Management of Requirements Changes in Sequence-Based Specifications

Lan Lin, Stacy J. Prowell, and Jesse H. Poore

January 2007

Department of Computer Science
University of Tennessee
Knoxville, TN 37996

Technical Report ut-cs-07-588

Abstract

Ordinary requirements come in many forms, natural languages, equations, tables, charts, predecessor systems, and ideas in the minds of domain experts. All forms can contain ambiguities, errors, and omissions. They change both during and after a phase of development. Sequence-based specification has in many field applications been effective in converting ordinary requirements to precise specifications through a constructive process. Algorithms for managing requirements changes meet a very great need in applications of sequence-based specification. In this paper we explore the change theory developed with the aid of an axiom system for sequence-based specification, and present algorithms for managing requirements changes of various kinds. This has established the basis for maximizing potential automation support and producing benefits in field applications as well as further development of sequence-based specification.

Keywords Software specification, requirements management, changing state diagrams, automata.

Introduction

At an age when software is embedded in everything from consumer appliances to automobiles to medical devices, dependability of software has become an urgent requirement. Methodologies and tools are in great demand to make sure conceptual mistakes are avoided in software development and code is correct by design. The sequence-based specification method [1] was developed to contribute to this goal converting ordinary, or typical requirements and requirements statements into mathematically precise specifications at an acceptable level of abstraction for deterministic systems, and has proven very useful and efficient in practice [2–7]. Meanwhile due to the iterative nature of modern software development, the need for managing changes of requirements in sequence-based specifications arises both as a consequence of practicing this method and as a general topic in software requirements engineering.

Requirements documents of all forms usually contain ambiguities, errors, and omissions. The goal of requirements analysis is to discover a precise specification from the informal requirements that resolves all these problems, assures completeness and consistency, yet remains understandable so that it can then be validated for correctness by application domain experts. The method of sequence-based specification was developed to bridge this gap through a constructive process.

The behavior of a software system can be specified in terms of external stimuli and responses [8]. It is fully encoded in a black box function that maps every possible sequence of stimuli to a response. To derive the black box function and a formal specification we apply two central techniques: sequence enumeration and sequence abstraction. Sequence enumeration involves systematically considering all stimulus sequences for any externally observable response and partitioning the infinite stimulus sequence space into a finite number of equivalence classes based on future behavior. Sequence abstraction further helps in controlling the growth of this inherently combinatorial process and avoiding any inefficient work. The foundations of the sequence-based specification method are laid out in [1]. Specifications written with this method are proven to be complete, consistent, and traceably correct.

If a sequence does not generate any externally observable response, we map it to a special null response 0. If the response to a sequence is physically impossible or otherwise makes no sense with respect to operational behavior, we map it to another special response ω for being illegal. When enumerated according to some pre-defined total ordering, each sequence is mapped either to an externally observable response, or 0, or ω, and checked to see if it is equivalent in the sense of Mealy state machines [9] to (and hence reduced to) a previously enumerated sequence. Two sequences are Mealy equivalent if and only if they always give the same response value however far extended by the same non-empty sequence, i.e., their future behaviors will not differ from each other. Four rules guide the enumeration process:

1. **(Reduction).** A sequence can only be reduced to an unreduced sequence.
2. **(Illegal Prefix).** A sequence mapped to ω need not be extended.
3. **(Reduced Prefix).** A reduced sequence need not be extended.

4. **(Extension).** Otherwise, a sequence must be extended by every stimulus.

The enumeration terminates (and is said to be complete) if all sequences of a certain length are either mapped to ω or reduced to prior sequences. Traces to tagged requirements or derived requirements are noted to justify decisions regarding any mapped value for responses or equivalences. Both the black box and the state box functions of Mills [8] are thereby constructed with traceability to the basis for the response and equivalence decisions, as illustrated below.

With a complete enumeration it is easy to obtain the mapped response for any stimulus sequence, no matter whether the sequence itself shows up in the enumeration or not. The algorithm (referred to as $\mathcal{A}$ hereafter) is put forward in [1] and proceeds as follows. We first check to see if the sequence is in the table; if so we read off its response value, otherwise we find its longest prefix that is in the table. If the prefix is mapped to *illegal*, the sequence is also mapped to *illegal*, otherwise we replace the prefix with its reduced value (in the equivalence column) in the original sequence and repeat the process on the new sequence until a mapped response is identified. Since reductions in the table are declared based on Mealy equivalence and the unreduced sequences imply equivalence classes that further represent states of the automaton, this process amounts to traversing the underlying Mealy machine and finding the output value for any input sequence. Therefore, once we obtain a complete and finite enumeration from the requirements, the black box function of the system is fully decoded.

It may be noticed that sequence-based specification has one distinct advantage over other state-based modeling methods like the Trace Assertion Method (TAM) [10], Z [11], or Software Cost Reduction (SCR) [12]. While most other methods deal with the description of the state machine rather than how to discover or invent the state machine, sequence-based specification provides a systematic way to discover and derive the automaton. In cases where understanding of the system under specification is incomplete, limited, or immature such that inventing the right state machine becomes difficult or infeasible, enumeration is especially productive. Likewise, the theory of managing requirements changes in sequence-based specifications differs from the conventional state change theory in that it is designed for active state machine development and revision.

For example, Korel [13] presented an approach of understanding model-based modifications that uses the original model and the modified model to compute the effect of the modifications through affecting and affected transitions. It assumes the availability of the modified model and bases the analysis on data and control dependence among inputs and outputs. The change might be the result of maintenance, error correction, or change in functionality driven by change in requirements. However, these algorithms would not support an original enumeration process, or complete the revision of changes in an enumeration.

Seawright and Brewer [14] present "production-based specification" and use a similar base of language-automata theory to convert grammar productions into hardware design. They, too, focus on external behavior relative to a well-defined system boundary and its interfaces, and generate Mealy-Moore state machine descriptions as an intermediate step toward circuit design. Although our algorithms could apply to production-based specification at their intermediate state machine, it is not clear how such changes would reflect in their original productions. Their production language is quite powerful, subject to debugging, and describes a larger class of Mealy machines than those represented by enumerations.

The fundamental difference is again that sequence-based specification systematically explores an evolving behavioral description as implied by requirements, and records the sequence equivalences entailed by the interpretation of requirements. Our change algorithms do not contain all the information necessary to complete the change process. They depend upon a human requirements analyst (specifier) to work systematically through the sequences in length-lexicographical order to make and document the specification decisions that will complete the changes.

Finally, we must acknowledge that many students of state machine theory over the years have been taught or required to invent algorithms to change state machines in various ways. This however, does not detract from our need for algorithms to support the enumeration process by which we convert ordinary requirements into precise specifications. This work requires that we manage changes that stem from deeper understanding as the work progresses from new news from the outside. The algorithms presented here manage changes with maximum tool support in a systematic, constructive process. We hope that the widespread familiarity with state machine diagrams and regular expressions will make this material somewhat intuitive even though it is used in an unfamiliar way.

Enumeration has proven to be practical in application and immensely powerful in eliciting errors and omissions in statements of requirements (even of mature systems). Significant gains in quality and productivity have been reported from projects carried out by the authors, and other groups in industry that have been trained to apply sequence-based specification [3, 4]. Broadfoot [5, 6] has been especially effective in the application of sequence enumeration, and has incorporated the method with elements of Communicating Sequential Processes (CSP) [15] and Failures Divergence Refinement (FDR) [16]. Enumeration has been applied in the automotive domain, for example in door control units [7] to support automated testing.

Application clarifies the extent to which inputs (and histories of inputs) can be partitioned into subsets that do not interact and, therefore, need not be enumerated together. This reduces what might be called the "breadth" of the input space from tens of inputs to usually fewer than ten per enumeration. The "length" corresponds to the accumulation of input information that is necessary to identify an indispensable state of the system. We find that if the length of input sequences appears to grow beyond about ten, this is a sign of poorly understood requirements or gratuitous complexity, both of which are occasions for reconsidering the requirements. In real systems, after the requirements are well understood, complete, consistent, and correct, one typically ends up with several enumerations each with a breadth of fewer than ten interacting inputs and a sequence length of less than ten. Of course, we have seen enumerations larger in both dimensions.

Enumerations result in specifications that are complete, consistent, and traceably correct. However, the beginning is always messy and we start over several times as the requirements are clarified and corrected. When one speaks of changes in requirements, there is an implication that it is a change from one set of complete, consistent, and correct requirements to another. The algorithms here assume a correct base and then address changes one at a time, at an elemental level in order to preserve completeness, consistency, and correctness. As these algorithms are considered one is struck by the far reaching consequences of seemingly innocuous and minor changes in requirements.

The SAFE Example

To acquaint the reader with both the sequence-based specification method and the enumeration process, we borrow a safe controller example from [1]. Enumeration is done with a prototype tool developed by UTK SQRL [1]. The completed specification is exported to html output. The screenshot in Fig. 1 shows the tagged requirements and derived requirements (D1-D2) for the simple safe controller, and the list of interfaces between the system and the environment. We diagram the system boundary in Fig. 2.

To make work efficient we introduce the abstraction of letting G denote entering the correct three digits in order and B denote entering the combination incorrectly but up to the first mistake. Stimuli and responses (outputs) are identified as in Fig. 3. An enumeration at this abstraction level is shown in Fig. 4.

In the enumeration tables if a sequence has (-, -) as a pair for its response and equivalence entries, the sequence is defined to be illegal. If it has a blank entry for response, the response is defined to be *null*. If it has "- - - - - - - - - - - -" for equivalence, it is defined to be an unreduced sequence.

The sequence-based specification tool also guides the user through state variable definition and canonical sequence analysis (Fig. 5), and generates a state box specification (Fig. 6). Notice that the state box tables have (,) as a possible (Door, Error) pair. This represents a trap state of the automaton reached by all illegal sequences. We draw this automaton diagram in Fig. 7, naming states after legal and unreduced sequences or the first illegal sequence B.

An Axiomatic Treatment

Motivated by the need to characterize an existing enumeration regardless of whether it was obtained by the enumeration process or in some other way (from TAM, SCR, or Z, for example), we developed an axiom system for sequence-based

[1] Software Quality Research Lab, Department of Computer Science, University of Tennessee at Knoxville, http://www.cs.utk.edu/sqrl/.

Requirements

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

Interfaces

door sensor

Description: The control software receives door and lock status from the sensor.
Requirement Trace: [5]

keypad

Description: The control software must receive digit presses and clear key presses from the keypad.
Requirement Trace: [1 , 2 , 6]

lock actuator

Description: The control software will control the lock.
Requirement Trace: [1 , 3 , 4 , 7]

power

Description: Power-on is detected by the control software.
Requirement Trace: [8]

Figure 1: Safe controller requirements and interfaces.

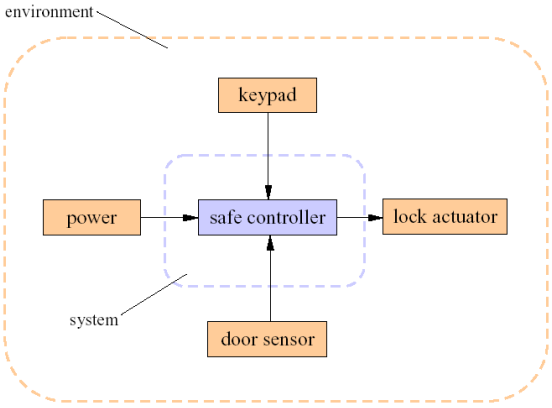


Figure 2: Safe controller system boundary.

Stimuli

B

Long Name: Bad combination entry
Description: Entering the combination incorrectly, but up to the first mistake. This is an abstract stimulus.
Sources: [keypad]
Requirement Trace: [2]

C

Long Name: Clear key press
Description: The clear key is pressed.
Sources: [keypad]
Requirement Trace: [2]

D

Long Name: Door closed
Description: Detect door closed.
Sources: [door sensor]
Requirement Trace: [4 , 5]

G

Long Name: Good combination entry
Description: Entering the correct three digits in order. This is an abstract stimulus.
Sources: [keypad]
Requirement Trace: [1 , 3]

L

Long Name: Power on with door locked
Description: Power on is detected, and the sensor reports that the door is closed and locked.
Sources: [door sensor , power]
Requirement Trace: [5 , 7 , 8]

U

Long Name: Power on with door unlocked
Description: Power on is detected, and the sensor reports that the door is unlocked.
Sources: [door sensor , power]
Requirement Trace: [4 , 5 , 6]

Outputs

lock

Long Name: Locking the door
Description: Lock the safe.
Destinations: [lock actuator]
Requirement Trace: [4]

unlock

Long Name: Unlocking the door
Description: Unlock the safe.
Destinations: [lock actuator]
Requirement Trace: [1 , 3 , 7]

Figure 3: Safe controller stimuli and responses.

