21st International Symposium on
Mathematical Theory of Networks and Systems
July 7-11, 2014. Groningen, The Netherlands

Input-Output Representations for Verifying Safety Properties

P.J. Collins

Abstract— **In this paper we consider the problem of finding input-output representations of discrete-time subsystems which contain sufficient information to effectively prove safety properties in a closed-loop system. The main difficulty is that safety requires knowledge of the behaviour over unbounded time intervals. We show that regular expressions over discretised inputs and outputs provide sufficient information, and that these can be effectively computed from a state-space representation given natural conditions on the system and the input, state and output spaces.**

Index Terms— **Safety verification, input-output system, regular expression, computability theory**

I. Introduction

In this paper, we consider the problem of finding an input-output representation of discrete-time control systems which contains sufficient information to computationally verify safety, stabilisation and recurrence properties.

The motivation for this problem is to support the development of software tools for the automated verification of large-scale sytems of heterogeneous components. For such systems, one wishes to have a canonical description of components depending only on the input and output spaces and the external behaviour of the component, but not explicity on the state space or the details of state space model. Indeed, by giving a generally-applicable framework, one is free to change the of the state-space model for a component, such as by providing a more detailed model if extra information about the system becomes available or is needed for the analysis of sensitive system properties, without changing the rest of the system.

The difficulty in finding such a representation lies in the nature of the verification problems considered. To verify safety, we need access to information about the infinite-time behaviour in a finite amount of data. In particular, even for discrete systems, the information provided by trajectory traces is insufficient, since an algorithm can only examine a finite piece of the trajectory in finite time. Direct use of algebraic representations such as Fliess series [] suffers from similar problems, and symbolic approaches can only be used for restricted classes of system.

The approach used here is natural, and proceeds in two stages. Firstly, we discretise the input, state and output spaces, and obtain a discrete automaton representation of the component. By using finer discretisations of the spaces, and if necessary, using more accuracy when computing the discrete automaton, we obtain a sequence of discrete automata representing the behaviour of the original sytem arbitrarily closely. We then represent the input-output behaviour of the discrete automata by regular expressions, which provide a description of the external behaviour without reference to the state space. The use of regular expressions to describe the input-output map is critical, since it is these which give useful information about the behaviour over *infinite* time intervals.

Since we aim to show that the use of regular expressions are sufficient to prove *any* verifyable property of the original state-space model, we need to use an appropriate theory of computation. We use the computable type theory developed in [2], which is a simplification of the theory of type-two effectivity of [10]. In the computable type theory, each class of mathematic objects (real numbers, continuous functions, open/closed/compact sets) is described by a *type* with a natural *representation* defining the information needed to describe an arbitrary object. Since the objects under consideration lie in uncountable sets, an infinite amount of information is needed to describe an arbitrary element; this information is provided by a *data stream* i.e. a sequence of symbols from some finite alphabet. A function or operator between types is *computable* if a data stream describing the result can be output by a digital computer (Turing machine) given any data stream descriping the arguments. Given the correct types for the system description, most system properties are *semidecidable* in general, but are decidable for robust instances. In this paper however, we will only be concerned with verification questions.

Similar discretisation approaches have been used in the study of dynamic systems, including [1] for optimal control, in [4] for discrete-time systems, and [5] in the context of hybrid systems. The use of regular expressions to describe the behaviour of finite automata is standard, as are algorithms to decide system-theoretic properties. The main novelty of this paper is to combine discretisation, refinement and the use of regular expressions, and the main contribution is to show that this gives a methodology sufficient to verify any robust safety, stabilisation or recurrence property. We also show that a direct representation of the input-output map is sufficient to prove properties related to reachability of a target set, but is insufficient to prove safety. We also introduce concepts from computable analysis, notably to explicitly discuss general conditions under which discretisations can be effectively computed.

The goal of working with external representations of a system is also shared by the behavioural approach of [8]. However, from the point of view of computability theory, the behavioural approach is insufficient unless the external space is a compact Hausdorff space. Further, in this paper

P.J. Collins is affiliated with the Departmet of Knowledge Engineering, Maastricht University, Postbus 616, 6200MD Maastricht, The Netherlands. E-mail: pieter.collins@maastrichtuniversity.nl.

ISBN: 978-90-367-6321-9

we use strict causality of the input-output mapping, which distinguishes the inputs from the outputs.

The paper is organised as follows. In Section II, we outline the existing theory of discrete-time input-output systems, regular expressions and computable analysis which we will use in the sequel. In Section IV, we formalise the discretisation of a system over continuous spaces. In Section V, we show how to represent the behaviour of the discretised system by regular expressions. Conclusions and perspectives for further work are given in Section VI. We mostly provide full proofs, but some technical results are only sketched.

II. Preliminaries

We write X^* to denote the set of words with elements in X. We write X^ω to denote the set of sequences with elements in elements in X, and denote the sequence $(x_0, x_1, x_2, \ldots)$ by $\vec{x}$. Given sequences $\vec{x}_1 \in X_1^\omega$ and $\vec{x}_2 \in X_2^\omega$, the product $(\vec{x}_1, \vec{x}_2) \in X_1^\omega \times X_2^\omega$ of is equivalent to the sequence of pairs $(x_{1,i}, x_{2,i}) \in (X_1 \times X_2)^\omega$. We write $\vec{x}|_k$ to denote the word $(x_0, x_1, \ldots, x_k)$.

We will use the notion of multivalued functions $F : X \to \mathcal{P}(Y)$, which we denote $F : X \rightrightarrows Y$. The *graph* of F is $\mathrm{graph}(F) = \{(x, y) \in X \times Y \mid F(x) \ni y\}$. If $F : X \rightrightarrows Y$ and $A \subset X$, then $F(A) = \bigcup\{F(x) \mid x \in A\}$. Composition of multivalued functions $F : X \rightrightarrows Y$ and $G : Y \rightrightarrows Z$ is defined by $G \circ F(x) := \bigcup\{G(y) \mid y \in F(x)\}$.

