The fixed point problem of a simple reversible language

The fixed point problem of a simple reversible language ☆

Armando B. Matos[a], Luca Paolini[b], Luca Roversi[b]

[a] *Departamento de Ciência de Computadores, Faculdade de Ciências, Universidade do Porto*
[b] *Università degli Studi di Torino, Dipartimento di Informatica, C.so Svizzera 185, 10149 Torino — Italy*

Abstract

SRL is a total programming language with distinctive features: (i) every program that mentions n registers defines a bijection $\mathbb{Z}^n \to \mathbb{Z}^n$, and (ii) the generation of the SRL-program that computes the inverse of that bijection can be automatic. Containing SRL a very essential set of commands, it is suitable for studying strengths and weaknesses of reversible computations.

We deal with the fixed points of SRL-programs. Given any SRL-program P, we are interested in the problem of deciding if a tuple of initial register values of P exists which remains unaltered after its execution. We show that the existence of fixed points in SRL is undecidable and complete in Σ_1^0. We show that such problem remains undecidable even when the number of registers mentioned by P is limited to 12. Moreover, if we restrict to the linear programs of SRL, i.e. to those programs where different registers control nested loops, then the problem is already undecidable for the class of SRL-programs that mention no more than 3712 registers. Last, we show that, except for trivial cases, finding if the number of fixed points has a given cardinality is also undecidable.

Keywords: Reversible Computing, Fixed points, Decidability

1. Introduction

The Loop languages are an important sub-class of the `while` programming languages [1, 2, 3, 4, 5]. Loop languages are the starting point to design the reversible languages SRL and ESRL, defined in [6] (see also [7]).

We start by recalling the distinctive features of SRL. Its programs operate on tuples of registers. Each register contains a value in $\mathbb{Z}$. So, a program

☆This paper revises and extends "The Fixed Point Problem for General and for Linear SRL Programs is Undecidable" in the Proceedings of the 16th Italian Conference on Theoretical Computer Science (Firenze, 9–11 September 2015).

Email addresses: armandobcm@yahoo.com (Armando B. Matos), luca.paolini@unito.it,paolini@di.unito.it (Luca Paolini), luca.roversi@unito.it,roversi@di.unito.it (Luca Roversi)

that mentions n registers defines a bijection $\mathbb{Z}^n \to \mathbb{Z}^n$. The class of functions that SRL defines is closed under inversion and the SRL-program that computes the inverse of another program can be automatically generated. Moreover, the instruction set of SRL is strictly contained in almost every current programming language. So, the properties of SRL hold for every programming language that contains it.

Computable bijections form a crucial subset of computable functions (see [6, 8, 9, 10]). In the classical setting, besides the theoretical motivations, studying SRL is interesting from truly pragmatic perspectives which range from lossless compression to cryptographic functions, passing through backtracking mechanisms. A survey is in [7]. Moreover, from a fundamental point of view, SRL supplies the core of total reversible programs that would allow to keep the energetic inefficiency of classical computation under control [11, 12, 13, 14, 6, 15, 9, 10, 7]. Last, but not least, reversible programs are also crucial in the quantum computing model under many perspectives [14, 16].

Consider the following SRL programs:

P0 :	$\mathsf{for}\, x(\mathsf{inc}\, r)$	P2 :	$\mathsf{for}\, y(\mathsf{for}\, x(\mathsf{inc}\, r))$
P1 :	$\mathsf{for}\, x(\mathsf{dec}\, r)$	P3 :	$\mathsf{for}\, x(\mathsf{for}\, x(\mathsf{dec}\, r))$

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. 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 when the value in y is positive. 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 minus x^2. The language ESRL extends SRL with the instruction that exchanges the contents of two registers.

The here above examples underline the three main aspects that differentiate both SRL and ESRL from non-reversible, i.e. 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; (iii) overwriting the content of a register is forbidden. 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. Instead, some limited version of cloning is compatible with reversible computing. Examples are the representation of the fan-out in [17] and the cloning of orthogonal states in [18, page 530].

Among the languages whose programs represent total reversible functions [6, 19, 20, 10], SRL and ESRL are the simplest ones, but they are far from being trivial. No language which is expressive enough to contain all the computable bijections can be effectively formalized. However, SRL, and similar languages,

are very expressive despite their terseness. It is an open question whether SRL is equally expressive as the languages presented in [19, 10]. The latter question and the decidability of many further questions about SRL are harder to answer than the corresponding ones about Loop languages.

