Master's Thesis

A Refined State Counting Algorithm for Model-Based Testing

Author:
Robert Sachtleben

Supervisor and first reviewer:
Prof. Dr. Jan Peleska

Second reviewer:
Prof. Dr. Carsten Lutz

July 19, 2018

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this work, I describe and rectify an error that has been observed in an algorithm designed to generate test cases for testing finite state machines. These test cases are supposed to be sufficient to prove that some finite state machines only behave in ways allowed by some other machine. Following from this property, the algorithm can be employed in model-based testing. Assuming that certain hypotheses hold, the generated tests are guaranteed to detect failures if a system does not conform to a given model of its desired behaviour.

1.1 Motivation

Finite state machines are a formalism suitable to describe many systems that have some internal state and that immediately respond to an external input with some output, possibly also modifying the internal state. Furthermore, using equivalence classes (see [**spillner2014software**]), methods of testing finite state machines can be applied to reactive system, as has been illustrated in [**Huang2016**].

Such results are of increasing importance and interest (see, for example, [**Petrenko1996**]) due to the widespread use of embedded systems, which can often be modeled as reactive systems or even as finite state machines. Many of these embedded systems are part of safety-critical systems, where a failure can cause great harm. When considering systems with non-deterministic behaviour, especially, it is thus often desirable to check whether a system conforms to a specification in the sense of being a reduction (i.e., whether every system behaviour is allowed in the specification).

Employing the concept of model-based testing, this determination can be achieved by automatically generating tests for models of both the system and the specification. This requires methods of producing such test cases that, when applied to system and specification, result in a failure being observed if (and only if) the system is not a reduction of the specification.

The algorithm described in this work and originally presented in [**hierons**] has been designed to generate test cases that satisfy this requirement. This algorithm differs from other methods in applying adaptivity to generate test cases depending on the observed behaviour of the system under test, resulting in a smaller number of test cases. An implementation of this algorithm has

been given by Romero Früh in his unpublished master's thesis (see [**aro**]) at the University of Bremen for the Research Group Operating Systems, Distributed Systems. In doing so, an error in the algorithm was observed: For some inputs, the generated test cases were insufficient to find an existing failure.

1.2 Contributions

This work provides a detailed explanation of the error observed when executing the algorithm presented in [**hierons**]. Following this explanation, I describe a minor modification to the algorithm that rectifies the observed error without a large increase in the algorithm's complexity. Here, I give the modified algorithm as both a formula and a program using simple grammar, which allows for program verification (following [**ProgramVerification**]).

In order to ensure that the modified algorithm generates a test suite sufficient to uncover all failures, I also perform a rigorous proof, beginning with the formulaic representation. For the program, I finally employ Hoare-Logic to prove its correctness.

1.3 Structure of this Document

In the first three chapters, I introduce the basic definitions. First, Chapter 2 introduces finite state machines and several properties and variations of such machines. Next, Chapter 3 explains the concept of model-based testing and provides two examples of methods used for generating tests, while Chapter 4 describes the techniques I employed in verifying programs. Chapter 5 introduces the algorithm presented in [**hierons**], the *adaptive state counting algorithm*, and explains the reasoning behind the way test cases are generated by it. Thereafter, Chapter 6 illustrates the observed error by giving a counterexample, followed by a short analysis of the causes of the insufficiency. Based on this analysis, Chapter 7 presents the modified algorithm, proves its correctness and provides a representation as a program that is also proven to be correct. Finally, Chapter 8 summarizes these results and describes possible future modifications.

Chapter 2

Finite State Machines

In this chapter, I introduce the concept of finite state machines and define several special properties of such machines required in later chapters. I then explain methods to reach and distinguish states in finite state machines and conclude by describing the concept of product machines.

Beginning with this chapter, I use the acronym FSM for finite state machine.

2.1 Basic Definitions

This section at first provides a definition of finite state machines and describes their graphical representation. Afterward, I introduce several functions that simplify the use of the *transition relation* of an FSM, and finally I define the language of a finite state machine.

The definition of finite state machines follows the one used in [**hierons**]:

Definition 2.1.1. A *finite state machine* (FSM) M is a tuple (S, s_1, X, Y, h) consisting of the following elements:

- a finite non-empty set of states S,
- an initial state $s \in S$,
- an input alphabet in the form of a finite set of symbols X,
- an output alphabet in the form of a finite set of symbols Y, and
- a transition relation h of type $S \times X \times S \times Y$.

Here $(s', x, s'', y) \in h$ describes that there exists a transition from state s of M to state s' of M for input $x \in X$, producing output $y \in Y$. ⊣

For example, let $M_E = (S, s_1, X, Y, h)$ be an FSM with the following com-

ponents:

$$\begin{aligned} S &= \{s_1, s_2, s_3, s_4\} \\ X &= \{a, b\} \\ Y &= \{0, 1\} \\ h &= \{(s_1, a, s_2, 0), (s_1, a, s_3, 1), (s_1, b, s_2, 0)\} \\ &\quad \cup \{(s_2, a, s_2, 1), (s_2, b, s_3, 1)\} \\ &\quad \cup \{(s_3, a, s_4, 0), (s_3, a, s_2, 1), (s_3, b, s_3, 0)\} \\ &\quad \cup \{(s_4, a, s_4, 0), (s_4, b, s_2, 1)\} \end{aligned}$$

Finite state machines can also be represented graphically by connecting the states with arrows such that an arrow from a state s' to a state s'' labeled with x/y represents the tuple (s', x, s'', y) in the transition relation. If multiple transitions exist from some state s' to some state s'', a single arrow may be used, which is then labeled for each such transition. Furthermore, the initial state is marked with an incoming arrow without any label. A graphical representation of M_E can then be given as illustrated in Figure 2.1.

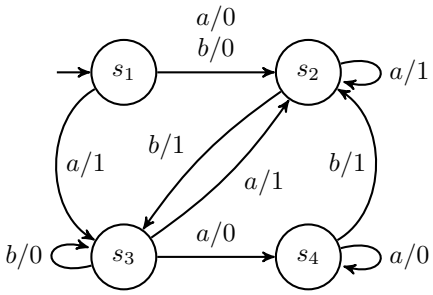


Figure 2.1: Graphical representation of example FSM M_E

The testing methods described in later chapters all rely on the repeated application of input symbols to an FSM and on observing the output symbols produced in accordance to the transition relation of the FSM. It is then often necessary to handle *sequences* of such symbols.

Definition 2.1.2. Let X be an input alphabet and Y an output alphabet. Then, a sequence $x_1, x_2, \ldots, x_k$ of elements $x_i \in X$ with $1 \leq i \leq k$ is called an *input sequence*. Next, a sequence of input/output pairs $(x_1, y_1), \ldots, (x_k, y_k)$, where for $1 \leq i \leq k$ it holds that $x_i \in X$ and $y_i \in Y$, is called an *input/output sequence* or an *IO sequence* for short. To avoid brackets, input/output pairs (x, y) are usually written as x/y.

The *length* of a sequence $\bar{x}$ denotes the number of its elements and is written $|\bar{x}|$. That is, the following holds:

$$|x_1, x_2, \ldots, x_k| = k$$

Finally, ϵ denotes the sequence of length 0, called *empty sequence*, for all of the above types of sequences. ⊣

In the following work, I write variables representing sequences and, in later chapters, trees with a bar over their names. Additionally, I do not distinguish between an element x_1 and the sequence $\bar{x} = x_1$ containing only the single element x_1. Furthermore, I adopt the following notation for IO sequences to allow easier referencing of their input and output *portions*: An IO sequence may be represented by a variable $\bar{x}/\bar{y}$ such that $\bar{x}$ refers to the *input portion* and $\bar{y}$ to the *output portion*. For example, in representing $x_1/y_1, x_2/y_2, \ldots, x_k/y_k$ by $\bar{x}/\bar{y}$ it holds that $\bar{x} = x_1, x_2, \ldots, x_k$ and $\bar{y} = y_1, y_2, \ldots, y_k$. Similarly, $x_1/y_1, x_2/y_2, \ldots, x_k/y_k$ is allowed to be written as $x_1, x_2, \ldots, x_k/y_1, y_2, \ldots, y_k$. It follows that for any IO sequence given in this form, the input and output portions have the same length.

The notation $\bar{x}\bar{x}'$ is used to represent the concatenation of sequences. For example let $\bar{x} = x_1, \ldots, x_k$ and $\bar{x}' = x'_1, \ldots, x'_n$. Then $\bar{x}\bar{x}' = x_1, \ldots, x_k, x'_1, \ldots, x'_n$. A sequence $\bar{x}\bar{x}'$ is called *extension* of $\bar{x}$ by $\bar{x}'$. Following from this notation and the convention that a single input is indistinguishable from the sequence containing only that input, it is then possible to write sequences without using commas to separate the inputs. Thus, for example, x_1, x_2, x_3 and $x_1x_2x_3$ are the same sequence.

For sets of sequences V and W, let $V.W$ denote the set produced by extending each sequence in V with each sequence in W:

$$V.W := \{\bar{x}\bar{x}' \mid \bar{x} \in V \wedge \bar{x}' \in W\}$$

For example,

$$\{aa, b\}.\{c, de\} = \{aac, aade, bc, bde\}$$

When concatenating IO sequence $\bar{x}_1/\bar{y}_1$ with $\bar{x}_2/\bar{y}_2$, the result is denoted as $\bar{x}_1\bar{x}_2/\bar{y}_1\bar{y}_2$. Also, in any sequence given in the form of $\bar{x}_1\bar{x}_2/\bar{y}_1\bar{y}_2$, it is assumed that the lengths of $\bar{x}_1$ and $\bar{y}_1$ (and hence also those of $\bar{x}_2$ and $\bar{y}_2$) are equal.

Finally, for some set X the notations X^* and X^i are used to denote the set of all finite sequences over X and all sequences over X of length $i \in \mathbb{N}$, respectively:

$$\begin{aligned} X^0 &:= \{\epsilon\} \\ X^{i+1} &:= X^i.X && \text{for } i \geq 1 \\ X^* &:= \bigcup_{i \in \mathbb{N}} X^i \end{aligned}$$

Here, for some input alphabet X and output alphabet Y, let X/Y denote the set of IO pairs over them:

$$X/Y := \{x/y \mid x \in X \wedge y \in Y\}$$

Thus, $(X/Y)^*$ denotes all IO sequences over X and Y.

Next, I introduce several functions based on the transition relation.

Definition 2.1.3. Let $M = (S, s_1, X, Y, h)$ be an FSM. Furthermore, let $x \in X$ be some element of the input alphabet, let $\bar{x} \in X^*$ be some nonempty input sequence, and let $\bar{y} \in Y^*$ be some nonempty output sequence. Then, consider the following functions:

- $h(s, x)$ generates all possible pairs of states reached and corresponding outputs produced by applying x to M in state s:

$$h(s, x) := \{(s', y) \mid (s, x, s', y) \in h\}$$

- $h(s, \bar{x})$ extends the previous function to input sequences: If $\bar{x}$ contains only a single symbol x, then $h(s, \bar{x}) = h(s, x)$. Otherwise, let $\bar{x} = \bar{x}'x$ such that

$$h(s, \bar{x}'x) := \{(s'', \bar{y}'y) \mid (s', \bar{y}') \in h(s, \bar{x}') \wedge (s'', y) \in h(s', x)\}$$

- $h^{\bar{y}}(s, \bar{x})$ produces all states reached by applying $\bar{x}$ to s such that the corresponding output is $\bar{y}$:

$$h^{\bar{y}}(s, \bar{x}) := \{s' \mid (s', \bar{y}) \in h(s, \bar{x})\}$$

- $h^{reach}(s, \bar{x})$ is the projection of $h(s, \bar{x})$ to the states that M might move to by applying $\bar{x}$ to s:

$$h^{reach}(s, \bar{x}) := \{s \mid \exists \bar{y} \in Y^* : h^{\bar{y}}(s, \bar{x}) \neq \emptyset\}$$

- $h^{out}(s, \bar{x})$ is the output projection of $h(s, \bar{x})$ such that

$$h^{out}(s, \bar{x}) := \{\bar{y} \mid h^{\bar{y}}(s, \bar{x}) \neq \emptyset\}$$

Finally, the special case of the empty sequence ϵ is handled as follows:

$$\begin{aligned} h(s, \epsilon) &= \{(s, \epsilon)\} \\ h^{\epsilon}(s, \epsilon) &= \{(s, \epsilon)\} \\ h^{reach}(s, \epsilon) &= \{s\} \\ h^{out}(s, \epsilon) &= \{\epsilon\} \end{aligned}$$

This represents the intuition that applying ϵ (i.e., no input) to any FSM does not perform any transition or produce any output. ⊣

For some state s, input sequence $\bar{x}$ and output sequence $\bar{y}$ such that $\bar{y} \in h^{out}(s, \bar{x})$, the state s is said to *react* or *respond* to $\bar{x}$ with $\bar{y}$ or $\bar{x}/\bar{y}$.

In M_E, the following results can be observed for some exemplary applications of the above functions:

$$\begin{aligned} h(s_1, a) &= \{(s_2, 0), (s_3, 1)\} \\ h(s_1, aab) &= \{(s_2, 101), (s_3, 011), (s_3, 111)\} \\ h^{10}(s_1, aa) &= \{s_4\} \\ h^{00}(s_1, aa) &= \emptyset \\ h^{out}(s1, aa) &= \{01, 10, 11\} \end{aligned}$$

The notion of applying input sequences to some state s can be extended to applying IO sequences to s, such that the latter describes applying the input portion of the sequence to s until the output portion equals the observed output.

Then, for some $s' \in h^{\bar{y}}(s, \bar{x})$, the state s' can be *reached* by applying $\bar{x}$ or $\bar{x}/\bar{y}$ to s. Here, if s is the initial state of the FSM, then s' it suffices to write that s' is reached by $\bar{x}$ (or $\bar{x}/\bar{y}$).

Furthermore, let *pref* be the function that calculates the set of all *prefixes* of a sequence. That is, for some alphabet Z and sequence $\bar{z} \in Z^*$, *pref* is defined such that

$$pref(\bar{z}) := \{\bar{z}' \in Z^* \mid \exists \bar{z}'' \in Z^* : \bar{z}'\bar{z}'' = \bar{z}\}$$

Notably, since $\epsilon \in Z^*$, the sequences ϵ and $\bar{z}$ are prefixes of $\bar{z}$. In some cases, only those prefixes are of interest that are not the original sequence. Here, a sequence $\bar{z}'$ is a *proper prefix* of $\bar{z}$ if $\bar{z}' \in pref(\bar{z}) \setminus \{\bar{z}\}$ holds.

An IO sequence $\bar{x}/\bar{y}$ is said to *visit* all states reached by some prefix of it. For example, a state s' is visited by the sequence x_1x_2/y_1y_2 applied to some state s, if any of the sequences ϵ, x_1/y_1, and x_1x_2/y_1y_2 applied to s reach s'. Also, a state s' is *repeated* along an IO sequence $\bar{x}/\bar{y}$ applied to some state s if there exist two distinct prefixes of $\bar{x}/\bar{y}$ that both reach s' if applied to s. In such cases, the sequence $\bar{x}/\bar{y}$ applied to s is also said to *repeat* s'.

Thus, for the example FSM M_E, the IO sequence $bbb/010$ reaches s_3 and visits, in order, s_1, s_2 and s_3, repeating s_3.

Finally, each state s of an FSM $M = (S, s_1, X, Y, h)$ defines a *language*, denoted by $L_M(s)$, that contains all IO sequences that can be produced by M by applying input sequences to s.

Definition 2.1.4. Let $M = (S, s_1, X, Y, h)$ be some FSM and let $s \in S$. Then, the language of s is defined as follows:

$$L_M(s) := \{\bar{x}/\bar{y} \mid \bar{x} \in X^* \wedge \bar{y} \in h^{out}(s, \bar{x})\}$$

The language of M itself, denoted by $L(M)$, is that of its initial state (i.e., $L(M) = L_M(s_1)$). ⊣

For some IO sequence $\bar{x}/\bar{y}$ contained in $L(M)$, that sequence is said to be *allowed* or *possible* in M.

In the following chapters, I generally consider two FSMs M_1 and M_2 over the same input and output alphabets with the intent to test what relationship M_2 has to M_1. Here, two relations are of interest: equivalence and reduction. Both relations are defined with respect to inputs and outputs only.

Definition 2.1.5. Let $M_1 = (S, s_1, X, Y, h_1)$ and $M_2 = (T, t_1, X, Y, h_2)$ be two FSMs. Then, two states $s \in S$ and $t \in T$ are *equivalent* if and only if their languages are identical, that is, if and only if $L_{M_1}(s) = L_{M_2}(t)$. Analogously M_2 is *equivalent* to M_1 if and only if $L(M_1) = L(M_2)$. The equivalence of two FSMs M_1 and M_2 is expressed by writing $M_1 \sim M_2$.

Similarly, $t \in T$ is a *reduction* of $s \in S$, written $t \preceq s$, if and only if $L_{M_2}(t) \subseteq L_{M_1}(s)$, that is, if and only if every sequence in the language of t is allowed in the language of s. Furthermore, M_2 is a *reduction* of M_1 if and only if $L(M_2) \subseteq L(M_1)$, and this relationship is denoted by $M_2 \preceq M_1$. ⊣

2.2 Special Properties of FSMs

This section covers some special properties that may hold for an FSM. Several of these are assumed to hold for the FSMs considered in later chapters.

