A Computable Type Theory for Control Systems

Pieter Collins

Abstract— **In this paper, we develop a theory of computable types suitable for the study of control systems. The theory uses type-two effectivity as the underlying computational model, but we quickly develop a type system which can be manipulated abstractly, but for which all allowable operations are guaranteed to be computable. We apply the theory to the study of hybrid systems, reachability analysis, and control synthesis.**

I. Introduction

For almost all important problems in control theory, including systems analysis, control synthesis and verification/validation, the use of digital computers to study mathematical models of the system is ubiquitous and indispensable. For system models in continuous time and space, numerical methods based on approximate floating-point arithmetic and convergent algorithms are usually used. While this is sufficient in most cases to obtain an understanding of the system behaviour and to obtain practical solutions to sufficiently robust control problems, in some cases we would like better guarantees than can be provided by approximate numerical methods. This may be the case if the system is safety-critical, if experimental testing of a proposed controller is infeasible or prohibitively expensive, if the problem is particularly sensitive to small changes in the system or its environment, or if there are reasons to mistrust the results of a numerical analysis. Ideally, we would like to develop numerical methods for which rigorous conclusions can be made about the system properties, such as that a controlled system satisfies its specification.

Unfortunately, as is well-known in the computer science literature, not all problems can be solved by digital computers. Another difficulty which arises when dealing with control systems is that many of the objects we must deal with, such as points in Euclidean space or continuous functions, are uncountable, and so cannot be represented exactly with a finite amount of data. We are therefore confronted with two problems, namely what is the best way to represent the objects of interest in a computational setting, and which operations (system properties) can we effectively compute in terms of these representations?

There are a number of existing approaches to continuous mathematics in which such computational issues are considered, form informal treatments in constructive analysis [BB85], through more formal approaches using domain theory [GHK$^+$03] or locale theory [Joh02], to the low-level theory of type-two effectivity [Wei00] and realizability theory [Bau00]. Each of these approaches has its advantages and disadvantages. Constructive analysis is closest to "working mathematics", and thus is easiest to work with, but the relationship to computation is unclear. The formal approaches of as domain and locale theory are powerful, but the terminology is sometimes obscure and they are not set up directly as a theory of computation (though the relationship with computation can be formalised). The theory of type-two effectivity provides a direct connection with digital computation, but has a cumbersome notation which makes it unwieldy to work with. However, at their core, each of these theories is based on approximation in topological and metric spaces, and the relationship between *infinite* data, representing the object of interest, and *finite* data, representing approximations.

The aim of this paper is twofold. Firstly, to give an exposition of a formal theory of computation in analysis which has a direct relation with machine computation is easy to understand and use for the mathematician or engineer, and powerful enough to deduce computability for several important properties of control systems. We use type-two effectivity to provide a direct link with real computation, but hide the low-level details in a theory of data types which can then be manipulated in a natural way. Secondly, to apply this theory to some important problems in control theory, including the evolution of hybrid systems, reachability analysis and control synthesis. We typically prove that an operator is computable by giving a high-level algorithm in terms of other computable operations.

We note that the purpose of the paper is to discuss issues of *computability*, that is, which problems *can* or *cannot* be effectively solved by digital computers. While it is also important to consider the computational complexity of the problem, and the design and implementation of practical and efficient algorithms, these considerations are beyond the scope of this paper. However, we believe that most problems of global analysis in nonlinear systems are likely to be intrinsically of exponential complexity in the state-space dimension, though for robust problems it should be able to develop rigorous state-space decomposition and reduction techniques to allow realistic industrial examples to be considered.

The paper is organised as follows. In Section II, we develop a theory of computable types for spaces of points, sets and functions. In Section III we apply the type theory to the study of nondeterministic dynamical systems, including multivalued maps and differential inclusions. In Section IV we apply the type theory to the study of control systems.

This research was supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO) Vidi grant 639.032.408.

P. Collins is with the Centrum Wiskunde en Informatica, Postbus 94079, 1090 GB Amsterdam, The Netherlands Pieter.Collins@cwi.nl

II. Computable Analysis and Type Theory

In this section, we develop a computable type theory for the mathematical objects which we need to study control systems. The theory is based on the theory of type-two effectivity of Weihrauch [Wei00], though the development here is more abstract, similar to that of Schröder [Sch02], [BSS07] and Escardo [Esc04].

A. Type theory

In a standard type theory (see [Bar84]), we start from a collection of *base types* in some category, and construct new types as products $X \times Y$ and exponentials (functions) Y^X (alternatively denoted $X \to Y$) in the category. The most important relationship between products and exponentials is an isomorphism between the type $Z^{X\times Y}$ and the type $(Z^Y)^X$ via the bijection $f \mapsto \hat{f}$ defined by $\hat{f}(x)(y) = f(x, y)$. In the category of sets, products correspond to the normal Cartesian product, and exponentials to functions from X to Y. A category is *Cartesian closed* if we can form products and exponentials with the required properties. The category of sets and functions is Cartesian closed.

In developing a computable type theory, we need to give a way of representing or *naming* objects of a type in a computational setting, and in determining which functions between objects are *computable*, in the sense that there is a program transforming a name of the argument into a name of the result. We will use binary sequences as names, and the computable functions will be closed under composition.

We are interested in computable types which are topological spaces, and for which the representation is *admissible* with respect to the topology. The full category of topological spaces and continuous maps is not Cartesian closed, so is unsuitable for developing a type theory. However, the full subcategory of *quotients of countably-based* (QCB) spaces, is Cartesian closed (see [ELS04]), and the resulting types can always be given a computable structure by means of admissible representations [Sch02]. Further, these spaces include all spaces encountered in the study of finite-dimensional control systems, including the real numbers, Euclidean space, manifolds, continuous functions on these spaces, and hyperspaces of open, closed and compact sets.