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. Moreover, we tackle the decidability of the cardinality of fixed points of SRL-program. The main technical tool to obtain our results is Hilbert's Tenth Problem (shortened in HTP) which look promising for dealing with *the* problem we may ask about SRL, and, in fact, about every programming language: "Given P and Q, are they equivalent?"

1.1. Rationale

As outlined, the first author introduces SRL in the context of Loop languages. He keeps SRL syntactically so simple that SRL is a candidate to be *the* simplest, among reversible programming languages, that can express total invertible functions only. However, "simple" does not mean "trivial". Showing that the fixpoint problem for SRL is undecidable gives further evidence about the expressiveness of SRL that we already knew from [21] which proves that programs of SRL exist whose output can be as large as any primitive recursive function. In particular, for every $k \in \mathbb{N}$, there is a program P_k in SRL whose output value is given by $T_k(n) \geq 2 \uparrow^k n$, i.e. a tower of exponentials with k occurrences of 2s.

On the other side, the last two authors characterize classes of total invertible functions according to formalisms and ideas typical of Recursion Theory [19, 20, 10]. The long term goal is to prove that some analogous of Kleene Normal Form Theorem [22] holds for Turing complete classes of recursive reversible functions, like the one in [10].

In this work we focus on total reversible functions only. Eventually, we are interested to the relation between SRL and some of the classes of total reversible functions in [19, 20], the simplest one being RPP [20]. RPP is the class of Reversible Primitive Recursive functions. It is Primitive Recursive Complete, the proof being the one we may expect: a "compilation" of any primitive recursive function into an equivalent representation of RPP.

Both SRL and RPP rely almost on the same set of computational primitives, but with some relevant differences that we summarize:

- The linear use of every single input and output by a function of RPP is more explicit than in a corresponding program of SRL;
- Both languages include some ability to say if the value of an argument to a function is greater, equal or smaller than zero. RPP let the discriminating primitives explicitly available under the form of a conditional selection (namely, an `if-then-else` construct). On the contrary, SRL (stricly)

incorporates tests inside the iteration condition, as outlined by means of the informal description of P0, P1, P2 and P3 in the first lines of our introduction.

The here above, apparently small, differences have some consequences. On one side, proving that RPP is equivalent to the Primitive Recursive Functions is relatively simple. On the other, the direct proof that SRL and RPP are equivalent is surprisingly difficult. Despite it is obvious that SRL is a sub-language of RPP, so far, we have not been able to prove, or disprove, that the converse holds, i.e. that RPP is, or is not, a sub-language of SRL. More precisely, it is an open problem if the conditional instruction of RPP can be implemented in SRL. With a positive answer, SRL would be Primitive Recursive Complete and would become the candidate as the simplest base for a characterization of reversible computable functions.

The above question led us to explore many possible alternatives to compare the expressive power of SRL and RPP. We started to investigate the difficulty of computational problems for programming languages like SRL and RPP. The problem that we see as paradigmatic is the *equivalence* between two programs. Namely, let $\mathcal{L}$ be SRL or RPP. Is it possible to answer: "Given the programs P, Q in $\mathcal{L}$ and that use the same set of registers, does $\mathsf{P}(x) = \mathsf{Q}(x)$ hold, for every tuple x of initial values for the registers?" In the reversible computational models we can answer the previous question if we know how to answer the *identity problem*: "Given a program R in $\mathcal{L}$, does $\mathsf{R}(x) = x$ hold for every (tuple) x?". Obviously, if R is the serial composition of P and the *inverse* of Q we also answer the previous equivalence problem between P and Q.

The fixpoint problem seems related to the identity problem because the universal quantification over x becomes an existential one. Section 6 is about how the undecidability of the fixpoint problem that we prove in this work relates to the undecidability of the identity problem.

Contents. This paper revises and extends [23]. The current introduction widens the one in [23] by explaining the motivations in more detail. Section 2 introduces the formal background necessary to understand this work. Section 3 illustrates the fixed points undecidability of SRL programs. Section 4 illustrates the fixed points undecidability of linear programs. Section 5 is new. It shows that the cardinality of the set of fixed points of an SRL program is undecidable. Section 6 contains final discussions.

