Applying string-rewriting to sequence-based specification

Robert Eschbach[a], Lan Lin[b,*], Jesse H. Poore[b]

[a]ITK Engineering AG, Luitpoldstraße 59, 76863 Herxheim, Germany
[b]Department of Electrical Engineering and Computer Science, The University of Tennessee, Min H. Kao Building, 1520 Middle Drive, Knoxville, TN 37996-3450, USA

Abstract

Sequence-based specification is a constructive method designed to convert ordinary functional requirements (that are often imprecisely and informally composed) into precise specifications. The method prompts a human requirements analyst to make the many decisions necessary to resolve the ambiguities, omissions, inconsistencies, and errors inherent in the original requirements document, and construct a complete, consistent, and traceably correct specification. We find that string-rewriting theory can be applied to make a number of these decisions automatically. In this paper we develop a theory of applying string-rewriting to sequence enumeration. We give prescriptions on how prefix rewrite rules and general string rewrite rules can be declared, and used later in the process to automatically make new equivalences thereby prompting the human for fewer decisions. Based on the results we present an enhanced enumeration process, in which one develops working enumerations and working reduction systems concurrently, applying string-rewriting to deduce new reductions as needed, until a complete enumeration is obtained. We present data from four published applications that shows the feasibility and applicability of applying string-rewriting. In addition to effort reduction we have observed the benefit of eliminating rework achieved by consistent decisions, as well as an additional opportunity string-rewriting provides for validation of specification decisions to requirements.

Keywords:
String-rewriting, Prefix string-rewriting, Sequence-based specification, Software specification, Abstract reduction system, Requirements engineering, Requirements elicitation

1. Introduction

Software development often starts with some form of functional *requirements*: ideas, verbal descriptions, documents, tables, charts, equations, diagrams, predecessor systems, competitor systems, or combinations of these. Generally, they contain ambiguities, omissions, and errors, hence are inconsistent, incomplete, and strictly speaking, incorrect. The *sequence-based specification* method [1, 2, 3] provides a systematic way to convert imprecise (and usually informal) requirements into precise software specifications at an early stage in the development cycle. The method treats discrete systems, and systems modeled as discrete based on abstractions of events.

*Corresponding author
Email address: `lin@eecs.utk.edu` (Lan Lin)

These specifications are important for later phases including both code development and testing [2, 4, 5, 6, 7, 8].

The method derives a rigorous specification from imperfect starting requirements through a constructive *sequence enumeration* process. The derivation exposes errors and omissions in the original requirements, which must be resolved by domain experts from authoritative sources. In this process, an effort is made by the specification writer to consider all possible scenarios of system use (all use cases). It proceeds by enumerating (in length-lexicographical order) finite sequences of elemental inputs (at some level of abstraction), deciding what the correct outputs of the system should be for the enumerated sequences according to the requirements, and identifying equivalences to earlier enumerated sequences based on the outputs to be generated on future inputs. The process terminates when there are no more sequences to enumerate by the enumeration rules. The primary product is a precise specification for the programmer and the tester. One by-product is documentation of the interpretation of requirements with traces to the authority for all the decisions made during specification.

Application of the method is facilitated and simplified with a prototype enumeration tool developed by the Software Quality Research Laboratory (SQRL) at the University of Tennessee [9]. A detailed description of the tool can be found in [10]. To produce a sequence-based specification in the tool, one only needs to give stimuli (inputs) and responses (outputs) short names at the beginning to facilitate enumeration; no other notation or syntax is required. The tool enforces enumeration rules explicitly by our recommended workflow and prompts the user for next steps. It maintains internal files (XML format) current with every action.

Enumeration of usage scenarios discovers a state machine of the intended system for implementation (the process terminates when all distinct states of the system are discovered). It prompts and relies on a human specifier (or requirements analyst) to make the many decisions (regarding response mappings, equivalence declarations, and authority) necessary to construct a specification. In application, a significant number of such decisions are observed as occurring in patterns. For example, some pairs of inputs commute with respect to sequencing; some are idempotent. More complex patterns may involve three or more inputs. These were previously recorded as a side issue and then checked against requirements to see if the patterns made sense, and against the enumeration to see if each instance had been treated consistently. These observations led to the systematic study of applying string-rewriting to sequence-based specification presented in this paper. The theory is implemented in our enumeration tool. Now more equivalence declarations can be handled automatically and consistently leading to fewer human errors. The result is an enhanced enumeration process.

In order to gain intuition for the process, we considered the following published applications:

- *Satellite Operations Software (SOS)*: the software component of a space vehicle that processes commands from the ground control system and supplies half-duplex communications between an uplink ground site and a downlink ground site [2]

- *Mine Pump Controller Software (MPCS)*: the control software of a mine pump that detects the water level, monitors the carbon monoxide, methane and airflow levels, and operates the pump with assistance from human operators [11]

- *Weigh-In-Motion Data Acquisition Processor (WIMDAP)*: the software for data acquisition used in a weigh-in-motion distributed system that acquires and processes data from individual load cells, performs real-time monitoring of the analog weight signal, and communicates asynchronously with the host computer [12]

Application	SOS	MPCS	WIMDAP
Number of stimuli	23	10	14
Terminating enumeration length	9	5	4
Sequences extended	11	22	13
Sequences analyzed	254	265	219
Potential sequences	1,883,023,236,984	111,111	41,371

Table 1: Sequences analyzed in the three case studies.

We constructed enumerations for the three case studies (see [13, 14, 15]) with the result shown in Table 1. From the table the effectiveness of enumeration in controlling the combinatorial growth of the number of sequences that need to be examined can be observed.

The crux of the matter is that an input set of size n will require consideration of $\Sigma_{i=0}^{k} n^i = (n^{k+1} - 1)/(n - 1)$ sequences, where k is the sequence length at which the enumeration process terminates (as it surely will). This need not be off-putting because there are mathematically sound ways to mitigate the combinatorial growth.

The method still requires hard work from the human specifier in understanding, clarifying, and eliciting requirements, but with a defined process and tool support the discovery of every detail of the state machine is by construction (rather than by intuition), and the traceability to the requirements makes it possible to verify its correctness. With a little tutoring software engineers, domain experts, business analysts, and customers can actively participate in the requirements analysis and specification process.

The paper is organized as follows. Section 2 introduces our terminology and notation. Section 3 introduces definitions and results for abstract reduction systems, string-rewriting systems, and prefix string-rewriting systems based on the classical literature. Section 4 gives an overview of the sequence-based specification method and its process, followed by an axiomatic definition. It also links a complete and finite enumeration to a convergent reduction system by prefix string-rewriting. In Section 5 we develop a theory of applying string-rewriting to sequence enumeration, give prescriptions on how prefix rewrite rules and general string rewrite rules can be declared, and present an enhanced enumeration process with application of string-rewriting. Section 6 is the case study in which we illustrate our theory with an automobile mirror control unit example, and discuss data obtained from applying the theory to the case studies of Table 1. Section 7 reports on related work, and Section 8 concludes this paper. The complete enumeration table for the mirror control example is given in the appendix.

2. Terminology and notation

We use $\mathbb{N}$ to denote the positive integers.

Let X be a set and $\rightarrow$ be a binary relation on X, then $\overset{*}{\rightarrow}$ is the reflexive and transitive closure of $\rightarrow$, and $\overset{*}{\leftrightarrow}$ the reflexive, symmetric, and transitive closure of $\rightarrow$.

The length of a string w is denoted by $|w|$. The empty string is denoted by λ. If Σ is a fixed (finite) alphabet, then Σ^* and Σ^+ denote the Kleene closure and the positive closure of Σ, respectively.

Unless stated explicitly as partial, a function $f : X \rightarrow Y$ is total (or complete). When f is partial, $dom f$ 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 $u \underset{f}{\triangleright} v$. The f will be dropped where it is clear from context. Define $u \not\mapsto r \leftrightarrow \exists r' \neq r.\, u \mapsto r'$.

3. String-rewriting system

The following definitions and results for string-rewriting systems are based on [16].

3.1. Abstract reduction system

Definition 3.1. An **abstract reduction system** (or a **reduction system**) is a structure $(X, \to)$, where X is a set and $\to$ is a binary relation on X. The relation $\to$ is the **reduction relation**. If $x \to y$ for some y, then x is **reducible**; otherwise, it is **irreducible**. If $x \overset{*}{\leftrightarrow} y$ for irreducible y, then y is a **normal form** for x.

Definition 3.2. A reduction system $(X, \to)$ is **confluent** if for all $w, x, y \in X$, $w \overset{*}{\to} x$ and $w \overset{*}{\to} y$ imply that there exists a $z \in X$, $x \overset{*}{\to} z$ and $y \overset{*}{\to} z$. It is **noetherian** if there is no infinite sequence $(x_1, x_2, \cdots)$ such that for all $i \in \mathbb{N}$, $x_i \to x_{i+1}$. It is **convergent** if it is both noetherian and confluent.

The words "noetherian" and "terminating" are both used in literature to describe reduction systems in which there exist no infinite reduction chains. We also use them interchangeably throughout the paper.

Theorem 3.3. *Let $R = (X, \to)$ be a reduction system. If R is convergent, then each $x \in X$ has a unique normal form.*

Proof. See Page 13, Theorem 1.1.12 in [16]. □

3.2. String-rewriting system

Definition 3.4. Let Σ be an alphabet. A **string-rewriting system** $\vdash$ on Σ is a subset of $\Sigma^* \times \Sigma^*$. Each element (l, r) of $\vdash$ is a **rewrite rule**, also written as $l \vdash r$. The **single-step reduction relation** $\to_\vdash$ on Σ^* that is induced by $\vdash$ is defined as follows: for any $u, v \in \Sigma^*$, $u \to_\vdash v$ iff there exists $l \vdash r$ such that for some $x, y \in \Sigma^*$, $u = xly$ and $v = xry$.

If $\vdash$ is a string-rewriting system on Σ, then $(\Sigma^*, \to_\vdash)$ is a reduction system.

Definition 3.5. Let Σ be an alphabet and R be a binary relation on Σ^*. Then R is a **partial order** if it is reflexive, antisymmetric, and transitive. It is a **total order** if it is a partial order and if, for all $x, y \in \Sigma^*$, either xRy, or yRx. It is a **strict partial order** if it is irreflexive, transitive, and therefore asymmetric. The relation R is **admissible** if for all $u, v, x, y \in \Sigma^*$, uRv implies $xuyRxvy$.

Each partial order $\preceq$ has an associated strict partial order $\prec$ defined by: $x \prec y$ iff $x \preceq y$ and $x \neq y$. Conversely, each strict partial order $\prec$ has an associated partial order $\preceq$ defined by: $x \preceq y$ iff $x \prec y$ or $x = y$.

Each total order $\preceq$ has an associated asymmetric (hence irreflexive) relation $\prec$, called a *strict total order* defined by: $x \prec y$ iff $x \preceq y$ and $x \neq y$. A strict total order is a strict partial order.

If X is a finite set equipped with a strict total order $\prec$, then we use $min(X)$ and $max(X)$ to denote the smallest and the greatest elements of X according to $\prec$, respectively.

Definition 3.6. Let $\prec$ be a strict partial order on Σ^*. It is **well-founded** if there is no infinite sequence $(x_1, x_2, \cdots)$ such that for all $i \in \mathbb{N}$, $x_{i+1} \prec x_i$.

Theorem 3.7. *Let $\vdash$ be a string-rewriting system on Σ. Then the reduction system $(\Sigma^*, \rightarrow_{\vdash})$ is noetherian iff there exists an admissible well-founded strict partial order $\prec$ on Σ^* such that $r \prec l$ holds for each $l \vdash r$.*

Proof. See Pages 42-43, Theorem 2.2.4 in [16]. $\square$

3.3. Prefix string-rewriting system

Definition 3.8. Let Σ be an alphabet. A **prefix string-rewriting system** $\models$ on Σ is a subset of $\Sigma^* \times \Sigma^*$. Each element (l, r) of $\models$ is a **prefix rewrite rule**, also written as $l \models r$. The **single-step reduction relation** $\rightarrow_{\models}$ on Σ^* that is induced by $\models$ is defined as follows: for any $u, v \in \Sigma^*$, $u \rightarrow_{\models} v$ iff there exists $l \models r$ such that for some $y \in \Sigma^*$, $u = ly$ and $v = ry$.

If $\models$ is a prefix string-rewriting system on Σ, then $(\Sigma^*, \rightarrow_{\models})$ is a reduction system.

4. Sequence-based specification

A software program implements the mapping rule for a mathematical function called the *black box function* [17]. Given the same inputs received in the same order (the same history of inputs) the program will yield the same output. Sequence-based specification converts functional requirements into a precise *black box specification* that describes the black box function of a system, namely, the system's behavior solely in terms of external input histories and outputs. The completed specification defines a mapping rule that uniquely determines an output for any finite sequence of inputs. The specification is constructed systematically through a process called sequence enumeration.