In order to develop a type theory, we start with some fundamental base types, including types of natural numbers $\mathbb{N}$, integers $\mathbb{Z}$ and real numbers $\mathbb{R}$. We also need a type $\mathbb{X}$ for the state space, and types for the inputs, disturbances and outputs. From these base types, we can construct types of continuous functions $\mathbb{C}(\mathbb{X}, \mathbb{Y})$ as $\mathbb{Y}^{\mathbb{X}}$. In order to construct types representing sets, we will also need the *Sierpinski type* $\mathbb{S}$ with elements $S = \{\top, \uparrow\}$ and open sets $\{\{\,\}, \{\top\}, \{\top, \uparrow\}\}$. Here, $\top$ denotes "true", and $\uparrow$ denotes "divergent" or "don't know", and can be thought of as representing the result of a computation which never halts. The type of open subsets of X can then be identified with the type $\mathbb{S}^{\mathbb{X}}$, since a set $U \subset X$ is open if, and only if, inclusion $x \in U$ defines a continuous function $\in_U : X \to S$.

B. Machine-computability

We fix a finite alphabet Σ, and give the space Σ^ω the product topology. In particular, Σ^ω is a countably-based, locally-compact zero-dimensional Hausdorff space. By considering *type-two* Turing machines [Wei00, Chapter 2] working on streams of data identified with Σ^ω, we define a set of *computable stream functions*, which are partial functions $\eta :\subset \Sigma^\omega \times \cdots \times \Sigma^\omega \to \Sigma^\omega$ whose domain is a G_δ set. (Recall that a G_δ-set is a countable intersection of open sets.) The set of computable stream functions is countable and closed under composition. We also define the uncountable set of *continuous stream functions* to be the continuous partial functions $\eta :\subset \Sigma^\omega \times \cdots \times \Sigma^\omega \to \Sigma^\omega$ with G_δ-domain.

In this paper, we do not need to concern ourselves with the exact definition of computable stream function. However, we will need the following results on continuous and computable functions, which can be found in [Wei00, Chapter 3].

Theorem 2.1:

1) There exists a surjective function $\delta :\subset \Sigma^\omega \to C(\Sigma^\omega; \Sigma^\omega)$ whose range consist of all continuous stream functions and a computable stream function $\varepsilon : \Sigma^\omega \times \Sigma^\omega \to \Sigma^\omega$ such that $\delta(p)(q) = \varepsilon(p, q)$ for all q.
2) There exists a computable function $\Sigma^* \to \Sigma^\omega$ whose range consists of all elements of $\mathrm{dom}(\delta)$ corresponding to machine-computable functions.

C. Computable types

In this section, we show how to define computable structures on mathematical objects, notably topological spaces.

Definition 2.2 (Representation): A *representation* of a set M is a partial surjective function $\delta :\subset \Sigma^\omega \to M$ whose domain is a G_δ set.

Representations δ_1 and δ_2 of M are *equivalent* if there are computable stream functions η_1, η_2 such that $\delta_1 \circ \eta_1 = \delta_2$ and $\delta_2 \circ \eta_2 = \delta_1$.

Definition 2.3 (Computable type): A *computable type* is an equivalent class of pairs (M, δ), where (M, δ_1) is equivalent to (M, δ_2) if δ_1 and δ_2 are equivalent representations. We shall usually denote a computable type with underlying set M by $\mathbb{M}$ or $\mathcal{M}$.

The following definition shows how representations induce a computable structure on general sets.

Definition 2.4 (Computable function): Let $f : M_1 \times \cdots \times M_k \to Y$ be a function, and $\delta_{M_i} :\subset \Sigma^\omega \to M_i$ and $\delta_Y :\subset \Sigma^\omega \to Y$ be representations. Then a continuous stream function $\eta :\subset (\Sigma^\omega)^k \to \Sigma^\omega$ *realises* f if $f(\delta_{X_1}(p_1), \ldots, \delta_{X_k}(p_k)) = \delta_Y(\eta(p_1, \ldots, p_k))$ for all $p_i \in \mathrm{dom}(\delta_i)$. We say f is *computable* if it is realised by a computable stream function.

By a slight abuse of terminology, we will sometimes say that x_0 is computable from $x_1, \ldots, x_k$ if $x_0 = f(x_1, \ldots, x_k)$ for some computable function f.

Recall that a continuous function $\delta : A \to X$ is a *quotient map* if whenever $\phi : A \to Y$ is continuous and $\delta(p) = \delta(q) \implies \phi(p) = \phi(q)$, then there exists a continuous function $g : A \to Y$ such that $\phi = g \circ \delta$. A continuous function $\delta : A \to X$ is *universal*, if whenever $\phi : A \to X$

is continuous, there exists a continuous function $\eta : A \to A$ such that $\phi = \delta \circ \eta$.

Definition 2.5 (Admissible representation): An representation δ of a topological space (X, τ) is *admissible representation* if it is a universal quotient map.

We require an admissible representation of a topological space to be a universal so that it captures all the topological information about a space. We require it to be a quotient map so that continuity on (X, τ) is reflected in Σ^ω. The *standard representations* of a second-countable Kolmogorov (T_0) space as defined in [Wei00, Chapter 3] are always admissible in the above sense.

Theorem 2.6: Let δ_X, δ_Y be admissible representations of spaces X and Y.

1) If $f : X \to Y$ is continuous, then there exists a continuous stream function η such that $\mathrm{dom}(\eta) = \mathrm{dom}(\delta_X)$ and $\delta_Y \circ \eta = f \circ \delta_X$.
2) Suppose $f : X \to Y$, and there exists a continuous stream function η such that $f \circ \delta_X = \delta_Y \circ \eta$. Then f is continuous.

The first part of the theorem says that any continuous function has a realiser; the second implies that any computable function is continuous.

