, *Foundations of Computational Intelligence Vol. 3*, number 203 in Studies in Computational Intelligence, Springer, Berlin, 2009 pp. 153–234.

[21] Liberti, L., N. Mladenović and G. Nannicini, *A good recipe for solving MINLPs*, in: V. Maniezzo, T. Stützle and S. Voß, editors, *Hybridizing metaheuristics and mathematical programming*, Annals of Information Systems **10** (2009), pp. 231–244.

[22] Monniaux, D., *Automatic modular abstractions for linear constraints*, in: *Principles of programming languages*, ACM (2009), pp. 140–151.

[23] Sankaranarayanan, S., H. Sipma and Z. Manna, *Scalable analysis of linear systems using mathematical programming*, in: R. Cousot, editor, *Verification, Model Checking and Abstract Interpretation*, LNCS **3385** (2005), pp. 25–41.

[24] Savelsbergh, M., *Preprocessing and probing techniques for mixed integer programming problems*, INFORMS Journal on Computing **6** (1994), pp. 445–454.

[25] Tarski, A., *A lattice-theoretical fixpoint theorem and its applications*, Pacific Journal of Mathematics **5** (1955), pp. 285–309.

[26] Williams, H., "Model Building in Mathematical Programming," Wiley, Chichester, 1999, 4th edition.

Appendix: Proofs

Lemma 5.7 $\forall P/I, b, X \quad F_{P/I}(b, X) = X \Rightarrow C_{P/I}(b, X)$.

Proof By induction on P/I. (Induction hypotheses are assumed implicitly.) Case $P = I$: trivial. Case $P = I\,Q$: assume $F_P(b, X, Y) = (X, Y)$. So by Definition 3.5, X and Y are fixed points of $F_I(b)$ and $F_Q(X_{|I|})$ respectively,

so $C_I(b, X)$ and $C_Q(X_{|I|}, Y)$ by induction hypothesis, so $C_P(b, X, Y)$ by Definition 4.1. Case $P = \texttt{skip}$: if $F_P(b, x) = x$ then $x = b$ by Definition 3.5, then $C_P(b, x)$ by Definition 4.1. Case $I = v_i \leftarrow \mathsf{expr}$: assume $F_I(b, x) = x$. So $x = b[i \leftarrow [\![\mathsf{expr}]\!]b]$ by Definition 3.5, so $C_I(b, x)$ by Lemma 5.2. Case $I = \mathsf{if\ test}\,\{R\}\,\{S\}$: assume $F_I(b, x, X, y, Y, z) = (x, X, y, Y, z)$. So by Definition 3.5, $x = \mathsf{T}(b, \mathsf{test})$, $y = \mathsf{F}(b, \mathsf{test})$, X and Y are fixed points of $F_R(x)$ and $F_S(y)$ respectively, and $z = X_{|R|} \cup^{\mathbb{B}} Y_{|S|}$. So $C_R(x, X)$ and $C_S(y, Y)$ by induction hypothesis and $\mathsf{UI}(X_{|R|}, Y_{|S|}, z)$ by Lemma 5.3. For constant tests, assume that test is $c \leq v_i$ (since the other case is similar). So $\mathsf{ConstInter}(b, x, y, c, i)$ by Lemma 5.5. For non-constant tests, by Lemma 5.6 we have $\mathsf{Inter}(b, x, y, j, i)$. Therefore $C_I(b, X, Y, z)$ by Definition 4.1. Case $I = \texttt{while}\ \mathsf{test}\,\{R\}$: assume $F_I(b, x, y, Y, z) = (x, y, Y, z)$. So by Definition 3.5, $x = b \cup^{\mathbb{B}}_w Y_{|R|}$, $y = \mathsf{T}(x, \mathsf{test})$, $Y = F_R(y, Y)$, and $z = \mathsf{F}(x, \mathsf{test})$. So $C_R(y, Y)$ by induction hypothesis and $\mathsf{UW}(b, Y_{|R|}, x)$ by Lemma 5.4. For constant tests, assume test is $c \leq v_i$ (since the other case is similar), so $\mathsf{ConstInter}(x, y, z, c, i)$ by Lemma 5.5; for non-constant tests, $\mathsf{Inter}(x, y, z, j, i)$ by Lemma 5.6. Therefore $C_I(b, x, Y, z)$ by Definition 4.1. □