Sequence Enumeration

Length 1

Prefix: lambda

Stimulus	Response	Equivalence	Requirement Trace
B	-	-	[D1]
C	-	-	[D1]
D	-	-	[D1]
G	-	-	[D1]
L		----------	[5]
U		----------	[5]

Length 2

Prefix: L

Stimulus	Response	Equivalence	Requirement Trace
B		----------	[1 , 2 , 7]
C		L	[2 , 7]
D	-	-	[8]
G	unlock	U	[1 , 3 , 7]
L		L	[5 , D2]
U		U	[5 , D2]

Prefix: U

Stimulus	Response	Equivalence	Requirement Trace
B		U	[6]
C		U	[6]
D	lock	L	[4]
G		U	[6]
L		L	[5 , D2]
U		U	[5 , D2]

Length 3

Prefix: L.B

Stimulus	Response	Equivalence	Requirement Trace
B		L.B	[2 , 7]
C		L	[2 , 7]
D	-	-	[8]
G		L.B	[2 , 7]
L		L	[5 , D2]
U		U	[5 , D2]

Figure 4: Safe controller enumeration.

State Variables

Door

Description: The status of the door – locked, unlocked, unknown, or don't care.
Possible Values: [**-----** , unlock , lock , unknown]

Error

Description: Whether there has been an error in entering the combination entry – true, false, or don't care.
Possible Values: [----- , true , false]

Canonical Sequence Analysis

Canonical Sequence	Door	Error
lambda	unknown	-----
L	lock	false
U	unlock	-----
L.B	lock	true

The canonical sequence analysis has been completed.

The canonical sequence analysis is disjoint.

Figure 5: Safe controller canonical sequence analysis.

State Box Specification

Stimulus: B, Bad combination entry

Initial State		Response	Final State	
Door	Error		Door	Error
unknown	-----	illegal		
lock	false	null	lock	true
unlock	-----	null	unlock	-----
lock	true	null	lock	true

Stimulus: C, Clear key press

Initial State		Response	Final State	
Door	Error		Door	Error
unknown	-----	illegal		
lock	false	null	lock	false
unlock	-----	null	unlock	-----
lock	true	null	lock	false

Stimulus: D, Door closed

Initial State		Response	Final State	
Door	Error		Door	Error
unknown	-----	illegal		
lock	false	illegal		
unlock	-----	lock	lock	false
lock	true	illegal		

Stimulus: G, Good combination entry

Initial State		Response	Final State	
Door	Error		Door	Error
unknown	-----	illegal		
lock	false	unlock	unlock	-----
unlock	-----	null	unlock	-----
lock	true	null	lock	true

Stimulus: L, Power on with door locked

Initial State		Response	Final State	
Door	Error		Door	Error
unknown	-----	null	lock	false
lock	false	null	lock	false
unlock	-----	null	lock	false
lock	true	null	lock	false

Stimulus: U, Power on with door unlocked

Initial State		Response	Final State	
Door	Error		Door	Error
unknown	-----	null	unlock	-----
lock	false	null	unlock	-----
unlock	-----	null	unlock	-----
lock	true	null	unlock	-----

Figure 6: Safe controller state box.

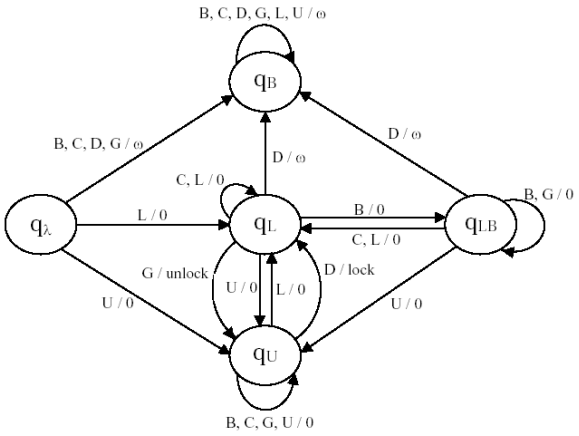


Figure 7: A state machine diagram for the safe controller.

specifications [17]. The axioms help to prove a number of important properties about sequence-based specifications; it also provides insight into the relationship of sequence-based specifications with other formal representations of a software program, including finite-state machines, regular expressions, and prefix-recursive functions. In addition the axiom system was essential to the development of algorithms to support requirements change management, the focus of this paper.

We chose to present the theory rather informally through the use of diagrams, with an attempt to motivate the enumeration axioms and give the reader a flavor of the axiom system. For a thorough and formal treatment see [17].

Suppose S is a stimulus set that is finite, non-empty, and equipped with a total order $<$ (the alphabetical order). Let the order be extended to a total order on S^* by length, and then alphabetically. Suppose R is a response set that contains 0, ω, and at least one other member. An enumeration is essentially a partial function $\mathcal{E} : S^* \to R \times S^*$ that maps certain stimulus sequences to (response, stimulus sequence) pairs. If u in S^* has a mapped value (r, v) by $\mathcal{E}$ (i.e., $\mathcal{E}(u) = (r, v)$), then we say u is mapped to response r (i.e., $u \mapsto r$, or alternatively $\mapsto (u) = r$) and reduced to sequence v (i.e., $u \triangleright v$, or alternatively $\triangleright(u) = v$). We use $u \not\mapsto r$ to denote u is mapped to a response other than r. When $\mathcal{E}$ further satisfies a list of axioms to be an enumeration, u being defined for $\mathcal{E}$ is equivalent to saying u is a sequence in that enumeration.

If u is mapped to ω, then u is illegal, otherwise it is legal. If u is reduced to u, then u is unreduced, otherwise it is reduced (to a prior sequence in the total ordering, according to Axiom 2 below). If u is legal and unreduced, then u is extensible.

If the following Axioms 1-6 hold for the partial function $\mathcal{E}$ it is called an enumeration. The enumeration is complete if Axiom 7 further holds; it is finite if Axiom 8 further holds. The enumeration is complete and minimal if Axiom 9 holds in addition to 1-7. The axioms are paraphrased informally as follows:

Axiom 1: The empty sequence must be mapped to *null*; (Empty sequence)

Axiom 2: Sequences can only be reduced to prior sequences or to themselves; (Partial order)

Axiom 3: Do not reduce to reduced sequences; (No reduction to reduced)

Axiom 4: Do not extend reduced sequences; (No extensions of reduced)

Axiom 5: Do not extend illegal sequences; (No extensions of illegals)

Axiom 6: Do not reduce illegal sequences to legal sequences unless all extensions of the legal sequence are also illegal; (Reducing illegals to legals)

Axiom 7: Every extensible sequence must be extended by every stimulus in a complete enumeration; (Completeness)

Axiom 8: There must be a finite number of sequences in a finite enumeration; (Finiteness)

Axiom 9: Unreduced sequences must be pairwise distinguishable in a complete and minimal enumeration. (Minimality)

Distinguishability as required by Axiom 9 is a relation over the set of all unreduced sequences in a complete enumeration. It is defined recursively as follows:

Rule 1: An illegal, unreduced sequence is distinguishable from an extensible sequence;

Rule 2: Two extensible sequences are distinguishable if they map to different responses when extended by (the same) one single stimulus;

Rule 3: Two extensible sequences are distinguishable if they reduce to two distinguishable sequences when extended by (the same) one single stimulus;

Rule 4: Two unreduced sequences are not distinguishable except by a finite number of applications of the above rules.

Two unreduced sequences are indistinguishable if and only if they are not distinguishable from each other. Indistinguishability is proven to be an equivalence relation.

For a complete enumeration $\mathcal{E}$, we further extend $\mapsto$ and $\triangleright$ to total functions $\hat{\mapsto}$ and $\hat{\triangleright}$ on S^* and prove that the extended functions agree with the unextended ones for every stimulus sequence that is in the enumeration.

Given any complete and finite enumeration $\mathcal{E}$ satisfying the axiomatic definition, we prove that $\hat{\mapsto}$ performs the same computation as the black box function denoted by $\mathcal{E}$ obtained by treating $\mathcal{E}$ as a result of the enumeration process and applying the algorithm $\mathcal{A}$ to it.

For a complete enumeration, the definitions of illegal / legal sequences can be extended to apply to any stimulus sequence by substituting $\hat{\mapsto}$ for $\mapsto$. A sequence is illegal if it is mapped to ω by the black box function $\hat{\mapsto}$, otherwise it is a legal sequence.

Following [9] a Mealy machine is defined as a 6-tuple $\langle Q, \Sigma, \Gamma, \delta, \nu, q_0 \rangle$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite output alphabet, q_0 in Q is the initial state, $\delta : Q \times \Sigma \to Q$ is a total transition function, and $\nu : Q \times \Sigma \to \Gamma$ is a total output function.

It is evident that every complete and finite enumeration corresponds to a Mealy machine, however, the converse is not true. Not every Mealy machine is obtainable through an enumeration. Those that can be obtained from complete and finite enumerations form a proper subset of all Mealy machines. Based on the additional properties shared among this subset and to ensure a 1-1 correspondence from all complete and finite enumerations onto it, we define an enumeration Mealy machine as a Mealy machine $\langle Q, \Sigma, \Gamma, \delta, \nu, q_0 \rangle$ satisfying the following constraints. First Γ has to properly contain $\{0, \omega\}$, and Σ has to be associated with a total order $<$ (alphabetical order), which can be further extended to Σ^* first by length, and then alphabetically. We extend ν as usual to sequences, assuming λ is mapped to 0 by the extended output function, and require five more conditions hold:

Condition 1. States are named q_0, q_1, ..., q_{n-1} if Q has n states;

Condition 2. Every state is reachable from the initial state (i.e., the automaton is connected);

Condition 3. For every state we compute a word for it that is the first word in canonical order taking the automaton from the initial state to the state in question, then a state with a higher index has a "greater" string associated according to the total ordering on Σ^*;

Condition 4. A state becomes a trap state if the computed word for it (i.e., its associated string) has ω as its output by the extended output function;

Condition 5. All outgoing arcs of a state have ω as output if at least one incoming arc has ω as output.

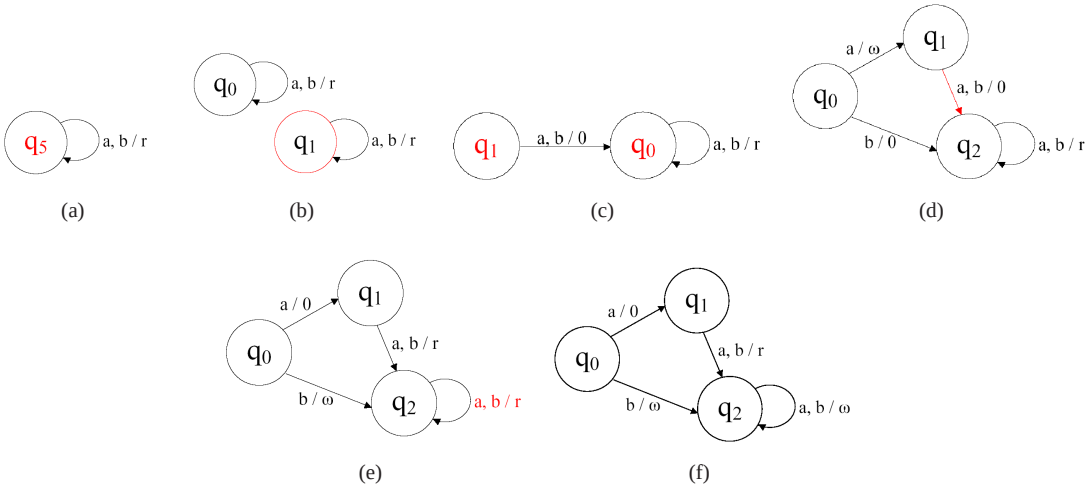


Figure 8: Diagrams of six Mealy machines that share an input alphabet $\Sigma = \{a, b\}$ with $a < b$ and an output alphabet $\Gamma = \{0, r, \omega\}$. (a) M_1. (b) M_2. (c) M_3. (d) M_4. (e) M_5. (f) M_6.

Among these conditions 1 and 3 together set the rules for labeling states; they ensure the 1-1 correspondence from the set of complete and finite enumerations onto the set of enumeration Mealy machines.

Suppose we have six Mealy machines $M_1 - M_6$ with diagrams shown in Fig. 8. They share an input alphabet $\Sigma = \{a, b\}$ with $a < b$ defining a total order on Σ, and an output alphabet $\Gamma = \{0, r, \omega\}$. Among them only M_6 qualifies as an enumeration Mealy machine; $M_1 - M_5$ fail to satisfy Conditions 1-5 respectively.