The purpose of this section is to introduce the sequence-based specification method and its axiomatic foundation presented in [1, 3, 18], upon which we build a theory for applying string-rewriting to sequence enumeration. Section 4.3 is new in this paper.

4.1. Process overview

The first step in applying sequence-based specification is to identify a *system boundary* that defines what is inside and what is outside the system. It consists of *interfaces*, through which information flows between the system and the external entities with which the system directly communicates, the software's *environment*.

Events (inputs, interrupts, invocations) in the environment that can affect system behavior are called *stimuli*. System behaviors observable in the environment are called *responses*. Stimuli and responses are collected for the identified system interfaces. We use S and R to denote the stimulus set and the response set, respectively. One then enumerates all stimulus sequences (that represent all scenarios of use of the system) first in the increasing order of length, and within the same length by any arbitrary strict total order.

For each enumerated sequence one identifies a unique response (which could be an ensemble of responses) as the intended response generated by the sequence. To facilitate theory we introduce two special responses: *null* (denoted by the symbol 0) and *illegal* (denoted by the symbol ω). If the sequence generates no externally observable behavior, it is mapped to 0; if the sequence is not physically realizable, it is mapped to ω; otherwise, the sequence is mapped to an observable response. A sequence is *illegal* when it maps to ω; otherwise, it is *legal*.

For each enumerated sequence, one also checks for *Mealy equivalence* between the current sequence and all previously enumerated sequences. Two sequences are *Mealy equivalent* if and only if they always generate the same response when extended by the same (non-empty) input sequence. Two Mealy equivalent sequences need not be mapped to the same response, but their responses with respect to future extensions must always agree. Mealy equivalent sequences represent the same state of the system when it is modeled as a Mealy machine (hence the name). If a sequence is not Mealy equivalent to any previously enumerated sequence, it is *unreduced*; otherwise, it is *reduced* to the previously enumerated (Mealy equivalent) sequence that is itself unreduced.

One starts with the empty sequence (denoted by λ) of length 0. To get all sequences of length $n+1$ ($n \geq 0$ is an integer) one extends all sequences of length n by every stimulus, and considers the extensions in the pre-defined strict total order (e.g., the lexicographical order). With the finitely many stimuli for any real application, there is an infinite number of stimulus sequences (of finite length); enumerating all of them is never necessary.

This inherently combinatorial process can be effectively controlled by two observations:

- If sequence u is reduced to a prior sequence v, there is no need to extend u by any stimulus for the next enumeration length, as the behaviors of the extensions are fully defined by the same extensions of v.

- If sequence u is illegal, there is no need to extend u by any stimulus for the next enumeration length, as all of the extensions must be illegal (i.e., physically unrealizable).

Therefore, only legal and unreduced (also called *extensible*) sequences of length n get extended by every stimulus for consideration at length $n+1$. The process continues until all sequences of a certain length are either illegal or reduced to prior sequences. Now, the enumeration is *complete* and *finite*; all extensible sequences have been extended. This terminating length is discovered in enumeration, and varies from application to application.

As the name suggests, sequence enumeration is the literal enumeration of stimulus sequences, the assignment of correct responses to each enumerated sequence, and the recording of sequence equivalences based on future behavior. The unreduced sequences represent essential states in the implemented system, whose number must be finite (and small) for any real application.

Table 2 shows the major steps of the enumeration process, with each step classified as either manual (work done by the human analyst) or automated (work performed automatically) or both.

The result is a fully documented, complete, and consistent black box specification. From it we can automatically generate a Mealy machine (in the form of a set of state box tables, or collectively a *state box specification*). We can also generate the control flow code by implementing the state box tables.

4.2. Enumeration

We now look at an enumeration solely as a mathematical object, and characterize it with a list of axioms regardless of whether the sequences were obtained by the enumeration process or in some other way.

Following [18], in the definitions below S and R are for the stimulus set and the response set, respectively.

Definition 4.1. 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 Axioms 1-6 hold for all u, v in S^* and x in S:

Step 1:	Tag (Number) the requirements.	manual
Step 2:	Define the interfaces that compose the system boundary.	manual
Step 3:	Define the stimuli associated with each interface.	manual
Step 4:	Define the responses associated with each interface.	manual
Step 5:	Start sequence enumeration with length $n = 0$. λ is mapped to 0 and unreduced.	automated
Step 6:	Repeat Steps 7-11 below until the enumeration is complete (i.e., until there are no extensible sequences of length n).	both
Step 7:	Extend all the extensible sequences of length n by every stimulus. List all the extensions in lexicographical order.	automated
Step 8:	Repeat Steps 9-10 below until all the sequences on the list (of length $n + 1$) have been considered (in lexicographical order).	both
Step 9:	Map the current sequence under consideration to a response (this may lead to identification of a new response). Trace the response decision to the tagged requirements (this may lead to discovery of derived requirements).	both
Step 10:	If the current sequence is Mealy equivalent to a prior unreduced sequence, reduce the current sequence to the prior sequence. Trace the equivalence decision to the tagged requirements (this may lead to discovery of derived requirements).	both
Step 11:	Increment n by 1.	automated

Table 2: Major steps of the enumeration process without string-rewriting support.

Axiom 1. $\lambda \mapsto 0$

Axiom 2. $u \triangleright v$ implies $v \prec u$ or $v = u$

Axiom 3. $u \triangleright v$ implies $v \triangleright v$

Axiom 4. $ux \in dom\,\mathcal{E}$ implies $u \triangleright u$

Axiom 5. $ux \in dom\,\mathcal{E}$ implies $u \not\mapsto \omega$

Axiom 6. $u \triangleright v, u \mapsto \omega, v \not\mapsto \omega, vx \in dom\,\mathcal{E}$ imply $vx \mapsto \omega$.

An enumeration $\mathcal{E} : S^* \to R \times S^*$ is **complete** iff Axiom 7 holds for all u in S^* and x in S:

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

An enumeration $\mathcal{E} : S^* \to R \times S^*$ is **finite** iff

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

Example 4.2. Let $\mathcal{E} : S^* \to R \times S^*$ be a partial function, where $S = \{a, b\}$, $R = \{0, \omega, r\}$, S^* has a strict total order $\prec$ defined by

$$\forall u, v \in S^*.\, |u| < |v| \to u \prec v$$
$$\forall u \in S^*.\, ua \prec ub$$
$$\forall u, v \in S^*.\, \forall x, y \in S.\, u \prec v \to ux \prec vy,$$

and

Sequence	Response	Equivalence
λ	0	λ
a	r	a
b	ω	a
aa	ω	a
ab	ω	a

Table 3: $\mathcal{E}$ of Example 4.2 in tabular form.

$$\begin{aligned}
\mathcal{E}(\lambda) &= (0, \lambda) \\
\mathcal{E}(a) &= (r, a) \\
\mathcal{E}(b) &= (\omega, a) \\
\mathcal{E}(aa) &= (\omega, a) \\
\mathcal{E}(ab) &= (\omega, a).
\end{aligned}$$

It is easily checked that $\mathcal{E}$ as defined satisfies Axioms 1-8 for a complete and finite enumeration. The same enumeration can be obtained in tabular form (see Table 3) following the process described earlier (for this symbolic example we ignore all the traces to the requirements). Here $dom\,\mathcal{E} = \{\lambda, a, b, aa, ab\}$ contains all and the only sequences in S^* that are actually enumerated.

Definition 4.3. Let $\mathcal{E} : S^* \to R \times S^*$ be an enumeration and $\prec$ be the associated strict total order on S^*. Then u is **illegal** iff $u \mapsto \omega$; u is **legal** iff $u \not\mapsto \omega$; u is **unreduced** iff $u \triangleright u$; u is **reduced** iff $u \triangleright v$ for $v \prec u$; u is **extensible** iff u is both legal and unreduced.

Example 4.4. Referring to $\mathcal{E}$ in Example 4.2, λ and a are both legal and unreduced (hence extensible); b, aa, and ab are both illegal and reduced.

Lemma 4.5. *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. By Axioms 1, 4, and 5. □

Lemma 4.6. *Let $\mathcal{E} : S^* \to R \times S^*$ be an enumeration. Then $u \in dom\,\mathcal{E}$, $u \neq \lambda$ imply for every proper prefix v of u, $v \triangleright v$.*

Proof. By Axiom 4. □

Definition 4.7. 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 $BB(\lambda) = 0$.

By Definition 4.7 a black box function maps every stimulus sequence to a response, with the empty sequence mapped to the null response. The requirements document always implies a black box function for the system to be developed, referred to as the *intended* black box function of the system. Sequence enumeration is performed to discover this intended black box function. With a completed enumeration this function can be computed algorithmically [3, 10, 18].

Twelve atomic algorithms for managing changes in the enumeration have been derived. All changes in requirements or an enumeration that come from resolving errors, omissions, and inconsistencies, as well as from outside the enumeration process (e.g., feature changes) can be made by a combination of the atomic change algorithms. In each case the tool makes all changes that are mathematically certain, and highlights all sequences that require reconsideration by analysts [10].

Definition 4.8. Let $\mathcal{E} : S^* \to R \times S^*$ be an enumeration, and $BB : S^* \to R$ be the intended black box function according to the requirements. Then $\mathcal{E}$ is **minimal** iff the following holds for all u, v in S^*:

$$u \triangleright u,\ v \triangleright v,\ u \neq v \text{ imply there exists a } w \text{ in } S^+ \text{ such that } BB(uw) \neq BB(vw).$$

An enumeration is minimal if the Mealy equivalence relation does not hold for any two different unreduced sequences.

Example 4.9. Referring to $\mathcal{E}$ in Example 4.2, λ and a are the only two unreduced sequences. Since $BB(\lambda a) = BB(a) = r$, $BB(aa) = \omega$, $\mathcal{E}$ is minimal.

4.3. Reduction system

A complete and finite enumeration $\mathcal{E} : S^* \to R \times S^*$ together with its associated strict total order $\prec$ on S^* defines a prefix string-rewriting system $\models$ on S as follows:

- If $u \triangleright v$, $v \prec u$, then $u \models v$.
- If $u \mapsto \omega$, $u \triangleright u$, then for all $x \in S$, $ux \models u$.

(1c)

Since $\mathcal{E}$ is finite, $\models$ contains finitely many prefix rewrite rules. The rules will be referred to as (1c) for “the first set of rules for complete and finite enumerations”.

Example 4.10. Referring to $\mathcal{E}$ in Example 4.2, it defines a prefix string-rewriting system $\models = \{(b, a), (aa, a), (ab, a)\}$. Here all the prefix rewrite rules happen to follow from the first rule in (1c).

Theorem 4.11. *Let $\models$ be the prefix string-rewriting system on S defined by (1c), then the induced reduction system $(S^*, \to_{\models})$ is convergent.*

Proof. The set of unreduced sequences in $\mathcal{E}$ is prefix-closed, that is, each prefix of an unreduced sequence must also be an unreduced sequence. The result follows from Theorem 14 and Theorem 22 of [19]. □

Corollary 4.12. *Let $\models$ be the prefix string-rewriting system on S defined by (1c). Then each u in S^* has a unique normal form in the induced reduction system $(S^*, \to_{\models})$. The normal forms compose $\{u : u \triangleright u\}$.*

Proof. Each u in S^* has a unique normal form by Theorems 3.3 and 4.11. Furthermore, u derives an unreduced sequence in $\mathcal{E}$ in finitely many steps, to which no prefix rewrite rules apply. Hence each normal form must be an unreduced sequence, and vice versa. □

The infinite set S^* is partitioned into finitely many blocks by the equivalence relation $\overset{*}{\leftrightarrow}_{\models}$, with unreduced sequences of $\mathcal{E}$ being representatives of these blocks. Each block corresponds to a system state. Transitions between states are determined by the reduction function $\triangleright$ of $\mathcal{E}$, or equivalently, the rules of (1c).

5. Applying string-rewriting to sequence enumeration

In this section we develop a theory of applying string-rewriting to sequence enumeration. We define a working enumeration as the mathematical representation of a partially completed (enumeration) work product, and pay special attention to a subset of working enumerations in which sequences are enumerated length-lexicographically. We give prescriptions on how one should declare prefix rewrite rules and string rewrite rules, and define working reduction systems for this subset of working enumerations. We prove that a working reduction system must be noetherian. Although it may not be confluent, the next sequence to be enumerated has a unique normal form. Based on the results we present a process to develop working enumerations and working reduction systems concurrently, applying string-rewriting to deduce new reductions as needed, until a complete enumeration is obtained. The theory presented in this section is new in this paper.

5.1. Working enumeration

Definition 5.1. Let $\mathcal{E} : S^* \to R \times S^*$ be a finite, minimal enumeration. Let $\prec$ be the associated strict total order on S^*. Then $\mathcal{E}$ is a **working enumeration** iff the following holds for all u in S^* and x in S:

$$u \not\mapsto \omega,\ u \triangleright u,\ ux \notin dom\,\mathcal{E} \text{ imply } max(dom\,\mathcal{E}) \prec ux.$$

In a working enumeration, unless the extension is greater than the greatest enumerated sequence so far, an extensible sequence is extended. By this definition, a complete, finite, and minimal enumeration is also a working enumeration, with all extensible sequences being extended. We define below for a working but not complete enumeration the next sequence to be enumerated.