A. Input-output systems

A *nondeterministic state-space system* with input space U, output space Y and state space X consists of a tuple $S = (F, H, W)$ such that

$$F : X \times U \rightrightarrows X, \quad H : X \rightrightarrows Y, \quad W \subset X. \tag{1}$$

The system is *non-blocking* if W is nonempty, and F, H have nonempty values. Given input sequence $\vec{u}$, a valid *trajectory* of (1) is a sequence $\vec{x}$ such that $x_0 \in W$ and $\forall n,\ x_{n+1} \in F(x_n, u_n)$. Given trajectory $\vec{x}$, a valid output sequence $\vec{y}$ satisfies $\forall n,\ y_n \in H(x_n)$.

A *nondeterministic input-output map* with input space U and output space Y is a function

$$M : U^\omega \rightrightarrows Y^\omega \tag{2}$$

satisfying the *strict causality* condition

$$\vec{u}_1|_k = \vec{u}_2|_k \implies M(\vec{u}_1)|_{k+1} = M(\vec{u}_2)|_{k+1} \tag{3}$$

The input-output mapping of the nondeterministic state-space system (1) is given by

$$M(\vec{u}) = \{\vec{y} \in Y^\omega \mid \exists\, \vec{x} \in X^\omega \text{ s.t. } x_0 \in W \text{ and } \forall n \in \mathbb{N},\ x_{n+1} \in F(x_n, u_n) \text{ and } y_n \in H(x_n)\}.$$

It is immediate that the input-output map is *strictly causal* i.e. outputs depend only on past inputs.

If the input space U is a singleton, then the system is a *closed-loop system*. The dynamic F of the state-space representation reduces to a multivalued function $F : X \rightrightarrows X$, and the input-output mapping reduces to a set of outputs $O \subset Y^\omega$.

If $U_1 = Y_2$ and $Y_2 = U_1$, then the *parallel composition* of input-ouput mappings $M_1 : U_1^\omega \rightrightarrows Y_1^\omega$ and $M_2 : U_2^\omega \rightrightarrows Y_2^\omega$, denoted $M_1||M_2$, is the output set

$$M_1||M_2 := \{(\vec{y}_1, \vec{y}_2) \equiv \vec{y}_{12} \mid \vec{y}_1 \in M_1(\vec{y}_2) \wedge \vec{y}_2 \in M_2(\vec{y}_1)\}$$

If $S_1 = (F_1, H_1, W_1)$ and $S_2 = (F_2, H_2, W_2)$ are state-space systems, then the parallel composition $S_1||S_2$ is the closed-loop system with state space $X_{12} = X_1 \times X_2$ and output space $Y_{12} = Y_1 \times Y_2$ defined by

$$F_{12}(x_1, x_2) = \{(x_1', x_2') \mid x_1' \in F_1(x_1, H_2(x_2)) \wedge x_2' \in F_2(x_2, H_1(x_1))\}.$$
$$H_{12}(x_1, x_2) = \{(y_1, y_2) \mid y_1 \in H_1(x_1) \wedge y_2 \in H_2(x_2)\}.$$

It is straightforward to show that the output of $S_1||S_2$ is the same as the parallel composition of the input-output maps.

If S_1 is a system with input-output mapping M_1, then we define $S_1||M_2 := M_1||M_2$.

We are interested in verifying properties of closed-loop systems. In all cases, we wish to show that some property holds for *all* possible output sequences $\vec{y} \in O$. In the notation of temporal logic [6], properties are expressed in terms of formulae Φ describing a set $S_\Phi \subset Y$.

- The *target property* $\forall\Diamond\Phi$ for target set $T_\Phi \subset Y$ is satisfied if $\forall\, \vec{y} \in O$, there exists $n \in \mathbb{N}$ such that $y_n \in T_\Phi$. We write $O \models \forall\Diamond T_\Phi$.
- The *safety property* $\forall\Box\Phi$ for safe set $S_\Phi \subset Y$ is satisfied if $\forall\, \vec{y} \in O$, $y_n \in S_\Phi$ for all n.
- The *stabilisation property* $\forall\Diamond\Box\Phi$ is satisfied if $\forall\, \vec{y} \in O$, there exists m such that $y_n \in S_\Phi$ for all $n \geq m$.

Note that the *recurrence property* $\forall\Box\Diamond\Phi$ is equivalent to $\forall\Box\forall\Diamond\Phi$, but stabilisation properties cannot be reformulated in this way since $\forall\Diamond\Box\Phi \not\Longrightarrow \forall\Diamond\forall\Box\Phi$ as shown in [3].

B. Regular expressions and finite automata

Regular expressions are an important way of describing the behaviour of discrete automata [9], [7]. A *regular expression* over a finite alphabet Z is a string of symbols defined recursively:

- The *empty string* ε is a regular expression.
- The *literal* z is a regular expression for all $z \in Z$.

For all regular expressions R, R_1, R_2,

- The *catenation* R_1R_2 (or $R_1{\cdot}R_2$) is a regular expression.
- The *alternation* $R_1|R_2$ is a regular expression.
- The *Kleene star* R^* is a regular expression.

Additionally, we use parentheses to indicate precedance, so

- (R) is a regular expression.

A regular expression R denotes a *regular language* $L(R) \subset Z^*$ as follows: $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$, $L(R_1R_2) = \{u_1u_2 \mid u_1 \in L(R_1) \wedge u_2 \in L(R_2)\}$, $L(R_1|R_2) = L(R_1) \cup L(R_2)$, and $L(R^*) = \bigcup_{k\in\mathbb{N}}\{u_1 \cdots u_k \mid \forall i,\ u_i \in L(R)\}$.

