The fixed point problem for general and for linear SRL programs is undecidable

Armando B. Matos[1], Luca Paolini[2], and Luca Roversi[2]

[1] University of Porto, Portugal
[2] University of Turin, Italy
armandobcm@yahoo.com, luca.paolini@unito.it, luca.roversi@unito.it

Abstract. SRL is a total programming language for reversible computing. In SRL, every program that mentions n registers defines a bijection $\mathbb{Z}^n \to \mathbb{Z}^n$. Since SRL contains only a very essential set of commands it is a language suitable for studying strengths and weaknesses of reversible computations.
We deal with fixed points in SRL, i.e. we are interested in the problem of deciding if there is a tuple of initial register values of a given program P that remains unaltered after the execution of P. First, we show that the existence of fixed points in SRL is undecidable and complete in Σ_1^0. Second, we show that such problem remains undecidable even when the number of registers mentioned by the program P is limited to 12. Moreover, if we restrict to linear programs of SRL, i.e. to those programs where different registers control nested loops, then the problem is undecidable already when programs mentions 3712 registers.

1 Introduction

Loop languages are an important sub-class of `while` programming languages (see for instance [3,6,15]) which characterize the class of primitive recursive functions [18,19]. The language of [18,19] is the starting point to design the reversible languages SRL and ESRL defined in [17] (see also [24]).

Each register contains a value in $\mathbb{Z}$; each program mentioning n registers defines a bijection $\mathbb{Z}^n \to \mathbb{Z}^n$. Consider the following SRL programs:

$$\begin{array}{llll} \mathsf{P0}: & \mathsf{for}\, x(\mathsf{inc}\, r) & \mathsf{P2}: & \mathsf{for}\, y(\mathsf{for}\, x(\mathsf{inc}\, r)) \\ \mathsf{P1}: & \mathsf{for}\, x(\mathsf{dec}\, r) & \mathsf{P3}: & \mathsf{for}\, x(\mathsf{for}\, x(\mathsf{dec}\, r)) \end{array}$$

The program P0 mentions the two registers x and r. If the initial value of x is strictly positive, P0 eventually *increments* the initial value of r by (the initial value of) x. If the initial value of x is negative, P0 eventually *decrements* the initial value of r by x. So, in any case, P0 is the inverse of P1. I.e., running P1 after P0, written as P0;P1, we compute the identity on the tuple (x, r). It is worth to note that SRL ensures the reversibility by forbidding the modification of a register driving a loop (as x in P0) in the loop-body. P2 iterates P1. The final value of r is its initial value plus the product xy. P2 is linear because no register driving a loop is mentioned in its body. Clearly, P3 is *non*-linear exactly

for the opposite reason. So, the final value of r that P3 yields is its initial value plus the square x^2. The language ESRL extends SRL with the instruction that exchanges the contents of two registers.

The examples underline the three main aspects that differentiate both SRL and ESRL from non-reversible (classic) Loop languages and which ensure they only compute bijections: (i) a program register may contain any, possibly negative, integer; (ii) every elementary operation is reversible and (iii) a projection, like ignoring registers (at output) or eliminating its content, is not allowed. This is similar, but not completely identical to what happens in Quantum Mechanics. General forms of "cloning" — copying the content of a register to another — and erasing are not possible. Only some limited versions of them exist. For example, the fan-out in [28] and the cloning of orthogonal states in [21, page 530] are limited cloning.

Among the languages whose programs represent exactly reversible functions, SRL and ESRL [17,23,22] are probably the simplest ones, but they are far from being trivial. For SRL and similar languages, finding undecidable decision problems and proving their undecidability is harder than for Loop languages exactly because SRL is extremely simple.

In this paper, we focus on the *fixed point problem* (shortened to "fixpoint problem") for SRL: "given a program P, is there an input tuple which is not modified by the execution of P?". We prove that the fixpoint problem is undecidable and complete in Σ_1^0. In particular, the undecidability shows up when the programs mention 12 registers and, in the case of linear programs, when they mention 3712 registers.

We present a reduction from Hilbert's Tenth Problem (shortened in HTP) to the fixpoint problem. Thus, an eventual algorithm that solves the fixpoint problem would give us an algorithm which, for any given Diophantine equation, decides whether the equation has a solution with all unknowns taking integer values (see [16] for more details). But such algorithm does not exist.

We focus on problems related to HTP which look promising for dealing with "*the*" problem we may ask on a programming language: "Given P and Q in SRL, are they equivalent?" This question is showed to be undecidable for straight-line programs by means of a proof that relies on HTP [8,10,9]. Since straight-line programs can be viewed as a sort of trace-semantics for programs in SRL, we conjecture that some results on such programs can also hold for SRL.

Contents. Section 2, page 2 contains the background necessary to understand this work. The main results of this paper are Theorem 1 (page 7, Section 3) and Theorem 3 (page 10, Section 4). Finally, in Section 5 (page 11) we present some conclusions and open problems.

2 Preliminaries

The sets of positive integers, non-negative integers, integers, and rational numbers are denoted by $\mathbb{N}^+$, $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$, respectively.

2.1 The language SRL

SRL is a language whose programs work on registers that store values of $\mathbb{Z}$. Each program P of SRL defines a bijection $\mathbb{Z}^n \to \mathbb{Z}^n$, where $n \in \mathbb{N}^+$ is the number of registers that occur in the definition of P. Such n registers are said to be *mentioned* or *used* in P. The program P^{-1} is the inverse of P and computes the inverse bijection (below we explain how to generate P^{-1} in an effective manner).

