Theoretical Computer Science 411 (2010) 360–376

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

An axiom system for sequence-based specification

Lan Lin [a,*], Stacy J. Prowell [b], Jesse H. Poore [a]

[a] Department of Electrical Engineering and Computer Science, The University of Tennessee, 203 Claxton Complex, 1122 Volunteer Blvd., Knoxville, TN 37996, USA
[b] Software Engineering Institute, Carnegie Mellon University, 4500 Forbes Avenue, Pittsburgh, PA 15213, USA

ARTICLE INFO

Article history:
Received 29 January 2008
Received in revised form 18 May 2009
Accepted 26 June 2009
Communicated by P. Stevens

Keywords:
Software specification
Sequence-based specification
Axiom system
Representation
Automaton
Mealy machine

ABSTRACT

This paper establishes an axiomatic foundation and a representation theorem for the rigorous, constructive process, called sequence-based specification, of deriving precise specifications from ordinary (informal) statements of functional requirements. The representation theorem targets a special class of Mealy state machines, and algorithms are presented for converting from the set of sequences that define the specification to the equivalent Mealy machine, and vice versa. Since its inception, sequence-based specification has been effectively used in a variety of real applications, with gains reported in quality and productivity. This paper establishes the mathematical foundation independently of the process itself.

1. Introduction

Modern software development processes for safety critical systems require rigorous methods for code development and testing to support dependability assurance cases [16]. Although process and methods are not sufficient to guarantee dependability, the case for dependability must rest, in large part, on rigorous development methods. To be credible for dependability cases, such methods must have sound mathematical foundations to support application and interpretation of results, provide audit trails of evidence to support correctness arguments, have effective tool support, and produce specifications that lend themselves to validation and formal verification. *Sequence-based specification* is an instance of such a method, with the axiom system and representation theorem presented here.

Safety critical systems usually have requirements expressed in work products with varying degrees of formality including natural language, notations of science and engineering, diagrams, tables of values and references to standards, among other artifacts. The sequence-based specification method was developed to convert such informal statements of functional requirements to precise specifications through a rigorous and constructive process called *sequence enumeration*. These specifications then become the basis for both code development and testing.

Sequence-based specifications support automated testing, by which we mean the automatic generation, execution and evaluation of test cases. The sequence-based specifications, together with specially designed tools, have been the basis for several successful industrial applications of this form of automated testing [4].

Sequence-based specification, through the consideration of all sequences of use, explores an evolving behavioral description of a software system. Since its inception, it has been effectively used in a variety of real applications, ranging from automotive components to medical devices to scientific instrumentation. Gains in quality and productivity have been reported from projects carried out by the authors [25,26] and other collaborators in industry [4,5,15]. In particular, Broadfoot

* Corresponding author. Tel.: +1 865 9748733.
E-mail addresses: lin@eecs.utk.edu (L. Lin), sprowell@cmu.edu (S.J. Prowell), poore@eecs.utk.edu (J.H. Poore).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.06.041

and Broadfoot (Hopcroft) [5,15] have incorporated the method with elements of *Communicating Sequential Processes* (CSP) [13] and *Failures Divergence Refinement* (FDR) [27]. Enumeration of usage scenarios has proved to be practical in application and immensely powerful in eliciting errors and omissions in statements of requirements (even of mature systems). It has also been demonstrated to result in specifications that lend themselves to the generation of code, or at least code structure (see [26], part III for a detailed example).

Sequence-based specification emerged from the functional treatment of software as described by Mills [20–22]. The development was most directly influenced by the *trace-assertion method* of Parnas [3,23] and the algebraic treatment of regular expressions by Brzozowski [6]. The primary distinction of this method from its nearest neighbors (the trace-assertion method [7–10,17,23] and *software cost reduction* [12]) is that the state machine for the software system to be built is discovered through systematic enumeration of input sequences and recording sequence equivalences based on future behavior. In trace-assertion specifications, the a priori automaton is conceived by the specification writer and is introduced indirectly by a set of important traces called *canonical traces*, a trace equivalence relation, and a rewriting system that transforms any trace to its canonical representative. For this reason, trace-assertion specifications are *descriptive*; they describe the function of a software system, whose automaton is obtained by experience and insight. Sequence-based specifications are *constructive*; they elicit and discover the details of the automaton and the system's behavior through a constructive (enumeration) process.

Sequence-based specifications take the form of a set of strings that are equivalent to a certain class of Mealy machines. In this paper, we explore an axiomatic treatment of sequence-based specification and use the axiom system to prove a number of important properties about sequence-based specifications. The need for an axiomatic approach first arose from our interest in looking at an enumeration solely as a mathematical object. We want to provide a list of axioms and show that, if a set of sequences satisfy these axioms, then it serves as a specification regardless of whether the sequences were obtained by the enumeration process or in some other way. The results go beyond, and extend to, a theoretical framework, in which concepts of sequence-based specification are defined formally, connections to the *black box functions* of Mills [22] are made, a precise relationship between enumerations and state machines is established, and an algorithm for enumeration minimization is developed. The axiom system provided the basis for tools to assist in managing requirements changes and their implications on specifications and design.

The axiomatic foundation for the constructive process of sequence-based specification might contribute to dependability assurance cases for software systems designed by use of the method. The axiom system improves our ability to prove assertions and has shown itself to be a practical tool in a number of activities stated below.

Checking enumerations. We have developed a prototype tool called Proto_Seq [1] to assist with sequence-based specification. Our experience with field applications shows that sometimes the practitioner "hacks" the sequences in the specification file (in XML format) to make changes instead of redoing the work in the tool. In such cases, it is critical to know whether the hacked sequences are correct by the enumeration rules as enforced explicitly by the normal workflow under tool control. By expressing these rules in a list of axioms, we provide the basis for checking a set of sequences to see if it is a correct formal specification, regardless of how the set was obtained.

Systematic transformation to state machines. A state machine can be generated from a sequence-based specification. In this paper we establish a representation theorem for enumerations in terms of a special subset of Mealy machines (called *enumeration Mealy machines*), and present algorithms to convert them from one form to the other.

Deriving requirements change algorithms. Change management is one of the most complex and difficult problems to deal with in requirements engineering. Using the results presented in this paper, we have developed a theory that exactly prescribes the impact of requirements changes on specifications and state machines [19]. The representation theorem, established here, provides a precise connection between enumerations and their corresponding Mealy machines, and hints at utilizing state machines to model and manage requirements changes and specification changes. The axiom system was indispensable in formulating all our change algorithms mathematically and proving their correctness. Following the theoretical framework presented in this paper, we are able to provide strong evidence about the correctness and completeness of our proposed change management theory.

Applying string rewriting techniques. In the course of constructing a sequence-based specification, various patterns might be observed. For example, some pairs of inputs commute with respect to sequencing; some inputs are idempotent. More complex patterns may involve three or more inputs. These ad hoc observations lead to the systematic study of applying string rewriting techniques to sequence-based specification, which we are currently working on with promising results. The axiom system presented in this paper provides a theoretical framework to integrate the string rewriting theory to enhance the enumeration process. The end product will be an enumeration in which a significant number of decisions are made automatically and consistently throughout.

Comparisons with other specification methods. The axiom system enables looking at specifications using various rigorous representations, including finite state machines, regular expressions, and prefix-recursive functions, hence the rigorous comparisons with other mathematically described specification methods. We have also developed a complete collection of algorithms for moving among enumerations, state machines, regular expression sets, and prefix-recursive function sets [18], which provides the potential to import results from theories that are well-established.

Model checking. Our work concerns the tool-assisted, systematic transformation of informal requirements to precise specifications to state machines. Since the enumeration is constructed in a rigorous manner, the state machine is discovered in a constructive process that assures completeness and traceable correctness. With the axiom system and

Table 1
Safe controller requirements.

Tag	Requirement
1	The combination consists of three digits (0–9) which must be entered in the correct order to unlock the safe. The combination is fixed in firmware.
2	Following an incorrect combination entry, a "clear" key must be pressed before the safe will accept further entry. The clear key also resets any combination entry.
3	Once the three digits of the combination are entered in the correct order, the safe unlocks and the door may be opened.
4	When the door is closed, the safe automatically locks.
5	The safe has a sensor which reports the status of the lock.
6	The safe ignores keypad entry when the door is open.
7	There is no external confirmation for combination entry other than unlocking the door.
8	It is assumed (with risk) that the safe cannot be opened by means other than combination entry while the software is running.
D1	Sequences with stimuli prior to system initialization are illegal by system definition.
D2	Re-initialization (power-on) makes previous history irrelevant.

the transformation theory built on it, it is not necessary to use model checking for properties such as completeness and determinism, nor for certain verification steps such as the correct refinement from black boxes to state boxes and the establishment of their behavioral equivalence [15].

The remainder of the paper is organized as follows. Section 2 introduces sequence-based specification with a simple example of a safe controller. Section 3 gives our terminologies and notations. Section 4 presents the axiom system. Since we remain interested in the box structure representations of Mills [22], we discuss the relationship of enumerations to the black box functions of [24]. Examples are developed on the basis of the safe controller in Section 2 to illustrate the theoretical details. Section 5 presents our transformation theory relevant to the representation of enumerations as Mealy machines. A representation theorem is established. An enumeration minimization algorithm is presented that connects to the classical literature on Mealy machine minimization. Section 6 concludes the paper with directions of the continuing work.

2. Background: Sequence-based specification

The first step in applying sequence-based specification is to identify a *system boundary* that defines what is inside and what is outside the software system to be developed. It consists of *interfaces* between the system and the external entities with which the system directly communicates. These entities compose the software's *environment*. Events (inputs, interrupts, invocations) in the environment that can affect system behavior are called *stimuli*. Observable system behaviors are called *responses*. Consider all finite sequences of stimuli. They represent all possible scenarios of system use.

Sequence-based specification facilitates deriving the system's behavior solely in terms of external stimuli and responses, through a process called sequence enumeration. As the name suggests, it is the literal enumeration of sequences of stimuli in increasing order of length, and within the same length in any arbitrary strict total order, and the assignment of correct responses to each enumerated sequence. Because some sequences may generate no externally observable behavior across the system boundary, we introduce a *null* response denoted by 0 and map such sequences to 0. We also introduce an *illegal* response denoted by ω to which we map all sequences that are not physically realizable. A sequence is *illegal* when it maps to ω; otherwise, it is *legal*.

For any enumerated sequence, besides mapping it to a response, we also check whether or not it can be equated to a previously enumerated sequence in the following sense: their responses to future stimuli will be identical according to the requirements. Such sequences are called *Mealy equivalent*, as they represent the same system state when the system is modeled as a Mealy machine. If a sequence is not found Mealy equivalent to any prior sequence, it is *unreduced*; otherwise, it is *reduced* to an unreduced (Mealy equivalent) prior sequence.