2. Technical 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 get 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 non-negative; 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 obtained 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 [6] and [20].

Example. 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}, 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 $A(x)$ be a predicate, i.e. a function from a set to truth values. The decision problem corresponding to A is: "given x (the instance) is $A(x)$ true?". 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, [22, 24, 25, 26, 27, 28, 29, 30].

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

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

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

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

Roughly speaking, this paper is about reducing Hilbert's Tenth Problem [31, 32, 33, 34], also known as the "Diophantine decision problem", to FIXPOINT(P) and to l-FIXPOINT(P). We shall find a constant c such that the problem FIXPOINT(P) is undecidable, even for the class of SRL programs that do not mention more than c registers.

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. An *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:*

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

Example. 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:

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

$\mathsf{HTP}(X)$ 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:

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

The difficulty of solving $\mathsf{HTP}(\mathcal{D})$ depends on $\mathcal{D}$, see [31, (1.3.1)], [35, 32, 34]. 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 [33] 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 the class of polynomials with at most 11 unknowns.

Definition 4. *If the system of equations consists of a single equation, then* $\mathsf{Equ}(\mathcal{D})$ *is our favourite notation 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. 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 5), 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 $\mathsf{P}(p)$ which is 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 multiplicative constant 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 [22], also FIXPOINT is, and this concludes the justification to Theorem 1. ⊠

We now proceed to the strengthening of Theorem 1. In [33, 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 $\mathsf{Equ}(\mathbb{Z})$ does not exceed 11. Since we know how to reduce $\mathsf{Equ}(\mathbb{Z})$ 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 at most 12 registers.*

4. 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*.*

This result corresponds to Theorem 1 (page 7). 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.

As 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 P(p) which $p(x, y)$ maps to, is the sequential composition of the following lines of code:

Program	Comment	
for x(inc a_1); for x_1(dec a_1);	$a_1 \leftarrow a_1 + \mathsf{x} - \mathsf{x}_1$	(a)
for x(inc a_2); for x_2(dec a_2);	$a_2 \leftarrow a_2 + \mathsf{x} - \mathsf{x}_2$	(b)
for y(inc b_1); for y_1(dec b_1);	$b_1 \leftarrow b_1 + \mathsf{y} - \mathsf{y}_1$	(c)
for x(for x_1(for x_2(for y(for y_1(inc s; inc s)))));	$s \leftarrow s + 2\mathsf{x}\mathsf{x}_1\mathsf{x}_2\mathsf{y}\mathsf{y}_1$	(d)
for x(for y(for y_1(dec s)));	$s \leftarrow s - \mathsf{x}\mathsf{y}\mathsf{y}_1$	(e)
inc s; 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 7. 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 have 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:

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

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 $\mathsf{P}(p)$. On the other hand, (7) is true because $0 = 2\mathsf{x}^3\mathsf{y}^2 - \mathsf{x}\mathsf{y}^2 + 2$ by assumption. Thus, 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:

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

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:

$$\mathsf{a}_1 = \mathsf{a}_1 + \mathsf{x} - \mathsf{x}_1 \qquad (8)$$
$$\mathsf{a}_2 = \mathsf{a}_2 + \mathsf{x} - \mathsf{x}_2 \qquad (9)$$
$$\mathsf{b}_1 = \mathsf{b}_1 + \mathsf{y} - \mathsf{y}_1 \qquad (10)$$
$$\mathsf{s} \ = \mathsf{s} + 2\mathsf{x}\mathsf{x}_1\mathsf{x}_2\mathsf{y}\mathsf{y}_1 - \mathsf{x}\mathsf{y}\mathsf{y}_1 + 2. \qquad (11)$$

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 [22], 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 for 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 $\sum_{x,\text{ unknown of } q}[d(x) - 1]$ 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 $\sum_{x,\text{ unknown of } q}[d(x) - 1]$ 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 can be bounded as

$$u(p) + 1 + 2u(p)(d(p) - 1) = u(p)(2d(p) - 1) + 1 \leq 2u(p)d(p).$$

So, $\mathsf{P}(q) \in l\text{-SRL}(\,2u(p)d(p)\,)$. ⊠

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).

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	
...	...	...	

Figure 1: The first two columns are from [32, 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 [32], the best upper bound on the "number of registers sufficient for undecidability" is 3 712.