The *syntax* of SRL-programs relies on four possible constructions: (i) $\mathsf{inc}\, x$ increments the content of the register x by 1; (ii) $\mathsf{dec}\, x$ decrements the content of the register x by 1; (iii) $\mathsf{P}_0;\mathsf{P}_1$ is the sequential composition of 2 programs; and, (iv) $\mathsf{for}\, r(\mathsf{P})$ iterates P as many times as the initial contents of r if that value is positive; otherwise, if r contains a negative value, then the iteration is on P^{-1}. Moreover, the constraint that (iv) must satisfy is that neither $\mathsf{inc}\, r$ nor $\mathsf{dec}\, r$ occur in P, leaving the content of r unchanged.

An SRL-program is *linear* if in every instruction of the form $\mathsf{for}\, r(\mathsf{P})$, the program P does not mention the register r. That is, P can contain neither $\mathsf{inc}\, r$, nor $\mathsf{dec}\, r$, nor $\mathsf{for}\, r(\mathsf{P}')$ instructions.

The inverse of an SRL-program is generated by transforming $\mathsf{inc}\, x$, $\mathsf{dec}\, x$, $\mathsf{P}_0;\mathsf{P}_1$ and $\mathsf{for}\, r(\mathsf{P})$ in $\mathsf{dec}\, x$, $\mathsf{inc}\, x$, ${\mathsf{P}_1}^{-1};{\mathsf{P}_0}^{-1}$ and $\mathsf{for}\, r(\mathsf{P}^{-1})$, respectively. For more information on SRL and its extensions, as well as results related with that language, consult for instance [17] and [22].

Example 1. Let P be the program $\mathsf{for}\, r(\mathsf{for}\, b(\mathsf{inc}\, a);\mathsf{for}\, a(\mathsf{inc}\, b))$. Let $\mathsf{a} = 0$, $\mathsf{b} = 1$, and $\mathsf{r} = n$ be the initial values of the corresponding registers. The final value of a is F_{2n}, i.e. the Fibonacci number with index $2n$ where we take F_k defined for every $k \in \mathbb{Z}$. (the extension of the definition of F_k to negative values of k is straightforward). We remark that F_{2n} is exponential in n. Moreover, the inverse of P is $\mathsf{for}\, r(\mathsf{for}\, a(\mathsf{dec}\, b);\mathsf{for}\, b(\mathsf{dec}\, a))$.

Definition 1 (Fixpoint of a program). *Let* P *be a* SRL *program that mentions n registers $\overline{x} = \langle x_1, \ldots, x_n \rangle$. Let* $\overline{\mathsf{x}}$*, a tuple of n integers, denote the initial values of $\overline{x}$. Let* $\mathsf{P}(\overline{\mathsf{x}})$ *be the tuple of the corresponding final contents after* P *is executed with the initial values* $\overline{\mathsf{x}}$*. If* $\mathsf{P}(\overline{\mathsf{x}}) = \overline{\mathsf{x}}$*, then the tuple* $\overline{\mathsf{x}}$ *is a* fixpoint *of* P*.*

2.2 Decision problems and reductions

Let A be a predicate, i.e. a function from a set to truth values. A decision problem related to A is an effective procedure able to say when the answer to "$A(x)$?" is YES, for every x. Let B be also a decision problem. A (many-one) reduction of A to B, written $A \leq B$, is an effective function f which maps every instance x of A to an instance $f(x)$ of B such that the answer to "$A(x)$?" is YES iff the answer to "$B(f(x))$?" is YES. The relation $\leq$ is transitive. If $A \leq B$ and A is undecidable, then B is also undecidable [25,5,14,4,1,2,7,26].

Definition 2 (FIXPOINT-decision problem). *Let* P *be a* SRL*-program then the* FIXPOINT*-decision problem is*

$$\mathsf{FIXPOINT}(\mathsf{P}) \equiv \text{"}\textit{Does}\ \mathsf{P}\ \textit{have a fixpoint?}\text{"}.$$

More explicitly, FIXPOINT(P) *is equivalent to: "Given a* SRL *program* P*, is there a tuple* $\bar{x}$ *such that* $\mathsf{P}(\bar{x}) = \bar{x}$ *?".*

Moreover, we write l-FIXPOINT(P) *whenever* P *is linear. We mean to ask: "Given a* linear SRL *program* P*, is there a tuple* $\bar{x}$ *such that* $\mathsf{P}(\bar{x}) = \bar{x}$*?".*

Roughly speaking, this paper is about reducing Hilbert's Tenth Problem [16,12,27,20], also known as the "Diophantine decision problem", to FIXPOINT(P) and to l-FIXPOINT(P). We shall find a constant c such that, if the number of registers mentioned by P does not exceed c, the problem FIXPOINT(P) is undecidable. We also prove analogous results for the problem l-FIXPOINT(P), that is, when P is linear.

2.3 Polynomials

Let $p(x_1, \ldots, x_n)$ be an integer polynomial, i.e. a polynomial with integer coefficients and unknowns $x_1, \ldots, x_n$. A polynomial is in *normal form* if: (i) it is a sum of monomials; (ii) each monomial has form $cx_1^{e_1}x_2^{e_2}\ldots x_k^{e_k}$ where $c \neq 0$ is a constant, the unknowns $x_1, \ldots, x_k$ are pairwise distinct, and $e_i \geq 1$, for every $1 \leq i \leq k$; (iii) no two monomials have the same unknowns with the same exponents.