First, only those FSM are of interest, where each contained state could be reached from the initial state via some input sequence, as unreachable states have no impact on the language of the FSM. Such FSMs are *initially connected* and an FSM $M = (S, s_1, X, Y, h)$ is initially connected if and only if the following holds:

$$\forall s \in S : \exists \bar{x}/\bar{y} \in (X/Y)^* : s \in h^{\bar{y}}(s_1, \bar{x})$$

This property does hold for M_E. If some FSM M is not initially connected, then an FSM M' can be created by removing from M all unreachable states. In this case $L(M) = L(M')$ trivially holds.

Second, an FSM $M = (S, s_1, X, Y, h)$ is *completely specified* if for each contained state there exists a transition for each symbol in the input alphabet. That is, M is completely specified if

$$\forall s \in S, x \in X : |h(s, x)| \geq 1$$

holds. This property also holds for M_E, as each state has transitions for both inputs a and b. In the following work, FSMs are considered to be completely specified. Any FSM that is not completely specified can be transformed into an FSM for which the above property holds, for example by introducing the missing transitions and directing them to some error state. Alternative procedures are given in [**hierons**].

Third, an FSM $M = (S, s_1, X, Y, h)$ is *deterministic*, if for each pair of state and input there is at most one transition in h that applies this input to that state. That is, M is deterministic if the following holds:

$$\forall s \in S, x \in X : |h(s, x)| \leq 1$$

Then, M_E is not deterministic, as for example s_1 can react to the input a in two different ways. In contrast to the previous properties, a non-deterministic FSM cannot be transformed into a deterministic FSM of the same language in a trivial way. For example, this transformation can be performed via *power-set construction* (see [**Hopcroft:2006:IAT:1196416**]), resulting in a possibly exponential increase in the number of states.

Fourth, an FSM $M = (S, s_1, X, Y, h)$ is *observable* if for each input/output pair and each state there exists at most one transition from this state with that input/output pair:

$$\forall s \in S, x \in X, y \in Y : |h^y(s, x)| \leq 1$$

That is, the state reached is uniquely determined by the input/output pair and the state to which the pair is applied. This property holds for M_E, as no state has two or more distinct transitions for the same input/output pair. As given in [**starke**], for each FSM, there exists an observable FSM of the same language, and hence all FSMs considered in the following work are assumed to be observable.

Note that for observable FSMs $M = (S, s_1, X, Y, h)$, it holds that for each state s and any $\bar{x}/\bar{y} \in L_M(s)$ there is exactly one state contained in $h^{\bar{y}}(s, \bar{x})$. Thus, in such cases, I often refer simply to the state $h^{\bar{y}}(s, \bar{x})$ when referencing the state in the singleton set.

Finally, an FSM $M = (S, s_1, X, Y, h)$ is *minimal* if no two states of it have the same language, that is, if the following holds:

$$\forall s, s' \in S : (s \neq s' \implies L_M(s) \neq L_M(s'))$$

Any FSM can be *minimised* (transformed into an equivalent minimal FSM) via the use of P_k tables as described in [**gill1962introduction**].

2.3 Reaching and Distinguishing States

Assuming all states of considered FSMs to be reachable, a further distinction can be made on whether a state can be reached in a deterministic way.

Definition 2.3.1. Let $M = (S, s_1, X, Y, h)$ be an FSM and $s \in S$ one of its states. Then s is *deterministically reachable* (d-reachable), if and only if there exists some input sequence $\bar{x} \in X^*$ such that $h^{reach}(s_1, \bar{x}) = \{s\}$. In this case, $\bar{x}$ *deterministically reaches* (d-reaches) s.

In particular, let s_1 be d-reached by the empty sequence ϵ, as no input is necessary to reach the initial state. ⊣

In the example FSM M_E, the state s_2 can be d-reached via the input b. However, the state s_4 is not d-reachable, because the only transition from another state to s_4 originates in s_3 and requires input a, to which s_3 can also react with a transition to s_2.

By definition, there might exist multiple responses to a sequence $\bar{x}$ d-reaching some state, as it is only required that each response to $\bar{x}$ reaches the exact same state. Regarding M_E, this is the case for d-reaching s_3 via the input sequence ab, where both $ab/01$ and $ab/10$ reach s_3.

In later chapters, there is be a need for sets of input sequences that d-reach all d-reachable states of some FSM. These are to constitute a basis to generate input sequences for *testing* (see the next chapter), as each such sequence serves to identify the state d-reached by it, providing valuable information.

Definition 2.3.2. Let $M = (S, s_1, X, Y, h)$ be some FSM, and let set $V \subseteq X^*$ be a set of input sequences such that $\epsilon \in V$. Then, V is a *deterministic state cover* of M if it contains ϵ and is a minimal set such that for every d-reachable state $s \in S$ there exists some input sequence $\bar{x} \in V$ that d-reaches s. ⊣

For example, the set $\{\epsilon, b, bb\}$ is a deterministic state cover of M_E, as it d-reaches s_1, s_2 and s_3, which together constitute all d-reachable states of M_E, and as no two distinct sequences in it reach the same state in M_E.

As each FSM by definition has only finitely many states, any deterministic state cover must be finite, since it contains at most one sequence for each state.

For testing purposes, there is also a need to *distinguish* states via input sequences that produce distinct responses. Here, for some $M = (S, s_1, X, Y, h)$, two states $s, s' \in S$ can be distinguished if there exists some input sequence $\bar{x} \in X^*$ such that s and s' do not react in exactly the same way to it, that is, if $h^{out}(s, \bar{x}) \neq h^{out}(s', \bar{x})$ holds. In this case, the sequence $\bar{x}$ *distinguishes* the states s and s'.

In M_E, each distinct pair of states is distinguished by at least one of the four input sequences of length 2, as illustrated in Table 2.1.

Table 2.1: Responses of M_E to various input sequences

State	Responses of M_E to					
	a	b	aa	ab	ba	bb
s_1	$0, 1$	0	$01, 10, 11$	$01, 10$	01	01
s_2	1	1	11	11	$10, 11$	10
s_3	$0, 1$	0	$00, 11$	$01, 11$	$00, 01$	00
s_4	0	1	00	01	11	11

In the example, states s_1 and s_2 can also be distinguished by a, as s_2 does not produce the response 0. However, both states share the output 1 for input a. Thus, by simply applying a once, both states might produce the same output. Furthermore, let M'_E be the FSM created from M_E by removing the transition $a/0$ from state s_1. Then, M'_E would be a reduction of M_E, but it would not be possible to distinguish s_1 and s_2 by a in M'_E, even though this input suffices for M_E. Later in this work, methods to distinguish states even in the above scenario are required. This leads to the concept of being able to *reliably distinguish* states.

Definition 2.3.3. Let $M = (S, s_1, X, Y, h)$ be some FSM with states $s, s' \in S$. Then, s and s' are *r(1)-distinguishable* if some input $x \in X$ exists such that $h^{out}(s, x) \cap h^{out}(s', x) = \emptyset$. Furthermore, s and s' are *r(k)-distinguishable* for some $k > 1$ if they are either already r(j)-distinguishable for some $j < k$, or if there exists some input $x \in X$ such that for all $y \in h^{out}(s, x) \cap h^{out}(s', x)$ the states $h^y(s, x)$ and $h^y(s', x)$ are r(k-1)-distinguishable.

States $s, s' \in S$ are then called *reliably distinguishable* (r-distinguishable) if there exists some $k \geq 1$ such that they are r(k)-distinguishable. ⊣

Revisiting the example FSM M_E and the responses described in Table 2.1, it can now be seen that the input sequence bb pairwise r-distinguishes s_1, s_2, s_3 and s_4. However, there exist cases where no single input sequence is sufficient to r-distinguish a pair of r-distinguishable states. In such cases, it is a possible strategy to choose different inputs depending on responses to previously applied inputs.

For some r-distinguishable states $s, s' \in S$ let $RD(s, s')$ denote a set of trees containing all trees T that satisfy the following. If T is a single node labelled with some $x \in X$, then $h^{out}(s, x) \cap h^{out}(s', x) = \emptyset$. Otherwise, the root of T is labelled with some $x \in X$ and for each $y \in h^{out}(s, x) \cap h^{out}(s', x)$ there exists an edge labelled y to the root of a tree $T' \in RD(h^y(s, x), h^y(s', x))$.

Each tree in $RD(s, s')$ then represents a set of inputs to r-distinguish s from s', chosen based on the outputs observed to previous inputs. For some $T \in RD(s, s')$, let $Seq(T)$ contain for each path from the root of T to a leaf the input sequences constructed from concatenating the labels of the nodes along the path. The set of input sequences $Seq(T)$ is then called an *r-distinguishing set* for s and s'. Another definition and construction of r-distinguishing sets can be found in [**hierons**] based on [**Petrenko1996**]. Furthermore, the above representation using trees is closely related to the use of adaptive test cases for the same purpose, which I describe in Section 5.3.

The notion of r-distinguishing sets then allows the definition of characterizing sets to be used in testing.

Definition 2.3.4. Let $M = (S, s_1, X, Y, h)$ be some FSM and $W \subseteq X^*$ a set of input sequences. Then, W *r-distinguishes* states $s, s' \in S$ if it contains a subset that is an r-distinguishing set for s and s'. Furthermore, W is a *characterizing set* of M if it r-distinguishes each pair of r-distinguishable states of M. ⊣

As shown in Table 2.1, each pair of states in M_E can be r-distinguished by the input sequence bb and hence $\{bb\}$ is a characterizing set for M_E.

2.4 Product Machines

When checking whether some FSM M_2 is the reduction of some FSM M_1, it can be useful to examine which sequences are allowed in both FSMs and which are only allowed in M_2. This can be expressed with the help of a *product machine*, expanded to include information about the sequences only allowed in M_2:

Definition 2.4.1. Let $M_1 = (S, s_1, X, Y, h_1)$ and $M_2 = (T, t_1, X, Y, h_2)$ be two observable and completely specified FSMs. Let $Fail$ be some state that is not contained in M_1 or M_2, and let $fail$ be some output not in Y. Then, the *product machine* of M_1 and M_2 is defined as

$$P(M_1, M_2) = ((S \times T) \cup \{Fail\}, (s_1, t_1), X, Y \cup \{fail\}, h_P)$$

where h_P is constructed such that the following holds:

$$\begin{aligned} h_P = &\{((s,t), x, (s',t'), y) \mid (s, x, s', y) \in h_1 \wedge (t, x, t', y) \in h_2\} && (1)\\ &\cup \{((s,t), x, Fail, y) \mid y \notin h_1^{out}(s, x) \wedge y \in h_2^{out}(t, x)\} && (2)\\ &\cup \{(Fail, x, Fail, fail) \mid x \in X\} && (3) \end{aligned}$$

For some state $(s, t) \in (S \times T)$, state $s \in S$ is called the *state-component* in M_1 and $t \in T$ the state-component in M_2. ⊣

The transition relation of $P(M_1, M_2)$ is thus created from the following three sets. The first set (1) contains transitions from states (s, t) for IO pairs x/y to states (s', t') such that x/y allows transition from s to s' and from t to t' in their respective FSMs. That is, via (1) all IO sequences allowed in both M_1 and M_2 are allowed in the product machine. Next, the set (2) describes transitions for states (s, t) and IO pairs x/y such that a corresponding transition exists for t in M_2, but not for s and hence not in M_1. All such transitions then lead to the state $Fail$, which, by (3), loops to itself for each input. Thus, (2) and (3) ensure that all IO sequences allowed only in M_2 lead to the fail state $Fail$.

Since M_1 and M_2 are observable, the resulting machine $P(M_1, M_2)$ is also observable by the definition of h_P. Analogously, the product machine is completely specified, as the additional state $Fail$ has a transition for every input.

This notion then allows for an alternative definition of reduction.

Lemma 2.4.2. *Let $M_1 = (S, s_1, X, Y, h_1)$ and $M_2 = (T, t_1, X, Y, h_2)$ be two observable and completely specified FSMs. Then, $M_2 \not\preceq M_1$ if and only if there exists an IO sequence $\bar{x}/\bar{y} \in L(M_2)$ that reaches $Fail$ in $P(M_1, M_2)$.*

Proof. First, consider the forward direction: There must exist some $\bar{x}'x/\bar{y}'y \in L(M_2) \setminus L(M_1)$. Without loss of generality, assume that no proper prefix of

$\bar{x}'x/\bar{y}'y$ is also contained in $L(M_2) \setminus L(M_1)$. Then, $\bar{x}'/\bar{y}' \in L(M_2) \cap L(M_1)$ holds. Hence, $\bar{x}'/\bar{y}'$ reaches some state (s,t) of the observable FSM $P(M_1, M_2)$ via the transitions in (1). Because $\bar{x}'x/\bar{y}'y$ is only allowed in M_2, it follows that $((s,t), x, Fail, y) \in h_P$, and thus $\bar{x}'x/\bar{y}'y$ reaches $Fail$.

Next, consider the reverse direction. Let $Fail$ be reached by some $\bar{x}/\bar{y} \in L(M_2)$, and let $\bar{x}'x/\bar{y}'y$ be the longest prefix of $\bar{x}/\bar{y}$ such that $\bar{y}'y$ does not contain $fail$. Thus, $\bar{x}'/\bar{y}' \in L(M_2) \cap L(M_1)$, since otherwise y would have to be $fail$. Then, due to its maximality, $\bar{x}'x/\bar{y}'y$ must reach $Fail$ and hence it holds that $\bar{x}'x/\bar{y}'y \in L(M_2) \setminus L(M_1)$. □

An example of a product machine is given in Figure 6.2 on page 55, which is based on FSMs M and M_I as defined in Figures 6.1a and 6.1b on page 54.

Chapter 3

Testing

Having defined FSMs and their properties in the previous chapter, I now describe how this concept can be used in the verification of real-world systems using *model-based testing*. I begin by introducing the basic definitions of this method and then show two methods of generating *test suites*: The W-Method introduced by [**Chow**] and the H-Method as described in [**dorofeeva2005improved**].

3.1 Model-Based Testing

The algorithms and methods presented in this work generate *test cases* based on a given FSM M. In turn, these test cases are then applied to some other FSM M_I to check whether the second FSM conforms to the first with respect to some *conformance relation*.

In the notion of *model-based testing* (MBT) applied in this work, M models the desired behaviour of a system and is called a *reference model*. Meanwhile, the *system under test* (i.e., the system to be tested against the reference model), abbreviated SUT, is assumed to behave like M_I. The validity of this assumption is ensured by the *testability hypothesis*, which states that the true behaviour of the SUT is equivalent to some FSM[1]. Of the two FSMs above, only the reference model is completely known. In contrast, the FSM representing the SUT is generally unknown, and its behaviour can only be examined by applying inputs and observing the produced outputs. In the following work, I sometimes refer to the reference model M as the *specification* FSM and to the FSM M_I representing the SUT as the SUT FSM.

More generally, model-based testing describes the use of models in testing. This can be achieved by modelling the tests, or, as used in this work, by generating tests from a model. Thus I have followed the notion of MBT as given in [**MBTReactive**]. Other variations of MBT can be found in [**MBTEssentials**].

[1] More specifically, the true behaviour of the SUT is to be equivalent to some element in the *fault domain*, see Def. 3.1.2

3.1.1 Test Cases and Test Suites

In its most abstract definition, a set of *test cases* $TC(Sig)$ for some signature[2] Sig is simply a set of objects such that there also exists a total relation $\underline{\text{pass}} \subseteq Sig \times TC(Sig)$ where for some model $M \in Sig$ and test case $t \in TC$ the expression $((M, tc) \in \underline{\text{pass}})$ denotes that M *passes* t. Elements in this relation are generally given in infix notation:

$$(M, tc) \in \underline{\text{pass}} \equiv M \ \underline{\text{pass}} \ tc$$

Test cases can then be collected in *test suites*, where a test suite TS for some signature Sig is a set of test cases such that a model $M \in Sig$ passes TS if and only if M passes each test case in TS.

In this work, I only compare FSMs that use the same alphabets. Observe here, that the output alphabet of an FSM can be arbitrarily extended without changing the language of the FSM, while the analogous claim for the input alphabet does not hold if the FSM is required to be completely specified.

Regarding FSMs, there are two particular pass-relations of interest: equivalence and reduction. Those are denoted by $\underline{\text{pass}}_{\sim}$ and $\underline{\text{pass}}_{\preceq}$, respectively.

For some sets X and Y then let $FSM(X, Y)$ denote the set of all observable and completely specified FSMs with input alphabet X and output alphabet Y. Then, test cases for FSMs can be defined as follows.

Definition 3.1.1. Given sets X and Y, an *FSM test case* is a pair $U = (U_p, U_f)$ of sets of IO sequences. Then, an FSM $M \in FSM(X, Y)$ passes U for equivalence if and only if all sequences in U_p are in the language of M, while no sequence in U_f is allowed in M. This can be expressed as follows:

$$M \ \underline{\text{pass}}_{\sim} \ U \Leftrightarrow U_p \subseteq L(M) \wedge U_f \cap L(M) = \emptyset$$

Furthermore, M passes U for reduction by simply not allowing any sequence contained in U_f:

$$M \ \underline{\text{pass}}_{\preceq} \ U \Leftrightarrow U_f \cap L(M) = \emptyset$$

Here, the sequences in U_p are called *pass sequences* and those in U_f are *fail sequences*.