In [17] we first give two algorithms, one for converting any complete and finite enumeration to an enumeration Mealy machine, the other for converting any enumeration Mealy machine to a complete and finite enumeration, then prove a representation theorem that asserts the two transformations as inverse transformations and establishes the 1-1 correspondence between the two sets of mathematical objects. The direct transformations between complete and finite enumerations and enumeration Mealy machines offer insight into the requirements change algorithms that follow.

Managing Requirements Changes

Requirements change both during and after a phase of development. Algorithms for managing requirements changes meet a very great need in field applications of sequence-based specifications. Changes of requirements may affect an existing enumeration in different ways. We categorize them as follows and consider one change of one type at a time when it comes to algorithm design.

The stimulus set could be changed as we identify a new stimulus across the system boundary or an old one no longer of interest. Possible changes of stimuli include adding a stimulus into the stimulus set, deleting a stimulus from the stimulus set, and combinations of stimulus addition and deletion in any order. We are curious about the impact of these operations on specifications as well as the relevance or irrelevance of order to the results.

The response mapping could be changed as we identify a new response for a sequence in the enumeration. The new response could be different from any element of the old response set and emerge from the new or changed requirements. A response change refers to changing the response value of a specific sequence in a complete and finite enumeration and handling all its consequences. Depending on the legality of this sequence before and after the change, the response change is classified as from legal to legal, legal to illegal, or illegal to legal; the latter two cases

Table 1: Summary of possible requirements changes.

stimulus changes	adding a stimulus		Algorithm 7
	deleting a stimulus		Algorithm 1
	combinations of addition and deletion		
response changes	from legal to legal		Algorithm 2
	from illegal to legal	for an unreduced sequence	Algorithm 5
		for a reduced sequence	Algorithm 8
	from legal to illegal	for a reduced sequence	Algorithm 9
		for an unreduced sequence	Algorithm 10
equivalence changes	for an unreduced illegal sequence		Algorithm 6
	for an unreduced legal sequence		Algorithm 3
	for a reduced illegal sequence		Algorithm 11
	for a reduced legal sequence	keeping it reduced	Algorithm 12
		making it unreduced	Algorithm 4

are also considered as legality changes. We must identify completely portions of the existing enumeration that need a corresponding change.

Similarly the reduction could be changed as we identify a new reduced value for a sequence in the enumeration. An equivalence change refers to changing the reduction (i.e., declared equivalence) of a specific sequence in a complete and finite enumeration and handling all its consequences. Here we try to capture all changes incurred by any single equivalence change.

We summarize all possible requirements changes to be discussed in Table 1. As mentioned before, our work is characterized by the systematic enumeration to construct and modify Mealy machines.

Part of the difficulty involved with managing requirements changes lies in the clarification and a precise specification of the problem itself. In any case when a change is to be made to an existing enumeration, if the specifier started all over again they would need only the following two types of information to complete the new specification:

- response of a certain sequence according to the new requirements;
- whether equivalence to a prior sequence should be claimed for a certain sequence according to the new requirements.

In automating this process such information should be obtained from the specifier through their understanding and interpretation of the new requirements if it cannot be obtained from the old enumeration or if the old enumeration suggests more than one possibility. Any assumption made by the algorithm to avoid or minimize interaction with the specifier needs to be explicitly stated.

To be precise the requirements changes we are to manage are essentially concrete changes to be made to an existing specification that reflect changes in the old requirements. They may suggest small or big changes that ripple through the specification under different assumptions about the new requirements. The key to solving the problem is in figuring out what else in the existing specification need or need not be changed, and under what assumptions.

Changes made to the old enumeration can be put in the following two categories:

- changes that the specification tool can make without any human intervention;
- changes that the specifier must make (e.g., extensions) or should consider (if questions need to be asked to make the changes).

In the analysis we make certain assumptions when needed, state all assumptions made in deriving the new specification, and highlight entries in the new enumeration that might be questionable under different assumptions. In this way we are always maintaining and evolving old specifications, while dealing with atomic changes one at a time, and keeping the specifier informed of portions of the current enumeration that may need further changes until the final specification is derived.

A Running Example: SAFE and Z-SAFE

Our algorithms for managing requirements changes are to be illustrated through a running example. To make it more interesting let us suppose we have a new feature added to our safe controller. The new one has an additional input Z which is an employee ID given by a finger print scan. Now it takes the combination entry plus a valid finger print ID (in whatever order) to unlock the safe. Setting up these two systems, we will show step by step how to derive one from the other by applying our twelve algorithms.

As noticed from the safe enumeration we might encounter entries "- - - - - - - - - - - -" in the equivalence column. This representation is used to make the display less visually cluttered and means to the person doing the work that the sequence is not reduced to a prior sequence. In our theoretical treatment, this means that the sequence is actually reduced to itself. Furthermore, in keeping with the theory, we make the first illegal sequence reduce to itself, and all other illegals reduce to the first illegal sequence. Now our safe enumeration takes the form in Table 2, and the algorithms that follow can be applied.

Likewise, the enumeration for the Z-SAFE (Table 3) and its state machine (Fig. 9) can be obtained independently. A few requirements are updated (or derived through enumeration) for the new system:

3: Once the three digits of the combination are entered in the correct order and a finger print scan is validated (before or after the combination entry), the safe unlocks and the door may be opened.

6: The safe ignores keypad entry and finger print scan when the door is open.

8: It is assumed (with risk) that the safe cannot be opened by means other than combination entry and valid finger print scan while the software is running.

D3: Re-scanning of finger prints makes previous scans irrelevant.

We introduce a predicate p for finger print accepted. As shown in [1], predicate refinement is a handy technique frequently used in enumerations to handle one form of non-determinism.

Table 3: Z-SAFE enumeration.

Sequence	Response	Equivalence	Trace
λ	0	λ	Method
B	ω	B	D1
C	ω	B	D1
D	ω	B	D1
G	ω	B	D1
L	0	L	5
U	0	U	5
Z	ω	B	D1
LB	0	LB	1,2,7
LC	0	L	2,7
LD	ω	B	8

LG	0	LG	1,3,7
LL	0	L	5,D2
LU	0	U	5,D2
$L[Z,p]$	0	$L[Z,p]$	8
$L[Z,\neg p]$	0	L	8
UB	0	U	6
UC	0	U	6
UD	$lock$	L	4
UG	0	U	6
UL	0	L	5,D2
UU	0	U	5,D2
UZ	0	U	6
LBB	0	LB	2,7
LBC	0	L	2,7
LBD	ω	B	8
LBG	0	LB	2,7
LBL	0	L	5,D2
LBU	0	U	5,D2
$LB[Z,p]$	0	$LB[Z,p]$	8
$LB[Z,\neg p]$	0	LB	8
LGB	0	LG	8
LGC	0	L	2
LGD	ω	B	8
LGG	0	LG	8
LGL	0	L	5,D2
LGU	0	U	5,D2
$LG[Z,p]$	$unlock$	U	1,3,7
$LG[Z,\neg p]$	0	LG	8
$L[Z,p]B$	0	$LB[Z,p]$	8
$L[Z,p]C$	0	$L[Z,p]$	2,8
$L[Z,p]D$	ω	B	8
$L[Z,p]G$	$unlock$	U	1,3,7
$L[Z,p]L$	0	L	5,D2
$L[Z,p]U$	0	U	5,D2
$L[Z,p][Z,p]$	0	$L[Z,p]$	8,D3
$L[Z,p][Z,\neg p]$	0	L	8,D3
$LB[Z,p]B$	0	$LB[Z,p]$	2,8
$LB[Z,p]C$	0	$L[Z,p]$	2,8
$LB[Z,p]D$	ω	B	8
$LB[Z,p]G$	0	$LB[Z,p]$	2,8
$LB[Z,p]L$	0	L	5,D2
$LB[Z,p]U$	0	U	5,D2
$LB[Z,p][Z,p]$	0	$LB[Z,p]$	2,8
$LB[Z,p][Z,\neg p]$	0	LB	8,D3

Table 2: SAFE enumeration.

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

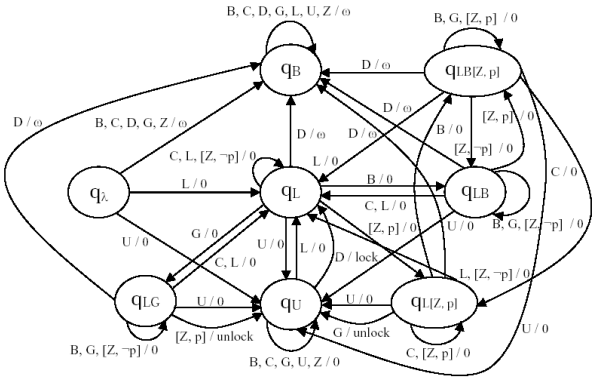


Figure 9: A state machine diagram for Z-SAFE.

Each of our twelve algorithms handles one type of requirements changes (under one circumstance if more than one exists). One thing to note about them is that they only apply to complete and finite enumerations. In reality we can always make an incomplete enumeration complete before applying any algorithm by mapping all undefined extensions to *illegal*. Outputs of these algorithms are proven to be complete and finite enumerations [17].

For the first part of the example let us start with Z-SAFE and reduce it to the original. Stimulus Z will have to be removed.

Deleting A Stimulus: Algorithm 1

As mentioned before, the representation theorem enables us to look at enumerations by way of automata. Most of our change algorithms were guided by changes to the corresponding state machines. Since every unreduced sequence in a complete and finite enumeration relates to a state of the underlying Mealy machine in such a way that it is the first string in canonical order taking the automaton from the initial state to that state, we name each state after its corresponding unreduced sequence in the automaton diagrams.

When we delete stimulus x (see Algorithm 1), in view of the automaton all arcs labeled with x will disappear, which may render some states unreachable from the initial state and being removed. For states that remain they might only be reachable from the initial state by strings "greater" than their original names, as the previous "smallest" paths may no longer exist in the new automaton. The key to this problem then lies in determining which states in the old automaton remain in the new automaton, and furthermore how they should be named in the new automaton.

Lines 3-29 of Algorithm 1 derive this partial mapping κ from old state names to new state names. First observe that a state whose name does not contain symbol x must remain in the new automaton and be named after the same string. This provides a base case for the definition of κ. Once we know a state that remains in the new automaton, we have two possibilities: either the new name outputs ω in the old automaton, or it outputs a legal response. In the former case this state must be a trap state in the new automaton. In the latter case we conclude that its successor state in the old automaton by any single arc other than the one labeled with x must remain in the new automaton as well, and be named after a string no greater than the new name of the known state concatenated with the label on the arc.

Constructing the resulting enumeration is then mechanical, as these new state names indicate the unreduced sequences in it. In Fig. 10(a) we depict transitions in the old automaton that imply a row in the new enumeration table. This row can be obtained either by rewriting an existing row in the old enumeration, or by adding a new row to the old enumeration. It represents a transition in the new automaton (Fig. 10(b)). Although the new transition looks exactly the same as the old one except for state renaming, the suggested changes in the enumeration could be much bigger and more non-intuitive to recognize. This figure restates the two rules we are to follow when building the new enumeration

Algorithm 1. (Stimulus deletion algorithm)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, a stimulus x in S.
Outputs: A complete and finite enumeration $\mathcal{E}'$ with stimulus set $S - \{x\}$.

---

1. Initialize $K_0 = \emptyset$, $K_1 = \emptyset$, $n = 0$.
2. Collect all unreduced sequences in $\mathcal{E}$ into the set U.
3. **For** each u in U
4. **If** u does not contain symbol x
5. **Then**
6. $\kappa(u) = u$ is an unreduced sequence in $\mathcal{E}'$;
7. $K_1 = K_1 \cup \{u\}$;
8. The response of u in $\mathcal{E}$ is the mapped response of $\kappa(u)$ under the black box function of $\mathcal{E}$ denoted by $\hat{\mapsto}(\kappa(u))$.
9. **Endif**
10. **Endfor**
11. **While** $K_n \neq K_{n+1}$
12. **Do**
13. Let $n = n + 1$.
14. **For** each non-empty sequence ps in $\mathcal{E}$, where s is a symbol not equal to x
15. **If** ps is reduced to sequence v in $\mathcal{E}$ for v not in K_n
16. **Then**
17. **If** p is in K_n with $\hat{\mapsto}(\kappa(p)) \neq \omega$
18. **Then**
19. Let $\kappa(p)s$ be a candidate for $\kappa(v)$.
20. **Endif**
21. **Endif**
22. **Endfor**
23. Initialize $K_{n+1} = K_n$.
24. **For** each v that are designated candidate values for the κ mapping in Steps 14-22
25. Choose the first candidate in canonical order as $\kappa(v)$;
26. $K_{n+1} = K_{n+1} \cup \{v\}$;
27. Compute the mapped response of $\kappa(v)$ under the black box function of $\mathcal{E}$, denoted by $\hat{\mapsto}(\kappa(v))$.
28. **Endfor**
29. **Enddo**
30. Initialize $\mathcal{E}'$ to contain the empty sequence λ only, with λ mapped to 0 and reduced to λ.
31. **For** each non-empty sequence us in $\mathcal{E}$ mapped to r and reduced to v, where s is a symbol not equal to x
32. **If** $\kappa(u)$ is defined with $\hat{\mapsto}(\kappa(u)) \neq \omega$
33. **Then**
34. Add a row for sequence $\kappa(u)s$ into $\mathcal{E}'$, mapping $\kappa(u)s$ to r and reducing $\kappa(u)s$ to $\kappa(v)$.
35. **Endif**
36. **Endfor**
37. **For** each unreduced illegal sequence u in $\mathcal{E}$
38. **If** $\kappa(u)$ is defined with $\hat{\mapsto}(\kappa(u)) \neq \omega$
39. **Then**
40. **For** each s in S not equal to x
41. Add a row for sequence $\kappa(u)s$ into $\mathcal{E}'$, mapping $\kappa(u)s$ to ω and reducing $\kappa(u)s$ to $\kappa(u)$.
42. **Endfor**
43. **Endif**
44. **Endfor**
45. Return $\mathcal{E}'$.

from the old one. We should be careful to extend in the new enumeration only extensible sequences (i.e., new state names that output a legal response in the old automaton). In case a state represents an unreduced and illegal sequence in the old enumeration but an extensible sequence in the new one, its extensions cannot be obtained by a search-and-substitution on the old enumeration entries, and have to be written out explicitly. It is worth noting that no entry ever need be highlighted for specifiers' attention.