For instance, $3x^2y+y^2-yx^2+5$ *is not* in normal form. The *unique* normal form of a given $p(x_1, \ldots, x_n)$ can be obtained by sorting in lexicographic order: (i) the unknowns within each monomial, and (ii) the monomials of the polynomial. For example, the normal form of $3xy(z+y^2)+2z-3y^3x$ is $3xyz+2z$. We shall often omit exponents equal to 1 as well as monomial coefficients c equal to 1. The null polynomial is denoted by 0.

Definition 3. *For any polynomial* $p(x_1, \ldots, x_n)$ *in normal form, we identify:*

$u(p)$	*the number of unknowns*
$\mathsf{deg}(x_i)$	*the maximum degree of the unknown* x_i
$\mathsf{deg}(p)$	*the maximum among* $\mathsf{deg}(x_1) \ldots \mathsf{deg}(x_n)$
$d(p)$	*the maximum degree of a monomial of* p*, or "the degree of the polynomial"* p*.*

Example 2. Let $p(x, y, z) = xy^3z + 2xyz^3 - z^4$. The polynomial p has three unknowns x, y and z, so $u(p) = 3$. The unknowns occur with various degrees in distinct monomials, but the maximal exponents are $\mathsf{deg}(x) = 1$, $\mathsf{deg}(y) = 3$ and $\mathsf{deg}(z) = 4$. So $4 = \mathsf{deg}(p) = \mathsf{deg}(z)$. Finally, $d(p) = 5$, given by summing up all the exponents of $2xyz^3$.

2.4 Hilbert's Tenth Problem and the Diophantine equation problem.

Let $\mathcal{D}$ be one among $\mathbb{N}^+$, $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{Q}$. As said, we shorten Hilbert's Tenth Problem by HTP. Its question about a given polynomial $p(x_1, \ldots, x_n)$ is:

$$\mathsf{HTP}(\mathcal{D})(p) \equiv \text{“ Does a tuple } \mathsf{x}_1, \ldots, \mathsf{x}_n \text{ of values in } \mathcal{D} \text{ exist such that } p(\mathsf{x}_1, \ldots, \mathsf{x}_n) = 0? \text{”}$$

$\mathsf{HTP}(\mathcal{D})$ can be straightforwardly generalized to questions on a *set* of integer polynomials $\{p_1(x_1, \ldots, x_n), \ldots, p_m(x_1, \ldots, x_n)\}$ as follows:

$$\mathsf{HTP}(\mathcal{D})(p_1, \ldots, p_m) \equiv \text{“ Does a tuple } \mathsf{x}_1, \ldots, \mathsf{x}_n \text{ of values in } \mathcal{D} \text{ exist such that } p_i(\mathsf{x}_1, \ldots, \mathsf{x}_n) = 0, \text{ for every } 1 \leq i \leq m \text{ ?”}$$

The difficulty of solving $\mathsf{HTP}(\mathcal{D})$ depends on $\mathcal{D}$ [16, (1.3.1)],[13,12,20]. If a given $\mathsf{HTP}(\mathcal{D})$ problem is undecidable, we can look for the least number of unknowns and the least degrees which suffice to keep $\mathsf{HTP}(\mathcal{D})$ undecidable. To our knowledge [27] contains the latest (very recent) improvements concerning the bounds on that number of unknowns. I.e., for a single equation, $\mathsf{HTP}(\mathbb{Z})$ is undecidable for polynomials with at least 11 unknowns.

Definition 4. *If the system of equations consists of a single equation, then* $\mathsf{Equ}(\mathcal{D})$ *is our favourite notations for* $\mathsf{HTP}(\mathcal{D})$*. Instead, the notation* $\mathsf{Sys}(\mathcal{D})$ *underlines that the given instance of* $\mathsf{HTP}(\mathcal{D})$ *has more than one Diophantine equation.*

3 Properties of the fixpoint problem for SRL

The first part of this section focuses on proving the next theorem.

Theorem 1. *The problem* FIXPOINT *as in Definition 2 (page 3), is undecidable and complete in* Σ_1^0.

Proving Theorem 1

The strategy is to reduce $\mathsf{Equ}(\mathbb{Z})$, i.e. $\mathsf{HTP}(\mathbb{Z})$ that deals with a single Diophantine equation, to FIXPOINT. We show how by means of a significant running example. The example illustrates how to map a polynomial to a program of SRL. The general case will follow easily. Let $p(x, y)$ be the polynomial:

$$p(x, y) = 2x^3y^2 - xy^2 + 2. \tag{1}$$

We map $p(x, y)$ to the program $\mathsf{P}(p)$, namely the sequential composition of the following three lines of code:

$$\mathsf{for}\ x(\mathsf{for}\ x(\mathsf{for}\ x(\mathsf{for}\ y(\mathsf{for}\ y(\mathsf{inc}\ s;\ \mathsf{inc}\ s))))); \tag{A}$$

$$\mathsf{for}\ x(\mathsf{for}\ y(\mathsf{for}\ y(\mathsf{dec}\ s))); \tag{B}$$

$$\mathsf{inc}\ s;\ \mathsf{inc}\ s. \tag{C}$$

Line (A) updates the content of s in accordance with the assignment:

$$s \leftarrow s + 2x^3y^2. \tag{2}$$

The factor x^3 follows from the three nested iterations driven by x. Analogously, the two nested iterations driven by y yield the factor y^2. Factor 2 comes from "$\mathsf{inc}\, s$; $\mathsf{inc}\, s$". Under the same idea, lines (B) and (C) produce:

$$s \leftarrow s - xy^2 \tag{3}$$

$$s \leftarrow s + 2, \tag{4}$$

respectively. The overall effect of $\mathsf{P}(p)$ is thus to update the initial value of s with the value of (1), for every fixed value of x and y.