The proof of Theorem 3 relies on two results of [32]. 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 $\mathbb{N}^+$ [32, 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 [32]), where $W_{\langle z,u,y\rangle}$ is a system of equations which we can think of as 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 is [32, 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 even if the instance p belongs to the class of polynomials with no more than u unknowns and monomial degree not exceeding d. So, Theorem 4 of [32] 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 [31] 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 9 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. Cardinality of the set of fixpoints

In this section we study some problems related with the number of fixpoints that a SRL program can have. The results we obtain are a consequence of a new reduction of Hilbert's Tenth problem to the SRL fixpoint problem.

Let $p(x_1, \ldots, x_n) = 0$ be a Diophantine equation. If we had to summarize the effect of the corresponding program $\mathsf{P}(p)$ in Section 3, we would write:

$$\mathsf{P}(p)(x_1, \ldots, x_n, s) = (x_1, \ldots, x_n, s + p(x_1, \ldots, x_n)) \ . \tag{12}$$

If $\mathsf{x}_1, \ldots, \mathsf{x}_n$ is a solution of $p(x_1, \ldots, x_n) = 0$, then infinitely many fixpoints $(\mathsf{x}_1, \ldots, \mathsf{x}_n, 0)$, $(\mathsf{x}_1, \ldots, \mathsf{x}_n, \pm 1)$, $(\mathsf{x}_1, \ldots, \mathsf{x}_n, \pm 2)$,... of $\mathsf{P}(p)$ exist because the initial value of the register s has no influence on the overall behavior of $\mathsf{P}(p)$.

Theorem 4 (Cardinality preserving map)**.** *Let $\mathcal{Z}$ be the set of solutions for a given Diophantine equation $p(x_1, \ldots, x_n) = 0$. A map Q from $p(x_1, \ldots, x_n)$ to a program $\mathsf{Q}(p)$ exists such that if $\mathcal{F}$ denotes the set of fixpoints of $\mathsf{Q}(p)$, then the cardinalities of $\mathcal{Z}$ and $\mathcal{F}$ coincide.*

For proving Theorem 4 we first introduce the map Q. Given $p(x_1, \ldots, x_n)$, by definition:

$$\mathsf{Q}(p) = \mathsf{for}\, s(\mathsf{inc}\, x_1);\ \mathsf{P}(p) \tag{13}$$

where $\mathsf{P}(p)$ is the program in SRL that we generate with the map used in the reduction of $\mathsf{HTP}(\mathbb{Z})$ to FIXPOINT in Section 3 and s is the register that eventually contains the result. We remark that, using any among $x_1, \ldots, x_n$ in place of x_1 would be irrelevant.

In our running example, i.e. $p(x, y) = 2x^3y^2 - xy^2 + 2$ in (1), the new map $\mathsf{Q}(p)$ produces:

$$\mathsf{for}\, s(\mathsf{inc}\, x); \tag{*}$$
$$\mathsf{for}\, x(\mathsf{for}\, x(\mathsf{for}\, x(\mathsf{for}\, y(\mathsf{for}\, y(\mathsf{inc}\, s;\ \mathsf{inc}\, s))))); \tag{A}$$
$$\mathsf{for}\, x(\mathsf{for}\, y(\mathsf{for}\, y(\mathsf{dec}\, s))); \tag{B}$$
$$\mathsf{inc}\, s;\ \mathsf{inc}\, s. \tag{C}$$

Compared with $\mathsf{P}(p)$ at (A), (B) and (C), the new line (*) modifies the content of the register x if and only if the initial contents of s is not 0. In general, it is easy to see that any eventual fixpoint of Q(p) must have the initial contents of s equal to 0.

We are now ready to show that Theorem 4 above is, in fact, a corollary of the next lemma.