Finally, let $TC(X, Y)$ denote the set of all test cases over alphabets X and Y. ⊣

When testing FSMs, I often simply use sets of input sequences to denote test suites: For some FSM $M = (S, s_1, X, Y, h)$ and some set of input sequences $T \subseteq X^*$, let the test suite $EQ(M, T)$ for equivalence and the test suite $RED(M, T)$ for reduction be defined as follows:

$$\begin{aligned} EQ(M,T) \quad &:= \Big\{ \big(\{\bar{x}/\bar{y} \mid \bar{x} \in T \wedge \bar{x}/\bar{y} \in L(M)\}, \\ &\qquad \{\bar{x}/\bar{y} \mid \bar{x} \in T \wedge \bar{y} \in Y^* \wedge \bar{x}/\bar{y} \notin L(M)\} \big) \Big\} \\ RED(M,T) &:= \Big\{ \big(\emptyset, \{\bar{x}/\bar{y} \mid \bar{x} \in T \wedge \bar{y} \in Y^* \wedge \bar{x}/\bar{y} \notin L(M)\} \big) \Big\} \end{aligned}$$

[2] A signature fixes the symbol sets of the structures considered. For the background of this notion in model theory see [**hodges1997shorter**] and [**diaconescu2008institution**].

That is, an FSM M' passes $EQ(M,T)$ if it reacts to each sequence in T in the exact same way as M, while it passes $RED(M,T)$ if it never reacts in a way not allowed in M.

If the desired conformance relation and reference model are obvious in a given context, I directly refer to the set of input sequences as a test suite.

3.1.2 Complete Testing Theories

Some of the methods presented in this and later chapters have the special property of generating a test suite TS for some reference model M such that for a certain set of FSMs F, each FSM $M' \in F$ passes TS if and only if M' conforms to M.

Such sets F can also be given as part of *fault models*.

Definition 3.1.2. Let Sig be some signature. A *fault model* $\mathcal{F}(M, \leq, Dom)$ then specifies a reference model $M \in Sig$, a conformance relation $\leq$ and some *fault domain* $Dom \subseteq Sig$. ⊣

In testing FSMs, fault domains are often determined by the FSM representing the reference model and some upper bound on the number of states. For example, for some observable and completely specified FSM $M = (S, s_1, X, Y, h)$, let FSM_M^m denote the set of all FSMs that use input and output alphabets X and Y, respectively, and that have at most m states for some $m \in \mathbb{N}$ with $m \geq |S|$. As a subset of special importance, let $DFSM_M^m \subset FSM_M^m$ denote the set of all deterministic and completely specified FSMs with at most m states, using the same alphabets as M.

Fault models then allow for the definition of the test suites with the above property.

Definition 3.1.3. Let TS be a test suite over signature Sig. TS is *complete* with respect to some fault model $F(M, \leq, Dom)$ if and only if both of the following properties hold:

- The test suite is *sound*: If an element of the fault domain conforms to the reference model, then it passes the test suite:

$$\forall M' \in Dom : M' \leq M \implies (\forall U \in TS : M' \underline{\text{pass}}\, U)$$

- The test suite is *exhaustive*: If an element of the fault domain passes the test suite, then it conforms to the reference model:

$$\forall M' \in Dom : (\forall U \in TS : M' \underline{\text{pass}}\, U) \implies M' \leq M$$

⊣

Finally, for some set of fault models there exists a *complete testing theory*, if for each contained fault model a test suite can be created that is complete for that model. That is, a mapping $TT : F \to \mathbb{P}(TC(Sig))$ for some set of fault models F is a *complete testing theory* if and only if for each $\mathcal{F} \in F$, the test suite $TT(\mathcal{F})$ is complete for $\mathcal{F}$.

3.1.3 Complete Testing Assumption

As described above, the FSM representing the SUT can only be explored by applying inputs and observing the corresponding outputs. While trivial for deterministic FSMs, this approach is problematic when testing non-deterministic machines since without further assumptions, it is impossible to determine whether all possible responses to some input have been observed. Here, I make the *complete testing assumption*, also named *fairness assumption*, that there exists some $k \in \mathbb{N}$ such that if an input sequence is applied k times to some FSM, then all responses of that FSM to the input are observed.

3.2 W-Method

The W-Method has been among the first complete testing theories developed for testing equivalence between deterministic and completely specified FSMs and was introduced in [**Chow**].

This method generates tests for some minimal and completely specified FSM $M = (S, s_1, X, Y, h)$. Let $|S| = n$, and furthermore, let V and W be some deterministic state cover and characterizing set of M, respectively. Then, the following set $\bar{W}$ of input sequences constitutes the test suite generated by the W-Method for the fault model $F(M, \sim, DFSM_M^m)$:

$$\overline{W} := V.\Big(\bigcup_{i=0}^{m-n+1} X^i\Big).W$$

The mapping of $F(M, \sim, DFSM_M^m)$ to the test suite for equivalence generated from $\overline{W}$ is a complete testing theory as proven in [**Chow**] and [**vasilevskii1973failure**].

3.3 H-Method

The H-Method is a slightly more complex method to generate test suites for the same fault models as the previous method. The following introduction is based on [**dorofeeva2005improved**], but to shorten the description I additionally assume all considered FSMs to be completely specified.

Let $M = (S, s_1, X, Y, h)$ be some deterministic, minimal and completely specified FSM with some deterministic state cover V. Then, a set $TS(M, m)$ of input sequences is the result of the H-method for M and some $m \in \mathbb{N}$ if $V.X^{m-n+1} \subseteq TS(M, m)$ and furthermore for the sets

$$\begin{aligned}
A &:= V \times V \\
B &:= V \times (V.(\textstyle\bigcup_{i=1}^{m-n+1} X^i)) \\
C &:= \{(\bar{v}\bar{x}, \bar{v}\bar{x}') \mid \bar{x}' \in (\textstyle\bigcup_{i=1}^{m-n+1} X^i) \wedge \bar{v} \in V \wedge \bar{x} \in pref(\bar{x}')\}
\end{aligned}$$

it holds that for each $(\bar{x}, \bar{x}') \in A \cup B \cup C$, if $\bar{x}$ and $\bar{x}'$ reach different states in M, then there exist sequences $\bar{x}\bar{w}$ and $\bar{x}'\bar{w}$ in $TS(M, m)$ such that $\bar{w}$ distinguishes the state reached by $\bar{x}$ from that reached by $\bar{x}'$.

Such a set $TS(M, m)$ can easily be produced by an algorithm that initializes its result with the set $V.X^{m-n+1}$ and then iterates through all pairs $(\bar{x}, \bar{x}') \in$

$A \cup B \cup C$, adding distinguishing extensions to its result only if for all $\bar{w}$ that distinguish the states reached by the pair the sequences $\bar{x}\bar{w}$ and $\bar{x}'\bar{w}$ are not yet contained in the result.

Then, the fault model for M and m is given as $\mathcal{F} = F(M, \sim, DFSM_M^m)$ and the above construction constitutes a mapping of $\mathcal{F}$ to the test suite represented by $TS(M, m)$. Finally, as proven in [**dorofeeva2005improved**], this mapping is a complete testing theory.

The H-Method thus differs from the W-Method mainly because it does not require a characterizing set of M to be applied after sequences. Instead, distinguishing sequences are only added and applied if necessary. Thus, the number of input sequences in a test suite generated for some FSM M via the H-Method can be significantly smaller than that in the test suite generated for the same M via the W-Method. This relationship has been observed in practice in a survey conducted by Dorofeeva et al in [**DOROFEEVA20101286**].

Chapter 4

Program Verification

As stated in the introductory chapter, a primary goal of this work is to prove the correctness of a modified version of the adaptive state counting algorithm. This requires a formalism with which to express the algorithm and to verify the correct behaviour of it.

For this task, I have chosen the language for programs given in Chapter 3 of [**ProgramVerification**], which is sufficiently expressive to allow for a concise representation of the algorithm. At the same time, this language contains only a small number of different statements, which allows for a short description of its syntax and semantics. Finally, this choice also enabled me to employ Hoare-Logic (see [**hoare1969axiomatic**]) to verify the behaviour of programs written in this language by the use of *pre-* and *postconditions*.

4.1 Programs

The algorithms presented in the following chapters are *deterministic programs* given as a string of symbols generated from the following grammar with the special keywords **skip**, **if**, **then**, **else**, **end**, **while** and **do**:

$$
\begin{aligned}
S ::= \ & \textbf{skip} && \text{empty statement} \\
| \ & v \leftarrow e && \text{assignment} \\
| \ & S \ \textbf{;} \ S && \text{sequential composition} \\
| \ & \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ end if} && \text{conditional statement} \\
| \ & \textbf{while } B \textbf{ do } S \textbf{ end while} && \text{loop}
\end{aligned}
$$

where v is a *variable*, e is some expression and B is a boolean expression.

For these five kinds of *statements*, the following intuitive semantics are used:

- The empty statement **skip** performs no actions and just terminates.
- The assignment $v \leftarrow e$ assigns the value of expression e to variable v.
- The sequential composition S_1 ; S_2 executes S_1 first and then executes S_2. This behaviour is associative, and hence, I omit placing enclosing brackets here.

- The conditional statement **if** B **then** S_1 **else** S_2 **end if** first evaluates the expression B. If the result is *true*, then S_1 is executed, otherwise S_2 is executed.
- The loop **while** B **do** S **od** first evaluates B. If the result is *false*, then the execution terminates. Otherwise, S is executed followed by the execution **while** B **do** S **od**, that is, followed by the next *iteration* of the loop.

To describe the goal of *total correctness* as given in the next section, a more rigorous definition of these semantics is required. This can be accomplished using stepwise execution of a program based on *configurations*. First, a *state* for some program S is a valuation function over the variables of S. For an expression e over variables in S, let $\sigma(e)$ denote its value with respect to state σ. Furthermore, let $\sigma[e/v]$ denote the state that assigns to v the value e and that valuates every variable other than v in the same way as σ. A state can also valuate variables not contained in the program, which is used to store external values, as illustrated by the Loop axiom in the following section. Next, a configuration $\langle S, \sigma\rangle$ consists of a state σ and a program S for which to perform the next step. Following the intuitive semantics, this allows for the definition of *transitions* between configurations:

$$\langle S_1, \sigma_1\rangle \to \langle S_2, \sigma_2\rangle$$

which denotes that executing the next step in S_1 based on σ_1 results in a new state σ_2, with S_2 being the remainder of S_1 still to be executed. *Termination* occurs if there are no steps left to execute. This is expressed by introducing the *empty program* E, such that a transition terminates if S_2 in the above formula is E. To introduce E into programs, let each program S be extended by some sequential composition with E, that is, let S **;** E and E **;** S be abbreviations for S. The following transition axioms serve as a definition of the semantics of deterministic programs:

1. $\langle \textbf{skip}, \sigma\rangle \to \langle E, \sigma\rangle$
2. $\langle u \leftarrow t, \sigma\rangle \to \langle E, \sigma[\sigma(t)/u]\rangle$
3. $\dfrac{\langle S_1, \sigma\rangle \to \langle S_2, \tau\rangle}{\langle S_1 \textbf{ ; } S, \sigma\rangle \to \langle S_2 \textbf{ ; } S, \tau\rangle}$
4. $\langle$**if** B **then** S_1 **else** S_2 **end if**$, \sigma\rangle \to \langle S_1, \sigma\rangle$ if $\sigma(B)$ is *true*
5. $\langle$**if** B **then** S_1 **else** S_2 **end if**$, \sigma\rangle \to \langle S_2, \sigma\rangle$ if $\sigma(B)$ is *false*
6. $\langle$**while** B **do** S **end while**$, \sigma\rangle \to \langle S$ **; while** B **do** S **end while**$, \sigma\rangle$ if $\sigma(B)$ is *true*
7. $\langle$**while** B **do** S **end while**$, \sigma\rangle \to \langle E, \sigma\rangle$ if $\sigma(B)$ is *false*

These transitions then finally allow for the definition of *transition sequences* and (terminating) *computations*: A transition sequence starting in σ for a program S is a sequence of configurations $\langle S_i, \sigma_i\rangle$ with $i \geq 0$ beginning at $\langle S, \sigma\rangle = \langle S_0, \sigma_0\rangle$ such that the following holds for all $\langle S_i, \sigma_i\rangle$ with $i > 0$ in the sequence:

$$\langle S_{i-1}, \sigma_{i-1}\rangle \to \langle S_i, \sigma_i\rangle$$

Then, a computation of S starting in σ is a transition sequence that cannot be extended. A computation is *terminating in* τ if it is finite and has a last element $\langle E, \tau \rangle$.

I use the additional statement v **:** T to *declare* a variable v of type T before the variable is used in any way. This statement does not perform any actions and is added only to increase readability. For multiple variables $v_1, \ldots, v_k$ of the same type T, declaration in a single line as $v_1, \ldots, v_n$ **:** T is also allowed. Furthermore, parts of a program can be annotated with *comments*. These begin with a small triangle ▷, and everything written in the same line following such a triangle is not part of the program (see for example lines 3 and 4 in Algorithm 4.1).

Let S be some program. Then $var(S)$ denotes the set of all variables in S and $change(S)$ denotes the set of all variables that appear on the left side of an assignment, that is, in the place of v in $v \leftarrow e$. Thus, $change(S)$ contains all variables that can be modified in the program. In all programs given in this work, each variable v is assigned some value before any expression containing v is evaluated.

As the algorithms presented in this work produce test suites based on some given FSM and some further *inputs*, they are written as *functions* that require (typed) *inputs* and *return* (typed) *outputs*. I generate functions from the following grammar:

$$
\begin{aligned}
F &::= \textbf{function } N \textbf{ (} \mathit{Inputs} \textbf{) : } T\ S \textbf{ return } e \textbf{ end function} \\
\mathit{Inputs} &::= \mathit{Input} \mid \mathit{Input} \textbf{ , } \mathit{Inputs} \\
\mathit{Input} &::= \textbf{in } v \textbf{ : } T
\end{aligned}
$$

where N is some string representing the *name* of the function, T is some *type*, S is a program generated by the previous grammar, and v and e are a variable and an expression, respectively.

For example, a function fac to calculate the factorial of some input x in an iterative way can be represented as shown in Algorithm 4.1.

Algorithm 4.1 Factorial

```
1:  function fac(in x : ℕ) : ℕ
2:      t, r : ℕ ;
3:      t ← 1 ;                          ▷ Iteration counter
4:      r ← 1 ;                          ▷ Value of fac(t)
5:      while t < x do
6:          t ← t + 1 ;
7:          r ← r * t
8:      end while
9:      return r
10: end function
```

4.2 Total Correctness

In this section, I describe the proof system for *total correctness* of deterministic programs as given in [**ProgramVerification**], which originates in [**hoare1969axiomatic**].

Following these works, program correctness is expressed using *correctness formulas*, that is, statements

$$\{p\}\ S\ \{q\}$$

over a program S and *assertions* p and q. These assertions p and q are called *precondition* and *postcondition*, respectively. A statement $\{p\}\ S\ \{q\}$ is then to be read as follows: *If p holds immediately before executing S, then q holds directly after having executed S.* More formally, the precondition describes a set of *initial states* (inputs) from which to start the program, while the postcondition contains all *final states* (outputs) that the program is desired to terminate in for those inputs. The conditions are expressions over variables and I say that a state is 'in' p (or q, respectively) if the expression holds with respect to the valuation given by the state. Thus, the formula *true* is satisfied by every state. Finally, the statement $\{p\}\ S\ \{q\}$ is *true* with respect to total correctness if every computation of S from an initial state in p terminates in some state in q.

The proof system presented in [**ProgramVerification**] then consists of the following six axioms, where $p[x/y]$ denotes the expression resulting from replacing each occurrence of y in p by x, such that, for example, $(a+b)[2/a] \equiv (2+b)$.

Axiom 1: Skip

$$\{p\}\ \textbf{skip}\ \{p\}$$

That is, **skip** does not modify the state of the program, and hence, input and output states are identical.

Axiom 2: Assignment

$$\{p[t/u]\}\ u \leftarrow t\ \{p\}$$

Assignments can be propagated 'backwards' in the conditions.

Axiom 3: Composition

$$\frac{\{p\}\ S_1\ \{r\}, \{r\}\ S_2\ \{q\}}{\{p\}\ S_1\ ;\ S_2\ \{q\}}$$

Conditions can be composed for identical intermediate state sets.

Axiom 4: Conditional

$$\frac{\{p \wedge B\}\ S_1\ \{q\}, \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{end if}\ \{q\}}$$

If a postcondition holds for a precondition irrelevant of the branch taken in a conditional statement, then it holds for the conditional statement itself.

Axiom 5: Consequence

$$\frac{p \implies p_1, \{p_1\}\ S\ \{q_1\}, q_1 \implies q}{\{p\}\ S\ \{q\}}$$

It is possible to strengthen preconditions and to weaken postconditions.

Axiom 6: Loop

$$\frac{\begin{array}{c}\{p \wedge B\}\ S\ \{p\} \\ \{p \wedge B \wedge t = z\}\ S\ \{t < z\} \\ p \implies t \geq 0\end{array}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{end while}\ \{p \wedge \neg B\}}$$

Here t is an integer expression, and z is a variable not occurring in p, B, S or t. The first premise requires that if an iteration is performed as B is *true*, then p holds before and after the iteration is performed. This p is called *loop invariant*. In the second premise, the variable z, which can never be modified in the program, stores some value t, which is required to have decreased after executing an iteration. Together with the third premise, which requires this t to be non-negative, this ensures that only a finite number of iterations can be performed. The expression t is hence called the *bound function*. If these three premises hold, then the loop does not allow for infinite computations (and hence terminates in a state satisfying $\neg B$), while still upholding the invariant p.

Chapter 5

Adaptive State Counting

In this chapter, I introduce the adaptive state counting algorithm as described in [**hierons**].

Given some FSMs M, representing the specification, and M_I, representing the behaviour of the SUT, the goal of this algorithm, as detailed in the following sections, is to generate a test suite such that M_I passes the test suite if and only if it is a reduction of M.

