INT. J. CONTROL, 1996, VOL. 65, NO. 2, 249–276

On the corrective control of sequential machines

JACOB HAMMER†

The paper deals with the design of controllers that correct faulty behaviour of sequential machines caused by corrupted inputs. Alternatively, the results can be interpreted as the design of controllers that steer a sequential machine from an unknown initial condition to a prescribed steady-state course. In these terms, the paper characterizes the uncertainties about the initial condition under which the prescribed steady-state course can be achieved. The paper is written within the input/output framework of nonlinear control, and is motivated in part by potential applications to molecular biology.

1. Introduction

In essence, a sequential machine is an entity that operates in a stepwise fashion, progressing from one step to the next in response to a stimulus. The stimulus could be external (e.g., a change in an input variable), or internal (e.g., a change in an internal variable, or a clock tick). Sequential machines are widely used as models for computing machinery, manufacturing equipment, traffic control, and many other applications. Sequential machines can also be used to model sequences of chemical reactions that progress in a stepwise manner (e.g., chain reactions, catabolic pathways, etc.). An important application in this context is the modelling of (biological) cell function, using the principles of molecular biology. Many of the basic processes that govern the operation of cells have a natural sequential structure. Some examples are: the Krebs cycle, the transcription of DNA into RNA, the translation of RNA into protein, and others. The modelling of various biological phenomena within the general framework of automata and language theory has been documented in the literature for quite some time (Rashevsky 1948, Sugita 1963, von Neumann 1966, Lindenmayer 1968, Kauffman 1969, Rozenberg and Salomaa 1975, IEEE 1974, the references cited in these works, and many others).

The present paper deals with the development of methods for the control of sequential machines. It provides techniques for the design of controllers that correct undesirable behaviour of sequential machines. The basic motivation is to formulate controllers that correct impaired function of biological cells. Control techniques have proven successful in a great variety of engineering, physical and physiological applications, and, in fact, are naturally invoked in various catabolic pathways (e.g. in the synthesis of the amino acids lysine, threonine and isoleucine in mammal cells). When combined with sequential models of cells, control techniques offer the prospect of providing new insight into the regulation of unacceptable behaviour of cells, such as the unrestrained division associated with pre-cancerous or cancerous transformations, or other malfunctions of the genetic system. By sequential models of cells

Received 13 February 1995. Revised 12 July 1995.

† Center for Mathematical System Theory, Department of Electrical Engineering, University of Florida, P.O. Box 116130, Gainesville, FL 32611-6130, U.S.A.

we mean empirical input/output models of cell function. These may include discrete-event approximations of continuous models, which have been created to facilitate simulation or control via a digital computer.

The discussion within the paper is independent of any specific application. We simply consider the control of a certain class of generalized sequential machines. Nonetheless, we often allude to potential applications to gain insight and motivation.

The sequential machines that we consider are defined to be suitable for modelling chemical or biochemical reaction systems within a medium. Here, each reactant molecule (or molecular complex) in the medium is regarded as a variable, called a *word*. The sequential machine implements the rules of chemistry that govern the reaction steps, changing the molecular population within the medium as the reaction evolves. Molecules that are externally injected into the medium are regarded as *input words*, whereas molecules (or complexes) present within the medium at the end of a reaction step are regarded as *output words*. The number of molecules within each category varies with reaction, step and circumstance, and may, of course, be quite large. However, in many systems of interest in molecular biology, the number of significant kinds of molecules seems manageable. For instance, the functioning of an E-coli bacterium probably involves no more than a few thousand different kinds of significant molecules; and the operation of a mammal cell probably involves no more than a few hundred thousands of different kinds of significant molecules (Alberts *et al.* 1989). The computational complexity involved with the modelling and control of sequential machines with thousands or even hundreds of thousands of distinct variables is not likely to be much higher than that required, for instance, for detailed numerical studies of fluid dynamics.

We shall regard a sequential machine as a nonlinear system, transforming input sequences (or words) into output sequences (or words). We concentrate on sequential machines which, due to some 'harmful' or 'defective' input words, have embarked on an unacceptable course. Our principal objective is to study the existence and the design of controllers which, once combined with the system, stear it back to an acceptable course. The controllers are combined with the system while the system is operating along an undesirable course, and aim to correct the behaviour; Thereupon, we refer to this problem as *corrective control*. We note that the problem of corrective control is equivalent to the problem of designing a controller that drives the system from an unknown initial condition to a prescribed steady-state course.

In very broad (and somewhat inaccurate) terms, necessary and sufficient conditions for the existence of a corrective controller are determined by an implication of the principle of causality. Qualitatively, and quite obviously, a corrective controller exists if and only if it is possible to detect a 'harmful' deviation in the behaviour of the system before it is too late to correct for it. Of course, this fundamental principle needs to be refined and presented in a computable form. In addition, effective techniques for the design of corrective controllers, whenever such controllers exist, need to be developed. These topics are discussed within the body of the paper.

A critical factor in our discussion is the fact that the information available about the system that needs to be controlled is incomplete. The existence of a corrective controller, as well as its design, need to be determined based on this incomplete information. For cases where corrective control cannot be achieved due to the lack of sufficient data about the system, we consider the question of characterizing the 'minimal' additional data that need to be provided in order to facilitate the existence of a corrective controller. As it turns out, in general, there is no unique answer to this question; the available data can be refined in various 'minimal' ways to facilitate corrective control. The class of all data sets that are sufficient for corrective control forms a partially ordered set (a poset) which is usually not a lattice. In general, there is no data set that constitutes an absolute minimum among all data sets that are sufficient for corrective control. Still, local minima exist within the poset, and techniques for their computation are discussed in §5. Once the local minima are known, an economical data set most convenient for measurement can be selected.

Although the present paper deals with objects within the general theory of automata and languages, the point of view adopted here is mostly within traditional control theory; we regard our systems as maps that transform sequences of (discrete) inputs into sequences of (discrete) outputs. Still, a background in the basic theory of automata, languages and discrete-event systems is helpful in the present context. It can be gained from Ginsburg (1962, 1966), Eilenberg (1974), Hoare (1976), Milner (1980), Arnold and Nivat (1980), and many other excellent sources.

The current widely spread interest in discrete-event systems within the research literature on systems and control was sparked by a series of papers written by W. M. Wonham and his coworkers during the early to mid-eighties (Ramadge and Wonham 1987, Vaz and Wonham 1986, Lin and Wonham 1988, and others). In these papers, basic notions of control theory, like controllability and observability, were extended into the theories of automata and languages, and were applied to the design of supervisors for discrete-event systems. These efforts evolved into a mathematical framework based on the theory of formal languages, with ω-languages playing an important role (Thistle and Wonham 1988, Kumar *et al.* 1992). The objective in this approach is to design a supervisor which, when combined with the system, produces a prescribed formal language. The existence of a supervisor depends on whether or not the prescribed language is contained within a certain maximal language, which is determined by the underlying system and the nature of the observations. These topics were further investigated by Cieslak *et al.* (1988), Cho and Marcus (1989), Ozveren and Willsky (1990), and the references cited in these papers.

For systems with a large number of states, the computational burden required for the design of supervisors is rather substantial. Computational aspects of supervisor design are discussed by Ramadge (1989), Tsitsiklis (1989), and others. More recently, the calculus of predicates was found to provide an alternative framework for the study of supervisory control (Kumar *et al.* 1993). Problems in supervisory control were also studied within a Petri net framework (Holloway and Krogh 1990, Sreenivas and Krogh 1992). Lastly, various issues related to the problem of stabilization of discrete-event systems were considered by Ozveren *et al.* (1991). An important difference between these results and the corrective control problem discussed in the present paper is that here the system that needs to be controlled is uncertain in the sense that its history is only partly known. In other words, in the present case the controlled system needs to be driven to a desired steady-state response from an initial condition that is not precisely known. The main emphasis is on the characterization of the maximal uncertainty about the initial condition that still permits the achievement of the objective.

To address the problem of corrective control, we adopt the classical input/output point of view of linear and nonlinear control theory. The basic formalism is introduced in §2. The remaining sections of the paper deal with the existence and the design of corrective controllers. Necessary and sufficient conditions for the existence of corrective controllers are derived in §3; §4 deals with the construction of corrective controllers and with a characterization of their capabilities; and §5 provides a characterization of the minimal information that is needed about the system's history in order for a corrective controller to exist.

Finally, to relate the present paper to the state space theory of discrete-event systems, we recall that the problem of corrective control is equivalent to the problem of designing a controller that drives the system from an unknown initial condition to a prescribed steady-state course. This specific problem has not been previously studied in the literature on discrete-event systems; but, in classical terms, the solution to the problem would depend on a complex interaction between the notions of controllability and observability, that would vary with the prescribed steady-state course. In the present paper we show that the problem can be treated in a relatively simple way through the input/output theory of nonlinear control systems. The key notion required for the solution is the classical notion of causality, that can be conveniently handled only within an input/output framework. Furthermore, input/output theory directly yields the structure of all dynamic controllers that achieve the desired control objective. The main result of the paper is a characterization of the uncertainty about the initial condition that still permits the achievement of the prescribed steady-state course. The set of all possible controllers that achieve the desired objective is also provided.

2. Words, sentences and interpreters

Consider a finite non-empty alphabet $A = \{a_1, \ldots, a_n\}$. A word w over the alphabet A is a concatenation $w = b_1 b_2 \ldots b_m$ of a finite number of characters $b_1, \ldots, b_m$ of A. In a word, the leftmost character is the first and the rightmost character is the last. Denote by A^* the set of all words over A. The fact that A is non-empty implies that A^* has infinitely many elements. The alphabet A could be a disjoint union of various alphabets. For instance, in biological applications it may represent the disjoint union of alphabets consisting of the DNA bases, the RNA bases and the protein bases, as well as other important ingredients.

The term *sentence* refers to a finite collection of (not necessarily distinct) words. The empty set $\varnothing$ is also regarded as a sentence. We emphasize that a sentence may include several copies of the same word. The class of all sentences of words over the alphabet A is denoted by S_A. It contains infinitely many elements. The cardinality of a sentence s is equal to the total number of words within the sentence, counting each word according to its multiplicity in s; it is denoted by $\# s$.

When discussing the inclusion of two sentences s_1 and s_2 within each other, the multiplicity of each word in each sentence is taken into account; The relation $s_1 \subset s_2$ indicates that all words of s_1 are also words of s_2; for words appearing in multiple copies, the number of copies in s_1 does not exceed the number of copies in s_2.

The present paper deals with systems that transform sequences of sentences into sequences of sentences. Formally, let $S(S_A)$ be the set of all sequences $s_0, s_1, s_2, \ldots$ of sentences, where $s_i \in S_A$ for all i. The index i is regarded as a step counter; a step may or may not be linked to a specific time duration. A *list* of sentences is any finite or infinite set of indexed sentences $s_i, s_{i+1}, \ldots$ For a sequence $s \in S(S_A)$ and two integers $j \geqslant i \geqslant 0$, we denoted by s_i the ith sentence of the sequence, and by s_i^j the list of sentences $s_i, s_{i+1}, \ldots, s_j$; the infinite list $s_i, s_{i+1}, s_{i+2}, \ldots$ is denoted by s_i^∞. If $j < i$, then s_i^j is the empty set. It is convenient to use the notation $(S(S_A))_i^j$ for the set of all lists $\{s_i, s_{i+1}, \ldots, s_j\}, j \geqslant i$.

In their most basic form, the systems we consider can be described by the following notion. A *primitive interpreter* is a map $\Sigma: D \to S(S_A)$, with a subset $D \subset S(S_A)$ as its domain, and $S(S_A)$ as its codomain (or range). It is a discrete system that accepts sequences of sentences as its input, and generates sequences of sentences as its output.

It is common to distinguish between two kinds of sequences generated by a system: the output sequence, which describes the system's performance; and a monitored sequence, which describes measurable quantities generated by the system that are constantly monitored and can be used to control the system. In this spirit, we define an *interpreter* as a map $\Sigma: D \to S(S_A) \times S(S_A): u \mapsto \Sigma u = (y, \mu)$ that generates the two sequences y and μ, where y is the *output sequence* and μ is the *monitored sequence* of the interpreter. In qualitative terms, y describes the 'product' of the interpreter, whereas μ describes those parts of the product that are being continuously monitored. The monitored sequence contains all the data that are available about the response of Σ. The pair (y, μ) is called the *augmented output sequence* of Σ. We shall use the abbreviation $SS_A := S(S_A) \times S(S_A)$.

For a subset $S \subset D$ and an interpreter $\Sigma: D \to SS_A$, denote by $\Sigma[S]$ the set of all augmented output sequences generated by input sequences from S. As usual, $\operatorname{Im} \Sigma := \Sigma[D]$ is the image of Σ. For an input sequence $u \in D$ and a pair of integers $j \geqslant i \geqslant 0$, we denote by $(\Sigma u)_i^j$ the list of augmented output elements $z_i, z_{i+1}, \ldots, z_j$, where $z := \Sigma u$ is the augmented output sequence.

An interpreter Σ can be regarded as a pair $\Sigma = (\Sigma_o, \Sigma_m)$ of primitive interpreters $\Sigma_o, \Sigma_m: D \to S(S_A)$, where for every input sequence $u \in D$ the output sequence of Σ is given by $y = \Sigma_o u$, and the monitored sequence is $\mu = \Sigma_m u$. The primitive interpreter Σ_o is the *output part* of Σ, and Σ_m is the *monitored part* of Σ.

The term *autonomous interpreter* refers to an interpreter that operates on its own with no external input sequence, i.e. an interpreter $\Sigma: \varnothing \to SS_A$. Autonomous interpreters are of particular interest to us, since they require no human operator to function correctly. Their output sequence is determined by their structure.

Consider, for a moment, the case of molecular biology. Here, a word represents a molecule or a molecular complex, and a sentence represents a collection of molecules, molecular complexes, etc. The interpreter represents the biochemical mechanisms which, at each reaction step, transform one set of molecules and molecular complexes into another. An input sentence represents a collection of molecules externally added to the medium at a reaction step; an output sentence represents a collection of molecules left within the medium after a reaction step. A monitored sentence represents a collection of molecules or molecular complexes whose presence within the medium at a specified step has been ascertained.

Clearly, a molecule or a molecular complex may be present in more than one copy within a collection, and the number of copies is chemically significant. Consequently, when sets of molecules and molecular complexes are combined, disjoint unions need to be used to preserve all duplicate copies.