Enumeration starts with the empty sequence (denoted by λ), which is mapped to 0 and unreduced. To form all sequences of length $n+1$ (n is a non-negative integer), we extend all sequences of length n by every stimulus. However, not all of them need to be considered for two reasons:

- If a sequence is illegal, its extensions need not be enumerated as they must all map to ω.
- If a sequence is reduced to a prior sequence, its extensions need not be enumerated as they must exhibit the same behavior as the same extensions of the prior sequence.

Therefore, we only extend both legal and unreduced sequences for the next enumeration length. When there are no more sequences to extend, the enumeration is *complete*.

The following example of a safe controller is from [24]. We will use it as a running example throughout the paper to illustrate the theories as we develop them. In this section, we show a complete enumeration of the safe obtained by following the process described above.

The requirements for the safe controller are given in Table 1. They are tagged for traceability and to ease the work of verifying the correctness of every decision made in the specification process. The last two rows in the table record two derived requirements (D1 and D2) discovered during sequence enumeration.

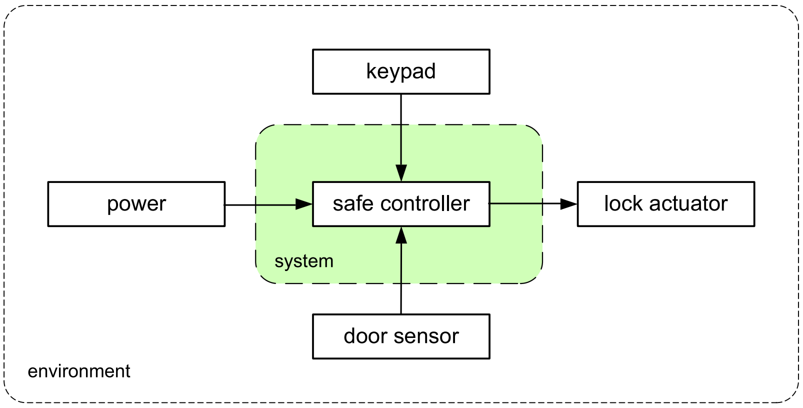

Fig. 1. Safe controller system boundary.

Table 2
Safe controller stimuli.

Stimulus	Description	Interface
B	Bad combination entry	Keypad
C	Clear key press	Keypad
D	Door closed	Door sensor
G	Good combination entry	Keypad
L	Power on with door locked	Power, door sensor
U	Power on with door unlocked	Power, door sensor

Table 3
Safe controller responses.

Response	Description	Interface
lock	Locking the door	Lock actuator
unlock	Unlocking the door	Lock actuator

The system boundary is cut between the system and a list of interfaces in the environment representing the external power, keypad, door sensor, and lock actuator. We diagram the system boundary in Fig. 1, and identify all stimuli and responses from the list of interfaces in Tables 2 and 3. To make work efficient, we discard treating the distinct digit presses (0–9) separately and, instead, use *G* to denote entering the correct three digits in order and *B* to denote entering an incorrect three-digit combination entry (any digit could be wrong). Note that *L* (power on with door locked) and *U* (power on with door unlocked) are two different stimuli, because the power and the door sensor are both outside the system boundary, and we are combining the power on stimulus (received from the power interface) and the door status stimulus (received from the door sensor interface) into *L* and *U*.

A complete enumeration is developed in Table 4, which can be read as follows. First, the empty sequence represents the initial system state (no input has been received). It is mapped to 0 and unreduced, as enforced by the rules of the method. Then it is extended by every stimulus to get all to-be-enumerated sequences of length one (the sequences *B* through *U*).

Among the length-one sequences, only two of them, namely *L* and *U*, are legal, as any event prior to the power-on event cannot be perceived by software; thus, the derived requirement *D*1. The sequences *L* and *U* represent two distinct states of the system other than the initial state (power is turned on but the door can be either locked or unlocked). They both map to 0 as there is no externally observable behavior when power is turned on, and they get extended by every stimulus to form all to-be-enumerated sequences of length two (the sequences *LB* through *UU*).

The length-two sequences are considered in lexicographical order (the order they show up in the table) for the generated responses and possible reductions to prior sequences. For instance, the sequence *LG* represents the usage scenario where power is first turned on with the door of the safe being locked followed by a good combination entry. This will unlock the door and, hence, the response of *LG* is mapped to *unlock*. The sequence *LG* gets the system to the same state as the prior sequence *U* as, in both cases, the system has power and the door is unlocked. The decisions regarding the response and the equivalence of *LG* are traced to the requirements 1, 3 and 7.

Similarly, the only legal and unreduced sequence of length two, the sequence *LB*, is extended to get all to-be-enumerated sequences of length three. There are six of them (the sequences *LBB* through *LBU*) and it turns out that all of them are either illegal or reduced to prior sequences, therefore, the enumeration terminates at length three.

The process is constructive and considers every scenario of use for correct response and equivalence according to the requirements as originally presented and corrected, refined or extended.

Table 4
Safe controller sequence enumeration.

Sequence	Response	Equivalence	Trace
λ	0		Method
B	ω		D1
C	ω	B	D1
D	ω	B	D1
G	ω	B	D1
L	0		5
U	0		5
LB	0		1,2,7
LC	0	L	2,7
LD	ω	B	8
LG	*unlock*	U	1,3,7
LL	0	L	5,D2
LU	0	U	5,D2
UB	0	U	6
UC	0	U	6
UD	*lock*	L	4
UG	0	U	6
UL	0	L	5,D2
UU	0	U	5,D2
LBB	0	LB	2,7
LBC	0	L	2,7
LBD	ω	B	8
LBG	0	LB	2,7
LBL	0	L	5,D2
LBU	0	U	5,D2

3. Terminology and notation

Let $\mathbb{Z}$ and $\mathbb{N}$ denote the integers and positive integers, respectively.

The length of a string w is denoted by $|w|$. The empty string is denoted by λ.

If X is a fixed (finite) alphabet, then X^* and X^+ denote the *Kleene closure* and the *positive closure* of X, respectively.

Unless stated explicitly as *partial*, a function $f : X \to Y$ is *total* (or *complete*). When f is partial, *domf* denotes the set of elements on which f is defined.

Consider partial functions of the form $f : X \to Y \times Z$. If $f(u) = (r, v)$, we write $u \underset{f}{\mapsto} r$ and $u \underset{f}{\rhd} v$. The f will be dropped where it is clear from context. Alternatively they are sometimes written as $\mapsto (u) = r$ and $\rhd(u) = v$. Define $u \not\mapsto r$ by $u \not\mapsto r \leftrightarrow \exists r' \neq r.\, u \mapsto r'$.

A set of strings is *prefix-closed* if every string in the set has all its prefixes also in the set.

4. Axiomatic sequence-based specification

4.1. Enumeration

We first define an enumeration as a mathematical object. Here S denotes a stimulus set that is non-empty and finite. R denotes a response set that includes the two special responses, 0 and ω, and at least one other (observable) response. $\prec$ denotes a strict total order among all stimulus sequences such that any shorter sequence must relate to any longer sequence by $\prec$; it represents an order in which stimulus sequences will be enumerated. An enumeration is defined as a partial function that maps certain stimulus sequences to (response, stimulus sequence) pairs. The domain of this partial function comprises all enumerated sequences. The enumeration function, when projected to the first component in the pair, defines a response mapping ($\mapsto$) for enumerated sequences. When projected to the second component, it defines a reduction function ($\rhd$) among enumerated sequences. The rules that guide the enumeration process are characterized in six axioms.

Definition 4.1 (*Enumeration*). Let S be a non-empty alphabet and R be a set that properly contains $\{0, \omega\}$. Let $\prec$ be a strict total order on S^* such that for all u, v in S^*, $|u| < |v|$ implies $u \prec v$. A partial function $\mathcal{E} : S^* \to R \times S^*$ is an *enumeration* iff the following hold for all x in S:

Axiom 1. $\lambda \mapsto 0$;
Axiom 2. $u \rhd v$ implies $v \prec u$ or $v = u$;
Axiom 3. $u \rhd v$ implies $v \rhd v$;
Axiom 4. $ux \in dom\,\mathcal{E}$ implies $u \rhd u$;
Axiom 5. $ux \in dom\,\mathcal{E}$ implies $u \not\mapsto \omega$;
Axiom 6. $u \rhd v$, $u \mapsto \omega$, $v \not\mapsto \omega$, $vx \in dom\,\mathcal{E}$ imply $vx \mapsto \omega$.

The paraphrasing below is an intuitive statement of the enumeration axioms. It uses ideas that are informally described in Section 2. Their precise definitions will be given shortly.

Axiom 1. The empty sequence is mapped to 0.
Axiom 2. A sequence can only be reduced to a prior sequence (according to $\prec$).
Axiom 3. A sequence can only be reduced to an unreduced sequence.
Axiom 4. A reduced sequence is not extended.
Axiom 5. An illegal sequence is not extended.
Axiom 6. An illegal sequence cannot be reduced to a legal sequence if the legal sequence when extended by a stimulus maps to a legal response (because an illegal sequence when extended must remain illegal, the illegal sequence and the legal sequence in question cannot be Mealy equivalent).

An enumeration becomes complete or finite when it satisfies additional axioms.

Definition 4.2 (*Complete Enumeration*). An enumeration $\mathcal{E} : S^* \to R \times S^*$ is *complete* iff the following holds for all x in S:

Axiom 7. $u \not\mapsto \omega, u \rhd u$ imply $ux \in dom\, \mathcal{E}$.

Definition 4.3 (*Finite Enumeration*). An enumeration $\mathcal{E} : S^* \to R \times S^*$ is *finite* iff

Axiom 8. $|R| \in \mathbb{N}, |dom\, \mathcal{E}| \in \mathbb{N}$.

Informally, we paraphrase the two axioms as follows.

Axiom 7. In a complete enumeration every legal and unreduced sequence is extended by every stimulus.
Axiom 8. In a finite enumeration the response set is finite and there are finitely many enumerated sequences.

To illustrate, we define in function form the safe enumeration developed in Section 2. Since the reduction function $\rhd$ requires a mapped value for every enumerated sequence, we treat each unreduced sequence as "being reduced to itself".