To provide a continuous illustration for the subsequent definitions, I begin by describing two such FSMs used as a running example. Then, I characterise the general idea of the adaptive state counting algorithm, beginning with an informal description of the non-adaptive variation this algorithm is based upon. Subsequently, I introduce *adaptive test cases*, which are used as a more adaptive way to distinguish states. Next, I define the two methods used to count states in the algorithm and explain how these are applied. These methods complete the required concepts to describe the test suite generated when applying the adaptive state counting algorithm. I conclude this chapter by giving a representation of the algorithm as a program using the language introduced in Chapter 4.

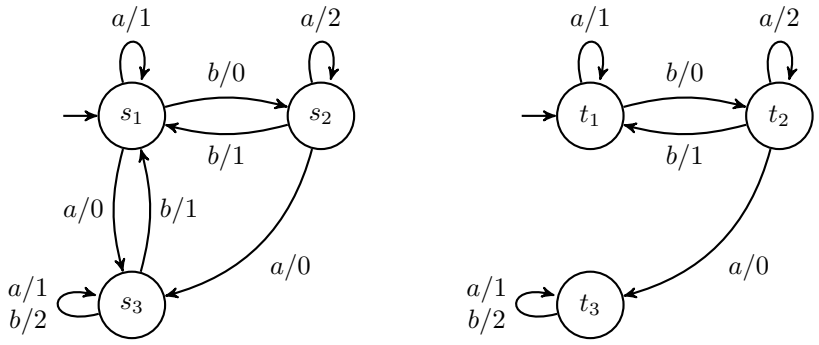
The algorithm as presented here and in [**hierons**] is incomplete. That is, the test suite it generates is not generally sufficient to check for reduction. This error is discussed in more detail in Chapter 6.

5.1 Accompanying Example

The following machines are used as concrete examples to accompany the description of the algorithm and thus to illustrate how the test suite is generated: For input alphabet $X^A = \{a, b\}$ and output alphabet $Y^A = \{0, 1, 2\}$, let $M^A = (S, s_1, X^A, Y^A, h)$ and $M_I^A = (T, t_1, X^A, Y^A, h_I)$ be two FSMs as described in Figure 5.1 on page 28, with M^A and M_I^A representing a specification and the behaviour of an SUT, respectively.

Both of these FSMs are completely specified and observable. Furthermore, M_I^A is essentially created by removing from M^A the transitions between s_1 and s_3, and hence, M_I^A trivially is a reduction of M^A.

Next, state s_3 of M^A is not deterministically reachable, while s_2 is deterministically reached by input b, as there exists only the single response 0 of M^A



(a) Specification FSM M^A (b) SUT FSM M_I^A

Figure 5.1: FSMs used as accompanying example

to it. Thus, the set

$$V^A = \{\epsilon, b\}$$

is a deterministic state cover of M^A.

Finally, state s_1 of M^A can be r-distinguished from both s_2 and s_3 via the single input b, and the latter two states can be r-distinguished by the single input a. Therefore, the set

$$W = \{a, b\}$$

is a characterizing set of M^A.

5.2 Idea

The adaptive state counting algorithm is based on the state counting algorithm described in [**hierons**] and [**Petrenko1996**].

Both algorithms generate test suites to check whether some FSM M_I is a reduction of some FSM M. In doing so, they both perform a breadth-first search, starting from the deterministic state cover of M, and trying to find a sequence of minimal length such that M_I reacts to this sequence in a way not allowed in M. Later, I call such a sequence a *minimal sequence to a failure*. The termination criterion for this search is based on *state counting* in the sense that, for some IO sequence $\bar{x}/\bar{y}$, the minimum number of states is calculated that M_I must contain for $\bar{x}/\bar{y}$ to never repeat a state in the product machine $P(M, M_I)$. If the count exceeds some previously fixed upper limit $m \in \mathbb{N}$ on the number of states of M_I, then a repetition must occur, and any sequence that contains such a repetition cannot be a prefix of a minimal sequence to a failure. Thus, there is no need to consider any extension of an input sequence $\bar{x}$, if the above holds for all IO sequence observed for $\bar{x}$ in M (or M_I in case of the adaptive algorithm). The algorithms therefore iteratively extend sequences and apply them after the sequences in the deterministic state cover, extending each sequence as long as it could possibly be a prefix of a minimal sequence to a failure.

For M^A and $m = 3$, the generation of the test suite using the state counting algorithm is performed roughly as follows. First, as the states of M^A are pairwise

r-distinguishable using W, and as ϵ and b reach different states in M^A, any observable and completely specified M_I that responds to

$$V.W = \{\epsilon, b\}.\{a, b\}$$

only in ways allowed in M^A must have at least two distinct states[1]. Thus, when next applying nonempty sequences after sequences in V, the lower bound on the number of states some M_I must have to avoid a repetition along $\bar{x}$ is always at least two. As $2 \leq m$, the search can also not yet be terminated.

The algorithm then begins extending sequences from each element in V. For example, consider first $b \in V$. For any sequence of length 1 (i.e., for any $\bar{x} \in X^A$) applied after b, the lower bound is 3, as additionally to the two states reached by V, some further state must be reached by its application (M_I is assumed to be completely specified), and as again the states in M^A are r-distinguishable.

This again is not a lower bound that exceeds $m = 3$, and hence, the algorithm subsequently considers the sequences of length 2. For every $\bar{x} \in X^A.X^A$ applied after $b \in V$, an FSM M_I that does react to $b\bar{x}$ only in ways allowed in M^A must have at least four different states to avoid a repetition along $\bar{x}$.

For example, consider $b\bar{x} = bbb$. In any M_I of the previous kind, the sequence ϵ reaches t_1, b reaches some $t_2 \neq t_1$, and bb and bbb reach some t_3 and t_4, respectively. Using the r-distinguishability in M^A as alluded to in footnote [1], $t_3 \neq t_4$, while also $t_3 \notin \{t_2, t_4\}$ and $t_4 \notin \{t_1, t_3\}$. However, it is still possible that $t_3 = t_1$ or $t_4 = t_2$. Both of these possibilities are to be prevented, as they constitute the kind of repetition to avoid: for example, if $t_4 = t_2$, then if M_I responds to some $bbb\bar{x}_1$ in a way not allowed in M^A, the same already holds for $b\bar{x}_1$, and hence, bbb cannot be extended to a sequence to a failure of the desired minimality.

Now the lower bounds for these sequences all exceed $m = 3$, and hence, as it is impossible that any M_I has more than m states, none of the sequences $\bar{x} \in X^A.X^A$ applied after $b \in V$ is free of repetition, which in turn shows that they cannot be extended to form a sequence to a failure of the desired kind. The breadth-first search for sequences to apply after $b \in V$ thus terminates.

Analogous results hold for the sequences to be applied after the remaining sequence in V, which is ϵ. Therefore, during the search the following sequences were considered:

$$B = V \cup V.X^A \cup V.X^A.X^A$$

As the characterizing set W is applied after any such sequence to make use of the r-distinguishability of states in M^A, the following sequences constitute the test suite TS:

$$TS = B.W = \{\epsilon, a, b, aa, ab, ba, bb, baa, bab, bba, bbb\}.\{a, b\}$$

Since the states in M^A are pairwise r-distinguishable, the non-adaptive state counting algorithm stops extending any sequence $\bar{x}$ with $|V| + |\bar{x}| > m$. For FSMs that do not have this property, the lower bound of a sequence needs to be calculated for the different sets of r-distinguishable states in M^A.

The *adaptive* state counting algorithm differs from the state counting algorithm by applying adaptivity in several ways. Here, adaptivity means that

[1]The connection between r-distinguishability in M^A and distinguishability in some M_I used here is described in the subsequent section.

the algorithm behaves differently depending on M_I, for example by applying different inputs depending on the observed outputs of M_I to previous inputs. This allows for a reduction of the number of input sequences to be applied to M_I, but at the same time, the generated test suite is usable only to check whether the specific FSM M_I is a reduction of M. As the algorithm presented in [**hierons**] produces incomplete test suites for some inputs, I defer the discussion concerning the relation of completeness and adaptivity to Section 7.6.

In Subsection 5.5.1, I illustrate the test suite for M^A and M_I^A generated using the adaptive state counting algorithm.

5.3 Adaptive Test Cases

The first application of adaptivity in the presented algorithm lies in the use of *adaptive test cases*, trees with directed edges similar to those used to describe r-distinguishability in Section 2.3. These test cases are used to apply test inputs depending on observed outputs and thus may reduce the number of input sequences applied to an FSM if specific outputs are not observed. They are defined as follows.

Definition 5.3.1. Let $\Upsilon(X, Y)$ denote the set of all *adaptive test cases* over input alphabet X and output alphabet Y. An adaptive test case $\bar{\sigma}$ in $\Upsilon(X, Y)$ is one of the following:

- *null*
- a pair (x, f) consisting of an input $x \in X$ to be applied and a (possibly partial) function $f : Y \to \Upsilon(X, Y)$ to handle the responses to x.

In the latter case, each $f(y)$ for $y \in Dom(f)$ is called a *subtest* of $\bar{\sigma}$. This relation extends recursively, that is, all subtests of some $f(y)$ are also subtests of $\bar{\sigma}$. ⊣

Adaptive test cases can be represented graphically by drawing *null* nodes as filled circles and drawing some (x, f) as a circle labelled x such that for each $y \in Dom(f)$ there exists a directed edge labelled y from that circle to the representation of test case $f(y)$.

For example, consider the adaptive test case $\bar{\sigma}_E \in \Upsilon(\{a, b\}/\{0, 1\})$ such that $\bar{\sigma}_E = (a, f_0)$ for functions defined as follows:

$$\begin{aligned} f_0(0) &:= (b, f_1) & f_0(1) &:= (a, f_2) \\ f_1(0) &:= \mathit{null} & f_1(1) &:= (b, f_3) \\ f_2(0) &:= \mathit{null} & f_2(1) &:= \mathit{null} \\ f_3(0) &:= \mathit{null} & f_3(1) &:= \mathit{null} \end{aligned}$$

Then, Figure 5.2 is a graphical representation of $\bar{\sigma}_E$:

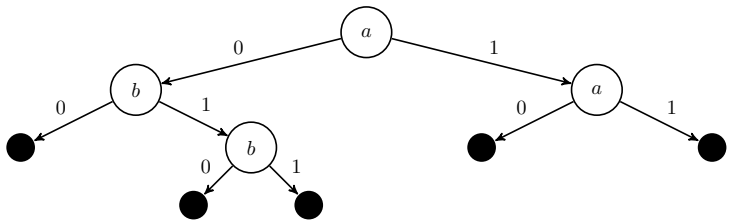


Figure 5.2: Graphical representation of $\bar{\sigma}_E$

The *height* of an adaptive test case can then be defined analogously to the height of a tree. The height of an adaptive test case $\bar{\sigma}$ is denoted as $height(\bar{\sigma})$ such that

$$height(\bar{\sigma}) := \begin{cases} 0 & \text{if } \sigma = null \\ 1 + \max\{height(f(y)) \mid y \in Dom(f)\} & \text{if } \bar{\sigma} = (x, f) \end{cases}$$

An adaptive test case $\bar{\sigma} = (x, f)$ is then *applied* to some FSM M by first applying to M its input x and then observing the produced output y. If there is no entry for y in f (i.e., if $y \notin Dom(f)$), then the application ends. Otherwise, the test case $f(y)$ is applied. Application also ends if a test case *null* is encountered. Any adaptive test case considered in this work is assumed to be representable by a finite tree (i.e., it has a finite height and finite alphabets), and hence, that its application to any FSM terminates.

Furthermore, for some output alphabet Y, adaptive test cases $\bar{\sigma}_1$ and $\bar{\sigma}_2$ can be *concatenated*, written $\bar{\sigma}_1.\bar{\sigma}_2$, by replacing each leaf of $\bar{\sigma}_1$ as follows. If the leaf is *null*, then is is replaced by $\bar{\sigma}_2$. Otherwise, if the leaf is some (x, f), then it is replaced by (x, f') where $f'(y) = \bar{\sigma}_2$ holds for all $y \in Y$. In this work, when concatenating adaptive test cases to be used in testing, the set Y is always implicitly given by the output alphabet of the specification FSM.

This method extends to sets Ω_1 and Ω_2 of adaptive test cases as follows:

$$\Omega_1.\Omega_2 := \begin{cases} \Omega_1 & \text{if } \Omega_2 = \emptyset \\ \{\bar{\sigma}_1.\bar{\sigma}_2 \mid \bar{\sigma}_1 \in \Omega_1 \wedge \bar{\sigma}_2 \in \Omega_2\} & \text{otherwise} \end{cases}$$

The special case of $\Omega_2 = \emptyset$ is handled in the same way as the case $\Omega_2 = \{null\}$. This is required, as in later applications, extensions are performed on possibly empty test sets.

In the presented algorithms, adaptive test cases are used to distinguish states. As this notion is based on the languages of those states, the set of IO sequences that can be observed when applying a test case to some state needs to be calculated. This calculation is performed by the following function, named IO.

Definition 5.3.2. Let $M = (S, s_1, X, Y, h)$ be some observable and completely specified FSM and let $s \in S$ be some state. Then, for an adaptive test case $\bar{\sigma} \in \Upsilon(X, Y)$, the set $IO(M, s, \bar{\sigma})$ is defined as follows:

- If $\bar{\sigma} = null$, then only the empty sequence is 'observed':

$$IO(M, s, null) := \{\epsilon\}$$

- If $\bar{\sigma} = (x, f)$ holds for some x and f, then the following holds:

$$IO(M, s, \bar{\sigma}) := \Big(\bigcup_{y \in h^{out}(s,x) \wedge y \notin Dom(f)} \{x/y\} \Big) \cup \Big(\bigcup_{y \in h^{out}(s,x) \wedge y \in Dom(f)} \{x/y\}.IO(M, h^y(s, x), f(y)) \Big)$$

  That is, for all responses y of M in state s to input x, it is checked whether f is defined for y. If it is not, then x/y is added to the result, as the adaptive test case provides no further inputs after this output. Otherwise, if $y \in Dom(f)$, then a set is created containing those sequences observed by applying $f(y)$ to the state s' reached by applying x/y to s in M, which is uniquely determined as M is observable and $y \in h^{out}(s, x)$. Then, $\{x/y\}$ is extended with this set, and the produced set is added to the result.

Therefore, IO is a function that maps its inputs, FSM $M = (S, s_1, X, Y, h)$, an adaptive test case $\bar{\sigma} \in \Upsilon(X, Y)$ and a state $s \in S$, to a subset of $(X/Y)^*$. ⊣

As an example, consider the FSM M^A described in Figure 5.1a and the adaptive test case $\bar{\sigma}_E$ given above. Applying $\bar{\sigma}_E$ to M^A in state s_1 then provides the following result:

$$IO(M^A, s_1, \bar{\sigma}_E) = \{aa/10, aa/11, ab/02, abb/010\}$$

Thus, this application observes several paths from the root to a leaf in $\bar{\sigma}_E$. In contrast, consider the application of the same test case to s_3:

$$IO(M^A, s_3, \bar{\sigma}_E) = \{aa/11\}$$

Here, for any input only one output is produced, and during application, M^A never leaves the state s_3, as the branch of $\bar{\sigma}_E$ that eventually applies b is never entered, guiding the application along a single path.

Following from the definition of the IO function, it is possible to treat input sequences as adaptive test cases: For some input sequence $\bar{x} = x_1, \ldots, x_k$ and output alphabet Y, an adaptive test case $\bar{\sigma}_{\bar{x}}^Y = (x_1, f_1)$ is constructed using functions $f_1, \ldots, f_k$ such that for all $1 \leq i < k$ the function f_i maps each $y \in Y$ to a test case (x_{i+1}, f_{i+1}), while f_k maps each $y \in Y$ to *null*. This essentially creates a tree such that along each path from the root to a leaf the nodes in the path are, in the order from the root, labelled $\bar{x}$ and such that for each $\bar{y} \in Y^k$ there exists a path in the tree such that the edges are, in order from the root, labeled $\bar{y}$. Then, the result of the IO function applying $\bar{\sigma}_{\bar{x}}^Y$ to some state s of some FSM M is equal to the set of all responses of applying $\bar{x}$ to s in M. Such $\bar{\sigma}_{\bar{x}}^Y$ are called the *adaptive test case for* $\bar{x}$ *in* Y. For empty input sequences, *null* is used.

In testing, input sequences are *followed* with adaptive test cases: For some set of input sequences $\mathcal{T} \in X^*$ and some set $\Omega \in \Upsilon(X, Y)$ of adaptive test cases, let $\mathcal{T}.\Omega$ denote the set of adaptive test cases where:

$$\mathcal{T}.\Omega := \bigcup_{\bar{x} \in \mathcal{T}} \{\bar{\sigma}_{\bar{x}}^Y\}.\Omega$$

That is, $\mathcal{T}.\Omega$ is created by converting the sequences in $\mathcal{T}$ to adaptive test cases and then concatenating them with Ω. The set Y is again implicitly given by the output alphabet of the specification FSM.

The definition of IO can be extended naturally to a function that takes as input a set of adaptive test cases instead of a single one. For some FSM $M = (S, s_1, X, Y, h)$, let Ω be a subset of $\Upsilon(X, Y)$. Then, for some state $s \in S$, the definition of IO is extended as follows:

$$IO(M, s, \Omega) := \bigcup_{\bar{\sigma} \in \Omega} IO(M, s, \bar{\sigma})$$

Based on the IO function, the use of adaptive test cases to (reliably) distinguish states can now be defined.

Definition 5.3.3. Let $M = (S, s_1, X, Y, h)$ be an observable FSM and $\bar{\sigma}$ an adaptive test case in $\Upsilon(X, Y)$. Then, for some states $s, s' \in S$, the test case $\bar{\sigma}$ *distinguishes* s and s' if and only if the following holds:

$$IO(M, s, \bar{\sigma}) \neq IO(M, s', \bar{\sigma})$$

That is, iff s and s' react differently to $\bar{\sigma}$.