Definition 5.2. Let $\mathcal{E} : S^* \to R \times S^*$ be a working enumeration that is not complete and $\prec$ be the associated strict total order on S^*. Then $next(\mathcal{E})$ is the smallest element of the set $\{ux : u \not\mapsto \omega, u \triangleright u, x \in S, ux \notin dom\,\mathcal{E}\}$ according to $\prec$.

Example 5.3. Let $\mathcal{E} : S^* \to R \times S^*$ be a finite and minimal enumeration, where $S = \{a, b\}$, $R = \{0, \omega, r\}$, S^* has a strict total order $\prec$ defined by

$$\begin{array}{l} \forall u, v \in S^*.\, |u| < |v| \to u \prec v \\ \forall u \in S^*.\, ua \prec ub \\ \forall u, v \in S^*.\, \forall x, y \in S.\, u \prec v \to ux \prec vy, \end{array}$$

and

$$\begin{array}{lcl} \mathcal{E}(\lambda) & = & (0, \lambda) \\ \mathcal{E}(a) & = & (r, a) \\ \mathcal{E}(b) & = & (\omega, a) \\ \mathcal{E}(aa) & = & (\omega, a). \end{array}$$

It is easily checked that $\mathcal{E}$ is a working enumeration, because $u = a$ and $x = b$ are the only values of u and x such that the premise of the implication is true, and the conclusion is also true ($max(dom\,\mathcal{E}) = aa \prec ab = ux$), hence the implication is true for $u = a$ and $x = b$. For all the other values of u and x the premise is false, hence the implication is true. Therefore, the implication holds for all u in S^* and for all x in S. Observe that $next(\mathcal{E}) = ab$.

A set of prefix rewrite rules and a prefix string-rewriting system follow similarly for a working enumeration as for a complete and finite enumeration (Section 4.3). Let $\mathcal{E} : S^* \to R \times S^*$ be a working enumeration. The following (prefix rewrite) rules are defined:

- If $u \triangleright v$, $v \prec u$, then $u \models v$.
- If $u \mapsto \omega$, $u \triangleright u$, then for all $x \in S$, $ux \models u$.

(1w)

The rules will be referred to as (1w) for "the first set of rules for working enumerations".

Example 5.4. Referring to $\mathcal{E}$ in Example 5.3, it defines a prefix string-rewriting system $\models\, = \{(b, a), (aa, a)\}$. Here all the prefix rewrite rules happen to follow from the first rule in (1w).

Theorem 5.5. *Let $\models$ be the prefix string-rewriting system on S defined by (1w), then the induced reduction system $(S^*, \to_{\models})$ is convergent.*

Proof. The left-hand-side (LHS) of any prefix rewrite rule is a one-symbol extension of an unreduced sequence in $\mathcal{E}$. Among the LHSs of all prefix rewrite rules, no string is a prefix of any other string, hence at most one rule applies for each step of prefix string-rewriting. The induced reduction system is confluent.

Given any u in S^*, in finite steps it either rewrites to some unreduced sequence in $\mathcal{E}$, or it rewrites to $u'xw$, where u' is an extensible sequence in $\mathcal{E}$, $x \in S$, $w \in S^*$, $u'x \notin dom\,\mathcal{E}$. The induced reduction system is noetherian.

Therefore, the induced reduction system is convergent. □

By Theorem 5.5, any u in S^* still has a unique normal form after prefix string-rewriting, with a working enumeration $\mathcal{E} : S^* \to R \times S^*$. However, when $\mathcal{E}$ is not complete, the state machine of the system is only partially discovered. As a result, the number of blocks in the partition of S^*, hence the number of normal forms, is infinite. In Example 5.4 the induced reduction system by prefix string-rewriting contains an infinite number of normal forms, as any string beginning with *ab* is a normal form.

Definition 5.6. Let $\mathcal{E} : S^* \to R \times S^*$ be a working enumeration, and $\prec$ be the associated strict total order on S^*. If $\prec$ is the (pre-determined and fixed) length-lexicographical order, then $\mathcal{E}$ is an **ordered working enumeration**.

Example 5.7. $\mathcal{E}$ in Example 5.3 is an ordered working enumeration, since $\prec$ as defined is the length-lexicographical order.

5.2. Declaring string rewrite rules

A working enumeration is the mathematical abstraction of a partially completed work product during the enumeration process. It defines a partially discovered state machine. In discovering the known states and transitions, some string rewrite rules could be identified that reflect general structures of the entire state machine, which can be exploited later to discover the unknown parts, i.e., infer reductions to prior sequences for newly enumerated sequences.

The string rewrite rules must be generalized from already identified sequence reductions under the $\triangleright$ mapping. We consider two situations:

- A single reduction suggests a string rewrite rule.

  Suppose $u \triangleright v$, $u = w_1 l w_2$, $v = w_1 r w_2$ for $w_1, w_2 \in S^*$, then $l \vdash r$ is a potential string rewrite rule.

- Two reductions suggest a string rewrite rule.

  Suppose $u \triangleright w$, $v \triangleright w$, $u = w_1 l w_2$, $v = w_1 r w_2$ for $w_1, w_2 \in S^*$, then $l \vdash r$ is a potential string rewrite rule.

The string rewrite rules are defined as follows for a working enumeration $\mathcal{E} : S^* \to R \times S^*$ with intended black box function $BB : S^* \to R$ and referred to as (2w):

If the following hold

- either $u \triangleright v$ or $u \triangleright w$, $v \triangleright w$ for $u = w_1 l w_2$, $v = w_1 r w_2$, $w_1, w_2 \in S^*$
- r being length-lexicographically smaller than l
- $\forall u \in S^*. \forall v \in S^+. BB(ulv) = BB(urv)$,

then $l \vdash r$.

(2w)

Example 5.8. Let $\mathcal{E} : S^* \to R \times S^*$ be a working enumeration for $S = \{a, b, c\}$, $R = \{0, \omega, r\}$ intended to discover the state machine diagrammed in Figure 1. Let the associated strict total order $\prec$ on S^* be the length-lexicographical order induced by the alphabetical order on S, and

$$\begin{array}{lcl}
\mathcal{E}(\lambda) & = & (0, \lambda) \\
\mathcal{E}(a) & = & (0, a) \\
\mathcal{E}(b) & = & (r, a) \\
\mathcal{E}(c) & = & (0, c) \\
\mathcal{E}(aa) & = & (r, c) \\
\mathcal{E}(ab) & = & (0, c).
\end{array}$$

The reduction from b to a leads to the discovery of a string rewrite rule $b \vdash a$ after checking for all the conditions required by (2w). The reductions from aa to c and from ab to c lead to the discovery of two new string rewrite rules $aa \vdash c$, $ab \vdash c$, and an already discovered string rewrite rule $b \vdash a$.

Theorem 5.9. *Let $\vdash$ be the string-rewriting system on S defined by (2w), then the induced reduction system $(S^*, \to_\vdash)$ is noetherian.*

Proof. Since the length-lexicographical order on S^* is an admissible well-founded strict partial order, the result follows from Theorem 3.7. □

Although the induced reduction system $(S^*, \to_\vdash)$ is noetherian, it may not be confluent.

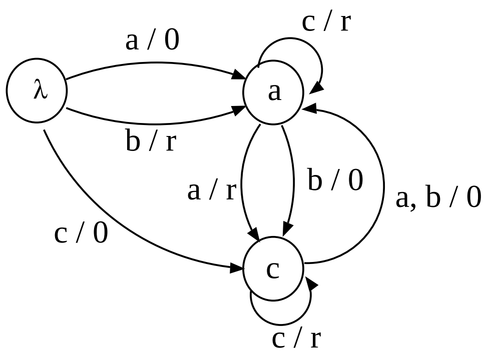

Figure 1: The state machine diagram to be discovered through the enumeration in Example 5.8.

5.3. Working reduction system

A working enumeration may have prefix rewrite rules defined by (1w) and string rewrite rules defined by (2w). If we combine these rules in a mixed reduction system, neither the noetherian nor the confluency property is guaranteed. However, if we restrict ourselves to ordered working enumerations, the noetherian property is guaranteed for the mixed reduction system, as proven in Theorem 5.11.

Definition 5.10. Let $\mathcal{E} : S^* \to R \times S^*$ be an ordered working enumeration. Let $\models$ be the prefix string-rewriting system on S defined by (1w), and $(S^*, \to_{\models})$ be the induced reduction system by prefix string-rewriting. Let $\vdash$ be the string-rewriting system on S defined by (2w), and $(S^*, \to_{\vdash})$ be the induced reduction system by string-rewriting. Then the reduction system defined by $(S^*, \to_{\models} \cup \to_{\vdash})$ is a **working reduction system**.

Theorem 5.11. *A working reduction system is noetherian.*

Proof. Consider a working reduction system as defined by Definition 5.10. Note that $r \prec l$ for each $l \models r$ defined by (1w) and each $l \vdash r$ defined by (2w), where $\prec$ is the associated length-lexicographical order on S^*. Since $\prec$ is admissible, $r \prec l$ for each $l \to_{\models} r$, and $r \prec l$ for each $l \to_{\vdash} r$. The theorem follows as $\prec$ is well-founded. □

Although confluency is not guaranteed, we can prove the next sequence to be enumerated must have a unique normal form. Hence a working reduction system can be used to predict what the next sequence to be enumerated should be reduced to, based on what is known about the system through the available prefix rewrite rules and string rewrite rules. If the unique normal form is different than the sequence itself, it suggests a sequence reduction. If the sequence itself and the derived unique normal form are identical, the human specifier takes over and considers any possible reduction based on Mealy equivalence.

Theorem 5.12. *Let $\mathcal{R} = (S^*, \to_{\models} \cup \to_{\vdash})$ be a working reduction system defined for an ordered working enumeration $\mathcal{E} : S^* \to R \times S^*$, where $\models$ and $\vdash$ are the prefix string-rewriting system and the string-rewriting system on S, respectively. Then $next(\mathcal{E})$ has a unique normal form.*

Proof. Suppose $next(\mathcal{E})$ has two normal forms u and v such that $u \neq v$.

Let ρ_{Me} denote the Mealy equivalence relation on S^*. By (1w) if $l \models r$ then $l \; \rho_{Me} \; r$, hence $lw \; \rho_{Me} \; rw$ for any w in S^*. We have $l \; \rho_{Me} \; r$ for each $l \rightarrow_{\models} r$. By (2w) if $l \vdash r$ then $w_1 l \; \rho_{Me} \; w_1 r$ for any w_1 in S^*, hence $w_1 l w_2 \; \rho_{Me} \; w_1 r w_2$ for any w_1, w_2 in S^*. We have $l \; \rho_{Me} \; r$ for each $l \rightarrow_{\vdash} r$. As a result, $next(\mathcal{E}) \, \rho_{Me} \, u \, \rho_{Me} \, v$.

Since reductions in $\mathcal{R}$ are only to prior sequences in length-lexicographical order, both u and v are prior to $next(\mathcal{E})$. Furthermore, u and v must be unreduced sequences of $\mathcal{E}$, as every sequence prior to $next(\mathcal{E})$ must reduce by prefix string-rewriting to an unreduced sequence of $\mathcal{E}$, from which no prefix or string rewrite rule could apply. Since $\mathcal{E}$ is minimal, we have $u = v$, a contradiction. □

Theorem 5.12 suggests that we can develop ordered working enumerations and working reduction systems concurrently through the process that follows, until a complete enumeration is constructed.