In formal terms, let $a, b \in S_A$ be two sentences. Denote by $a \dot{\cup} b$ the disjoint union of a and b, i.e. the sentence that consists of the aggregate of all the words of a and of b, with all copies of duplicate words included. Similarly, the disjoint union $s \dot{\cup} u$ of two sequences of sentences $s, u \in S(S_A)$, is again a sequence of sentences $y \in S(S_A)$, given by $y_k := s_k \dot{\cup} u_k, k = 0, 1, 2, \ldots$

The notion of causality plays an important role in our discussion, since, for interpreters that evolve in time, only causal interpreters are possible. An interpreter $\Sigma: D \to SS_A$ is *causal* (respectively, *strictly causal*) if, for every pair of input sequences

$u, v \in D$ and for every integer $k \geqslant 0$, the equality $u_0^k = v_0^k$ implies that $(\Sigma u)_0^k = (\Sigma v)_0^k$ (respectively, $(\Sigma u)_0^{k+1} = (\Sigma v)_0^{k+1}$). These are, of course, the standard definitions.

Let $S \subset S(S_A)$ be a family of sequences. The value set V of S is the smallest set of sentences $V \subset S_A$ that satisfies $s_k \in V$ for all sequences $s \in S$ and all integers $k \geqslant 0$. The value set consists of all sentences that may appear at a step of any sequence belonging to S. For an interpreter $\Sigma = (\Sigma_o, \Sigma_m) : D \to SS_A$, the value set of D is called the *input value set*; the value set of $\operatorname{Im} \Sigma_o$ is the *output value set*; and the value set of $\operatorname{Im} \Sigma_m$ is the *monitored value set*.

A set $D \subset S(S_A)$ is a *uniform domain* if it is entirely determined by its value set V, i.e. if it consists of all sequences $u \in S(S_A)$ satisfying $u_k \in V, k = 0, 1, 2, \ldots$ A uniform domain is said to be induced by its value set V.

Let $\Sigma : D \to SS_A$ be an interpreter with the input value set V_{in}. When the domain D of Σ is uniform, every sentence of V_{in} may appear as an input sentence of Σ at any input step. When the domain D is not uniform, some sentences within V_{in} are permitted as input sentences of Σ only at certain input steps. For example, in biochemical applications a uniform domain simply means that any combination of relevant molecules can be injected into the reaction system at any reaction step, as the case usually is. Of course, some of these molecules may create undesirable reactions; a control algorithm will determine which molecules are to be injected at each step in order to achieve desirable results.

2.1. *Recursive interpreters*

Of particular interest are interpreters that can be characterized through a finite recursive structure, as we now discuss. Let X be a finite non-empty set, let

$$f : S_A \times X \to S_A \times X : (s, x) \mapsto (f_i(s, x), f_x(s, x))$$

and $h_o, h_m : S_A \to S_A$ be functions, and let $(s_0, x_0) \in S_A \times X$ be a pair of elements. The list $(X, f, h_o, h_m, s_0, x_0)$ induces an interpreter $\Sigma : D \to SS_A$ in the following way. For every input sequence $u \in D$, the augmented output sequence $(y, \mu) = \Sigma u$ is obtained by the recursion

$$\left.\begin{aligned} (s_{k+1}, x_{k+1}) &= f[(s_k \dot{\cup} u_k), x_k] \\ y_k &= h_o(s_k \dot{\cup} u_k) \\ \mu_k &= h_m(s_k \dot{\cup} u_k), \quad k = 0, 1, 2, \ldots \end{aligned}\right\} \tag{1}$$

The set X is called the *state set* of the interpreter Σ, and an element of X is called a *state*. The pair (s_k, x_k) is called the *status* of Σ at the step k, and (s_0, x_0) is the initial status. Note the distinction between 'state' and 'status'. The output sentence y_k represents the significant products of the interpreter at the step k. The sentence μ_k represents the products whose presence has been ascertained at step k.

We refer to (1) as a recursive representation of the interpreter Σ; the function f is called a recursion function of Σ. An interpreter that admits a recursive representation is called a recursive interpreter. Throughout our discussion, a recursive interpreter is always given together with its initial status (s_0, x_0), which is regarded as part of the description of the interpreter. This, however, does not imply that the status of the interpreter is precisely known at a later time when control is initiated. As discussed in the next section, control of the interpreter starts at a step $\kappa \geqslant 0$, and the exact status of the interpreter at that step may not be known.

In a biochemical context, the input sentence u_k represents the molecules that are externally injected into the medium at step k. The sentence s_k represents the results of reaction step $k-1$. The entire set of molecules or molecular complexes present within the medium at the step k consists of s_k and of the molecules u_k injected externally at the step, and is given by the disjoint union $s_k \dot{\cup} u_k$; these determine the results of reaction step k, together, of course, with the state x_k of the interpreter. The sequence $s = \{s_0, s_1, s_2, \ldots\} \in S(S_A)$ is called the *internal sequence* of Σ, and the sequence $s \dot{\cup} u$ is called the *medium sequence* of Σ. The internal value set of Σ is the set $\mathrm{Im} f_i$, which includes all sentences that may appear as elements of the internal sequence s of Σ, excluding possibly the initial one s_0.

Finally, (1) represents a time-invariant system, since the step counter k does not appear as a separate argument in the recursion. In our biochemical context, this simply reflects the (common) assumption that the laws of chemistry do not change with time.

Remark 1: A comment about compartmentalization. In molecular biology, it is known that the effects of certain molecules are restricted to single individual cell organelles, whereas other molecules have a broader influence, affecting entire cells, entire families of cells or the entire cell population. This situation is referred to as compartmentalization; the range of activity of each molecule is described by an appropriate compartment, which varies, of course, from molecule to molecule. Compartmentalization can be accommodated within the present mathematical framework through the use of an addressing scheme. Each word that represents a molecule is preceded by an address prefix code that indicates the compartment within which it is active. The interpreter Σ decodes the address prefix, and restricts the effects of the word to the compartment corresponding to that address. In this way, compartmentalization becomes a modelling issue, and requires no separate attention within the general framework.

We next discuss several qualitative issues that arise from the definition of a recursive interpreter. First, note that the number of words within each one of the sentences $s_k, u_k, k = 0, 1, 2, \ldots$, is unspecified and may vary with k. When each word is regarded as an input or an output variable, this indicates that the number of input or output variables of the interpreter is not fixed, and may vary from step to step. The number of characters may vary from word to word.

The notion of an interpreter is a generalization of the notion of a sequential machine. For instance, a single-input single-output sequential machine is obtained from the definition of an interpreter by imposing the following four restrictions: restrict all words to one character length; fix the cardinalities $\# s_k = 1$ and $\# u_k = 1$ for all k; set $\mu_k = y_k = s_k$ for all k; and take a recursion function f that is independent of s_k. Thus, a (standard) sequential machine is a particular case of an interpreter.

Further, we comment that the definition of a recursive interpreter contains a redundancy in the sense that one of the quantities x_k or s_k can be eliminated. Indeed, the pair (x_k, s_k) can equivalently be regarded either as an internal variable or as a state, thus eliminating the need for two separate terms. Nevertheless, from a practical point of view, it is convenient to distinguish between the state set x_k and the internal set s_k, since they may represent physically distinct entities. For example, the internal set may represent molecules, whereas the state set may represent states of a computing machine that implements a control algorithm within the biochemical system.

As a final comment on the nature of the definition (1), consider the alternative formulation

$$(s_{k+1}, x_{k+1}) = f[(s_k, u_k, x_k], \quad k = 0, 1, 2, \ldots \tag{2}$$

which seems more general at first glance. Yet the use of an address prefix to distinguish between the elements of s_k and those of u_k in the disjoint union $s_k \dot{\cup} u_k$ renders (2) equivalent to (1). For biochemical systems, (1) seems preferable, since the outcome of a reaction step is determined by the molecules present, regardless of whether their origin is s_k or u_k.

2.2. *Control of interpreters*

The basic objective of the present paper is to develop techniques for the control of recursive interpreters. Control is achieved by combining a given interpreter Σ with a controller C. The controller generates an input sequences for Σ, which, in turn, induces a desirable output sequence from Σ. The information available to the controller during its operation consists of the monitored sequences of Σ; in addition to that, the controller may be prompted by an external reference sequence v taken from a domain $D_c \subset S(S_A)$. Specifically, consider a recursive interpreter $\Sigma = (\Sigma_m, \Sigma_o): D \to SS_A$ given by (1), where u is the input sequence and (y, μ) is the augmented output sequences of Σ. The input sequence u of Σ is generated by a controller $C: (\operatorname{Im} \Sigma_m) \times D_c \to D: (\mu, v) \mapsto C(\mu, v)$ according to

$$u = C(\mu, v) \tag{3}$$

where $v \in D_c$ is the external reference sequence of C. (As seen in the next paragraph, the external reference sequence v serves as the input sequence of the composite system created by Σ and C.) The controller C is required to be causal. Also, since the monitored sentence $\mu_k = h_m(s_k \dot{\cup} u_k)$ may depend on u_k, we require C to be strictly causal in its first variable; This makes (1) into an explicit, rather than an implicit, expression for u, thus simplifying various statements. The input sentence u_{k+1} of Σ is then determined by the sentences $\mu_0, \mu_1, \ldots, \mu_k$ and $v_0, \ldots, v_{k+1}$. A controller $C(\mu, v)$ that is strictly causal in μ and causal in v is called a *semistrictly causal controller*.

The combination of Σ with the controller C is denoted by Σ_c. It is an interpreter $\Sigma_c: D_c \to SS_A: v \mapsto \Sigma_c v = (y, \mu)$ given by the equations

$$(y, \mu) = \Sigma u$$
$$u = C(\mu, v)$$

The first equation directly shows that every output sequence of Σ_c is also an output sequence of Σ, and we have

$$\operatorname{Im} \Sigma_c \subset \operatorname{Im} \Sigma \tag{4}$$

Note that the controller C needs to operate in synchronization with Σ (Hoare 1976). At each step k of Σ, the controller C needs to inject the sentence u_k into the medium. This requires coordination between the controller and the interpreter, and is an important implementation issue. Its resolution varies from one application to another, and is outside the scope of the present paper.

A controller C is *autonomous* if it has no external input sequence v. A semistrictly causal autonomous controller is, in fact, a strictly causal map $C:\operatorname{Im}\Sigma_m \to D:\mu \mapsto C(\mu)$. The combination interpreter Σ_c is then an autonomous system, described by

$$(y,\mu) = \Sigma u$$
$$u = C(\mu)$$

Here, C determines the input sentence u_{k+1} of Σ from the monitored sentences $\mu_0, \mu_1, \ldots, \mu_k$ of Σ. An autonomous contoller automatically executes the various dynamical manipulations necessary to achieve desirable performance, and does not require the interference of a human operator. For this reason, it is of major practical interest. It is the subject of the next section.

3. Autonomous corrective control of recursive interpreters

This section deals with the basic aspects of corrective control of interpreters, using autonomous controllers. An important point is the fact that the information provided about the interpreters is incomplete. Let $\Sigma: D \to SS_A$ be a recursive interpreter with the representation (1). Due to a faulty input at an unknown step, the interpreter Σ has embarked on an unacceptable course. Departure from acceptable behaviour has been detected, and an autonomous controller C is combined with Σ at the step $\kappa \geqslant 0$ to provide correction. The controller C acts as a corrective controller.

The input history of the interpreter Σ up to the step κ at which control initiates may not be precisely known. The only information available is that the initial input list $u_0, u_1, \ldots, u_\kappa$ of Σ belongs to a given subset $\mathscr{I}(\kappa) \subset (D)_0^\kappa$ of possible initial input lists, called the *initial input set* of Σ. The initial input set $\mathscr{I}(\kappa)$ contains all the data available at the step κ about the input history of Σ. Clearly, the smaller the set $\mathscr{I}(\kappa)$ is, the more accurate is the information. At the step κ, the interpreter Σ is combined with the (strictly causal) autonomous controller C. The inputs of Σ from the step $\kappa+1$ and on are provided as outputs of the controller C, and are therefore known.

The combination Σ_c of Σ and C sets out on a corrective course, and, assuming that correction is effective, acceptable behaviour will commence at a future step. The specific step at which acceptable behaviour commences is not specified; rather, acceptable behaviour is characterized by specifying a desirable tail for the output sequence. The objective is to devise, if possible, a controller C that steers the interpreter Σ from an unacceptable beginning to an acceptable tail of its output sequence. The notion of tail needs to be made more accurate.

A *tail* of a sequence s_i^∞ is any subsequence s_k^∞, where $k \geqslant i$ is an integer. A *tail set* is a non-empty set of tails of sequences. A tail set is *complete* if it contains all the tails of each one of its elements. A complete tail set always contains infinitely many elements.

Every tail set E is associated with a complete tail set $\bar{E}$, called the *completion* of E, that consists of all elements of E and all their tails. Clearly, when E is itself a complete tail set, it is equal to its completion.

As a simple example of completion, consider a single sequence $s \in S(S_A)$. Then

$$T(s) := \bigcup_{k \geqslant 0} s_k^\infty$$

is a complete tail set, called the complete tail set of the sequence s.

The following proposition is easy to verify.

Proposition 1: *The intersection of two complete tail sets is a complete tail set, and so is the union of two complete tail sets.*

Returning now to corrective control, recall that the interpreter $\Sigma = (\Sigma_o, \Sigma_m)$ operates without a controller up to the step κ, at which it is combined with the autonomous controller C. During operation, the controller has access only to the monitored sequence of Σ, i.e. to the output sequence of Σ_m. Due to strict causality, the first output of the controller C occurs at the step $\kappa+1$, and, being autonomous, $C:(\mathrm{Im}\,\Sigma_m)_\kappa^\infty \to (D)_{\kappa+1}^\infty$. Consider a specific initial input list $u_0, \ldots, u_\kappa$ of Σ, and let $\gamma_{\kappa+1}^\infty$ denote the output list of the controller C when Σ is started from this initial input list. The entire input sequence v of Σ is then given by the concatenation $v = (u_0, \ldots, u_\kappa, \gamma_{\kappa+1}, \gamma_{\kappa+2}, \ldots)$. Clearly, for fixed Σ and C, the sequence $\gamma_{\kappa+1}^\infty$ generated by the controller can depend only on the initial input list $u_0, \ldots, u_\kappa$ of Σ. It will be convenient to use the notation

$$\left.\begin{aligned} C\{u_0^\kappa\} &= \gamma_{\kappa+1}^\infty \\ \Sigma_c\{u_0^\kappa\} &:= \Sigma v \\ \Sigma_{co}\{u_0^\kappa\} &:= \Sigma_o v \\ \Sigma_{cm}\{u_0^\kappa\} &:= \Sigma_m v \end{aligned}\right\} \tag{5}$$

to denote the response of the controller C and the interpreter–controller combination Σ_c for the initial input list $u_0, \ldots, u_\kappa$. We can now formulate our main subject.

