9-2016

Computability of the Avoidance Set and of the Set-Valued Identification Problem

Anthony Welte
École Nationale Supérieure de Techniques Avancées Bretagne, tony.welte@gmail.com

Luc Jaulin
École Nationale Supérieure de Techniques Avancées Bretagne, lucjaulin@gmail.com

Martine Ceberio
University of Texas at El Paso, mceberio@utep.edu

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

Comments:
Technical Report: UTEP-CS-16-64
To appear in *Journal of Uncertain Systems*

Computability of the Avoidance Set and of the Set-Valued Identification Problem

Anthony Welte[1], Luc Jaulin[1],
Martine Ceberio[2], and Vladik Kreinovich[2]
[1]Lab STICC
École Nationale Supérieure
de Techniques Avancées Bretagne
(ENSTA Bretagne)
2 rue François Verny
29806 Brest, France
tony.welte@gmail.com, lucjaulin@gmail.com
[2]Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, Texas 79968, USA
mceberio@utep.edu, vladik@utep.edu

Abstract

In some practical situations, we need to find the *avoidance set*, i.e., the set of all initial states for which the system never goes into the forbidden region. Algorithms are known for computing the avoidance set in several practically important cases. In this paper, we consider a general case, and we show that, in some reasonable sense, the corresponding general problem is always algorithmically solvable. A similar algorithm is possible for another general system-related problem: the problem of describing the set of all possible states which are consistent with the available measurement results.

1 Formulation of the Problem

In control, we usually deal with robots (or other controlled devices) whose states s are described by tuples of real numbers $s = (s_1 \ldots, s_d)$. The dynamics of such devices is usually described by a system of differential equations

$$\frac{ds_i}{dt} = f_i(s(t)),$$

for a known computable functions $f_i(s)$. In most practical situations, we can use these equations to compute, for each initial state s_0 at the starting moment t_0, and for each moment of time $t < t_0$, the state $s(s_0, t)$ of the system at the moment t; see, e.g., [2, 5, 10, 12].

Often in control, we have a set S of states that a robot (or other controlled device) needs to avoid. Because of this necessity:

- once we know how the states change in time, i.e., once we know the algorithm $s(t, s_0)$ that describes how the state s at moment t depends on t and on the initial state s_0,
- we need to find the set S_0 of all the initial states for which the trajectory avoids the forbidden set S for all moments of time from the starting moment t_0 to a given future moment T.

In other words, we want to find the *avoidance set*

$$S_0 = \{s_0 : s(t, s_0) \notin S \text{ for all } t \in [t_0, T]\}.$$

There exist algorithms for solving this problem in some specific situations; see, e.g., [3, 7, 9].

In this paper, we analyze the general problem of computing the avoidance set, and we show that this problem is, in some reasonable sense, algorithmically computable.

2 What Is Computable: A Brief Reminder

Need for this reminder. To formulate our problem, let us recall what is means for a function to be computable and – since we are interested in avoiding a given set – what it means for a set to be computable. Both these computability notions are based on the notion of a computable number, which will be introduced first.

What is computable: general idea. When we say that some object is computable, this means that we have an algorithm that is able to compute this object with any given accuracy.

Computable real numbers. Informally, a real number x is computable if, for any given accuracy, we can efficiently compute a binary-rational (or, more generally, a rational) number that approximates x with the given accuracy.

Thus, we arrive at the following definition (see, e.g., see, e.g., [1, 6, 11]):

Definition 1. *By a* computable real number*, we mean a pair consisting of a real number x and an algorithm that, given an integer n, returns a rational number r_n for which $|r_n - x| \le 2^{-n}$.*

A computable tuple can be naturally defined as a tuple of computable numbers.

Definition 2. *By a* computable tuple*, we mean a tuple $x = (x_1, \ldots, x_n)$ consisting of computable real numbers $x_1, \ldots, x_n$.*

On the set of all the tuples, we can naturally define the Euclidean distance $d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$. It is easy to see that for every two computable real numbers, we can algorithmically compute the distance $d(x, y)$.

In this paper, we will use the notation $D_r(x) \stackrel{\text{def}}{=} \{y : d(x, y) \le r\}$.