Lemma 1. *Let $\mathcal{Z}$ be the set of solutions of the Diophantine equation $p(x_1, \ldots, x_n) = 0$. Then, $(\mathsf{x}_1, \ldots, \mathsf{x}_n) \in \mathcal{Z}$ if, and only if, $\mathsf{Q}(p)(\mathsf{x}_1, \ldots, \mathsf{x}_n, 0) = (\mathsf{x}_1, \ldots, \mathsf{x}_n, 0)$.*

Proof. Let us start by assuming $(\mathsf{x}_1, \ldots, \mathsf{x}_n) \in \mathcal{Z}$, i.e. $p(\mathsf{x}_1, \ldots, \mathsf{x}_n) = 0$. Then,

$\mathsf{Q}(p)$ behaves as follows:

$$\begin{aligned}\mathsf{Q}(p)(\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_n,0) &= (\mathsf{for}\, s(\mathsf{inc}\, x_1);\, \mathsf{P}(p))(\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_n,0)\\ &= (\mathsf{x}_1+0,\mathsf{x}_2,\ldots,\mathsf{x}_n,0+p(\mathsf{x}_1+0,\ldots,\mathsf{x}_n))\\ &= (\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_n,0).\end{aligned}$$

In the other direction, let $\mathsf{Q}(p)(\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_n,0) = (\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_n,0)$. Then:

$$\begin{aligned}\mathsf{Q}(p)(\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_n,0) &= (\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_n,0) &(14)\\ &= (\mathsf{x}_1+\mathsf{s},\mathsf{x}_2,\ldots,\mathsf{x}_n,\mathsf{s}+p(\mathsf{x}_1+\mathsf{s},\mathsf{x}_2,\ldots,\mathsf{x}_n)) &(15)\end{aligned}$$

where s is the value of the register s. The equation (15) is the tuple that $\mathsf{Q}(p)(\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_n,\mathsf{s})$ effectively computes. Since (14) holds by assumption, necessarily $\mathsf{s}=0$ and $p(\mathsf{x}_1+0,\mathsf{x}_2,\ldots,\mathsf{x}_n)=0$, i.e. $(\mathsf{x}_1+0,\mathsf{x}_2\ldots,\mathsf{x}_n)\in\mathcal{Z}$. □

Corollary 3. *Programs of* SRL *exist with exactly k fixpoints, for any $k\in\mathbb{N}$. Moreover, programs of* SRL *exist with an infinite number of fixpoints.*

Proof. It is easy to see that multivariate polynomials exist with exactly k solutions, for any $k\in\mathbb{N}$. For instance, for $k=3$ and variables x_1 and x_2 we can have $(x_1-1)(x_1-2)(x_1-3)(x_2^2+1)$. There are also multivariate polynomials with an infinite number of solutions, for instance, x_1x_2 (for $k=2$). The corresponding SRL programs inherits the same fixpoint cardinalities, thanks to Theorem 4. □

Let $C=\{0,1,2,\ldots,\aleph_0\}$ where $\aleph_0$ is the cardinality of natural numbers. Let $\emptyset\subset A\subset C$. Let $\mathcal{Z}$ be the set of solutions of a given Diophantine equation $p(x1,\ldots,x_n)=0$. In [36], Davis says that we cannot decide if the cardinality of $\mathcal{Z}$ belongs to A. Namely, no general algorithms exists for determining the "dimension" of $\mathcal{Z}$.

Corollary 4. *Let $\emptyset\subset A\subset\{0,1,2,\ldots,\aleph_0\}$. We cannot decide if the cardinality of the set of fixpoints of a program of* SRL *belongs to A.*

Proof. Use Theorem 4 above and [36, Theorem in page 552]. □

Albeit the cardinality of $\mathcal{F}$ can be decided only for trivial cases, some further interesting cardinality problem is still open. We comment on this in Section 6.

6. Conclusions and future work

In this work have investigated the expressiveness of SRL by studying the decidability of a computational problem which is a classical one for paradigmatic programming languages. We prove that "Does a given SRL program P have a fixed point $\overline{x}$?", that is, "$\exists\overline{x}:\mathsf{P}(\overline{x})=\overline{x}$?", is undecidable. We look at our results as significant because, broadly speaking, simple decision problems about very simple programming languages, like SRL is, are expected to be decidable.

In extreme synthesis, the fixpoint problem for SRL is undecidable because it "contains" the problem that asks for the existence of a solution of a polynomial equation which, in its turn, contains the undecidable Turing machine (or universal register language) halting problem. We deepen the above results in two directions. First, we show that upper bounds on the number of programming registers exist which imply the undecidability: 12 registers are enough in the general case and 3712 in the linear one. Second, we show that, except in trivial cases, deciding if the set of fixpoints of a SRL program has a given cardinality is also undecidable.

6.1. Future work

Plans for future work are manifold.