Concurrent Enumeration Process

```
Step 1:  Let $\mathcal{E}_0 = \{(\lambda, (0, \lambda))\}, \mathcal{R}_0 = (S^*, \rightarrow_0) = (S^*, \emptyset), i = 0$.
Step 2:  Repeat the steps below until $\mathcal{E}_i$ is complete.
Step 3:  Derive the (unique) normal form $v$ of $next(\mathcal{E}_i)$ in $\mathcal{R}_i$.
Step 4:  The human specifier defines the response $r$ of $next(\mathcal{E}_i)$.
Step 5:  If $v = next(\mathcal{E}_i)$, then the human specifier redefines $v$ such that $next(\mathcal{E}_i)$ is reduced to $v$ by Mealy equivalence and the enumeration rules. If $next(\mathcal{E}_i)$ cannot be reduced to any prior sequence, let $v = next(\mathcal{E}_i)$.
Step 6:  Let $\mathcal{E}_{i+1} = \mathcal{E}_i \cup \{(next(\mathcal{E}_i), (r, v))\}$.
Step 7:  Let $\mathcal{R}_{i+1} = (S^*, \rightarrow_{i+1})$, where $\rightarrow_{i+1} = \rightarrow_i$.
Step 8:  If $next(\mathcal{E}_i) \mapsto r$, $next(\mathcal{E}_i) \triangleright v$ define any prefix rewrite rule $l \models r$ by (1w), then $\rightarrow_{i+1} = \rightarrow_{i+1} \cup \{(ly, ry) : y \in S^*\}$.
Step 9:  If the human specifier identifies any string rewrite rule $l \vdash r$ by (2w) given $next(\mathcal{E}_i) \triangleright v$, then $\rightarrow_{i+1} = \rightarrow_{i+1} \cup \{(xly, xry) : x, y \in S^*\}$.
Step 10: Let $i = i + 1$.
```

Theorem 5.13. *The end product of the concurrent enumeration process (a set of sequences with responses and reductions) satisfies Axioms 1-8 for a complete and finite enumeration.*

Proof. By construction. □

Example 5.14. Let $\mathcal{E} : S^* \rightarrow R \times S^*$ be an ordered working enumeration for $S = \{a, b\}$, $R = \{0, \omega, r\}$ intended to discover the state machine diagrammed in Figure 2. Let the associated length-lexicographical order on S^* be based on the alphabetical order on S, and

$\mathcal{E}(\lambda)$	$=$	$(0, \lambda)$		
$\mathcal{E}(a)$	$=$	$(0, a)$		
$\mathcal{E}(b)$	$=$	(r, a)	$b \vdash a$	
$\mathcal{E}(aa)$	$=$	(r, aa)		
$\mathcal{E}(ab)$	$=$	$(0, aa)$		*using* $b \vdash a$
$\mathcal{E}(aaa)$	$=$	(r, a)	$aaa \vdash a$	
$\mathcal{E}(aab)$	$=$	$(0, a)$	$aab \vdash a$	*using* $b \vdash a$, $aaa \vdash a$.

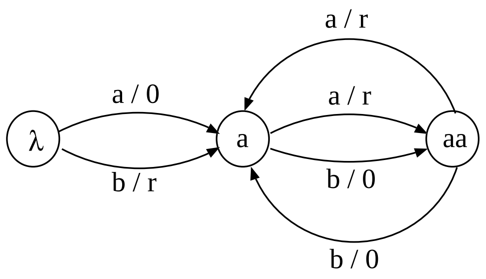

Figure 2: The state machine diagram to be discovered through the enumeration in Example 5.14.

Here we list the discovered prefix or string rewrite rules beside the corresponding reductions. If a reduction is automatically derived by prefix string-rewriting and/or string-rewriting, we also list the specific rules that have been used in the derivation. For instance, the reduction from b to a leads to the discovery of a string rewrite rule $b \vdash a$, which is used later to derive the reduced value of ab ($ab \rightarrow_{\vdash} aa$). The reduction from ab to aa leads to a prefix rewrite rule $ab \models aa$ and a string rewrite rule $ab \vdash aa$, but both are subsumed by the available string rewrite rule $b \vdash a$, hence omitted. The reduction from aaa to a leads to the discovery of a string rewrite rule $aaa \vdash a$, together with the earlier rule $b \vdash a$, it automatically derives the reduced value of aab ($aab \rightarrow_{\vdash} aaa \rightarrow_{\vdash} a$). The reduction from aab to a leads to a new string rewrite rule $aab \vdash a$ to be added to the set of discovered rules.

5.4. The impact of errors and changes

With the basic enumeration process, human errors and change decisions have similar impacts. The two basic changes are to a response decision or an equivalence decision. We urge that each decision in the enumeration process be associated with a citation to authority for that decision, and that the citation be changed when the decision is changed.

Likewise the introduction of string-rewriting introduces the following opportunities for human errors or changes:

- The human declares an equivalence between a pair of sequences that is later considered wrong (hence a wrong prefix rewrite rule).
- The human misses declaring an equivalence between a pair of sequences (hence a missed prefix rewrite rule).
- The human declares a string rewrite rule that is later considered wrong.
- The human misses an opportunity to declare a string rewrite rule.

Some are errors that must be corrected, while others are not and simply result in more work than might have been necessary. All changes result in rework.

The concurrent enumeration process can be adapted for enumerations that might contain such problems and the reduction systems they define. In Step 3 of the process more than one normal form could be found for the next sequence to be enumerated. We may insert a step afterwards to repeatedly revise (or fix) the enumeration and the reduction system until a unique normal form is derived for the next-to-be-enumerated sequence.

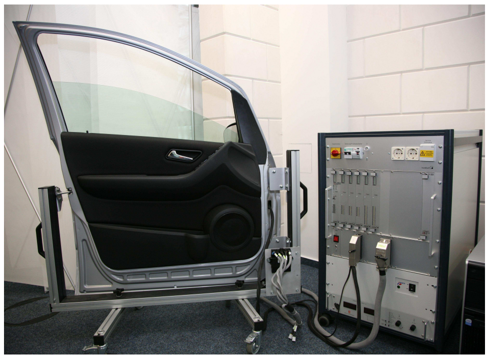

Figure 3: Functions of the driver side car mirror ECU in the testing environment.

This new step requires interactions with the atomic change algorithms [10], because changing a specific entry in the enumeration may necessitate changes to other entries. These changes may cascade to still more changes. While it is beyond human intuition to identify all possible changes, all the instances that must be reconsidered are flagged algorithmically. Details of the interactions are beyond the scope of this paper.

6. Example: Driver side car mirror electronic control unit

In this section we demonstrate how to apply string-rewriting to sequence enumeration through the example of an automobile mirror control unit. The example was taken from a case study that demonstrated a complete tool chain from requirements to sequence-based specification to automated statistical testing to quality certification [7]. It was modified slightly with a reduced set of requirements to make it fit the paper limit but still illustrate all features of the theory. Figure 3 shows the functions of the driver side car mirror electronic control unit (ECU) in the testing environment. The example was produced using our enumeration tool, which prompts the human at each step, enforces all theory, maintains complete documentation, checkpoints each step, and supports the change algorithms and string-rewriting. All the tables are presented in a format similar to the way information is maintained by the tool.

6.1. Requirements, system boundary, stimuli and responses

The original requirements for the driver side car mirror ECU are restated in Table 4, where each sentence from the original statements of requirements is numbered (tagged) for easy reference. When we enumerate, we trace our decisions regarding responses and reductions to the tagged requirements.

We identify a system boundary that cuts the interfaces between the system, the driver side car mirror ECU, and the external power, the CAN bus, sensors, actuators, and human users. The system boundary is diagrammed in Figure 4. The interfaces are listed in Table 5. From the interfaces we collect all possible stimuli (Table 6) and responses (Table 7) across the system boundary.

In identifying the interfaces and their associated stimuli and responses, as well as in constructing a complete enumeration of the mirror controller, we found that supplemental information was needed to give explicit authority for enumeration decisions regarding responses and

Trace Tag	Requirement
1	There is a switch that toggles for adjustment of either the driver side or the passenger side mirror.
2	The driver side electronic control unit (ECU) initializes when the car key is in start position.
3	The driver side ECU processes inputs coming from position sensors and users.
4	The driver side ECU produces outputs to actuators and sends messages to other ECUs.
5	The control area network (CAN) bus is used for communication among ECUs.
6	Signals for the passenger side mirror are put on the CAN bus and sent to the passenger side ECU.
7	Each mirror can be adjusted in the vertical and has extreme up and down positions.
8	Each mirror can be adjusted in the horizontal and has extreme inward and outward positions.
9	If requested movement cannot be made because the mirror is already in an extreme position, an error message is generated and sent via the CAN bus.

Table 4: Requirements for the driver side car mirror ECU.

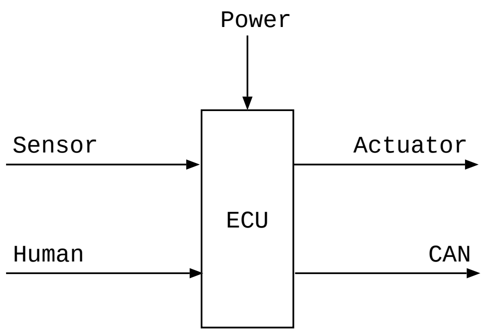

Figure 4: A system boundary for the driver side car mirror ECU.

Interface	Description	Trace
Actuator	The actuators	4
CAN	The CAN bus	4, 5
Human	The human users	3
Power	The power	2
Sensor	The position sensors	3

Table 5: Interfaces for the driver side car mirror ECU.

Stimulus	Long Name	Description	Interface	Trace
MHI	Mirror horizontal inward	The horizontal inward movement of selected mirror	Human	8
MHO	Mirror horizontal outward	The horizontal outward movement of selected mirror	Human	8
MVD	Mirror vertical down	The vertical down movement of selected mirror	Human	7
MVU	Mirror vertical up	The vertical up movement of selected mirror	Human	7
PHEI	Position horizontal extreme inward	The horizontal position report of the driver side mirror indicating the extreme inward position is reached	Sensor	8
PHEO	Position horizontal extreme outward	The horizontal position report of the driver side mirror indicating the extreme outward position is reached	Sensor	8
PHNE	Position horizontal no extreme	The horizontal position report of the driver side mirror indicating no extreme position is reached	Sensor	8
PVED	Position vertical extreme down	The vertical position report of the driver side mirror indicating the extreme down position is reached	Sensor	7
PVEU	Position vertical extreme up	The vertical position report of the driver side mirror indicating the extreme up position is reached	Sensor	7
PVNE	Position vertical no extreme	The vertical position report of the driver side mirror indicating no extreme position is reached	Sensor	7
SM	Switch mirror	The switch toggled for selection of the driver side or the passenger side mirror	Human	1
START	Start	Car key in start position	Power	2

Table 6: Stimuli for the driver side car mirror ECU.

Response	Long Name	Description	Interface	Trace
CERR	CAN mirror movement failure	An error message is generated and put on the CAN bus when the mirror is already in an extreme position and cannot make the requested movement.	CAN	9
CMHI	CAN mirror horizontal inward	A message is generated and put on the CAN bus for the passenger side ECU for the horizontal inward movement of the passenger side mirror.	CAN	6, 8
CMHO	CAN mirror horizontal outward	A message is generated and put on the CAN bus for the passenger side ECU for the horizontal outward movement of the passenger side mirror.	CAN	6, 8
CMVD	CAN mirror vertical down	A message is generated and put on the CAN bus for the passenger side ECU for the vertical down movement of the passenger side mirror.	CAN	6, 7
CMVU	CAN mirror vertical up	A message is generated and put on the CAN bus for the passenger side ECU for the vertical up movement of the passenger side mirror.	CAN	6, 7
HI	Horizontal inward movement	The horizontal inward movement of the driver side mirror	Actuator	8
HO	Horizontal outward movement	The horizontal outward movement of the driver side mirror	Actuator	8
VD	Vertical down movement	The vertical down movement of the driver side mirror	Actuator	7
VU	Vertical up movement	The vertical up movement of the driver side mirror	Actuator	7

Table 7: Responses for the driver side car mirror ECU.

Trace Tag	Derived Requirement
D1	It is physically impossible for the ECU to experience an input without power.
D2	There is no externally observable response across the system boundary when ignition is turned on.
D3	Mirror adjustment commands are ignored unless the position signal has been received.
D4	There is no externally observable response when mirror position signal is received.
D5	There is no externally observable response when the mirror selection switch toggles.
D6	Re-powering on makes previous history irrelevant.
D7	When ignition is turned on, the default mirror selection is on the driver side.
D8	When the mirror selection switch goes to the passenger side, any received or to-be-received driver side mirror position report will be ignored. Updated position signals are expected once the switch goes back to the driver side.

Table 8: Derived requirements for the driver side car mirror ECU.

equivalences. We improvised such information as "derived requirements" in Table 8, whose trace tags begin with a "D". They are subject to validation by application domain experts. During the enumeration we trace our decisions to both the original and the derived requirements.

6.2. Sequence enumeration

We apply the process presented in Section 5.3 to develop a complete enumeration of the mirror controller. The stimuli are alphabetically ordered in Table 6, based on which we enumerate stimulus sequences length-alphabetically. Stimuli are concatenated to string prefixes with periods.

The enumeration is performed in tabular form (see Table 12 in the appendix). We enumerate stimulus sequences under the "Sequence" column. Their mapped responses (by $\mapsto$) and reductions (by $\triangleright$) are defined under the "Response" and the "Equivalence" columns, respectively. Under the "Trace" column we trace the decisions regarding responses and reductions to the tagged requirements and derived requirements. We also note the discovered prefix or string rewrite rules under the "Rule" column. Let the rows of the table be indexed by i (i runs from 0 to 216; there are 217 rows in the complete enumeration). If we extract a subtable that includes all the rows from Row 0 up to and including Row i, then the subtable defines the working enumeration $\mathcal{E}_i$ as well as the working reduction system $\mathcal{R}_i$ described by the process (Section 5.3).