In general there are many non-equivalent representations admissible for a given topological space. The following theorem [Bau00] asserts the existence of a canonical type $\mathbb{R}$ for the real numbers.

Theorem 2.7: There is a unique equivalence class of admissible representation of $\mathbb{R}$ making arithmetic computable, and comparison $<$ semidecidable. We call the corresponding type the *real number type*, denoted $\mathbb{R}$.

D. Types of continuous functions

We now give the main results of computability of operations on continuous functions. The first theorem asserts computability of the fundamental operations on types.

Theorem 2.8: Let X, Y and Z be spaces with admissible representations δ_X, δ_Y and δ_Z respectively. Let $\mathbb{X}$, $\mathbb{Y}$ and $\mathbb{Z}$ denote the types of (X, δ_X), (Y, δ_Y) and (Z, δ_Z). Then there is a computable bijection between the types $\mathbb{C}(\mathbb{X}, \mathbb{Y}; \mathbb{Z})$ and $\mathbb{C}(\mathbb{X}; \mathbb{C}(\mathbb{Y}, \mathbb{Z}))$ taking f to $\hat{f}$, where $\hat{f}(x)(y) = f(x, y)$.

The next theorem asserts computability of function evaluation.

Theorem 2.9: Let X and Y be spaces with admissible representations δ_X and δ_Y. Then there is a representation $\delta_{[X \to Y]}$ of $C(X; Y)$ such that evaluation $\varepsilon(f, x) = f(x)$ is a computable function from $C(X; Y) \times X$ to Y. Further, any two such representations are computably equivalent, so define a canonical type $\mathbb{C}(\mathbb{X}; \mathbb{Y})$.

The last theorem shows that if we can effectively evaluate a function $f : X \to Y$, then we can compute a name. We will repeatedly use this result to prove computability of a function type by showing that it can be effectively evaluated.

Theorem 2.10: Suppose $\mathbb{X}$ and $\mathbb{Y}$ are computable types, and $f : X \to Y$. Then given an algorithm to effectively evaluate $f(x)$ for all $x \in X$, we can effectively compute a name for f in $\mathbb{C}(\mathbb{X}; \mathbb{Y})$

E. Point and set types

We now use the computable function types to derive the set types we need to study systems. In addition to the well-known classes of open, closed and compact sets, we introduce a computable type which we call the *overt set type*. Objects of this type are closed sets, but the data describing an object is different from that of the closed set type.

Definition 2.11 (Types of set):

1) We identify the type of *open sets*, denoted $\mathbb{O}(\mathbb{X})$ with the type of continuous function $\mathbb{X} \to \mathbb{S}$ by $f_U(x) = \top \iff x \in U$.
2) We identify the type of *closed sets*, denoted $\mathbb{A}(\mathbb{X})$ with the type of continuous function $\mathbb{X} \to \mathbb{S}$ by $f_A(x) = \top \iff x \notin A$.
3) We identify the type of *overt sets*, denoted $\mathbb{V}(\mathbb{X})$ with continuous functions $g_A : \mathbb{O}(\mathbb{X}) \to \mathbb{S}$ satisfying $g_A(U \cup V) = g_A(U) \vee g_A(V)$. Then g_A defines a point set A by $x \in A \iff g_A(U) = \top$ whenever $x \in U$. Also, $A = \mathrm{cl}\,\mathrm{rng}(\tilde{g}_A)$ for some $\tilde{g}_A : \mathbb{N} \to X$
4) We identify the type of *compact sets*, denoted $\mathbb{K}(\mathbb{X})$ with continuous functions $h_C : \mathbb{O}(\mathbb{X}) \to \mathbb{S}$ satisfying $h_C(U \cap V) = h_C(U) \wedge h_C(V)$. Then h_C defines a point set C by $x \in C \iff x \in U$ whenever $h_C(U) = \top$. Also, $C = \mathrm{rng}(\tilde{h}_C)$ for some $\tilde{h}_C : \{0,1\}^\omega \to \mathbb{X}$.

Remark 2.12: In applications to control theory, open sets are often used in the *specification* of a system behaviour, such as a set of safe states, or a set of target states. Overt sets are used when we wish to consider *existential* properties of the behaviour of a system, such as the set of points reachable for a particular *choice* of control strategy. Compact sets are used when we with to consider *universal* properties of the behaviour of a system, such as the set of points reachable under *all possible* disturbances.

Most of the basic set-theoretic operations, including union and intersection, are computable without additional assumptions on X. However, we will sometimes need to work in Hausdorff spaces with a computable apartness relation.

Definition 2.13: $\mathbb{X}$ is a *effectively separated* if the function $s : \mathbb{X} \times \mathbb{X} \to \mathbb{S}$, $s(x, y) = \top \iff x \neq y$ is computable.

We have the following computability results on set types. The proof is entirely straightforward using Definition 2.11 and Theorem 2.10.

Theorem 2.14: The following operations are computable:

1) Finite intersection $\mathcal{O} \times \mathcal{O} \to \mathcal{O}$ and arbitrary union $\mathcal{O}^{\mathbb{N}} \to \mathcal{O}$.
2) Finite union $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$ and arbitrary intersection $\mathcal{A}^{\mathbb{N}} \to \mathcal{A}$.
3) Complement $\mathcal{O} \to \mathcal{A}$ and $\mathcal{A} \to \mathcal{O}$.
4) Countable union $\mathcal{V}^{\mathbb{N}} \to \mathcal{V}$ and finite union $\mathcal{K} \times \mathcal{K} \to \mathcal{K}$.
5) Finite intersection $\mathcal{V} \times \mathcal{O} \to \mathcal{V}$ and $\mathcal{K} \times \mathcal{A} \to \mathcal{K}$.
6) Singleton $\mathcal{X} \to \mathcal{V}$ and $\mathcal{X} \to \mathcal{K}$.
7) If $\mathbb{X}$ is effectively separated, closure $\mathcal{O} \to \mathcal{V}$ and identity $\mathcal{K} \to \mathcal{A}$.
8) Closed convex hull $\mathbb{V} \to \mathbb{V}$ and convex hull $\mathbb{K} \to \mathbb{K}$.
9) Preimage $\mathbb{C}(\mathbb{X}; \mathbb{Y}) \times \mathbb{O}(\mathbb{Y}) \to \mathbb{O}(\mathbb{X})$.
10) Image $\mathbb{C}(\mathbb{X}; \mathbb{Y}) \times \mathbb{V}(\mathbb{X}) \to \mathbb{V}(\mathbb{Y})$ and $\mathbb{C}(\mathbb{X}; \mathbb{Y}) \times \mathbb{K}(\mathbb{X}) \to \mathbb{K}(\mathbb{Y})$.

III. Computability for Nondeterministic Systems

We now use the computable type theory developed in the previous section to give some results on computable properties of dynamic systems. When considering solutions of nondeterministic systems, we are often interested in function spaces with set-valued types.

A *multivalued map* is a function $F : X \to \mathcal{P}(Y)$, where $\mathcal{P}(Y)$ is the set of all subsets of Y. For a set $A \subset X$, we define $F(A) = \bigcup\{F(x) \mid x \in A\}$. For $B \subset Y$, we define $F^{\circ -1}(B) = \{x \in X \mid F(x) \cap B \neq \emptyset\}$ and $F^{\bullet -1}(B) = \{x \in X \mid F(x) \subset B\}$. Note that $F^{\bullet -1}(B) = X \setminus F^{\circ -1}(Y \setminus B)$. For $F : X \to \mathcal{P}(Y)$ and $G : Y \to \mathcal{P}(Z)$, we define $G \circ F : X \to \mathcal{P}(Z)$ by $G \circ F(x) = G(F(x))$. We say that F is *lower-semicontinuous* if $F^{\circ -1}(V)$ is open whenever V is open, and *upper-semicontinuous* if $F^{\circ -1}(B)$ is closed whenever B is closed (equivalenty, $F^{\bullet -1}(V)$ is open whenever V is open).

A. Spaces of multiflows

The set of solutions of a dynamic system is the space of continuous functions $\xi : T \to X$, where T is the time domain, and X is the state space. For an autonomous system, we require *time-invariance*, that if ξ is a solution and $s \in T$, then the function defined by $\eta(t) = \xi(t+s)$ is also a solution. We also require the *property of state*, that if ξ and η are solutions with $\xi(s) = \eta(s)$, then there is a solution ζ with $\zeta(t) = \xi(t)$ for $t \leqslant s$, and $\zeta(t) = \eta(t)$ for $t \geqslant s$.

For a deterministic system, there is only one trajectory through a given initial state. The solution space may be represented either as a function $\phi : X \times T \to X$, or as a function $\hat{\phi} : X \to C(T; X)$. By the exponentiation property, the types $\mathbb{X} \times \mathbb{T} \to \mathbb{X}$ and $\mathbb{X} \to \mathbb{C}(\mathbb{T}; \mathbb{X})$ are equivalent.

In a nondeterministic system there may be may different trajectories with the same initial state. If the time domain is $\mathbb{R}$, we call the resulting dynamics a *multiflow*. The simplest way of representing the solution space is as the *behaviour* of the system, which is simply the set of all solutions, $\Phi \in \mathcal{P}(C(T; X))$. However, we can also represent the solution space as the function $\hat{\Phi} : X \to \mathcal{P}(C(T; X))$ such that $\hat{\Phi}(x) = \{\xi \in \Phi \mid \xi(0) = x\}$. Another useful representation is in terms of the finite *reachability* operator, $\tilde{\Phi} : X \times T \to \mathcal{P}(X)$.

The following lemma gives relations between the different representations of multiflows.

Lemma 3.1:

1) The types $\hat{\Phi} : \mathbb{X} \to \mathbb{K}(\mathbb{C}(\mathbb{T}; \mathbb{X}))$ satisfying $\xi(0) = x$ for all $\xi \in \hat{\Phi}(x)$, and the types $\tilde{\Phi} : \mathbb{X} \times \mathbb{T} \to \mathbb{K}(\mathbb{X})$ are equivalent. We can also compute $\hat{\Phi}$ from Φ, and Φ from $\tilde{\Phi}$ if X is compact.
2) if $\mathbb{T} = \mathbb{R}$, the type $\hat{\Phi} : \mathbb{X} \to \mathbb{V}(\mathbb{C}(\mathbb{T}; \mathbb{X}))$ is contains strictly more information than both $\Phi \in \mathbb{V}(\mathbb{C}(\mathbb{T}; \mathbb{X}))$ and $\Phi : \mathbb{X} \times \mathbb{T} \to \mathbb{V}(\mathbb{X})$. The latter are incomparable.

B. Computability theory for multivalued maps

We now consider computability for nondeterministic systems in discrete-time. We first give a result on the computability properties of continuous functions F from X to a hyperspace of subsets of Y; in particular, for $F : X \to \mathcal{V}(Y)$ and $F : X \to \mathcal{K}(Y)$.

Theorem 3.2: The following are computably equivalent:

1) $F : \mathbb{X} \to \mathbb{V}(\mathbb{Y})$, $F^{\circ -1} : \mathbb{O}(\mathbb{Y}) \to \mathbb{O}(\mathbb{X})$ and $F : \mathbb{V}(\mathbb{X}) \to \mathbb{V}(\mathbb{Y})$.
2) $F : \mathbb{X} \to \mathbb{K}(\mathbb{Y})$, $F^{\circ -1} : \mathbb{A}(\mathbb{Y}) \to \mathbb{A}(\mathbb{X})$ and $F : \mathbb{K}(\mathbb{X}) \to \mathbb{K}(\mathbb{Y})$.