Lower bounds on the number of registers. Strengthen the above lower bounds on the number of registers that let FIXPOINT undecidable.

Cardinality of FIXPOINT. Investigate the decidability of quantitative questions. Let $\mathsf{FIXPOINT}_i(\mathsf{P})$ denote the problem: "Does the set of fixpoints of the program P have cardinality i?", for some i. Complementarily, let $\mathsf{NOT\text{-}FIXPOINT}_i(\mathsf{P})$ denote the problem: "Does the set of inputs which are not fixpoints of the program P have cardinality i?", for some i. Possible questions are:

1. Does any $i \in \mathbb{N}^+$ exist such that $\mathsf{NOT\text{-}FIXPOINT}_i(\mathsf{P})$ is true, for some SRL programs P? As a comment, we remark that Davis' [36] is not useful to answer this question. The reason is that there is no polynomial whose number of integer *non-solutions* is finite and positive. Equivalently, if a tuple of integers is not a solution of a given polynomial, then infinitely many tuples exist which are not solutions of that polynomial. However "infinite" is not equivalent to saying that *every* tuple of integer values *is not* a solution.
2. Do both $\mathsf{FIXPOINT}_i(\mathsf{P})$ and $\mathsf{FIXPOINT}_j(\mathsf{P})$ always belong to the same class of the Arithmetic Hierarchy? We conjecture that $\mathsf{FIXPOINT}_{\aleph_0}(\mathsf{P})$ cannot belong to the classes that contain every $\mathsf{FIXPOINT}_i(\mathsf{P})$, for $i > 0$. Moreover, any among $\mathsf{FIXPOINT}_i(\mathsf{P})$, with $i > 0$, can be in the classes which $\mathsf{FIXPOINT}_0(\mathsf{P})$ belongs to.

The question at Point 1 here above is crucial, because it represents the next step in our fixed point search. Answering to it when $i = 0$ would amount to answering about *the equivalence* between two programs of SRL. *That is why we see the study of* FIXPOINT *and of "counting" the number of fixpoints of programs of* SRL *relevant.*

SRL *and Counter Machines.* As suggested by an anonymous referee, a good way to study the decidability of decision problems over SRL might resume to analogous results on Counter Machines automata (CM for short).

A first justification to that suggestion says that a wide source of (un)decidability results on CM variants exists. The reason is that a CM with 2 counters

is equivalent to a Turing machine, but suitable restrictions on registers allow to define weaker CMs.

A second reason is that CM, at first glance, looks similar to SRL, especially if we restrict to BRCM, i.e. *bounded-reversal* CM. The following table summarizes common aspects of the two computational models:

	CM	SRL
Memory	Counter	Register
Arithmetic operators	Increment/Decrement	Increment/Decrement
State	Explicit in the structure of an automata	Hidden in the flow structure of a program
Computational restrictions	Limited number of increment/decrement reversals	Built-in as finite iterations

However, also neat differences exist between the two models. We outline them in the following table:

	CM	SRL
Register Domain	(Typically restricted to) $\mathbb{N}$	$\mathbb{Z}$
Input	On-line	Registers with initial values
Flow	(Non)Deterministic automata	Reversible program
Termination	Accepting or rejecting configurations	Registers with final values in them
Control operators	Conditional jump	No jump intructions

whose entries we briefly comment about here below.

Concerning the *Register Domain*, in principle, it is possible to devise a variant of SRL working on $\mathbb{N}$, instead than $\mathbb{Z}$, because $\mathbb{Z}$ can be represented by pairs of natural numbers that can express negative values, if we think of subtracting the second component of the pair from the first one, for example. However, even if we restricted SRL to work on $\mathbb{N}$ its main purpose would be to represent bijective functions. On the other side, the goal of CM is to accept or reject strings with the help of the values that their counters may assume. Typically, (un)decidability results relative to CM refer to counter machines where the decrement operation cannot make the value of any counter negative. Exceptions are blind and partially blind CM in [37] by Greibach, whose counters can assume negative values. Currently, we hardly see how to accomodate the global blocking condition, which stops a partially blind CM when a register becomes negative, in a reversible setting. Instead, we think it is sensible to explore how to accommodate the global accepting condition of blind CM which successfully holds whenever, starting from a configuration whose counters are all set to 0,

eventually reaches again a situation with all its counters set to 0. This behavior might correspond to a careful use of input and ancillary registers in a program of SRL which are expected to contain 0 at the end of its reversible computation. Essentially, this idea has been also considered in [10] for another reversible language.