Example 4.4. Let $\mathcal{E} : S^* \to R \times S^*$ be a partial function, where $S = \{B, C, D, G, L, U\}$, $R = \{lock, unlock, 0, \omega\}$, S^* has a strict total order $\prec$ defined by

$$\begin{array}{l}
\forall u, v \in S^*.\, |u| < |v| \to u \prec v \\
\forall u \in S^*.\, uB \prec uC \prec uD \prec uG \prec uL \prec uU \\
\forall u, v \in S^*.\, \forall x, y \in S.\, u \prec v \to ux \prec vy,
\end{array}$$

and

$$\begin{array}{llll}
\mathcal{E}(\lambda) = (0, \lambda) & \mathcal{E}(B) = (\omega, B) & \mathcal{E}(C) = (\omega, B) & \mathcal{E}(D) = (\omega, B) \\
\mathcal{E}(G) = (\omega, B) & \mathcal{E}(L) = (0, L) & \mathcal{E}(U) = (0, U) & \mathcal{E}(LB) = (0, LB) \\
\mathcal{E}(LC) = (0, L) & \mathcal{E}(LD) = (\omega, B) & \mathcal{E}(LG) = (unlock, U) & \mathcal{E}(LL) = (0, L) \\
\mathcal{E}(LU) = (0, U) & \mathcal{E}(UB) = (0, U) & \mathcal{E}(UC) = (0, U) & \mathcal{E}(UD) = (lock, L) \\
\mathcal{E}(UG) = (0, U) & \mathcal{E}(UL) = (0, L) & \mathcal{E}(UU) = (0, U) & \mathcal{E}(LBB) = (0, LB) \\
\mathcal{E}(LBC) = (0, L) & \mathcal{E}(LBD) = (\omega, B) & \mathcal{E}(LBG) = (0, LB) & \mathcal{E}(LBL) = (0, L) \\
\mathcal{E}(LBU) = (0, U). & & &
\end{array}$$

It is easily checked that $\mathcal{E}$ as defined satisfies Axioms 1–8 for a complete and finite enumeration.

An enumerated sequence can be classified as legal or illegal, reduced or unreduced.

Definition 4.5 (*Illegal/legal Sequence*). Let $\mathcal{E} : S^* \to R \times S^*$ be an enumeration. Then u is an *illegal* sequence in $\mathcal{E}$ iff $u \mapsto \omega$, and u is a *legal* sequence in $\mathcal{E}$ iff $u \not\mapsto \omega$.

Definition 4.6 (*Unreduced/reduced Sequence*). Let $\mathcal{E} : S^* \to R \times S^*$ be an enumeration. Then u is an *unreduced* sequence in $\mathcal{E}$ iff $u \rhd u$, and u is a *reduced* sequence in $\mathcal{E}$ iff $u \rhd v$ for $v \prec u$. Let $U_{\mathcal{E}} = \{u : u \rhd u\}$.

$U_{\mathcal{E}}$ denotes the set of all unreduced sequences in the enumeration $\mathcal{E}$.

Alternatively, we call a sequence that is both legal and unreduced an extensible sequence. $E_{\mathcal{E}}$ denotes the set of all extensible sequences in the enumeration $\mathcal{E}$.

Definition 4.7 (*Extensible Sequence*). Let $\mathcal{E} : S^* \to R \times S^*$ be an enumeration. Then u is an *extensible* sequence in $\mathcal{E}$ iff u is legal and unreduced in $\mathcal{E}$. Let $E_{\mathcal{E}} = \{u : u \not\mapsto \omega, u \rhd u\}$.

Note that both $U_{\mathcal{E}}$ and $E_{\mathcal{E}}$ are prefix-closed sets by Axioms 4 and 5.

Example 4.8. Given the enumeration $\mathcal{E}$ defined in Example 4.4, $U_{\mathcal{E}} = \{\lambda, B, L, U, LB\}$, $E_{\mathcal{E}} = \{\lambda, L, U, LB\}$.

In the enumeration process, one might miss identifying the equivalences between enumerated sequences and extend sequences that should have been reduced (to prior sequences). These are called "missed equivalences" [24]. As it pointed out, missed equivalences will not make an enumeration wrong; they only introduce redundant states into the system, and the specification writer ends up doing more work than necessary.

Given a complete enumeration, unreduced sequences can be checked against each other to see if their responses to future stimuli will always agree, as implied by the completed enumeration. If so, such unreduced sequences are indistinguishable, in the sense that they represent redundant system states, and missed equivalences can be identified among them. The indistinguishability relation ($\sim$) is defined on its complement relation, the relation of distinguishability ($\not\sim$) among all unreduced sequences.

Definition 4.9 (*Distinguishability/Indistinguishability*). Let $\mathcal{E} : S^* \to R \times S^*$ be a complete enumeration. The relation $\not\sim \subset U_{\mathcal{E}} \times U_{\mathcal{E}}$ is the smallest relation satisfying the following rules:

Rule 1. $(a \mapsto \omega, \exists x \in S.\, bx \not\mapsto \omega) \to a \not\sim b$;
Rule 2. $(\exists x \in S.\, \mapsto (ax) \neq \mapsto (bx)) \to a \not\sim b$;
Rule 3. $(a \not\sim b, \exists x \in S.\, (cx \rhd a, dx \rhd b)) \to c \not\sim d$.

The relation $\sim \subset U_{\mathcal{E}} \times U_{\mathcal{E}}$ is defined by $\forall a, b \in U_{\mathcal{E}}.\, a \sim b \leftrightarrow (a, b) \notin \not\sim$.

Two unreduced sequences in a complete enumeration are distinguishable if their responses to future stimuli could be different. This happens in the following situations:

- One is illegal; the other is legal. The extension of the legal sequence by one arbitrary stimulus remains legal. (Rule 1)
- Both are legal. Their extensions by one arbitrary but the same stimulus map to different responses. (Rule 2)
- Both are legal. Their extensions by one arbitrary but the same stimulus reduce to two already distinguishable sequences. (Rule 3)

If an unreduced sequence is distinguishable from all its prior unreduced sequences, it is called canonical. A canonical sequence is an unreduced sequence with no missed equivalences.

Definition 4.10 (*Canonical Sequence*). Let $\mathcal{E} : S^* \to R \times S^*$ be a complete enumeration. Then u is a *canonical* sequence in $\mathcal{E}$ iff $u \in U_{\mathcal{E}}$ and $\forall v \in U_{\mathcal{E}}.\, u \sim v \to u \prec v$ or $u = v$.

A complete enumeration is minimal when it satisfies an additional axiom.

Definition 4.11 (*Minimal Enumeration*). A complete enumeration $\mathcal{E} : S^* \to R \times S^*$ is *minimal* iff

Axiom 9. $u \rhd u, v \rhd v, u \neq v$ imply $u \not\sim v$.

Informally,

Axiom 9. A complete enumeration is minimal when all its unreduced sequences are pairwise distinguishable.

Example 4.12. Given the enumeration $\mathcal{E}$ defined in Example 4.4, by Definition 4.9 we have

$\lambda \not\sim B$	as	$B \mapsto \omega, L \mapsto 0(Rule1)$	$B \not\sim L$	as	$B \mapsto \omega, LB \mapsto 0(Rule1)$
$\lambda \not\sim L$	as	$G \mapsto \omega, LG \mapsto unlock(Rule2)$	$B \not\sim U$	as	$B \mapsto \omega, UB \mapsto 0(Rule1)$
$\lambda \not\sim U$	as	$D \mapsto \omega, UD \mapsto lock(Rule2)$	$B \not\sim LB$	as	$B \mapsto \omega, LBB \mapsto 0(Rule1)$
$\lambda \not\sim LB$	as	$B \mapsto \omega, LBB \mapsto 0(Rule2)$	$L \not\sim U$	as	$LG \mapsto unlock, UG \mapsto 0(Rule2)$
$U \not\sim LB$	as	$UD \mapsto lock, LBD \mapsto \omega(Rule2)$	$L \not\sim LB$	as	$LG \mapsto unlock, LBG \mapsto 0(Rule2)$.

Since elements in $U_{\mathcal{E}}$ are pairwise distinguishable, they are all canonical sequences. $\mathcal{E}$ satisfies Axiom 9 as well for a complete, finite, and minimal enumeration.

The nine axioms have captured different aspects of an enumeration. However, it is necessary to show that these nine axioms are consistent, none contradicts another; that they are independent, none implies another. These properties are established by the following lemma.

Lemma 4.13.

i. *Axioms 1–9 for enumerations are consistent;*
ii. *Axioms 1–8 for enumerations are independent.*

Proof. For (i) we show an example enumeration satisfying all the nine axioms. Let the partial function $\mathcal{E} : S^* \to R \times S^*$ be defined as follows for $S = \{a\}, R = \{0, \omega, r\}$: $\mathcal{E}(\lambda) = (0, \lambda)$, $\mathcal{E}(a) = (r, \lambda)$. S^* is equipped with the strict total order $\prec$ defined by $a^n \prec a^{n+1}$ $(n \in \mathbb{Z}, n \geq 0)$.

For (ii) it suffices to show Axiom i cannot be derived from the rest of the axioms, where $1 \leq i \leq 8$. We exhibit an example partial function that satisfies Axioms 1–8 except Axiom i for each i. □

Although Axioms 1–8 do not imply Axiom 9 (we can exhibit another example that satisfies Axioms 1–8 but not Axiom 9), Axiom 9 entails Axioms 1–7 in the sense that Axioms 1–7 are a necessary accompaniment of Axiom 9 rather than a direct logical implication, as $\not\sim$ is only defined for a complete enumeration.

The next two lemmas characterize all the enumerated sequences in an enumeration.

Lemma 4.14. *Let $\mathcal{E} : S^* \to R \times S^*$ be an enumeration. Then $u \in dom\,\mathcal{E}$ implies one of the following must hold:*

i. *u is reduced;*
ii. *u is unreduced and $u \mapsto \omega$;*
iii. *u is extensible.*

Proof. A direct consequence of Axiom 2. $\square$

Lemma 4.15. *Let $\mathcal{E} : S^* \to R \times S^*$ be an enumeration. Then $u \in dom\,\mathcal{E}$ implies $u = \lambda$ or $u = u'x$, where $x \in S$ and u' is an extensible sequence.*

Proof. A direct consequence of Axioms 1, 4 and 5. $\square$

In a complete and minimal enumeration, every unreduced sequence must be canonical, as proved below.

Lemma 4.16. *Let $\mathcal{E} : S^* \to R \times S^*$ be a complete and minimal enumeration. Then $u \in U_{\mathcal{E}}$ implies u is canonical.*

Proof. Suppose the implication is not true, and we have $u \in U_{\mathcal{E}}$ but u is not canonical. Then there must exist v in $U_{\mathcal{E}}$ such that $u \sim v$ and it is not the case that either $u \prec v$ or $u = v$, hence $v \neq u$. However, $u \in U_{\mathcal{E}}$, $v \in U_{\mathcal{E}}$, $v \neq u$ imply $u \not\sim v$, since $\mathcal{E}$ is a minimal enumeration, a contradiction. $\square$

4.2. Black box function

Following [21,22] a black box function maps every stimulus sequence to a response. The empty sequence is mapped to the null response.