The class of regular languages is closed under complementation and intersection. Further, regular expressions representing $\overline{L(R)} := Z^* \setminus L(R)$ and $L(R_1) \cap L(R_2)$ can be effectively constructed. Given regular expressions R_1 and R_2, the language inclusion predicate $L(R_1) \subset L(R_2)$ is decidable, and hence so is equality $L(R_1) = L(R_2)$.

In this paper, we will be interested in ω-*languages*, which are sets of sequences rather than sets of words. A regular ω-expression is a string defined recursively: If R is a regular expression and S, S_1, S_2 are regular ω-expressions,

- The *catenation* RS is a regular ω-expression.
- The *alternation* $S_1|S_2$ is a regular ω-expression.
- The *sequentialisation* R^ω is a regular ω-expression, as long as $\varepsilon \notin L(R)$.

The language $L(R^\omega)$ is defined $\{u_0u_1u_2\cdots \mid \forall i,\ u_i \in R\}$.

The class of regular languages is precisely the class recognised by finite *automata*. A (nondeterministic) finite automaton A consists of a finite set Q of states, a transition relation $T : Q \times Z \rightrightarrows Q$, a set Q_0 of *initial* states, and a set Q_f of *accepting* states. A word $z_0z_1\cdots z_{n-1}$ is *accepted* by A if there exists a list of states $q_0, q_1, \ldots, q_k$ such that $q_0 \in Q_0$, $q_{k+1} \in T(q_k, z_k)$ for all k, and $q_n \in Q_f$. The language $L(A)$ *recognised* by A is the set of all accepted words. Similarly, a sequence $\vec{z}$ is accepted by A using the *Büchi acceptance criterion* if there is a sequence $\vec{q}$ such that $q_0 \in Q_0$, $q_{k+1} \in T(q_k, z_k)$ for all k, and $\{n \in \mathbb{N} \mid q_n \in Q_f\}$ is infinite.

Given a regular expression or regular ω-expression, it is straightforward to construct an automaton recognising its language, and vice-versa [9], [7].

C. Computable analysis

In computability theory, computations are performed by *Turing machines*, which are finite-state machines with access to an infinite memory. In *computable analysis* [10], one works with objects of uncountable sets, and needs to use descriptions in some sequence space Σ^ω, where Σ is a finite alphabet. Computation is an infinite process, and yields a continuous partial function $\eta : \Sigma^\omega \rightharpoonup \Sigma^\omega$.

In order to relate computation over Σ^ω to a set of mathematical objects X, one describes objects of X via a *representation*, which is a partial surjective function $\delta : \Sigma^\omega \rightharpoonup X$. A sequence $\vec{p} \in \Sigma^\omega$ such that $\delta(p) = x$ is a δ-*name* of X. A representation δ must provide useful information about an element $x \in X$ from a *finite* portions of a δ-name $\vec{p}$ of x. Typically, this means that δ is *admissible* with respect to the induced *quotient topology* on X, which essentially means that a δ-name of x *only* gives information about the open sets containing x.

A function $f : X \to Y$ is *computable* if there is a machine-computable function $\eta : \Sigma^\omega \rightharpoonup \Sigma^\omega$ such that $\delta_Y \circ \eta = f \circ \delta_X$. Computable functions are necessarily continuous. An element $x \in X$ is *computable* if there is a machine-computable function η with no inputs which outputs a δ-name of x.

Representations which can be effectively converted between each other induce the same computable operations, and are considered equivalent. A *computable type* $\mathcal{X}$ is an equivalence class of pairs $(X, [\delta])$ under effective equivalence of representatives. We emphasise that a computable type is more than just a set of objects; it also specifies a way of *describing* objects of the sets. Different representations of the same underlying set provide different kinds of information, from which different properties can be deduced.

Computable types form a Cartesian closed category, so has a *terminal* (singleton) type $\mathfrak{I}$, canonical product types $\mathcal{X}_1 \times \mathcal{X}_2$ with computable projections $\pi_{1,2}$, and canonical function types denoted $\mathcal{X} \to \mathcal{Y}$ or $\mathcal{Y}^\mathcal{X}$, with a computable evaluation $\mathcal{Y}^\mathcal{X} \times \mathcal{X} \to \mathcal{Y}$ and equivalence $(\mathcal{X} \times \mathcal{Y}) \to \mathcal{Z} \equiv \mathcal{X} \to (\mathcal{Y} \to \mathcal{Z})$. The category of computable types also supports countable products $\prod_{n=1}^\infty \mathcal{X}_n$.

There are natural types $\mathbb{N}$ and $\mathbb{R}$ of natural and real numbers, respectively, supporting the usual arithmetical operations, but equality on $\mathbb{R}$ is undecidable. The sequence type $\mathcal{X}^\omega = \prod_{n=0}^\infty \mathcal{X}$ is equivalent to the type $\mathbb{N} \to \mathcal{X}$.

There is natural *Sierpinski* type $\mathbb{S}$ with values $\{\top, \uparrow\}$, where the element $\uparrow$ of S denotes "unknown". A *verifyable* predicate on $\mathcal{X}$ is a computable function $\mathcal{X} \to \mathbb{S}$.

Since a subset of a space X is open if, and only if, its characteristic function is continuous $X \to \{\top, \uparrow\}$, we identify the type of *open* subsets $\mathcal{O}(\mathcal{X})$ with $(\mathcal{X} \to \mathbb{S})$ via the membership predicate $U \ni x$. Thus computable open sets U are those for which the predicates $U \ni x$ are verifyable, The type of *closed* sets $\mathcal{A}(\mathcal{X})$ is similarly identified with $(\mathcal{X} \to \mathbb{S})$ via the predicate $A \not\ni x$. We identify the type of *compact* subsets of $\mathcal{X}$ with a subtype $\mathcal{K}(\mathcal{X})$ of $(\mathcal{O}(\mathcal{X}) \to \mathbb{S})$ via the subset predicate $C \subset U$; note $C \subset (U \cap V) \iff (C \subset U) \wedge (C \subset V)$. Thus computable compact sets C are those for which the predicates $C \subset U$ are verifyable.