Concerning the general case, for any polynomial p, the corresponding $\mathsf{P}(p)$ uses as many registers as the number of unknowns of p, plus one register s which is incremented by the value of $p(x, y)$.

Every monomial of p "is transformed in" blocks of iterations nested as many times as required to obtain the corresponding exponents. The nested blocks eventually operate on a program that only contains sequences of $\mathsf{inc}\, s$, or $\mathsf{dec}\, s$, which determine the constant multiplicative factor of the monomial.

We are now ready to comment on how "$\mathsf{P}(p)$ has a fixpoint, if, and only if, $p(x, y) = 0$ has a solution", following our running example.

"If $\mathsf{P}(p)$ has a fixpoint, then $p(x, y) = 0$ has a solution".
Let us assume that the tuple $(\mathsf{x}, \mathsf{y}, \mathsf{s})$ used as input for the registers (x, y, s) is a fixpoint of $\mathsf{P}(p)$. The execution of $\mathsf{P}(p)$ from $(\mathsf{x}, \mathsf{y}, \mathsf{s})$ — let us denote it as $\mathsf{P}(p)(\mathsf{x}, \mathsf{y}, \mathsf{s})$ — necessarily replaces the value $\mathsf{s} + 2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2$ for s. We can formalize the overall effect as follows:

$$\mathsf{P}(p)(\mathsf{x}, \mathsf{y}, \mathsf{s}) = (\mathsf{x}, \mathsf{y}, \mathsf{s} + 2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2).$$

Being $(\mathsf{x}, \mathsf{y}, \mathsf{s})$ a fixpoint of $\mathsf{P}(p)$, we must have $\mathsf{s} = \mathsf{s} + 2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2$ which is possible only if $2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2 = 0 = p(\mathsf{x}, \mathsf{y})$. Thus (x, y) is a solution of $p(x, y)=0$. ⊠

"If $p(x, y) = 0$ has a solution, then $\mathsf{P}(p)$ has a fixpoint".
Let us assume that the tuple (x, y) is a solution of $p(x, y) = 0$. Execute $\mathsf{P}(p)(\mathsf{x}, \mathsf{y}, \mathsf{s})$, i.e. the program $\mathsf{P}(p)$ whose variables (x, y, s) assume the initial values x, y and s, respectively, with some arbitrarily fixed value s. Then:

$$\begin{aligned}\mathsf{P}(p)(\mathsf{x}, \mathsf{y}, \mathsf{s}) &= (\mathsf{x}, \mathsf{y}, \mathsf{s} + 2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2) = (\mathsf{x}, \mathsf{y}, \mathsf{s} + p(\mathsf{x}, \mathsf{y})) \\ &= (\mathsf{x}, \mathsf{y}, \mathsf{s} + 0) = (\mathsf{x}, \mathsf{y}, \mathsf{s}),\end{aligned}$$

which means that $(\mathsf{x}, \mathsf{y}, \mathsf{s})$ is a fixpoint of $\mathsf{P}(p)$. Since $\mathsf{Equ}(\mathbb{Z})$ is Σ_1^0-complete [25], also FIXPOINT is, and this concludes the justification to Theorem 1. ⊠

We now proceed to the strengthening of Theorem 1. In [27, Part 1: Corollary 1.1 (p. 5)] it is stated that the problem $\mathsf{Equ}(\mathbb{Z})$ keeps being undecidable even

if the number of unknowns of the polynomials of Equ($\mathbb{Z}$) does not exceed 11. Since we know how to reduce Equ($\mathbb{Z}$) to FIXPOINT in the general case, and in this reduction the number of program registers equals the number of unknowns plus 1 ($2 + 1 = 3$ in our running example), we can improve our result.

Corollary 1. FIXPOINT *is undecidable and complete in* Σ_1^0 *already when* FIXPOINT *contain programs that mention 12 registers.*

4 Properties of the linear fixpoint problem for SRL

Let us recall from Section 2 that a program P of SRL is *linear*, if nested iterations driven by the same register are forbidden.

Definition 5 (Linear SRL programs). *Let* $n \in \mathbb{N}$. *Then* l-SRL(n) *denotes the subset of* linear SRL *programs that use no more than* n *registers.*

The first part of this section focuses on the next theorem.

Theorem 2. *The problem* l-FIXPOINT *is undecidable and complete in* Σ_1^0.

It corresponds to Theorem 1 (page 5). The proof strategy reduces Equ($\mathbb{Z}$) to l-FIXPOINT. The proof is somewhat more complex than that of Theorem 1 because the number of registers in a linear program of SRL that represents the polynomial p of a Diophantine equation depends on the number of unknowns of p *and* on its monomial degree (Definition 3, page 3).

Corollary 2. *Let* p *be a polynomial which is an instance of* Equ($\mathbb{Z}$). *Let* $u(p)$ *and* $d(p)$ *be respectively the number of unknowns and the monomial degree of* p. *It is possible to reduce the problem* $p = 0$ *to the fixpoint problem of a program* $P(p)$ *that belongs to* l-SRL($2u(p)d(p)$).