Let $\bar{\sigma} = (x, f)$. Then, $\bar{\sigma}$ *r-distinguishes* s and s' if and only if for all $y \in h^{out}(s, x) \cap h^{out}(s', x)$ it holds that both $y \in Dom(f)$ and the test case $f(y)$ r-distinguishes the state $h^y(s, x)$ from the state $h^y(s', x)$.

These notions are extended to sets of adaptive test cases as follows: A set $\Omega \subseteq \Upsilon(X, Y)$ distinguishes (r-distinguishes) states s and s' if it contains some test case $\bar{\sigma} \in \Omega$ that distinguishes (r-distinguishes) s and s'. ⊣

Thus, an adaptive test case $\bar{\sigma}$ r-distinguishes state s and s' if its graphical representation constitutes a tree in $RD(s, s')$ (see Section 2.3) if the *null* nodes and all edges to such nodes are removed.

The next Lemma shows that, if for some adaptive test case $\bar{\sigma}$ and states s and s' the responses of s to $\bar{\sigma}$ are disjoint from those of s' to $\bar{\sigma}$, then the requirements for r-distinguishability of s and s' are met.

Lemma 5.3.4. *Let $M = (S, s_1, X, Y, h)$ be an observable FSM with states $s, s' \in S$. Then, an adaptive test case $\bar{\sigma} \in \Upsilon(X, Y)$ with $\bar{\sigma} \neq null$ r-distinguishes s and s' if and only if the following holds:*

$$IO(M, s, \bar{\sigma}) \cap IO(M, s', \bar{\sigma}) = \emptyset$$

Proof. Case 1: $\Longrightarrow$. Proof by induction on the height of $\bar{\sigma}$. Here, the case for a height of 1 holds immediately, as for the single input that $\bar{\sigma}$ applies, the reactions of s_1 and s_2 need to be disjoint for $\bar{\sigma}$ to r-distinguish them. For the induction step suppose that $\bar{\sigma} = (x, f)$ is of height $n > 1$ and r-distinguishes s and s'. I perform this step via proof by contradiction: Suppose that $IO(M, s, \bar{\sigma}) \cap IO(M, s', \bar{\sigma})$ is nonempty. Then, it must contain some $x\bar{x}/y\bar{y}$ such that $y \in h^{out}(s, x) \cap h^{out}(s', x)$. As $\bar{\sigma}$ r-distinguishes s and s', test case $f(y)$ must r-distinguish $s_y = h^y(s, x)$ and $s'_y = h^y(s', x)$, which is only possible if its height is at least 1. By definition, $height(f(y)) < n$, and hence by the induction hypothesis, it follows that $IO(M, s_y, f(y)) \cap IO(M, s'_y, f(y)) = \emptyset$. This contradicts the existence of $\bar{x}/\bar{y} \in IO(M, s, f(y)) \cap IO(M, s', f(y))$ and therefore of $x\bar{x}/y\bar{y} \in IO(M, s, \bar{\sigma}) \cap IO(M, s', \bar{\sigma})$.

Case 2: $\Longleftarrow$. Again, I apply induction on the height of $\bar{\sigma}$. The base case for $height(\bar{\sigma}) = 1$ once more follows immediately. For the induction step, assume that $\bar{\sigma} = (x, f)$ has height $n > 1$ and satisfies $IO(M, s, \bar{\sigma}) \cap IO(M, s', \bar{\sigma}) = \emptyset$. Let $y \in h^{out}(s, x) \cap h^{out}(s', x)$ be some shared response to x. Then, $y \in Dom(f)$, as otherwise $x/y \in IO(M, s, \bar{\sigma}) \cap IO(M, s', \bar{\sigma})$. Let $s_y = h^y(s, x)$ and $s'_y = h^y(s', x)$. From the definition of the IO function and the assumption that $IO(M, s, \bar{\sigma}) \cap IO(M, s', \bar{\sigma}) = \emptyset$ holds, it follows that $IO(M, s_y, f(y)) \cap IO(M, s'_y, f(y)) = \emptyset$. Thus, as $height(f(y)) < n$, by the induction hypothesis it holds that $f(y)$ r-distinguishes s_y and s'_y. Finally, as the above holds for all $y \in h^{out}(s, x) \cap h^{out}(s', x)$, the desired result follows. □

Having established the concept of r-distinguishability via adaptive test cases, the notion of characterizing sets can be similarly extended.

Definition 5.3.5. A set $\Omega \subseteq \upsilon(X, Y)$ of adaptive test cases is an *adaptive characterizing set* for an FSM $M = (S, s_1, X, Y, h)$ if and only if for every pair of r-distinguishable states $s, s' \in S$ there exists an adaptive test case $\bar{\sigma} \in \Omega$ that r-distinguishes s and s'. ⊣

Furthermore, reduction relations between states and FSMs can also be described with respect to one or more adaptive test cases: Let $M = (S, s_1, X, Y, h)$ and $M_I = (T, t_1, X, Y, h_I)$ be observable FSMs. Then, for some $\bar{\sigma} \in \Upsilon(X, Y)$ and $\Omega \subseteq \Upsilon(X, Y)$:

- $t \in T$ is a reduction of $s \in S$ *on* $\bar{\sigma}$ if and only if $IO(M_I, t, \bar{\sigma}) \subseteq IO(M, s, \bar{\sigma})$ holds. This is denoted as $t \preceq_{\bar{\sigma}} s$.
- $t \in T$ is a reduction of $s \in S$ *on* Ω if and only if $IO(M_I, t, \Omega) \subseteq IO(M, s, \Omega)$ holds. This is denoted as $t \preceq_{\Omega} s$.
- M_I is a reduction of M *on* $\bar{\sigma}$ if and only if $t_1 \preceq_{\bar{\sigma}} s_1$ holds. This is denoted as $M_I \preceq_{\bar{\sigma}} M$.
- M_I is a reduction of M *on* Ω if and only if $t_1 \preceq_{\Omega} s_1$ holds. This is denoted as $M_I \preceq_{\Omega} M$.

These definitions allow the use of adaptive test cases as test cases (in the sense of Chapter 3) in testing for reduction.

Definition 5.3.6. Suppose that M represents the specification while M_I represents the behaviour of the SUT. The $\underline{pass}$ relation for reduction using adaptive test cases is then defined as follows:

$$M_I \ \underline{pass}_{\preceq} \ \bar{\sigma} \equiv M_I \preceq_{\bar{\sigma}} M$$

That is, M_I *passes* $\bar{\sigma}$ if M_I responds to $\bar{\sigma}$ only in ways possible in M. ⊣

Finally, using the above definitions for reduction relations, it is possible to show that if two states are r-distinguishable, then their respective reductions are distinguishable:

Lemma 5.3.7. *Let Ω be a set of adaptive test cases that r-distinguishes states s and s' of some FSM M. Furthermore, let M' be some FSM with states t, t' such that $t \preceq_{\Omega} s$ and $t' \preceq_{\Omega} s'$. M and M' are both assumed to be completely specified and observable. Then Ω distinguishes t and t' in M'.*

Proof. Let $\bar{\sigma} \in \Omega$ be an adaptive test case that r-distinguishes s and s'. By definition, such $\bar{\sigma}$ must exist and also $\bar{\sigma} \neq null$. Due to t and t' being reductions on Ω with respect to s and s', the following holds: $IO(M', t, \bar{\sigma}) \subseteq IO(M, s, \bar{\sigma})$ and $IO(M', t', \bar{\sigma}) \subseteq IO(M, s', \bar{\sigma})$. Furthermore, from Lemma 5.3.4 it follows that $IO(M, s, \bar{\sigma}) \cap IO(M, s', \bar{\sigma}) = \emptyset$ and hence the following property holds:

$$IO(M, t, \bar{\sigma}) \cap IO(M, t', \bar{\sigma}) \subseteq IO(M, s, \bar{\sigma}) \cap IO(M, s', \bar{\sigma}) = \emptyset$$

Finally, as $\bar{\sigma} \neq null$ and M' is completely specified, $IO(M, t, \bar{\sigma}) \neq IO(M, t', \bar{\sigma})$ also holds, which provides the desired result. □

Therefore, after having calculated a set Ω of adaptive test cases that r-distinguishes states s, s' of some machine M, such a set can be used to distinguish states in some other FSM M', provided that for the states s, s' in M and t, t' in M' in question the corresponding reduction relations on Ω have been established. As illustrated above, the existence of such a relation can easily be established by applying Ω to the relevant states and comparing the responses.

5.4 Counting States

The concept of the presented algorithm, to extend sequences until a repetition must have occurred, is based on various methods to calculate lower bounds on the number of states in the FSM representing the SUT. This constitutes the second application of adaptivity in the presented algorithm, as these bounds are calculated based on observed responses of the SUT.

Throughout this section, I use $M = (S, s_1, X, Y, h)$ to denote the FSM representing the specification and $M_I = (T, t_1, X, Y, h_I)$ to denote the FSM representing the SUT. Both machines are assumed to be observable and completely specified. Additionally, it is assumed that there is an upper bound $m \in \mathbb{N}$ on the number of states in M_I (i.e., $|T| \leq m$), which is also at least as large as the number of states in M (i.e., $m \geq |S|$). Furthermore, V denotes a deterministic state cover of M.

5.4.1 Response Sets

The first method to check, how many different states can be reached with a given set of input sequences, is based on the responses observed by applying adaptive test cases at the end of these sequences. To accomplish this, I first define a function B that is similar to the IO function, but instead of passing the state from which to apply the adaptive test cases, it applies the test cases after an IO sequence.

Definition 5.4.1. For FSM M_I, an IO sequence $\bar{x}/\bar{y} \in L(M)$ and a set of adaptive test cases $\Omega \subseteq \Upsilon(X, Y)$ let

$$B(M_I, \Omega, \bar{x}/\bar{y}) := IO(M_I, h_I^{\bar{y}}(s, \bar{x}), \Omega)$$

The resulting set $B(M_I, \Omega, \bar{x}/\bar{y})$ denotes the *response set* or *behaviour* of the state t reached by $\bar{x}/\bar{y}$ in M_I to Ω, that is, the set of all IO sequences observed by applying Ω to t. ⊣

Then, the following proposition follows immediately from the definitions of B and IO, as states are distinct if they produce different response sets to the same inputs.

Proposition 5.4.2. *For IO sequences $\bar{x}/\bar{y}, \bar{x}'/\bar{y}' \in L(M_I)$ and a set of adaptive test cases $\Omega \subseteq \Upsilon(X,Y)$ the following holds:*

$$B(M_I, \Omega, \bar{x}/\bar{y}) \neq B(M_I, \Omega, \bar{x}'/\bar{y}') \Leftrightarrow h_I^{\bar{y}}(s_1, \bar{x}) \neq h_I^{\bar{y}'}(s_1, \bar{x}')$$

Therefore, $B(M_I, \Omega, \bar{x}/\bar{y}) \neq B(M_I, \Omega, \bar{x}'/\bar{y}')$ implies that the sequences $\bar{x}/\bar{y}$ and $\bar{x}'/\bar{y}'$ reach different states in M_I.

Thus, using a set of adaptive test cases Ω, a lower limit on the number of states reached by some set of input sequences can be calculated by applying Ω to each reached state and then counting the number of distinct response sets produced.

Definition 5.4.3. For FSM M_I, a set of input sequences $\mathcal{T} \subseteq X^*$ and a set of adaptive test cases $\Omega \subseteq \Upsilon(X,Y)$, let $D(M_I, \Omega, \mathcal{T})$ be defined as follows:

$$D(M_I, \Omega, \mathcal{T}) := \{B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{x} \in \mathcal{T} \wedge \bar{x}/\bar{y} \in L(M_I)\}$$

$D(M_I, \Omega, \mathcal{T})$ is the set of *distinct* responses of M_I to Ω *after* $\mathcal{T}$. ⊣

By the previous proposition, each element in $D(M_I, \Omega, \mathcal{T})$ must represent at least one state.

As an example, consider the adaptive test cases $\bar{\sigma}_E^1$ and $\bar{\sigma}_E^2$ illustrated in Figure 5.3:

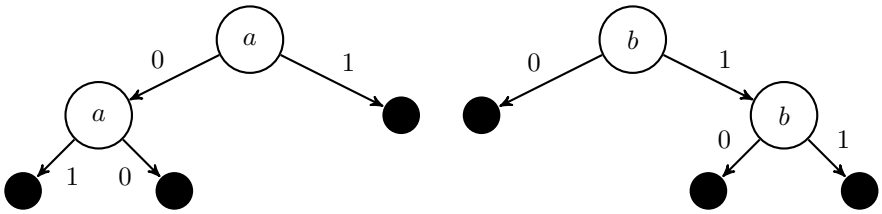


Figure 5.3: Adaptive test cases $\bar{\sigma}_E^1$ (left) and $\bar{\sigma}_E^2$ (right)

Let $\Omega = \{\bar{\sigma}_E^1, \bar{\sigma}_E^2\}$. Applying this set to the FSM M^A after the input a produces the following exemplary results:

$$B(M^A, \Omega, a/0) = \{a/1, b/2, bb/10\}$$
$$B(M^A, \Omega, a/1) = \{a/1, aa/01, b/0\}$$
$$D(M^A, \Omega, \{a\}) = \{\{a/1, b/2, bb/10\}, \{a/1, aa/01, b/0\}\}$$

Therefore, by applying the input sequence a to M_A and following it with Ω, it is possible to deduce without further knowledge about M_A that applying a not only produces two different responses, but also that two distinct states are reached by it.

5.4.2 States Visited by a Sequence

The second method considers, how many distinct states are visited by a single sequence. More specifically, as the algorithm extends sequences from a deterministic state cover V, it examines how many times, and by which prefixes, some specific state is visited by a sequence $\bar{x}/\bar{y}$ applied after an input sequence in V.

Definition 5.4.4. For IO sequences $\bar{v}/\bar{v}'$ and $\bar{x}/\bar{y}$ such that $\bar{v}\bar{x}/\bar{v}'\bar{y} \in L(M) \cap L(M_I)$ and some state $s \in S$, let the set $R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y})$ be defined as follows:

$$R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}) := \{\bar{v}\bar{x}'/\bar{v}'\bar{y}' \mid \bar{x}'/\bar{y}' \in pref(\bar{x}/\bar{y}) \setminus \{\epsilon\} \wedge s = h^{\bar{v}'\bar{y}'}(s_1, \bar{v}\bar{x}')\}$$

That is, $R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y})$ contains all prefixes of $\bar{v}\bar{x}/\bar{v}'\bar{y}$ that are longer than $\bar{v}/\bar{v}'$ and *reach* s in M. ⊣

For example, consider the sequence $bbba/0102$ applied to M^A after the empty sequence $\epsilon \in V$. For s_2, the set $R(M^A, s_2, \epsilon, bbba/0102)$ then contains all prefixes of $bbba/0102$ that reach s_2, that is, the sequences $b/0$, $bbb/010$ and $bbba/0102$. As the first of these sequences, $b/0$, is contained in the deterministic state cover V^A, the sequence $bbba/0102$, when considered by the algorithm (as described in subsequent sections), is split into $b/0$ and $bba/102$, with the corresponding set $R(M^A, s_2, b/0, bba/102)$ containing only $bbb/010$ and $bbba/0102$ since only nonempty prefixes of $bba/102$ are considered.

The number of such prefixes is used in the algorithm based on the following property.

Lemma 5.4.5. *Let $\bar{v}\bar{x}/\bar{v}'\bar{y} \in L(M) \cap L(M_I)$ be an IO sequence and $s \in S$ some state. Let q be the state of the product machine $P(M, M_I)$ that is reached by $\bar{v}/\bar{v}'$. If no state of the product machine is repeated along $\bar{x}/\bar{y}$ when applied to q, then the sequences in $R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y})$ must reach $|R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y})|$ different states in M_I.*

Proof. All states reached by a sequence in $R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y})$ have the same state-component in M, namely s. Thus, these states can only differ by their state-component in M_I. Also, as no state is repeated along $\bar{x}/\bar{y}$ applied to q, each sequence in $R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y})$ must reach a different state in $P(M, M_I)$. This is only possible if no two states reached by a sequence in $R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y})$ have the same state-component in M_I and hence the result follows. □

This concept can be strengthened to include the reaction of M_I to the input sequences in V. In doing so, it is useful to consider the set that captures all possible reactions of M_I to V, which constitutes another application of adaptivity:

Definition 5.4.6. For some deterministic state cover $V = \{\bar{v}_1, \ldots, \bar{v}_n\}$ of M, let $Perm(V, M_I)$ contain all *permutations* of reactions of M_I to every $\bar{v}_i \in V$ as follows:

$$Perm(V, M_I) := \left\{\{\bar{v}_1/\bar{v}_1, \ldots, \bar{v}_n/\bar{v}_n'\} \;\middle|\; \forall 1 \leq i \leq n : \bar{v}_1' \in h_I^{out}(s, \bar{v})\right\}$$

If the machine M_I is known from the context, then I write V' as a shorter identifier to denote the set $Perm(V, M_I)$. ⊣

Notably, each $V'' \in V'$ must contain the empty sequence ϵ, as ϵ is by definition contained in V, and there exists only the response ϵ to it for any FSM in any state.

I have shown that a set $V = V^A = \{\epsilon, b\}$ is a deterministic state cover for M^A. As the initial state s_1 of M^A allows only a single reaction to b, then V' contains only the single element $\{\epsilon, b/0\}$.

In contrast, for the example FSM M_E given in Figure 2.1 on page 8, state s_3 could be d-reached via ab, and hence $W_E = \{\epsilon, b, ab\}$ is deterministic state cover of M_E, for which there are two different sets of reactions: $\{\epsilon, b/0, ab/01\}$ and $\{\epsilon, b/0, ab/10\}$.