Important computable operators on sets are singleton $\mathcal{X} \to \mathcal{K}(\mathcal{X})$ given by $x \mapsto \{x\}$, intersection $\mathcal{A}(\mathcal{X}) \times \mathcal{K}(\mathcal{X}) \to \mathcal{K}(\mathcal{X})$, and set-image $(\mathcal{X} \to \mathcal{K}(\mathcal{Y})) \times \mathcal{K}(\mathcal{X}) \to \mathcal{K}(\mathcal{Y})$ given by $F, C \mapsto F(C)$.

A type is *effectively Hausdorff* if inequality is verifyable, from which it follows that every compact set is computably closed, and is *effectively compact* if every closed set is computably compact. If $F : \mathcal{X} \to \mathcal{K}(\mathcal{Y})$, then $\mathrm{graph}(F)$ is computable in $\mathcal{K}(\mathcal{X} \times \mathcal{Y})$ if $\mathcal{X}$ is effectively compact, and in $\mathcal{A}(\mathcal{X} \times \mathcal{Y})$ if $\mathcal{Y}$ is effectively Hausdorff. The real number type $\mathcal{R}$ and all subtypes and products are effectively Hausdorff, and any product of closed bounded intervals with computable endpoints is effectively compact.

A type $\mathcal{X}$ has the *effective Tychonoff property* if the natural inclusion $(\mathcal{K}(\mathcal{X}))^\omega \hookrightarrow \mathcal{K}(\mathcal{X}^\omega)$ is computable; it is known that all *countably-based* spaces have this property. If $\mathcal{X}$ has the effective Tychonoff property, then given a map $G : \mathcal{X}^* \to \mathcal{K}(\mathcal{X})$, the countable skew product $\{\vec{x} \in \mathcal{X}^\omega \mid \forall n \in \mathbb{N},\ x_{n+1} \in G(\vec{x}|_n)\}$ is computable in $\mathcal{K}(\mathcal{X}^\omega)$.

If $p : \mathcal{K}(\mathcal{X}) \to \mathbb{S}$ is a verifyable property and $p(C) = \top$, then there is an open set U such that $p(C')$ holds for *any* compact set $C' \subset U$. Further, given a sequence of compact sets C_n such that $C_{n+1} \subset C_n$ for all n and $\bigcap_{n=0}^\infty C_n = C$, there exists m such that $p(C_n)$ holds for any $n \geq m$.

We note that although any subset S of a discrete space Q is decidable, the *information* on S given by a $\mathcal{K}(Q)$-name merely suffices to compute a sequence of exact sets $S_n \subset Q$ such that $S_{n+1} \subset S_n$ for all n, and $S_n = S$ for n sufficiently large. However, for a given n, we do *not* have

enough information to decide whether S_n actually *equals* S. We write $\mathcal{P}(Q)$ for the type of *exact* subsets of Q, for which membership is decidable.

D. Computable systems theory

Combining the theory of Sections II-A and II-C allows us to develop effective computability for systems theory. Given types $\mathcal{U}$, $\mathcal{X}$, $\mathcal{Y}$, we can describe a state space system (F, H, W) by taking

$$F \in \mathcal{X} \times \mathcal{U} \to \mathcal{K}(\mathcal{X}),\ H \in \mathcal{X} \to \mathcal{K}(\mathcal{Y}),\ W \in \mathcal{K}(\mathcal{X}).$$

The use of the compact set types is appropriate, since we wish to prove properties about *all* trajectories. Similarly, the type of input-output maps M is naturally taken to be $\mathcal{U}^\omega \to \mathcal{K}(\mathcal{Y}^\omega)$, and the type of output sequences is $\mathcal{K}(\mathcal{Y}^\omega)$.

The statement "the target property is verifyable from the output sequences" is then interpreted as computability of the operator $\mathcal{K}(\mathcal{Y}^\omega) \times \mathcal{O}(\mathcal{Y}) \to \mathcal{S}$ define by the predicate $\forall \vec{x} \in O,\ \exists n \in N,\ \vec{x}_n \in T$.

III. Input-output maps

In this section, we show that the input-output map is sufficient to prove target properties, but the output set is insufficient to prove safety.

Lemma 1:

1) The input-output map $M_S : \mathcal{U}^\omega \to \mathcal{K}(\mathcal{Y}^\omega)$ is computable from a state-space system $S = (F, H, W)$ with $F \in (\mathcal{X} \times \mathcal{U} \to \mathcal{K}(\mathcal{X}))$, $H \in (\mathcal{X} \to \mathcal{K}(\mathcal{Y}))$ and $W \in \mathcal{K}(\mathcal{X})$.
2) Suppose $\mathcal{Y}_1 = \mathcal{U}_2$ and $\mathcal{Y}_2 = \mathcal{U}_1$. Then the parallel composition $O_{12} = M_1 || M_2 \in \mathcal{K}((\mathcal{Y}_1 \times \mathcal{Y}_2)^\omega)$ of input-output maps $M_1 \in \mathcal{U}_1^\omega \to \mathcal{K}(\mathcal{Y}_1^\omega)$ and $M_2 \in \mathcal{U}_2^\omega \to \mathcal{K}(\mathcal{Y}_2^\omega)$ is computable.
3) Let $T \in \mathcal{O}(\mathcal{Y})$ and $O \in \mathcal{K}(\mathcal{Y}^\omega)$. Then the target predicate $\forall \vec{y} \in O,\ \exists n,\ y_n \in T$ is verifyable from names of T and O.