Proving Theorem 2

We define a reduction from Equ($\mathbb{Z}$) — single equation Diophantine problem over $\mathbb{Z}$ — to l-FIXPOINT, the fixpoint problem on linear SRL.

Like in the proof of Theorem 1 we illustrate how the proof works by means of an example. The general case will follow intuitively and, in fact, will be summarized by Corollary 2. Once again, let:

$$p(x, y) = 2x^3y^2 - xy^2 + 2. \tag{5}$$

be the polynomial already used to illustrate how the proof of Theorem 1 works.

The program $\mathsf{P}(p)$ which $p(x,y)$ maps to, is the sequential composition of the following lines of code:

Program	Comment	
$\mathsf{for}\, x(\mathsf{inc}\, a_1);\ \mathsf{for}\, x_1(\mathsf{dec}\, a_1);$	$a_1 \leftarrow a_1 + \mathsf{x} - \mathsf{x}_1$	(a)
$\mathsf{for}\, x(\mathsf{inc}\, a_2);\ \mathsf{for}\, x_2(\mathsf{dec}\, a_2);$	$a_2 \leftarrow a_2 + \mathsf{x} - \mathsf{x}_2$	(b)
$\mathsf{for}\, y(\mathsf{inc}\, b_1);\ \mathsf{for}\, y_1(\mathsf{dec}\, b_1);$	$b_1 \leftarrow b_1 + \mathsf{y} - \mathsf{y}_1$	(c)
$\mathsf{for}\, x(\mathsf{for}\, x_1(\mathsf{for}\, x_2(\mathsf{for}\, y(\mathsf{for}\, y_1(\mathsf{inc}\, s;\ \mathsf{inc}\, s)))));$	$s \leftarrow s + 2\mathsf{x}\mathsf{x}_1\mathsf{x}_2\mathsf{y}\mathsf{y}_1$	(d)
$\mathsf{for}\, x(\mathsf{for}\, y(\mathsf{for}\, y_1(\mathsf{dec}\, s)));$	$s \leftarrow s - \mathsf{x}\mathsf{y}\mathsf{y}_1$	(e)
$\mathsf{inc}\, s;\ \mathsf{inc}\, s$	$s \leftarrow s + 2.$	(f)

The whole sequential composition belongs to l-SRL(9). The registers used are x, x_1, x_2, a_1, a_2, y, y_1, b_1, and s. By x, x_1, x_2, y, and y_1 we denote the initial values of the homonyms registers. We remark that the code at lines (d) and (e) linearize in some sense, the code (A) and (B) of the non linear case at page 5. We now comment on why "$p(x,y) = 0$ has a solution if, and only if, the linear program $\mathsf{P}(p)$ has a fixpoint" holds, by following our running example.

"If $p(x,y) = 0$ has a solution, then $\mathsf{P}(p)$ has a fixpoint".
Let us assume that the tuple (x,y) of instances of (x,y) is a solution of $p(x,y) = 0$.

Execute $\mathsf{P}(p)(\mathsf{x},\mathsf{x},\mathsf{x},\mathsf{a}_1,\mathsf{a}_2,\mathsf{y},\mathsf{y},\mathsf{b}_1,\mathsf{s})$, i.e. the program $\mathsf{P}(p)$ whose variables x, x_1, x_2, a_1, a_2, y, y_1, b_1 and s assume their initial values x, x, x, a_1, a_2, y, y, b_1, and s, respectively, for some arbitrarily fixed a_1, a_2, b_1, and s. Then:

$$\begin{aligned}
&\mathsf{P}(p)(\mathsf{x},\mathsf{x},\mathsf{x},\mathsf{a}_1,\mathsf{a}_2,\mathsf{y},\mathsf{y},\mathsf{b}_1,\mathsf{s}) = \\
&= (\mathsf{x},\mathsf{x},\mathsf{x},\mathsf{a}_1 + \mathsf{x} - \mathsf{x},\mathsf{a}_2 + \mathsf{x} - \mathsf{x},\mathsf{y},\mathsf{y},\mathsf{b}_1 + \mathsf{y} - \mathsf{y},\mathsf{s} + 2\mathsf{x}\mathsf{x}\mathsf{x}\mathsf{y}\mathsf{y} - \mathsf{x}\mathsf{y}\mathsf{y} + 2) \qquad (6)\\
&= (\mathsf{x},\mathsf{x},\mathsf{x},\mathsf{a}_1,\mathsf{a}_2,\mathsf{y},\mathsf{y},\mathsf{b}_1,\mathsf{s} + 2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2)\\
&= (\mathsf{x},\mathsf{x},\mathsf{x},\mathsf{a}_1,\mathsf{a}_2,\mathsf{y},\mathsf{y},\mathsf{b}_1,\mathsf{s}) \qquad (7)
\end{aligned}$$

where (6) holds because x, x_1, x_2, y and y_1 never change while a_1, a_2, b_1 and s get updated in accordance with the behavior of $\mathsf{P}(p)$. Instead, (7) is true because $0 = 2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2$ by assumption. I.e. the linear program $\mathsf{P}(p)$ of SRL has a fixpoint. ⊠