Now we apply Algorithm 1 to delete Z. Unreduced sequences in Table 3 are identified as λ, B, L, U, LB, LG, $L[Z,p]$, $LB[Z,p]$. Among them strings not containing $[Z,p]$ or $[Z,\neg p]$ as a symbol are defined for κ and included in K_1:

$$\begin{array}{lcll} \kappa(\lambda) & = & \lambda & \hat{\mapsto} \ 0 \\ \kappa(B) & = & B & \hat{\mapsto} \ \omega \\ \kappa(L) & = & L & \hat{\mapsto} \ 0 \\ \kappa(U) & = & U & \hat{\mapsto} \ 0 \\ \kappa(LB) & = & LB & \hat{\mapsto} \ 0 \\ \kappa(LG) & = & LG & \hat{\mapsto} \ 0. \end{array}$$

They happen to be all the unreduced sequences defined for κ; the loop terminates at $K_2 = K_1$.

The derivation of $\mathcal{E}'$ from $\mathcal{E}$ and κ is shown in Table 4. See the resulting enumeration in Table 5 and the state machine in Fig. 11.

After deleting stimulus Z we notice that LG should be mapped to $unlock$ instead of 0. This requires a response change of LG from a legal value to another legal value.

Changing A Response from Legal to Legal: Algorithm 2

In any case of a response change, we encounter the need to differentiate a general sequence in the original enumeration from the one whose response is to be changed. We use the upright letter **u** in bold face to denote the latter. Likewise, we use upright boldface **r** for its new response after the change, and to differentiate it from a general response in the original enumeration. One thing we are sure about **u** is that it cannot be the empty sequence λ, whose response can only be 0 and not be changed. Hence we are able to split **u** into a prefix **p** and a symbol **s** that are both in upright boldface letters as determined by **u**.

Consider the setup in Algorithm 2. In the old automaton (Fig. 12(a)) we have an outgoing arc from state **p** to state $\triangleright(\mathbf{u})$ (i.e., the reduced value of **u** in $\mathcal{E}$) labeled with stimulus **s** and associated with a legal response $\mapsto (\mathbf{u})$. This output on the arc is to be changed to another legal output **r**. Performing this change on the arc gives us a new enumeration Mealy machine (Fig. 12(b)).

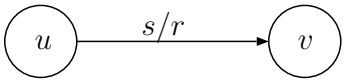
Applying Algorithm 2 to Table 5 we arrive at the enumeration in Table 6. This is a single change to a single sequence. The state machine looks the same as Fig. 11 except that the arc from q_L to q_{LG} labeled with G is now associated with output $unlock$.

Our next task is to change the equivalence of LG from LG to the prior sequence U. This is an equivalence change for an unreduced legal sequence.

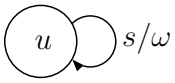
Changing An Equivalence for An Unreduced Legal Sequence: Algorithm 3

Following the notation for response changes, in any case of an equivalence change, we use upright boldface letters **u** and **v** to denote respectively the sequence whose equivalence is to be changed and its new reduced value after the change. We claim also that **u** cannot be the empty sequence λ, hence it can be split into prefix sequence **p** and current stimulus **s**, both in upright boldface letters as determined by **u**.

$$\forall u \in E. \forall s \in S. (us \mapsto r, us \triangleright v, \kappa(u) \not\hat{\mapsto} \omega, s \neq x)$$

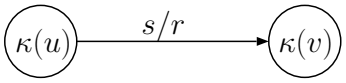


$$\forall u \in U. \forall s \in S. (u \mapsto \omega, \kappa(u) \not\hat{\mapsto} \omega, s \neq x)$$

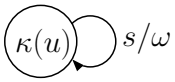


(a)

$$\forall u \in E. \forall s \in S. (us \mapsto r, us \triangleright v, \kappa(u) \not\hat{\mapsto} \omega, s \neq x)$$



$$\forall u \in U. \forall s \in S. (u \mapsto \omega, \kappa(u) \not\hat{\mapsto} \omega, s \neq x)$$



(b)

Figure 10: Automaton diagrams for deleting stimulus x. (a) Before deleting x. (b) After deleting x.

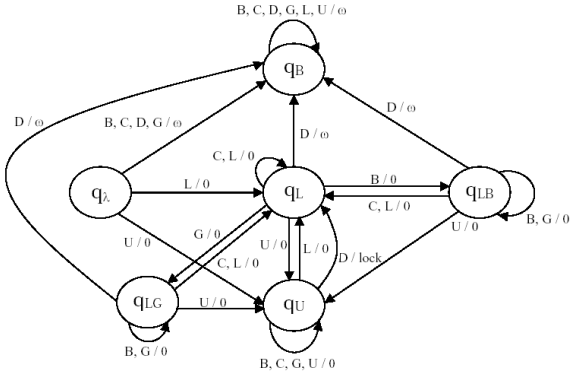


Figure 11: A state machine diagram for Z-SAFE after deleting Z.

Table 4: Derivation of $\mathcal{E}'$ from $\mathcal{E}$ and κ for deleting stimulus Z from Z-SAFE.

$\mathcal{E}(\lambda) = (0, \lambda)$	$\mathcal{E}'(\lambda) = (0, \lambda)$
$\mathcal{E}(B) = (\omega, B)$	$\mathcal{E}'(B) = (\omega, B)$
$\mathcal{E}(C) = (\omega, B)$	$\mathcal{E}'(C) = (\omega, B)$
$\mathcal{E}(D) = (\omega, B)$	$\mathcal{E}'(D) = (\omega, B)$
$\mathcal{E}(G) = (\omega, B)$	$\mathcal{E}'(G) = (\omega, B)$
$\mathcal{E}(L) = (0, L)$	$\mathcal{E}'(L) = (0, L)$
$\mathcal{E}(U) = (0, U)$	$\mathcal{E}'(U) = (0, U)$
$\mathcal{E}(LB) = (0, LB)$	$\mathcal{E}'(LB) = (0, LB)$
$\mathcal{E}(LC) = (0, L)$	$\mathcal{E}'(LC) = (0, L)$
$\mathcal{E}(LD) = (\omega, B)$	$\mathcal{E}'(LD) = (\omega, B)$
$\mathcal{E}(LG) = (0, LG)$	$\mathcal{E}'(LG) = (0, LG)$
$\mathcal{E}(LL) = (0, L)$	$\mathcal{E}'(LL) = (0, L)$
$\mathcal{E}(LU) = (0, U)$	$\mathcal{E}'(LU) = (0, U)$
$\mathcal{E}(UB) = (0, U)$	$\mathcal{E}'(UB) = (0, U)$
$\mathcal{E}(UC) = (0, U)$	$\mathcal{E}'(UC) = (0, U)$
$\mathcal{E}(UD) = (lock, L)$	$\mathcal{E}'(UD) = (lock, L)$
$\mathcal{E}(UG) = (0, U)$	$\mathcal{E}'(UG) = (0, U)$
$\mathcal{E}(UL) = (0, L)$	$\mathcal{E}'(UL) = (0, L)$
$\mathcal{E}(UU) = (0, U)$	$\mathcal{E}'(UU) = (0, U)$
$\mathcal{E}(LBB) = (0, LB)$	$\mathcal{E}'(LBB) = (0, LB)$
$\mathcal{E}(LBC) = (0, L)$	$\mathcal{E}'(LBC) = (0, L)$
$\mathcal{E}(LBD) = (\omega, B)$	$\mathcal{E}'(LBD) = (\omega, B)$
$\mathcal{E}(LBG) = (0, LB)$	$\mathcal{E}'(LBG) = (0, LB)$
$\mathcal{E}(LBL) = (0, L)$	$\mathcal{E}'(LBL) = (0, L)$
$\mathcal{E}(LBU) = (0, U)$	$\mathcal{E}'(LBU) = (0, U)$
$\mathcal{E}(LGB) = (0, LG)$	$\mathcal{E}'(LGB) = (0, LG)$
$\mathcal{E}(LGC) = (0, L)$	$\mathcal{E}'(LGC) = (0, L)$
$\mathcal{E}(LGD) = (\omega, B)$	$\mathcal{E}'(LGD) = (\omega, B)$
$\mathcal{E}(LGG) = (0, LG)$	$\mathcal{E}'(LGG) = (0, LG)$
$\mathcal{E}(LGL) = (0, L)$	$\mathcal{E}'(LGL) = (0, L)$
$\mathcal{E}(LGU) = (0, U)$	$\mathcal{E}'(LGU) = (0, U)$

Table 5: Z-SAFE enumeration after deleting stimulus Z.

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

Algorithm 2. (Response change algorithm – from legal to legal)
Inputs: A complete and finite enumeration $\mathcal{E}$, a legal response **r**, a non-empty sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to a legal response other than **r**.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is mapped to **r**.

---

1. Initialize $\mathcal{E}'$ to contain no sequence.
2. **For** each sequence u in $\mathcal{E}$ mapped to r and reduced to v
3. **If** u = **u**
4. **Then**
5. Add a row for sequence u into $\mathcal{E}'$, mapping u to **r** and reducing u to v.
6. **Else**
7. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to v.
8. **Endif**
9. **Endfor**
10. Return $\mathcal{E}'$.

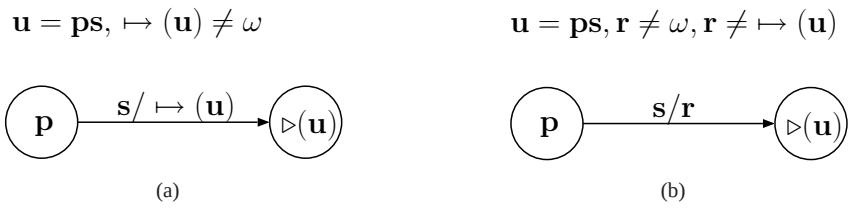


Figure 12: Automaton diagrams for changing the response of **u** from legal to legal. (a) Before the response change. (b) After the response change.

Table 6: Z-SAFE enumeration after deleting stimulus Z and changing the response of LG from 0 to $unlock$.

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

Algorithm 3. (Equivalence change algorithm – for an unreduced legal sequence)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, a non-empty unreduced sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to a legal response, an unreduced prior sequence **v** in $\mathcal{E}$.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is reduced to **v**.