Computable functions. Informally, a function from tuples to real numbers is computable if we can compute $f(x)$ for each computable tuple x. For each computable tuple, we only know its approximation. Thus, we need to know, for each desired accuracy in $f(x)$, with what accuracy we need to compute x to get $f(x)$ with the desired accuracy.

In the following text, we will take into account that in real life, all the parameters describing a state are bounded. Thus, all the corresponding functions are defined on a appropriate computable box $B = [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]$, i.e., on a box with computable endpoints $\underline{x}_i$ and $\overline{x}_i$.

Definition 3. *By a* computable function*, we mean a triple consisting of a real-valued function $f : B \to \mathbb{R}$ defined on a computable box B and the following two algorithms:*

- *an algorithm that, given a rational-valued tuple $x \in B$, natural number n, computes the value $f(x)$; and*
- *an algorithm that, given a natural number n, returns a natural number m for which $d(x, x') \in 2^{-m}$ implies $|f(x) - f(x')| \le 2^{-n}$.*

It is known that most usual functions are computable, e.g., arithmetic operations, minimum and maximum, etc. It is also known that a superposition of computable functions is computable. So, e.g., the sum or the minimum of two computable functions is also computable.

It is also known that for there exists an algorithm that, given a computable function $f(x)$ on an interval with computable endpoints, computes the supremum $\sup\limits_{x \in S} f(x)$ and the infimum $\inf\limits_{x \in S} f(x)$ of $f(x)$ on S – and if the original function also computably depended on some parameter, the resulting inf and sup function are also computable in terms of this parameter.

Computable sets of real-values tuples. Since we only know the tuples with some accuracy, we can therefore only know the sets with some accuracy. It is reasonable to say that a set A is an *ε-approximation* to a set B if:

- every element $a \in A$ is ε-close to some element from $b \in B$, and
- every element $b \in B$ is ε-close to some element from $a \in A$.

Comment. For every two sets A and B, the infimum of all real numbers $\varepsilon > 0$ with this property is known as the *Hausdorff distance* $d_H(A, B)$. In terms of the Hausdorff distance, the above property takes the form $d_H(A, B) \le \varepsilon$.

In the computer, we can only list finitely many tuples, thus, we can only have finite sets. So, we arrive at the following definition.

Definition 4. *By a* computable set*, we mean a pair consisting of a set S and an algorithm that, given a natural number n, produce a finite list of elements L_n for which:*

- *for every element $s \in S$, there exists $x \in L_n$ for which $d(s, x) \le 2^{-n}$; and*
- *for every element $x \in L_n$, there exists $s \in S$ for which $d(x, s) \le 2^{-n}$.*

Comments. Since we can only measure locations etc. with some accuracy, if s is a limit point of the set S, i.e., if $s = \lim s_n$ for $s_n \in S$, then, no matter how accurately we measure s, we will never be able to distinguish it from the appropriate element s_n. Thus, we can never be able to tell that the point s does not belong to the set S. Because of this, it makes sense to assume that the computable set contains all its limit points, i.e., that it is topologically *closed*. For example, if the computable set S contains all rational valued from the interval $[0, 1]$, then it should contain the whole interval as well.

For a metric space, an ε-close finite set is known as an *ε-net*. It is known that a closed set has a finite ε-net for every ε if and only if this set is compact. Because of this, computable sets are also known as *computable compact sets.*

One can check, e.g., that every box $[\underline{x}_1, \underline{x}_1] \times \ldots [\underline{x}_n, \overline{x}_n]$ with computable bounds $\underline{x}_i$ and $\overline{x}_i$ is a computable set, and that every ball $D_r(x)$ with computable r and x is also computable. It is known that for every computable set S, the distance $d(x, S) \stackrel{\text{def}}{=} \inf\limits_{s \in S} d(x, s)$ from a point x to this set is a computable function of x.

Computable set means an outer approximation. When a set S is computable, this means that for every n, the set S is contained in the union of finitely many balls $D_{2^{-n}}(x)$ of radius 2^{-n} with centers in different points $x \in L_n$:

$$S \subseteq \bigcup_{x \in L_n} D_{2^{-n}}(x).$$

One can easily see that this union is 2^{-n}-close to the original set S. When $n \to \infty$, this union gets closer and closer to the original set S. Thus, when we say that a set is computable, we mean that we have more and more accurate *outer* approximations to this set.