We label prefix rewrite rules $P_1, P_2, \ldots,$ and string rewrite rules $S_1, S_2, \ldots$. To avoid clutter, if a reduction $u \triangleright v$ itself expresses a prefix rewrite rule $u \models v$, or a string rewrite rule $u \vdash v$, we only put the label in the "Rule" column but leave the rule out. Whenever string-rewriting is applied to find a new reduction, we put a note below the reduction listing the rewrite rules used, and shade the corresponding row in the enumeration table.

To begin with, $\mathcal{E}_0$ corresponds to Row 0 of Table 12. The empty sequence, as the first enumerated sequence, maps to 0 because no stimuli have been received, and repeats itself in the "Equivalence" column because there is no prior sequence to reduce it to. The corresponding working reduction system $\mathcal{R}_0$ has no rules defined.

Since $next(\mathcal{E}_0)$ = MHI, its unique normal form in $\mathcal{R}_0$ is derived (it is MHI itself). The human specifier defines the response of MHI based on the requirements (the response is ω because of the derived requirement D1 in Table 8). Since MHI itself is a normal form, the human specifier takes over and checks if it can be reduced to a prior sequence based on Mealy equivalence (it turns out MHI cannot be reduced). Now $\mathcal{E}_1$ contains in addition the defined response and reduction for MHI and corresponds to Rows 0-1 of Table 12. From Row 1 we have a set of prefix rewrite rules by (1w). Rules $P_1 - P_{12}$ are thus defined in the "Rule" column.

Now $next(\mathcal{E}_1)$ = MHO, whose unique normal form in $\mathcal{R}_1$ is derived (it is MHO itself). Similarly the human specifier defines its response as ω by D1, and reduces it to MHI based on Mealy equivalence. Rows 0-2 define $\mathcal{E}_2$. Row 2 also leads to a new prefix rewrite rule P_{13} by (1w). Rules $P_1 - P_{13}$ are all the rules that define the working reduction system $\mathcal{R}_2$.

Continuing in this fashion, at Row 13 the first string rewrite rule S_1 is discovered and added to the set of rules to define the working reduction system $\mathcal{R}_{13}$. Here the string rewrite rule START.MHI $\vdash$ START subsumes the prefix rewrite rule START.MHI $\models$ START.

As illustrated by Table 12, we continue discovering new rewrite rules as we enumerate, extending a reduction system while we extend an enumeration. The rules accumulate until they are able to make a prediction of a future reduction, then string-rewriting is applied automatically to deduce the new reduction.

The first automatic sequence reduction is derived at Row 111. Using the rules that define the working reduction system $\mathcal{R}_{110}$, we have START.PHEI.PVED.MVD $\rightarrow_{\vdash}$ START.PHEI.PVED (here the string rewrite rule S_{35}: PVED.MVD $\vdash$ PVED is the only rule used to derive the normal form). In this mirror example, over half of the reductions for the length-four sequences (69 out of 108) are obtained through automatic string-rewriting. This accounts for almost a third of the total reductions (69/217) that need to be considered for the complete enumeration.

Once the rewrite rules are discovered, they can be checked against the requirements to see if correct decisions have been made, and against each other to see if such decisions have been made consistently throughout the specification. In Table 9 we organize some of the identified rules into groups, with similarly structured string rewrite rules put into the same group, and consider the semantics associated with each. Whenever a pattern can be extracted from a group of rules, we put it below the rules within the group.

For example, Group II shows a decision regarding receiving consecutive position reports in the same axis (either horizontal or vertical). Only the latest report is important (this can be validated with application domain experts). Since there are three reports for each axis, the number of such rules is $3 \times 3 \times 2 = 18$. The fact that Group II contains all the 18 rules demonstrates that this decision has been made consistently throughout the enumeration.

Note that the completed enumeration in Table 12 is not the end product of applying sequence-based specification. Our enumeration tool automatically generates from it black box tables and state box tables which collectively define a state machine with 19 states and 228 transitions between states. From the state box tables the control flow code can be (and is typically) generated by selecting a high-level software architecture, an implementation for stimulus gathering, an implementation for response generation, an implementation for state collection, and an implementation for each entry in the state box tables ([4] reported about a third of 20000 lines of code in a case study developing the control software embedded in a complex manufacturing machine). In addition, the graph of a Markov chain usage model can be derived from the state machine developed in the enumeration process with the arcs being annotated with execution probabilities to model the system usage. The usage model can then be enhanced by commands to drive automated test execution and evaluation.

I	S_6: PHEI.MHI $\vdash$ PHEI	S_{16}: PHEO.MHO $\vdash$ PHEO
	S_{35}: PVED.MVD $\vdash$ PVED	S_{48}: PVEU.MVU $\vdash$ PVEU
II	S_{10}: PHEI.PHEI $\vdash$ PHEI	S_{11}: PHEI.PHEO $\vdash$ PHEO
	S_{12}: PHEI.PHNE $\vdash$ PHNE	S_{19}: PHEO.PHEI $\vdash$ PHEI
	S_{20}: PHEO.PHEO $\vdash$ PHEO	S_{21}: PHEO.PHNE $\vdash$ PHNE
	S_{28}: PHNE.PHEI $\vdash$ PHEI	S_{29}: PHNE.PHEO $\vdash$ PHEO
	S_{30}: PHNE.PHNE $\vdash$ PHNE	S_{40}: PVED.PVED $\vdash$ PVED
	S_{41}: PVED.PVEU $\vdash$ PVEU	S_{42}: PVED.PVNE $\vdash$ PVNE
	S_{52}: PVEU.PVED $\vdash$ PVED	S_{53}: PVEU.PVEU $\vdash$ PVEU
	S_{54}: PVEU.PVNE $\vdash$ PVNE	S_{64}: PVNE.PVED $\vdash$ PVED
	S_{65}: PVNE.PVEU $\vdash$ PVEU	S_{66}: PVNE.PVNE $\vdash$ PVNE
	$x.y \vdash y$, where $x, y \in$ {PHEI, PHEO, PHNE}	
	$x.y \vdash y$, where $x, y \in$ {PVED, PVEU, PVNE}	
III	S_{13}: PHEI.SM $\vdash$ SM	S_{22}: PHEO.SM $\vdash$ SM
	S_{31}: PHNE.SM $\vdash$ SM	S_{43}: PVED.SM $\vdash$ SM
	S_{55}: PVEU.SM $\vdash$ SM	S_{67}: PVNE.SM $\vdash$ SM
	x.SM $\vdash$ SM, where $x \in$ {PHEI, PHEO, PHNE, PVED, PVEU, PVNE}	
IV	S_{37}: PVED.PHEI $\vdash$ PHEI.PVED	S_{38}: PVED.PHEO $\vdash$ PHEO.PVED
	S_{39}: PVED.PHNE $\vdash$ PHNE.PVED	S_{49}: PVEU.PHEI $\vdash$ PHEI.PVEU
	S_{50}: PVEU.PHEO $\vdash$ PHEO.PVEU	S_{51}: PVEU.PHNE $\vdash$ PHNE.PVEU
	S_{61}: PVNE.PHEI $\vdash$ PHEI.PVNE	S_{62}: PVNE.PHEO $\vdash$ PHEO.PVNE
	S_{63}: PVNE.PHNE $\vdash$ PHNE.PVNE	
	$x.y \vdash y.x$, where $x \in$ {PVED, PVEU, PVNE}, $y \in$ {PHEI, PHEO, PHNE}	
V	S_{69}: SM.MHI $\vdash$ SM	S_{70}: SM.MHO $\vdash$ SM
	S_{71}: SM.MVD $\vdash$ SM	S_{72}: SM.MVU $\vdash$ SM
	SM.$x \vdash$ SM, where $x \in$ {MHI, MHO, MVD, MVU}	

Table 9: Grouping rewrite rules based on structures and semantics.

6.3. Discussion

We can augment our theory and discover more string rewrite rules by taking into consideration sequences that do not show up in the enumeration table, or sequences with specific patterns. Here are two observations.

First, suppose $u \triangleright w$ in an ordered working enumeration $\mathcal{E} : S^* \to R \times S^*$ with length-lexicographical order $\prec$, $v \prec max(dom\,\mathcal{E})$ but $v \notin dom\,\mathcal{E}$, and v derives w in the corresponding working reduction system. For the intended black box function, if the following hold:

- $u = w_1 l w_2$, $v = w_1 r w_2$ for $w_1, w_2 \in S^*$
- $r \prec l$
- $\forall u \in S^*. \forall v \in S^+. BB(ulv) = BB(urv)$,

then $l \vdash r$ is a string rewrite rule.

In the mirror example instead of declaring

$$S_{33} : \text{START.PVED.MHI} \vdash \text{START.PVED}$$

we could have noticed that

$$\text{START.PVED.MHI} \triangleright \text{START.PVED}$$

and

$$\text{START.MHI.PVED} \to_{\vdash} \text{START.PVED},$$

and after checking that all the other conditions are satisfied, we could have declared a new string rewrite rule

$$S'_{33} : \text{PVED.MHI} \vdash \text{MHI.PVED},$$

which can be used in deriving the reductions for three length-four sequences as follows:

$$\begin{array}{lll} & \text{START.PHEI.PVED.MHI} & \\ \to_{\vdash} & \text{START.PHEI.MHI.PVED} & (\text{by} S'_{33} : \text{PVED.MHI} \vdash \text{MHI.PVED}) \\ \to_{\vdash} & \text{START.PHEI.PVED} & (\text{by} S_6 : \text{PHEI.MHI} \vdash \text{PHEI}) \\ \\ & \text{START.PHEO.PVED.MHI} & \\ \to_{\vdash} & \text{START.PHEO.MHI.PVED} & (\text{by} S'_{33} : \text{PVED.MHI} \vdash \text{MHI.PVED}) \\ \to_{\vdash} & \text{START.PVED} & (\text{by} S_{15} : \text{START.PHEO.MHI} \vdash \text{START}) \\ \\ & \text{START.PHNE.PVED.MHI} & \\ \to_{\vdash} & \text{START.PHNE.MHI.PVED} & (\text{by} S'_{33} : \text{PVED.MHI} \vdash \text{MHI.PVED}) \\ \to_{\vdash} & \text{START.PVED} & (\text{by} S_{24} : \text{START.PHNE.MHI} \vdash \text{START}). \end{array}$$

The string rewrite rule S'_{33} is discovered from an enumerated sequence START. PVED.MHI and a sequence that does not get enumerated START.MHI.PVED. Rules that could have been discovered by the same observation include S'_{34}: PVED. MHO $\vdash$ MHO.PVED, S'_{45}: PVEU.MHI $\vdash$ MHI.PVEU, S'_{46}: PVEU.MHO $\vdash$ MHO. PVEU, S'_{57}: PVNE.MHI $\vdash$ MHI.PVNE, and S'_{58}: PVNE.MHO $\vdash$ MHO.PVNE, each of which would lead to automatic reduction derivations for three length-four sequences.

Second, after declaring START.START $\triangleright$ START in Table 12, we could have noticed that this can be generalized to a rule scheme:

	With presented theory	With presented theory and observations
For length-four sequences only	$69/108 \approx 63.9\%$	$96/108 \approx 88.9\%$
For all the sequences in the enumeration	$69/217 \approx 31.8\%$	$103/217 \approx 47.5\%$

Table 10: Percentages of automatic sequence reductions by applying rewriting techniques.

Length	Sequences Extended	Sequences Analyzed	Reductions by String-Rewriting	Reductions by Humans	Potential Sequences
0	0	1	0	1	1
1	1	12	0	12	12
2	1	12	0	12	144
3	7	84	7	77	1,728
4	9	108	96	12	20,736
total	18	217	103	114	22,621

Table 11: Sequences analyzed in the car mirror ECU enumeration.

$$\text{START}.x_1.x_2.\cdots.x_n.\text{START} \vdash \text{START},$$

$$\text{where } x_i \text{ is any possible stimulus}, i, n \in \mathbb{N}, i \leq n.$$

It basically says that power-on resets the system and makes previous history irrelevant. Applying this rule scheme, the reductions could have been automatically derived for 16 sequences that are of length greater than two and end with the START stimulus.

With these observations almost 90% of the length-four sequences ($(69+18+9)/108 \approx 88.9\%$) and half of the total reductions ($(69 + 18 + 16)/217 \approx 47.5\%$) would be derived automatically by string-rewriting. Table 10 shows the data collected from this example.

6.4. Effectiveness of enumeration with string-rewriting

The most difficult part of doing an enumeration is identifying or recognizing reductions of sequences based on Mealy equivalence. With string-rewriting some reductions can be handled automatically. In Table 11 we list for the car mirror ECU example and for each enumeration length the number of sequences extended from the previous enumeration length, the actual number of sequences analyzed, as well as the potential number of sequences to be considered. For sequences that are actually analyzed, we record how many reductions are handled by string-rewriting and how many are handled by humans.