Lemma 5.8 $\forall P/I, b, X \quad C_{P/I}(b, X) \Rightarrow F_{P/I}(b, X) \subseteq X.$

Proof By induction on P/I. (Induction hypotheses are assumed implicitly.) Case $P = I$: trivial. Case $P = I\,Q$: assume $C_P(b, X, Y)$. So $C_I(b, X)$ and $C_Q(X_{|I|}, Y)$ by Definition 4.1, so we have $F_I(b, X) \subseteq X$ and $F_Q(X_{|I|}, Y) \subseteq Y$ by induction hypothesis, so $F_P(b, X, Y) \subseteq (X, Y)$ by Definition 3.5. Case $P = \texttt{skip}$: if $C_P(b, x)$ then $x = b$ By Definition 4.1, so $F_P(b, x) = x$ by Definition 3.5. Case $I = v_i \leftarrow \mathsf{expr}$: assume $C_I(b, x)$. So $x = b[i \leftarrow [\![\mathsf{expr}]\!]b]$ by Lemma 5.2, so $F_I(b, X) = X$ by Definition 3.5. Case $I = \mathsf{if\ test}\,\{R\}\,\{S\}$: constant tests: assume test is $c \leq v_i$ (since the other case is similar). Assume $C_I(b, x, X, y, Y, z)$. So by Definition 4.1, $C_R(x, X)$ and $C_S(y, Y)$ and $\mathsf{ConstInter}(b, x, y, c, i)$ and $\mathsf{UI}(X_{|R|}, Y_{|S|}, z)$. So $x = \mathsf{T}(b, \mathsf{test})$ and $y = \mathsf{F}(b, \mathsf{test})$ by Lemma 5.5. Non-constant tests: $C_R(x, X)$ and $C_S(y, Y)$ and $\mathsf{Inter}(b, x, y, j, i)$ and $\mathsf{UI}(X_{|R|}, Y_{|S|}, z)$. So $x = \mathsf{T}(b, \mathsf{test})$ and $y = \mathsf{F}(b, \mathsf{test})$ by Lemma 5.6. Also $F_R(x, X) \subseteq X$ and $F_S(y, Y) \subseteq Y$ by induction hypothesis and $X_{|R|} \cup^{\mathbb{B}} Y_{|S|} \subseteq z$ by Lemma 5.3. Therefore $F_I(b, x, X, y, Y, z) \subseteq (x, X, y, Y, z)$ by Definition 3.5. Case $I = \texttt{while}\ \mathsf{test}\,\{R\}$: constant tests: assume test is $c \leq v_i$ (since the other case is similar). Assume $C_I(b, x, y, Y, z)$. So by Definition 4.1, $C_R(y, Y)$ and $\mathsf{UW}(b, Y_{|R|}, x)$ and $\mathsf{ConstInter}(x, y, z, c, i)$. So $y = \mathsf{T}(x, \mathsf{test})$ and $z = \mathsf{F}(x, \mathsf{test})$ by Lemma 5.5. Non-constant tests: $C_R(x, X)$ and $C_S(y, Y)$ and $\mathsf{Inter}(b, x, y, j, i)$ and $\mathsf{UI}(X_{|R|}, Y_{|S|}, z)$. Also $F_R(y, Y) \subseteq Y$ by induction hypothesis and $b \cup^{\mathbb{B}}_w Y_{|R|} \subseteq x$ by Lemma 5.4. Therefore $F_I(b, x, y, Y, z) \subseteq (x, y, Y, z)$ by Definition 3.5. □

Lemma 5.9 $\forall P/I, b, X \quad C_{P/I}(b, X) \wedge C_{P/I}(b, X') \wedge X \subset X' \Rightarrow O_{P/I}(X) < O_{P/I}(X')$.