One needs to be careful when dealing with computable functions on computable sets. When dealing with computable functions on computable set, one needs to be careful, since seemingly natural things turned out to be not computable. For example, it may seem reasonable to assume that for every

computable function f on a computable set S, the set $S_f \stackrel{\text{def}}{=} \{s : f(s) \geq 0\}$ is computable, but this is not true: no algorithm is possible that, given a computable function f on a computable set S, provides the lists L_n corresponding to the set S_f.

This negative result follows from the know result that it is not possible, given a Turing machine, to check whether it halts or not. Based on each Turing machine, we can design a computable number a by taking:

- $r_n = 2^{-n}$ if the corresponding Turing machine did not stop by moment n and
- $r_n = 2^{-t}$ if it stopped at moment $t \leq n$.

For this computable real number a, we have $a = 0$ if and only if the original Turing machine never stops. We can then define a function $f(x) = -a \cdot x$ on the interval $S = [0, 1]$:

- for $a = 0$, the set S_f is the while interval, while
- for $a > 0$, the set S_f consists of only one point 0.

Thus, if we could approximate the set S_f with accuracy 0.5, we would be able to tell whether $a = 0$ and $a > 0$, and we have already shown that this is not possible.

A similar negative result shows that the closure of a complement to a computable set is not necessarily computable. To be more precise, there is no algorithm that, given such a closure, would return the corresponding lists L_n. Indeed, let us define the computable set corresponding to a given Turing machine as follows. For each n, let us take:

- $L_n = \{-1, 0, 2^{-n}, 2 \cdot 2^{-n}, \ldots, 1 - 2^{-n}, 1\}$ if the Turing machine did not stop by moment n and
- $L_n = \{-1, 0, 2^{-t}, 2 \cdot 2^{-t}, \ldots, 1 - 2^{-t}, 1\}$ if it stopped at moment $t \leq n$.

Then:

- if the Turing machine does not stop, the resulting set S is $\{-1\} \cup [0, 1]$, and the closure of its complement $[-1, 1] - S$ is the interval $[-1, 0]$;
- on the other hand, if it stops at some moment t, then

$$S = \{-1, 0, 2^{-t}, 2 \cdot 2^{-t}, \ldots, 1 - 2^{-t}, 1\},$$

  and the closure of the complement $[-1, 1] - S$ is the whole interval $[-1, 1]$.

If we could approximate this closure with accuracy 0.5, we would be able to tell whether the Turing machine halts or not, and we know that this is not possible.

3 Analysis of the Problem

Analysis of the problem. For every moment t, the requirement that the state $s(t, s_0)$ does not belong to the set S can be equivalently formulated as $d(s(t, s_0), S) > 0$.

A state is usually described by listing the values of finitely many parameters characterizing this state: $s = (s_1, \ldots, s_n)$. We have also mentioned that, since all the parameters are usually bounded, the set of possible states is a computable box B.

The dependence $s = s(t, s_0)$ is usually continuous (and computable) for $s_0 \in B$. Hence, the function that maps a real number t into the distance $d(s(t, s_0), S)$ is also continuous. Therefore, the required that this distance is positive for all $t \in [t_0, T]$ can be equivalently reformulated as

$$F(s_0) \stackrel{\text{def}}{=} \inf_{t \in [t_0, T]} d(s(t, s_0), S) > 0.$$

As we mentioned earlier, the infimum of a computable function is also computable, so the function $F(s_0)$ is computable. So, what we want is, given a computable function $F(s_0)$, to approximate the set $\{s_0 : F(s_0) > 0\}$ as closely as possible.

Need for inner approximations. The usual technique of computable sets provides us with *outer* approximations to this set. We also want to have *inner* approximations; namely, we want to make sure that:

- if we have selected a point s_0 from avoidance set,
- then even if we implement the corresponding starting state with some accuracy, we will still be in the avoidance set.

In other words, we want to make sure that the whole neighborhood of the corresponding points s_0 belong to the desired avoidance set.