The problem of autonomous corrective control: let $\Sigma = (\Sigma_o, \Sigma_m): D \to SS_A$ be a recursive interpreter with an initial input set $\mathscr{J}(\kappa)$, and let T be a complete tail set. Devise, if possible, an autonomous controller $C:(\mathrm{Im}\,\Sigma_m)_\kappa^\infty \to (D)_{\kappa+1}^\infty$ that satisfies the following. For every initial input list $u_0^\kappa \in \mathscr{J}(\kappa)$, the complete tail set $(T(\Sigma_{co}\{u_0^\kappa\})$ of the output sequence of the interpreter–controller combination satisfies $T(\Sigma_{co}\{u_0^\kappa\}) \cap T \neq \varnothing$.

In the above problem the tail set T is called the *target tail set* of Σ; it represents 'acceptable' or 'desirable' behaviour. The controller C is said to steer Σ from $\mathscr{J}(\kappa)$ to T. If a controller C that steers Σ from $\mathscr{J}(\kappa)$ to T exists, then T and $\mathscr{J}(\kappa)$ are said to be *compatible* (for the interpreter Σ). The issue of compatibility is, of course, of central interest to us.

Remark 2: In essence, the problem of corrective control deals with the control of interpreters whose status is not exactly known at the step κ at which control is initiated. In our current presentation, the uncertainty about this status is represented through an uncertainty about the initial input list of the interpreter, expressed by the set $\mathscr{J}(\kappa)$. Alternatively, one could directly describe the indeterminacy of the status at the step κ as a set of possible stati.

A notion analogous to the usual control theoretic notion of reachability is relevant in the present context. A complete tail T is *reachable* from an initial input list $\alpha \in \mathscr{J}(\kappa)$ if there is a sequence $u \in (D)_{\kappa+1}^\infty$ such that $T(\Sigma_o \alpha u) \cap T \neq \varnothing$. The following statement is a consequence of the definitions.

Proposition 2: *Let $\Sigma = (\Sigma_o, \Sigma_m): D \to SS_A$ be an interpreter with the initial input set $\mathscr{J}(\kappa)$ and the target tail set T. If T is compatible with $\mathscr{J}(\kappa)$, then T is reachable from every element of $\mathscr{J}(\kappa)$.*

However, it will become clear later that even if T is reachable from every initial input list $\alpha \in \mathscr{I}(\kappa)$, it may still be the case that T is incompatible with $\mathscr{I}(\kappa)$ (see Lemma 1 below).

An *autonomous open loop controller* C for the interpreter $\Sigma : D \rightarrow SS_A$ is a controller that generates a fixed sequence, independent of the response of Σ, i.e. $C : \varnothing \rightarrow (D)_{\kappa+1}^{\infty}$. Information about Σ, the initial input set $\mathscr{I}(\kappa)$, and the target tail set T is incorporated into the design of C, but no updates about the response of Σ are provided during operation.

From a practical perspective, open-loop autonomous controllers are quite attractive, since they do not require monitoring of the response of Σ. However, as one might expect, the capabilities of open-loop controllers are rather limited. The following necessary and sufficient condition for the existence of an open-loop controller is directly implied by the fact that such controller can generate only one fixed output sequence.

Proposition 3: *Let $\Sigma = (\Sigma_o, \Sigma_m) : D \rightarrow SS_A$ be an interpreter with the initial input set $\mathscr{I}(\kappa)$ and the target tail set T. An open-loop autonomous controller that steers Σ from $\mathscr{I}(\kappa)$ to T exists if and only if the following holds. There is a sequence $w \in (D)_{\kappa+1}^{\infty}$ such that, for every list $\alpha \in \mathscr{I}(\kappa)$, one has $T(\Sigma_o \alpha w) \cap T \neq \varnothing$.*

The question of whether an open-loop controller is feasible or not depends, among other factors, on the size of the initial input set $\mathscr{I}(\kappa)$. Consider, for instance, the simple case where $\mathscr{I}(\kappa)$ contains only one single list $\alpha = u_0^{\kappa}$, i.e. when exact data about the input history of Σ are available. In this case, Proposition 3 reduces to the statement that an open-loop controller exists if and only if the target tail set T is reachable from α. Yet, in view of Proposition 2, the latter is a necessary condition for the existence of any controller that steers Σ from $\mathscr{I}(\kappa)$ to T. Thus, if the input history of Σ is known precisely, an open-loop controller is feasible whenever any other controller is feasible.

However, in most cases of practical interest, the initial input set $\mathscr{I}(\kappa)$ contains more than just one list (i.e. exact information about the input history of Σ is not available). In many of these cases, different initial input lists require different continuations in order to steer Σ to the target trail set T. An open-loop controller, which is capable of generating only one predetermined continuation, would then be inadequate. Instead, one would resort to a feedback controller, i.e. to a usual autonomous controller $C : (\operatorname{Im} \Sigma_m)_{\kappa}^{\infty} \rightarrow (D)_{\kappa+1}^{\infty}$. We have here a demonstration of the well known principle that feedback controllers are necessitated by incomplete (or uncertain) data about the system being controlled.

Consider an autonomous controller $C : (\operatorname{Im} \Sigma_m)_{\kappa}^{\infty} \rightarrow (D)_{\kappa+1}^{\infty}$ combined at the step $\kappa \geqslant 0$ with the interpreter $\Sigma = (\Sigma_m, \Sigma_o) : D \rightarrow SS_A$. Let $\mathscr{I}(\kappa)$ be the initial input set, and let T be the target tail set of Σ. As before, let $\gamma_{\kappa+1}^{\infty} = C\{u_0^{\kappa}\}$ be the output list of the controller C corresponding to the initial input list $u_0^{\kappa} \in \mathscr{I}(\kappa)$ of Σ, and let $\mu := \Sigma_{cm}\{u_0^{\kappa}\}$ be the monitored sequence of the interpreter–controller combination. The entire input list of Σ is then $v = (u_0, \ldots, u_{\kappa}, \gamma_{\kappa+1}, \gamma_{\kappa+2}, \ldots)$. By the strict causality of C, the sentence γ_{j+1} is determined by the elements $\mu_{\kappa}, \mu_{\kappa+1}, \ldots, \mu_j$, for all $j \geqslant \kappa$. This justifies the notation.

$$(C\mu_{\kappa}^{j})_{j+1} := \gamma_{j+1}, \quad j \geqslant \kappa \tag{6}$$

In particular, $\gamma_{\kappa+1} = (C\mu_{\kappa})_{\kappa+1}$. By the causality of Σ, the monitored list μ_{κ}^{j} is determined by the input list $v_0, v_1, \ldots, v_j$ of Σ, so we shall write

$$(\Sigma_m v_0 v_1 \ldots v_j)_{\kappa}^{j} := \mu_{\kappa}^{j}, \quad j \geqslant \kappa$$

For each list $\alpha \in \mathscr{J}(\kappa)$, let $U_1(\alpha)$ be the set of all sequences $w \in (D)_{\kappa+1}^{\infty}$ satisfying $T(\Sigma_o \alpha w) \cap T \neq \varnothing$. In other words, $U_1(\alpha)$ consists of all continuations of the list α that lead Σ to T. Clearly, a necessary condition for the existence of a corrective controller is that $U_1(\alpha) \neq \varnothing$ for all $\alpha \in \mathscr{J}(\kappa)$, i.e. that T be reachable from every $\alpha \in \mathscr{J}(\kappa)$. However, in general, this condition is not sufficient. The following necessary and sufficient condition for the existence of an autonomous feedback controller is basically a consequence of strict causality. We shall discuss computable forms of it in subsequent sections.

Lemma 1: *Let $\Sigma = (\Sigma_o, \Sigma_m): D \to SS_A$ be a causal interpreter with the initial input set $\mathscr{J}(\kappa)$ and the target tail set T. For each $\alpha \in \mathscr{J}(\kappa)$, let $U_1(\alpha)$ be the set of all sequences $u \in (D)_{\kappa+1}^{\infty}$ satisfying $T(\Sigma_o \alpha u) \cap T \neq \varnothing$. Then, the following two statements are equivalent.*

(a) T is compatible with $\mathscr{J}(\kappa)$ for Σ.

(b) There is a function $F: \mathscr{J}(\kappa) \to \bigcup_{\alpha \in \mathscr{J}(\kappa)} U_1(\alpha): \alpha \mapsto F(\alpha)$ for which the following holds. For each pair $\alpha, \alpha' \in \mathscr{J}(\kappa)$ and for each integer $k \geqslant \kappa$, one has $(F(\alpha))_{k+1} = (F(\alpha'))_{k+1}$ whenever $(\Sigma_m \alpha F(\alpha))_{\kappa}^{k} = (\Sigma_m \alpha' F(\alpha'))_{\kappa}^{k}$.

Proof: We show first that (*a*) implies (*b*). Assume that T is compatible with $\mathscr{J}(\kappa)$; then, there is a strictly causal autonomous controller $C: (\operatorname{Im} \Sigma_m)_{\kappa}^{\infty} \to (D)_{\kappa+1}^{\infty}$ that steers Σ from $\mathscr{J}(\kappa)$ to T. Using the notation of (5), the controller C generates, for each $\alpha \in \mathscr{J}(\kappa)$, a list $v(\alpha) := C\{\alpha\} \in (D)_{\kappa+1}^{\infty}$ satisfying $T(\Sigma_o \alpha v(\alpha)) \cap T \neq \varnothing$. Clearly, $v(\alpha) \in U_1(\alpha)$, and we obtain a function $F: \mathscr{J}(\kappa) \to \bigcup_{\alpha \in \mathscr{J}(\kappa)} U_1(\alpha): \alpha \mapsto F(\alpha) := v(\alpha)$. Now, let $k \geqslant \kappa$ be an integer, and let $\alpha, \alpha' \in \mathscr{J}(\kappa)$ be two lists for which $(\Sigma_m \alpha F(\alpha))_{\kappa}^{k} = (\Sigma_m \alpha' F(\alpha'))_{\kappa}^{k}$. Then, using the strict causality of C and (6), we have $(F(\alpha))_{k+1} = (C(\Sigma_m \alpha F(\alpha))_{\kappa}^{k})_{k+1} = (C(\Sigma_m \alpha' F(\alpha'))_{\kappa}^{k})_{k+1} = (F(\alpha'))_{k+1}$, and the final part of the Lemma is necessary.

To prove the converse direction, i.e. that (*b*) implies (*a*), assume that (*b*) holds. Using (*b*), we construct an autonomous strictly causal controller C that steers Σ from $\mathscr{J}(\kappa)$ to T as follows. First, some notation. Let V_{m} be the monitored value set of Σ. Given an integer $j \geqslant \kappa$ and a list of sentences $\beta_\kappa, \ldots, \beta_j \in V_{\mathrm{m}}$, let $E(\beta_\kappa^j)$ be the (possibly empty) class of all elements $\alpha \in \mathscr{J}(\kappa)$ satisfying $(\Sigma_m \alpha F(\alpha))_{\kappa}^{j} = \beta_\kappa^j$, where F is the function of (*b*). By (*b*) one has $(F(\alpha))_{j+1} = (F(\alpha'))_{j+1}$ for all pairs $\alpha, \alpha' \in E(\beta_\kappa^j)$. This allows us to define a map $C: (\operatorname{Im} \Sigma_m)_{\kappa}^{\infty} \to (D)_{\kappa+1}^{\infty}$ in the following way. For every integer $j \geqslant \kappa$ and for every list $\beta_\kappa, \ldots, \beta_j \in V_{\mathrm{m}}$ set $(C\beta_\kappa^j)_{j+1} := \varnothing$ if $E(\beta_\kappa^j) = \varnothing$; otherwise, pick any element $\alpha \in E(\beta_\kappa^j)$, and set

$$(C\beta_\kappa^j)_{j+1} := (F(\alpha))_{j+1}$$

The map C is clearly strictly causal. Furthermore, in terms of the notation of (5), the combination Σ_{c} satisfies $\Sigma_{\mathrm{c}}\{\alpha\} = \Sigma \alpha F(\alpha)$ for every $\alpha \in \mathscr{J}(\kappa)$. Since $F(\alpha) \in U_1(\alpha)$, it follows that $T(\Sigma_{\mathrm{co}}\{\alpha\}) \cap T \neq \varnothing$ for all $\alpha \in \mathscr{J}(\kappa)$; whence, the controller C steers Σ from $\mathscr{J}(\kappa)$ to T. $\square$

We summarize separately the structure of the controller C derived in the above proof.

Corollary 1: *Let $\Sigma = (\Sigma_o, \Sigma_m): D \to SS_A$ be a causal interpreter having the initial input set $\mathscr{J}(\kappa)$ and the target tail set T. Let V_{m} be the monitored value set of Σ. Assume that* *the condition (b) of Lemma 1 holds. For every integer $j \geqslant \kappa$, and for every list of elements $\beta_\kappa, \ldots, \beta_j \in V_{\mathrm{m}}$, let $E(\beta^j_\kappa)$ be the class of all elements $\alpha \in \mathscr{J}(\kappa)$ satisfying $(\Sigma_{\mathrm{m}}\alpha F(\alpha))^j_\kappa = \beta^j_\kappa$. Then, the assignment*

$$(C\beta^j_\kappa)_{j+1} := \begin{cases} \varnothing & \text{if } E(\beta^j_\kappa) = \varnothing \\ (F(\alpha))_{j+1}, & \text{otherwise, where } \alpha \in E(\beta^j_\kappa) \end{cases}$$

induces a strictly causal controller $C:(\operatorname{Im}\Sigma_m)^\infty_\kappa \to (D)^\infty_{\kappa+1}$ that steers Σ from $\mathscr{J}(\kappa)$ to T.

In subsequent sections we shall see that a computable and implementable construction can be directly derived from the Corollary in cases of practical interest, where appropriate finiteness requirements hold.

4. Compatibility of target sets

Up to this point, the discussion has been on a rather general level, without regard to whether the various quantities or computational procedures are finite or infinite. In this section, we focus our attention on cases of practical interest, which are inherently of a finite nature. This will allow us to develop implementable forms of corrective controllers. The following notion is instrumental.