Proof:

1) For given $\vec{u}$, the set $\{(x_0, y_0) \mid x_0 \in W \wedge y_0 \in H(x_0)\}$ is computable in $\mathcal{K}(\mathcal{X} \times \mathcal{Y})$. Given (x_n, y_n), the set $(x_{n+1}, y_{n+1}) \mid x_{n+1} \in F(x_n, y_) \wedge y_{n+1} \in H(x_{n+1})$ is computable in $\mathcal{K}(\mathcal{X} \times \mathcal{Y})$. Hence the set of sequences $(\vec{x}, \vec{y}) \in \mathcal{X}^\omega \times \mathcal{Y}^\omega \equiv (\mathcal{X} \times \mathcal{Y})^\omega$ is computable in $\mathcal{K}((\mathcal{X} \times \mathcal{Y})^\omega)$. The result follows by projecting onto $\mathcal{Y}^\omega$.
2) Define $O_i = \bigcup_{\vec{u}_i \in \mathcal{U}_i^\omega} \{\vec{u}_i\} \times M_i(\vec{u}_i)$ for $i = 1, 2$. Since each $\mathcal{U}_i$ is effectively compact, so is $\mathcal{U}_i^\omega$, and O_i is computable in $\mathcal{K}(\mathcal{U}_i^\omega \times \mathcal{Y}_i^\omega) \equiv \mathcal{K}((\mathcal{Y}_1 \times \mathcal{Y}_2)^\omega)$. Since each $\mathcal{Y}_i$ is effectively Hausdorff, so is $(\mathcal{Y}_1 \times \mathcal{Y}_2)^\omega$, and hence O_i are also computable in $\mathcal{A}(((\mathcal{Y}_1 \times \mathcal{Y}_2)^\omega)$. The result follows since $O_{12} = O_1 \cap O_2$.
3) The projections $\pi_n : \mathcal{Y}^\omega \to \mathcal{Y}$ defined by $\pi_n(\vec{y}) = y_n$ are computable functions. Since preimage and countable union of open sets are computable, $\bigcup_{n=0}^\infty \pi_n^{-1}(T)$ is computable in $\mathcal{O}(\mathcal{Y}^\omega)$ from a name of T. Hence $O \subset \bigcup_{n=0}^\infty \pi_n^{-1}(T)$ is verifyable from a name of O. ■

Combining the above results, we immediately obtain:

Theorem 2: Let $\mathcal{U}_1 = \mathcal{Y}_2$ and $\mathcal{U}_2 = \mathcal{Y}_1$. Then the operator $\left(\mathcal{U}_1^\omega \to \mathcal{K}(\mathcal{Y}_1^\omega)\right) \times \left(\mathcal{U}_2^\omega \to \mathcal{K}(\mathcal{Y}_2^\omega)\right) \times (\mathcal{O}(\mathcal{Y}_1 \times \mathcal{Y}_2)) \to \mathcal{S}$ defined by $(M_1, M_2, T_{12}) \mapsto M_1 || M_2 \models \forall\Diamond T_{12}$ is computable.

In other words, the information provided by a name of input-output map M_1 of a system S_1 is sufficient to prove target properties whenever S_1 forms a closed-loop system in parallel composition with another system S_2 described by its input-output map M_2.

Theorem 3: The function $\mathcal{K}(\mathcal{Y}^\omega) \times \mathcal{O}(\mathcal{Y}) \to \mathcal{S}$ given by $(O, S) \mapsto O \models \forall\Box S$ is not computable.

In other words, the information provided by an output generator is insufficient to prove safety.

Proof: Let X be the set $\{\pm 2^{-n} \mid n \in \mathbb{N}\} \cup \{0\}$. Define initial point $w = -1$ and output map $h : X \to \{0, 1\}$ by $h(1) = 1$ and $h(x) = 0$ for $x \neq 1$.

We define systems S_∞ with maps $f_\infty : X \to X$ given by

$$f_\infty(x) = \begin{cases} x/2 \text{ if } x < 0; \\ x \text{ if } x = 0, 1; \\ 2x \text{ if } 0 < x < 1. \end{cases}$$

We define $f_{\pm n}$ as the modification of f_∞ with $f_{\pm n}(-2^{-n}) = \pm 2^{-n}$, and let $S_{\pm n} = (f_{\pm n}, h, w)$.

Then each S_{-n} is robustly safe, and has a unique output sequence $O_{-n} = \{0^\omega\}$. Each S_{+n} is unsafe, and has a unique output sequence $O_{+n} = \{0^{2n+1}1^\omega\}$. The system S_∞ is safe, with unique output sequence $O_\infty = \{0^\omega\}$. However, arbitrarily small perturbations are unsafe, with possible output sequences $\{0^m 1^\omega\}$ for sufficiently large m.

Since arbitrarily small perturbations of S_∞ are unsafe, safety of S_∞ cannot be proved from its output set $O_\infty = \{0^\omega\}$. Since $O_{-n} = O_\infty$ for all $n \in \mathbb{N}$, safety of S_{-n} cannot be proved from the output sequence O_{-n}, even though safety is robust. ■

Note that it is actually the case that for any output set O, it is impossible to prove any nontrivial safety property given a name of O as an element of $\mathcal{K}(\mathcal{Y}^\omega)$. An alternative proof of Theorem 3 would be to simply observe that *any* safe output sequence is the limit of unsafe output sequences.

Intuitively, the main obstruction to using the set of output sequences to prove safety is that the standard representation of sequences only gives information about *finite* time intervals from a finite piece of information, whereas to prove safety we need to know about *infinite* time intervals.