4 Main Computational Result

Proposition. *There exists an algorithm that, given a computable function $f(x)$ on an n-dimensional computable box B and a natural number m, produces a finite list of tuples $x^{(1)}, \ldots, x^{(N)} \in B$ and a computable real number $r > 0$ such that*

$$\{x : f(x) > 2^{-m}\} \subseteq \bigcup_{i=1}^{N} D_r\left(x^{(i)}\right) \subseteq \{x : f(x) > 0\}.$$

Comment. This results provides both the outer approximations and the desired inner approximations to the avoidance set.

Notational comment. Here, as before, $D_r\left(x^{i)}\right)$ denotes the set of all the points x for which $d\left(x, x^{(i)}\right) \le r$.

Practical comment. Strictly speaking, we are not exactly computing the set $\{x : f(x) > 0\}$, but it is OK: e.g., for the function $F(s_0)$, the requirement $F(s_0) > 2^{-m}$ means that we keep the distance from the to-be-avoided set to be at least 2^{-m}. For large m, from the practical viewpoint, 2^{-m} is the same as 0. (So, if we strengthen our requirement this way, we are not missing too many initial states.)

Mathematical comment. The Proposition holds not only for tuples of real numbers, but also for functions on general metrics spaces – e.g., for quantum states which are elements of the Hilbert space, an infinite-dimensional analog of the finite-dimensional Euclidean space. Let us therefore consider computable sets in a general metric space.

Informally, we need to have approximating elements. Each approximating element can be described in a computer, so it must be described by a finite sequence of symbols – and thus, encoded by a natural number. Thus, we can describe these approximating elements as a sequence of elements $a_1, \ldots, a_n, \ldots$

Since we are interested in computable metric spaces, there should be an algorithm that, given two natural numbers m and n, computes the distance $d(a_m, a_n)$. A general element of a metric space is computable if, given n, we can find an element a_m which is 2^{-n}-close to this element. So, we arrive at the following definition.

Definition 5.

- *By a* computable metric space*, we mean a triple consisting of a metric space M with distance $d(a, b)$, a sequence of elements $a_1, \ldots, a_n, \ldots$ which is everywhere dense in M, and an algorithm that, given natural numbers m and n, returns a computable number $d(a_m, a_n)$.*
- *By a* computable element *of a computable metric space, we mean a pair consisting of an element $a \in M$ and an algorithm that, given a natural number n, returns an integer $k(n)$ for which*

$$d(a, a_{k(n)}) \le 2^{-n}.$$

For every two computable elements $a, a' \in M$, we can use the appropriate approximations to compute the distance $d(a, a')$. The above notion of a computable set and a computable function can be naturally extended to general computable metric spaces.

Definition 6. *By a* computable function *from a computable metric space M to a computable metric space B we mean a triple consisting of a function $f : M \to B$ and the following two algorithms:*

- *an algorithm that, given a natural number n, computes $f(a_n) \in B$; and*
- *an algorithm that, given a natural number n, returns a natural number m for which $d(a, a') \in 2^{-m}$ implies $d(f(a), f(a')) \le 2^{-n}$.*

Proof of the Proposition.

1°. Let us first construct the desired tuples $x^{(1)}, \ldots, x^{(N)}$ and the desired value r.

Since the function $f(x)$ is computable, there exists an integer p for which $d(x, x') \le 2^{-p}$ implies that $|f(x) - f(x')| \le 2^{-(m+2)}$. We then choose $r = 2^{-p}$.

For each variable x_i, let us list the values $\underline{x}_i$, $\underline{x}_i + h$, $\underline{x}_i + 2h$, $\ldots$, $\overline{x}_i$, where $h \le \dfrac{2^{-p}}{\sqrt{n}}$, and let us list all possible tuples combining these values.