We can use these properties to compute the forward-time evolution of discrete-time nondeterministic systems.

Corollary 3.3: The behaviour of a discrete-time system is computable in the following cases:

1) If $f : \mathbb{X} \to \mathbb{X}$, then $\hat{\phi} : \mathbb{X} \to \mathbb{C}(\mathbb{N}, \mathbb{X})$ is computable from f.
2) If $F : \mathbb{X} \to \mathbb{V}(\mathbb{X})$, then $\hat{\Phi} : \mathbb{X} \to \mathbb{V}(\mathbb{C}(\mathbb{N}, \mathbb{X}))$ is computable from F.
3) If $F : \mathbb{X} \to \mathbb{K}(\mathbb{X})$, then $\hat{\Phi} : \mathbb{X} \to \mathbb{K}(\mathbb{C}(\mathbb{N}, \mathbb{X}))$ is computable from F.

C. Computability theory for differential systems

We now consider the computability of systems defined by differential equations or differential inclusions. For simplicity, we assume that X is a Euclidean space, though the results extend to differential manifolds and locally-compact Banach spaces. To prove the results of this section we need to go back to first principles to solve the differential systems, including the classical Arzela-Ascoli and Michael theorems.

Theorem 3.4: Let $f : X \to X$ be locally-Lipschitz continuous. Then the solution operator of $\dot{x} = f(x)$ is computable $\mathbb{C}(\mathbb{X}; \mathbb{X}) \times \mathbb{X} \to \mathbb{C}(\mathbb{R}, \mathbb{X})$.

The proof is essentially standard [DM70]. The locally-Lipschitz condition can be weakened to to simply requiring uniqueness of solutions [Ruo96].

We now turn to nondeterministic differential systems as defined by differential inclusions $\dot{x} \in F(x)$. For an introduction to differential inclusions, see [AC84]. Following the well-known solution concept of Filippov, we may first need to compute the convex hull of the right-hand side. The continuous case was proved in [PVB96], but easily splits into the lower- and upper-semicontinuous cases. Full proofs can be found in [CG09].

Theorem 3.5:

1) Let F be locally-Lipschitz lower-semicontinuous with closed convex values. Then the solution operator $(F, x) \mapsto \hat{\Phi}_F(x)$ of $\dot{x} \in F(x)$ is computable $\mathbb{C}(\mathbb{X}; \mathbb{V}(\mathbb{X})) \times \mathbb{X} \to \mathbb{V}(\mathbb{C}(\mathbb{R}, \mathbb{X}))$.
2) Let F be upper-semicontinuous with compact convex values. Then the solution operator of $\dot{x} \in F(x)$ is computable $\mathbb{C}(\mathbb{X}; \mathbb{K}(\mathbb{X})) \times \mathbb{X} \to \mathbb{K}(\mathbb{C}(\mathbb{R}, \mathbb{X}))$.

IV. Computability in Control Theory

In this section we now give some applications of the computability type theory to problems in control and systems theory. We focus on three problems, namely the evolution of hybrid systems, computation of reachable and viable sets, and control synthesis. Note that using the computable types developed earlier, many of the proofs are almost trivial.

A. Evolution of hybrid systems

A hybrid system comprises continuous evolution interspersed with discrete jumps. Since from Section III, the evolution of continuous- and discrete-time systems are computable, we focus on the problem of *event detection*. An event occurs whenever the state of the system crosses a *guard set* G Consider the case of a hybrid system defined as the tuple (X, F, G, R), where $\dot{x} \in F(x)$ defines the continuous dynamics, the guard set G is crossed when $g(x) = 0$, and the reset is given by $x' \in R(x)$.

Suppose $\xi(t)$ is a continuous trajectory with $g(\xi(0)) < 0$, and $g(\xi(t)) > 0$ for some $t > 0$. Then clearly the trajectory ξ crosses the guard set at some time. We define the *hitting time* τ_h by $\tau_h(\xi) = \sup\{t \in \mathbb{R} \mid g(\xi(t)) < 0\}$ and the *crossing time* τ_c as $\tau_c(\xi) = \inf\{t \in \mathbb{R} \mid g(\xi(t)) > 0\}$. Clearly $\tau_h(\xi) \leqslant \tau_c(\xi)$, but the two need not be equal, in general. If $\tau_h(\xi) = \tau_c(\xi)$, then we say that ξ crosses g *transversely* at $\tau = \tau_h(\xi)$. Otherwise, it may be the case that $\xi(t)$ slides along the guard set G between τ_h and τ_c, or grazes G before crossing later. We define the *touching time set* as $\tau(\xi) = \{t \in \mathbb{R} \mid g(\xi(t)) = 0 \wedge \forall s \leqslant t,\ g(\xi(s)) \leqslant 0\}$.

Lemma 4.1: The set of trajectories ξ with $g(\xi(0)) < 0$ and $g(\xi(t)) > 0$ for some $t > 0$ is computable in $\mathbb{O}(\mathbb{C}(\mathbb{R}; \mathbb{X}))$.

Theorem 4.2: The touching time set $\tau(g, \xi)$ is computable as a function $\mathbb{C}(\mathbb{X}, \mathbb{R}) \times \mathbb{C}(\mathbb{R}, \mathbb{X}) \to \mathbb{A}(\mathbb{R})$. If $g(\xi(t)) > 0$ for some $t > 0$, then $\tau(g, \xi)$ can also be computed in $\mathbb{K}(\mathbb{R})$.