Definition 1: An interpreter $\Sigma: D \to SS_A$ is *bounded* if it has a recursive representation of the form (1) with a finite state set, and if the following hold: the domain D is uniform; and the input, internal, monitored and output value sets of Σ are all finite.

Our discussion of the corrective control of bounded interpreters depends on certain periodic properties, which we now examine. A sequence $s \in S(S_A)$ is ultimately periodic if there is a pair of integers $\eta \geqslant 0, \tau > 0$ such that $s_{i+\tau} = s_i$ for all $i \geqslant \eta$. The smallest such integer τ is called the *ultimate period* of the sequence; the first such integer η indicates the step at which periodicity commences. If $\eta = 0$ the sequence is periodic. The following simple conclusion is closely related to a well known property of finite automata.

Proposition 4: *The augmented output sequence of a bounded autonomous interpreter $\Sigma: \varnothing \to SS_A$ is ultimately periodic. Its period does not exceed $n := (\# V_t)(\# X)$, where V_t is the internal value set of Σ, and X is its state set. Furthermore, periodicity commences within the first n steps.*

Proof: Being a bounded autonomous interpreter, Σ has a recursive representation

$$\left.\begin{aligned} (s_{k+1}, x_{k+1}) &= f(s_k, x_k) \\ y_k &= h_o(s_k) \\ \mu_k &= h_m(s_k), k = 0, 1, 2, \ldots \end{aligned}\right\} \tag{7}$$

Note that Σ has no input sequence, since it is autonomous. Now, let V_t be the internal value set of Σ, and let X be its state set, both of which are finite sets by the boundedness of Σ. Then, the set

$$\{(s, x): s \in V_t, x \in X\}$$

is clearly finite, containing only $(\# V_t)(\# X)$ points. This implies that the sequence of pairs $(s_0, x_0), (s_1, x_1), (s_2, x_2), \ldots$ must have two identical elements in it, separated by at most $(\# V_t)(\# X)$ steps. In other words, there is a pair of integers $m \geqslant 0, \tau > 0$ satisfying $(s_m, x_m) = (s_{m+\tau}, x_{m+\tau})$, with $m, \tau \leqslant (\# V_t)(\# X)$. However, the recursion (7) implies that

$(s_{m+j}, x_{m+j}) = (s_{m+\tau+j}, x_{m+\tau+j})$, or $(s_{(m+j)}, x_{(m+j)}) = (s_{(m+j)+\tau}, x_{(m+j)+\tau})$, for all integers $j \geqslant 0$. Thus, the sequence $(s_0, x_0), (s_1, x_1), (s_2, x_2), \ldots$ is ultimately periodic with a period not exceeding τ, and its periodicity commences at a step no later than m, where $m, \tau \leqslant (\# V_t)(\# X)$. Since the augmented output sequence of Σ is determined here by s through the equation $(y, \mu) = (h_o(s), h_m(s))$, the assertion follows. □

Consider now a bounded interpreter Σ with a target tail set T that is compatible with the initial input set $\mathscr{J}(\kappa)$ of Σ, and let C be a corrective controller that steers Σ from $\mathscr{J}(\kappa)$ to T. The combination of C with Σ operates as an autonomous interpreter, and generates a tail $t \in T$. Now from a practical perspective the only case of interest is the one where the interpreter–controller combination Σ_c constitutes a bounded interpreter; otherwise, the controller would not have a finite implementation. When Σ_c is bounded, Proposition 4 implies that t must be ultimately periodic; thus, we may restrict our attention to target tail sets that consist entirely of ultimately periodic sequences. Furthermore, ultimately periodic sequences always have periodic tails, and whence it suffices to consider target tail sets that consist of periodic tails only. This leads to the following definition.

Definition 2: A *bounded target tail* set is a complete tail set that consists of all tails of a finite collection of periodic sequences.

An autonomous controller that steers a bounded interpreter Σ towards a bounded target tail set has to generate an input sequence for Σ that elicits from Σ an ultimately periodic output sequence. We examine next some simple properties of input sequences that generate ultimately periodic output sequences of (non-autonomous) bounded interpreters.

It is quite easy to construct an example of a (non-injective) bounded interpreter $\Sigma: D \to SS_A$ where an ultimately periodic output sequence is generated by an input sequence that is not ultimately periodic. For instance, consider a constant interpreter $\Sigma: D \to SS_A$ that produces the constant output sequence $a, a, a, \ldots$ for every input sequence, and that has a uniform domain D whose value set contains at least two distinct sentences. Then the output sequence of Σ is clearly periodic for every input sequence; since D contains sequences that are not ultimately periodic, we see that an ultimately periodic output sequence can be generated by an input sequence that is not ultimately periodic. The next statement indicates, however, that, for a bounded interpreter, one can always find an ultimately periodic input sequence that generates any given ultimately periodic output sequence.

Proposition 5: *Let $\Sigma = (\Sigma_o, \Sigma_m): D \to SS_A$ be a bounded interpreter. For every ultimately periodic output sequence $w \in \operatorname{Im} \Sigma_o$, there is an ultimately periodic input sequence $v \in D$ satisfying $w = \Sigma_o v$.*

Proof: *Let $u \in D$* be an input sequence for which the output sequence $w := \Sigma_o u$ is ultimately periodic. Since w is ultimately periodic, there is a pair of integers $\eta \geqslant 0$, $\tau > 0$ such that

$$w_{i+\tau} = w_i \tag{8}$$

for all integers $i \geqslant \eta$. Let x be the sequence of states induced by u, and let s be the internal sequence. Since Σ is recursive, we have

$$\begin{aligned}(s_{k+1}, x_{k+1}) &= f[(s_k \dot{\cup} u_k), x_k] \\ w_k &= h_o(s_k \dot{\cup} u_k), k = 0, 1, 2, \ldots\end{aligned} \tag{9}$$

Consider the sequence of triples

$$(s_\eta, x_\eta, u_\eta), (s_{\eta+\tau}, x_{\eta+\tau}, u_{\eta+\tau}), (s_{\eta+2\tau}, x_{\eta+2\tau}, u_{\eta+2\tau}), \ldots \tag{10}$$

Let V_t be the internal value set, let X be the state set, and let V_{in} be the input value set of Σ. By the boundedness of Σ, the set V_t, X, and V_{in} are finite. Consequently, the set of triples

$$\{(a, b, c) \mid a \in V_t, b \in X, \text{ and } c \in V_{\text{in}}\}$$

has finite cardinality $\theta \leqslant (\# V_t)(\# X)(\# V_{\text{in}})$. Applying this fact to the sequence (10), it follows that there is a pair of integers $m > n \geqslant 0$, where $(m-n) \leqslant \theta$ and $n \leqslant \theta$, such that

$$(s_{\eta+m\tau}, x_{\eta+m\tau}, u_{\eta+m\tau}) = (s_{\eta+n\tau}, x_{\eta+n\tau}, u_{\eta+n\tau}) \tag{11}$$

Since $w_{i+\tau} = w_i$ for all integers $i \geqslant \eta$, we also have

$$(w_{\eta+m\tau}, s_{\eta+m\tau}, x_{\eta+m\tau}, u_{\eta+m\tau}) = (w_{\eta+n\tau}, s_{\eta+n\tau}, x_{\eta+n\tau}, u_{\eta+n\tau}) \tag{12}$$

Define now the concatenated input sequence

$$v := u_0^{\eta+n\tau-1} u_{\eta+n\tau}^{\eta+m\tau-1} u_{\eta+n\tau}^{\eta+m\tau-1} u_{\eta+n\tau}^{\eta+m\tau-1} \ldots \tag{13}$$

which is clearly ultimately periodic. Since D is a uniform domain, $v \in D$. Apply v as an input sequence of Σ, denoting by σ the resulting internal sequence; by ξ the resulting sequence of states; and by ω the resulting output sequence. Since a recursive interpreter is defined together with its initial status, we have

$$\sigma_0 = s_0, \quad \xi_0 = x_0 \tag{14}$$

The recursive representation of Σ yields

$$\left.\begin{aligned} (\sigma_{k+1}, \xi_{k+1}) &= f[(\sigma_k \dot{\cup} v_k), \xi_k] \\ \omega_k &= h_o(\sigma_k \dot{\cup} v_k), \quad k = 0, 1, 2, \ldots \end{aligned}\right\} \tag{15}$$

From (13), we obtain that $v_0^{\eta+m\tau-1} := u_0^{\eta+m\tau-1}$, which, together with (14) and (15), yields

$$\begin{aligned} \xi_0^{\eta+m\tau} &= x_0^{\eta+m\tau} \\ \sigma_0^{\eta+m\tau} &= s_0^{\eta+m\tau} \end{aligned} \tag{16}$$

and

$$\omega_0^{\eta+m\tau-1} = w_0^{\eta+m\tau-1} \tag{17}$$

Assume now for a moment that

$$\sigma_{\eta+[n+i(m-n)]\tau}^{\eta+[n+(i+1)(m-n)]\tau} = s_{\eta+n\tau}^{\eta+m\tau} \tag{18}$$

for all integers $i \geqslant 0$. Then, in view of (13), it follows that

$$(\sigma \dot{\cup} v)_{\eta+[n+i(m-n)]\tau}^{\eta+[n+(i+1)(m-n)]\tau-1} = (s \dot{\cup} u)_{\eta+n\tau}^{\eta+m\tau-1}$$

for all integers $i \geqslant 0$. Since $\omega_k = h_o(\sigma_k \dot{\cup} v_k)$ for all $k \geqslant 0$, we obtain that

$$\omega_{\eta+[n+i(m-n)]\tau}^{\eta+[n+(i+1)(m-n)]\tau-1} = w_{\eta+n\tau}^{\eta+m\tau-1}$$

for all integers $i \geqslant 0$; in view of (8) and (17), this shows that $\omega = w$, and w can be generated by the ultimately periodic input sequence v. (Note, however, that the pair (σ, ξ) may not be identical to the pair (s, x)). Thus, our proof will conclude upon proving (18).

We prove (18) by induction. Simultaneously, it will be convenient to also show that

$$\xi_{\eta+[n+(i+1)(m-n)]\tau} = x_{\eta+n\tau} \tag{19}$$

for all integers $i \geqslant 0$. Consider first the case $i = 0$. By (16), we have, in particular, that $\sigma_{\eta+n\tau}^{\eta+m\tau} = s_{\eta+n\tau}^{\eta+m\tau}$ and that $\xi_{\eta+m\tau} = x_{\eta+m\tau}$. When the latter is combined with (11), we obtain $\xi_{\eta+m\tau} = x_{\eta+m\tau} = x_{\eta+n\tau}$. Whence, (18) and (19) hold for the case $i = 0$.

In preparation for induction, assume that (18) and (19) hold for the case $i = j$. Then, $\sigma_{\eta+[n+(j+1)(m-n)]\tau} = s_{\eta+n\tau}$ and $\xi_{\eta+[n+(j+1)(m-n)]\tau} = x_{\eta+n\tau}$; and, by (13), $v_{\eta+[n+(j+1)(m-n)]\tau}^{\eta+[n+(j+2)(m-n)]\tau-1} = u_{\eta+n\tau}^{\eta+m\tau-1}$. This shows that the recursion (15) over the interval $k = \eta+[n+(j+1)(m-n)]\tau, \ldots, \eta+[n+(j+2)(m-n)]\tau-1$ is identical to the recursion (9) over the interval $k = \eta+n\tau, \ldots, \eta+m\tau-1$. Thus, both recursions yield the same result, i.e.

$$\sigma_{\eta+[n+(j+1)(m-n)]\tau}^{\eta+[n+(j+2)(m-n)]\tau} = s_{\eta+n\tau}^{\eta+m\tau} \tag{20}$$

and

$$\xi_{\eta+[n+(j+1)(m-n)]\tau}^{\eta+[n+(j+2)(m-n)]\tau} = x_{\eta+n\tau}^{\eta+m\tau} \tag{21}$$

However (20) is (18) for the case $i = j+1$; and (21) together with (11) show that $\xi_{\eta+[n+(j+2)(m-n)]\tau} = x_{\eta+m\tau} = x_{\eta+n\tau}$, which is (19) for the case $i = j+1$. By induction, this corroborates the validity of (18) and (19) for all integers $i \geqslant 0$. In view of earlier remarks, the proof of the Proposition is now complete. □

The proof contained the following bound on the period of the ultimately periodic input sequence of Proposition 5.

Corollary 2: *In Proposition 5 the period of the ultimately periodic input sequence v does not exceed $\tau(\# V_t)(\# X)(\# V_{\text{in}})$, and periodicity of v commences no later than step $\eta+\tau(\# V_t)(\# X)(\# V_{\text{in}})$. Here, τ is the period of the output sequence w, and the periodicity of w commences at the step η; also, X is the state set, V_t is the internal value set, and V_{in} is the input value set of the bounded interpreter Σ.*

We next examine some basic properties of tail sets of periodic sequences. Let $c := \{u_0, u_1, \ldots, u_{\tau-1}\}$ be a list of sentences $u_j \in S_A, j = 0, 1, \ldots, \tau-1$. Using concatenation, we create from the list c the periodic sequence $\chi(c) := ccc\ldots$ A *cycle* of $\chi(c)$ is any cyclic permutation of the list c, i.e. c itself or any list of the form $\{u_j, u_{j+1}, \ldots u_{\tau-1}, u_0, u_1, \ldots, u_{j-1}\}, j \in \{1, \ldots, \tau-1\}$. Despite some abuse of notation, it will be convenient to denote every cycle of $\chi(c)$ by the letter c. The length of the cycle c is τ in the present case. Given several lists $c_1, c_2, \ldots, c_m$, we denote by

$$\chi(c_1, c_2, \ldots, c_m) := \bigcup_{i=1,\ldots,m} \chi(c_i)$$

the union of all respective periodic sequences. By definition, every bounded tail set is of the form $\chi(c_1, c_2, \ldots, c_m)$ for some cycles $c_1, \ldots, c_m$.

4.1. *Basic properties of corrective controllers*

Our next objective is to derive necessary and sufficient conditions for the compatibility of an initial input set $\mathscr{I}(\kappa)$ with a bounded target tail set, namely, with a target tail set of the form $\chi(c_1, c_2, \ldots, c_m)$. We start with the case where $m = 1$, i.e. with the problem of steering Σ from $\mathscr{I}(\kappa)$ towards one specific periodic tail $\chi(c)$.