---

1. Initialize $K_0 = \emptyset$, $K_1 = \emptyset$, $n = 0$.
2. Collect all unreduced sequences in $\mathcal{E}$ into the set U.
3. **For** each u in U
4. **If** u does not contain **u** as a prefix
5. **Then**
6. $\kappa(u) = u$ is an unreduced sequence in $\mathcal{E}'$;
7. $K_1 = K_1 \cup \{u\}$;
8. The response of u in $\mathcal{E}$ is the mapped response of $\kappa(u)$ under the black box function of $\mathcal{E}$ denoted by $\hat{\mapsto}(\kappa(u))$.
9. **Endif**
10. **Endfor**
11. **While** $K_n \neq K_{n+1}$
12. **Do**
13. Let $n = n + 1$.
14. **For** each non-empty sequence ps other than **u** in $\mathcal{E}$, where s is a stimulus
15. **If** ps is reduced to sequence v in $\mathcal{E}$ for v not in K_n
16. **Then**
17. **If** p is in K_n with $\hat{\mapsto}(\kappa(p)) \neq \omega$
18. **Then**
19. Let $\kappa(p)s$ be a candidate for $\kappa(v)$.
20. **Endif**
21. **Endif**
22. **Endfor**
23. Initialize $K_{n+1} = K_n$.
24. **For** each v that are designated candidate values for the κ mapping in Steps 14-22
25. Choose the first candidate in canonical order as $\kappa(v)$;
26. $K_{n+1} = K_{n+1} \cup \{v\}$;
27. Compute the mapped response of $\kappa(v)$ under the black box function of $\mathcal{E}$, denoted by $\hat{\mapsto}(\kappa(v))$.
28. **Endfor**
29. **Enddo**
30. Initialize $\mathcal{E}'$ to contain the empty sequence λ only, with λ mapped to 0 and reduced to λ.
31. Add a row for sequence **u** into $\mathcal{E}'$, mapping **u** to its response in $\mathcal{E}$ and reducing **u** to **v**.
32. **For** each non-empty sequence ux other than **u** in $\mathcal{E}$ mapped to r and reduced to v, where x is a stimulus
33. **If** $\kappa(u)$ is defined with $\hat{\mapsto}(\kappa(u)) \neq \omega$
34. **Then**
35. Add a row for sequence $\kappa(u)x$ into $\mathcal{E}'$, mapping $\kappa(u)x$ to r and reducing $\kappa(u)x$ to $\kappa(v)$.
36. **Endif**
37. **Endfor**
38. **For** each unreduced illegal sequence u in $\mathcal{E}$
39. **If** $\kappa(u)$ is defined with $\hat{\mapsto}(\kappa(u)) \neq \omega$
40. **Then**
41. **For** each stimulus x
42. Add a row for sequence $\kappa(u)x$ into $\mathcal{E}'$, mapping $\kappa(u)x$ to ω and reducing $\kappa(u)x$ to $\kappa(u)$.
43. **Endfor**
44. **Endif**
45. **Endfor**
46. Return $\mathcal{E}'$.

Consider the setup in Algorithm 3. Since **u** is unreduced and legal, it must have been extended by every stimulus. Any prior sequence in $\mathcal{E}$ that is unreduced could be a candidate for the new reduced value **v** of **u**. When we map **u** to **v** the outgoing arc from state **p** labeled with **s** is redirected to state **v** (Fig. 13), hence states in the old automaton whose names contain **u** as a prefix may or may not be reachable from the initial state. We note that these states correspond to unreduced sequences in $\mathcal{E}$ that are either **u** or extensions of **u**.

Applying a similar strategy as for stimulus deletion, we compute a partial mapping κ from old state names to new state names. Basically if a state is still reachable from the initial state in the new diagram, it is defined for κ with the mapped value being the first sequence in canonical order taking the new automaton from the initial state to this state, otherwise, it is not defined for κ representing a state that gets removed. In essence for each state whose name contains **u** as a prefix, we check if there exists a path that does not contain **u** as a prefix from the initial state to it in the old automaton such that no illegal output is generated along this path (otherwise some intermediate states may become trap states and the path may be interrupted in the new automaton). We also search for the smallest possible paths among all qualified ones and name the states after them.

Once we figure out κ, we have all unreduced sequences in $\mathcal{E}'$, from which constructing $\mathcal{E}'$ is straightforward by applying the rules in Lines 30-45 of Algorithm 3 in a similar fashion as for stimulus deletion, further illustrated in Fig. 13.

Now we apply Algorithm 3 to Table 6. Unreduced sequences are identified as λ, B, L, U, LB, LG. Strings that do not contain LG as a prefix are defined for κ and included in K_1:

$$\begin{array}{lcll} \kappa(\lambda) & = & \lambda & \hat{\mapsto} \ 0 \\ \kappa(B) & = & B & \hat{\mapsto} \ \omega \\ \kappa(L) & = & L & \hat{\mapsto} \ 0 \\ \kappa(U) & = & U & \hat{\mapsto} \ 0 \\ \kappa(LB) & = & LB & \hat{\mapsto} \ 0. \end{array}$$

They happen to be all the unreduced sequences defined for κ; the loop terminates at $K_2 = K_1$.

The derivation of $\mathcal{E}'$ from $\mathcal{E}$ and κ is mechanical and omitted. As a result we get the safe enumeration in Table 2 and its state machine in Fig. 7.

For the second half of the example we start with the original and build it up to the Z-SAFE. The first change made is the response of LG from $unlock$ to 0.

Applying Algorithm 2 we get a slightly different enumeration than Table 2 (LG is mapped to 0 instead of $unlock$) and a slightly different state machine than Fig. 7 (the arc from q_L to q_U labeled with G is associated with output 0 instead of $unlock$). Next LG is considered for an equivalence change, as LGD would map to *illegal* in the new system but UD would map to $lock$.

After some thought we decide to reduce LG to itself, as it represents a state that confirms a correct combination entry and awaits a valid finger print scan. This is an equivalence change for a reduced legal sequence to make it unreduced.

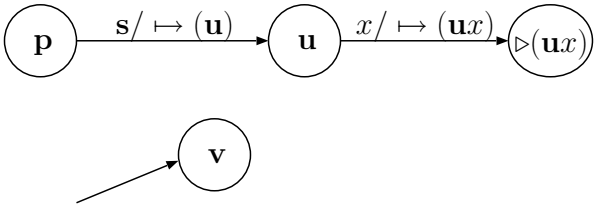
Changing An Equivalence for A Reduced Legal Sequence and Making it Unreduced: Algorithm 4

Consider the setup in Algorithm 4. Since **u** is reduced and legal, when we make it an unreduced sequence in $\mathcal{E}'$, it needs to be extended by every stimulus. We simply assume all extensions are mapped to *illegal* and reduced to themselves to keep the solution simple and neutral (Fig. 14). Meanwhile we highlight all the new extensions to inform the user of the need to address these sequences in a short while.

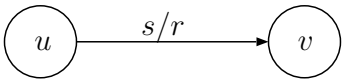
After applying Algorithm 4 we have the enumeration in Table 7 and the state machine in Fig. 15.

Now it is time to work on the highlighted entries. First we would like to change the response of LGB from ω to 0. This is a response change from illegal to legal for an unreduced sequence.

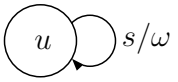
$\mathbf{u} = \mathbf{ps}, \mapsto (\mathbf{u}) \neq \omega, \mathbf{v} \in U, \mathbf{v} < \mathbf{u}$
$\forall x \in S$



$\forall u \in E. \forall s \in S. (us \mapsto r, us \triangleright v, us \neq \mathbf{u}, \kappa(u) \hat{\not\mapsto} \omega)$

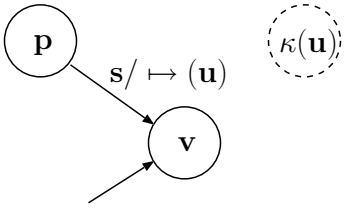


$\forall u \in U. \forall s \in S. (u \mapsto \omega, \kappa(u) \hat{\not\mapsto} \omega)$

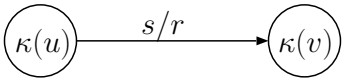


(a)

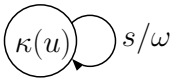
$\mathbf{u} = \mathbf{ps}, \mapsto (\mathbf{u}) \neq \omega, \mathbf{v} \in U, \mathbf{v} < \mathbf{u}$



$\forall u \in E. \forall s \in S. (us \mapsto r, us \triangleright v, us \neq \mathbf{u}, \kappa(u) \hat{\not\mapsto} \omega)$



$\forall u \in U. \forall s \in S. (u \mapsto \omega, \kappa(u) \hat{\not\mapsto} \omega)$



(b)

Figure 13: Automaton diagrams for changing the equivalence of **u** to **v** for unreduced legal **u**. (a) Before the equivalence change. (b) After the equivalence change. State $\kappa(\mathbf{u})$ in dashed line indicates that the state does not exist if **u** is not defined for κ.

Algorithm 4. (Equivalence change algorithm – for a reduced legal sequence which becomes unreduced)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, a reduced sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to a legal response, **v** being **u** itself.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is reduced to **v**.

---

1. Initialize $\mathcal{E}'$ to contain no sequence.
2. **For** each sequence u in $\mathcal{E}$ mapped to r and reduced to v
3. **If** u = **u**
4. **Then**
5. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to **v**.
6. **For** each stimulus x
7. Add a row for sequence ux into $\mathcal{E}'$, mapping ux to ω and reducing ux to ux;
8. Highlight the row for sequence ux in $\mathcal{E}'$.
9. **Endfor**
10. **Else**
11. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to v.
12. **Endif**
13. **Endfor**
14. Return $\mathcal{E}'$.

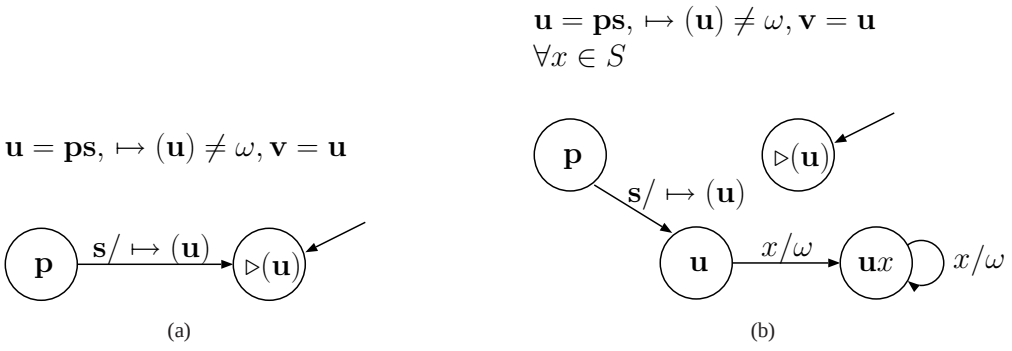


Figure 14: Automaton diagrams for changing the equivalence of **u** to **v** for reduced legal **u** and making it unreduced. (a) Before the equivalence change. (b) After the equivalence change.

Table 7: SAFE enumeration after changing the response of LG from $unlock$ to 0 and the equivalence of LG from U to LG.

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

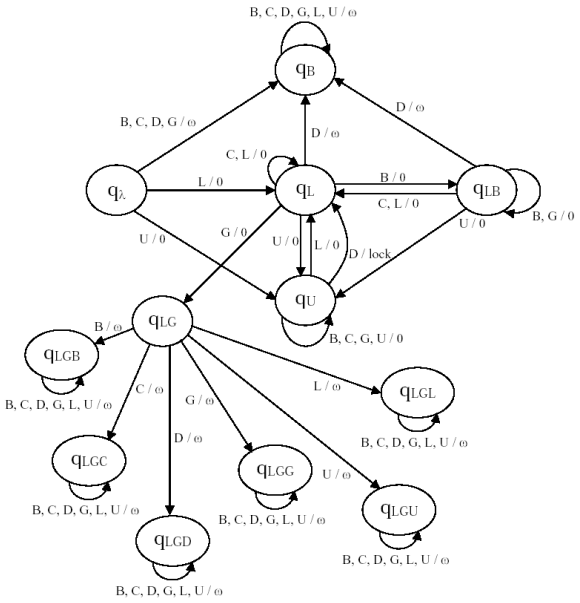


Figure 15: A state machine diagram for the safe controller after changing the response of LG from $unlock$ to 0 and the equivalence of LG from U to LG.

Algorithm 5. (Response change algorithm – from illegal to legal for an unreduced sequence)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, a legal response **r**, an unreduced sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to ω.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is mapped to **r**.

---

1. Initialize $\mathcal{E}'$ to contain no sequence.
2. **For** each sequence u in $\mathcal{E}$ mapped to r and reduced to v
3. **If** u = **u**
4. **Then**
5. Add a row for sequence u into $\mathcal{E}'$, mapping u to **r** and reducing u to u.
6. **For** each stimulus x
7. Add a row for sequence ux into $\mathcal{E}'$, mapping ux to ω and reducing ux to ux;
8. Highlight the row for sequence ux in $\mathcal{E}'$.
9. **Endfor**
10. **Else**
11. **If** $r = \omega$ and v = **p**
12. **Then**
13. Add a row for sequence u into $\mathcal{E}'$, mapping u to ω and reducing u to u.
14. **Else**
15. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to v.
16. **Endif**
17. **Endif**
18. **Endfor**
19. Return $\mathcal{E}'$.