Definition 4.17 (*Black Box Function*). Let S be a non-empty alphabet and R be a set that properly contains $\{0, \omega\}$. A *black box function* is a total function $BB : S^* \to R$ with $BB(\lambda) = 0$.

With a complete enumeration, we have enough information to deduce the mapped response for any stimulus sequence, whether or not it is actually enumerated. We first define an extension of the two functions, $\mapsto$ and $\triangleright$, algebraically, then prove the extended response mapping $\hat{\mapsto}$ is a black box function of particular interest.

Definition 4.18. Let $\mathcal{E} : S^* \to R \times S^*$ be a complete enumeration. We extend $\mapsto$ and $\triangleright$ to total functions on S^* as follows:

$$\begin{array}{ll}
\hat{\triangleright} : S^* \to dom\,\mathcal{E} & \hat{\mapsto} : S^* \to R \\
\hat{\triangleright}(\lambda) = \lambda & \hat{\mapsto}(\lambda) = 0 \\
\forall u \in S^*. \forall x \in S. & \forall u \in S^*. \forall x \in S. \\
\hat{\triangleright}(ux) = \begin{cases} \hat{\triangleright}(u) & \text{if } \hat{\triangleright}(u) \mapsto \omega \\ \triangleright(\hat{\triangleright}(u)x) & \text{if } \hat{\triangleright}(u) \not\mapsto \omega \end{cases} & \hat{\mapsto}(ux) = \begin{cases} \omega & \text{if } \hat{\triangleright}(u) \mapsto \omega \\ \mapsto (\hat{\triangleright}(u)x) & \text{if } \hat{\triangleright}(u) \not\mapsto \omega. \end{cases}
\end{array}$$

We show the two extended functions are well-defined total functions.

Lemma 4.19. *$\hat{\triangleright} : S^* \to dom\,\mathcal{E}$ and $\hat{\mapsto} : S^* \to R$ are well-defined given a complete enumeration $\mathcal{E} : S^* \to R \times S^*$.*

Proof. It suffices to show $\hat{\triangleright}(u)$ and $\hat{\mapsto}(u)$ are well-defined for any u in S^*.

Let $S_i = \{u \in S^* : |u| = i\}$ for $i \in \mathbb{Z}, i \geq 0$.

$\hat{\triangleright}(u)$ and $\hat{\mapsto}(u)$ are well-defined iff either $u = \lambda$ (thus $\hat{\triangleright}(u) = \lambda$, $\hat{\mapsto}(u) = 0$) or $u = u'x$, where $x \in S$, and (i) $\hat{\triangleright}(u')$ is well-defined, (ii) $\hat{\triangleright}(u') \in dom\,\mathcal{E}$, and (iii) $\hat{\triangleright}(u')x \in dom\,\mathcal{E}$ if $\hat{\triangleright}(u') \not\mapsto \omega$.

Let P_i be true iff for all u in S_i, $\hat{\triangleright}(u)$ and $\hat{\mapsto}(u)$ are well-defined. Induct on i to show P_i. $\square$

$\hat{\mapsto}$ and $\hat{\triangleright}$ are $\mapsto$ and $\triangleright$ extended from $dom\,\mathcal{E}$ to S^*, respectively. We would like the extended functions to agree with the unextended functions on the shared part of the domain, as proved by the following theorem. We also observe that the range of $\hat{\triangleright}$ is the set of all unreduced sequences in $\mathcal{E}$ ($U_{\mathcal{E}}$).

Theorem 4.20. *For any $u \in dom\,\mathcal{E}$, $\hat{\mapsto}(u) = \mapsto (u)$, $\hat{\triangleright}(u) = \triangleright(u)$.*

Proof. Induct on $|u|$. $\square$

In [24] an algorithm is given that computes a black box function implied by any complete and finite enumeration. We will describe the algorithm here as a mathematical function. First we define its domain and codomain:

$\mathfrak{E} = \{\mathcal{E} : \mathcal{E}$ is a complete and finite enumeration$\}$,

$\mathfrak{B} = \{BB : BB$ is a black box function with a finite codomain$\}$.

The algorithm can be defined as a total function $\beta : \mathfrak{E} \to \mathfrak{B}$.

Algorithm 4.21 (*Black Box Function Computation*).

```
1. Let $\mathcal{E} : S^* \to R \times S^*$ be in $\mathfrak{E}$.
2. $BB : S^* \to R$ is defined by:
3.     $\forall u \in dom\,\mathcal{E}.\, BB(u) = \mapsto(u)$
4.     $\forall u \in (S^* - dom\,\mathcal{E}).\, (u = u'xw, u' \mapsto \omega, x \in S, w \in S^*) \to BB(u) = \omega$
5.     $\forall u \in (S^* - dom\,\mathcal{E}).\, (u = u'xw, u' \not\mapsto \omega, x \in S, w \in S^*, u'x \notin dom\,\mathcal{E}) \to BB(u) = BB(\rhd(u')xw)$.
6. $\beta(\mathcal{E}) = BB$.
```

Given a complete and finite enumeration $\mathcal{E}$ and an arbitrary stimulus sequence u, the algorithm proceeds as follows to find the mapped response for u. If u is an enumerated sequence, we read off its response from $\mathcal{E}$ (Line 3); otherwise, we find the longest prefix u' of u that is enumerated. Such a sequence u' must exist, as the empty sequence must have been extended by every single stimulus. There are two possibilities: either u' is illegal, or u' is reduced to a prior sequence; otherwise u' must have been extended and there would exist a longer prefix of u than u' that gets enumerated in $\mathcal{E}$, a contradiction. If u' is illegal, u is mapped to ω (Line 4); otherwise, we replace u' with the prior sequence it is reduced to in the original sequence u, and repeat the algorithm on the new sequence (Line 5), until a mapped response is finally derived.

We show β is a well-defined function.

Lemma 4.22. *$\beta : \mathfrak{E} \to \mathfrak{B}$ is well-defined.*

Proof. Given $\mathcal{E} : S^* \to R \times S^*$ in $\mathfrak{E}$, R is finite and $BB(\lambda) = \mapsto(\lambda) = 0$. It remains to show BB is a well-defined function. Consider any string u in S^*. The only difficult case is when $u \notin dom\,\mathcal{E}$, and its longest prefix in $dom\,\mathcal{E}$ (denoted by u') has $u' \not\mapsto \omega$ (Line 5). We claim u' must be reduced (if not, $u'x$ would have been a longer prefix of u in $dom\,\mathcal{E}$). Now we consider a new string formed by replacing u' with $\rhd(u')$ in u. If $\rhd(u') \mapsto \omega$, $BB(u) = \omega$; otherwise, we have $\rhd(u')x$ in $dom\,\mathcal{E}$. Since $|w|$ is finite, if we continue in this fashion, finally either $BB(u) = \omega$ by Line 4, or $BB(u) = BB(v) = \mapsto(v)$ by Line 3 for some v in $dom\,\mathcal{E}$. □

Example 4.23. Given the enumeration $\mathcal{E}$ defined in Example 4.4, we apply Algorithm 4.21 to find the response for an arbitrary stimulus sequence, say $LBCUDG$. Let the black box function be denoted by BB, then $BB(LBCUDG) = BB(LUDG) = BB(UDG) = BB(LG) = unlock$.

Example 4.24. Given the enumeration $\mathcal{E}$ defined in Example 4.4, we have the following algebraic derivation

$$\begin{aligned}
\hat{\rhd}(L) &= \rhd(\hat{\rhd}(\lambda)L) &&= \rhd(L) &&= L\\
\hat{\rhd}(LB) &= \rhd(\hat{\rhd}(L)B) &&= \rhd(LB) &&= LB\\
\hat{\rhd}(LBC) &= \rhd(\hat{\rhd}(LB)C) &&= \rhd(LBC) &&= L\\
\hat{\rhd}(LBCU) &= \rhd(\hat{\rhd}(LBC)U) &&= \rhd(LU) &&= U\\
\hat{\rhd}(LBCUD) &= \rhd(\hat{\rhd}(LBCU)D) &&= \rhd(UD) &&= L\\
\widehat{\mapsto}(LBCUDG) &= \mapsto(\hat{\rhd}(LBCUD)G) &&= \mapsto(LG) &&= unlock.
\end{aligned}$$

The result is not coincidental, but holds for every possible stimulus sequence. As proved by the following theorem, the extended response mapping $\widehat{\mapsto}$ is the same black box function computed by Algorithm 4.21.

Theorem 4.25. *Let $\mathcal{E} : S^* \to R \times S^*$ be a complete and finite enumeration. Then $\widehat{\mapsto} = \beta(\mathcal{E})$.*

Proof. $\widehat{\mapsto}$ and $\beta(\mathcal{E})$ as defined are both black box functions from S^* to R. Since $\beta(\mathcal{E})$ is defined recursively as

$$\beta(\mathcal{E})(u) = \begin{cases} \mapsto(u) & \text{if } u \in dom\,\mathcal{E}\\ \omega & \text{if } u = u'xw, u' \mapsto \omega, x \in S, w \in S^*\\ \beta(\mathcal{E})(\rhd(u')xw) & \text{if } u = u'xw, u' \not\mapsto \omega, x \in S, w \in S^*, u'x \notin dom\,\mathcal{E}, \end{cases}$$

it suffices to show

$$\widehat{\mapsto}(u) = \begin{cases} \mapsto(u) & \text{if } u \in dom\,\mathcal{E}\\ \omega & \text{if } u = u'xw, u' \mapsto \omega, x \in S, w \in S^*\\ \widehat{\mapsto}(\rhd(u')xw) & \text{if } u = u'xw, u' \not\mapsto \omega, x \in S, w \in S^*, u'x \notin dom\,\mathcal{E}. \end{cases}$$

If $u \in dom\,\mathcal{E}$, by Theorem 4.20 $\widehat{\mapsto}(u) = \mapsto(u)$. If $u \notin dom\,\mathcal{E}$, we write $u = u'xw$, where $u' \in dom\,\mathcal{E}$, $x \in S$, $w \in S^*$, but $u'x \notin dom\,\mathcal{E}$. Let $xw = vy$, where $y \in S$, then $u = u'xw = u'vy$.

If $u' \mapsto \omega$, induct on $|v|$ to show $\widehat{\mapsto}(u'vy) = \omega$.

If $u' \not\mapsto \omega$, induct on $|v|$ to show $\hat{\rhd}(u'v) = \hat{\rhd}(\rhd(u')v)$, hence $\widehat{\mapsto}(u'vy) = \widehat{\mapsto}(\rhd(u')vy)$ by the definition of $\widehat{\mapsto}$. □

With the extended $\widehat{\mapsto}$ function, we can prove that the indistinguishability relation $\sim$ over all unreduced sequences is an equivalence relation. This result will be used in Section 5.3 to merge indistinguishable unreduced sequences for enumeration minimization.