Concerning the *Input*, being the CM automata, they typically operate on an input which is a stream of characters. Programs of SRL model a functional-like computation which assumes that all the required input is available in the registers when the computation starts.

Concerning the *Flow*, we think that Morita outlines which is the point in [38, 15]. He essentially observes that to make a CM reversible, it is necessary to explicitly restrict its underlying automaton with some kind of backward determinism. At that point, applying (un)decidability results that hold on CM, to SRL, does not look an immediate task.

Concerning the *Termination*, beside the above observations on blind and partially blind CM, we remark that the global blocking condition that can stop any blind CM configures those automata as computing *partial* functions. This is not at all contemplated in the interpretation of SRL programs which compute *total* functions.

Finally, concerning *Control operators*, the instruction set of a CM typically contains conditional jumps, something which, to us, is not at all immediately available in SRL. By the way, were we able to encode some sort of `if-then-else` in SRL, we would prove its equivalence with the class of Reversible Primitive Permutations in [19, 20], by incontrovertibly making SRL the simplest primitive recursive complete reversible programming language.

All that said, we anyway conclude by sketching how a possible embedding from CM into a program of SRL could work. The starting point could be [39] by Straßburger. He shows that the problem of deriving a formula in a specific version of Linear Logic — he calls it NEL — is undecidable because 2-counter machines (2CM) can be mapped to formulas of NEL. The culprit of the result is to show that accepting an initial configuration by a 2CM becomes the problem of deriving a suitable formula in NEL. We conjecture that an analogous result holds once the current operational semantics of SRL is rephrased as a deductive system inspired in NEL.

References

[1] A. R. Meyer, D. M. Ritchie, The complexity of loop programs, in: Proceedings of 22nd National Conference of the ACM, Association for Computing Machinery, 1967, pp. 465–469.

[2] A. R. Meyer, D. M. Ritchie, Computational complexity and program structure, Tech. rep., IBM, iBM Res. Report RC 1817 (1967).

[3] T. Crolard, S. Lacas, P. Valarcher, On the expressive power of the Loop language, Nordic Journal of Computing 13 (1-2) (2006) 46–57.

[4] B. H. Enderton, Computability Theory: an Introduction to Recursion Theory, 1st Edition, Academic Press, 2010.

[5] L. Kristiansen, K.-H. Niggl, On the computational complexity of imperative programming languages, Theoretical Computer Science 318 (1-2) (2004) 139–161.

[6] A. B. Matos, Analysis of a simple reversible language, Theoretical Computer Science 290 (3) (2003) 2063–2074.

[7] K. Perumalla, Introduction to Reversible Computing, CRC Press, 2014.

[8] L. Paolini, M. Piccolo, L. Roversi, A Certified Study of a Reversible Programming Language, in: T. Uustalu (Ed.), 21st International Conference on Types for Proofs and Programs (TYPES 2015), Vol. 69 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2018, pp. 7:1–7:21. doi:10.4230/LIPIcs.TYPES.2015.7.

[9] L. Paolini, M. Zorzi, qPCF: a language for quantum circuit computations, in: T. Gopal, G. Jäger, S. Steila (Eds.), 14th Annual Conference on Theory and Applications of Models of Computation, Vol. 10185 of Lecture Notes in Computer Science, Springer, Germany, 2017, pp. 455–469. doi:10.1007/978-3-319-55911-7_33.
URL http://dx.doi.org/10.1007/978-3-319-55911-7_33

[10] L. Paolini, M. Piccolo, L. Roversi, On a class of reversible primitive recursive functions and its turing-complete extensions, New Generation Computing 36 (3) (2018) 233–256. doi:10.1007/s00354-018-0039-1.

[11] H. B. Axelsen, R. Glück, What do reversible programs compute?, in: M. Hofmann (Ed.), Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings, Vol. 6604 of Lecture Notes in Computer Science, Springer, 2011, pp. 42–56. doi:10.1007/978-3-642-19805-2_4.

[12] H. B. Axelsen, R. Glück, On reversible turing machines and their function universality, Acta Inf. 53 (5) (2016) 509–543. doi:10.1007/s00236-015-0253-y.