Proof: Define $\gamma_{\xi,g}(t) = g(\xi(t))$ and $\mu_{\xi,g}(t) = \sup\{g(\xi(s)) \mid s \in [0, t]\}$ which is a computable function (see [Wei00]). Then $\tau(g, \xi) = \gamma^{-1}_{\xi,g}(\{0\}) \cap \mu^{-1}_{\xi,g}((-\infty, 0])$ so is computable. If $g(\xi(t)) > 0$, then $\tau(g, \xi) = \overline{\tau(g, \xi)} \cap [0, t]$, so is effectively compact. ∎

Theorem 4.3: Consider a hybrid system where F is convex-compact-valued, and R is compact-valued. Then set of points $\Psi(X_0)$ reachable after the first event is computable as a compact set.

Proof: The set of points reachable after the first event of a continuous solution ξ is $R(\xi(\tau(g, \xi))$, which is computable in $\mathbb{K}(\mathbb{X})$ from $\xi \in \mathbb{C}(\mathbb{R}; \mathbb{X})$. The set of trajectories with inital condition X_0 is $\Phi_F(X_0)$, so is computable in $\mathbb{K}(\mathbb{C}(\mathbb{R}; \mathbb{X}))$. Then $\Psi(X_0)$ is the union of $R(\xi(\tau(g, \xi))$ for ξ in the compact set $\Phi_F(X_0)$, so is computable in $\mathbb{K}(\mathbb{X})$. ∎

Unfortunately, the set of points reachable after the first event is not computable as an overt set. In this case, event detection is easier in the context of backwards reachability

Theorem 4.4: Consider a hybrid system where Φ_F is an overt multiflow and R is overt-valued. Let V be an open set. Define $\Psi^{\circ -1}(V)$ to be the set of points for which there is a solution for which the state is in V immediately after the first jump. Then $\Psi^{\circ -1}(V)$ is computable as an open set.

Proof: The trajectory ξ crosses G in $R^{-1}(V)$ if $\xi(\tau(g, \xi)) \subset V$. Since $R : \mathbb{X} \to \mathbb{V}(\mathbb{X})$, $U = R^{\circ -1}(V)$ is computable in $\mathbb{O}(X)$. Since $\tau(g, \xi)$ is compact and ξ is continuous, $W = \{\xi \mid \tau(g, \xi) \subset U\}$ is computable in $\mathbb{O}(\mathbb{C}(\mathbb{R}; \mathbb{X}))$. Since $\Phi : \mathbb{X} \to \mathbb{V}(\mathbb{C}(\mathbb{R}; \mathbb{X}))$, $\Phi^{\circ -1}(W)$ is computable in $\mathbb{O}(\mathbb{X})$. We have $\Psi^{\circ -1}(V) = \{x \mid \exists \xi \in \Phi_F(x) \text{ s.t. } \xi(\tau(g, \xi)) \subset R^{-1}\}) = \Phi^{\circ -1}_F(\{\xi(\tau(g, \xi)) \subset R^{\circ -1}\})$, so $\Psi^{\circ -1}(V)$ is computable in $\mathbb{O}(\mathbb{X})$. ∎

B. Reachable and viable sets

We now apply the results of Section III-B to prove computability of some infinite-time operators in discrete-time dynamical systems. Computability of the viability kernel was considered in [SP94], and of reachable sets was considered in [Col05]. Some extensions to hybrid systems are given in [ALQS02], [GT06].

We will need the following result, which shows that we can separate compact and closed sets.

Lemma 4.5: There is a recursively enumerable set D of pairs $(A^\circ, \overline{A}) \in \mathbb{O} \times \mathbb{K}$ such that for any compact K and open U, there exist $(A^\circ, \overline{A})$ such that $K \subset A^\circ$ and $\overline{A} \subset U$.

We define the *reachable set* of a system $F : X \to \mathcal{P}(X)$ with initial state set X_0 as

$$\operatorname{reach}(F, X_0) = \{x \in X \mid \exists \text{ solution } \xi \text{ of } F \text{ and } t \in T \text{ with } \xi(0) \in X_0 \text{ and } \xi(t) = x\}.$$

Theorem 4.6: The reachable set operator reach is computable as a function $\mathbb{C}(\mathbb{X}; \mathbb{V}(\mathbb{X})) \times \mathbb{V}(\mathbb{X}) \to \mathbb{V}(\mathbb{X})$, but not as a function $\mathbb{C}(\mathbb{X}; \mathbb{K}(\mathbb{X})) \times \mathbb{K}(\mathbb{X}) \to \mathbb{K}(\mathbb{X})$.

Proof: We can write $\operatorname{reach}(F, X_0) = \bigcup_{i=0}^{\infty} R_i$, where $R_0 = X_0$ and $R_{i+1} = R_i \cup F(R_i)$. Then $\operatorname{reach} : C(X; \mathcal{V}(X)) \times \mathcal{V}(X) \to \mathcal{V}(X)$ is computable since all operations are computable. However, reach fails to be computable from $C(\mathbb{X}; \mathbb{K}(\mathbb{X})) \times \mathbb{K}(\mathbb{X})$ to $\mathbb{K}(\mathbb{X})$ even if X is compact since it is easy to show that reach is not upper-semicontinuous in parameters, as in Example 4.7. ∎

Example 4.7: Consider the system $f : \mathbb{R} \to \mathbb{R}$ defined by $f_\epsilon(x) = \epsilon + x + x^2 - x^4$. Then $\operatorname{reach}(f_0, \{-1/2\}) \subset [-1, 0]$, but $\operatorname{reach}(f_\epsilon, \{-1/2\}) \not\subset [-1, 1/2]$ for any $\epsilon > 0$.