Lemma 4.26. *Let $\mathcal{E} : S^* \to R \times S^*$ be a complete enumeration. The relation $\sim\, \subset U_{\mathcal{E}} \times U_{\mathcal{E}}$ is an equivalence relation.*

Proof. We show $a \not\sim b \leftrightarrow \exists w \in S^+.\, \widehat{\mapsto}(aw) \neq \widehat{\mapsto}(bw)$, hence $a \sim b \leftrightarrow \forall w \in S^+.\, \widehat{\mapsto}(aw) = \widehat{\mapsto}(bw)$. We claim $\sim$ is an equivalence relation as equality is an equivalence relation. □

5. Transformation theory

5.1. Enumeration Mealy machine

A complete and finite enumeration encodes a finite state automaton with Mealy outputs (a Mealy machine). The inverse image of every unreduced sequence under $\hat{\triangleright}$ is a block in the partitioning of S^* and corresponds to a state. The $\triangleright$ and the $\mapsto$ functions respectively determine the transition function and the output function of the Mealy machine. We define below a special subset of Mealy machines and explore their relationship to enumerations.

Definition 5.1 (*Enumeration Mealy Machine*). An *enumeration Mealy machine* is a Mealy machine $M = \langle Q, \Sigma, \Gamma, \delta, \nu, q_0 \rangle$, where Γ properly contains $\{0, \omega\}$, a strict total order $\prec$ exists for Σ^* such that for all u, v in Σ^*, $|u| < |v|$ implies $u \prec v$, and the following hold assuming $\hat{\delta}$ is a function from $Q \times \Sigma^*$ to Q defined by (i) for all q in Q, $\hat{\delta}(q, \lambda) = q$, and (ii) for all q in Q, w in Σ^*, and a in Σ, $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$, and $\hat{\nu}$ is a function from Σ^* to Γ defined by (i) $\hat{\nu}(\lambda) = 0$, and (ii) for all w in Σ^* and a in Σ, $\hat{\nu}(wa) = \nu(\hat{\delta}(q_0, w), a)$:

Constraint 1. $Q = \{q_0, \dots, q_{n-1}\}$;
Constraint 2. For all q in Q, there exists w in Σ^* such that $\hat{\delta}(q_0, w) = q$;
Constraint 3. Let *min X* denote the smallest element of X according to $\prec$ for $X \subset \Sigma^*$. Then $c_0 \prec \cdots \prec c_{n-1}$, where $c_i = min\,\{z : \hat{\delta}(q_0, z) = q_i\}$;
Constraint 4. For all q_i in Q and a in Σ, $\hat{\nu}(c_i) = \omega$ implies $\delta(q_i, a) = q_i$;
Constraint 5. For all q in Q and a, b in Σ, $\nu(q, a) = \omega$ implies $\nu(\delta(q, a), b) = \omega$.

We would like an enumeration Mealy machine to be a Mealy machine that can be derived from a complete and finite enumeration. Therefore, its output alphabet Γ must contain the two special responses (0 and ω) and at least one other observable response, and all input strings must observe a strict total order $\prec$ such that any shorter string must relate to any longer string by $\prec$. Following the convention in [14], we extend the output function to all input strings such that (i) the output for the empty string is 0, and (ii) the output for any non-empty string is the output of the last transition on the path generated by the string from the initial state.

Five constraints are enforced. Some are results of the enumeration rules or axioms (Constraint 2, Constraint 5). Others are added to ensure a one-to-one correspondence between complete and finite enumerations and enumeration Mealy machines (it will become clear in Section 5.2 why they are needed). Informally, we describe the constraints below.

Constraint 1. For an n-state enumeration Mealy machine, the set of states are named $q_0, \dots, q_{n-1}$.
Constraint 2. Every state is reachable from (i.e., connected from) the initial state q_0.
Constraint 3. Let c_i be the first word according to $\prec$ that takes the automaton from the initial state q_0 to the state q_i, then the sequence of words $c_0, \dots, c_{n-1}$ are ordered by $\prec$.
Constraint 4. If c_i as defined above has ω as its output, then the state q_i is a trap state (i.e., all outgoing arcs from q_i return to q_i).
Constraint 5. If one incoming arc to the state q_i has ω as its output, then all outgoing arcs from q_i have ω as their output.

Example 5.2. The enumeration $\mathcal{E}$ defined in Example 4.4 encodes a state machine M defined below. In Section 5.2 we will show how to derive M mechanically from $\mathcal{E}$, and vice versa.

Let $M = \langle Q, S, R, \delta, \nu, q_0 \rangle$, where $Q = \{q_0, q_1, q_2, q_3, q_4\}$, S, R, and the strict total order $\prec$ on S^* are defined the same as for $\mathcal{E}$, and $\delta : Q \times S \to Q$ and $\nu : Q \times S \to R$ are defined as in Fig. 2. It can be checked that M satisfies the five constraints required for an enumeration Mealy machine.

In the enumeration process, as unreduced sequences are being identified, states of the system are being discovered. If all the unreduced sequences are canonical, they represent distinct system states; otherwise, there are missed equivalences among indistinguishable unreduced sequences, and the states they represent are indistinguishable in the following sense: from such states, if we inject the same non-empty input string, we will get the same output. (We can extend the output function following the convention in [14] such that the output for any non-empty input string injected to any state is the output of the last transition on the path generated by the string from the given state.) We define the distinguishability/indistinguishability relations among states of an enumeration Mealy machine as follows, one as the complement of the other.

Definition 5.3 (*Distinguishability/indistinguishability*). Let $M = \langle Q, \Sigma, \Gamma, \delta, \nu, q_0 \rangle$ be an enumeration Mealy machine. The relation $\not\sim \subset Q \times Q$ is the smallest relation satisfying the following rules:

Rule 1. $(\exists x \in \Sigma \,.\, \nu(a, x) \neq \nu(b, x)) \to a \not\sim b$;
Rule 2. $(a \not\sim b, \exists x \in \Sigma \,.\, (\delta(c, x) = a, \delta(d, x) = b)) \to c \not\sim d$.

The relation $\sim \subset Q \times Q$ is defined by $\forall a, b \in Q \,.\, a \sim b \leftrightarrow (a, b) \notin \not\sim$.

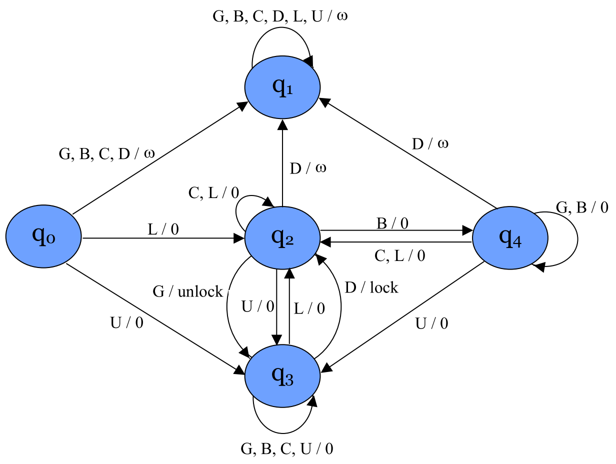

Fig. 2. A state machine for the safe controller.

Two states of an enumeration Mealy machine are distinguishable if the same non-empty input string injected to the two states could generate different outputs. This happens in the following situations:

- From the two states, one arbitrary but the same input generates different outputs. (Rule 1)
- From the two states, one arbitrary but the same input leads to two already distinguishable states. (Rule 2)

We can combine (merge) indistinguishable states of an enumeration Mealy machine and simplify the automaton. If all the states are pairwise distinguishable, the enumeration Mealy machine is minimal.

Definition 5.4 (*Minimal Enumeration Mealy Machine*). An enumeration Mealy machine $M = \langle Q, \Sigma, \Gamma, \delta, \nu, q_0\rangle$ is *minimal* iff for all a, b in Q, $a \neq b$ implies $a \not\sim b$.

Example 5.5. Given the enumeration Mealy machine M defined in Example 5.2, by Definition 5.3, Rule 1

$q_0 \not\sim q_1$	as	$\nu(q_0, L) = 0, \nu(q_1, L) = \omega$	$q_1 \not\sim q_2$	as	$\nu(q_1, B) = \omega, \nu(q_2, B) = 0$
$q_0 \not\sim q_2$	as	$\nu(q_0, B) = \omega, \nu(q_2, B) = 0$	$q_1 \not\sim q_3$	as	$\nu(q_1, B) = \omega, \nu(q_3, B) = 0$
$q_0 \not\sim q_3$	as	$\nu(q_0, B) = \omega, \nu(q_3, B) = 0$	$q_1 \not\sim q_4$	as	$\nu(q_1, B) = \omega, \nu(q_4, B) = 0$
$q_0 \not\sim q_4$	as	$\nu(q_0, B) = \omega, \nu(q_4, B) = 0$	$q_2 \not\sim q_3$	as	$\nu(q_2, G) = unlock, \nu(q_3, G) = 0$
$q_3 \not\sim q_4$	as	$\nu(q_3, D) = lock, \nu(q_4, D) = \omega$	$q_2 \not\sim q_4$	as	$\nu(q_2, G) = unlock, \nu(q_4, G) = 0,$

hence states in Q are pairwise distinguishable, and M is a minimal enumeration Mealy machine.

With the extended output function defined for (state, non-empty input string) pairs, we can prove the indistinguishability relation among all states of an enumeration Mealy machine is an equivalence relation.

Lemma 5.6. *Let $M = \langle Q, \Sigma, \Gamma, \delta, \nu, q_0\rangle$ be an enumeration Mealy machine. The relation $\sim \subset Q \times Q$ is an equivalence relation.*

Proof. We show $a \not\sim b \leftrightarrow \exists u \in \Sigma^+ . \nu(a, u) \neq \nu(b, u)$, where ν is extended to $Q \times \Sigma^+$ by: $\forall q \in Q . \forall u \in \Sigma^* . \forall a \in \Sigma . \nu(q, ua) = \nu(\hat{\delta}(q, u), a)$. Therefore, $a \sim b \leftrightarrow \forall u \in \Sigma^+ . \nu(a, u) = \nu(b, u)$. We claim $\sim$ is an equivalence relation as equality is an equivalence relation. □

5.2. Enumeration to and from state machine

We first present two algorithms that transform complete and finite enumerations to enumeration Mealy machines, and vice versa. Then we establish a representation theorem. These two algorithms are also new contributions of this paper.