IV. Discretisation

A *discretisation* of a set X is a finite cover $\widehat{X}$ of X. We identify elements $\hat{x} \in \widehat{X}$ with subsets of X. Given discretisations $\widehat{X}$ of X and $\widehat{Y}$ of Y, and a function $F : X \rightrightarrows Y$, we define the discretisation $\widehat{F} : \widehat{X} \rightrightarrows \widehat{Y}$ by

$$\begin{aligned} \widehat{F}(\hat{x}) &= \{\hat{y} \in \widehat{Y} \mid \exists\, x \in \hat{x},\ \exists\, y \in \hat{y},\ y \in F(x)\} \\ &= \{\hat{y} \mid F(\hat{x}) \cap \hat{y} \neq \emptyset\}. \end{aligned}$$

Given a discrete function $\widehat{F} : \widehat{X} \rightrightarrows \widehat{Y}$, we define the undiscretisation $\widetilde{F} : X \rightrightarrows Y$ by

$$\begin{aligned} \widetilde{F}(x) &= \{y \in Y \mid \exists\, \hat{x} \ni x,\ \exists\, \hat{y} \ni y,\ \hat{y} \in \widehat{F}(\hat{x})\} \\ &= \bigcup \{\hat{y} \in F(\hat{x}) \mid \hat{x} \ni x\}. \end{aligned}$$

The undiscretisation of the discretisation is then

$$\widetilde{F}(x) = \{y \in Y \mid \exists\, \hat{x} \ni x,\ \hat{y} \ni y,\ F(\hat{x}) \cap \hat{y} \neq \emptyset\}.$$

We henceforth consider discretisations by computable closed compact sets, so $\widehat{X} \subset \mathcal{AK}(\mathcal{X})$. Since discretisations are finite, this means that $\mathcal{X}$ must be compact.

Lemma 4: Let $\widehat{X}$ and $\widehat{Y}$ be discretisations of, respectively, $\mathcal{X}$ and $\mathcal{Y}$ by computable closed compact sets. Then given $F : \mathcal{X} \to \mathcal{K}(\mathcal{Y})$, the discretisation $\widehat{F}$ is computable $\widehat{X} \to \mathcal{K}(\widehat{Y})$.

Proof: For given $\hat{x} \in \widehat{X}$, since $\hat{x}$ is a computable compact set, $F(\hat{x})$ is computable in $\mathcal{K}(\mathcal{Y})$, and since $\hat{y}$ is a computable closed set, the predicate $F(\hat{x}) \cap \hat{y} = \emptyset$ is verifyable. Hence $\widehat{F}(\hat{x})$ is computable in $\mathcal{A}(\widehat{Y}) \equiv \mathcal{K}(\widehat{Y})$. ∎

Lemma 5: Let $\widehat{X}$ and $\widehat{Y}$ be discretisations of, respectively, $\mathcal{X}$ and $\mathcal{Y}$ by computable closed compact sets. Then given $\widehat{F} : \widehat{X} \to \mathcal{K}(\widehat{Y})$, the undiscretisation $\widetilde{F}$ is computable $\mathcal{X} \to \mathcal{K}(\mathcal{Y})$.

Proof: For given $x \in X$, the predicate $x \notin \hat{x}$ is verifyable since $\hat{x}$ is a computable closed set. Hence $\{\hat{x} \in \widehat{X} \mid x \in \hat{x}\}$ is a computable as a closed set, and hence as a compact set. Then $\widehat{F}(\{\hat{x} \mid x \in \hat{x}\})$ is computable in $\mathcal{K}(\widehat{Y})$. Since each $\hat{x}$ is a computable compact set, $\bigcup\{\hat{y} \in \widehat{F}(\{\hat{x} \mid x \in \hat{x}\})\}$ is computable in $\mathcal{K}(\mathcal{Y})$. ∎

We now show that if discretisations of $\mathcal{X}$ and $\mathcal{Y}$ can be made arbitrarily fine, then the over-approximation of discreting and undiscretising $F : X \to \mathcal{K}(Y)$ can be made arbitrarily small. A cover $\mathcal{R}$ of X is said to *refine* a cover $\mathcal{S}$ if $\forall R \in \mathcal{R},\ \exists S \in \mathcal{S},\ R \subset S$.

Definition 6 (Effectively discretisable type): A type $\mathcal{X}$ is *effectively discretisable* if given any countable open cover $\mathcal{U}$ of X, it is possible to construct a discretisation $\widehat{X}$ of X by computable closed compact sets such that $\widehat{X}$ refines $\mathcal{U}$.

Clearly, any compact subset of Euclidean space is effectively discretisable, since we can simply take covers $\widehat{X}_k$ to consist of boxes of width 2^{-k}.

Lemma 7: Let $\mathcal{X}$ and $\mathcal{Y}$ be effectively discretisable compact Hausdorff spaces, let $F : \mathcal{X} \to \mathcal{K}(\mathcal{Y})$, and let W be a neighbourhood of $\mathrm{graph}(F)$. Then there exists open covers $\mathcal{U}$ of $\mathcal{X}$ and $\mathcal{V}$ of $\mathcal{Y}$ such that for any discretisation $\widehat{X}$ refining $\mathcal{U}$ and $\widehat{Y}$ refining $\mathcal{V}$, the corresponding widened function $\widetilde{F} : \mathcal{X} \to \mathcal{K}(\mathcal{Y})$ has $\mathrm{graph}(\widetilde{F}) \subset W$.