We also applied the concurrent enumeration process to the three case studies outlined in Table 1, and found that 17 out of 254 reductions for the satellite operations software, 47 out of 265 reductions for the mine pump control software, and 58 out of 219 reductions for the weigh-in-motion data acquisition processor were automatically derived by string-rewriting (for details see [13, 14, 15]). The benefit of applying string-rewriting depends on the application and the skill of the analyst. For instance, the small number of automatic reduction derivations for the satellite operations software is because the state machine is essentially a chain with little branching. In any case the discovered rewrite rules help articulate unstated patterns or facts that are implicit in the requirements and provide additional criteria for validating specification decisions to requirements.

7. Related work

Sequence-based specification emerged from the functional treatment of software as described by Mills [17, 20, 21]. The development was most directly influenced by the *trace assertion method* of Parnas [22, 23] and the algebraic treatment of regular expressions by Brzozowski [24].

In [22, 23] specifications are used as a reference document, and tabular forms are introduced to ease specification reading, writing, and checking. The trace assertion method was used to write module interface specifications following the "information hiding" principle, defining only the expected external properties of a module with no reference to design decisions regarding data representations or algorithms that are likely to change. A trace is a sequence of access procedure or function calls. A module's behavior is described in terms of traces and assertions about trace legality, equivalence of traces, and the return values of legal traces that end in function calls.

McLean [25] presented a formal foundation for trace specification by providing a syntax, semantics, and formal derivation system (a set of inference rules) by which assertions can be derived from trace specifications in a way that can be verified mechanically. The foundational framework is based on first-order logic with equivalence and legality defined as predicates. It is assumed that the empty trace is legal, the prefix of a legal trace is legal, and that only legal traces can return values. The author gave both syntactic and semantic definitions of consistency and totalness (completeness), and methods for proving specifications consistent and total, with the establishment of the soundness and completeness theorems.

A comparison between trace assertions and enumerations leads to the following observations:

- Legal traces are the event sequences that will not result in a non-normal use of the module. Legal stimulus sequences are those that are physically realizable.

- Trace equivalence is based on both current and future legality, and the return value for future program behavior. Equivalences among stimulus sequences are based on future behavior only.

- Assertions about traces can be written in an arbitrary order. Sequences of stimuli are considered in length-lexicographical order to enforce completeness and consistency.

- Non-determinism is allowed for trace specifications, and used for don't-care situations (i.e., situations in which the module user does not need to specify the behavior completely as several behaviors are equally acceptable, and one wants to give the implementor a choice), but the implementation would still be deterministic. Non-determinism in enumeration is treated mathematically as under-specified blocks of a partition in sequence-based specifications. This is the natural treatment since the method successively partitions the domain until all the states are identified as a block of the partition induced by the equivalence relation.

The trace assertion method was used on time-dependent systems like communication protocols [26]. As pointed out by Hoffman and Snodgrass [27], when used for complex modules it quickly becomes difficult to ensure specification consistency (whether the semantic assertions are mutually contradictory) and completeness (whether they completely characterize a module's behavior). To make reading and writing trace specifications for complex modules manageable, they proposed five heuristics including basing the specification on a normal form, and structuring the semantics (trace assertions) according to normal form prefixes. Normal form traces are representative traces for each equivalence class as determined by trace equivalence. The authors proposed a heuristic for choosing normal forms, and based the assertions on one-call extensions of normal form traces. This structure is similar to the one-symbol extensions of extensible sequences in the enumeration process for sequence-based specification. The difference is that the choice of normal form traces is free and heuristic, while the enumeration process forces unreduced sequences to be the smallest sequences in length-lexicographical order for each equivalence class. Canonical traces were introduced in [28, 29] as representatives of equivalence classes. The choice of canonical traces remains arbitrary, except that the empty trace is canonical.

Later work on the trace assertion method includes [30, 31, 19, 32]. One primary distinction of sequence-based specification is the constructive process it defines to discover a state machine of the system. In trace specifications, the *a priori* automaton is normally conceived by the specification writer through experience and insight, and described indirectly using traces and assertions about traces. Rewriting systems were studied for *deterministic* versions of the trace assertion method [28, 29, 31, 19]. The free choice of canonical traces in [28, 29], and a prefix-closed set of unreduced sequences by construction for any sequence-based specification (i.e., each prefix of an unreduced sequence must also be an unreduced sequence), manifest in the respective applications of rewriting.

The general trace rewriting used in [28, 29] is in essence prefix string-rewriting. The authors found it similar to conditional term rewriting. To address possibly non-terminating rewriting sequences, the general rewriting relation was modified (also for the purpose of simulating a trace specification). "Smart trace rewriting" was introduced to avoid unfruitful rewriting steps, resulting in a constrained prefix string-rewriting system that is both terminating and confluent. Another trace rewriting strategy, called stepwise rewriting, was introduced to support online simulation. Stepwise rewriting locally employs smart rewriting.

Trace rewriting systems in [31, 19] transform any input word of a connected semiautomaton (an automaton with possibly infinite states and inputs and no final states, in which every state is reachable from the initial state) to its canonical form algorithmically. Directly from a set of generators for state equivalence, the authors constructed a confluent prefix string-rewriting system. In general the rewriting system may allow infinite derivations with an arbitrary set of canonical words (traces) chosen for every state. It is proved that if one imposes the condition of prefix-continuity on the set of canonical words, the prefix string-rewriting system becomes terminating. Since the semiautomaton may contain an infinite number of states, there might be infinitely many prefix rewrite rules. The authors further connected such prefix string-rewriting systems to ground term rewriting systems, viewing the former as a special case of the latter.

The use of prefix-continuous canonical languages is crucial to the well-behaved rewriting system. A set of words is prefix-continuous if, whenever a word w and a prefix u of w are in the set, then all the prefixes of w longer than u are also in the set. Prefix-continuous sets include prefix-closed sets as a special case. Since the set of unreduced sequences in a sequence-based specification is prefix-closed by construction, it follows that the constructed prefix string-rewriting system is both terminating and confluent. In addition, we looked at how general string rewrite rules could be incorporated to expedite the automaton discovery process from requirements.

Our work differs from the previous work in that rewriting techniques are applied to assist in the discovery of a state machine from informal requirements, and to augment the enumeration process with increasing degree of automation. We used unconditional forms of both string-rewriting and prefix string-rewriting, and combined them into a mixed reduction system.

8. Conclusion

The various patterns observed in field applications [2, 4, 6, 7] of the sequence-based specification method have led to the systematic study of applying string-rewriting to sequence-based specification as presented in this paper. We find that the enumeration process can be enhanced, with string rewrite rules being discovered along the way and used to expedite the process as well as to support requirements validation. Application of the theoretical framework presented here keeps the enhanced enumeration process sound.

There is more interplay between the human specifier and the enumeration tool. As the user reduces a sequence to a previously enumerated sequence, the reduction expresses a prefix rewrite rule. An additional automation step becomes possible. If the user sees a more general string rewrite rule, he/she may declare the rule. Later on in the process for any newly enumerated sequence, the tool identifies the sequence it should be reduced to, by applying all the available prefix and string rewrite rules discovered so far.

The degree to which application of these rewrite rules will expedite the enumeration process will vary with the application. The tool enforces the mathematics but hides the details. The greatest benefit results from the savings in both time and labor achieved by consistent decisions throughout. Another benefit results from the opportunity string-rewriting provides for validation of specification decisions to requirements. This is valuable in offering a new insight or articulating an important fact about the requirements that was unstated.

Current research is focused on other practical matters relevant to the application and development of sequence-based specification. For instance, we expect a thorough treatment of abstractions and abstraction management to produce benefits in application. As application is usually facilitated by separation of inputs that do not interact (to reduce the size of the input alphabet in an enumeration), composition of sequence-based specifications is also of interest. We are extending the discrete sequence-based specification method to directly handle timing, additional forms of non-determinism, and continuity for hybrid and switching systems. The preliminary results are promising [33, 34].

Appendix

Table 12: An enumeration for the driver side car mirror ECU.

Row	Sequence	Response	Equivalence	Trace	Rule
0	λ	0	λ	Method	
1	MHI	ω	MHI	D1	$P_1 - P_{12}$: MHI.$x \models$ MHI (x is any possible stimulus)
2	MHO	ω	MHI	D1	P_{13}
3	MVD	ω	MHI	D1	P_{14}
4	MVU	ω	MHI	D1	P_{15}
5	PHEI	ω	MHI	D1	P_{16}
6	PHEO	ω	MHI	D1	P_{17}
7	PHNE	ω	MHI	D1	P_{18}

8	PVED	ω	MHI	D1	P_{19}
9	PVEU	ω	MHI	D1	P_{20}
10	PVNE	ω	MHI	D1	P_{21}
11	SM	ω	MHI	D1	P_{22}
12	START	0	START	2, D2	
13	START.MHI	0	START	D3	S_1
14	START.MHO	0	START	D3	S_2
15	START.MVD	0	START	D3	S_3
16	START.MVU	0	START	D3	S_4
17	START.PHEI	0	START.PHEI	D4	
18	START.PHEO	0	START.PHEO	D4	
19	START.PHNE	0	START.PHNE	D4	
20	START.PVED	0	START.PVED	D4	
21	START.PVEU	0	START.PVEU	D4	
22	START.PVNE	0	START.PVNE	D4	
23	START.SM	0	START.SM	D5	
24	START.START	0	START	2, D2, D6	S_5
25	START.PHEI. MHI	CERR	START.PHEI	8, 9	S_6: PHEI.MHI $\vdash$ PHEI
26	START.PHEI. MHO	HO	START	8	S_7
27	START.PHEI. MVD	0	START.PHEI	D3	S_8
28	START.PHEI. MVU	0	START.PHEI	D3	S_9
29	START.PHEI. PHEI	0	START.PHEI	D4	S_{10}: PHEI.PHEI $\vdash$ PHEI
30	START.PHEI. PHEO	0	START.PHEO	D4	S_{11}: PHEI.PHEO $\vdash$ PHEO
31	START.PHEI. PHNE	0	START.PHNE	D4	S_{12}: PHEI.PHNE $\vdash$ PHNE
32	START.PHEI. PVED	0	START.PHEI. PVED	D4	
33	START.PHEI. PVEU	0	START.PHEI. PVEU	D4	
34	START.PHEI. PVNE	0	START.PHEI. PVNE	D4	
35	START.PHEI. SM	0	START.SM	D5, D7, D8	S_{13}: PHEI.SM $\vdash$ SM
36	START.PHEI. START	0	START	2, D2, D6	S_{14}
37	START.PHEO. MHI	HI	START	8	S_{15}
38	START.PHEO. MHO	CERR	START.PHEO	8, 9	S_{16}: PHEO.MHO $\vdash$ PHEO
39	START.PHEO. MVD	0	START.PHEO	D3	S_{17}
40	START.PHEO. MVU	0	START.PHEO	D3	S_{18}
41	START.PHEO. PHEI	0	START.PHEI	D4	S_{19}: PHEO.PHEI $\vdash$ PHEI
42	START.PHEO. PHEO	0	START.PHEO	D4	S_{20}: PHEO.PHEO $\vdash$ PHEO
43	START.PHEO. PHNE	0	START.PHNE	D4	S_{21}: PHEO.PHNE $\vdash$ PHNE

44	START.PHEO. PVED	0	START.PHEO. PVED	D4	
45	START.PHEO. PVEU	0	START.PHEO. PVEU	D4	
46	START.PHEO. PVNE	0	START.PHEO. PVNE	D4	
47	START.PHEO. SM	0	START.SM	D5, D7, D8	S_{22}: PHEO.SM ⊢ SM
48	START.PHEO. START	0	START	2, D2, D6	S_{23}
49	START.PHNE. MHI	HI	START	8	S_{24}
50	START.PHNE. MHO	HO	START	8	S_{25}
51	START.PHNE. MVD	0	START.PHNE	D3	S_{26}
52	START.PHNE. MVU	0	START.PHNE	D3	S_{27}
53	START.PHNE. PHEI	0	START.PHEI	D4	S_{28}: PHNE.PHEI ⊢ PHEI
54	START.PHNE. PHEO	0	START.PHEO	D4	S_{29}: PHNE.PHEO ⊢ PHEO
55	START.PHNE. PHNE	0	START.PHNE	D4	S_{30}: PHNE.PHNE ⊢ PHNE
56	START.PHNE. PVED	0	START.PHNE. PVED	D4	
57	START.PHNE. PVEU	0	START.PHNE. PVEU	D4	
58	START.PHNE. PVNE	0	START.PHNE. PVNE	D4	
59	START.PHNE. SM	0	START.SM	D5, D7, D8	S_{31}: PHNE.SM ⊢ SM
60	START.PHNE. START	0	START	2, D2, D6	S_{32}
61	START.PVED. MHI	0	START.PVED	D3	S_{33}
62	START.PVED. MHO	0	START.PVED	D3	S_{34}
63	START.PVED. MVD	CERR	START.PVED	7, 9	S_{35}: PVED.MVD ⊢ PVED
64	START.PVED. MVU	VU	START	7	S_{36}
65	START.PVED. PHEI	0	START.PHEI. PVED	D4	S_{37}: PVED.PHEI ⊢ PHEI.PVED
66	START.PVED. PHEO	0	START.PHEO. PVED	D4	S_{38}: PVED.PHEO ⊢ PHEO.PVED
67	START.PVED. PHNE	0	START.PHNE. PVED	D4	S_{39}: PVED.PHNE ⊢ PHNE.PVED
68	START.PVED. PVED	0	START.PVED	D4	S_{40}: PVED.PVED ⊢ PVED
69	START.PVED. PVEU	0	START.PVEU	D4	S_{41}: PVED.PVEU ⊢ PVEU
70	START.PVED. PVNE	0	START.PVNE	D4	S_{42}: PVED.PVNE ⊢ PVNE
71	START.PVED. SM	0	START.SM	D5, D7, D8	S_{43}: PVED.SM ⊢ SM