We will describe these algorithms as mathematical functions. Following the notation in Section 4.2, $\mathfrak{E}$ denotes the set of all complete and finite enumerations. We define

$\mathfrak{M} = \{M : M \text{ is an enumeration Mealy machine}\}$.

The transformation from enumeration to state machine is defined as a total function $\Phi : \mathfrak{E} \to \mathfrak{M}$.

Algorithm 5.7 (*Enumeration to State Machine Transformation*).

```
1. Let $\mathcal{E} : S^* \to R \times S^*$ be in $\mathfrak{E}$.
2. Let $U_{\mathcal{E}} = \{u : u \rhd u\} = \{c_0, \ldots, c_{n-1}\}$, where $c_0 \prec \cdots \prec c_{n-1}$.
3. Let $Q = \{q_0, \ldots, q_{n-1}\}$.
4. $\delta : Q \times S \to Q$ is defined by:
5.     $\forall c_i \in U_{\mathcal{E}} . \forall c_j \in U_{\mathcal{E}} . \forall a \in S . c_i a \rhd c_j \to \delta(q_i, a) = q_j$
```

```
6.     $\forall c_i \in U_{\mathcal{E}}.\forall a \in S.\, c_i \mapsto \omega \to \delta(q_i, a) = q_i$.
7. $\nu : Q \times S \to R$ is defined by:
8.     $\forall c_i \in U_{\mathcal{E}}.\forall a \in S.\forall r \in R.\, c_i a \mapsto r \to \nu(q_i, a) = r$
9.     $\forall c_i \in U_{\mathcal{E}}.\forall a \in S.\, c_i \mapsto \omega \to \nu(q_i, a) = \omega$.
10. $\Phi(\mathcal{E}) = \langle Q, S, R, \delta, \nu, q_0\rangle$.
```

Given any complete and finite enumeration $\mathcal{E}$ (Line 1), we find all its unreduced sequences and order them by the pre-defined strict total order $\prec$ over all stimulus sequences (Line 2). They will get mapped to states $q_0, \ldots, q_{n-1}$ (assume there are n states in total), respectively (Line 3), and a smaller sequence by $\prec$ will be mapped to a state with a smaller index (this will satisfy Constraints 1 and 3 required in the definition for an enumeration Mealy machine). If the state q_i is mapped from a legal and unreduced sequence c_i (hence c_i must have been extended in $\mathcal{E}$), the transitions out of q_i as well as their associated outputs can be defined by existing rows in the enumeration table that define the responses and reductions for the extensions of c_i by every single stimulus (Lines 5 and 8); otherwise, the state q_i must be mapped from an illegal and unreduced sequence c_i, hence q_i is made a trap state (Line 6) with all its outgoing arcs outputting ω (Line 9), in order to satisfy Constraints 4 and 5 required for an enumeration Mealy machine.

The transformation from state machine to enumeration is defined as a total function $\Psi : \mathfrak{M} \to \mathfrak{E}$.

Algorithm 5.8 (*State Machine to Enumeration Transformation*).

```
1. Let $M = \langle Q, \Sigma, \Gamma, \delta, \nu, q_0\rangle$ be in $\mathfrak{M}$, where $Q = \{q_0, \ldots, q_{n-1}\}$.
2. Let $c_0, \ldots, c_{n-1}$ be defined by $c_i = min\,\{z : \hat{\delta}(q_0, z) = q_i\}$.
3. Let $Q' = \{q_i : q_i \in Q, \hat{\nu}(c_i) \neq \omega\}$.
4. $\mathcal{E} : \Sigma^* \to \Gamma \times \Sigma^*$ is defined by:
5.     $\lambda \mapsto 0,\ \lambda \rhd \lambda$.
6.     $\forall q_i \in Q'.\forall a \in \Sigma.\forall q_j \in Q.\, \delta(q_i, a) = q_j \to c_i a \rhd c_j$
7.     $\forall q_i \in Q'.\forall a \in \Sigma.\forall r \in \Gamma.\, \nu(q_i, a) = r \to c_i a \mapsto r$.
8. $\Psi(M) = \mathcal{E}$.
```

Given any enumeration Mealy machine M with the state set Q (Line 1), we first compute for every state the smallest possible word according to the pre-defined strict total order $\prec$ over all input strings that takes the automaton from the initial state to the state (Line 2). The computed words represent all the unreduced sequences in the converted enumeration. We collect all the states whose computed words do not map to ω by the extended output function (on all input strings) into the set Q' (Line 3). The elements in Q' now correspond to all legal and unreduced sequences in the resulting enumeration. We construct the enumeration by extending only these legal and unreduced sequences, referring to the transition function for equivalence declarations (Line 6) and the output function for response mappings (Line 7). As a special case, the empty sequence is mapped to 0 and unreduced (Line 5).

In the representation theorem that follows, we show the two transformation functions are well-defined total functions, and establish the one-to-one correspondence between complete and finite enumerations and enumeration Mealy machines.

Theorem 5.9 (*Representation Theorem*).

i. *$\Phi : \mathfrak{E} \to \mathfrak{M}$ is a well-defined total function;*
ii. *$\Psi : \mathfrak{M} \to \mathfrak{E}$ is a well-defined total function;*
iii. *$\Phi : \mathfrak{E} \to \mathfrak{M}$ and $\Psi : \mathfrak{M} \to \mathfrak{E}$ are inverse bijections.*

Proof. For (i) it suffices to show $\Phi(\mathcal{E}) \in \mathfrak{M}$ for all $\mathcal{E}$ in $\mathfrak{E}$. For (ii) it suffices to show $\Psi(M) \in \mathfrak{E}$ for all M in $\mathfrak{M}$. For Part (iii) it suffices to show $\Phi \circ \Psi = 1_{\mathfrak{M}}$ and $\Psi \circ \Phi = 1_{\mathfrak{E}}$, where $1_{\mathfrak{M}} : \mathfrak{M} \to \mathfrak{M}$ and $1_{\mathfrak{E}} : \mathfrak{E} \to \mathfrak{E}$ are identity mappings. □

Example 5.10. Given the enumeration $\mathcal{E}$ defined in Example 4.4, we apply Algorithm 5.7 to obtain the state machine M defined in Example 5.2, and then apply Algorithm 5.8 to transform it back.

Unreduced sequences in $U_{\mathcal{E}}$ are mapped as follows according to the strict total order $\prec$ on S^*: $\lambda = c_0$, $B = c_1$, $L = c_2$, $U = c_3$, $LB = c_4$. They correspond to states $q_0, q_1, \ldots, q_4$ respectively. It follows that $\Phi(\mathcal{E}) = M$.

Starting from M, we can compute the first words according to the strict total order $\prec$ on S^* taking the automaton from the starting state to each state: $c_0 = \lambda$, $c_1 = B$, $c_2 = L$, $c_3 = U$, $c_4 = LB$. They correspond to all the unreduced sequences in the resulting enumeration $\Psi(M)$. Since $c_1 = B$ is the only sequence among them that is mapped to ω by $\hat{\nu}$, all the others are extended by every input symbol in $\Psi(M)$, as required by Algorithm 5.8. Again, it follows that $\Psi(M) = \Psi(\Phi(\mathcal{E})) = \mathcal{E}$.

With the representation theorem, it becomes intuitively clear that computing the mapped response for any stimulus sequence from a complete and finite enumeration by Algorithm 4.21 or equivalently, through the derivation of $\hat{\mapsto}$ as defined in Definition 4.18, simulates computing the output generated by the input string by the extended output function of the corresponding Mealy machine, as proved below.

Theorem 5.11. *Let $\mathcal{E} : S^* \to R \times S^*$ be in $\mathfrak{E}$, and let $\Phi(\mathcal{E}) = \langle Q, S, R, \delta, \nu, q_0\rangle$. Then $\hat{\mapsto} = \hat{\nu}$.*

Proof. $\hat{\mapsto}$ and $\hat{\nu}$ as defined are both total functions from S^* to R. Since we have shown in the proof of Theorem 4.25 that $\hat{\mapsto}$ can be defined recursively as

$$\hat{\mapsto}(u) = \begin{cases} \mapsto(u) & \text{if } u \in dom\,\mathcal{E} \\ \omega & \text{if } u = u'xw,\, u' \mapsto \omega,\, x \in S,\, w \in S^* \\ \hat{\mapsto}(\rhd(u')xw) & \text{if } u = u'xw,\, u' \not\mapsto \omega,\, x \in S,\, w \in S^*,\, u'x \notin dom\,\mathcal{E}, \end{cases}$$

it suffices to show

$$\hat{\nu}(u) = \begin{cases} \mapsto(u) & \text{if } u \in dom\,\mathcal{E} \\ \omega & \text{if } u = u'xw,\, u' \mapsto \omega,\, x \in S,\, w \in S^* \\ \hat{\nu}(\rhd(u')xw) & \text{if } u = u'xw,\, u' \not\mapsto \omega,\, x \in S,\, w \in S^*,\, u'x \notin dom\,\mathcal{E}. \end{cases}$$

The proof is mechanical and follows the definition of $\Phi(\mathcal{E})$. □

Example 5.12. As a third way to derive the mapped response for an arbitrary stimulus sequence *LBCUDG*, given the enumeration $\mathcal{E}$ defined in Example 4.4, we do an algebraic derivation of $\hat{\nu}(LBCUDG)$, where ν is the output function of $\Phi(\mathcal{E})$.

$$\begin{aligned} \hat{\nu}(LBCUDG) &= \nu(\hat{\delta}(q_0, LBCUD), G) \\ &= \nu(\delta(\delta(\delta(\delta(\delta(\hat{\delta}(q_0, \lambda), L), B), C), U), D), G) \\ &= \nu(\delta(\delta(\delta(\delta(\delta(q_0, L), B), C), U), D), G) \\ &= \nu(\delta(\delta(\delta(\delta(q_2, B), C), U), D), G) \\ &= \nu(\delta(\delta(\delta(q_4, C), U), D), G) \\ &= \nu(\delta(\delta(q_2, U), D), G) \\ &= \nu(\delta(q_3, D), G) \\ &= \nu(q_2, G) \\ &= unlock. \end{aligned}$$

The representation theorem also leads to observations in presenting a bigger picture of the sequence-based specification method and understanding the theoretical ramifications.

Let us consider again the function $\beta : \mathfrak{E} \to \mathfrak{B}$ defined in Section 4.2. We first claim it is not injective, by the example below.