We define the *chain-reachable set* of F as limit of all ϵ-orbits, or equivalently as

$$\operatorname{chain\,reach}(F, X_0) = \bigcap \{U \in \mathcal{O}(X) | \operatorname{cl}(U) \text{ is compact, and } X_0 \cup F(\operatorname{cl}(U)) \subset U\}.$$

Theorem 4.8:

Proof: It is clear that $\operatorname{chain\,reach}(F, X_0) = \bigcap\{\overline{A} \mid (A^\circ, \overline{A}) \in D \text{ and } X_0 \cup F(\overline{A}) \subset A^\circ\}$, proving computability. The proof of optimality involves considering perturbations, and can be found in [Col07]. ∎

The *viability kernel* of a multivalued map F and a set S is given by

$$\operatorname{viab}(F, S) = \{x \in X \mid \exists \text{ solution } \xi \text{ s.t. } x = \xi(0) \text{ and } \forall t \in T,\ x(t) \in S\}.$$

Theorem 4.9: The viability kernel operator $\operatorname{viab}(F, S)$ is computable as a function $\mathbb{C}(\mathbb{X}, \mathbb{K}(\mathbb{X})) \times \mathbb{A}(\mathbb{X}) \to \mathbb{A}(\mathbb{X})$.

Proof: Write $\operatorname{viab}(F, A) = \bigcap_{i=0}^{\infty} S_i$, where $S_0 = S$ and $S_{i+1} = S_i \cap F^{\circ -1}(S_i)$. ∎

The viability kernel is not computable as an open or overt set. However, we can define a *robust viability kernel*

$$\operatorname{robust\,viab}(F, S) = \{C \in \mathcal{K}(X) \mid C \subset S \cap F^{\circ -1}(\operatorname{int}(C))\}.$$

Theorem 4.10: The robust viability kernel operator $\operatorname{robust\,viab}(F, S)$ is computable as a function $\mathbb{C}(\mathbb{X}, \mathbb{V}(\mathbb{X})) \times \mathbb{O}(\mathbb{X}) \to \mathbb{O}(\mathbb{X})$.

Proof: Write $\operatorname{robust\,viab}(F, S) = \bigcup\{A^\circ \mid (A^\circ, \overline{A}) \in D \text{ and } \overline{A} \subset S \cap F^{\circ -1}(A^\circ)\}$. ∎

C. Control synthesis

A *noisy control system* with state space X, input space U and noise space V is a function $f : X \times U \times V \to X$. We assume that U is an overt space and V a compact space, and define $F_U : X \to \mathcal{P}(X \times U)$, $F_U(x) = \{(x, u) \mid u \in U\}$, and $F_V : X \times U \to \mathcal{P}(X)$ by $F_V(x, u) = \{f(x, u, v) \mid v \in V\}$.

The *controllable set* of $\mathrm{ctrl}(f, T, S)$ with target set T and safe set S is determined recursively by $T_0 = T \cap S$ and $T_{i+1} = T_i \cup \{x \in X \mid \exists u \in U,\ \forall v \in V,\ f(x, u, v) \in T_i\} \cap S$.

Theorem 4.11: The controllable set operator $\mathrm{ctrl} : \mathbb{C}(\mathbb{X}, \mathbb{U}, \mathbb{V}; \mathbb{X}) \times \mathbb{O}(\mathbb{X}) \times \mathbb{O}(\mathbb{X}) \to \mathbb{O}(\mathbb{X})$ is computable.

Proof: The functions $F_U : X \to \mathcal{V}(X \times U)$ and $F_V : (X \times U) \to \mathcal{K}(X)$ can be computed from f, U and V. Write $T_{i+1} = T_i \cup (F_U^{\circ -1}(F_V^{\bullet -1}(T_i)) \cap S)$ and $C = \bigcup_{i=0}^{\infty} T_i$. ∎

A *state feedback control law* is a function $g : X \to U$. Since there are systems which are controllable by a discontinuous state feedback, but not a continuous feedback, we first compute a *supervisor*, which is a multivalued function $G : X \rightrightarrows U$ such that taking $u_n \in G(x_n)$ always gives a valid trajectory. If G is open-valued, we can then construct a deterministic feedback law by taking $g(x) \in G(x)$.

Theorem 4.12: If $\mathrm{ctrl}(f, T, S) \supset X_0$, then there is a computable supervisor $G : \mathbb{X} \to \mathbb{O}(\mathbb{U})$.

Proof: From $T_{i+1} = T_i \cup F_U^{\circ -1}(F_V^{\bullet -1}(T_i))$ and $X_0 \subset T_n$, find open B_i, C_i such that $\overline{C}_0 \subset T$, $X_0 \subset \bigcup_{i=0}^{n} B_i$, and for all i, $\overline{C}_i \subset B_i$ and $B_{i+1} \subset F_U^{\circ -1}(F_V^{\bullet -1}(C_i))$. For $x \in B_i \setminus \bigcup_{j=0}^{i-1} \overline{C}_j$, define $G_i(x) = \{u \in U \mid (x, u) \in F_V^{\bullet -1}(C_{i-1})\}$, and define $G(x) = \bigcup \{G_i(x) \mid x \in B_i \setminus \bigcup_{j=0}^{i-1} \overline{C}_j\}$. ∎

A system with *partial observations* with output space Y and measurement noise space W is defined by $f : X \times U \times V \to X$ and $h : X \times W \to Y$. Define $H(x) = \{h(x, w) \mid w \in W\}$. An *observer* for the system takes values $\hat{X} \in \mathcal{K}(X)$, with with initialisation $\hat{X}_0 = X_0 \cap H^{\circ -1}(y_0)$ and update rule $\hat{X}_{n+1} = \hat{F}(\hat{X}_n, u_n, y_{n+1}) = F_V(\hat{X}_n, u_n) \cap H^{\circ -1}(y_{n+1})$.

