4-2012

Algorithmics of Checking Whether a Mapping Is Injective, Surjective, and/or Bijective

E. Cabral Balreira
Trinity University, ebalreir@trinity.edu

Olga Kosheleva
The University of Texas at El Paso, olgak@utep.edu

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-12-15
To appear in *Proceedings of the Fifth International Workshop on Constraint Programming and Decision Making CoProD'12*, Novosibirsk, Russia, September 23, 2012.

Algorithmics of Checking Whether a Mapping Is Injective, Surjective, and/or Bijective

E. Cabral Balreira[1], Olga Kosheleva[2], and Vladik Kreinovich[2]

[1] Department of Mathematics, Trinity University
San Antonio, TX 78212 USA ebalreir@trinity.edu
[2] University of Texas at El Paso, El Paso, TX 79968, USA
{olgak,vladik}@utep.edu

Abstract. In many situations, we would like to check whether an algorithmically given mapping $f : A \to B$ is injective, surjective, and/or bijective. These properties have a practical meaning: injectivity means that the events of the action f can be, in principle, reversed, while surjectivity means that every state $b \in B$ can appear as a result of the corresponding action. In this paper, we discuss when algorithms are possible for checking these properties.

1 Formulation of the Problem

States of real-life systems change with time. In some cases, this change comes "by itself", from laws of physics: radioactive materials decays, planets go around each other, etc. In other cases, the change comes from our interference: e.g., a spaceship changes trajectory after we send a signal to an engine to perform a trajectory correction. In many situations, we have equations that describe this change, i.e., we know a function $f : A \to B$ that transform the original state $a \in A$ into a state $f(a) \in b$ at a future moment of time. In such situations, the following two natural problems arise.

The first natural question is: Are the changes reversible? For example, when we erase the value of the variable in a computer, by replacing it with 0s, the changes are not reversible: there is not trace of the original value left, and so reconstructing the original value is not possible. In such situations, two different original states $a \neq a'$ leads to the exact same new state $f(a) = f(a')$. If different states $a \neq a'$ always lead to different states $f(a) \neq f(a')$, then, in principle, we can reconstruct the original state a based on the new state $f(a)$. In mathematical terms, mapping $f : A \to B$ that map different elements into different ones are called *injective*, so the question is whether a given mapping is injective.

The second natural question is: Are all the states $b \in B$ possible as a result of this dynamics, i.e., is it true that every state $b \in B$ can be obtained as $f(a)$ for some $a \in A$. In mathematical terms, mappings that have this property are called *surjective.*

We may also want to check whether a mapping is both injective and surjective, i.e., in mathematical terms, whether it is a *bijection.*

Thus, in practice, it is important to be able to check whether a given mapping is injective, surjective, or bijective; see, e.g., [1]. In this paper, we analyze this problem from an algorithmic viewpoint.

2 Case of Polynomial and, More Generally, Semi-Algebraic Mappings

Case study. Let us first consider the case when the set A and B are *semi-algebraic* sets, i.e., when each of these sets is characterized by a finite collection of polynomial equalities and inequalities with rational coefficients. For example, the upper half of the unit circle centered at the point $(0, 0)$ is a semi-algebraic set, since it can be described as the set of all the pairs (x_1, x_2) that satisfy two polynomial inequalities: $x_1^2 + x_2^2 \le 1$ and $x_2 \ge 0$.

We also assume that the mapping $f : A \to B$ is semi-algebraic – in the sense that the graph $\{(a, f(a) : a \in A\}$ of this function is a semi-algebraic set. For example, every polynomial mapping is, by definition, semi-algebraic. Polynomial mappings are very important, since every continuous function on bounded set can be, within any given accuracy, approximated by a polynomial. Since in practice, we only know the actual consequences of each action with some accuracy, this means that every action can be represented by a polynomial mapping.

First result: algorithms are possible. In the polynomial case and, more generally, in the semi-algebraic case, all three above questions are algorithmically decidable:

Proposition 1. *There exists an algorithm, that, given two semi-algebraic sets A and B and a semi-algebraic mapping $f : A \to B$, checks whether f is injective, surjective, and/or bijective.*

Proof. Under the conditions of the proposition, each of the relations $a \in A$, $b \in B$, and $f(a) = b$ can be described by a finite set of polynomial equalities and inequalities. A polynomial is, by definition, a composition of additions and multiplications. Thus, both the injectivity and surjectivity can be described in terms of the first order language with variables running over real numbers, and elementary formulas coming from addition, multiplication, and equality. Namely, injectivity can be described as

$$\forall a\, \forall a'\, \forall b\, ((a \in A \,\&\, a' \in A \,\&\, f(a) = b \,\&\, f(a') = b \,\&\, b \in B) \to a = a'),$$

and surjectivity can be described as $\forall b\, (b \in B \to \exists a\, (a \in A \,\&\, f(a) = b))$. For such formulas, there is an algorithm – originally proposed by Tarski and later modified by Seidenberg – that decides whether a given formula is true or not; see, e.g., [3, 6]. Thus, our problems are indeed algorithmically decidable.