Example 5.13. Consider $\mathcal{E}_1$, $\mathcal{E}_2$ in $\mathfrak{E}$, where $\mathcal{E}_1 : S^* \to R \times S^*$, $\mathcal{E}_2 : S^* \to R \times S^*$ for $S = \{a\}$, $R = \{0, \omega, r\}$, and the strict total order $\prec$ on S^* defined by $a^n \prec a^{n+1}$ for all non-negative integer n:

$$\begin{array}{lll} \mathcal{E}_1(\lambda) = (0, \lambda) & \mathcal{E}_1(a) = (r, \lambda) & \\ \mathcal{E}_2(\lambda) = (0, \lambda) & \mathcal{E}_2(a) = (r, a) & \mathcal{E}_2(aa) = (r, a). \end{array}$$

Note that $\beta(\mathcal{E}_1) = \beta(\mathcal{E}_2) = BB$ for $\mathcal{E}_1 \neq \mathcal{E}_2$, where $BB : S^* \to R$ is defined by $BB(\lambda) = 0$, and $\forall u \in S^+ . BB(u) = r$, hence β is not injective.

This implies that there may exist more than one complete and finite enumeration for a black box function whose codomain is finite. In practice, we often encounter situations in which different people come up with different enumerations (specifications) that describe the behavior of the same software system under development. If they are all done correctly, all of them should produce the same black box function as implied by requirements with only variations in the form.

Next we claim neither is β surjective, by the following argument.

From the representation theorem we see $|\mathfrak{E}| = |\mathfrak{M}|$. Since every enumeration Mealy machine is a special deterministic finite state automaton, the number of all enumeration Mealy machines is countably infinite. Therefore, $|\mathfrak{E}|$ is a countable number. However, consider a proper subset of $\mathfrak{B}$ defined by $\mathfrak{B}' = \{BB : BB$ is an element of $\mathfrak{B}$ from S^* to R, where $S = \{a\}$, $R = \{0, \omega, r\}\}$. $|\mathfrak{B}'|$, and hence $|\mathfrak{B}|$, is uncountable by the diagonalization method. Since $|\mathfrak{E}|$ is countable, but $|\mathfrak{B}|$ is uncountable, β cannot be surjective.

The above discussion implies that there may exist black box functions with finite codomains for which no complete and finite enumerations can be found. This result should not be surprising as, applying the same counting argument, we could deduce that there exist black box functions with finite codomains that are not computable by any Turing machine, which agrees with our understanding that there are simply more functions than Turing machines to compute them [14]. Sequence-based specification was developed to work with real-world problems that can be programmed for computers. The method concerns itself with such problems whose black box functions compose a small portion of $\mathfrak{B}$.

5.3. Enumeration and state machine minimization

As illustrated in Section 4.1, a complete and finite enumeration may contain missed equivalences and hence not be minimal. We give an algorithm that converts any complete and finite enumeration to a complete, finite, and minimal enumeration that computes the same black box function. This algorithm is also a new contribution of this paper. With the representation theorem, it should become clear that enumeration minimization amounts to merging indistinguishable unreduced sequences that represent indistinguishable states in the corresponding state machine.

Let $\mathfrak{E}' = \{\mathcal{E} : \mathcal{E} \text{ is a complete, finite, and minimal enumeration}\}$.

Enumeration minimization is defined as a total function $\alpha : \mathfrak{E} \to \mathfrak{E}'$.

Algorithm 5.14 (*Enumeration Minimization*).

```
1. Let $\mathcal{E} : S^* \to R \times S^*$ be in $\mathfrak{E}$.
2. Let $U_{\mathcal{E}} = \{u : u \rhd u\} = \{c_0, \ldots, c_{n-1}\}$, where $c_0 \prec \cdots \prec c_{n-1}$.
3. Let $U'_{\mathcal{E}} = U_{\mathcal{E}} - \{c_j : \exists\, i < j.\, c_i \sim c_j\}$.
4. Let $\mathcal{E}' : S^* \to R \times S^*$ be defined by:
5.     $\lambda \mapsto 0, \lambda \rhd \lambda$.
6.     $\forall c_i \in U'_{\mathcal{E}}.\, \forall a \in S.\, (c_i a \underset{\mathcal{E}}{\rhd} c_j, c_j \in U'_{\mathcal{E}}) \to c_i a \underset{\mathcal{E}'}{\rhd} c_j$
7.     $\forall c_i \in U'_{\mathcal{E}}.\, \forall a \in S.\, (c_i a \underset{\mathcal{E}}{\rhd} c_j, c_j \notin U'_{\mathcal{E}}, c_j \sim c_k, c_k \in U'_{\mathcal{E}}) \to c_i a \underset{\mathcal{E}'}{\rhd} c_k$
8.     $\forall c_i \in U'_{\mathcal{E}}.\, \forall a \in S.\, \forall r \in R.\, c_i a \underset{\mathcal{E}}{\mapsto} r \to c_i a \underset{\mathcal{E}'}{\mapsto} r$.
9. $\alpha(\mathcal{E}) = \mathcal{E}'$.
```

Given any complete and finite enumeration $\mathcal{E}$ (Line 1), we collect in the set $U_{\mathcal{E}}$ all its unreduced sequences and order them according to the pre-defined strict total order $\prec$ on all stimulus sequences (Line 2). We further reduce $U_{\mathcal{E}}$ to $U'_{\mathcal{E}}$ to contain only the canonical sequences that are distinguishable from all their prior unreduced sequences (Line 3). The elements in $U_{\mathcal{E}}$ but not in $U'_{\mathcal{E}}$ are to be merged with their indistinguishable counterparts in $U'_{\mathcal{E}}$. The resulting enumeration $\mathcal{E}'$ is constructed as follows. The empty sequence is defined explicitly (Line 5). Only canonical sequences in $U'_{\mathcal{E}}$ are extended. To define these extensions, we refer to $\mathcal{E}$ for responses (Line 8) and reductions (Lines 6–7). If the reduction in $\mathcal{E}$ is to an element outside of $U'_{\mathcal{E}}$, the reduction in $\mathcal{E}'$ is made to its indistinguishable counterpart in $U'_{\mathcal{E}}$ (Line 7).

We show α is well-defined and a surjection. The constructed object must satisfy all the nine axioms required for a complete, finite, and minimal enumeration.

Lemma 5.15. *$\alpha : \mathfrak{E} \to \mathfrak{E}'$ is a well-defined surjection.*

Proof. We show $\alpha(\mathcal{E}) \in \mathfrak{E}'$ for all $\mathcal{E}$ in $\mathfrak{E}$. Note that $\mathfrak{E}' \subsetneq \mathfrak{E}$. Since $\alpha(\mathcal{E}) = \mathcal{E}$ for all $\mathcal{E}$ in $\mathfrak{E}'$, α is surjective. □

Consider any complete and finite enumeration $\mathcal{E}$. It implies a black box function which can be computed in three different but equivalent ways: by Algorithm 4.21, by the extended response mapping, or by the extended output function of the converted Mealy machine. The black box function, thus computed, will be referred to as the enumeration's "underlying" black box function.

As established by the following theorem, enumeration minimization does not change the underlying black box function.

Theorem 5.16. $\beta = \beta \circ \alpha$.

Proof. Let $\mathcal{E} : S^* \to R \times S^*$ be in $\mathfrak{E}$. Induct on $|u|$ to show $\underset{\mathcal{E}}{\hat{\rhd}}(u) \sim \underset{\alpha(\mathcal{E})}{\hat{\rhd}}(u)$ in $\mathcal{E}$ for all u in S^*. Then $\underset{\mathcal{E}}{\hat{\mapsto}} = \underset{\alpha(\mathcal{E})}{\hat{\mapsto}}$, hence $\beta(\mathcal{E}) = \beta(\alpha(\mathcal{E}))$ by Theorem 4.25. □

In parallel, we give an enumeration Mealy machine minimization algorithm, also expressed in function form. This algorithm is essentially the same as the existing Mealy machine minimization algorithm in [11], except that we are dealing with a special subset of Mealy machines and, hence, we name the states of the minimized automaton in a particular way that guarantees all the constraints for enumeration Mealy machines are still satisfied after minimization.

Let $\mathfrak{M}' = \{M : M \text{ is a minimal enumeration Mealy machine}\}$.

Enumeration Mealy machine minimization is defined as a total function $\gamma : \mathfrak{M} \to \mathfrak{M}'$.

Algorithm 5.17 (*Enumeration Mealy Machine Minimization*).

```
1. Let $M = \langle Q, \Sigma, \Gamma, \delta, \nu, q_0 \rangle$ be in $\mathfrak{M}$.
2. Let $Q' = Q - \{q_j : \exists\, i < j.\, q_i \sim q_j\}$.
3. Let $Q'' = \{q_0, \ldots, q_{|Q'|-1}\}$.
4. Let $\theta : Q \to Q''$ be defined by:
5.     $\forall q \in Q.\, (q \sim q_i, q_i \in Q', |\{q_j : q_j \in Q', j < i\}| = k) \to \theta(q) = q_k$.
6. Let $\delta' : Q'' \times \Sigma \to Q''$ and $\nu' : Q'' \times \Sigma \to \Gamma$ be defined by:
7.     $\forall q \in Q'.\, \forall a \in \Sigma.\, \forall p \in Q.\, \delta(q, a) = p \to \delta'(\theta(q), a) = \theta(p)$
8.     $\forall q \in Q'.\, \forall a \in \Sigma.\, \forall r \in \Gamma.\, \nu(q, a) = r \to \nu'(\theta(q), a) = r$.
9. Let $M' = \langle Q'', \Sigma, \Gamma, \delta', \nu', q_0 \rangle$.
10. $\gamma(M) = M'$.
```

Given any enumeration Mealy machine M (Line 1), its state set Q can be partitioned into equivalence classes based on the indistinguishability relation. From each equivalence class, we pick the state with the smallest index and collect them in the set Q' (Line 2). They must be pairwise distinguishable, and are the states that remain in the minimized automaton. Since states in Q' only constitute a subset of the original state set Q, they need to be renamed from q_0 to $q_{|Q'|-1}$ (Line 3), without changing the ordering of indices among them. We define a θ function that maps any state in Q to its indistinguishable and renamed counterpart in Q'' (Lines 4–5). A new Mealy machine M' is constructed from M by including only states in Q' (now renamed in Q'') and redirecting arcs to elements outside of Q' to their indistinguishable and renamed counterparts in Q'' (Lines 6–9).

We show γ is well-defined and a surjection. The constructed object must satisfy the definition for a minimal enumeration Mealy machine.

Lemma 5.18. *$\gamma : \mathfrak{M} \to \mathfrak{M}'$ is a well-defined surjection.*

Proof. We show $\gamma(M) \in \mathfrak{M}'$ for all M in $\mathfrak{M}$. Note that $\mathfrak{M}' \subsetneq \mathfrak{M}$. Since $\gamma(M) = M$ for all M in $\mathfrak{M}'$, γ is surjective. □

As expected, enumeration Mealy machine minimization does not change the extended output function.