Then, for each point $x \in B$ and for every i, we can find an h-close listed value. By combining these listed values, we can find a listed tuple s in which each i-th difference differs by h ($|x_i - s_i| \le h$), and thus, the Euclidean distance is bounded:

$$d(x, s) = \sqrt{\sum_{i=1}^{n} (x_i - s_i)^2} \le \sqrt{h^2 + \ldots + h^2 \ (n \text{ times})} =$$

$$h \cdot \sqrt{n} = 2^{-p}.$$

For each of the listed tuples x, we compute $f(x)$ with accuracy $2^{-(m+2)}$, resulting in a rational value $r(x)$ for which $|f(x) - r(x)| \le 2^{-(m+2)}$. Let us now select, as $x^{(1)}, \ldots, x^{(N)}$, those listed tuples x for which $r(x) > 2^{-(m+1)}$.

2°. Let us now prove that $\bigcup\limits_{i=1}^{N} D_r\left(x^{(i)}\right) \subseteq \{x : f(x) > 0\}$.

In other words, let us prove that if $d\left(x, x^{(i)}\right) \le r$, then $f(x) > 0$. Indeed, by our choice of p and $r = 2^{-p}$, the inequality $d\left(x, x^{(i)}\right) \le r$ implies that $\left|f(x) - f\left(x^{(i)}\right)\right| \le 2^{-(m+2)}$.

By the choice of the values $x^{(i)}$, we have $r\left(x^{(i)}\right) > 2^{-(m+1)}$, where, by definition of $r(x)$, we have $\left|r\left(x^{(i)}\right) - f\left(x^{(i)}\right)\right| \le 2^{-(m+2)}$. Thus,

$$f\left(x^{(i)}\right) \ge r\left(x^{(i)}\right) - 2^{-(m+2)} > 2^{-(m+1)} - 2^{-(m+2)} = 2^{-(m+2)}.$$

From $\left|f(x) - f\left(x^{(i)}\right)\right| \le 2^{-(m+2)}$, we can now conclude that

$$f(x) \ge f\left(x^{(i)}\right) - 2^{-(m+2)} > 2^{-(m+2)} - 2^{-(m+2)} = 0,$$

so indeed $f(x) > 0$.

3°. To complete the proof, let us prove that $\{x : f(x) > 2^{-m}\} \subseteq \bigcup\limits_{i=1}^{N} D_r\left(x^{(i)}\right)$.

In other words, we need to prove that if $f(x) > 2^{-m}$, then we have $d\left(x, x^{(i)}\right) \le r$ for some i.

Indeed, let $f(x) > 2^{-m}$. By construction of the listed tuples, there is a listed tuple s for which $d(s, x) \le 2^{-p} = r$. Let us show that the tuple s is

among the tuples $x^{(i)}$. Indeed, by our choice of p and r, we can conclude that $|f(x) - f(s)| \le 2^{-(m+2)}$ and thus, that

$$f(s) \ge f(x) - 2^{-(m+2)} > 2^{-m} - 2^{m+2} = 3 \cdot 2^{-(m+2)}.$$

So, from $|r(s) - f(s)| \le 2^{-(m+2)}$, we conclude that

$$r(s) \ge f(s) - 2^{-(m+2)} > 3 \cdot 2^{-(m+2)} - 2^{-(m+2)} = 2 \cdot 2^{-(m+2)} = 2^{-(m+1)},$$

i.e., that $r(s) > 2^{-(m+1)}$. Satisfying this inequality was exactly the criterion that we used for choosing the points $x^{(i)}$. Since the listed point s satisfies this inequality, we thus conclude that s is one of the points $x^{(i)}$. So, the inequality $d(x, s) \le r$ means that $d\left(x, x^{(i)}\right) \le r$ for an appropriate i.

The statement is proven, and so is the proposition.

5 Computability of the Set-Valued Identification Problem

Practical need for set estimation. In many practical situations, we are interested in the values of physical quantities $x_1, \ldots, x_n$ which are difficult or impossible to measure directly. For example, we may be interested in the spatial coordinates of a robot.

Since we cannot measure these quantities directly, we measure them indirectly, i.e., we measure quantities $y_1, \ldots, y_J$ which are connected to the desired quantities x_i in the known way, as $y_j = f_j(x_1, \ldots, x_n)$. In some situations, instead of this dependence, we know the dependence of the corresponding quantity y_j on the desired quantities x_i and on the auxiliary quantities $c_1, \ldots, c_\ell$ which, in their turn, can be measured directly: $y_j = f_j(x_1, \ldots, x_n, c_1, \ldots, c_\ell)$.