"If P has a fixpoint, then $p(x,y) = 0$ has a solution".
Let $(\mathsf{x},\mathsf{x}_1,\mathsf{x}_2,\mathsf{a}_1,\mathsf{a}_2,\mathsf{y},\mathsf{y}_1,\mathsf{b}_1,\mathsf{s})$, inputs for the registers $(x,x_1,x_2,a_1,a_2,y,y_1,b_1,s)$, be a fixpoint of $\mathsf{P}(p)$. Let $\mathsf{P}(p)(\mathsf{x},\mathsf{x}_1,\mathsf{x}_2,\mathsf{a}_1,\mathsf{a}_2,\mathsf{y},\mathsf{y}_1,\mathsf{b}_1,\mathsf{s})$ denote the execution of $\mathsf{P}(p)$ from $(\mathsf{x},\mathsf{x}_1,\mathsf{x}_2,\mathsf{a}_1,\mathsf{a}_2,\mathsf{y},\mathsf{y}_1,\mathsf{b}_1,\mathsf{s})$. By following what happens to the values of $x,x_1,x_2,a_1,a_2,y,y_1,b_1$ and s from (a) through (f) in the program above, we obtain:

$$\begin{aligned}
&\mathsf{P}(p)(\mathsf{x},\mathsf{x}_1,\mathsf{x}_2,\mathsf{a}_1,\mathsf{a}_2,\mathsf{y},\mathsf{y}_1,\mathsf{b}_1,\mathsf{s}) = \\
&\quad (\mathsf{x},\mathsf{x}_1,\mathsf{x}_2,\mathsf{a}_1 + \mathsf{x} - \mathsf{x}_1,\mathsf{a}_2 + \mathsf{x} - \mathsf{x}_2,\mathsf{y},\mathsf{y}_1,\mathsf{b}_1 + \mathsf{y} - \mathsf{y}_1,\mathsf{s} + 2\mathsf{x}\mathsf{x}_1\mathsf{x}_2\mathsf{y}\mathsf{y}_1 - \mathsf{x}\mathsf{y}\mathsf{y}_1 + 2)
\end{aligned}$$

where $\mathsf{x}, \mathsf{x}_1, \mathsf{x}_2, \mathsf{y}$, and y_1 remain constant because they only drive iterations. Being $(\mathsf{x}, \mathsf{x}_1, \mathsf{x}_2, \mathsf{a}_1, \mathsf{a}_2, \mathsf{y}, \mathsf{y}_1, \mathsf{b}_1, \mathsf{s})$ a fixpoint of $\mathsf{P}(p)$, we must satisfy the following series of constraints:

$$\mathsf{a}_1 = \mathsf{a}_1 + \mathsf{x} - \mathsf{x}_1 \tag{8}$$

$$\mathsf{a}_2 = \mathsf{a}_2 + \mathsf{x} - \mathsf{x}_2 \tag{9}$$

$$\mathsf{b}_1 = \mathsf{b}_1 + \mathsf{y} - \mathsf{y}_1 \tag{10}$$

$$\mathsf{s} = \mathsf{s} + 2\mathsf{x}\mathsf{x}_1\mathsf{x}_2\mathsf{y}\mathsf{y}_1 - \mathsf{x}\mathsf{y}\mathsf{y}_1 + 2. \tag{11}$$

As the left and right-hand side of (8), (9) and (10) must be equal, forcefully:

$$\mathsf{x}_1 = \mathsf{x} \qquad \mathsf{x}_2 = \mathsf{x} \qquad \mathsf{y}_1 = \mathsf{y}.$$

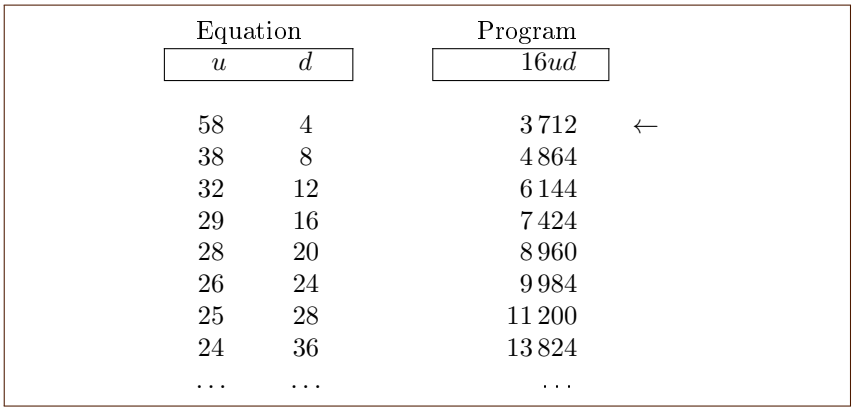
Distributing them inside (11) yields the constraint that: $\mathsf{s} = \mathsf{s} + 2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2$ which we can satisfy only if $0 = 2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2 = p(\mathsf{x}, \mathsf{y})$, i.e. $p(\mathsf{x}, \mathsf{y})$ has a solution. Since $\mathsf{Equ}(\mathbb{Z})$ is Σ_1^0-complete [25] l-FIXPOINT is and this concludes the justification of Theorem 2. ⊠

4.1 Proving Corollary 2

We want to exploit our running example to estimate a reasonably tight bound on the amount of registers that $\mathsf{P}(p)$ uses, where $\mathsf{P}(p)$ is the program defined in the above reduction. Looking at $\mathsf{P}(p)$ as the sequential composition of (a) through (f) we can state the following.