[13] C. H. Bennett, Notes on Landauer's principle, reversible computation, and Maxwell's Demon, Studies in History and Philosophy of Modern Physics 34 (2003) 501–510.

[14] A. D. Vos, Reversible Computing - Fundamentals, Quantum Computing, and Applications, Wiley, 2010.

[15] K. Morita, Theory of Reversible Computing, Monographs in Theoretical Computer Science. An EATCS Series, Springer, 2017. doi:10.1007/978-4-431-56606-9.

[16] L. Paolini, L. Roversi, M. Zorzi, Quantum programming made easy, in: T. Ehrhard, M. Fernández, V. d. Paiva, L. Tortora de Falco (Eds.), Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Oxford, UK, 7-8 July 2018, Vol. 292 of Electronic Proceedings in Theoretical Computer Science, Open Publishing Association, 2019, pp. 133–147. doi:10.4204/EPTCS.292.8.

[17] T. Toffoli, Reversible computing, in: J. de Bakker, J. van Leeuwen (Eds.), Automata, Languages and Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 1980, pp. 632–644.

[18] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition, 10th Edition, Cambridge University Press, New York, NY, USA, 2011.

[19] L. Paolini, M. Piccolo, L. Roversi, A Class of Reversible Primitive Recursive Functions, Electronic Notes in Theoretical Computer Science 322 (18605) (2016) 227–242.

[20] L. Paolini, M. Piccolo, L. Roversi, A Class of Recursive Permutations which is Primitive Recursive Complete, Tech. rep., Università di Torino, submitted to TCS. (2016).

[21] A. B. Matos, L. Paolini, L. Roversi, The output of SRL transformations: lower bound of the growth rate, (Preliminary Res. Notes) (2017).

[22] H. Rogers, Theory of Recursive Functions and Effective Computability, MIT Press Cambridge, MA, 1967.

[23] A. Matos, L. Paolini, L. Roversi, The fixed point problem for general and for linear srl programs is undecidable, in: A. Aldini, M. Bernardo (Eds.), 19th Italian Conference on Theoretical Computer Science, ICTCS 2018, Vol. 2243 of CEUR Workshop Proceedings, 2018, pp. 128–139.

[24] M. Davis, Computability and Unsolvability, Dover, 1985.

[25] S. C. Kleene, Introduction to Metamathematics, North-Holland, 1952, reprinted by Ishi Press on 2009.

[26] N. Cutland, Computability: An Introduction to Recursive Function Theory, Cambridge University Press, 1980.

[27] G. Boolos, P. J. Burgess, C. R. Jeffrey, Computability and Logic, CUP, 2007.

[28] S. B. Cooper, Computability Theory, CRC Press Company, 2004.

[29] H. Hermes, Enumerability, Decidability, Computability, Springer-Verlag, 1969.

[30] R. I. Soare, Turing Computability: Theory and Applications, Springer, 2016, Department of Mathematics, The University of Chicago.

[31] Y. Matiyasevich, Hilbert's Tenth Problem, MIT Press, 1993.

[32] J. P. Jones, Undecidable diophantine equations, Bull. Amer. Math. Soc. (N.S.) 3 (2) (1980) 859–862.

[33] Z.-W. Sun, Further results on Hilbert's Tenth Problem, Cornell University Library (2017).
URL `https://arxiv.org/abs/1704.03504`

[34] M. B. Nathanson, Additive Number Theory The Classical Bases, Springer Science & Business Media, 1996.

[35] N. D. Jones, Computability and Complexity: From a Programming Perspective, Foundations of Computing Series, MIT Press, 1997.

[36] M. Davis, On the number of solutions of Diophantine equations, Proceedings of the American Mathematical Society 35 (2) (1972) 552–554.

[37] S. Greibach, Remarks on blind and partially blind one-way multi-counter machines, Theoretical Computer Science 7 (3) (1978) 311 – 324. doi:https://doi.org/10.1016/0304-3975(78)90020-8.

[38] K. Morita, Universality of a reversible two-counter machine, Theoretical Computer Science 168 (2) (1996) 303 – 320. doi:https://doi.org/10.1016/S0304-3975(96)00081-3.

[39] L. Straßburger, System NEL is undecidable, Electr. Notes Theor. Comput. Sci. 84 (2003) 166–177.