The results $\widetilde{y}_j$ and $\widetilde{c}_k$ of the corresponding measurements are, in general, different from the actual (unknown) values y_j and c_k of these quantities. In many practical situations, the only information that we have about the measurement errors $\Delta y_j \stackrel{\text{def}}{=} \widetilde{y}_j - y_j$ and $\Delta c_k \stackrel{\text{def}}{=} \widetilde{c}_k - c_k$ are the upper bounds Δ_{yj} and Δ_{ck} on their absolute values: $|\Delta y_j| \le \Delta_{yj}$ and $|\Delta c_k| \le \Delta_{ck}$; see, e.g., [8].

In this case, after we get the measurement results $\widetilde{y}_j$ and $\widetilde{c}_k$, the only information that we have about the desired values x_i is that there exists y_j and c_k for which:

- $y_j = f_j(x_1, \ldots, x_n, c_1, \ldots, c_\ell)$ for all j;
- $|y_j - \widetilde{y}_j| \le \Delta_{yj}$ for all j; and
- $|c_k - \widetilde{c}_k| \le \Delta_{ck}$ for all k.

Our goal is to find the set X of all tuples $x = (x_1, \ldots, x_n)$ that are consistent with the measurement results – i.e., which satisfy the above three requirements.

What is known and what we do in this section. There exist several efficient algorithms for computing the set X in several important situations; see, e.g., [4]. In this section, we analyze the algorithmic computability of this problem in the *general* situation, for general computable functions f_j.

Analysis of the problem. The requirement on the tuple x is that for some $c_k \in \mathbf{c}_k \stackrel{\text{def}}{=} [\widetilde{c}_k - \Delta_{ck}, \widetilde{c}_k + \Delta_{ck}]$, we have $\Delta_{yj} - |f_j(x,c) - \widetilde{y}_j| \geq 0$ for all j, i.e., equivalently, that we have $F(x,c) \geq 0$, where we denoted:

$$F(x,c) \stackrel{\text{def}}{=} \min_{j=1,\ldots,J} (\Delta_{yj} - |f_j(x,c) - \widetilde{y}_j|).$$

This requirement, in its turn, can be equivalently represented as $f(x) \geq 0$, where we denoted

$$f(x) \stackrel{\text{def}}{=} \inf_{c_k \in \mathbf{c}_k} F(x,c).$$

In accordance with the above-mentioned results, the functions $F(x,c)$ and $f(x)$ are computable.

Conclusion. The set X of all the states consistent with our knowledge has the form $\{x : f(x) \geq 0\}$. Thus, we can apply the Proposition and conclude that this set is – in some reasonable sense – computable.

To be more precise, we compute a set which is not exactly equal to the desired set $X = \{x : f(x) \geq 0\}$, but which is intermediate between close sets $\{x : f(x) > 2^{-m}\}$ and $\{x : f(x) > 0\}$.

For our identification problem, the condition $f(x) > 2^{-m}$ means, in effect, that we consider bounds $\Delta_{yj} - 2^{-m}$ instead of the original bounds Δ_{yj}. As we have mentioned earlier, for large m, the new bound is practically indistinguishable from Δ_{yj}. So, from the practical viewpoint, our Proposition indeed implies that the set X of all possible states is computable.

6 The Above Algorithms Can Be Extended to the General Case of Quantified Constraints

In the avoidance problem, we considered a constraint of the type $f(t,c) > 0$ for all t, where $f(t,c)$ is a computable function. To find the set of all the values c that satisfy this constraint, we reformulated this constraint in an equivalent for $F(c) > 0$, where the function $F(c) \stackrel{\text{def}}{=} \min\limits_{t} f(t)$ is also computable.

In the set-valued identification problem, we were interested in the constraint of the type $f(t,c) \geq 0$ for *some* t. This constraint was reformulated in an equivalent form $F(c) \geq 0$, for a computable function $F(c) \stackrel{\text{def}}{=} \max\limits_{t} f(t,c)$.