Next, the function R^+ combines the previous concepts.

Definition 5.4.7. Let $V'' \in V'$ be a set of responses of M_I to V and $s \in S$ some state of M. Furthermore, let $\bar{v}/\bar{v}'$ and $\bar{x}/\bar{y}$ be some IO sequences such that $\bar{v}\bar{x}/\bar{v}'\bar{y} \in L(M) \cap L(M_I)$. Then, the set $R^+(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$ is defined as follows:

$$R^+(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') := \\ R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}) \cup \{\bar{v}_1/\bar{v}'_1 \in V'' \mid \{s\} = h^{\bar{v}'_1}(s_1, \bar{v}_1)\}$$

That is, it is defined as the result of adding to $R(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y})$ all sequences in V'' that reach s in M. ⊣

Since V is a deterministic state cover of M, at most one sequence in V d-reaches s and hence at most one sequence is added, compared to the set generated by the R function for the same inputs.

For example, consider FSM M^A and sequence $bbba/0102$, and split the latter into a prefix $b/0$ and suffix $bba/102$. Having established that for M^A and its deterministic state cover $V = V^A = \{\epsilon, b\}$ it holds that $V' = \{V''\}$ with $V'' = \{\epsilon, b/0\}$, the result of the R^+ function is as follows:

$$R^+(M^A, s_2, b/0, bba/102, V'') = \{bbb/010, bbba/0102\} \cup \{b/0\}$$

Finally, the result of Lemma 5.4.5 can be extended for R^+ as follows.

Lemma 5.4.8. *Let $\bar{v}\bar{x}/\bar{v}'\bar{y} \in L(M) \cap L(M_I)$ be an IO sequence and $s \in S$ some state. Let q be the state of the product machine $P(M, M_I)$ that is reached by $\bar{v}/\bar{v}'$. If no state of the product machine is repeated along $\bar{x}/\bar{y}$ when applied to q and no state visited by $\bar{x}/\bar{y}$ applied to q is d-reached by some $\bar{v} \in V$, then for all $V'' \in V'$, the sequences in $R^+(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$ must reach $|R^+(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')|$ different states in M_I.*

I omit the analogous proof.

5.4.3 A Lower Bound

The previously introduced notions constitute all required components for the main tool used in the adaptive state counting algorithm to control the extension of sequences, the function LB. This function calculates for some given IO sequence a lower bound on the number of states in M_I, under the assumption that no state of the product machine $P(M, M_I)$ is repeated.

Let Ω be an adaptive characterizing set of M. Furthermore, let $S_1 \subseteq S$, $\bar{v} \in V$, $V'' \in V'$, $\bar{v}/\bar{v}' \in V''$ and $\bar{v}\bar{x}/\bar{v}'\bar{y} \in L(M) \cap L(M_I)$. Finally, let $\mathcal{T} \subseteq X^*$ denote a set of input sequences.

Definition 5.4.9. The *lower bound* function LB is defined as follows:

$$LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}, S_1, \Omega, V'') :=$$
$$\sum_{s \in S_1} \left| R^+(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') \right| + \tag{LB1}$$
$$\left| D(M_I, \Omega, \mathcal{T}) \setminus \left(\bigcup_{\substack{s' \in S_1, \\ \bar{x}_1/\bar{y}_1 \in R^+(M, s', \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')}} B(M_I, \Omega, \bar{x}_1/\bar{y}_1) \right) \right| \tag{LB2}$$

That is, it is defined as the sum of two parts (LB1) and (LB2). ⊣

For example, consider FSMs M^A and M_I^A and the sequence $bb/01$, and split the latter into $b/0$ and $b/1$. Furthermore, let $\mathcal{T} = V^A \cup V^A.X^A = \{\epsilon, a, b, ba, bb\}$, $S_1 = \{s_1, s_2, s_3\}$, and $V'' = \{\epsilon, b/0\}$. Finally, let Ω be the set $\{\bar{\sigma}_a^{Y^A}, \bar{\sigma}_b^{Y^A}\}$ (i.e., the adaptive test cases whose application is equivalent to simply applying the sequences a and b and observing all outputs). As established above, $\{a, b\}$ is a characterizing set of M^A, and the states of M^A are pairwise r-distinguishable. Then, in calculating $LB(M^A, M_I^A, b/0, b/1, \mathcal{T}, S_1, \Omega, V'')$, the following intermediate results are used:

$$R^+(M^A, s_1, b/0, b/1, V'') = \{bb/01\} \cup \{\epsilon\} = \{\epsilon, bb/01\}$$
$$R^+(M^A, s_2, b/0, b/1, V'') = \emptyset \cup \{b/0\} = \{b/0\}$$
$$R^+(M^A, s_3, b/0, b/1, V'') = \emptyset \cup \emptyset = \emptyset$$

$$B(M_I^A, \Omega, \epsilon) = \{a/1, b/0\}$$
$$B(M_I^A, \Omega, b/0) = \{a/0, a/2, b/1\}$$
$$B(M_I^A, \Omega, bb/01) = \{a/1, b/0\}$$

$$D(M_I^A, \Omega, \mathcal{T}) = \{B(M_I^A, \Omega, \bar{x}/\bar{y}) \mid \bar{x}/\bar{y} \in \{\epsilon, a/1, b/0, ba/00, ba/02, bb/01\}\}$$

where $D(M_I^A, \Omega, \mathcal{T})$ thus contains the response sets of all three states in M_I^A to Ω since ϵ, $b/0$ and $ba/00$ reach t_1, t_2 and t_3, respectively. Hence, the following holds:

$$D(M_I^A, \Omega, \mathcal{T}) = \{\{a/1, b/0\}, \{a/0, a/2, b/1\}, \{a/1, b/2\}\}$$

Thus, the three R^+ sets together contain three sequences, while $D(M_I^A, \Omega, \mathcal{T})$ contains exactly one response set to Ω not observed after any of these three sequences. Therefore, for these inputs the lower bound is calculated to be 4.

The significance of each of the two parts of the LB function, beginning with (LB1), is as follows.

By Lemma 5.4.8, it holds that if no state of the product machine is repeated, then for each $s \in S_1$ the sequences in $R^+(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$ must reach $|R^+(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')|$ different states in M_I. Assume that for all states $s', s'' \in S$ with $s' \neq s''$ it holds that Ω distinguishes every state of M_I reached by a sequence in $R_{s'}^+ = R^+(M, s', \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$ from every state of M_I reached

by a sequence in $R^+_{s''} = R^+(M, s'', \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$. This is only possible if the sets of states reached by sequences in $R^+_{s''}$ are disjoint from those reached by sequences in $R^+_{s''}$. Thus, again using Lemma 5.4.8, it follows that under these conditions $\sum_{s \in S_1} \left| R^+(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') \right|$ different states of M_I are reached by prefixes of $\bar{x}/\bar{y}$ extending $\bar{v}/\bar{v}'$.

The assumption of Ω distinguishing reached states automatically holds if both of the following conditions hold. (1) The states in S_1 are pairwise r-distinguishable. (2) For all $s \in S$ it holds that each state t of M_I reached by some sequence in $R^+(M, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$ satisfies $t \preceq_\Omega s$. That is, M_I is a reduction of M on Ω along every state reached by extending $\bar{v}/\bar{v}'$ with some prefix of $\bar{x}/\bar{y}$. The assumption then follows from Lemma 5.3.7.

The second term, (LB2), counts the number of distinct response sets of M_I to Ω observed for all input sequences in $\mathcal{T}$ that are not also observed for any state of M_I reached by extending $\bar{v}/\bar{v}'$ with some prefix of $\bar{x}/\bar{y}$. By Proposition 5.4.2, each distinct response set implies the existence of a distinct state in M_I, and hence, there are at least (LB2) additional states in M_I that have not been accounted for in (LB1).

In Lemma 5.4.11, I describe the assumptions under which the LB function is to be applied in the algorithm. When formulating this Lemma, I organized its assumption in three parts: prerequisites, repetitions by prefix and repetitions by state cover, which are defined using the predicates $Prereq$, Rep_{Pre} and Rep_{Cov} as follows:

Definition 5.4.10. First, the formula

$$Prereq(M, M_I, V, V'', S_1, \Omega, \mathcal{T}, \bar{v}/\bar{v}', \bar{x}/\bar{y})$$

describes the *prerequisites* and holds if and only if the following conditions are satisfied:

1. The sequence $\bar{v}/\bar{v}'$ is the maximum length prefix of $\bar{v}\bar{x}/\bar{v}'\bar{y}$ in V''.
2. All sequences in $\mathcal{T}$ have been followed with Ω, and the resulting sequences have been applied to M_I sufficiently often to ensure that all responses have been observed.
3. No failure has been observed during application of $\mathcal{T}.\Omega$. That is, for no test in $\mathcal{T}.\Omega$ the SUT M_I produced a response not contained in the responses of M to the same sequence (i.e., $M_I \preceq_{\mathcal{T}.\Omega} M$ holds).
4. $\mathcal{T}$ contains every sequence in V and every sequence $\bar{v}\bar{x}''$ where $\bar{x}'' \in pref(\bar{x})$.
5. For all $s_1, s_2 \in S_1$ with $s_1 \neq s_2$, it holds that Ω distinguishes every state reached by some sequence in $R^+(M, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$ from every state of M_I reached by a sequence in $R^+(M, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$.

Next, the formula

$$Rep_{Pre}(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y})$$

describes *repetitions by prefix* and holds if and only if there exist some $\bar{x}_1/\bar{y}_1 \in pref(\bar{x}/\bar{y}) \setminus \{\epsilon\}$ and $\bar{x}_2/\bar{y}_2 \in pref(\bar{x}/\bar{y})$ with $\bar{x}_1/\bar{y}_1 \neq \bar{x}_2/\bar{y}_2$, such that both $\bar{v}\bar{x}_1/\bar{v}'\bar{y}_1$ and $\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2$ reach the same state q' of the product machine of M

and M_I if applied to its initial state. That is, it holds if and only if there is some repetition of states along $\bar{x}/\bar{y}$ after $\bar{v}/\bar{v}'$ or if the state reached by $\bar{v}/\bar{v}'$ is also reached by a non-empty prefix of $\bar{x}/\bar{y}$ applied after $\bar{v}/\bar{v}'$.

Finally, the formula

$$Rep_{Cov}(M, M_I, V'', \bar{v}/\bar{v}', \bar{x}/\bar{y})$$

describes *repetitions by state cover*. This formula holds if and only if there exist some $\bar{x}_1/\bar{y}_1 \in pref(\bar{x}/\bar{y}) \setminus \{\epsilon\}$ and $\bar{v}_1/\bar{v}_1' \in V''$ such that $\bar{v}\bar{x}_1/\bar{v}'\bar{y}_1$ and $\bar{v}_1/\bar{v}_1'$ reach the same state q' of the product machine of M and M_I if applied to its initial state. That is, it holds if and only if some state reached along $\bar{x}/\bar{y}$ after $\bar{v}/\bar{v}'$ is also reached by some sequence in V''. ⊣

I shorten the above expressions to $Prereq$, Rep_{Pre}, and Rep_{Cov}, respectively, if their parameters are obvious in the current context.

These predicates then allow the concise presentation of the following Lemma.

Lemma 5.4.11. *Suppose that Prereq, Rep_{Pre} and Rep_{Cov} hold for M, M_I, $\bar{v}/\bar{v}'$, $\bar{x}/\bar{y}$, $\mathcal{T}$, S_1, Ω and V''. Then M_I must have at least*

$$LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}, S_1, \Omega, V'')$$

different states.

Proof. This Lemma follows directly from the explanations of the components (LB1) and (LB2) of the LB function as given on page 39. □

The algorithm employs this Lemma to extend input sequences, beginning from V, as long as they might constitute prefixes of *minimal sequences to failures* that extend some $\bar{v} \in V$:

Definition 5.4.12. A sequence $\bar{x}_f/\bar{y}_f = \bar{x}_f' x_f/\bar{y}_f' y_f$ is a *sequence to a failure* if and only if $\bar{x}_f'/\bar{y}_f' \in L(M) \cap L(M_I)$ and $\bar{x}_f' x_f/\bar{y}_f' y_f \in L(M_I) \setminus L(M)$.

Furthermore, a sequence to a failure $\bar{x}_f/\bar{y}_f = \bar{v}\bar{x}/\bar{v}'\bar{y}$, where $\bar{v}$ is the maximum length prefix of $\bar{x}_f$ in V, is a *minimal sequence to a failure extending* V, if there exists no sequence to a failure $\bar{v}_o\bar{x}_o/\bar{v}_o'\bar{y}_o$ such that $\bar{v}_o \in V$ and $|\bar{x}_o| < |\bar{x}|$, that is, if no failure can be reached by applying a sequence shorter than $\bar{x}$ after some sequence in V.

This concept can be extended to input sequences as follows: Some input sequence $\bar{x}_f$ is a *minimal sequence to a failure extending* V if and only if there exists some output sequence $\bar{y}_f$ such that $\bar{x}_f/\bar{y}_f$ is a minimal sequence to a failure extending V. ⊣

Analogous, in terms of the product machine $P(M, M_I)$ a sequence $\bar{x}_f/\bar{y}_f$ is a minimal sequence to a failure extending V if the following is true:

- $\bar{x}_f/\bar{y}_f = \bar{v}\bar{x}_f' x_f/\bar{v}'\bar{y}_f' y_f$ where $\bar{v}$ is the largest prefix of $\bar{x}_f$ that is in V,
- $\bar{v}\bar{x}_f'/\bar{v}'\bar{y}_f'$ reaches some state q other than $Fail$ in $P(M, M_I)$ such that there exists a transition x_f/y_f from q to $Fail$, and
- there exists no sequence $\bar{x}_o$ with $|\bar{x}_o| < |\bar{x}_f' x_f|$ such that the extension of some $\bar{v} \in V$ with $\bar{x}_o$ is a sequence to a failure.

Searching for such minimal sequences is sufficient according to the following Lemma:

Lemma 5.4.13. *If $L(M_I) \setminus L(M)$, then there exists some minimal sequence to a failure extending V.*

Proof. From the assumption, it follows that the state $Fail$ is reachable in the product machine $P(M, M_I)$. Next, for each $\bar{v} \in V$, let $F_{\bar{v}}$ denote the set of all sequences $\bar{x}$ such that there exists some sequence to a failure $\bar{v}\bar{x}/\bar{v}'\bar{y} \in L(M_I)$ that also satisfies that no state q' of $P(M, M_I)$ is repeated along $\bar{x}/\bar{y}$ applied to the state reached by $\bar{v}/\bar{v}'$. Each such F set must be finite since there are only finitely many sequences that can be applied to some FSM without repeating any state. As $\epsilon \in V$ and $Fail$ is reachable, there must exist some sequence to a failure $\bar{v}_f\bar{x}_f/\bar{v}'_f\bar{y}_f$ such that $\bar{v}_f$ is the maximum length prefix of $\bar{v}_f\bar{x}_f$ in V. From this sequence, by removing all cycles and thus all repetitions of states along $\bar{x}_f/\bar{y}_f$ applied after $\bar{v}_f/\bar{v}'_f$, a sequence $\bar{x}'_f/\bar{y}'_f$ can be created that must be contained in $F_{\bar{v}_f}$.

Thus, there exists at least one nonempty F set and hence, as V and the F sets are finite, some $\bar{v}_s \in V$ and $\bar{x}_s \in F_{\bar{v}_s}$ must exist such that the following holds:

$$\forall \bar{v} \in V : \forall \bar{x} \in F_{\bar{v}} : |\bar{x}_s| \leq |\bar{x}|$$

That is, $\bar{v}_s\bar{x}_s$ is a minimal sequence to a failure extending V. □

For the purpose of proving $M_I \not\preceq M$, it is then sufficient to find a single minimal sequence to a failure extending V. Conditions (1) and (3) of $Prereq$ reflect this restriction. When checking some sequence $\bar{x}/\bar{y}$, only such partitionings $\bar{x}/\bar{y} = \bar{v}\bar{x}_1/\bar{v}'\bar{y}_1$ are considered, where $\bar{v}/\bar{v}'$ is the maximum length prefix in V''. Calculating the lower bound for some shorter prefix in V'' would immediately cause a repetition by state cover and thus violate the assumptions of Lemma 5.4.11, regardless of whether $\bar{x}/\bar{y}$ can form a prefix of a minimal sequence to a failure extending V. Furthermore, if M_I reacts to some test in $\mathcal{T}.\Omega$ in a way not allowed in M (i.e., if a failure can be observed when applying $\mathcal{T}.\Omega$), then a sequence to a failure is already found. It is possible to ensure that this is a minimal sequence to a failure extending V by iteratively extending the sequences to check, starting from the reactions of M_I to V.

The algorithm extends a sequence in this manner until the lower bound exceeds m. If all assumptions of Lemma 5.4.11 are satisfied, then the calculated lower bound can not be valid, as the number of states of M_I is at most m. I use the predicate $Size_m$ to denote that the lower bound is *valid* with respect to $m \geq |T|$. The formula

$$Size_m(M, M_I, V'', S_1, \Omega, \mathcal{T}, \bar{v}/\bar{v}', \bar{x}/\bar{y})$$

holds if and only if the lower bound on the number of states of M_I, calculated by LB, is valid. In other words, it holds if and only if the following is *true*:

$$LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}, S_1, \Omega, V'') \leq m$$

Again, I omit the parameters if obvious in the context.