In order to guarantee control to the target set within the safe set, we require $\hat{X}_n \subset T$ for some n, and $\hat{X}_i \subset S$ for all $i \leqslant n$. We therefore define sets $T_i \subset \mathcal{K}(X)$ by $T_0 = \{C \in \mathcal{K}(S) \mid C \subset T\}$ and $T_{i+1} = \{C \in \mathcal{K}(S) \mid \exists u \in U,\ \forall y \in Y,\ \hat{F}(C, u, y) \in T_i\}$. Note that $\{C \in \mathcal{K}(X) \mid C \subset U\}$ is open in $\mathcal{K}(X)$ for U open in X. Then T_{i+1} is the set of all state estimates for which we can choose an input such that the next state estimate is guaranteed to be in T_i. The problem is solvable if, and only if, there exists n such that $\bigcup_{i=1}^{n} T_i \supset \{C \in \mathcal{K}(X) \mid \exists y \in Y \text{ s.t. } C = X_0 \cap H^{\circ -1}(y)\}$.

The following theorem is given in [Col08].

Theorem 4.13: Define the controllable set operator ctrl by $X_0 \in \mathrm{ctrl}(f, h, S, T)$ if X_0 is controllable into T within S under the system (f, h). Then $\mathrm{ctrl} : \mathbb{C}(\mathbb{X}, \mathbb{U}, \mathbb{V}; \mathbb{X}) \times \mathbb{C}(\mathbb{X}, \mathbb{W}; \mathbb{Y}) \times \mathbb{O}(\mathbb{X}) \times \mathbb{O}(\mathbb{X}) \to \mathbb{O}(\mathbb{K})$ is computable.

In a similar way to the case of state feedback, we can construct a supervisor $G : \mathbb{K}(\mathbb{X}) \to \mathbb{O}(\mathbb{U})$ solving the control problem. However, in order to realise the control strategy, we need to replace the state estimator update $\hat{F}$ by a finite automaton approximation, and the supervisor G by a function on this automaton. This process is not entirely straightforward since the space $\mathcal{K}(X)$ is not Hausdorff.

V. Conclusions

In this paper, we have developed a computable type theory sufficient for allowing the analysis of control systems. We have studied systems properties such as reachability and viability, problems of control synthesis, and event detection in hybrid systems. Further extensions involve infinite-dimensional systems and stochastic systems, verification of system properties given by temporal logics, and consideration of issues of computational complexity. Work is in progress to provide efficient implementations of the computable operators given here in the tool Ariadne [BCC+06].

References

[AC84] Jean-Pierre Aubin and Arrigo Cellina. *Differential inclusions*. Springer-Verlag, 1984.

[ALQS02] Jean-Pierre Aubin, John Lygeros, Marc Quincampoix, and Shankar Sastry. Impulse differential inclusions: A viability approach to hybrid systems. *IEEE Trans. Automatic Control*, 47(1):2–20, 2002.

[Bar84] H. P. Barendregt. *The lambda calculus*. North-Holland Publishing Co., 1984.

[Bau00] Andrej Bauer. *The Realizability Approach to Computable Analysis and Topology*. PhD thesis, Carnegie Mellon University, 2000.

[BB85] Errett Bishop and Douglas Bridges. *Constructive analysis*. Springer-Verlag, 1985.

[BCC+06] Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proceedings of MTNS*, pages 1269–1267, 2006.

[BSS07] Ingo Battenfeld, Matthias Schröder, and Alex Simpson. A convenient category of domains. In *Computation, Meaning, and Logic*, volume 172 of *Electron. Notes Theor. Comput. Sci.*, pages 69–99. Elsevier, 2007.

[CG09] Pieter Collins and Daniel Graça. Effective computability of solutions of differential inclusions — the ten thousand monkeys approach. *J. Universal Comp. Sci.*, 15(6):1162–1185, 2009.

[Col05] Pieter Collins. Continuity and computability of reachable sets. *Theoret. Comput. Sci.*, 341(1-3):162–195, 2005.

[Col07] Pieter Collins. Optimal semicomputable approximations to reachable and invariant sets. *Theory Comput. Syst.*, 41(1):33–48, 2007.

[Col08] Pieter Collins. Computability of controllers for discrete-time semicontinuous systems. In *Proceedings of the MTNS*, 2008.

[DM70] J. W. Daniel and R. E. Moore. *Computation and Theory in Ordinary Differential Equations*. Freeman, 1970.

[ELS04] Martín Escardó, Jimmie Lawson, and Alex Simpson. Comparing Cartesian closed categories of (core) compactly generated spaces. *Topology Appl.*, 143(1-3):105–145, 2004.

[Esc04] Martín Escardó. Synthetic topology of data types and classical spaces. *Electronic Notes in Theoretical Computer Science*, 87:21–156, 2004.

[GHK+03] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. *Continuous lattices and domains*. Cambridge University Press, 2003.

[GT06] R. Goebel and A. R. Teel. Solutions to hybrid inclusions via set and graphical convergence with stability theory applications. *Automatica J. IFAC*, 42(4):573–587, 2006.

[Joh02] Peter T. Johnstone. *Sketches of an elephant: a topos theory compendium. Vol. 1*. Oxford University Press, 2002.

[PVB96] Anuj Puri, Pravin Varaiya, and Vivek Borkar. Epsilon-approximation of differential inclusions. In *Hybrid Systems III*, volume 1066 of *LNCS*, pages 362–376. Springer, 1996.

[Ruo96] K. Ruohonen. An effective Cauchy-Peano existence theorem for unique solutions. *Internat. J. Found. Comput. Sci.*, 7(2):151–160, 1996.

[Sch02] Matthias Schröder. *Admissible Representations for Continuous Computations*. PhD thesis, FernUniversität Hagen, 2002.

[SP94] Patrick Saint-Pierre. Approximation of the viability kernel. *Appl. Math. Optim.*, 29(2):187–209, 1994.

[Wei00] Klaus Weihrauch. *Computable analysis*. Springer-Verlag, 2000.