A similar idea can be applied to general quantified constraints, i.e., constraints of the type

$$\forall t_1 \, \exists t_2 \, \ldots \, \forall t_k \, \exists t_{k+1} \, f(t_1, t_2, \ldots, t_k, t_{k+1}, c) > 0$$

or

$$\forall t_1\, \exists t_2\, \ldots\, \forall t_k\, \exists t_{k+1}\, f(t_1, t_2, \ldots, t_k, t_{k+1}, c) \geq 0$$

for some computable function $f(t_1, t_2, \ldots, t_k, t_{k+1}, c)$.

Indeed, the condition $\exists t_{k+1}\, f(t_1, \ldots, t_k, t_{k+1}, c) > 0$ is equivalent to $f_1(t_1, \ldots, t_k, c))$, where the function

$$f_1(t_1, \ldots, t_k, c) \stackrel{\text{def}}{=} \max_{t_{k+1}} f(t_1, \ldots, t_k, t_{k+1}, c)$$

is also computable.

Thus, the condition

$$\forall t_k\, (\exists t_{k+1}\, f(t_1, \ldots, t_{k-1}, t_k, t_{k+1}, c) > 0)$$

is equivalent to $\forall t_k\, f_1(t_1, \ldots, t_{k-1}, c) > 0$ and so, to $f_2(t_1, \ldots, t_{k-1}, c) > 0$, where the function

$$f_2(t_1, \ldots, t_{k-1}) = \min_{t_k} f_1(t_1, \ldots, t_{k-1}, t_k, c) = \min_{t_k} \max_{t_{k+1}} f(t_1, \ldots, t_k, t_{k+1}, c)$$

is also computable.

In general, each quantified constraint is thus equivalent to, correspondingly, $F(c) > 0$ or $F(c) \geq 0$, where

$$F(c) \stackrel{\text{def}}{=} \min_{t_1} \max_{t_2} \ldots \min_{t_k} \max_{t_{k+1}} f(t_1, t_2, \ldots, t_k, t_{k+1}, c)$$

is a computable function.

Thus, by using the above algorithms, we can compute the corresponding sets.

Acknowledgments

This work was supported in part by the National Science Foundation grants CAREER 0953339, HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721, and by an award "UTEP and Prudential Actuarial Science Academy and Pipeline Initiative" from Prudential Foundation. This research was performed during Anthony Welte's visit to the University of Texas at El Paso.

References

[1] E. Bishop, *Foundations of Constructive Analysis*, McGraw-Hill, New York, 1967.

[2] A. Chaputot, J. Alexander dit Sandretto, and O. Mullier, *SYNIBEX*, http://perso.ensta-paristech.fr/č̃haputot/dynibex, 2015.

[3] B. Desrochers and L. Jaulin, "Computing a guaranteed approximation for the zone explored by a robot", *IEEE Transaction on Automatic Control*, 2016.

[4] L. Jaulin, M. Kiefer, O. Dicrit, and E. Walter, *Applied Interval Analysis*, Springer, London, 2001.

[5] O. Heimlich, *GNU Octave Interval Package. Version 1.4.1,* http://octave.sourceforge.net/interval/, 2016.

[6] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1997.

[7] T. Le Mézo, L. Jailin, and B. Zerr, "Inner approximation of a capture basin of a dynamical system", *Abstracts of the 9th Summer Workshop on Interval Methods SWIM'2016*, Lyon, France, June 19–22, 2016.

[8] S. G. Rabinovich, *Measurement Errors and Uncertainty: Theory and Practice*, Springer Verlag, Berlin, 2005.

[9] N. Ramdani and N. S. Nedialkov, "Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint propagation techniques", *Nonlinear Analysis: Hybrid Systems*, 2011, Vol. 5, No. 2, pp. 149–162.

[10] N. Revol, K. Makino, and M. Berz, "Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY", *Journal of Logic and Algebraic Pprogramming*, 2005, Vol. 64, No. 1, pp. 135–154.

[11] K. Weihrauch, *Computable Analysis*, Springer Verlag, Berlin, 2000.

[12] D. Wilczak and P. Zgliczynski, "C^r-Lohner algorithm", *Schedae Informaticae*, 2011, Vol. 20, pp. 9–42.