Proof By induction on P/I. (Induction hypotheses are assumed implicitly.) Case $P = I$: trivial. Case $P = I\,Q$: assume $C_P(b, X, Y)$ and $C_P(b, X', Y')$ and $(X, Y) \subset (X', Y')$. So by Definition 4.1, $C_I(b, X)$ and $C_Q(X_{|I|}, Y)$ (resp. with the prime). Also $O_I(X) \leq O_I(X')$ and $O_Q(Y) \leq O_Q(Y')$ by Lemma 5.9 and assumption. If $X \subset X'$ then $O_I(X) < O_I(X')$ by induction hypothesis. If $X = X'$ then $Y \subset Y'$, then $O_Q(Y) < O_Q(Y')$ by induction hypothesis. In both cases $O_P(X, Y) < O_P(X', Y')$ by Definition 4.1. Case $P = \texttt{skip}$: if $C_P(b, x)$ and $C_P(b, x')$ then $x = b = x'$ by Definition 4.1. Case $I = v_i \leftarrow \mathsf{expr}$: assume $C_I(b, x)$ and $C_I(b, x')$ and $x \subset x'$. So $x = x'$ by Definition 4.1 and Lemma 5.2. Case $I = \mathsf{if\ test}\,\{R\}\,\{S\}$: constant test: assume test is $c \leq v_i$ (since the other case is similar). Assume $C_I(b, x, X, y, Y, z)$ and $C_I(b, x', X', y', Y', z')$ and $(x, X, y, Y, z) \subset (x', X', y', Y', z')$. So by Definition 4.1, we have $C_R(x, X)$, $C_S(y, Y)$, and $\mathsf{ConstInter}(b, x, y, c, i)$ (resp. with the prime). So $x' = x$ by Lemma 5.5 applied twice, hence $C_R(x, X)$ and $C_R(x, X')$ (resp. with y and Y). Also $O_R(X) \leq O_R(X')$ and $O_S(Y) \leq O_S(Y')$ by Lemma 5.9 and assumption, and, further, $\sum_{1 \leq i \leq n} (u(z)_i - \ell(z)_i) \leq \sum_{1 \leq i \leq n} (u(z')_i - \ell(z')_i)$ by assumption. If $X \subset X'$ (resp. $Y \subset Y'$) then $O_R(X) < O_R(X')$ (resp. $O_S(Y) < O_S(Y')$) by induction hypothesis. If $X = X'$ and $Y = Y'$, then $z \subset z'$ by assumption, so $z' \neq \emptyset$, so $z \neq \emptyset$ by Definition 4.1 and Lemma 5.3, so $\sum_{1 \leq i \leq n} (u(z)_i - \ell(z)_i) < \sum_{1 \leq i \leq n} (u(z')_i - \ell(z')_i)$. In any case $O_P(x, X, y, Y, z) < O_P(x', X', y', Y', z')$ by Definition 4.1. The argument for non-constant tests is similar. Case $I = \texttt{while}\ \mathsf{test}\,\{R\}$: constant tests: assume test is $c \leq v_i$ (since the other case is similar). Assume $C_I(b, x, y, Y, z)$ and $C_I(b, x', y', Y', z')$ and $(x, y, Y, z) \subset (x', y', Y', z')$. So by Definition 4.1, we have $C_R(y, Y)$ and $\mathsf{UW}(b, Y_{|R|}, x)$ (resp. with the prime). By assumption $O_R(Y) \leq O_R(Y')$ and $\sum_{1 \leq i \leq n} (u(x)_i - \ell(x)_i) \leq \sum_{1 \leq i \leq n} (u(x')_i - \ell(x')_i)$. If $Y \subset Y'$ then $O_R(Y) < O_R(Y')$ by induction hypothesis. Now assume $Y = Y'$. So $\mathsf{UW}(b, Y_{|R|}, x)$ and $\mathsf{UW}(b, Y_{|R|}, x')$. If $x \subset x'$ then $x' \neq \emptyset$, then $x \neq \emptyset$ by Lemma 5.4, then $\sum_{1 \leq i \leq n} (u(x)_i - \ell(x)_i) < \sum_{1 \leq i \leq n} (u(x')_i - \ell(x')_i)$, then $O_I(x, Y, z) < O_I(x', Y', z')$. If $x = x'$ then $y = y'$ and $z = z'$ by Lemma 5.5, contradiction. In any case $O_I(x, y, Y, z) < O_I(x', y', Y', z')$ by Definition 4.1. The argument for non-constant tests is similar. □

Theorem 5.10 $F_P(b, X) = X \wedge (F_P(b, Y) = Y \Rightarrow X \subseteq Y) \Rightarrow M_P(b, X)$.

Proof Let X be the lfp of $F_P(b)$. By Lemma 5.7, $C_P(b, X)$. Assume Y such that $C_P(b, Y)$ and $O_P(Y) \leq O_P(X)$. By Lemma 5.8, Y is a post-fixed point of $F_P(b)$, so $X \subseteq Y$ by Tarski [25], so $Y = X$ by contraposition of Lemma 5.10 and assumption $O_P(Y) \leq O_P(X)$. □