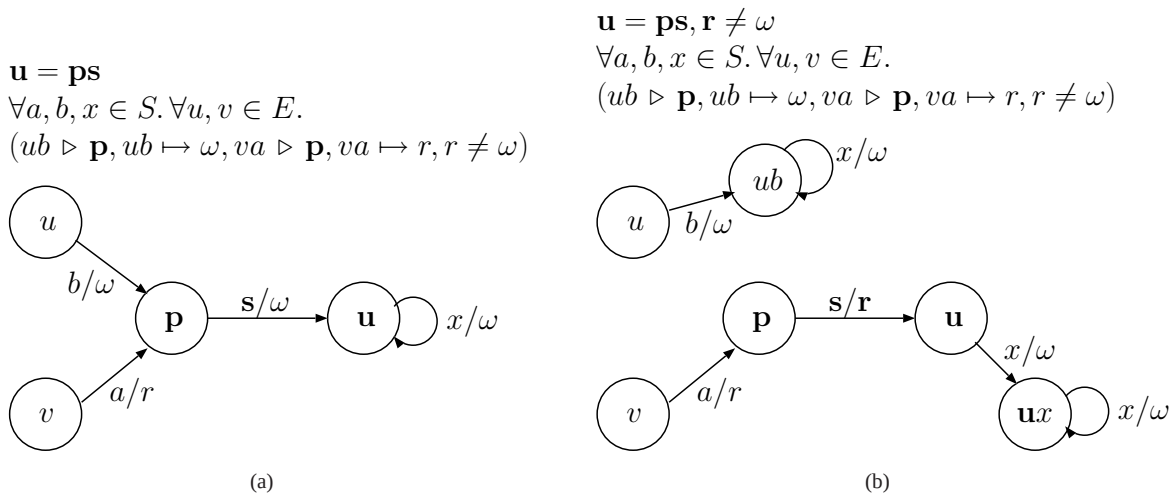


Figure 16: Automaton diagrams for changing the response of **u** from illegal to legal for unreduced **u**. (a) Before the response change. (b) After the response change.

Changing A Response from Illegal to Legal for An Unreduced Sequence: Algorithm 5

Consider the setup in Algorithm 5. Since **u** is unreduced, after we change its response from ω to a legal value **r** it becomes an extensible sequence that needs to be extended. As before we make all extensions mapped to *illegal*, reduced to themselves, and highlighted. We also check for illegal sequences that are reduced to **p** in $\mathcal{E}$. If any such sequence exists, it cannot be reduced to **p** in $\mathcal{E}'$, as **ps** would no longer map to *illegal*. We reduce all such sequences to themselves and keep the rest of the enumeration the same (Fig. 16).

Applying Algorithm 5 to Table 7 produces the enumeration in Table 8.

Then we apply Algorithm 3 for an equivalence change of LGB from LGB to LG and obtain Table 9. The state machine is shown in Fig. 17.

Similarly as for LGB we complete the response and the equivalence changes for LGC, LGG, LGL, and LGU by applying Algorithms 5 and 4 together. The resulting enumeration and state machine are shown in Table 10 and Fig. 18 respectively.

For the remaining highlighted sequence LGD, we want to change its equivalence to the first illegal sequence B. This is an equivalence change for an unreduced illegal sequence.

Changing An Equivalence for An Unreduced Illegal Sequence: Algorithm 6

Consider the setup in Algorithm 6. Since **u** is unreduced and illegal, the new reduced value **v** of **u** has to satisfy several conditions so that the change will be meaningful and not contradict Axioms 2, 3, and 6 for an enumeration. In particular **v** has to be a prior unreduced sequence that either maps to *illegal* or has all extensions map to *illegal*. We simply check every sequence in $\mathcal{E}$; if it is reduced to **u** we reduce it to **v** in $\mathcal{E}'$. Hence **u** becomes a reduced sequence in $\mathcal{E}'$. State **u** becomes an isolated state and is removed in the new automaton (Fig. 19).

Applying Algorithm 6 to Table 10 gives us the enumeration in Table 5 and the state machine in Fig. 11.

Next we work on the newly added stimulus Z.

Table 8: SAFE enumeration after changing the response of LG from $unlock$ to 0, the equivalence of LG from U to LG, and the response of LGB from ω to 0.

Sequence	Response	Equivalence	Trace
λ	0	λ	Method
B	ω	B	D1
C	ω	B	D1
D	ω	B	D1
G	ω	B	D1
L	0	L	5
U	0	U	5
LB	0	LB	1,2,7
LC	0	L	2,7
LD	ω	B	8
LG	0	LG	1,3,7
LL	0	L	5,D2
LU	0	U	5,D2
UB	0	U	6
UC	0	U	6
UD	$lock$	L	4
UG	0	U	6
UL	0	L	5,D2
UU	0	U	5,D2
LBB	0	LB	2,7
LBC	0	L	2,7
LBD	ω	B	8
LBG	0	LB	2,7
LBL	0	L	5,D2
LBU	0	U	5,D2
LGB	0	LGB	8
LGC	ω	LGC	
LGD	ω	LGD	
LGG	ω	LGG	
LGL	ω	LGL	
LGU	ω	LGU	
$LGBB$	ω	$LGBB$	
$LGBC$	ω	$LGBC$	
$LGBD$	ω	$LGBD$	
$LGBG$	ω	$LGBG$	
$LGBL$	ω	$LGBL$	
$LGBU$	ω	$LGBU$	

Table 9: SAFE enumeration after changing the response of LG from $unlock$ to 0, the equivalence of LG from U to LG, the response of LGB from ω to 0, and the equivalence of LGB from LGB to LG.

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

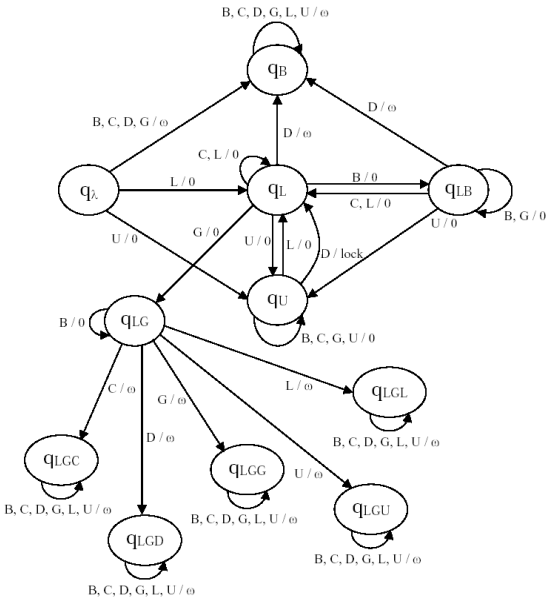


Figure 17: A state machine diagram for the safe controller after changing the response of LG from $unlock$ to 0, the equivalence of LG from U to LG, the response of LGB from ω to 0, and the equivalence of LGB from LGB to LG.

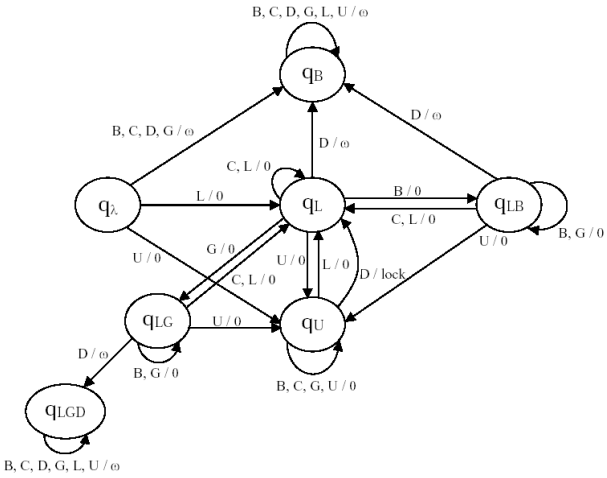


Figure 18: A state machine diagram for the safe controller after changing the response of LG from $unlock$ to 0, the equivalence of LG from U to LG, and a few more response and equivalence changes for LGB, LGC, LGG, LGL, LGU.

Table 10: SAFE enumeration after changing the response of LG from $unlock$ to 0, the equivalence of LG from U to LG, and a few more response and equivalence changes for LGB, LGC, LGG, LGL, LGU.

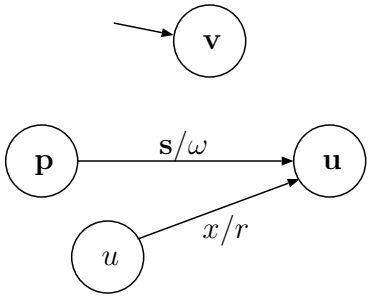
Sequence	Response	Equivalence	Trace
λ	0	λ	Method
B	ω	B	D1
C	ω	B	D1
D	ω	B	D1
G	ω	B	D1
L	0	L	5
U	0	U	5
LB	0	LB	1,2,7
LC	0	L	2,7
LD	ω	B	8
LG	0	LG	1,3,7
LL	0	L	5,D2
LU	0	U	5,D2
UB	0	U	6
UC	0	U	6
UD	$lock$	L	4
UG	0	U	6
UL	0	L	5,D2
UU	0	U	5,D2
LBB	0	LB	2,7
LBC	0	L	2,7
LBD	ω	B	8
LBG	0	LB	2,7
LBL	0	L	5,D2
LBU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

Algorithm 6. (Equivalence change algorithm – for an unreduced illegal sequence)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, an unreduced sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to ω, an unreduced prior sequence **v** that is either mapped to ω or have all one-stimulus extensions mapped to ω in $\mathcal{E}$.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is reduced to **v**.

---

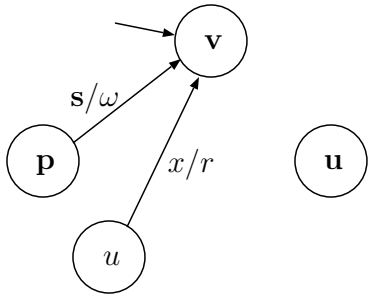
1. Initialize $\mathcal{E}'$ to contain no sequence.
2. **For** each sequence u in $\mathcal{E}$ mapped to r and reduced to v
3. **If** v = **u**
4. **Then**
5. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to **v**.
6. **Else**
7. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to v.
8. **Endif**
9. **Endfor**
10. Return $\mathcal{E}'$.

$\mathbf{u} = \mathbf{ps}, \mathbf{v} \in U, \mathbf{v} < \mathbf{u},$
$\mathbf{v} \mapsto \omega$ or $\forall a \in S. \mathbf{v}a \mapsto \omega$
$\forall u \in E. \forall x \in S. (ux \mapsto r, ux \triangleright \mathbf{u})$



(a)

$\mathbf{u} = \mathbf{ps}, \mathbf{v} \in U, \mathbf{v} < \mathbf{u},$
$\mathbf{v} \mapsto \omega$ or $\forall a \in S. \mathbf{v}a \mapsto \omega$
$\forall u \in E. \forall x \in S. (ux \mapsto r, ux \triangleright \mathbf{u})$



(b)

Figure 19: Automaton diagrams for changing the equivalence of **u** to **v** for unreduced illegal **u**. (a) Before the equivalence change. (b) After the equivalence change.

Algorithm 7. (Stimulus addition algorithm)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, a stimulus x not in S.
Outputs: A complete and finite enumeration $\mathcal{E}'$ with stimulus set $S \cup \{x\}$.

---

1. Collect all extensible sequences in $\mathcal{E}$ into the set E.
2. Initialize $\mathcal{E}'$ to contain no sequence.
3. **For** each sequence u in $\mathcal{E}$ mapped to r and reduced to v
4. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to v.
5. **Endfor**
6. **For** each u in E
7. Add a row for sequence ux into $\mathcal{E}'$, mapping ux to ω and reducing ux to ux;
8. Highlight the row for sequence ux.
9. **Endfor**
10. Return $\mathcal{E}'$.

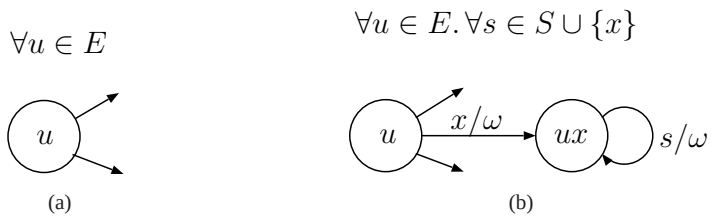


Figure 20: Automaton diagrams for adding stimulus x. (a) Before adding x. (b) After adding x.

Adding A Stimulus: Algorithm 7

When we add a new stimulus x (see Algorithm 7), every extensible sequence is to be extended by x. We simply make all extensions mapped to *illegal* and reduced to themselves (Fig. 20). These entries in the table are highlighted. One must consider each sequence and redefine them by performing response or equivalence changes as required by the new requirements.

The enumeration after adding in Z is in Table 11.

More changes (see Table 12) are performed in the listed order by applying the available algorithms and predicate refinement. The details are similar to examples already shown, hence omitted. As a consequence we obtain the Z-SAFE enumeration (Table 3) and state machine (Fig. 9); our example is completed.

For completeness we give formal statements of the algorithms not covered by our running example as well as their proofs by picture as follows.