This allows the formulation of Lemma 5.4.11 as follows:

$$Prereq \wedge \neg Rep_{Pre} \wedge \neg Rep_{Cov} \implies Size_m$$

which is equivalent to

$$\neg Size_m \implies \neg Prereq \vee Rep_{Pre} \vee Rep_{Cov}$$

Thus, if the sequence $\bar{v}\bar{x}/\bar{v}'\bar{y}$, with $\bar{v}$ being the maximum length prefix of $\bar{v}\bar{x}$ in V, exceeds m with the lower bound calculated for it, that is, if

$$LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}, S_1, \Omega, V'') > m$$

holds, then $Size_m$ does not hold. In turn, at least one of the assumptions of Lemma 5.4.11 must be violated and thus the prerequisites do not hold or a repetition occurs.

Recall the exemplary calculation of the lower bound for M^A and M_I^A based on the sequence $bb/01$, and additionally suppose that $m = 3$. Then, $\neg Size_m$ holds, as the calculated lower bound of 4 exceeds m. Furthermore, the choices of $\mathcal{T}$, Ω, V and V'' satisfy *Prereq*. Since no repetition in the sense of Rep_{Pre} occurs, it follows that Rep_{Cov} must hold. This is true, as the sequence $bb/01$ reaches t_1 in M_I^A, which is also reached by $\epsilon \in V^A$. Here it is also true that $bb/01$ cannot be a prefix to a minimal sequence to a failure extending V^A: For every sequence to a failure $bb\bar{x}_f/01\bar{y}_f$ (which is extending $b/0$), the sequence $\bar{x}_f/\bar{y}_f$ (which is extending ϵ) is also a sequence to a failure, where trivially $|\bar{x}_f| < |b\bar{x}_f|$ holds.

The adaptive state counting algorithm is based on the assumption that if for all responses of M_I to some input sequence $\bar{x}$, the lower bound exceeds m for some $S_1 \subseteq S$ and $V'' \in V'$, then $\bar{x}$ cannot be a prefix of a minimal sequence to a failure extending V, and hence, $\bar{x}$ need not be extended.

5.5 Generated Test Suite

In this section, I define the test suite $\mathcal{T}.\Omega$ for some $\mathcal{T} \subseteq X^*$ generated by the adaptive state counting algorithm, based on the description given in [**hierons**]. This formulaic representation of the algorithm does not terminate upon observing a failure, which is left to the program representation in Section 5.6.

As in the previous sections, I use $M = (S, s_1, X, Y, h)$ to denote the FSM representing the specification and $M_I = (T, t_1, X, Y, h_I)$ to denote the FSM representing the SUT. Both machines are assumed to be observable and completely specified, while $m \in \mathbb{N}$ with some $m \geq |S|$ is an upper bound on the number of states in M_I (i.e., $|T| \leq m$). Furthermore, let V and Ω denote a deterministic state cover and an adaptive characterizing of M, respectively.

First, I define a function that calculates for some sequence its maximum length prefix that is also contained in a given set of sequences, which is required for the previously introduced notions. Let $mcp(\bar{z}, W)$ denote the *maximum length common prefix* $\bar{z}'$ of some sequence $\bar{z} \in Z^*$ and some set of sequences $W \subseteq Z^*$ such that the following holds:

$$mcp(\bar{z}, W) = \bar{z}' \Leftrightarrow \bar{z}' \in pref(\bar{z}) \cap W \wedge \forall \bar{z}'' \in pref(\bar{z}) \cap W : |\bar{z}'| \geq |\bar{z}''|$$

Such a common prefix must always exist when considering $W = V$ and some input sequence $\bar{z}$, as V contains ϵ.

The algorithm then intuitively constructs the set $\mathcal{T}$ as follows. Initially, the set of currently considered sequences $\mathcal{C}$ and the result $\mathcal{T}$ are set to V. Next,

$\mathcal{T}$ is extended by iteratively first applying $\mathcal{C}.\Omega$ and then extending all those sequences in $\mathcal{C}$ by each element of X that might still form prefixes of minimal sequences to failures extending V, setting the resulting sequences as the new value of $\mathcal{C}$ and adding them to $\mathcal{T}$. This process is performed as long as there are sequences to extend in $\mathcal{C}$. The final value of $\mathcal{T}$ followed by Ω, which is always added to distinguish states, is then the test suite generated by the algorithm.

More formally, the following sets of sequences are used to indicate the state of the algorithm in some iteration $i \in \mathbb{N}$ with $i > 0$:

- $\mathcal{C}_i$ contains all sequences considered for extension at the start of iteration i. These are called the *current* sequences in iteration i.
- $\mathcal{R}_i$ contains all sequences deleted from $\mathcal{C}$ in iteration i. These are the *removed* sequences in iteration i.
- $\mathcal{E}_i$ contains all sequences of $\mathcal{C}_i$ that have not been deleted during iteration i, *extended* by the alphabet X.
- $\mathcal{T}_i$ contains all sequences applied to the SUT during iteration i and previous iterations. These are called the *tested* sequences up to and including iteration i.

Those sets are constructed recursively as follows.

Definition 5.5.1. Let $\mathcal{T}_i$, $\mathcal{C}_i$, $\mathcal{R}_i$ and E_i for some iteration $i \in \mathbb{N}$ with $i > 0$ be given by the following definitions:

$$
\begin{aligned}
\mathcal{R}_0 &:= \emptyset \\
\mathcal{C}_1 &:= V \\
\mathcal{T}_1 &:= V \\
\mathcal{E}_n &:= (\mathcal{C}_n \setminus \mathcal{R}_n).X && \text{for } n \geq 1 \\
\mathcal{C}_{n+1} &:= \mathcal{E}_n \setminus \mathcal{T}_n && \text{for } n \geq 1 \\
\mathcal{T}_{n+1} &:= \mathcal{T}_n \cup \mathcal{C}_{n+1} && \text{for } n \geq 1
\end{aligned}
$$

with the set of removed sequences R_{n+1} for $n \geq 0$ being the most complex expression, as several requirements for removal have to be checked:

$$
\begin{aligned}
\mathcal{R}_{n+1} := \Big\{ \bar{x}' \in \mathcal{C}_{n+1} \;\Big|\; & \\
& \forall \bar{y}' \in h_I^{out}(t_1, \bar{x}') : \exists S_1 \subseteq S, V'' \in V', \bar{v}/\bar{v}' \in V'', \bar{x}/\bar{y} \in (X/Y)^* : \\
& \quad \bar{v}\bar{x}/\bar{v}'\bar{y} = \bar{x}'/\bar{y}' && (1) \\
& \quad \wedge\, \bar{v}/\bar{v}' = mcp(\bar{x}'/\bar{y}', V'') && (2) \\
& \quad \wedge\, \forall s_1, s_2 \in S_1, s_1 \neq s_2 : && (3) \\
& \qquad \forall \bar{x}_1/\bar{y}_1 \in R^+(M, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\
& \qquad\quad \forall \bar{x}_2/\bar{y}_2 \in R^+(M, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\
& \qquad\qquad B(M_I, \Omega, \bar{x}_1/\bar{y}_1) \neq B(M_I, \Omega, \bar{x}_2/\bar{y}_2) \\
& \quad \wedge\, LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}_{n+1}, S_1, \Omega, V'') > m \Big\} && (4)
\end{aligned}
$$

A sequence $\bar{x} \in \mathcal{R}_i$ is said to be *removed* in iteration i, while a sequence $\bar{x} \in \mathcal{C}_i \setminus \mathcal{R}_i$ is said to be *extended* in iteration i. ⊣

That is, a sequence $\bar{x}' \in C_{n+1}$ is contained in $\mathcal{R}_{n+1}$ if for every response $\bar{y}'$ of M_I to it there exist $S_1 \subseteq S$ and $V'' \in V'$ such that

- By (1) the sequence $\bar{x}'/\bar{y}'$ is equal to a sequence $\bar{v}\bar{x}/\bar{v}'\bar{y}$.
- By (2) the above splitting of $\bar{x}'/\bar{y}'$ satisfies the requirement that $\bar{v}/\bar{v}'$ is the maximum length prefix of $\bar{x}'/\bar{y}'$ in V''.
- By (3) the set of adaptive test cases Ω distinguishes, for all states s_1, s_2 in M where $s_1 \neq s_2$, each state of M_I reached by some sequence in $R^+(M, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$ from each state of M_I reached by a sequence in $R^+(M, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$.
- By (4) the lower bound exceeds m.

5.5.1 Test Suite for the Accompanying Example

Having provided the formulaic representation of the adaptive state counting algorithm, I now provide an exemplary application of it to the FSMs given in Figure 5.1 on page 28. As presented in Section 5.1, the set $V^A = \{\epsilon, b\}$ can be used as a deterministic state cover, which induces only the single permutation $V'' = \{\epsilon, b/0\}$.

Next, analogously to the example of the LB function, from the characterizing set $W = \{a, b\}$ I derive the adaptive characterizing set $\Omega = \{\bar{\sigma}_a^{Y^A}, \bar{\sigma}_b^{Y^A}\}$, which is essentially the same as W. Then, as Ω r-distinguishes every pair of distinct states of M^A, when choosing sets of states S_1 during the calculation of the lower bound, it is sufficient to only consider $S_1 = S$ since for any $S_2 \subset S$ the calculated lower bound cannot, by definition, exceed that calculated for $S_1 \supset S_2$.

Finally, to keep the example short and to allow for comparison of the result to that obtained by using the non-adaptive state counting algorithm in Section 5.2, I set m to 3, which is the number of states of both M^A and M_I^A. Thus, all required arguments are established and the calculation of the sets that are used in generating the test suite for testing whether M_I^A is a reduction of M^A can be performed.

First, $\mathcal{T}_1 = \mathcal{C}_1 = V$ holds and thus, when calculating $\mathcal{R}_1$, so far only the responses of t_1 and t_2 to Ω have been observed. As a result, $\mathcal{R}_1 = \emptyset$, since for both sequences the calculated lower bound is just 2, because there are two d-reachable states in M^A.

In the next iteration, $\mathcal{C}_2 = \mathcal{C}_1.X \setminus V = \{a, ba, bb\}$ and also $\mathcal{T}_2 = \mathcal{T}_1 \cup \mathcal{C}_2 = V \cup V.X$ follows. Thus, in particular $ba \in \mathcal{T}_2$ holds, which, as $ba/00$ reaches t_3 in M_I^A, allows the observation of the response set of t_3 to Ω. This additional observation is sufficient to have $\mathcal{R}_2 = \{a, bb\}$, as either of those sequences extends some sequence in V by a single input, adding a sequence to the R^+ sets, while at the same time not visiting t_3, so that the knowledge of the response set of t_3 is sufficient to indicate that there would have to exist a fourth state in M_I^A, in addition to the three visited states, if no repetition were to occur. However, as $m = 3$, such a repetition must occur and the sequences are removed. This does not apply to $ba/00$, as this sequence reaches t_3, thus directly observing the response set of that state.

Therefore, in the subsequent iteration, the set $\mathcal{C}_3$ contains only the extensions of ba, namely baa and bab. Both are of sufficient length to require a lower bound

of 4 to avoid repetitions, which again exceeds m. Thus $\mathcal{C}_4 = \emptyset$, which also holds for any later iteration, and which indicates that the algorithm terminates.

The exact calculations for the lower bounds of the above sequences are given in Table 5.1. Here, for each input sequence $\bar{x}'$ and response $\bar{y}'$ to it in M_I^A, $\bar{x}'/\bar{y}'$ is split into $\bar{v}/\bar{v}'$ and $\bar{x}/\bar{y}$ such that $\bar{x}'/\bar{y}' = \bar{v}\bar{x}/\bar{v}'\bar{y}$ where $\bar{v}/\bar{v}'$ is the maximum length prefix of $\bar{x}'\bar{y}'$ in $V'' = \{\epsilon, b/0\}$. Next, for $i \in \{1, 2, 3\}$, the column $R^+_{s_i}$ represents the sequences in $R^+(M^A, s_i, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$. The subsequent column NV then contains all states of M_I^A not visited by any sequence in $R^+_{s_1}$, $R^+_{s_2}$ or $R^+_{s_3}$. Thus, for the sequences of the second and third iteration, if NV contains t_3, then the response set of t_3 to Ω is not observed along the sequence and the V'' sequences, and therefore, the second term in the calculation of the lower bound is 1. Finally, LB contains the lower bound calculated for the IO sequence. All sequences that do not exceed $m = 3$ with their value for LB (i.e., all sequences that cannot be removed and must be extended) are highlighted in grey.

$\bar{x}'$	$\bar{y}'$	$\bar{v}/\bar{v}'$	$\bar{x}/\bar{y}$	$R^+_{s_1}$	$R^+_{s_2}$	$R^+_{s_3}$	NV	LB
ϵ	ϵ	ϵ	ϵ	ϵ	$b/0$	–	t_3	2
b	0	$b/0$	ϵ	ϵ	$b/0$	–	t_3	2
a	1	ϵ	$a/1$	$\epsilon, a/1$	$b/0$	–	t_3	4
ba	00	$b/0$	$a/0$	ϵ	$b/0$	$ba/00$	–	3
	02	$b/0$	$a/2$	ϵ	$b/0, ba/02$	–	t_3	4
bb	01	$b/0$	$b/1$	$\epsilon, bb/01$	$b/0$	–	t_3	4
baa	001	$b/0$	$aa/01$	ϵ	$b/0$	$ba/00, baa/001$	–	4
	020	$b/0$	$aa/20$	ϵ	$b/0, ba/02$	$baa/020$	–	4
	022	$b/0$	$aa/22$	ϵ	$b/0, ba/02, baa/022$	–	t_3	5
bab	002	$b/0$	$ab/02$	ϵ	$b/0$	$ba/00, bab/002$	–	4
	021	$b/0$	$ab/21$	$\epsilon, bab/021$	$b/0, ba/02$	–	t_3	5

Table 5.1: Lower bounds for the sequences of the example

The test suite generated by the adaptive state counting algorithm for the above input thus is $\mathcal{T}_3.\Omega$, which is essentially equivalent to the following:

$$\{\epsilon, a, b, ba, bb, baa, bab\}.\{a, b\}$$

For comparison, the test suite generated by the non-adaptive algorithm, given in Section 5.2, is the following:

$$\{\epsilon, a, b, aa, ab, ba, bb, baa, bab, bba, bbb\}.\{a, b\}$$

The adaptive algorithm did not extend a and ba in the second iteration, which causes the non-adaptive test suite to contain the following sequences not applied by the adaptive algorithm:

$$\{aa, ab, bba, bbb\}.\{a, b\}$$

This exemplifies that the test suites generated by the adaptive algorithm are contained in those generated by the non-adaptive algorithm for the same machines, which has also been proven by Hierons in [**hierons**].

As described for a similar example in [**hierons**], the above also illustrates a remaining weakness of the adaptive algorithm due to repeatedly executing the same sequences: When applying $V.\Omega$ in the first iteration, especially ba and bb are applied. Both are contained in $V.X$ and hence are prefixes of sequences in

$(V.X).\Omega$. Thus, they are applied again during the second iteration. However, it is not always possible to skip the application of such sequences, as it might not be known whether they are prefixes of sequences to be applied in later iterations. To alleviate this weakness, Hierons in [**hierons**] suggests the use of heuristics, for example by applying selected maximal sequences generated via (non-adaptive) state counting. Note that this problem might become more apparent in the program version of the algorithm in the subsequent section, as there the application of Ω is less obscured than in the formulaic representation.

The above example also exhibits two more features that might be used to further limit the number of sequences applied. First, in the second iteration the sequence bb is followed with both a and b. However, the sequence bb in M^A visits only s_1 and s_2 and hence following it by a alone would suffice to r-distinguish the reached state s_1 from s_2 which is reached by the prefix b of bb.

Second, for the sequence ba applied in the second iteration, to which the responses 00 and 02 are observed in M_I^A, only $ba/00$ does not exceed m with its lower bound. Thus, for any extension of ba considered in later iterations, the lower bound trivially exceeds m if 02 is a prefix of the observed response of M_I^A. Therefore, when, for example, applying $baa \in V.X.X$ in the following iteration, the application might stop after having applied ba as soon as the response 02 was observed, and would only need to apply baa fully when observing responses with prefix 00.

To exploit one or both of the above observations, several modifications of the algorithm would be necessary. This results mainly from the fact that, if some input sequence is not followed by all tests in Ω or is not fully executed for every response, then the second term of the lower bound (i.e., the number of known response sets not observed along an IO sequence) requires possibly substantial changes to remain valid. I provide sketches of such modifications when discussing possible future improvements of the algorithm in Section 8.1.

5.5.2 Properties of the Generated Sets

I follow this example by providing some further observations and details on the properties of the generated sets and some relationships between them.

First, with each iteration, the maximum length of the sequences considered grows by 1, and all added sequences are extensions of already contained sequences.

Lemma 5.5.2. *For all $i > 1$, the formula $\mathcal{T}_i \setminus \mathcal{T}_{i-1} \subseteq \mathcal{T}_{i-1}.X \setminus \mathcal{T}_{i-1}$ holds.*

Proof. The result follows from $\mathcal{T}_i \setminus \mathcal{T}_{i-1} \subseteq \mathcal{C}_i \subseteq \mathcal{C}_{i-1}.X \subseteq \mathcal{T}_{i-1}.X$. □

According to the definition, $\mathcal{C}_i \cap \mathcal{T}_{i-1} = \emptyset$ and $\mathcal{C}_{i-1} \subseteq \mathcal{T}_{i-1}$ and hence $\mathcal{C}_i \cap \mathcal{C}_{i-1} = \emptyset$ hold. Therefore, an analogous result follows for the $\mathcal{C}$ sets.

Corollary 5.5.3. *For all $i > 1$, it holds that $\mathcal{C}_i \subseteq \mathcal{C}_{i-1}.X \setminus C_{i-1}$.*

Thus, for any $i > 1$, the set $\mathcal{C}_i$ is a subset of $V.X^{i-1}$, as $\mathcal{C}_1 = V$ and in each iteration the sequences are extended by only a single element of X.