When $\chi(c)$ is compatible with $\mathscr{I}(\kappa)$, there is, by definition, an autonomous controller that steers Σ from $\mathscr{I}(\kappa)$ to $\chi(c)$. The next statement shows that such a controller can be constructed so that its own output sequence also is ultimately periodic. This basically guarantees that the controller is implementable. The implementation issue is discussed later in this section.

Lemma 2: *Let $\Sigma: D \to SS_A$ be a bounded interpreter with initial input set $\mathcal{J}(\kappa)$, input value set V_{in}, internal value set V_t, and state set X. Assume that $\chi(c)$ forms a target tail set that is compatible with $\mathcal{J}(\kappa)$, and let τ be the length of the cycle c. Then there is a strictly causal autonomous controller C_δ that steers Σ from $\mathcal{J}(\kappa)$ to $\chi(c)$ and satisfies the following conditions for every element $\alpha \in \mathcal{J}(\kappa)$.*

The output sequence $C_\delta\{\alpha\}$ of the controller is ultimately periodic.

The period τ_δ of $C_\delta\{\alpha\}$ satisfies $\tau_\delta \leqslant \tau\{(\# V_t)(\# X)(\# V_{\text{in}})^2\}^{(\#\mathcal{J}(\kappa))}$, and periodicity of $C_\delta\{\alpha\}$ commences by the step $\eta_\delta \leqslant \kappa + 1 + \{(\# V_t)(\# X)(\# V_{\text{in}})^2\}^{(\#\mathcal{J}(\kappa))}$.

Proof: Since $\chi(c)$ is compatible with $\mathcal{J}(\kappa)$ there is, by definition, a strictly causal autonomous controller $C: (\operatorname{Im} \Sigma_{\text{m}})_\kappa^\infty \to (D)_{\kappa+1}^\infty$ that steers Σ from $\mathcal{J}(\kappa)$ to $\chi(c)$. Our objective is to show that in such case a controller C_δ satisfying the Lemma also exists.

For every $\alpha \in \mathcal{J}(\kappa)$, denote by $C\{\alpha\}$ the sequence generated by the controller C for the initial input list α. (Recall that $C\{\alpha\}$ starts at the step $\kappa+1$.) Let Σ_{m} be the monitored part of Σ. Then, by the strict causality of C, the following holds. For every pair $\alpha, \alpha' \in \mathcal{J}(\kappa)$ and for every integer $k \geqslant \kappa$, one has

$$(C\{\alpha\})_{\kappa+1}^{k+1} = (C\{\alpha'\})_{\kappa+1}^{k+1} \text{ whenever } (\Sigma_{\text{m}}\, \alpha C\{\alpha\})_\kappa^k = (\Sigma_{\text{m}}\, \alpha' C\{\alpha'\})_\kappa^k \tag{22}$$

The interpreter Σ, being bounded, has a recursive representation (1), with internal sequence s, state sequence x, monitored sequence μ, and output sequence y. For every $\alpha \in \mathcal{J}(\kappa)$, let $s(\alpha)$, $x(\alpha)$, $\mu(\alpha)$ and $y(\alpha)$ be the respective sequences generated by the recursive representation of Σ when Σ is driven by the concatenation $\alpha C\{\alpha\}$. Since C steers Σ to the periodic tail $\chi(c)$, the sequence $y(\alpha)$ is ultimately periodic with cycle c.

Let V_{in} be the input value set, let V_t be the internal value set, and let X be the state set of Σ. By the boundedness of Σ, these sets are all finite. Consider the quintuple

$$q(\alpha)_k := (s(\alpha)_k, x(\alpha)_k, \mu(\alpha)_k, y(\alpha)_k, (C\{\alpha\})_{k+1})$$

for fixed $\alpha \in \mathcal{J}(\kappa)$ and $k \geqslant \kappa$. Note that $(C\{\alpha\})_{k+1} \in V_{\text{in}}$, and that, according to (1), $\mu(\alpha)_k$ and $y(\alpha)_k$ are both determined by $s(\alpha)_k \in V_t$ and by $u_k = (C\{\alpha\})_k \in V_{\text{in}}$. Thus, the total number of possible distinct quintuples $q(\alpha)_k$ cannot exceed

$$\rho := (\# V_t)(\# X)(\# V_{\text{in}})^2 \tag{23}$$

The cardinality n of the initial input value set $\mathcal{J}(\kappa)$ must clearly satisfy $n \leqslant (\# V_{\text{in}})^{\kappa+1}$. Let $\alpha_1, \ldots, \alpha_n$ be the elements of $\mathcal{J}(\kappa)$. For every integer $k = \kappa, \kappa+1, \ldots,$ define the list of n quintuples

$$\Delta_k := \{q(\alpha_1)_k, q(\alpha_2)_k, \ldots, q(\alpha_n)_k\}$$

By (23), the total number r of possible lists Δ_k, for each integer k, satisfies

$$r \leqslant n^* := \{(\# V_t)(\# X)(\# V_{\text{in}})^2\}^n \tag{24}$$

Let $\eta(\alpha)$ be the step at which periodicity of $y(\alpha)$ commences, and let τ be the period of $\chi(c)$, which is also the ultimate period of $y(\alpha)$. Denote

$$\eta := \max\{\eta(\alpha_1), \ldots, \eta(\alpha_n), \kappa\}$$

and consider the subsequence $\Delta_\eta, \Delta_{\eta+\tau}, \Delta_{\eta+2\tau}, \ldots$ In view of (24), there is a pair of integers $m > p \geqslant 0$, $(m-p) \leqslant n^*$, such that $\Delta_{\eta+m\tau} = \Delta_{\eta+p\tau}$.

Consider the list $\Delta_\kappa^{\eta+p\tau-1}$, which is non-empty only when $\eta+p\tau-1 \geqslant \kappa$. We construct a list λ as follows.

If $[(\eta+p\tau-1)-\kappa] \leqslant n^*$, set $\lambda := \Delta_\kappa^{\eta+p\tau-1}$.

If $[(\eta+p\tau-1)-\kappa] > n^*$, there must be a pair of integers $j > i$, where $(\eta+p\tau-1) \geqslant j > i \geqslant \kappa$, for which $\Delta_i = \Delta_j$. Construct a new list $\lambda_1 := \{\Delta_\kappa, \Delta_{\kappa+1}, \dots, \Delta_i, \Delta_{j+1}, \dots, \Delta_{\eta+p\tau-1}\}$ by omitting $\Delta_{i+1}, \dots, \Delta_j$; the number of elements in the list λ_1 is given by $\sigma_1 := (\eta+p\tau-1)-\kappa-(j-i)$. If $\sigma_1 > n^*$, repeat the same process on λ_1 to obtain a list λ_2, and then on λ_2 to obtain a list λ_3, etc., until a list λ_q is obtained whose number of elements σ_q satisfies $\sigma_q \leqslant n^*$. Then, set $\lambda := \lambda_q$.

With the list λ, use concatenation to construct the sequence

$$\delta := \lambda \Delta_{\eta+p\tau}^{\eta+m\tau-1} \Delta_{\eta+p\tau}^{\eta+m\tau-1} \Delta_{\eta+p\tau}^{\eta+m\tau-1} \dots \tag{25}$$

the elements of this sequence are renumbered consecutively as $\delta_\kappa, \delta_{\kappa+1}, \delta_{\kappa+2}, \dots$, where each element δ_k is a list of n quintuples that we denote by

$$\delta_k = \{(s_\delta(\alpha_1)_k, x_\delta(\alpha_1)_k, \mu_\delta(\alpha_1)_k, y_\delta(\alpha_1)_k, (C_\delta \alpha_1)_{k+1}), (s_\delta(\alpha_2)_k, x_\delta(\alpha_2)_k, \mu_\delta(\alpha_2)_k,$$
$$y_\delta(\alpha_2)_k, (C_\delta \alpha_2)_{k+1}), \dots, (s_\delta(\alpha_n)_k, x_\delta(\alpha_n)_k, \mu_\delta(\alpha_n)_k, y_\delta(\alpha_n)_k, (C_\delta \alpha_n)_{k+1})\}$$

Apply now the sequence $(C_\delta(\alpha_i))_{\kappa+1}, (C_\delta(\alpha_i))_{\kappa+2}, \dots$ as an input sequence for Σ with the initial input list α_i. From our construction and the recursive representation (1), it follows that the output sequence of Σ becomes $y_\delta(\alpha_i)$, the state sequence is $x_\delta(\alpha_i)$, and the monitored sequence is $\mu_\delta(\alpha_i)$, for all $i = 1, \dots, n$. Based on the fact that the original output sequence $y(\alpha_i)$ is ultimately periodic with cyclic c, our construction implies that $y_\delta(\alpha_i)$ is also ultimately periodic with cycle c for all $i = 1, \dots, n$.

Consider the assignment

$$\mu_\delta(\alpha) \mapsto C_\delta(\alpha), \quad \alpha \in \mathscr{I}(\kappa) \tag{26}$$

In view of (22) and (25), it follows by construction that for all $\alpha, \alpha' \in \mathscr{I}(\kappa)$ and for all integers $k \geqslant \kappa$

$$(C_\delta(\alpha))_{\kappa+1}^{k+1} = (C_\delta(\alpha'))_{\kappa+1}^{k+1} \text{ whenever } (\Sigma_{\mathrm{m}} \alpha C_\delta(\alpha))_\kappa^k = (\Sigma_{\mathrm{m}} \alpha' C_\delta(\alpha'))_\kappa^k$$

Whence, by Corollary 1 the assignment (26) induces a strictly causal controller $C_\delta : (\mathrm{Im}\, \Sigma_{\mathrm{m}})_\kappa^\infty \to (D)_{\kappa+1}^\infty$. As indicated earlier, the sequence $C_\delta(\alpha)$, when used as the input sequence of Σ with the initial input list $\alpha \in \mathscr{I}(\kappa)$, elicits from Σ the output sequence $y_\delta(\alpha)$, and $y_\delta(\alpha)$ is ultimately periodic with cycle c. Consequently, C_δ steers Σ from $\mathscr{I}(\kappa)$ to $\chi(c)$. Finally, (25) shows that, for all $i = 1, \dots, n$, the sequence $C_\delta(\alpha_i)$ is ultimately periodic, with period not exceeding $(m-p)\tau \leqslant n^*\tau$, and periodicity commencing by the step $\sigma_q \leqslant n^*$. Thus, the controller C_δ satisfies the Lemma with $\tau_\delta \leqslant \tau n^*$ and $\eta_\delta \leqslant \kappa+1+n^*$. □

As we can see, the complexity of the controller C_δ increases exponentially with the uncertainty about the initial input list of Σ, described by $\# \mathscr{I}(\kappa)$.

The technique employed in the proof of Lemma 2 can be used to derive a finite test that determines whether or not there exists a corrective controller that steers Σ from $\mathscr{I}(\kappa)$ to the periodic tail $\chi(c)$. To this end, let $\Sigma = (\Sigma_{\mathrm{o}}, \Sigma_{\mathrm{m}}) : D \to SS_A$ be a bounded interpreter with the input value set V_{in}, the internal value set V_t, and the state set X. Let $\mathscr{I}(\kappa)$ be the initial input set of Σ, and let the periodic tail $\chi(c)$ be its target tail set. Let τ be the length of the cycle c, and define the integers

$$\left.\begin{aligned} \theta &:= \tau[(\# V_t)(\# X)(\# V_{\mathrm{in}})^2]^{(\# \mathscr{I}(\kappa))} \\ \upsilon &:= [(\# V_t)(\# X)(\# V_{\mathrm{in}})^2]^{(\# \mathscr{I}(\kappa))} \end{aligned}\right\} \tag{27}$$

Let $\varepsilon \geqslant \kappa$ be an integer. For each pair of elements $\alpha \in \mathscr{I}(\kappa)$ and $u \in (D)_{\kappa+1}^{\varepsilon+1}$, let $y(\alpha u)_k$, $\mu(\alpha u)_k$, $s(\alpha u)_k$, and $x(\alpha u)_k$ be the output, monitored, internal and state sentences, respectively, induced by the input list αu at the step $k, k = 0, \ldots, \varepsilon+1$. For each integer $k = \kappa, \ldots, \varepsilon$, define the quintuple

$$Q(\alpha u)_k := (x(\alpha u)_k, s(\alpha u)_k, \mu(\alpha u)_k, y(\alpha u)_k, u_{k+1}) \tag{28}$$

The following statement provides a finite test to determine whether or not $\chi(c)$ is compatible with $\mathscr{I}(\kappa)$.

Proposition 6: *Let $\chi(c)$ be a periodic tail with period τ, let θ and υ be given by* (27), *and use the notation of* (28). *Then, $\chi(c)$ is compatible with $\mathscr{I}(\kappa)$ for the bounded interpreter $\Sigma: D \to SS_A$ if and only if, for every element $\alpha \in \mathscr{I}(\kappa)$, there is a list $u(\alpha) \in (D)_{\kappa+1}^{\kappa+\theta+\upsilon+1}$ that satisfies the following.*

(*a*) *There is an integer $\gamma > 0$ such that $(Q(\alpha u(\alpha)))_{\kappa+\theta+\upsilon-\gamma\tau} = (Q(\alpha u(\alpha)))_{\kappa+\theta+\upsilon}$; the list $(y(\alpha u(\alpha)))_{\kappa+\theta+\upsilon-\gamma\tau}^{\kappa+\theta+\upsilon}$ consists of γ cycles c; and $\theta+\upsilon-\gamma\tau \geqslant 0$.*

(*b*) *For each pair $\alpha, \alpha' \in \mathscr{I}(\kappa)$ and for every integer $k = \kappa, \kappa+1, \ldots, \kappa+\theta+\upsilon$, one has $(u(\alpha))_{\kappa+1}^{k+1} = (u(\alpha'))_{\kappa+1}^{k+1}$ whenever $(\mu(\alpha u(\alpha)))_{\kappa}^{k} = (\mu(\alpha' u(\alpha')))_{\kappa}^{k}$.*

According to the Proposition, in order to test whether or not $\chi(c)$ is compatible with $\mathscr{I}(\kappa)$, one would simply search over all pairs $\alpha \in \mathscr{I}(\kappa)$ and $u \in (D)_{\kappa+1}^{\kappa+\theta+\upsilon+1}$ to find if lists $u(\alpha)$, $\alpha \in \mathscr{I}(\kappa)$, that satisfy conditions (*a*) and (*b*) of the Proposition exist. Note that for each u and α the entries of Q can be computed directly from the recursive representation (1) of Σ. Since the input value set V_{in} is finite, the search is finite, and can be executed by computer.