- $\mathsf{P}(p)$ must have one register per every unknown of $p(x, y)$ plus one register which stores the value of $p(x, y)$. So we have the three registers x, y and s. For a generic polynomial q, this means that $\mathsf{P}(q)$ needs $u(q) + 1$ registers.
- $\mathsf{P}(p)$ also uses a pair x_1, x_2 of registers. All together, x, x_1 and x_2 compute x^3 for any value x that we assign to the unknown x of the polynomial p. Equivalently, x_1 and x_2 stand for the linearization of x inside $\mathsf{P}(p)$ and they are as many as the degree $d(x)$ of x minus 1 because x counts for one occurrence of itself, of course. The meaning of y_1 is analogous. Together with y it counts for the two occurrences of y in $\mathsf{P}(p)$ that we use for the unknown y of p to obtain y^2. For a generic polynomial q, this means that $\mathsf{P}(q)$ also needs $\left[\sum_{x \text{ unknown of } q} d(x)\right] - 1$ registers.
- Finally, $\mathsf{P}(p)$ also uses a pair of registers a_1, a_2. They are the key ingredient that let the reduction work. Specifically, every a_i assures that the corresponding x_i is in fact a copy of x. The meaning of b_1 (there is an unique b_i in our running example) is analogous. It ensures that y_1 contains the same value as y. For a generic polynomial q, this boils down to have as many registers as the number of *copies* of the unknowns of q. So $\mathsf{P}(q)$ needs $\left[\sum_{x \text{ unknown of } q} d(x)\right] - 1$ further registers.

Summarizing, given an instance $q(x_1, \ldots, x_m) = 0$ of $\mathsf{Equ}(\mathbb{Z})$, the linear program $\mathsf{P}(q)$ of SRL uses as many as $u(p)+1+2u(p)(d(p)-1)$ registers, which is a number bounded by $u(p)+1+2u(p)(d(p)-1) = u(p)(2d(p)-1)+1 \leq 2u(p)d(p)$. So, $\mathsf{P}(q) \in l\text{-}\mathsf{SRL}(\,2u(p)d(p)\,)$. ⊠



Equation		Program	
u	d	$16ud$	
58	4	3 712	←
38	8	4 864	
32	12	6 144	
29	16	7 424	
28	20	8 960	
26	24	9 984	
25	28	11 200	
24	36	13 824	
...	...	...	

Fig. 1. The first two columns are from [12, page 861]. For each row they state that the $\mathsf{Equ}(\mathbb{N}^+)$ sub-problem that asks for the existence of zeros of a Diophantine equations over $\mathbb{N}^+$ (with coefficients in $\mathbb{Z}$) *with no more than u unknowns and monomial degree not exceeding d*, is undecidable. The third column expresses that l-FIXPOINT is undecidable even considering programs with no more than $16ud$ registers only. Thus, from the data in [12], the best upper bound on the "number of registers sufficient for undecidability" is 3 712.

4.2 Strengthening Theorem 2

Theorem 2 can be strengthened like Theorem 1.

Theorem 3. *The problem l-*FIXPOINT *is undecidable and complete in Σ_1^0 for linear programs in l-*SRL(3712)*.*

The proof of Theorem 3 relies on two results of [12]. The first result presents an universal system of Diophantine equations which includes the *parameters* x, z, u and y and 28 unknowns that take values in $\mathbb{N}^+$ [12, Theorem 3]. Fixing values for x, y, u and z yields a set $S_{x,y,u,z}$ of Diophantine equations, i.e. an instance of $\mathsf{Sys}(\mathbb{N}^+)$. Then, $S_{x,y,u,z}$ has a solution if and only if $x \in W_{\langle z,u,y\rangle}$ (notation of [12]), where $W_{\langle z,u,y\rangle}$ is a system of equations which we can think of as much expressive as a universal Turing machine. The system $S_{x,y,u,z}$ can be transformed into a single Diophantine equation $s_{x,y,u,z}$ over $\mathbb{N}^+$ whose polynomial has monomial degree 4 and 58 unknowns, besides the above x, y, u and z. Therefore, $s_{x,y,u,z}$ belongs to $\mathsf{Equ}(\mathbb{N}^+)$.

The second result we need from [12] is Theorem 4. It lists a sequence of pairs (u, d) of natural numbers as in Figure 1. The decision problem $p = 0$ of $\mathsf{Equ}(\mathbb{N}^+)$ is undecidable if p belongs to the class of polynomials with no more than u unknowns and monomial degree not exceeding d. So, Theorem 4 of [12] directly implies that $s_{x,y,u,z}$ is undecidable and Σ_1^0-complete whenever the u and the d of the polynomial it involves occur in Figure 1.

Standard techniques [16] imply that we can reduce the instance $s_{x,y,u,z}$ of $\mathsf{Equ}(\mathbb{N}^+)$ to $s'_{x,y,u,z}$ of $\mathsf{Equ}(\mathbb{Z})$ so that the polynomial p of $s'_{x,y,u,z}$ has $4 \times 58 =$

232 unknowns and monomial degree equal to $2 \times 4 = 8$. Corollary 2 at page 7 applies to $s'_{x,y,u,z}$, so that that the number of registers that the linear program $\mathsf{P}(p)$ uses is $2 \times 232 \times 8 = 3712$. Being $s'_{x,y,u,z}$ undecidable implies that l-FIXPOINT is undecidable for $\mathsf{P}(p) \in l\text{-}\mathsf{SRL}(3712)$. ⊠

5 Final remarks

Among the non trivial total reversible programming languages, SRL is perhaps the simplest one. In this work we prove that a natural decision problem, namely "does a given SRL program P have a fixpoint $\overline{x}$?", that is, "$\exists \overline{x} : \mathsf{P}(\overline{x}) = \overline{x}$?", is undecidable. Given the simplicity of the problem and the limitations of the language SRL, this is a significant result because simple decision problems about very simple programming languages "tend to be" decidable.