72	START.PVED. START	0	START	2, D2, D6	S_{44}
73	START.PVEU. MHI	0	START.PVEU	D3	S_{45}
74	START.PVEU. MHO	0	START.PVEU	D3	S_{46}
75	START.PVEU. MVD	VD	START	7	S_{47}
76	START.PVEU. MVU	CERR	START.PVEU	7, 9	S_{48}: PVEU.MVU $\vdash$ PVEU
77	START.PVEU. PHEI	0	START.PHEI. PVEU	D4	S_{49}: PVEU.PHEI $\vdash$ PHEI.PVEU
78	START.PVEU. PHEO	0	START.PHEO. PVEU	D4	S_{50}: PVEU.PHEO $\vdash$ PHEO.PVEU
79	START.PVEU. PHNE	0	START.PHNE. PVEU	D4	S_{51}: PVEU.PHNE $\vdash$ PHNE.PVEU
80	START.PVEU. PVED	0	START.PVED	D4	S_{52}: PVEU.PVED $\vdash$ PVED
81	START.PVEU. PVEU	0	START.PVEU	D4	S_{53}: PVEU.PVEU $\vdash$ PVEU
82	START.PVEU. PVNE	0	START.PVNE	D4	S_{54}: PVEU.PVNE $\vdash$ PVNE
83	START.PVEU. SM	0	START.SM	D5, D7, D8	S_{55}: PVEU.SM $\vdash$ SM
84	START.PVEU. START	0	START	2, D2, D6	S_{56}
85	START.PVNE. MHI	0	START.PVNE	D3	S_{57}
86	START.PVNE. MHO	0	START.PVNE	D3	S_{58}
87	START.PVNE. MVD	VD	START	7	S_{59}
88	START.PVNE. MVU	VU	START	7	S_{60}
89	START.PVNE. PHEI	0	START.PHEI. PVNE	D4	S_{61}: PVNE.PHEI $\vdash$ PHEI.PVNE
90	START.PVNE. PHEO	0	START.PHEO. PVNE	D4	S_{62}: PVNE.PHEO $\vdash$ PHEO.PVNE
91	START.PVNE. PHNE	0	START.PHNE. PVNE	D4	S_{63}: PVNE.PHNE $\vdash$ PHNE.PVNE
92	START.PVNE. PVED	0	START.PVED	D4	S_{64}: PVNE.PVED $\vdash$ PVED
93	START.PVNE. PVEU	0	START.PVEU	D4	S_{65}: PVNE.PVEU $\vdash$ PVEU
94	START.PVNE. PVNE	0	START.PVNE	D4	S_{66}: PVNE.PVNE $\vdash$ PVNE
95	START.PVNE. SM	0	START.SM	D5, D7, D8	S_{67}: PVNE.SM $\vdash$ SM
96	START.PVNE. START	0	START	2, D2, D6	S_{68}
97	START.SM. MHI	CMHI	START.SM	6, D7	S_{69}: SM.MHI $\vdash$ SM
98	START.SM. MHO	CMHO	START.SM	6, D7	S_{70}: SM.MHO $\vdash$ SM
99	START.SM. MVD	CMVD	START.SM	6, D7	S_{71}: SM.MVD $\vdash$ SM

100	START.SM.MVU	CMVU	START.SM	6, D7	S_{72}: SM.MVU $\vdash$ SM
101	START.SM.PHEI	0	START.SM	D4, D7, D8	S_{73}
102	START.SM.PHEO	0	START.SM	D4, D7, D8	S_{74}
103	START.SM.PHNE	0	START.SM	D4, D7, D8	S_{75}
104	START.SM.PVED	0	START.SM	D4, D7, D8	S_{76}
105	START.SM.PVEU	0	START.SM	D4, D7, D8	S_{77}
106	START.SM.PVNE	0	START.SM	D4, D7, D8	S_{78}
107	START.SM.SM	0	START	1, D5	S_{79}
108	START.SM.START	0	START	2, D2, D6	S_{80}
109	START.PHEI.PVED.MHI	CERR	START.PHEI.PVED	8, 9	S_{81}: PHEI.PVED.MHI $\vdash$ PHEI.PVED
110	START.PHEI.PVED.MHO	HO	START.PVED	8	S_{82}: PHEI.PVED.MHO $\vdash$ PVED
111	START.PHEI.PVED.MVD	CERR	START.PHEI.PVED Using S_{35}	7, 9	
112	START.PHEI.PVED.MVU	VU	START.PHEI	7	S_{83}: PHEI.PVED.MVU $\vdash$ PHEI
113	START.PHEI.PVED.PHEI	0	START.PHEI.PVED Using S_{37}, S_{10}	D4	S_{84}: PHEI.PVED.PHEI $\vdash$ PHEI.PVED
114	START.PHEI.PVED.PHEO	0	START.PHEO.PVED Using S_{38}, S_{11}	D4	S_{85}: PHEI.PVED.PHEO $\vdash$ PHEO.PVED
115	START.PHEI.PVED.PHNE	0	START.PHNE.PVED Using S_{39}, S_{12}	D4	S_{86}: PHEI.PVED.PHNE $\vdash$ PHNE.PVED
116	START.PHEI.PVED.PVED	0	START.PHEI.PVED Using S_{40}	D4	
117	START.PHEI.PVED.PVEU	0	START.PHEI.PVEU Using S_{41}	D4	
118	START.PHEI.PVED.PVNE	0	START.PHEI.PVNE Using S_{42}	D4	
119	START.PHEI.PVED.SM	0	START.SM Using S_{43}, S_{13}	D5, D7, D8	
120	START.PHEI.PVED.START	0	START	2, D2, D6	S_{87}
121	START.PHEI.PVEU.MHI	CERR	START.PHEI.PVEU	8, 9	S_{88}: PHEI.PVEU.MHI $\vdash$ PHEI.PVEU
122	START.PHEI.PVEU.MHO	HO	START.PVEU	8	S_{89}: PHEI.PVEU.MHO $\vdash$ PVEU

123	START.PHEI.PVEU.MVD	VD	START.PHEI	7	S_{90}: PHEI.PVEU.MVD $\vdash$ PHEI
124	START.PHEI.PVEU.MVU	CERR	START.PHEI.PVEU Using S_{48}	7, 9	
125	START.PHEI.PVEU.PHEI	0	START.PHEI.PVEU Using S_{49}, S_{10}	D4	S_{91}: PHEI.PVEU.PHEI $\vdash$ PHEI.PVEU
126	START.PHEI.PVEU.PHEO	0	START.PHEO.PVEU Using S_{50}, S_{11}	D4	S_{92}: PHEI.PVEU.PHEO $\vdash$ PHEO.PVEU
127	START.PHEI.PVEU.PHNE	0	START.PHNE.PVEU Using S_{51}, S_{12}	D4	S_{93}: PHEI.PVEU.PHNE $\vdash$ PHNE.PVEU
128	START.PHEI.PVEU.PVED	0	START.PHEI.PVED Using S_{52}	D4	
129	START.PHEI.PVEU.PVEU	0	START.PHEI.PVEU Using S_{53}	D4	
130	START.PHEI.PVEU.PVNE	0	START.PHEI.PVNE Using S_{54}	D4	
131	START.PHEI.PVEU.SM	0	START.SM Using S_{55}, S_{13}	D5, D7, D8	
132	START.PHEI.PVEU.START	0	START	2, D2, D6	S_{94}
133	START.PHEI.PVNE.MHI	CERR	START.PHEI.PVNE	8, 9	S_{95}: PHEI.PVNE.MHI $\vdash$ PHEI.PVNE
134	START.PHEI.PVNE.MHO	HO	START.PVNE	8	S_{96}: PHEI.PVNE.MHO $\vdash$ PVNE
135	START.PHEI.PVNE.MVD	VD	START.PHEI	7	S_{97}: PHEI.PVNE.MVD $\vdash$ PHEI
136	START.PHEI.PVNE.MVU	VU	START.PHEI	7	S_{98}: PHEI.PVNE.MVU $\vdash$ PHEI
137	START.PHEI.PVNE.PHEI	0	START.PHEI.PVNE Using S_{61}, S_{10}	D4	S_{99}: PHEI.PVNE.PHEI $\vdash$ PHEI.PVNE
138	START.PHEI.PVNE.PHEO	0	START.PHEO.PVNE Using S_{62}, S_{11}	D4	S_{100}: PHEI.PVNE.PHEO $\vdash$ PHEO.PVNE
139	START.PHEI.PVNE.PHNE	0	START.PHNE.PVNE Using S_{63}, S_{12}	D4	S_{101}: PHEI.PVNE.PHNE $\vdash$ PHNE.PVNE
140	START.PHEI.PVNE.PVED	0	START.PHEI.PVED Using S_{64}	D4	
141	START.PHEI.PVNE.PVEU	0	START.PHEI.PVEU Using S_{65}	D4	
142	START.PHEI.PVNE.PVNE	0	START.PHEI.PVNE	D4	

			Using S_{66}		
143	START.PHEI. PVNE.SM	0	START.SM Using S_{67}, S_{13}	D5, D7, D8	
144	START.PHEI. PVNE.START	0	START	2, D2, D6	S_{102}
145	START.PHEO. PVED.MHI	HI	START.PVED	8	S_{103}: PHEO.PVED.MHI $\vdash$ PVED
146	START.PHEO. PVED.MHO	CERR	START.PHEO. PVED	8, 9	S_{104}: PHEO.PVED.MHO $\vdash$ PHEO.PVED
147	START.PHEO. PVED.MVD	CERR	START.PHEO. PVED Using S_{35}	7, 9	
148	START.PHEO. PVED.MVU	VU	START.PHEO	7	S_{105}: PHEO.PVED.MVU $\vdash$ PHEO
149	START.PHEO. PVED.PHEI	0	START.PHEI. PVED Using S_{37}, S_{19}	D4	S_{106}: PHEO.PVED.PHEI $\vdash$ PHEI.PVED
150	START.PHEO. PVED.PHEO	0	START.PHEO. PVED Using S_{38}, S_{20}	D4	S_{107}: PHEO.PVED.PHEO $\vdash$ PHEO.PVED
151	START.PHEO. PVED.PHNE	0	START.PHNE. PVED Using S_{39}, S_{21}	D4	S_{108}: PHEO.PVED.PHNE $\vdash$ PHNE.PVED
152	START.PHEO. PVED.PVED	0	START.PHEO. PVED Using S_{40}	D4	
153	START.PHEO. PVED.PVEU	0	START.PHEO. PVEU Using S_{41}	D4	
154	START.PHEO. PVED.PVNE	0	START.PHEO. PVNE Using S_{42}	D4	
155	START.PHEO. PVED.SM	0	START.SM Using S_{43}, S_{22}	D5, D7, D8	
156	START.PHEO. PVED.START	0	START	2, D2, D6	S_{109}
157	START.PHEO. PVEU.MHI	HI	START.PVEU	8	S_{110}: PHEO.PVEU.MHI $\vdash$ PVEU
158	START.PHEO. PVEU.MHO	CERR	START.PHEO. PVEU	8, 9	S_{111}: PHEO.PVEU.MHO $\vdash$ PHEO.PVEU
159	START.PHEO. PVEU.MVD	VD	START.PHEO	7	S_{112}: PHEO.PVEU.MVD $\vdash$ PHEO
160	START.PHEO. PVEU.MVU	CERR	START.PHEO. PVEU Using S_{48}	7, 9	
161	START.PHEO. PVEU.PHEI	0	START.PHEI. PVEU Using S_{49}, S_{19}	D4	S_{113}: PHEO.PVEU.PHEI $\vdash$ PHEI.PVEU
162	START.PHEO. PVEU.PHEO	0	START.PHEO. PVEU Using S_{50}, S_{20}	D4	S_{114}: PHEO.PVEU.PHEO $\vdash$ PHEO.PVEU