Proof: (Sketch) We can first construct discretisation of $\mathcal{X}$ such that for all x, $\{x\} \times F(\hat{x}) \subset W$ for all $x \in X$. Hence W is a neighbourhood of the graph of $x \mapsto \bigcup\{F(\hat{x}) \mid \hat{x} \ni x\}$. We can then construct a discretisation of $\mathcal{Y}$ such that $(x, y) \in W$ whenever $x \in \hat{x}$, $y \in \hat{y}$ and $F(\hat{x}) \cap \hat{y} \neq \emptyset$. ∎

Given input, state and output spaces U, X and Y with discretisations $\widehat{U}$, $\widehat{X}$ and $\widehat{Y}$, and an input-output system $S = (F, H, W)$ over U, X, Y, we can construct a discretised system $\widehat{S} = (\widehat{F}, \widehat{H}, \widehat{W})$ with $\widehat{F} : \widehat{X} \times \widehat{U} \to \mathcal{K}(\widehat{X})$, $\widehat{H} : \widehat{X} \to \mathcal{K}(\widehat{Y})$ and $\widehat{W} : \mathcal{K}(\widehat{W})$. Ideally, we would like the $\widehat{F}$, $\widehat{H}$ and $\widehat{W}$ to have *exact* values, rather than just compact (over-approximating) values. We say a *discrete input-output automaton* is an input-output system (F, H, W) over *discrete* sets (U, X, Y) such that $F : X \times U \to \mathcal{P}(X)$, $H : X \to \mathcal{P}(Y)$ and $W \in \mathcal{P}(X)$.

The main result of this section is to show that given a sequence of successively finer discretisations of U, X and Y, and an input-output system S, we can construct a sequence of discrete input-output automata whose undiscretisations are arbitrarily close to the original system.

Theorem 8: Suppose the spaces U, X, Y are effectively discretisable. Then there exist a sequence of input-output automata $\widehat{A}_n$ on $(\widehat{U}_n, \widehat{X}_n, \widehat{Y}_n)$ which is effectively computable from S such that the undiscretised automata $\widetilde{A}_n = (\widetilde{F}_n, \widetilde{H}_n, \widetilde{W}_n)$ converge to (F, H, W) as $n \to \infty$.

Proof: Without loss of generality, we can assume that the discretisations $\widehat{U}_n, \widehat{X}_n, \widehat{Y}_n$ are successive refinements. By Lemma 4, we can compute the automata A_n with $\widehat{F}_n$, $\widehat{H}_n$ and $\widehat{W}_n$ taking compact values. However, we can terminate the computations of $\widehat{F}_n$, $\widehat{H}_n$ and $\widehat{W}_n$ after m steps to obtain a discrete automata $A_{n,m}$ with $\widehat{F}_{n,m}$, $\widehat{H}_{n,m}$ and $\widehat{W}_{n,m}$ having *exact* values, Since the discretisations of U, X, Y are successive refinements, we can ensure that $\widehat{F}_{m,m}$ refines $\widehat{F}_{n,m}$ for all $n \leq m$. By Lemma 7, by taking m sufficiently large, the undiscretisations of $\widetilde{F}_{m,m}$, $\widetilde{H}_{m,m}$ and $\widetilde{W}_{m,m}$ systems converge to (F, H, W). ∎

Note that the sequence $\widehat{A}_n$ is an *extrinsic* property of S; it depends on the names of (F, H, W) used to construct the discretisations.

The information provided by the sequence of discretised automata $\widehat{A}_n$ gives rise to a sequence of systems converging to the original, the sequence $\widehat{A}_n$ is sufficient to prove *any* provable property of an input-ouput system:

Corollary 9: Let $S = (F, H, W)$ be an input-output system, and $\widehat{A}_n = (\widehat{F}_n, \widehat{H}_n, \widehat{W}_n)$ a sequence of input-output automata given by Theorem 8. Then any verifyable property of S can be proved from some element of $\widehat{A}_n$.

V. The regular expression representation

We have seen that we can discretise an input-output system by constructing a sequence of discrete input-output automata. However, these automata still have the form of a state-space representation. In order to convert to a state-space free representation, we use regular expressions.

An input-output automaton $A = (F, H, W)$ over finite sets $\widehat{U}, \widehat{X}, \widehat{Y}$ can be viewed as an automaton with labelled transitions having initial states W and a transition from x to x' labelled (y, u) if $y \in H(x)$ and $x' \in F(x, u)$. The *language* L_A of $A = (F, H, W)$ is the set of all sequences $(y_0, u_0, y_1, u_1, \ldots)$ such that there exists $\vec{x} \in X^\omega$ such that $x_0 \in W$, $x_{n+1} \in F(x_n, u_n)$ and $y_n \in H(x_n)$, i.e. $\vec{y} \in M_A(\vec{u})$. Note that we can represent the language either as a subset of $(Y \times U)^\omega$ or of $(Y \cup U)^\omega$. In our formalism, the language of an input-output automaton is that of a nondeterministic Büchi automaton for which all states are accepting.

The language of an input-output automaton is a regular ω-language. It is easy to show that the language must also be a closed (equivalently, compact) subset of $(Y \times U)^\omega$. The complement is therefore open, so there is a subset B of $(Y \times U)^*$ such that $L_A = \{\vec{z} \in (Y \times U)^\omega \mid \forall k \in \mathbb{N},\ z|_k \notin B\}$.

In other words the set B contains disallowed prefixes of elements of L_A. Further, the language B can be generated by adding a fail state z and transitions (y, u) from x to z if $y \notin H(x)$.

Although the ω-language of an input-ouput automaton can be computed as a regular ω-expression, it is convenient to use a a modified semantics for regular expressions which succinctly generates only closed ω-languages. To do this, we do not use the empty string ε or repetition * We introduce an operator $^+$, which means "zero or more repetitions, possibly infinitely many", and is defined by $R^+S \equiv R^*S|R^\omega$. For example, the complete regular expression a^+bc^ω expands to $a^*bc^\omega|a^\omega$.