Proof: Assume first that $\chi(c)$ is compatible with $\mathscr{I}(\kappa)$. Then, setting $u(\alpha) := (C_\delta\{\alpha\})_{\kappa+1}^{\kappa+\theta+\upsilon+1}$, where C_δ is the controller constructed in the proof of Lemma 2, it follows directly that (*a*) and (*b*) are satisfied.

Conversely, assume that (*a*) and (*b*) hold. For each element $\alpha \in \mathscr{I}(\kappa)$, consider the concatenated input sequence

$$v(\alpha) := (u(\alpha))_{\kappa+1}^{\kappa+\theta+\upsilon+1} (u(\alpha))_{\kappa+\theta+\upsilon+1-\gamma\tau}^{\kappa+\theta+\upsilon+1} (u(\alpha))_{\kappa+\theta+\upsilon+1-\gamma\tau}^{\kappa+\theta+\upsilon+1} \cdots \tag{29}$$

In view of (*a*) and the proof of Lemma 2, the output sequence $\Sigma_o(\alpha v(\alpha))$ is ultimately periodic with a cycle c. Furthermore, in view of the Proof of Lemma 2, condition (*b*) implies that the assignment

$$C\{\alpha\} := v(\alpha) \tag{30}$$

induces a strictly causal autonomous controller C that steers Σ from $\mathscr{I}(\kappa)$ to $\chi(c)$. Whence, $\chi(c)$ is compatible with $\mathscr{I}(\kappa)$. □

Proposition 6 shows that, for a bounded interpreter Σ, the question of whether or not a periodic target tail set $\chi(c)$ is compatible with the initial input set $\mathscr{I}(\kappa)$ is a decideable question. The description of the controller given by (30) is not directly suitable for implementation, since it refers to the initial input list α, which is not given. An implementable representation of a corrective controller is provided later in this section.

Proposition 6 can be directly generalized to provide the following finite test for the compatibility of any bounded target tail set with the initial input set $\mathscr{I}(\kappa)$. The proof is similar to that of Proposition 6.

Theorem 1: *Let $\Sigma : D \to SS_A$ be a bounded interpreter, with input value set V_{in}, internal value set V_t, state set X, initial input set $\mathscr{I}(\kappa)$, and bounded target tail set $\chi(c_1, \dots, c_m)$. Let τ_i be the length of the cycle c_i. Denote $\tau := \max\{\tau_1, \dots, \tau_m\}$, and, with this value of τ, let θ and υ be given by (27). Then, the target tail set $\chi(c_1, \dots, c_m)$ is compatible with $\mathscr{I}(\kappa)$ for Σ if and only if, for each element $\alpha \in \mathscr{I}(\kappa)$, there is a list $u(\alpha) \in (D)_{\kappa+1}^{\kappa+\theta+\upsilon+1}$ that satisfies the following.*

(*a*) *There is an integer $\gamma > 0$ and a cycle $c_i \in \{c_1, \dots, c_m\}$ such that $(Q(\alpha u(\alpha)))_{\kappa+\theta+\upsilon-\gamma\tau_i} = (Q(\alpha u(\alpha)))_{\kappa+\theta+\upsilon}$; the list $(y(\alpha u(\alpha)))_{\kappa+\theta+\upsilon-\gamma\tau_i}^{\kappa+\theta+\upsilon}$ consists of γ cycles c_i; and $\theta + \upsilon - \gamma\tau_i \geqslant 0$.*

(*b*) *For each pair $\alpha, \alpha' \in \mathscr{I}(\kappa)$ and for every integer $k = \kappa, \kappa+1, \dots, \kappa+\theta+\upsilon$, one has $(u(\alpha))_{\kappa+1}^{k+1} = (u(\alpha'))_{\kappa+1}^{k+1}$ whenever $(\mu(\alpha u(\alpha)))_{\kappa}^{k} = (\mu(\alpha' u(\alpha')))_{\kappa}^{k}$.*

Note that the integer γ in Theorem 1 may depend on the element $\alpha \in \mathscr{I}(\kappa)$.

The theorem provides a finite test that determines whether or not the target tail set $\chi(c_1, \dots, c_m)$ is compatible with $\mathscr{I}(\kappa)$. The test is performed by searching among all lists $u \in (D)_{\kappa+1}^{\kappa+\theta+\upsilon+1}$ for lists $u(\alpha)$ that satisfy conditions (*a*) and (*b*) of the Theorem. The search can be programmed on a digitial computer.

Theorem 1 can also be used to characterize the class of all bounded tail sets that are compatible with the given initial input set $\mathscr{I}(\kappa)$. Of course, altogether, there may be an infinite number of bounded target tail sets that are compatible with $\mathscr{I}(\kappa)$. To avoid the need to deal with infinite sets, these can be divided into finite families as follows.

Consider a target tail of the form $\chi(c_1, \dots, c_m\}$, and let τ_i be the length of the cycle $c_i, i = 1, \dots, m$. A slight reflection shows that the number of distinct cycles that can be induced by an autonomous controller C in combination with an interpreter Σ cannot exceed the number of initial input lists contained in $\mathscr{I}(\kappa)$. Consequently, we can restrict our attention to the case $m \leqslant \# \mathscr{I}(\kappa)$. Let $\mathscr{C}(\Sigma, \tau)$ denote the class of all bounded tails $\chi(c_1, \dots, c_m)$ that are compatible with the initial input set $\mathscr{I}(\kappa)$ for Σ, where $m \leqslant \# \mathscr{I}(\kappa)$ and $\tau_i \leqslant \tau, 1 = 1, \dots, m$. The class $\mathscr{C}(\Sigma, \tau)$ can be derived directly through Theorem 1, by searching over all possible candidates, of which there is only a finite number for each τ. Once the class $\mathscr{C}(\Sigma, \tau)$ is known for the largest τ of interest, a desirable target tail set that is compatible with Σ can be selected from it, if one exists. This approach to selecting a target tail set is usually more practical than an arbitrary prespecification of the target tail set, which might turn out to be incompatible with the initial input set of Σ.

In analogy with (29) and (30), the following formula can be readily shown to provide the response of a strictly causal autonomous controller that steers Σ from $\mathscr{I}(\kappa)$ to $\chi(c_1, \dots, c_m)$.

Lemma 3: *Assume that a class of lists $\{u(\alpha)\}, \alpha \in \mathscr{I}(\kappa)$, satisfying conditions (a) and (b) of Theorem 1 exists. Then, in the notation of the Theorem, the controller C with the response*

$$C\{\alpha\} = (u(\alpha))_{\kappa+1}^{\kappa+\theta+\upsilon+1} (u(\alpha))_{\kappa+\theta+\upsilon+1-\gamma\tau_i}^{\kappa+\theta+\upsilon+1} (u(\alpha))_{\kappa+\theta+\upsilon+1-\gamma\tau_i}^{\kappa+\theta+\upsilon+1} \cdots$$

for all $\alpha \in \mathscr{I}(\kappa)$, is a strictly causal autonomous controller that steers Σ from $\mathscr{I}(\kappa)$ to $\chi(c_1, \dots, c_m)$.

The formula provided for the controller by Lemma 3 is not in implementable form, since it includes a reference to the initial input list α. We consider next the derivation of an implementable formula for the corrective controller. To this end, we shall need to reformulate some of our results into a somewhat more algebraic form.

4.2. *Input/output formulae for the controllers*

Let $\Sigma = \{\Sigma_o, \Sigma_m\}: D \to SS_A$ be a bounded interpreter with the initial input set $\mathcal{J}(\kappa)$. Let V_{in} be the input value set of Σ, let V_t be its internal value set, and let V_m be its monitored value set. By causality, the monitored sentence μ_k generated by Σ at the step k is determined by the input values $u_0, u_1, \dots, u_k$ of Σ. To emphasize this fact, we use the notation

$$\mu(u_0, u_1, \dots, u_k) := \mu_k$$

We define a family of functions $\{\sigma_{\kappa+i}\}_{i=0}^{\infty}$, where $\sigma_{\kappa+i}$ is a function $\mathcal{J}(\kappa) \times (V_{\text{in}})^i \to (\operatorname{Im} \Sigma_m)_{\kappa}^{\kappa+i}$ given by

$$\sigma_{\kappa+i}(\alpha, u_{\kappa+1}, \dots, u_{\kappa+i}) := (\mu(\alpha), \mu(\alpha u_{\kappa+1}), \dots, \mu(\alpha u_{\kappa+1}, \dots, u_{\kappa+i}))$$

The family of functions $\{\sigma_{\kappa+i}\}_{i=0}^{\infty}$ is, of course, directly determined by the monitored part Σ_m of Σ.

In general, given two functions $f: A \to B$ and $g: A \to C$, it is said that the function f factors over the function g if there is a function $h: C \to B$ such that $f = hg$. As is well known, the function f factors over the function g if and only if every equivalence class of the equivalence kernel of g is contained within an equivalence class of the equivalence kernel of f (MaClane and Birkhoff 1979). In intuitive terms, this means that f must be constant over all sets over which g is constant.

For every integer $i \geqslant 0$, let $\Phi_{\kappa+i+1}$ be the family of all functions $\phi: \mathcal{J}(\kappa) \times (V_{\text{in}})^i \to V_{\text{in}}$ that factor over the function $\sigma_{\kappa+i}$. Namely, a function $\phi: \mathcal{J}(\kappa) \times (V_{\text{in}})^i \to V_{\text{in}}$ belongs to $\Phi_{\kappa+i+1}$ exactly when there is a function $\psi: (\operatorname{Im} \Sigma_{\text{m}})_{\kappa}^{\kappa+i} \to V_{\text{in}}$ satisfying

$$\phi = \psi \sigma_{\kappa+i} \tag{31}$$

When the interpreter Σ is bounded, it follows by the finite cardinality of all value sets that, for each integer $i \geqslant 0$, the family $\Phi_{\kappa+i+1}$ contains only a finite number of members. These members can all be computed for a given i from the equivalence kernel of the function $\sigma_{\kappa+i}$, as follows. Let the equivalence kernel of $\sigma_{\kappa+i}$ consist of ε equivalence classes $e_1, \dots, e_\varepsilon$. Then, $\Phi_{\kappa+i+1}$ has $(\# V_{\text{in}})^\varepsilon$ members; each member of $\Phi_{\kappa+i+1}$ is obtained by choosing ε (not necessarily distinct) sentences $a_1, \dots, a_\varepsilon \in V_{\text{in}}$, and defining a function ϕ by setting $\phi(b) := a_i$ for all $b \in e_i$, $i = 1, \dots, \varepsilon$.

Let $\alpha \in \mathcal{J}(\kappa)$ be an element, and let $\phi_1, \dots, \phi_j$ be a list of functions, where $\phi_i \in \Phi_{\kappa+i}$ for all $i = 1, \dots, j$. Define the concatenation

$$u(\alpha\phi_1 \dots \phi_j) := u_{\kappa+1} \dots u_{\kappa+j}$$

where $u_{\kappa+1} := \phi_1(\alpha)$ and, given $u_{\kappa+i}$ for some integer $i \geqslant 1, i < j$, set $u_{\kappa+i+1} := \phi_{i+1}(\alpha u_1 \dots u_i)$. It is convenient to use the notation $\phi := \phi_1 \dots \phi_j$ for the combined function, and $u(\alpha\phi) := u(\alpha\phi_1 \dots \phi_j)$. We also denote by

$$\Phi(j) := \{\phi = \phi_1 \phi_2 \dots \phi_{j+1} \mid \phi_i \in \Phi_{\kappa+i}, i = 1, \dots, j+1\}$$

the family of all such combined functions. For a pair of elements $\alpha \in \mathcal{J}(\kappa)$ and $\phi \in \Phi(j)$, let $(x(\alpha\phi))_k := (x(\alpha u(\alpha\phi)))_k$, $k = 0, \dots, \kappa+j+1$, be the states generated by the input list $\alpha u(\alpha\phi)$; and similarly for the other relevant lists. Denote

$$Q(\alpha\phi)_k := (x(\alpha\phi)_k, s(\alpha\phi)_k, \mu(\alpha\phi)_k, y(\alpha\phi)_k, u(\alpha\phi)_{k+1}), \quad k = \kappa, \kappa+1, \dots, \kappa+j \tag{32}$$

as in (28). Then, the next statement is a reformulation of Theorem 1.

Corollary 3: *In the notation of Theorem 1, the bounded target tail set $\chi(c_1, \dots, c_m)$ is compatible with the initial input set $\mathscr{J}(\kappa)$ for Σ if and only if there is a member $\phi \in \Phi(\theta+\upsilon)$ that satisfies the following. For each element $\alpha \in \mathscr{J}(\kappa)$, there is an integer $\gamma > 0$ and a cycle $c_i \in \{c_1, \dots, c_m\}$ such that*

(a) $Q(\alpha\phi)_{\kappa+\theta+\upsilon-\gamma\tau_i} = Q(\alpha\phi)_{\kappa+\theta+\upsilon}$, *where* $\theta+\upsilon-\gamma\tau_i \geqslant 0$ *and* $Q(\alpha\phi)$ *is given by (32); and*

(b) the list $(y(\alpha\phi))_{\kappa+\theta+\upsilon-\gamma\tau_i}^{\kappa+\theta+\upsilon}$ *consists of* γ *of cycles* c_i.

Proof: A slight reflection upon the definition of the family of functions $\Phi(\theta+\upsilon)$ shows that the following is true. An ensemble of lists $\{u(\alpha) \in (D)_{\kappa+1}^{\kappa+\theta+\upsilon+1}, \alpha \in \mathscr{J}(\kappa)\}$ satisfies condition (b) of Theorem 1 if and only if there is a member of $\phi \in \Phi(\theta+\upsilon)$ such that

$$u(\alpha) = u(\alpha\phi) \tag{33}$$

for all $\alpha \in \mathscr{J}(\kappa)$. Combining this with the fact that conditions (a) and (b) of Corollary 3 are just a restatement of condition (a) of Theorem 1, the assertion follows. □

Furthermore, (33) shows that Lemma 3 takes the following form.