Changing A Response from Illegal to Legal for A Reduced Sequence: Algorithm 8

This algorithm is the same as Algorithm 5 except that there are no extensions of **u** after the response change as it is a reduced sequence in $\mathcal{E}$ and we assume it remains so in $\mathcal{E}'$, as Fig. 21 illustrates.

Changing A Response from Legal to Illegal for A Reduced Sequence: Algorithm 9

When we map **u** to *illegal* we also reduce it to itself representing a newly added trap state (Fig. 22). The rest of the enumeration remains the same.

Changing A Response from Legal to Illegal for An Unreduced Sequence: Algorithm 10

Since **u** is legal and unreduced in $\mathcal{E}$, it must have been extended by every stimulus. When we map it to ω we can reduce it to itself representing a newly added trap state (Fig. 23). The outgoing arc from state **p** labeled with **s** is then redirected to this new trap state. States in the old automaton named after a string that contains **u** as a prefix may or may not be reachable from the initial state as a result. These states that will possibly be removed correspond to unreduced sequences in $\mathcal{E}$ that are either **u** or extensions of **u**.

We have encountered a very similar situation when we handle an equivalence change for an unreduced legal sequence, except that there we redirect the arc to an existing state in the automaton instead of a newly added trap state. We employ the same strategy to construct the resulting enumeration (Fig. 23).

Table 11: SAFE enumeration after all response and equivalence changes incurred by LG, and adding stimulus Z.

Sequence	Response	Equivalence	Trace
λ	0	λ	Method
B	ω	B	D1
C	ω	B	D1
D	ω	B	D1
G	ω	B	D1
L	0	L	5
U	0	U	5
Z	ω	Z	
LB	0	LB	1,2,7
LC	0	L	2,7
LD	ω	B	8
LG	0	LG	1,3,7
LL	0	L	5,D2
LU	0	U	5,D2
LZ	ω	LZ	
UB	0	U	6
UC	0	U	6
UD	$lock$	L	4
UG	0	U	6
UL	0	L	5,D2
UU	0	U	5,D2
UZ	ω	UZ	
LBB	0	LB	2,7
LBC	0	L	2,7
LBD	ω	B	8
LBG	0	LB	2,7
LBL	0	L	5,D2
LBU	0	U	5,D2
LBZ	ω	LBZ	
LGB	0	LG	8
LGC	0	L	2
LGD	ω	B	8
LGG	0	LG	8
LGL	0	L	5,D2
LGU	0	U	5,D2
LGZ	ω	LGZ	

Table 12: Remaining Changes Made to Obtain Z-SAFE.

1	changing the equivalence of Z from Z to B
2	changing the response of $L[Z, p]$ from ω to 0
3	changing the response of $L[Z, \neg p]$ from ω to 0
4	changing the equivalence of $L[Z, \neg p]$ from $L[Z, \neg p]$ to L
5	changing the response of UZ from ω to 0
6	changing the equivalence of UZ from UZ to U
7	changing the response of $LB[Z, p]$ from ω to 0
8	changing the response of $LB[Z, \neg p]$ from ω to 0
9	changing the equivalence of $LB[Z, \neg p]$ from $LB[Z, \neg p]$ to LB
10	changing the response of $LG[Z, p]$ from ω to $unlock$
11	changing the equivalence of $LG[Z, p]$ from $LG[Z, p]$ to U
12	changing the response of $LG[Z, \neg p]$ from ω to 0
13	changing the equivalence of $LG[Z, \neg p]$ from $LG[Z, \neg p]$ to LG
14	changing the response of $L[Z, p]B$ from ω to 0
15	changing the equivalence of $L[Z, p]B$ from $L[Z, p]B$ to $LB[Z, p]$
16	changing the response of $L[Z, p]C$ from ω to 0
17	changing the equivalence of $L[Z, p]C$ from $L[Z, p]C$ to $L[Z, p]$
18	changing the equivalence of $L[Z, p]D$ from $L[Z, p]D$ to B
19	changing the response of $L[Z, p]G$ from ω to $unlock$
20	changing the equivalence of $L[Z, p]G$ from $L[Z, p]G$ to U
21	changing the response of $L[Z, p]L$ from ω to 0
22	changing the equivalence of $L[Z, p]L$ from $L[Z, p]L$ to L
23	changing the response of $L[Z, p]U$ from ω to 0
24	changing the equivalence of $L[Z, p]U$ from $L[Z, p]U$ to U
25	changing the response of $L[Z, p][Z, p]$ from ω to 0
26	changing the equivalence of $L[Z, p][Z, p]$ from $L[Z, p][Z, p]$ to $L[Z, p]$
27	changing the response of $L[Z, p][Z, \neg p]$ from ω to 0
28	changing the equivalence of $L[Z, p][Z, \neg p]$ from $L[Z, p][Z, \neg p]$ to L
29	changing the response of $LB[Z, p]B$ from ω to 0
30	changing the equivalence of $LB[Z, p]B$ from $LB[Z, p]B$ to $LB[Z, p]$
31	changing the response of $LB[Z, p]C$ from ω to 0
32	changing the equivalence of $LB[Z, p]C$ from $LB[Z, p]C$ to $L[Z, p]$
33	changing the equivalence of $LB[Z, p]D$ from $LB[Z, p]D$ to B
34	changing the response of $LB[Z, p]G$ from ω to 0
35	changing the equivalence of $LB[Z, p]G$ from $LB[Z, p]G$ to $LB[Z, p]$
36	changing the response of $LB[Z, p]L$ from ω to 0
37	changing the equivalence of $LB[Z, p]L$ from $LB[Z, p]L$ to L
38	changing the response of $LB[Z, p]U$ from ω to 0
39	changing the equivalence of $LB[Z, p]U$ from $LB[Z, p]U$ to U
40	changing the response of $LB[Z, p][Z, p]$ from ω to 0
41	changing the equivalence of $LB[Z, p][Z, p]$ from $LB[Z, p][Z, p]$ to $LB[Z, p]$
42	changing the response of $LB[Z, p][Z, \neg p]$ from ω to 0
43	changing the equivalence of $LB[Z, p][Z, \neg p]$ from $LB[Z, p][Z, \neg p]$ to LB

Algorithm 8. (Response change algorithm – from illegal to legal for a reduced sequence)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, a legal response **r**, a reduced sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to ω.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is mapped to **r**.

---

1. Initialize $\mathcal{E}'$ to contain no sequence.
2. **For** each sequence u in $\mathcal{E}$ mapped to r and reduced to v
3. **If** u = **u**
4. **Then**
5. Add a row for sequence u into $\mathcal{E}'$, mapping u to **r** and reducing u to v.
6. **Else**
7. **If** $r = \omega$ and v = **p**
8. **Then**
9. Add a row for sequence u into $\mathcal{E}'$, mapping u to ω and reducing u to u.
10. **Else**
11. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to v.
12. **Endif**
13. **Endif**
14. **Endfor**
15. Return $\mathcal{E}'$.

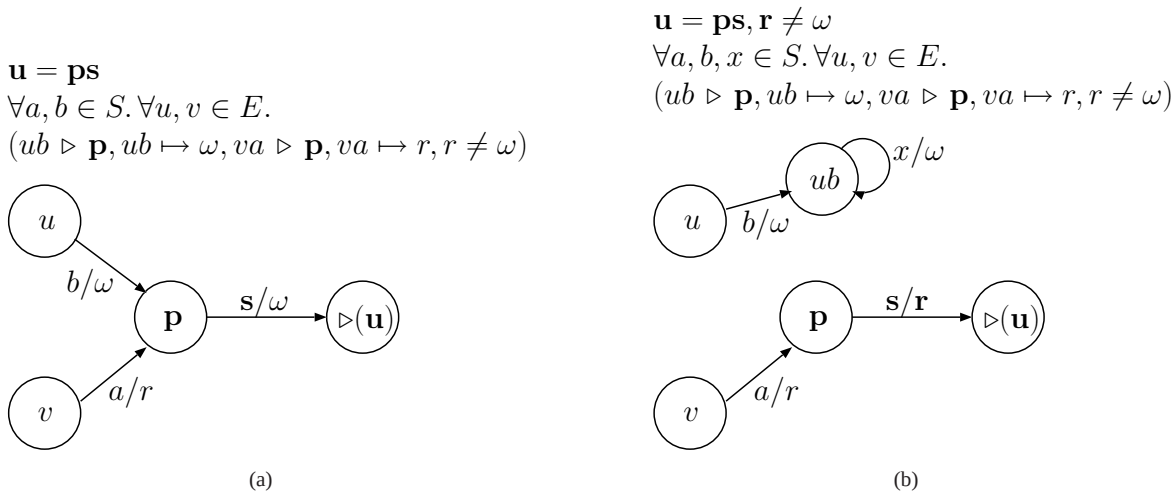


Figure 21: Automaton diagrams for changing the response of **u** from illegal to legal for reduced **u**. (a) Before the response change. (b) After the response change.

Algorithm 9. (Response change algorithm – from legal to illegal for a reduced sequence)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, a reduced sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to a legal response.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is mapped to ω.

---

1. Initialize $\mathcal{E}'$ to contain no sequence.
2. **For** each sequence u in $\mathcal{E}$ mapped to r and reduced to v
3. **If** u = **u**
4. **Then**
5. Add a row for sequence u into $\mathcal{E}'$, mapping u to ω and reducing u to u.
6. **Else**
7. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to v.
8. **Endif**
9. **Endfor**
10. Return $\mathcal{E}'$.

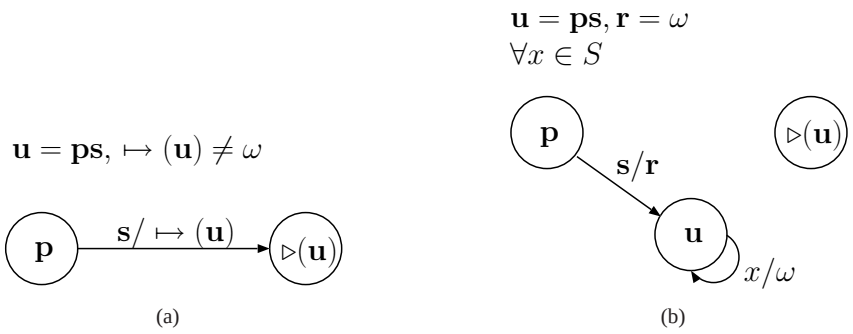


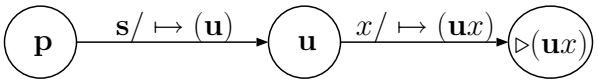
Figure 22: Automaton diagrams for changing the response of **u** from legal to illegal for reduced **u**. (a) Before the response change. (b) After the response change.

Algorithm 10. (Response change algorithm – from legal to illegal for an unreduced sequence)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, a non-empty unreduced sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to a legal response.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is mapped to ω.

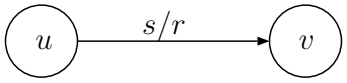
---

1. Initialize $K_0 = \emptyset$, $K_1 = \emptyset$, $n = 0$.
2. Collect all unreduced sequences in $\mathcal{E}$ into the set U.
3. **For** each u in U
4. **If** u does not contain **u** as a prefix
5. **Then**
6. $\kappa(u) = u$ is an unreduced sequence in $\mathcal{E}'$;
7. $K_1 = K_1 \cup \{u\}$;
8. The response of u in $\mathcal{E}$ is the mapped response of $\kappa(u)$ under the black box function of $\mathcal{E}$ denoted by $\hat{\mapsto}(\kappa(u))$.
9. **Endif**
10. **Endfor**
11. **While** $K_n \neq K_{n+1}$
12. **Do**
13. Let $n = n + 1$.
14. **For** each non-empty sequence ps other than **u** in $\mathcal{E}$, where s is a stimulus
15. **If** ps is reduced to sequence v in $\mathcal{E}$ for v not in K_n
16. **Then**
17. **If** p is in K_n with $\hat{\mapsto}(\kappa(p)) \neq \omega$
18. **Then**
19. Let $\kappa(p)s$ be a candidate for $\kappa(v)$.
20. **Endif**
21. **Endif**
22. **Endfor**
23. Initialize $K_{n+1} = K_n$.
24. **For** each v that are designated candidate values for the κ mapping in Steps 14-22
25. Choose the first candidate in canonical order as $\kappa(v)$;
26. $K_{n+1} = K_{n+1} \cup \{v\}$;
27. Compute the mapped response of $\kappa(v)$ under the black box function of $\mathcal{E}$, denoted by $\hat{\mapsto}(\kappa(v))$.
28. **Endfor**
29. **Enddo**
30. Initialize $\mathcal{E}'$ to contain the empty sequence λ only, with λ mapped to 0 and reduced to λ.
31. Add a row for sequence **u** into $\mathcal{E}'$, mapping **u** to ω and reducing **u** to **u**.
32. **For** each non-empty sequence ux other than **u** in $\mathcal{E}$ mapped to r and reduced to v, where x is a stimulus
33. **If** $\kappa(u)$ is defined with $\hat{\mapsto}(\kappa(u)) \neq \omega$
34. **Then**
35. Add a row for sequence $\kappa(u)x$ into $\mathcal{E}'$, mapping $\kappa(u)x$ to r and reducing $\kappa(u)x$ to $\kappa(v)$.
36. **Endif**
37. **Endfor**
38. **For** each unreduced illegal sequence u in $\mathcal{E}$
39. **If** $\kappa(u)$ is defined with $\hat{\mapsto}(\kappa(u)) \neq \omega$
40. **Then**
41. **For** each stimulus x
42. Add a row for sequence $\kappa(u)x$ into $\mathcal{E}'$, mapping $\kappa(u)x$ to ω and reducing $\kappa(u)x$ to $\kappa(u)$.
43. **Endfor**
44. **Endif**
45. **Endfor**
46. Return $\mathcal{E}'$.