We now consider $\widehat{U}$, $\widehat{Y}$ which are discretisations of U and Y by closed, compact sets, and a regular expression $\widehat{R}$ over $(\widehat{Y} \times \widehat{U})$. An element of $L_{\widehat{R}}$ is a sequence of the form $(\hat{y}_0, \hat{u}_0, \hat{y}_1, \hat{u}_1, \ldots)$. We define the input-output map $\mathcal{M}_{\widehat{R}}(\vec{u}) = \{\vec{y} \in Y^\omega \mid \exists(\hat{y}_0, \hat{u}_0, \hat{y}_1, \hat{u}_1, \ldots) \in L_{\widehat{R}},\ \forall k \in \mathbb{N}, u_k \in \hat{u}_k \wedge y_k \in \hat{y}_k\}$.

We can now define the *regular expression representation* of the type of input-ouput maps $\mathcal{U}^\omega \to \mathcal{K}(\mathcal{Y}^\omega)$, and by extension to the type of input-output systems.

Definition 10: The *regular expression representation* of input-output maps encodes M by a sequence of tuples $(\widehat{U}_i, \widehat{Y}_i, \widehat{R}_i)$ such that $\widehat{U}_i, \widehat{Y}_i$ are discretisations of U, Y by closed, compact sets, and $\widehat{R}_i$ is a regular expression over $\widehat{Y}_i \times \widehat{U}_i$ such that for all i, $\mathcal{M}_{\widehat{R}_{i+1}}$ is a refinement of $\widehat{R}_i$, and $M = \bigcap_{i=0}^{\infty} \mathcal{M}_{\widehat{R}_i}$. The *regular expression representation* of state-space systems encodes $S = (F, H, W)$ by a sequence of regular expressions describing the languages of the automata of Theorem 8.

Combining the previous results, we obtain the main result of this article, which is that the regular expression representation of the input-ouput map of an input-ouput system provides sufficient information to prove any verifyable property of the system.

Theorem 11: Let $S = (F, H, W)$ be a non-blocking input-output system such that $F : X \times U \to \mathcal{K}(X)$, $H : X \to \mathcal{K}(Y)$ and $W \in \mathcal{K}(X)$. Then any property of S verifyable from (F, H, W) is also verifyable from the regular expression representation of the input-output map of S generated by a convergent sequence of discretised input-output automata generated by Theorem 8.

Proof: Immediate from Corollary 9, the construction of an automaton recoginising a language, and decidability of language equivalence for discrete automata. ∎

We therefore have verifyability of robust temporal logic properties of closed-loop systems:

Corollary 12: The target safety, recurrence and stabilisation properties for the parallel composition of S_1 with another system S_2 are verifyable from the regular expression representations of S_1 and S_2.

VI. Summary and Conclusions

In this paper, we have considered the problem of computing an input-output description of a discrete-time state-space system which contains sufficient information to prove temporal logic properties of the system with a closed-loop controller. We showed that the standard representation of the input-output map contains insufficient information, but that a representation based on regular expressions obtained by discretising the system is sufficient; indeed, contains sufficient information to verify *any* property which is verifyable from the original state-space description. In future work, we plan to implement the computation of the representation for low-dimensional state-space systems.

The main practical implication of this result is that regular expressions provide a convenient and powerful state-space-independent description of subsystems of a complex control system. Rather than compute an infinite *sequence* of regular expressions, one would compute a regular expression which captures a "sufficiently robust" discrete abstraction of the dynamics. This regular expression could then be used as a concrete format to be used for further analysis, such as in a discrete model-checking tool. Of course, one need not compute the regular expression directly, but instead store a finite automaton describing the dynamics. The usefulness of this approach in a practical situation is likely to be determined by the availability of consise, accurate discrete descriptions of the original system, and it is likely that sophisticated techniques for computing and simplifying discretisations will be needed to obtain good results.

These results can be largely extended to systems on spaces which are *effectively locally-compact* rather than compact. They can also be extended to continuous- and hybrid-time systems; the only real difficulty is in handling time correctly.

References

[1] D. Bertsekas, "Convergence of discretization procedures in dynamic programming," *IEEE Trans. Automatic Control*, vol. 20, no. 3, pp. 415–419, 1975.

[2] P. Collins, "Computable analysis with applications to dynamic systems," Centrum voor Wiskunde en Informatica, Tech. Rep., 2010, cWI Research Report MAC-1002.

[3] P. Collins and I. S. Zapreev, "Computable semantics for CTL* on discrete-time and continuous-space dynamic systems," *Int. J. Found. Comput. Sci.*, vol. 22, no. 4, pp. 801–821, 2011.

[4] M. Dellnitz, G. Froyland, and O. Junge, "The algorithms behind GAIO-set oriented numerical methods for dynamical systems," in *Ergodic theory, analysis, and efficient simulation of dynamical systems*. Springer, 2001, pp. 145–174.

[5] M. Fränzle, "Analysis of hybrid systems: An ounce of realism can save an infinity of states," in *Proceedings of Computer Science Logic*, ser. LNCS 1683, J. Flum and M. Rodríguez-Artalejo, Eds. Springer, 1999, pp. 126–140.

[6] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems. Specification*. New York: Springer-Verlag, 1992.

[7] D. Perrin and J.-E. Pin, *Infinite Words: Automata, Semigroups, Logic and Games*. Elsevier, 2004.

[8] J. W. Polderman and J. C. Willems, *Introduction to mathematical systems theory: A behavioral approach*, ser. Texts in Applied Mathematics. New York: Springer-Verlag, 1998, vol. 26.

[9] M. Sipser, *Introduction to the Theory of Computation*. Thomson, 1996.

[10] K. Weihrauch, *Computable analysis*, ser. Texts in Theoretical Computer Science. An EATCS Series. Berlin: Springer-Verlag, 2000, an introduction.