Corollary 4: *Assume that conditions (a) and (b) of Corollary 3 are satisfied for the function $\phi = \phi_1\phi_2\dots\phi_{\theta+\upsilon+1} \in \Phi(\theta+\upsilon)$. Then, the response of the strictly causal autonomous controller C of Lemma 3 that steers Σ from $\mathscr{J}(\kappa)$ to $\chi(c_1, \dots, c_m)$ is given by*

$$C\{\alpha\} = u(\alpha\phi_1\phi_2\dots\phi_{\theta+\upsilon+1})(u(\alpha\phi_1\phi_2\dots\phi_{\theta+\upsilon+1}))_{\kappa+\theta+\upsilon+1-\gamma\tau_i}^{\kappa+\theta+\upsilon+1}\dots(u(\alpha\phi_1\phi_2\dots\phi_{\theta+\upsilon+1}))_{\kappa+\theta+\upsilon+1-\gamma\tau_i}^{\kappa+\theta+\upsilon+1}\dots$$

for all $\alpha \in \mathscr{J}(\kappa)$.

For future use, it will be convenient to remove the dependence on the cycle length τ_i from the statement of Corollary 4. This is done in the next Corollary, at the expense of considering somewhat longer lists of functions.

Corollary 5: *Assume that conditions (a) and (b) of Corollary 3 are satisfied. Then there are an integer $\delta > 0$ and a function $\varphi = \varphi_1\varphi_2\dots\varphi_{\theta+\upsilon+\delta+1} \in \Phi(\theta+\upsilon+\delta)$ such that the following hold for all $\alpha \in \mathscr{J}(\kappa)$.*

(a) The list $(y(\alpha\varphi))_{\theta+\upsilon}^{\theta+\upsilon+\delta}$ consists of an integer number of copies of one of the cycles $\{c_1, \dots, c_m\}$;

(b) $Q(\alpha\varphi)_{\theta+\upsilon} = Q(\alpha\varphi)_{\theta+\upsilon+\delta}$; *and*

(c) the controller C with the response

$$C\{\alpha\} = u(\alpha\varphi_1\varphi_2\dots\varphi_{\theta+\upsilon+\delta+1})(u(\alpha\varphi_1\varphi_2\dots\varphi_{\theta+\upsilon+\delta+1}))_{\theta+\upsilon+1}^{\theta+\upsilon+\delta+1}\dots(u(\alpha\varphi_1\varphi_2\dots\varphi_{\theta+\upsilon+\delta+1}))_{\theta+\upsilon+1}^{\theta+\upsilon+\delta+1}\dots$$

represents a strictly causal controller that steers Σ from the initial input set $\mathscr{J}(\kappa)$ to the target tail set $\chi(c_1, \dots, c_m)$.

The Corollary follows from Corollaries 3 and 4; for each element $\alpha \in \mathscr{J}(\kappa)$, set $\delta(\alpha) := \gamma\tau_i$, where γ and τ_i are from Corollary 3 part (a), and take $\delta > 0$ to be a least common multiple of all $\delta(\alpha)$, $\alpha \in \mathscr{J}(\kappa)$. The result follows then similarly to Corollary 4 by considering longer portions of the sequences. We omit the details here.

The expression for the controller C in Corollary 5 part (c) still refers to the (possibly unknown) initial input list α, and whence is not in implementable form. This reference, however, can now be eliminated by using the factorization (31), and the following statement is obtained.

Theorem 2: *Assume that the conditions of Corollary 5 are satisfied for the function $\varphi := \varphi_1\dots\varphi_{\theta+\upsilon+\delta+1} \in \Phi(\theta+\upsilon+\delta)$. Using (31), factor $\varphi_i = \psi_{i+\kappa}\sigma_{\kappa+i-1}, i = 1, \dots, \theta+\upsilon+\delta+1$.*

Then, an autonomous strictly causal controller $C:(\mathrm{Im}\,\Sigma)_\kappa^\infty \to (D)_{\kappa+1}^\infty$ *that steers the bounded interpreter* Σ *from the initial input set* $\mathscr{J}(\kappa)$ *to the target tail set* $\chi(c_1, \ldots, c_m)$ *is given by the concatenation*

$$C\mu = v_{\kappa+1} v_{\kappa+2} \ldots v_{\theta+\upsilon+\delta+1} v_{\theta+\upsilon+1} \ldots v_{\theta+\upsilon+\delta+1} v_{\theta+\upsilon+1} \ldots v_{\theta+\upsilon+\delta+1} \ldots$$

where $v_j := \psi_j(\mu_\kappa \mu_{\kappa+1} \ldots \mu_{j-1}), j = \kappa+1, \ldots, \theta+\upsilon+\delta+1$, *and where* μ *is the monitored sequence of the interpreter–controller combination.*

The controller C of Theorem 2 is a feedback controller up to the step $\theta+\upsilon+\delta+1$; thereafter it can be regarded as an open-loop controller, producing a periodic input sequence for Σ with the cycle $\{v_{\theta+\upsilon+1} \ldots v_{\theta+\upsilon+\delta+1}\}$ of previously generated values. The fact that the control algorithm turns into an open-loop algorithm at some point is not surprising, since once the periodic part of the ultimately periodic input sequence of Σ is reached, the continuation of the sequence becomes predictable, and does not need to be recomputed.

The controller of Theorem 2 can be implemented by using the functions $\{\psi_j\}$ to create the required feedback for the steps $\kappa+1$ to $\theta+\upsilon+\delta+1$, and thereafter by simply repeating periodically the appropriate part of the previously generated list. Thus, we have obtained an implementable autonomous controller C that steers a bounded interpreter Σ from a given initial input set to a desired bounded target tail set, whenever such a controller exists.

Remark 3: Characterization of all controllers: we comment that Theorem 2 can be used to characterize the set of all strictly causal autonomous controllers with a finite implementation that steer Σ from $\mathscr{J}(\kappa)$ to a specified bounded target tail set. The set of all such controllers is determined by following the path toward the derivation of Theorem 2, including along the way all members of the following two sets: the set of all controllers C_δ that satisfy the first part of Lemma 2; and the set of all relevant solutions ψ of the factorization (31).

Finally, we note that Theorem 2 provides a general bound on the necessary complexity of a corrective controller.

5. Compatibility of initial input sets

Let $\Sigma: D \to SS_A$ be a causal interpreter with a uniform domain, with the initial input set $\mathscr{J}(\kappa)$ and the target tail set T. In case the desired target tail set is not compatible with the given initial input set, it may still be possible to achieve compatibility by changing the initial input set. The initial input set can only be changed by collecting more accurate data about the history of the interpreter Σ. This has the effect of replacing the initial input set $\mathscr{J}(\kappa)$ by one of its subsets. It is therefore of interest to characterize the class $\mathscr{C}(\mathscr{J}(\kappa), T)$ of all subsets of $\mathscr{J}(\kappa)$ that are compatible with T, when used as initial input sets for Σ. The class $\mathscr{C}(\mathscr{J}(\kappa), T)$ indicates all the various ways in which the initial input data about Σ can be refined to achieve compatibility with the desired target tail set T. Recall that, by definition, an autonomous corrective controller that steers Σ to T exists if and only if the initial input set of Σ is compatible with T.

The present section deals with the determination of the class $\mathscr{C}(\mathscr{J}(\kappa), T)$. Compatibility is always with respect to the interpreter Σ and the target tail set T. We start with some elementary properties.

First, it is quite clear that if there is an autonomous controller C that steers $\mathcal{J}(\kappa)$ to T, then the same controller also steers every subset of $\mathcal{J}(\kappa)$ to T. This yields the following statement.

Lemma 4: *If $\mathcal{J}(\kappa)$ is compatible with T, then so is every subset of $\mathcal{J}(\kappa)$.*

In intuitive terms, a smaller initial input set embodies more accurate data about the history of the interpreter Σ. Lemma 4 simply states, as one would expect, that more accurate data do not hamper the prospects of corrective control. In particular Lemma 4 implies that every intersection of input sets that are compatible with T is also compatible with T. Note, however, that a union of input sets that are compatible with T is not necessarily compatible with T. Indeed, consider the case where $\mathcal{J}(\kappa)$ consists of two elements α_1, α_2, and let $U_1(\alpha_i)$ be the set of all input sequences $u \in (D)^{\infty}_{\kappa+1}$ satisfying $T(\Sigma\alpha_i u) \cap T \neq \emptyset$, i.e. the set of all continuations that lead from α_i to $T, i = 1, 2$. Assume that $U_1(\alpha_1)$ and $U_1(\alpha_2)$ are both non-empty, but that their intersection is empty, so that no single continuation can lead from both initial input lists to T. Finally, assume that the monitoring function h_{m} of Σ is a constant function. Clearly, in this case, the monitored sequence generated by Σ is always the same, and whence no controller can provide a continuation that leads to T, when there is an uncertainty as to whether α_1 or α_2 is the initial input list. Thus, though the initial input sets $\{\alpha_1\}$ and $\{\alpha_2\}$ are both compatible with T, their union $\{\alpha_1, \alpha_2\}$ is not.

However, the logical negation of Lemma 4 directly implies that a union of input sets that are incompatible with T is always incompatible with T. We formally state this fact below.

Lemma 5: *An initial input set $\mathcal{J}(\kappa)$ that contains a subset that is incompatible with T, is itself incompatible with T.*

In particular, the union of an initial input set that is compatible with T with one that is incompatible, is incompatible with T. Still, the intersection of two input sets that are incompatible with T is not necessarily incompatible with T.

Consider a bounded interpreter $\Sigma : D \to SS_A$ with the initial input set $\mathcal{J}(\kappa)$ and the bounded target tail set T, where $\mathcal{J}(\kappa)$ is incompatible with T. For every $\alpha \in \mathcal{J}(\kappa)$, let $U_1(\alpha)$ be the set of all input sequences $u \in (D)^{\infty}_{\kappa+1}$ satisfying $T(\Sigma\alpha u) \cap T \neq \emptyset$, i.e. the set of all continuations that lead from α to T. It is readily seen that the class $\mathscr{C}(\mathcal{J}(\kappa), T)$ is non-empty if and only if there is at least one list $\alpha \in \mathcal{J}(\kappa)$ for which $U_1(\alpha) \neq \emptyset$.

We regard the class $\mathscr{C}(\mathcal{J}(\kappa), T)$ as a partially ordered set (a poset), under the usual relation of set inclusion. The *meet* of two elements $c_1, c_2 \in \mathscr{C}(\mathcal{J}(\kappa), T)$ is given by their intersection $c_1 \cap c_2$, which, according to Lemma 4, always belongs to $\mathscr{C}(\mathcal{J}(\kappa), T)$. The *join* of c_1 and c_2 is given by their union $c_1 \cup c_2$, whenever it belongs to $\mathscr{C}(\mathcal{J}(\kappa), T)$. We refer to $\mathscr{C}(\mathcal{J}(\kappa), T)$ as the compatibility poset; the compatibility poset depends, of course, on Σ, as well as on $\mathcal{J}(\kappa)$ and T.

As noted earlier, the union of two initial input sets that are compatible with T, is not always compatible with T. Consequently, the poset $\mathscr{C}(\mathcal{J}(\kappa), T)$ may not contain the join of some of its members, and whence, in general, does not form a lattice. From our current perspective, the most important implication of this fact is that $\mathscr{C}(\mathcal{J}(\kappa), T)$ does not always contain a 'global maximum'. In other words, in general, there is no largest compatible initial input set contained within $\mathcal{J}(\kappa)$. Recall that a larger compatible initial input set means that a corrective controller can be built with less accurate information about the input history of Σ. Of course, one would like to use as little information as possible, to reduce measurement complexity.

A member $c \in \mathscr{C}(\mathscr{I}(\kappa), T)$ is a *local maximum* if it is not a strict subset of any other member of $\mathscr{C}(\mathscr{I}(\kappa), T)$. To provide an intuitive perspective, assume that $\mathscr{C}(\mathscr{I}(\kappa), T)$ is not empty, i.e. that there is at least one initial list within $\mathscr{I}(\kappa)$ that is compatible with the target tail set T. When the initial input set $\mathscr{I}(\kappa)$ is not itself compatible with T, it must be replaced by a member $\mathscr{I}_c(\kappa)$ of $\mathscr{C}(\mathscr{I}(\kappa), T)$ to facilitate corrective control. This is achieved by collecting additional data about the input history of Σ up to the step κ. Clearly, the larger the set $\mathscr{I}_c(\kappa)$ is, the less additional information needs to be gathered about the input history of Σ. Each local maximum of $\mathscr{C}(\mathscr{I}(\kappa), T)$ provides a compatible initial input set that is 'largest' in the sense of not being contained within any other compatible initial input set. The local maxima of $\mathscr{C}(\mathscr{I}(\kappa), T)$ characterize the various 'minimal' ways in which additional data about the initial input history of Σ can create compatibility with the target tail set T. Below, we develope a finite procedure that determines the entire compatibility poset $\mathscr{C}(\mathscr{I}(\kappa), T)$ for a bounded target tail set T.

Let $\Sigma = (\Sigma_o, \Sigma_m): D \to SS_A$ be a bounded interpreter with the initial input set $\mathscr{I}(\kappa)$ and the target tail set T. Recall that, due to boundedness, $\mathscr{I}(\kappa)$ is a finite set; Let $\alpha_1, \ldots, \alpha_n$ be the elements of $\mathscr{I}(\kappa)$. As before, for each list $\alpha_j \in \mathscr{I}(\kappa)$, let $U_1(\alpha_j)$ be the set of all input sequences $u \in (D)^{\infty}_{\kappa+1}$ satisfying $T(\Sigma\alpha u) \cap T \neq \varnothing$.

Next, for each pair of lists $\alpha_i, \alpha_j \in \mathscr{I}(\kappa), i \neq j$, let $U_2(\alpha_i, \alpha_j)$ be the set of all pairs of sequences (u, u'), where $u \in U_1(\alpha_i)$ and $u' \in U_1(\alpha_j)$, and the following holds.

$$(u)^{k+1}_{\kappa+1} = (u')^{k+1}_{\kappa+1}, \quad \text{whenever } (\Sigma_m \alpha_i u)^k_\kappa = (\Sigma_m \alpha_j u')^k_\kappa, \quad k = \kappa, \kappa+1, \ldots \tag{34}$$

The sets $U_2(\alpha_i, \alpha_j)$ play an important role in the present context. First, combining Lemmas 1 and 5 with (34), we obtain the following lemma.

Lemma 6: *If $U_2(\alpha_i, \alpha_j)$ is empty, then there is no member of $\mathscr{C}(\mathscr{I}(\kappa), T)$ that contains both initial input lists α_i and α_j.*

If $U_2(\alpha_i, \alpha_j)$ is non-empty, then the class $\{\alpha_i, \alpha_j\}$ belongs to $\mathscr{C}(\mathscr{I}(\kappa), T)$.