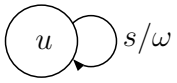
$\mathbf{u} = \mathbf{ps}, \mapsto (\mathbf{u}) \neq \omega$
$\forall x \in S$



$\forall u \in E. \forall s \in S. (us \mapsto r, us \triangleright v, us \neq \mathbf{u}, \kappa(u) \hat{\not\mapsto} \omega)$

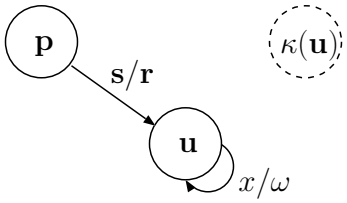


$\forall u \in U. \forall s \in S. (u \mapsto \omega, \kappa(u) \hat{\not\mapsto} \omega)$

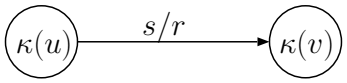


(a)

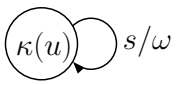
$\mathbf{u} = \mathbf{ps}, \mathbf{r} = \omega$
$\forall x \in S$



$\forall u \in E. \forall s \in S. (us \mapsto r, us \triangleright v, us \neq \mathbf{u}, \kappa(u) \hat{\not\mapsto} \omega)$



$\forall u \in U. \forall s \in S. (u \mapsto \omega, \kappa(u) \hat{\not\mapsto} \omega)$



(b)

Figure 23: Automaton diagrams for changing the response of **u** from legal to illegal for unreduced **u**. (a) Before the response change. (b) After the response change. State $\kappa(\mathbf{u})$ in dashed line indicates that the state does not exist if **u** is not defined for κ.

Algorithm 11. (Equivalence change algorithm – for a reduced illegal sequence)
Inputs: A complete and finite enumeration $\mathcal{E}$ with stimulus set S, a reduced sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to ω, an unreduced prior sequence **v** different from the reduced value of **u** in $\mathcal{E}$ that is either mapped to ω or have all one-stimulus extensions mapped to ω, or **v** being **u** itself.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is reduced to **v**.

---

1. Initialize $\mathcal{E}'$ to contain no sequence.
2. **For** each sequence u in $\mathcal{E}$ mapped to r and reduced to v
3. **If** u = **u**
4. **Then**
5. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to **v**.
6. **Else**
7. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to v.
8. **Endif**
9. **Endfor**
10. Return $\mathcal{E}'$.

Changing An Equivalence for A Reduced Illegal Sequence: Algorithm 11

Since **u** is reduced and illegal, the new reduced value **v** of **u** could be **u** itself, or any prior unreduced sequence as long as it is different from the old reduced value of **u**, and either maps to *illegal* or has all extensions map to *illegal*. Except for reducing **u** to **v** no change needs to be made to obtain $\mathcal{E}'$ (Fig. 24).

Changing An Equivalence for A Reduced Legal Sequence and Keeping it Reduced: Algorithm 12

Since **u** is a reduced legal sequence, if **u** remains as a reduced sequence after the change, any unreduced prior sequence different from the old reduced value of **u** could be chosen as its new reduced value **v**. Except for reducing **u** to **v** no change needs to be made to obtain $\mathcal{E}'$ (Fig. 25).

Combinations of Stimulus Addition and Deletion

It is proven in [17] that in case we have more than one stimulus to be added or deleted, or both addition and deletion are to be performed, the order in which these operations are applied does not affect the final result (both the enumeration and the highlighted entries, if there are any).

Summary

Sequence-based specification has in many field applications shown its effectiveness in converting informal requirements to precise specifications through a constructive process. Theory for managing changes of requirements in sequence-based specifications has a huge practical impact on maintaining specifications over time in the presence of change. In this paper we explore the change theory developed with the aid of an axiom system for sequence-based specification, and present algorithms for adding and deleting inputs, changing outputs of sequences of use, changing their legality status, and changing the equivalences that ultimately define the state space of the specification. Each change algorithm is illustrated with an example and an informal proof. The axiomatic approach turns out to be essential in developing these algorithms to help push various requirements changes through to changes in sequence-based specifications, and prove important properties of the algorithms. This has established the basis for the maximum degree of tool support for managing requirements changes for sequence-based specifications.

Sequence-based specification is a notation-free and syntax-free system, beyond giving stimuli and responses short names to facilitate enumeration. The specification tool maintains internal files (XML format) with every action and can generate the full documentation at any time. After the specification is complete, and following canonical sequence

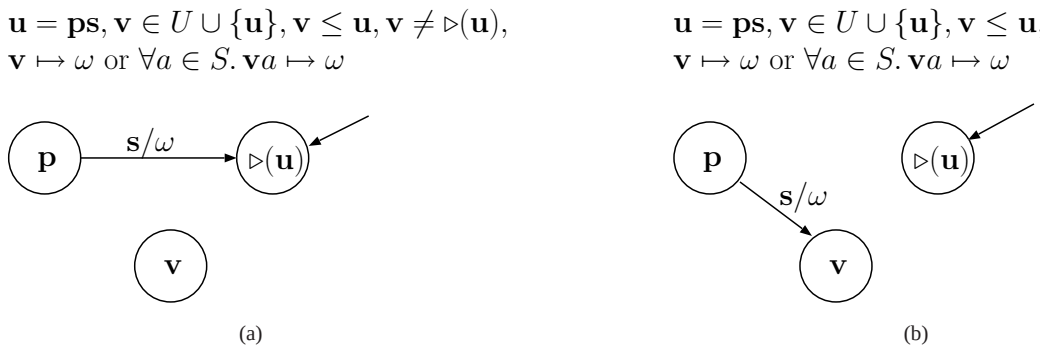


Figure 24: Automaton diagrams for changing the equivalence of **u** to **v** for reduced illegal **u**. (a) Before the equivalence change. (b) After the equivalence change.

Algorithm 12. (Equivalence change algorithm – for a reduced legal sequence which keeps reduced)
Inputs: A complete and finite enumeration $\mathcal{E}$, a reduced sequence **u** (which can be split into a prefix sequence **p** and a stimulus **s**) in $\mathcal{E}$ mapped to a legal response, an unreduced prior sequence **v** different from the reduced value of **u** in $\mathcal{E}$.
Outputs: A complete and finite enumeration $\mathcal{E}'$ in which **u** is reduced to **v**.

---

1. Initialize $\mathcal{E}'$ to contain no sequence.
2. **For** each sequence u in $\mathcal{E}$ mapped to r and reduced to v
3. **If** u = **u**
4. **Then**
5. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to **v**.
6. **Else**
7. Add a row for sequence u into $\mathcal{E}'$, mapping u to r and reducing u to v.
8. **Endif**
9. **Endfor**
10. Return $\mathcal{E}'$.

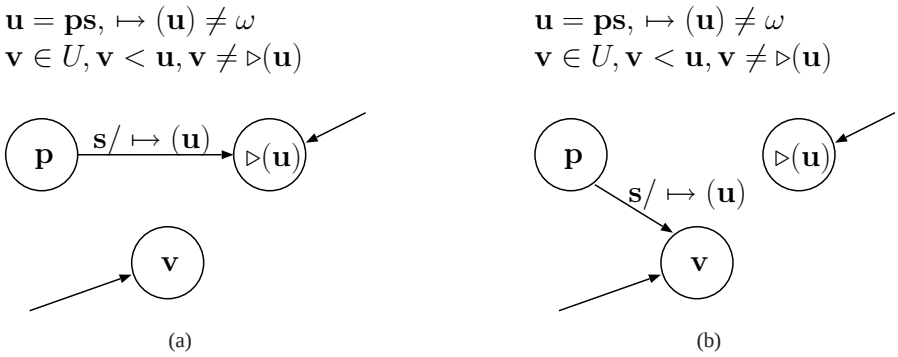


Figure 25: Automaton diagrams for changing the equivalence of **u** to **v** for reduced legal **u** and keeping it reduced. (a) Before the equivalence change. (b) After the equivalence change.

analysis, it is rendered in the form of state transition tables. Of course, the specification could be represented in various other state-based systems. Since we also have algorithms [17] to convert enumerations to and from prefix-recursive functions, and regular expression sets, the possibilities for connecting with other systems are quite extensive. For example, we generate a directed graph from the specification for use as the structure of Markov chain testing models [18]. One could also generate other tables and schema from the XML representation, as well as code and testing models [3, 5]. Sequence-based specification is all about the discovery, invention, and maintenance of the specification and agnostic about how the result is represented and used. Since most other specification notations and representations stand in a one-to-one correspondence with enumerations, and those corresponding enumerations can be produced algorithmically and tested against the sequence-based specification axioms, it follows that the consequences of changes in requirements are similar for most representations. The algorithms presented here apply.

References

[1] S. J. Prowell and J. H. Poore, "Foundations of Sequence-Based Software Specification", *IEEE Transactions on Software Engineering*, May 2003, 29(5), pp.417-429.

[2] S. J. Prowell and J. H. Poore, "Sequence-Based Software Specification of Deterministic Systems", *Software - Practice and Experience*, March 1998, 28(3), pp.329-344.

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

[4] S. J. Prowell and T. W. Swain, "Sequence-Based Specification of Critical Software Systems", *Proceedings of the Fourth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls and Human-Machine Interface Technology (NPIC & HMIT)*, American Nuclear Society, 2004.

[5] G. H. Broadfoot and P. J. Broadfoot, "Academia and Industry Meet: Some Experiences of Formal Methods in Practice", *Proceedings of the Tenth Asia-Pacific Software Engineering Conference*, IEEE CS Press, 2003, pp.49-59.

[6] P. J. Hopcroft and G. H. Broadfoot, "Combining the Box Structure Development Method and CSP for Software Development", *Electronic Notes in Theoretical Computer Science*, 2005, 128(6), pp.127-144.

[7] T. Bauer, T. Beletski, F. Boehr, R. Eschbach, D. Landman and J. H. Poore, "From Requirements to Automated Testing of QUASAR Aussenspiegeleinstellung", *Fraunhofer IESE Technical Report 007.07E*, January 2007.

[8] H. D. Mills, "Stepwise Refinement and Verification in Box-Structured Systems", *IEEE Computer*, June 1988, 21(6), pp.23-36.

[9] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[10] R. Janicki and E. Sekerinski, "Foundations of the Trace Assertion Method of Module Interface Specification", *IEEE Transactions on Software Engineering*, July 2001, 27(7), pp.577-598.

[11] J. M. Spivey, *The Z Notation - A Reference Manual*, Prentice-Hall, 1989.

[12] C. L. Heitmeyer, J. Kirby, B. G. Labaw and R. Bharadwaj, "SCR*: A Toolset for Specifying and Analyzing Software Requirements", *Proceedings of the Tenth International Conference on Computer Aided Verification*, Springer-Verlag, 1998, pp.526-531.

[13] B. Korel and L. H. Tahat, "Understanding Modifications in State-Based Models", *Proceedings of the Twelfth IEEE International Workshop on Program Comprehension*, IEEE CS Press, 2004.

[14] A. Seawright and F. Brewer, "Clairvoyant: A Synthesis System for Production-Based Specification", *IEEE Transactions on VLSI Systems*, June 1994, 2(2), pp.172-185.

[15] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[16] A. W. Roscoe, "Model-Checking CSP", *A Classical Mind: Essays in Honor of C. A. R. Hoare*, Prentice Hall, 1994, pp.353-378.

[17] L. Lin, *Management of Requirements Changes in Sequence-Based Software Specifications*, PhD dissertation, University of Tennessee, 2006, http://sqrl.cs.utk.edu/btw/files/lin.pdf.

[18] J. H. Poore and C. J. Trammell, "Engineering Practices for Statistical Testing", *Crosstalk*, April 1998, 11(4), pp.24-28.