It is interesting to remark that, while the undecidability of the Diophantine decision and similar problems can be proved by reducing the Turing machine (or universal register language) halting problem to the existence of a solution of a polynomial equation (see for instance [29, Section 5.5] and [11]), we proved the undecidability of the fixpoint problem for a very limited total reversible register language by reducing $\mathsf{Equ}(\mathbb{Z})$ to it.

We also established bounds on the number of registers which imply the undecidability: 12 registers are enough in the general case and 3 712 in the linear one.

We think that there are better upper bounds for that number of registers, perhaps for the general and, almost certainly, for the linear SRL programs. Several other open problems are suggested by this research. For instance, given a positive integer k, are there SRL programs having exactly k fixpoints? And also: is there a reduction $\mathsf{Equ}(\mathbb{Z}) \leq \mathsf{FIXPOINT}$ such that the number of roots of the polynomial is equal to the number of fixpoints of the corresponding SRL program?

References

1. Boolos, .G., Burgess, P.J., Jeffrey, C.R.: Computability and Logic. CUP (2007)
2. Cooper, S.B.: Computability Theory. CRC Press Company (2004)
3. Crolard, T., Lacas, S., Valarcher, P.: On the expressive power of the Loop language. Nordic Journal of Computing **13**(1-2), 46–57 (2006)
4. Cutland, N.: Computability: An Introduction to Recursive Function Theory. Cambridge University Press (1980)
5. Davis, M.: Computability and Unsolvability. Dover (1985)
6. Enderton, B.H.: Computability Theory: an Introduction to Recursion Theory. Academic Press, 1 edn. (2010)
7. Hermes, H.: Enumerability, Decidability, Computability. Springer-Verlag (1969)
8. Ibarra, O.H., Leininger, B.S.: Straight-line programs with one input variable. SIAM Journal on Computing **11**(1), 1–14 (1982)
9. Ibarra, O.H., Leininger, B.S.: On the simplification and equivalence problems for straight-line programs. J. ACM **30**(3), 641–656 (1983)

10. Ibarra, O.H., Leininger, B.S.: On the zero-inequivalence problem for loop programs. J. Comput. Syst. Sci. **26**(1), 47–64 (1983)
11. Jones, J.P., Matiyasevich, Y.: Register machine proof of the theorem on exponential diophantine representation of enumerable sets. Journal Symbolic Logic **49**, 818–829 (1984)
12. Jones, J.P.: Undecidable diophantine equations. Bull. Amer. Math. Soc. (N.S.) **3**(2), 859–862 (1980)
13. Jones, N.D.: Computability and Complexity: From a Programming Perspective. Foundations of Computing Series, MIT Press (1997)
14. Kleene, S.C.: Introduction to Metamathematics. North-Holland (1952), reprinted by Ishi press on 2009
15. Kristiansen, L., Niggl, K.H.: On the computational complexity of imperative programming languages. Theoretical Computer Science **318**(1-2), 139–161 (Jun 2004)
16. Matiyasevich, Y.: Hilbert's Tenth Problem. MIT Press (1993)
17. Matos, A.B.: Analysis of a simple reversible language. Theoretical Computer Science **290**(3), 2063–2074 (2003)
18. Meyer, A.R., Ritchie, D.M.: The complexity of loop programs. In: Proceedings of 22nd National Conference of the ACM. pp. 465–469. Association for Computing Machinery (1967)
19. Meyer, A.R., Ritchie, D.M.: Computational complexity and program structure. In: IBM Research Report RC 1817. IBM (1967)
20. Nathanson, M.B.: Additive Number Theory The Classical Bases. Springer Science & Business Media (1996)
21. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press, New York, NY, USA, 10th edn. (2011)
22. Paolini, L., Piccolo, M., Roversi, L.: A class of recursive permutations which is primitive recursive complete (2016), Submitted
23. Paolini, L., Piccolo, M., Roversi, L.: A class of reversible primitive recursive functions. Electronic Notes in Theoretical Computer Science **322**(18605), 227–242 (2016)
24. Perumalla, K.: Introduction to Reversible Computing. CRC Press (2014)
25. Rogers, H.: Theory of Recursive Functions and Effective Computability. MIT Press Cambridge, MA (1967)
26. Soare, R.I.: Turing Computability: Theory and Applications. Springer (2016), department of Mathematics, the University of Chicago
27. Sun, Z.W.: Further results on Hilbert's Tenth Problem (2017), https://arxiv.org/abs/1704.03504, Cornell University Library
28. Toffoli, T.: Reversible computing. In: de Bakker, J., van Leeuwen, J. (eds.) Automata, Languages and Programming. pp. 632–644. Springer Berlin Heidelberg, Berlin, Heidelberg (1980)
29. Yuri, M.: Hilbert's tenth problem: what was done and what is to be done. In: Denef, J., Lipshitz, L., Pheidas, T., Geel, J.V. (eds.) Hilbert's Tenth Problem: Relations with Arithmetic and Algebraic Geometry (Contemporary Mathematics, 270). Amer. Math. Soc. (2000)