Using the sets $U_2(\alpha_i, \alpha_j), i,j = 1, \ldots, n, i \neq j$, we can construct the entire class $\mathscr{C}(\mathscr{I}(\kappa), T)$ in the following way. Let $\alpha_{i_1}, \ldots, \alpha_{i_p}$ be any p elements of $\mathscr{I}(\kappa), p = 3, \ldots, n$. Define the set $U_p(\alpha_{i_1}, \ldots, \alpha_{i_p})$ as the set of all p-tuples $(u_{i_1}, \ldots, u_{i_p})$ of sequences for which

$$(u_{i_j}, u_{i_k}) \in U_2(\alpha_{i_j}, \alpha_{i_k}), \quad \text{for all } j, k = 1, \ldots, p, j \neq k \tag{35}$$

Note that $U_p(\alpha_{i_1}, \ldots, \alpha_{i_p})$ is obtained from the sets $\{U_2(\alpha_{i_j}, \alpha_{i_k})\}$ via (35) by a screening process. The compatibility poset $\mathscr{C}(\mathscr{I}(\kappa), T)$ can now be characterized as follows.

Proposition 7: *For the initial input set $\mathscr{I}(\kappa) = \{\alpha_1, \ldots, \alpha_n\}$, the compatibility poset $\mathscr{C}(\mathscr{I}(\kappa), T)$ consists of all subsets $\{\alpha_{i_1}, \ldots, \alpha_{i_p}\} \subset \mathscr{I}(\kappa), p = 1, \ldots, \# \mathscr{I}(\kappa)$, for which $U_p(\alpha_{i_1}, \ldots, \alpha_{i_p}) \neq \varnothing$.*

Proof: Let $p \in \{1, \ldots, \# \mathscr{I}(\kappa)\}$ be an integer, and let $A(\kappa) = \{\alpha_{i_1}, \ldots, \alpha_{i_p}\} \subset \mathscr{I}(\kappa)$ be a subset. Assume first that $U_p(\alpha_{i_1}, \ldots, \alpha_{i_p}) \neq \varnothing$; we show that in such case, the initial input set $A(\kappa)$ is compatible with the target tail set T for Σ. To this end, let $(u_{i_1}, \ldots, u_{i_p})$ be any element of $U_p(\alpha_{i_1}, \ldots, \alpha_{i_p})$. Consider the function $F: A(\kappa) \to \bigcup_{\alpha \in A(\kappa)} U_1(\alpha): \alpha_{i_j} \mapsto F(\alpha_{i_j}) := u_{i_j}, j = 1, \ldots, p$. Then, (34) and (35) imply that condition (b) of Lemma 1 is satisfied for the initial input set $A(\kappa)$ with the present function F. This, by the same Lemma, entails that $A(\kappa)$ is compatible with the target tail set T for Σ.

Conversely, assume that the initial input set $A(\kappa)$ is compatible with the target tail set T for Σ. We show that then $U_p(\alpha_{i_1}, \ldots, \alpha_{i_p}) \neq \varnothing$. Indeed, since $A(\kappa)$ is compatible with T, part (b) of Lemma 1 is valid. Let F be the function of part (b) of Lemma 1, and set $(u_{i_1}, \dots, u_{i_p}) := (F(\alpha_{i_1}), \dots, F(\alpha_{i_p}))$. Then, by statement (*b*) of Lemma 1, condition (35) holds for $(u_{i_1}, \dots, u_{i_p})$, so $(u_{i_1}, \dots, u_{i_p}) \in U_p(\alpha_{i_1}, \dots, \alpha_{i_p})$, and $U_p(\alpha_{i_1}, \dots, \alpha_{i_p}) \neq \emptyset$. This concludes our proof. □

Of course, Proposition 7 cannot be directly used to find the members of the compatibility class $\mathscr{C}(\mathscr{J}(\kappa), T)$, since $U_p(\alpha_{i_1}, \dots, \alpha_{i_p})$ consists of infinite sequences. Notwithstanding, when the interpreter Σ and the target tail set T are both bounded, the proposition can be modified into a finite procedure that yields all members of the compatibility poset $\mathscr{C}(\mathscr{J}(\kappa), T)$, as discussed next.

Consider again the bounded interpreter $\Sigma = (\Sigma_o, \Sigma_m): D \to SS_A$ with the input value set V_{in}, the internal value set V_t, the state set X, the initial input set $\mathscr{J}(\kappa)$, and the bounded target tail set $T = \chi(c_1, \dots, c_m)$. Let τ_i be the length of the cycle $c_i, i = 1, \dots, m$; denote $\tau := \max\{\tau_1, \dots, \tau_m\}$; let θ and υ be the integers given by (27) for this value of τ, and let Q be given by (28). For each element $\alpha_j \in \mathscr{J}(\kappa)$, let $\mathscr{U}_1(\alpha_j)$ be the set of all lists $u \in (D)_{\kappa+1}^{\kappa+\theta+\upsilon+1}$ for which the following holds.

There is an integer $\gamma > 0$ and a cycle $c_i \in \{c_1, \dots, c_m\}$ such that $(Q(\alpha_j u))_{\kappa+\theta+\upsilon-\gamma\tau_i} = (Q(\alpha_j u))_{\kappa+\theta+\upsilon}$; the list $(y(\alpha u(\alpha)))_{\kappa+\theta+\upsilon-\gamma\tau_i}^{\kappa+\theta+\upsilon}$ consists of γ cycles c_i; and $\theta + \upsilon - \gamma\tau_i \geqslant 0$.

Note that the set $\mathscr{U}_1(\alpha_j)$ can be obtained by a finite screening process. Next, for each pair of elements $\alpha_i, \alpha_j \in \mathscr{J}(\kappa), i \neq j$, let $\mathscr{U}_2(\alpha_i, \alpha_j)$ be the set of all pairs of lists (u, u'), where $u \in \mathscr{U}_1(\alpha_i)$ and $u' \in \mathscr{U}_1(\alpha_j)$, and

$$(u)_{\kappa+1}^{k+1} = (u')_{\kappa+1}^{k+1} \quad \text{whenever } (\Sigma_m \alpha_i u)_\kappa^k = (\Sigma_m \alpha_j u')_\kappa^k, \quad k = \kappa, \kappa+1, \dots, \kappa+\theta+\upsilon \tag{36}$$

The class $\mathscr{U}_2(\alpha_i, \alpha_j)$ is obtained from the sets $\mathscr{U}_1(\alpha_i)$ and $\mathscr{U}_1(\alpha_j)$ by a screening process, which is finite since all involved sets are finite. Theorem 1 yields then the following finite version of Lemma 6.

Lemma 7: *Let T be a bounded target tail set for the bounded interpreter Σ, with the initial input set $\mathscr{J}(\kappa) = \{\alpha_1, \dots, \alpha_n\}$.*

(a) If $\mathscr{U}_2(\alpha_i, \alpha_j)$ is empty, then there is no member of $\mathscr{C}(\mathscr{J}(\kappa), T)$ that contains both initial input lists α_i and α_j.

(b) If $\mathscr{U}_2(\alpha_i, \alpha_j)$ is non-empty, then $\{\alpha_i, \alpha_j\}$ belongs to $\mathscr{C}(\mathscr{J}(\kappa), T)$.

Next, for every subset $\{\alpha_{i_1}, \dots, \alpha_{i_p}\} \subset \mathscr{J}(\kappa)$ of p elements, $p = 3, \dots, n$, define the set $\mathscr{U}_p(\alpha_{i_1}, \dots, \alpha_{i_p})$ as the collection of all lists $(u_{i_1}, \dots, u_{i_p})$ that satisfy

$$(u_{i_j}, u_{i_k}) \in \mathscr{U}_2(\alpha_{i_j}, \alpha_{i_k}), \quad \text{for all } j, k = 1, \dots, p, j \neq k \tag{37}$$

Note that the set $\mathscr{U}_p(\alpha_{i_1}, \dots, \alpha_{i_p})$ is obtained by a screening process over a finite number of candidates.

When Theorem 1 is combined with Proposition 7, we obtain the following statement, which yields a finite technique for the derivation of the compatibility poset $\mathscr{C}(\mathscr{J}(\kappa), T)$ of Σ.

Theorem 3: *Let T be a bounded target tail set for the bounded interpreter Σ, with the initial input set $\mathscr{J}(\kappa) = \{\alpha_1, \dots, \alpha_n\}$. The compatibility poset $\mathscr{C}(\mathscr{J}(\kappa), T)$ consists of all subsets $\{\alpha_{i_1}, \dots, \alpha_{i_p}\} \subset \mathscr{J}(\kappa), p = 1, \dots, \# \mathscr{J}(\kappa)$, for which $\mathscr{U}_p(\alpha_{i_1}, \dots, \alpha_{i_p}) \neq \emptyset$.*

Theorem 3 allows us to derive the compatibility poset $\mathscr{C}(\mathscr{J}(\kappa), T)$ for Σ through a finite procedure, as mentioned. Every member of $\mathscr{C}(\mathscr{J}(\kappa), T)$ forms an initial input set that is compatible with T for Σ, and every initial input set that is compatible with T for

Σ belongs to $\mathscr{C}(\mathscr{J}(\kappa), T)$. Once the compatibility poset is known, its local maxima can be found directly by partially ordering the members. A convenient compatible initial input set can be selected from $\mathscr{C}(\mathscr{J}(\kappa), T)$ (if one exists); a corrective controller that steers Σ from the selected initial input set to the target tail set T can then be constructed using Theorem 2.

An important issue that is outside the scope of the present paper is, of course, the physical implementation of corrective controllers. In a digitial circuit environment, corrective controllers can be directly implemented from their mathematical models using standard techniques and components. In biochemical systems, however, the transition from a mathematical model to an implementation remains, to a large extent, an important open issue. Appropriate tools and techniques that bridge the gap between a mathematical model of a controller and its actual biochemical implementation wait to be developed. In practically all other areas of engineering, tools that bridge the span between mathematical models and implementations are well established.

References

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J. D., 1989, *Molecular Biology of the Cell*, second edition (New York: Garland).

Arnold, A., and Nivat, M., 1980, Controlling behaviors of systems: some basic concepts and some applications. *Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science* (Berlin: Springer-Verlag).

Cho, H., and Marcus, S. I., 1989, Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations. *Mathematical Systems Theory*, **22**, 177–211.

Cieslak, R., Desclaux, C., Fawaz, A., and Varaiya, P., 1988, Supervisory control of discrete event processes with partial observations. *IEEE Transactions on Automatic Control*, **33**, 249–260.

Eilenberg, S., 1974, *Automata, Languages, and Machines* (New York: Academic Press).

Ginsburg, S., 1962, *An Introduction to Mathematical Machine Theory* (Reading, Massachusetts, U.S.A.: Addison-Wesley); 1966, *The Mathematical Theory of Context Free Languages* (New York: McGraw-Hill).

Hoare, C. A. R., 1976, *Communicating Sequential Processes* (New Jersey, U.S.A.: Prentice Hall).

Holloway, L. E., and Krogh, B. H., 1990, Synthesis of feedback control logic for a class of controlled Petri nets. *IEEE Transactions on Automatic Control*, **35**, 514–523.

IEEE Computer Systems Society (Conference publications), 1974, *Proceedings of the 1974 Conference on Biologically Motivated Automata Theory*, McLean, Virginia, U.S.A.

Kauffman, S. A., 1969, Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology* **22**, 437–467.

Kumar, R., Garg, V., and Marcus, S. I., 1992, On supervisory control of sequential behaviors. *IEEE Transactions on Automatic Control*, **37**, 1978–1985; 1993, Predicates and predicate transformers for supervisory control of discrete-event systems. *IEEE Transactions on Automatic Control*, **38**, 232–247.

Lin, F, and Wonham, W. M., 1988, On observability of discrete-event systems. *Information Science*, **44**, 173–198.

Lindenmayer, A., 1968, Mathematical models for cellular interactions in development, Parts I and II. *Journal of Theoretical Biology*, **18**, 280–315.

MacLane, S., and Birkhoff, G., 1979, *Algebra* (New York: Macmillan).

Milner, R., 1980, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science (Berlin: Springer-Verlag).

von Neumann, J., 1966, *The Theory of Self-reproducing Automata*, edited and completed by A. W. Burks, (Urbana, Illinois, U.S.A.: University of Illinois Press).

Ozveren, C. M., and Willsky, A. S., 1990, Observability of discrete-event dynamic systems. *IEEE Transactions on Automatic Control*, **35**, 797–807.

Ozveren, C. M., Willsky, A. S., and Antsaklis, P. J., 1991, Stability and stabilizability of discrete event systems. *Journal of the Association for Computing Machinery*, **38**, 730–752.

Prusinkiewicz, P., Lindenmayer, A., and Hanan, J., 1990 *The Algorithmic Beauty of Plants* (New York: Springer-Verlag).

Ramadge, P. J., 1989, Some tractable supervisory control problems for discrete event systems modeled by Buchi automata. *IEEE Transactions on Automatic Control*, **34**, 10–19.

Ramadge, P. J., and Wonham, W. M., 1987, Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, **25**, 206–230.

Rashevsky, N., 1948, *Mathematical Biophysics* (The University of Chicago Press).

Rozenberg, G., and Salomaa, A., 1975, *L Systems*, Lecture Notes in Computer Science, Vol. 15 (Berlin: Springer-Verlag).

Sreenivas, R. S., and Krogh, B. H., 1992, On Petri net models of infinite state supervisors, *IEEE Transactions on Automatic Control*, **37**, 274–277.

Sugita, M., 1963, Functional analysis of chemical systems in vivo using a logical circuit equivalent. II. The idea of a molecular automaton. *Journal of Theoretical Biology*, **4**, 179–189.

Thistle, J. G., and Wonham, W. M., 1988, On the synthesis of supervisors subject to ω-language specifications. *Proceedings of the 22nd Annual Conference on Information Sciences and Systems*, Princeton University, Princeton, New Jersey, U.S.A., pp. 440–444.

Tsitsiklis, J. N., 1989, On the control of discrete-event dynamical systems. *Mathematics of Control, Signals and Systems*, **2**, 95–107.

Vaz, A. F., and Wonham, W. M., 1986, On supervisor reduction in discrete-event systems. *International Journal of Control*, **44**, 475–491.