Remark 1. One of the main open problems in this area is Jacobian Conjecture, according to which every polynomial map $f : C^n \to C^n$ from n-dimensional complex space into itself for which the Jacobi determinant $\det\left(\dfrac{\partial f_i}{\partial x_j}\right)$ is equal to 1

is injective; see, e.g., [2]. This is an open problem, but for any given dimension n and for any given degree d of the polynomial, the validity of the corresponding case of this conjecture can be resolved by applying the Tarski-Seidenberg algorithm.

How efficient are the corresponding algorithms? The following results show that the existence of the above algorithms do not mean that these algorithms are necessary efficient, even for polynomial mappings.

Proposition 2. *The problem of checking whether a given polynomial mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ is injective is, in general, NP-hard.*

Proof. By definition, a problem is NP-hard if every problem from the class NP can be reduced to it; see, e.g., [5]. Thus, to prove that this problem is NP-hard, let us reduce a known NP-hard problem to it. As such a known problem, we take a *subset problem*: given $n+1$ positive integers $s_1, \ldots, s_n, S$, check whether there exist values $x_1, \ldots, x_n \in \{0, 1\}$ for which $\sum\limits_{i=1}^{n} s_i \cdot x_i = S$. For each instance of this problem, let us form the following polynomial mapping $f(x_1, \ldots, x_n, x_{n+1}) = (x_1, \ldots, x_n, P(x_1, \ldots, x_n) \cdot x_{n+1})$, where $P(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \sum\limits_{i=1}^{n} x_i^2 \cdot (1 - x_i)^2 + \left(\sum\limits_{i=1}^{n} s_i \cdot x_i - S \right)^2$. If the original instance of the subset sum problem has a solution $(x_1, \ldots, x_n)$, then for this solution, we have $P(x_1, \ldots, x_n) = 0$ and thus, vectors $(x_1, \ldots, x_n, 0) \neq (x_1, \ldots, x_n, 1)$ are mapped into the same vector $(x_1, \ldots, x_n, 0)$, so f is not injective.

Vice versa, if the original instance of the subset sum problem does not have a solution, then $P(x_1, \ldots, x_n)$ is always positive – otherwise, the tuple $(x_1, \ldots, x_n)$ would be a solution. Thus, once we know $y = (y_1, \ldots, y_n, y_{n+1}) = f(x_1, \ldots, x_n, x_{n+1})$, we can recover x_i as follows: $x_i = y_i$ for $i \leq n$ and $x_{n+1} = y_{n+1}/P(x_1, \ldots, x_n)$. So, the above mapping f is injective if and only if the original instance of the subset problem has a solution. The reduction is proven, so the problem of checking injectivity is indeed NP-hard.

Proposition 3. *The problem of checking whether a given polynomial mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ is surjective is, in general, NP-hard.*

Proof. This is proven by the same reduction as in the previous proof: when $P(x_1, \ldots, x_n) = 0$ for some $x_1, \ldots, x_n$, then the element $(x_1, \ldots, x_n, 1)$ is not in the range of the mapping; on the other hand, when P is always positive, the mapping is surjective.

Proposition 4. *The problem of checking whether a given polynomial mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ is bijective is, in general, NP-hard.*

Proof. This is proven by the same reduction as in the previous two proofs.

Remark 2. It would be interesting to find out whether the following problems are NP-hard:

– checking whether a given injective polynomial mapping is also surjective, and
– checking whether a given surjective polynomial mapping is also injective.

Polynomial mapping with computable coefficients. For such mappings, the corresponding questions become algorithmically undecidable. A real number x is called *computable* if there exists an algorithm that, given a natural number n, returns a rational number r_n which is 2^{-m}-close to x. Equivalently, instead of specifying the sequence 2^{-n}, we can require the existence of an algorithm that, given a rational number $\varepsilon > 0$, produces a rational number which is ε-close to x; see, e.g., [4, 7].

Proposition 5. *No algorithm is possible that, given a polynomial mapping $f : \mathbb{IR}^n \to \mathbb{IR}^n$ with computable coefficients, decides whether this mapping is injective.*

Proof. The proof is based on the known fact that no algorithm is possible that, given a computable real number a, decides whether this number is equal to 0 or not. We can thus take $n = 1$ and $f(x) = a \cdot x$. This mapping is injective if and only if $a \neq 0$. Since we cannot algorithmically decide whether $a \neq 0$, we thus cannot algorithmically check whether a given mapping is injective.