163	START.PHEO. PVEU.PHNE	0	START.PHNE. PVEU Using S_{51}, S_{21}	D4	S_{115}: PHEO.PVEU.PHNE $\vdash$ PHNE.PVEU
164	START.PHEO. PVEU.PVED	0	START.PHEO. PVED Using S_{52}	D4	
165	START.PHEO. PVEU.PVEU	0	START.PHEO. PVEU Using S_{53}	D4	
166	START.PHEO. PVEU.PVNE	0	START.PHEO. PVNE Using S_{54}	D4	
167	START.PHEO. PVEU.SM	0	START.SM Using S_{55}, S_{22}	D5, D7, D8	
168	START.PHEO. PVEU.START	0	START	2, D2, D6	S_{116}
169	START.PHEO. PVNE.MHI	HI	START.PVNE	8	S_{117}: PHEO.PVNE.MHI $\vdash$ PVNE
170	START.PHEO. PVNE.MHO	CERR	START.PHEO. PVNE	8, 9	S_{118}: PHEO.PVNE.MHO $\vdash$ PHEO.PVNE
171	START.PHEO. PVNE.MVD	VD	START.PHEO	7	S_{119}: PHEO.PVNE.MVD $\vdash$ PHEO
172	START.PHEO. PVNE.MVU	VU	START.PHEO	7	S_{120}: PHEO.PVNE.MVU $\vdash$ PHEO
173	START.PHEO. PVNE.PHEI	0	START.PHEI. PVNE Using S_{61}, S_{19}	D4	S_{121}: PHEO.PVNE.PHEI $\vdash$ PHEI.PVNE
174	START.PHEO. PVNE.PHEO	0	START.PHEO. PVNE Using S_{62}, S_{20}	D4	S_{122}: PHEO.PVNE.PHEO $\vdash$ PHEO.PVNE
175	START.PHEO. PVNE.PHNE	0	START.PHNE. PVNE Using S_{63}, S_{21}	D4	S_{123}: PHEO.PVNE.PHNE $\vdash$ PHNE.PVNE
176	START.PHEO. PVNE.PVED	0	START.PHEO. PVED Using S_{64}	D4	
177	START.PHEO. PVNE.PVEU	0	START.PHEO. PVEU Using S_{65}	D4	
178	START.PHEO. PVNE.PVNE	0	START.PHEO. PVNE Using S_{66}	D4	
179	START.PHEO. PVNE.SM	0	START.SM Using S_{67}, S_{22}	D5, D7, D8	
180	START.PHEO. PVNE.START	0	START	2, D2, D6	S_{124}
181	START.PHNE. PVED.MHI	HI	START.PVED	8	S_{125}: PHNE.PVED.MHI $\vdash$ PVED
182	START.PHNE. PVED.MHO	HO	START.PVED	8	S_{126}: PHNE.PVED.MHO $\vdash$ PVED

183	START.PHNE. PVED.MVD	CERR	START.PHNE. PVED Using S_{35}	7, 9	
184	START.PHNE. PVED.MVU	VU	START.PHNE	7	S_{127}: PHNE.PVED.MVU $\vdash$ PHNE
185	START.PHNE. PVED.PHEI	0	START.PHEI. PVED Using S_{37}, S_{28}	D4	S_{128}: PHNE.PVED.PHEI $\vdash$ PHEI.PVED
186	START.PHNE. PVED.PHEO	0	START.PHEO. PVED Using S_{38}, S_{29}	D4	S_{129}: PHNE.PVED.PHEO $\vdash$ PHEO.PVED
187	START.PHNE. PVED.PHNE	0	START.PHNE. PVED Using S_{39}, S_{30}	D4	S_{130}: PHNE.PVED.PHNE $\vdash$ PHNE.PVED
188	START.PHNE. PVED.PVED	0	START.PHNE. PVED Using S_{40}	D4	
189	START.PHNE. PVED.PVEU	0	START.PHNE. PVEU Using S_{41}	D4	
190	START.PHNE. PVED.PVNE	0	START.PHNE. PVNE Using S_{42}	D4	
191	START.PHNE. PVED.SM	0	START.SM Using S_{43}, S_{31}	D5, D7, D8	
192	START.PHNE. PVED.START	0	START	2, D2, D6	S_{131}
193	START.PHNE. PVEU.MHI	HI	START.PVEU	8	S_{132}: PHNE.PVEU.MHI $\vdash$ PVEU
194	START.PHNE. PVEU.MHO	HO	START.PVEU	8	S_{133}: PHNE.PVEU.MHO $\vdash$ PVEU
195	START.PHNE. PVEU.MVD	VD	START.PHNE	7	S_{134}: PHNE.PVEU.MVD $\vdash$ PHNE
196	START.PHNE. PVEU.MVU	CERR	START.PHNE. PVEU Using S_{48}	7, 9	
197	START.PHNE. PVEU.PHEI	0	START.PHEI. PVEU Using S_{49}, S_{28}	D4	S_{135}: PHNE.PVEU.PHEI $\vdash$ PHEI.PVEU
198	START.PHNE. PVEU.PHEO	0	START.PHEO. PVEU Using S_{50}, S_{29}	D4	S_{136}: PHNE.PVEU.PHEO $\vdash$ PHEO.PVEU
199	START.PHNE. PVEU.PHNE	0	START.PHNE. PVEU Using S_{51}, S_{30}	D4	S_{137}: PHNE.PVEU.PHNE $\vdash$ PHNE.PVEU
200	START.PHNE. PVEU.PVED	0	START.PHNE. PVED Using S_{52}	D4	
201	START.PHNE. PVEU.PVEU	0	START.PHNE. PVEU Using S_{53}	D4	
202	START.PHNE. PVEU.PVNE	0	START.PHNE. PVNE	D4	

			Using S_{54}		
203	START.PHNE. PVEU.SM	0	START.SM Using S_{55}, S_{31}	D5, D7, D8	
204	START.PHNE. PVEU.START	0	START	2, D2, D6	S_{138}
205	START.PHNE. PVNE.MHI	HI	START.PVNE	8	S_{139}: PHNE.PVNE.MHI $\vdash$ PVNE
206	START.PHNE. PVNE.MHO	HO	START.PVNE	8	S_{140}: PHNE.PVNE.MHO $\vdash$ PVNE
207	START.PHNE. PVNE.MVD	VD	START.PHNE	7	S_{141}: PHNE.PVNE.MVD $\vdash$ PHNE
208	START.PHNE. PVNE.MVU	VU	START.PHNE	7	S_{142}: PHNE.PVNE.MVU $\vdash$ PHNE
209	START.PHNE. PVNE.PHEI	0	START.PHEI. PVNE Using S_{61}, S_{28}	D4	S_{143}: PHNE.PVNE.PHEI $\vdash$ PHEI.PVNE
210	START.PHNE. PVNE.PHEO	0	START.PHEO. PVNE Using S_{62}, S_{29}	D4	S_{144}: PHNE.PVNE.PHEO $\vdash$ PHEO.PVNE
211	START.PHNE. PVNE.PHNE	0	START.PHNE. PVNE Using S_{63}, S_{30}	D4	S_{145}: PHNE.PVNE.PHNE $\vdash$ PHNE.PVNE
212	START.PHNE. PVNE.PVED	0	START.PHNE. PVED Using S_{64}	D4	
213	START.PHNE. PVNE.PVEU	0	START.PHNE. PVEU Using S_{65}	D4	
214	START.PHNE. PVNE.PVNE	0	START.PHNE. PVNE Using S_{66}	D4	
215	START.PHNE. PVNE.SM	0	START.SM Using S_{67}, S_{31}	D5, D7, D8	
216	START.PHNE. PVNE.START	0	START	2, D2, D6	S_{146}

Acknowledgement

The authors would like to thank Klaus Madlener for several fruitful discussions and important comments on string-rewriting, as well as the use of the COSY tool developed by the Technical University of Kaiserslautern, Germany.

The authors also acknowledge with thanks our communication with David Parnas regarding this research. We benefited greatly from his knowledge of research literature on string-rewriting that was relevant to our work, and for helpful comments on an early draft of the paper.

References

[1] S. J. Prowell, J. H. Poore, Sequence-based software specification of deterministic systems, Software: Practice and Experience 28 (1998) 329–344.

[2] S. J. Prowell, C. J. Trammell, R. C. Linger, J. H. Poore, Cleanroom Software Engineering: Technology and Process, Addison-Wesley, Reading, MA, 1999.

[3] S. J. Prowell, J. H. Poore, Foundations of sequence-based software specification, IEEE Transactions on Software Engineering 29 (2003) 417–429.

[4] 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, Chiang Mai, Thailand, pp. 49–59, 2003.

[5] 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, Columbus, OH, 2004.

[6] P. J. Hopcroft, G. H. Broadfoot, Combining the box structure development method and CSP for software development, Electronic Notes in Theoretical Computer Science 128 (2005) 127–144.

[7] T. Bauer, T. Beletski, F. Boehr, R. Eschbach, D. Landmann, J. Poore, From requirements to statistical testing of embedded systems, in: Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems, Minneapolis, MN, pp. 3–9, 2007.

[8] J. M. Carter, L. Lin, J. H. Poore, Automated functional testing of Simulink control models, in: Proceedings of the 1st Workshop on Model-Based Testing in Practice, Berlin, Germany, pp. 41–50, 2008.

[9] Proto_Seq, 2011. ESP Project. http://sqrl.eecs.utk.edu/esp/index.html.

[10] L. Lin, S. J. Prowell, J. H. Poore, The impact of requirements changes on specifications and state machines, Software: Practice and Experience 39 (2009) 573–610.

[11] M. Joseph (Ed.), Real-Time Systems: Specification, Verification and Analysis, Prentice Hall International, London, United Kingdom, 1996.

[12] Weigh-In-Motion, 2006. Weigh-In-Motion, Cube Management, and Marking User Manual, Oak Ridge National Laboratory, Oak Ridge, TN, Version 0.8.2.

[13] SOS, 2011. Satellite Operations Software Enumeration. `http://sqrl.eecs.utk.edu/btw/files/SOS_sr.html`.

[14] MPCS, 2011. Mine Pump Controller Software Enumeration. `http://sqrl.eecs.utk.edu/btw/files/MPCS_sr.html`.

[15] WIMDAP, 2011. Weigh-In-Motion Data Acquisition Processor Enumeration. `http://sqrl.eecs.utk.edu/btw/files/WIMDAP_sr.html`.

[16] R. V. Book, F. Otto, String-Rewriting Systems, Springer-Verlag, Berlin, Germany, 1993.

[17] H. D. Mills, The new math of computer programming, Comminications of the ACM 18 (1975) 43–48.

[18] L. Lin, S. J. Prowell, J. H. Poore, An axiom system for sequence-based specification, Theoretical Computer Science 411 (2010) 360–376.

[19] J. Brzozowski, H. Jürgensen, Representation of semiautomata by canonical words and equivalences, International Journal of Foundations of Computer Science 16 (2005) 831–850.

[20] R. C. Linger, H. D. Mills, B. I. Witt, Structured Programming: Theory and Practice, Addison-Wesley, Boston, MA, 1979.

[21] H. D. Mills, Stepwise refinement and verification in box-structured systems, IEEE Computer 21 (1988) 23–36.

[22] 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, Venice, Italy, pp. 211–236, 1978.

[23] D. L. Parnas, Y. Wang, The trace assertion method of module interface specification, Technical Report 89-261, Queens University, 1989.

[24] J. Brzozowski, Derivatives of regular expressions, Journal of the ACM 11 (1964) 481–494.

[25] J. McLean, A formal method for the abstract specification of software, Journal of the ACM 31 (1984) 600–627.

[26] D. Hoffman, The trace specification of communications protocols, IEEE Transactions on Computers C34 (1985) 1102–1113.

[27] D. Hoffman, R. T. Snodgrass, Trace specifications: Methodology and models, IEEE Transactions on Software Engineering 14 (1988) 1243–1252.

[28] Y. Wang, D. L. Parnas, Simulating the behavior of software modules by trace rewriting, in: Proceedings of the 15th International Conference on Software Engineering, Baltimore, MD, pp. 14–23, 1993.

[29] Y. Wang, D. L. Parnas, Simulating the behavior of software modules by trace rewriting, IEEE Transactions on Software Engineering 20 (1994) 750–759.

[30] R. Janicki, E. Sekerinski, Foundations of the trace assertion method of module interface specification, IEEE Transactions on Software Engineering 27 (2001) 577–598.

[31] J. Brzozowski, H. Jürgensen, Theory of deterministic trace-assertion specifications, Technical Report CS-2004-30, University of Waterloo, 2004.

[32] 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 (2007) 1065–1087.

[33] J. M. Carter, Sequence-Based Specification of Embedded Systems, Ph.D. thesis, University of Tennessee,

Knoxville, 2009.

[34] W. T. Swain, Application of hybrid sequence-based specification to a data acquisition processor, Technical Report UT-CS-11-669, University of Tennessee, Knoxville, 2011. `http://sqrl.eecs.utk.edu/btw/files/WIM-HSBS-TR.pdf`.