Theorem 5.19. *Let $M = \langle Q, \Sigma, \Gamma, \delta, \nu, q_0\rangle$ be in $\mathfrak{M}$, and $\gamma(M) = \langle Q', \Sigma, \Gamma, \delta', \nu', q_0\rangle$. Then $\hat{\nu} = \hat{\nu'}$.*

Proof. The Q' as stated in the theorem corresponds to the Q'' in Algorithm 5.17. We show $\hat{\nu} = \hat{\nu'}$ for M and M' as defined in the algorithm. Induct on $|u|$ to show $\delta(q_0, u) \sim \delta'(q_0, u)$ in M for all u in Σ^*. □

The result that follows is exemplified by Example 5.10. As we transform between enumerations and enumeration Mealy machines, minimality is preserved.

Theorem 5.20.

i. $\mathcal{E} \in \mathfrak{E}' \leftrightarrow \Phi(\mathcal{E}) \in \mathfrak{M}'$;
ii. $M \in \mathfrak{M}' \leftrightarrow \Psi(M) \in \mathfrak{E}'$.

Proof. We show for every $\mathcal{E}$ in $\mathfrak{E}'$, the constructed $\Phi(\mathcal{E})$ is in $\mathfrak{M}'$, and for every M in $\mathfrak{M}'$, the constructed $\Psi(M)$ is in $\mathfrak{E}'$. The "if" parts of both (i) and (ii) follow from the established "only if" parts and the representation theorem. □

The domain of Φ (or Ψ, respectively) can then be restricted to a subset $\mathfrak{E}'$ (or $\mathfrak{M}'$), with the codomain being reduced accordingly to $\mathfrak{M}'$ (or $\mathfrak{E}'$). Let Φ' and Ψ' denote the restricted transformation functions. We have a representation theorem for complete, finite, and minimal enumerations in terms of minimal enumeration Mealy machines.

Theorem 5.21. *$\Phi' : \mathfrak{E}' \to \mathfrak{M}'$ and $\Psi' : \mathfrak{M}' \to \mathfrak{E}'$ are inverse bijections.*

Proof. By Theorem 5.20 $\Phi' : \mathfrak{E}' \to \mathfrak{M}'$ and $\Psi' : \mathfrak{M}' \to \mathfrak{E}'$ are well-defined total functions. We show $\Phi' \circ \Psi' = 1_{\mathfrak{M}'}$ and $\Psi' \circ \Phi = 1_{\mathfrak{E}'}$, where $1_{\mathfrak{M}'} : \mathfrak{M}' \to \mathfrak{M}'$ and $1_{\mathfrak{E}'} : \mathfrak{E}' \to \mathfrak{E}'$ are identity mappings. □

Because we have expressed both the transformation algorithms and the minimization algorithms as mathematical functions, we can apply function composition and prove that the diagram in Fig. 3 commutes, hence transformations (to enumerations or state machines) and minimizations (of the corresponding form) can be applied in an arbitrary order without affecting the final result.

Theorem 5.22.

i. $\Phi \circ \alpha = \gamma \circ \Phi$;
ii. $\alpha \circ \Psi = \Psi \circ \gamma$.

Proof. For (i) both sides are total functions from $\mathfrak{E}$ to $\mathfrak{M}'$. Let $\mathcal{E} : S^* \to R \times S^*$ be in $\mathfrak{E}$. Let $M = \Phi(\mathcal{E}) = \langle Q, S, R, \delta, \nu, q_0\rangle$, $M_1 = \Phi \circ \alpha(\mathcal{E}) = \langle Q_1, S, R, \delta_1, \nu_1, q_0\rangle$, and $M_2 = \gamma \circ \Phi(\mathcal{E}) = \langle Q_2, S, R, \delta_2, \nu_2, q_0\rangle$. It follows from Theorems 5.11 and 5.19, and the proof of Theorem 5.16 that $\hat{\nu}_1 = \underset{\alpha(\mathcal{E})}{\hat{\mapsto}} = \underset{\mathcal{E}}{\hat{\mapsto}} = \hat{\nu} = \hat{\nu}_2$. Given $M_1, M_2 \in \mathfrak{M}'$ we claim $Q_1 = Q_2$, $\delta_1 = \delta_2$, and $\nu_1 = \nu_2$, hence $M_1 = M_2$.

Part (ii) follows from Part (i) and the representation theorem. □

6. Conclusions and future work

The axiomatization of sequence-based specification has proved its worth on a theoretical basis, as demonstrated in this paper. The relationship between enumerations and state machines is now precise with the representation theorem. We believe that the axiom system will facilitate the comparison of sequence-based specification with other mathematically described methods, as well as the transformation of artifacts from one method to another. We hope that this will contribute to the eventual unification of formal ideas for software development.

$$\begin{tikzcd}
\mathfrak{E} \arrow[r, shift left, "\Phi"] \arrow[d, "\alpha"'] & \mathfrak{M} \arrow[l, shift left, "\Psi"] \arrow[d, "\gamma"] \\
\mathfrak{E}' \arrow[r, shift left, "\Phi'"] & \mathfrak{M}' \arrow[l, shift left, "\Psi'"]
\end{tikzcd}$$

Fig. 3. A transition diagram.

The axiomatization of sequence-based specification has proved its worth on a practical basis, as demonstrated in our theory for managing requirements changes [19], the change algorithms and their implementation in a prototype enumeration tool. The axiom system was essential to the development of the change algorithms that maximize the potential automation support for stimulus addition and deletion, response changes, legality changes, and equivalence changes. These algorithms meet a very great need in field application of sequence-based specification, and have a huge practical impact on maintaining specifications over time in the presence of change.

Further work is underway to improve the sequence-based specification method. Field use of the method has revealed that large productivity gains are possible with the adoption of ideas from string rewriting theory. Effective application of these ideas will require corresponding enhancement of the Proto_Seq tool.

Effective abstractions are key to the productive use of the method, especially with respect to handling timing, interrupts, and continuity. We expect that a thorough algebraic treatment of abstraction will produce benefits in application. We need better capability to introduce, remove, change abstractions, compare enumerations with different abstractions, and to minimize information loss during abstraction. Our fieldwork has demonstrated that abstractions occur in patterns; as field experience expands, we anticipate automated support for abstraction management. Definition of abstractions within enumeration tools, as well as their automatic removal, is important for code generation.

We are also addressing timing and continuity directly by attempting to extend the constructive process to handle timing and continuity in the forms typically encountered in applications of Simulink [2] and similar development environments. The axiom system presented in this paper provides a theoretical framework to explore these lines of inquiry.

Acknowledgment

The authors wish to thank Michael D. Vose for reviewing an earlier version of the paper and the many insightful suggestions and discussions, and the anonymous referees for their helpful comments to improve this paper.

References

[1] ESP Project. http://sqrl.eecs.utk.edu/esp/index.html.
[2] The Mathworks. http://www.mathworks.com/products/simulink.
[3] W. Bartussek, D.L. Parnas, Using assertions about traces to write abstract specifications for software modules, in: Proceedings of the 2nd Conference of the European Cooperation on Informatics, Springer-Verlag, 1978, pp. 211–236.
[4] T. Bauer, F. Bohr, D. Landmann, T. Beletski, R. Eschbach, J.H. Poore, From requirements to statistical testing of embedded systems, in: Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems, IEEE Computer Society, 2007, pp. 3–9.
[5] G.H. Broadfoot, P.J. Broadfoot, Academia and industry meet: Some experiences of formal methods in practice, in: Proceedings of the 10th Asia-Pacific Software Engineering Conference, IEEE Computer Society, 2003, pp. 49–59.
[6] J. Brzozowski, Derivatives of regular expressions, Journal of the ACM 11 (4) (1964) 481–494.
[7] J. Brzozowski, Representation of a class of nondeterministic semiautomata by canonical words, Theoretical Computer Science 356 (2006) 46–57.
[8] J. Brzozowski, H. Jürgensen, Theory of deterministic trace-assertion specifications, Technical Report CS-2004-30, University of Waterloo, 2004.
[9] J. Brzozowski, H. Jürgensen, Representation of semiautomata by canonical words and equivalences, International Journal of Foundations of Computer Science 16 (5) (2005) 831–850.
[10] J. Brzozowski, H. Jürgensen, Representation of semiautomata by canonical words and equivalences, part II: Specification of software modules, International Journal of Foundations of Computer Science 18 (5) (2007) 1065–1087.
[11] A. Gill, Introduction to the Theory of Finite-State Machines, McGraw-Hill, New York, 1962.
[12] C.L. Heitmeyer, Software cost reduction, in: J.J. Marciniak (Ed.), Encyclopedia of Software Engineering, Wiley-Interscience, 2001.
[13] C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.
[14] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.
[15] P.J. Hopcroft, G.H. Broadfoot, Combining the box structure development method and CSP for software development, Electronic Notes in Theoretical Computer Science 128 (6) (2005) 127–144.
[16] D. Jackson, M. Thomas, L.I. Millett (Eds.), Software for Dependable Systems: Sufficient Evidence? National Research Council, The National Academies Press, Washington, DC, 2007.

[17] R. Janicki, E. Sekerinski, Foundations of the trace assertion method of module interface specification, IEEE Transactions on Software Engineering 27 (7) (2001) 577–598.
[18] L. Lin, Management of requirements changes in sequence-based software specifications, Ph.D. Thesis, University of Tennessee, 2006, http://sqrl.eecs.utk.edu/btw/files/lin.pdf.
[19] L. Lin, S.J. Prowell, J.H. Poore, The impact of requirements changes on specifications and state machines, Software – Practice and Experience 39 (6) (2009) 573–610.
[20] R.C. Linger, H.D. Mills, B.I. Witt, Structured Programming: Theory and Practice, Addison-Wesley, 1979.
[21] H.D. Mills, The new math of computer programming, Communications of the ACM 18 (1) (1975) 43–48.
[22] H.D. Mills, Stepwise refinement and verification in box-structured systems, IEEE Computer 21 (6) (1988) 23–36.
[23] D.L. Parnas, Y. Wang, The trace assertion method of module interface specification, Technical Report 89-261, Queens University, 1989.
[24] S.J. Prowell, J.H. Poore, Foundations of sequence-based software specification, IEEE Transactions on Software Engineering 29 (5) (2003) 417–429.
[25] S.J. Prowell, W.T. Swain, Sequence-based specification of critical software systems, in: Proceedings of the 4th American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls and Human-Machine Interface Technology, 2004.
[26] S.J. Prowell, C.J. Trammell, R.C. Linger, J.H. Poore, Cleanroom Software Engineering: Technology and Process, Addison-Wesley-Longman, 1999.
[27] A.W. Roscoe, Model-checking CSP, in: A Classical Mind: Essays in Honor of C.A.R. Hoare, Prentice Hall, 1994, pp. 353–378.