Proposition 6. *No algorithm is possible that, given a polynomial mapping $f : \mathbb{IR}^n \to \mathbb{IR}^n$ with computable coefficients, decides whether this mapping is surjective.*

Proposition 7. *No algorithm is possible that, given a polynomial mapping $f : \mathbb{IR}^n \to \mathbb{IR}^n$ with computable coefficients, decides whether this mapping is bijective.*

Proof. These two results are proven by the same reduction as the previous proposition.

Proposition 8. *No algorithm is possible that, given an injective polynomial mapping $f : [0, 1] \to [0, 1]$ with computable coefficients, decides whether this mapping is also surjective.*

Proof. Indeed, for all $a \in [0, 0.5]$, the mapping $f(x) = (1 - a^2) \cdot x$ is injective, but it is surjective only for $a = 0$.

Proposition 9. *No algorithm is possible that, given an surjective polynomial mapping $f : \mathbb{IR}^n \to \mathbb{IR}^n$ with computable coefficients, decides whether this mapping is also injective.*

Proof. Indeed, for $n = 1$, the mapping $f(x) = -a^2 \cdot x^2 + x^3$ is always surjective, but it is injective only when $a^2 = 0$, i.e., when $a = 0$.

3 General Case

Analytical expressions. If instead of allowing computable numbers, we allow general analytical expressions, i.e., expression in terms of elementary constants such as π and elementary functions such as sin, the above problems remain algorithmically undecidable. Indeed, according to Matiyasevich's solution of the tenth Hilbert problem, it is not algorithmically possible to check whether a given polynomial equality $F(x_1, \ldots, x_n) = 0$ has an integer solution. Thus, we can form a function as in the proof of Propositions 2, 3, and 4, with $P(x_1, \ldots, x_n) = \sum_{i=1}^{n} \sin^2(\pi \cdot x_i) + F^2(x_1, \ldots, x_n)$. Here, $P = 0$ if and only if the equation $F = 0$ has an integer solution.

General computable case. For a computable mapping f between computable compact sets A and B [4, 7], we can efficiently check *approximate* injectivity and surjectivity. For example, instead of checking whether $f(a) = f(a')$ implies $a = a'$, we can check, for given $\varepsilon > 0$ and $\delta > 0$, whether $d(f(a), f(a')) \le \delta$ implies $d(a, a') \le \varepsilon$, i.e., whether $m \stackrel{\text{def}}{=} \max\{d(a, a') : d(f(a), f(a')) \le \delta\} \le \varepsilon$. It is known that between every two values $0 \le \underline{\delta} < \overline{\delta}$, there exists a δ for which the set $\{d(f(a), f(a')) \le \delta\}$ is a computable compact [4] and thus, for which m is computable. Thus, if we have two computable numbers $0 \le \underline{\varepsilon} < \overline{\varepsilon}$, we can check whether $m \ge \overline{\varepsilon}$ or $m \not\ge \underline{\varepsilon}$. So, within each two intervals $(\underline{\delta}, \overline{\delta})$ and $(\underline{\varepsilon}, \overline{\varepsilon})$, we can algorithmically find values δ and ε for which the question of (δ, ε)-injectivity is algorithmically decidable.

For surjectivity, a natural idea is to check whether every $b \in B$ is ε-close to some $f(a)$, i.e., where $s \stackrel{\text{def}}{=} \max_{b \in B} \min_{a \in A} d(b, f(a)) \le \varepsilon$. For computable mappings, s is computable, thus, with each interval $(\underline{\varepsilon}, \overline{\varepsilon})$, we can algorithmically find a value ε for which the question of ε-surjectivity is algorithmically decidable.

Acknowledgments. This work was supported in part by the National Science Foundation grants HRD-0734825 and DUE-0926721, and by Grant 1 T36 GM078000-01 from the National Institutes of Health.

References

1. Balreira, E. C.: Foliations and global inversion, Commentarii Mathematici Helvetici. **85**(1) (2010) 73–93
2. Bass, H., Connell, E. H., Wright, D.: The Jacobian Conjecture: reduction of degree and formal expansion of the inverse, Bull. Amer. Math. Soc. **7**(2) (1982) 287–330
3. Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry, Springer-Verlag, Berlin (2006)
4. Bishop, E., Bridges, D. S.: Constructive Analysis, Springer, N.Y. (1985)
5. Papadimitriou, C. H.: Computational Complexity, Addison Wesley, San Diego (1994)
6. Tarski, A.: A Decision Method for Elementary Algebra and Geometry, 2nd ed., Berkeley and Los Angeles (1951)
7. Weihrauch, K.: Computable Analysis, Springer-Verlag, Berlin (2000)