Furthermore, by construction the (nonempty) $\mathcal{C}$ sets of different iterations are disjoint.

Lemma 5.5.4. *Suppose that $i, j \geq 1$ with $i \neq j$. Then $\mathcal{C}_j \cap \mathcal{C}_i = \emptyset$ holds.*

Proof. Without loss of generality, assume that $1 \leq j < i$. Then, by definition, $\mathcal{C}_i \cap \mathcal{T}_{i-1} = \emptyset$ and also $\mathcal{T}_{i-1} = \bigcup_{k=1}^{i-1} C_k$. The result follows. □

Since for all $i \in \mathbb{N}$ it holds that $\mathcal{R}_i \subseteq \mathcal{C}_i$, the analogous result holds for the $\mathcal{R}$ sets. More importantly, an analogous result then also holds for the $\mathcal{E}$ sets, which implies that no sequence is ever considered for extension in two different iterations.

The final two results in this section describe how containment in some $\mathcal{T}$ set depends on prefixes. First, I show that for each sequence in some $\mathcal{T}_i$ that is an extension of some $\bar{v} \in V$, all prefixes of this sequence that are extensions of $\bar{v}$ are also in $\mathcal{T}_i$. I begin by proving the result for the maximum length proper prefix.

Lemma 5.5.5. *For each sequence $\bar{x}_1 = \bar{v}\bar{x}x$ contained in $\mathcal{T}_i$ such that $\bar{v}$ is the maximum prefix of $\bar{x}_1$ in V, the prefix $\bar{v}\bar{x}$ is also contained in $\mathcal{T}_i$.*

Proof. Because of $\bar{v}\bar{x}x \in \mathcal{T}_i$, there must exist some $k \leq i$ such that $\bar{v}\bar{x}x \in C_k$. Here, $k > 1$ holds, as $\mathcal{C}_1 = V$. Then, from $\bar{v}\bar{x}x \in \mathcal{C}_k = ((\mathcal{C}_{k-1} \setminus \mathcal{R}_{k-1}).X) \setminus \mathcal{T}_{k-1}$, it follows that $\bar{v}\bar{x} \in \mathcal{C}_{k-1} \subseteq \mathcal{T}_{k-1} \subseteq \mathcal{T}_i$. □

By repeated application of this Lemma, the inclusion relation stated in it can be expanded to all prefixes.

Corollary 5.5.6. *Let $\bar{x}_1 = \bar{v}\bar{x} \in \mathcal{T}_i$ be a sequence such that $\bar{v}$ is the maximum prefix of $\bar{x}_1$ in V. Then, for every prefix $\bar{x}'$ of $\bar{x}$, the sequence $\bar{v}\bar{x}'$ is contained in $\mathcal{T}_i$.*

Finally, a sequence $\bar{x}$ is not contained in $\mathcal{T}_i$ if and only if either a prefix of it has been removed or a prefix of it is contained in $\mathcal{C}_i \setminus \mathcal{R}_i$. I begin by proving the forward direction:

Lemma 5.5.7. *Let $\bar{x}_1 = \bar{v}\bar{x} \notin \mathcal{T}_i$ be some sequence and let $\bar{v}$ be the maximum prefix of $\bar{x}_1$ in V. Then, exactly one of the following holds:*

1. *There exists some $\bar{x}_r \in pref(\bar{x})$ and $j \leq i$ such that $\bar{v}\bar{x}_r \in \mathcal{R}_j$.*
2. *There exists some $\bar{x}_c \in pref(\bar{x})$ such that $\bar{v}\bar{x}_c \in \mathcal{C}_i \setminus \mathcal{R}_i$.*

Proof. First, I show that at least one of the conditions (1) and (2) must hold, using a proof by contradiction: Assume that no $\bar{x}_r$ satisfying (1) and no $\bar{x}_c$ satisfying (2) exist. By definition, $\bar{v} \in \mathcal{C}_1$, and thus, as by the assumption no prefix $\bar{v}\bar{x}'$ of $\bar{x}_1$ is ever contained in some $\mathcal{R}_k$ for $k \leq i$, each prefix $\bar{v}\bar{x}'' \in \mathcal{T}_i$ of $\bar{x}_1$ is extended. That is, for all $1 \leq k \leq i$ it holds that $\{\bar{v}\bar{x}^{k-1}\}.X \subseteq \mathcal{E}_k$, where $\bar{x}^k$ is the prefix of $\bar{x}$ of length k. Hence, $|\bar{x}| \geq i$ and $\bar{v}\bar{x}^{(i-1)} \in \mathcal{C}_i \subseteq \mathcal{T}_i$. By the assumption, in particular also $\bar{v}\bar{x}^{(i-1)} \notin \mathcal{R}_i$ holds. Then, $\bar{v}\bar{x}^{(i-1)} \in \mathcal{C}_i \setminus \mathcal{R}_i$ follows, providing a contradiction.

Next, suppose that there exist both some $\bar{x}_r \in pref(\bar{x})$ and $0 < j \leq i$ satisfying (1), and some $\bar{x}_c \in pref(\bar{x})$ satisfying (2). By the same argument as above, the prefix of $\bar{x}_1$ contained in $\mathcal{C}_i$ is created by $(i-1)$ single element extensions from $\bar{v}$. By Corollary 5.5.3 and the maximality of length of $\bar{v}$, no prefix of $\bar{x}_1$ larger than $\bar{v}\bar{x}^{(i-1)}$ can be contained in $\mathcal{T}_i$. Therefore, $\bar{x}^{(i-1)}$ is the only possible choice for $\bar{x}_c$. Then, using Corollary 5.5.6 on $\bar{v}\bar{x}^{(i-1)} \in \mathcal{C}_i \subseteq \mathcal{T}_i$, for $0 < n \leq i$ it holds that $\bar{v}\bar{x}^{(n-1)} \in \mathcal{C}_n$ and $\bar{v}\bar{x}^{(n-1)} \notin \mathcal{R}_n$. Thus, $\bar{x}_r$ must be

longer than $\bar{x}^{(i-1)}$, which, due to the maximality of length of $\bar{v}$, is not possible if $\bar{v}\bar{x}_r$ is also to be contained in some $\mathcal{R}_j$ with $j \leq i$.

It follows that either (1) or (2) must be satisfied. □

Now, the reverse remains to be proven:

Lemma 5.5.8. *For each sequence $\bar{x}_1 = \bar{v}\bar{x}$ with $\bar{v}$ being the maximum prefix of $\bar{x}_1$ in V, it holds that if there exist a proper prefix $\bar{x}'$ of $\bar{x}$ and some $j \leq i$ such that $\bar{v}\bar{x}' \in \mathcal{R}_j$ or $\bar{v}\bar{x}' \in \mathcal{C}_i \setminus \mathcal{R}_i$, then $\bar{x}_1 \notin \mathcal{T}_i$:*

Proof. Case 1) Suppose that $\bar{v}\bar{x}' \in \mathcal{R}_j$. From $\bar{v}\bar{x}' \in \mathcal{R}_j$ it follows that $\bar{v}\bar{x}' \in \mathcal{C}_j$ and $\bar{v}\bar{x}' \notin \mathcal{T}_{j-1}$. Furthermore, $\{\bar{v}\bar{x}'\}.X \not\subseteq \mathcal{C}_{j+1} \subseteq \mathcal{E}_j$. Then, as no sequence is extended twice, $\{\bar{v}\bar{x}'\}.X \not\subseteq \mathcal{T}_i$. Thus, not all prefixes of $\bar{x}_1$ that extend $\bar{v}$ are contained in $\mathcal{T}_i$. Suppose now that $\bar{x}_1 \in \mathcal{T}_i$. Then, by Corollary 5.5.6 all prefixes of $\bar{x}_1$ that are extensions of $\bar{v}$ are required to be in $\mathcal{T}_i$. As this is not the case, $\bar{x}_1 \notin \mathcal{T}_1$ must hold.

Case 2) Suppose that $\bar{v}\bar{x}' \in \mathcal{C}_i \setminus \mathcal{R}_i$. As $\bar{v}$ is the maximum length prefix of $\bar{x}_1$ in V, $\bar{v}\bar{x}'$ has been created by $i-1$ single input extensions to $\bar{v}$ and is thus the maximum length prefix of $\bar{x}_1$ in $\mathcal{T}_i$. Since $\bar{v}\bar{x}'$ is a proper prefix of $\bar{x}_1$, this implies that $\bar{x}_1$ again cannot be contained in $\mathcal{T}_i$. □

Due to $\epsilon \in V$, the above properties hold for all sequences over the input alphabet X.

5.5.3 Final Iteration

As a test suite generation algorithm is practically applicable only if it produces a finite test suite in finite time, I prove in this subsection that the test suite returned by the adaptive state counting algorithm is generated in a finite number of iterations. If this holds, then the test suite is also finite, since, as described in Lemma 5.5.2, it grows in each iteration only by a finite number of input sequences.

The above definition of the sets $\mathcal{T}$ allows for an infinite number of iterations, but only a finite number of iterations is required until the $\mathcal{T}$ sets of all subsequent iterations remain unchanged, as proven in the following Lemma.

Lemma 5.5.9. *There exists some $i \in \mathbb{N}$ such that $\mathcal{T}_i = \mathcal{T}_{i+1}$. The smallest such i is at most $m^2 + 2$.*

Proof. Recall that M_I has at most m states. Suppose that $\bar{x}'$ is an input sequence contained in $\mathcal{C}_{m^2+2}$. By Corollary 5.5.3, $\bar{x}' = \bar{v}\bar{x}$ such that $|\bar{x}| = m^2 + 1$ and $\bar{v}$ is the largest prefix of $\bar{x}'$ in V. Then, for any response $\bar{v}'\bar{y}$ of M_I to this sequence, the IO sequence $\bar{x}/\bar{y}$ applied after $\bar{v}/\bar{v}'$ must visit at least one state of M at least $m+1$ times. Let s be such a state and let $S_1 = \{s\}$ and $V'' \in V'$. Furthermore, let $\bar{v}_1/\bar{v}'_1$ be the largest prefix of $\bar{v}\bar{x}/\bar{v}'\bar{y}$ in V''. Observe that no sequence in V'' can be a prefix of $\bar{v}\bar{x}/\bar{v}'\bar{y}$ larger than $\bar{v}/\bar{v}'$ due to the $\bar{v}$ being of maximal length. Thus, the following holds:

$$LB(M, M_I, \bar{v}_1/\bar{v}'_1, \bar{x}/\bar{y}, \mathcal{T}_{m^2+1}, S_1, \Omega, V'') > m$$

where Ω is not required to distinguish any states, as S_1 is a singleton set. Therefore, $\bar{x}' \in \mathcal{R}_{m^2+2}$ and any sequence in $\mathcal{C}_{m^2+2}$ is also contained in $\mathcal{R}_{m^2+2}$. Hence, $\mathcal{C}_{m^2+3} = \mathcal{E}_{m^2+2} = \emptyset$ and $\mathcal{T}_{m^2+2} = \mathcal{T}_{m^2+3}$. □

By definition, $\mathcal{T}_i = \mathcal{T}_{i+1}$ implies $\mathcal{C}_{i+1} = \emptyset$ and thus also $\mathcal{C}_j = \emptyset$ for all $j \geq i+1$. That is, the result $\mathcal{T}_i$ does not change for any iterations performed after iteration i. The smallest such i is then called the *final iteration* of the adaptive state counting algorithm and the set $\mathcal{T}_i$ the *result* of the algorithm. Since the calculation of $\mathcal{T}_i$ is based on some adaptive characterization set Ω of M and applies Ω to sequences in $\mathcal{T}_i$, the set of tests actually applied to M_I is then $\mathcal{T}_i.\Omega$.

I conclude this section by revisiting Lemma 5.5.7 and limiting it to the final iteration $\mathcal{T}_i = \mathcal{T}_{i+1}$, as for this iteration it holds that $\mathcal{C}_i \setminus \mathcal{R}_i \subseteq \mathcal{C}_i = \emptyset$.

Corollary 5.5.10. *For each sequence $\bar{x}_1 = \bar{v}\bar{x}$ with $\bar{v}$ being the maximum prefix of $\bar{x}_1$ in V, it holds that if $\bar{x}_1 \notin \mathcal{T}_i = \mathcal{T}_{i+1}$, then there must exist some proper prefix $\bar{x}'$ of $\bar{x}$ and some $j \leq i$ such that $\bar{v}\bar{x}' \in \mathcal{R}_j$.*

5.6 An Adaptive State Counting Algorithm

In this section, I present a simple implementation of the adaptive state counting algorithm described in [**hierons**] in the form of a program following the syntax given in Chapter 4.

5.6.1 Helper Functions

To collect the functions used in the algorithm for easier reference, I first provide a recap of the relevant functions defined in preceding chapters. Let $M = (S, s_1, X, Y, h)$, $M_I = (T, t_1, X, Y, h_I)$, $s, s' \in S$, $S_1 \subseteq S$, $x \in X$, $\bar{x} \in X^*$, $y \in Y$, $\bar{y} \in Y^*$, $V, \mathcal{T} \subseteq X^*$, $V'' \in V'$, $\Omega \subseteq \Upsilon(X,Y)$ and let $f : Y \to \Upsilon(X,Y)$ be a partial function.

First are the functions based on some transition relation h:

$$h(s,\bar{x}) := \begin{cases} \{(s,\epsilon)\} & \text{if } \bar{x} = \epsilon \\ \{(s',y) \mid (s,x,s',y) \in h\} & \text{if } \bar{x} = x \\ \{(s'',\bar{y}'y) \mid (s',\bar{y}') \in h(s,\bar{x}') \wedge (s'',y) \in h(s',x)\} & \text{if } \bar{x} = \bar{x}'x \end{cases}$$

$$h^{\bar{y}}(s,\bar{x}) := \{s' \mid (s',\bar{y}) \in h(s,\bar{x})\}$$

$$h^{reach}(s,\bar{x}) := \begin{cases} \{s\} & \text{if } \bar{x} = \epsilon \\ \{s \mid \exists \bar{y} \in Y^* : h^{\bar{y}}(s,\bar{x}) \neq \emptyset\} & \text{otherwise} \end{cases}$$

$$h^{out}(s,\bar{x}) := \begin{cases} \{\epsilon\} & \text{if } \bar{x} = \epsilon \\ \{\bar{y} \mid h^{\bar{y}}(s,\bar{x}) \neq \emptyset\} & \text{otherwise} \end{cases}$$

Figure 5.4: Functions on the transition relation

Next are the functions for applying adaptive test cases and observing response sets:

$$
\begin{aligned}
IO(M,s,(x,f)) &:= \Big(\bigcup_{y \in h^{out}(s,x) \wedge y \notin Dom(f)} \{x/y\} \Big) \\
&\quad \cup \Big(\bigcup_{y \in h^{out}(s,x) \wedge y \in Dom(f)} \{x/y\}.IO(M, h^y(s,x), f(y)) \Big) \\
IO(M,s,null) &:= \{\epsilon\} \\
IO(M,s,\Omega) &:= \bigcup_{\bar{\sigma} \in \Omega} IO(M,s,\bar{\sigma}) \\
B(M_I,\Omega,\bar{x}/\bar{y}) &:= IO(M_I, h_I^{\bar{y}}(s,\bar{x}),\Omega) \\
D(M_I,\Omega,\mathcal{T}) &:= \{B(M_I,\Omega,\bar{x}/\bar{y}) \mid \bar{x} \in \mathcal{T} \wedge \bar{x}/\bar{y} \in L(M_I)\}
\end{aligned}
$$

Figure 5.5: Functions for adaptive test cases

Finally, the following functions are used to count states and to calculate the lower bound:

$$
\begin{aligned}
&mcp(\bar{x},V) = \bar{x}' \Leftrightarrow \\
&\qquad \bar{x}' \in pref(\bar{x}) \cap V \wedge \forall \bar{x}'' \in pref(\bar{x}) \cap V : |\bar{x}'| \geq |\bar{x}''| \\
&R(M,s,\bar{v}/\bar{v}',\bar{x}/\bar{y}) := \\
&\qquad \{\bar{v}\bar{x}'/\bar{v}'\bar{y}' \mid \bar{x}'/\bar{y}' \in pref(\bar{x}/\bar{y}) \setminus \{\epsilon\} \wedge s = h^{\bar{v}'\bar{y}}(s_1, \bar{v}\bar{x})\} \\
&Perm(V, M_I) := \\
&\qquad \big\{ \{\bar{v}_1/\bar{v}_1, \ldots, \bar{v}_n/\bar{v}_n'\} \;\big|\; \forall 1 \leq i \leq n : \bar{v}_1' \in h_I^{out}(s,\bar{v}) \big\} \\
&R^+(M,s,\bar{v}/\bar{v}',\bar{x}/\bar{y},V'') := \\
&\qquad R(M,s,\bar{v}/\bar{v}',\bar{x}/\bar{y}) \cup \{\bar{v}_1/\bar{v}_1' \in V'' \mid \{s'\} = h^{\bar{v}_1'}(s_1,\bar{v}_1)\} \\
&LB(M,M_I,\bar{v}/\bar{v}',\bar{x}/\bar{y},\mathcal{T},S_1,\Omega,V'') := \\
&\qquad \sum_{s \in S_1} \Big| R^+(M,s,\bar{v}/\bar{v}',\bar{x}/\bar{y},V'') \Big| + \\
&\qquad \Big| D(M_I,\Omega,\mathcal{T}) \setminus \big(\bigcup_{\substack{s' \in S_1, \\ \bar{x}_1/\bar{y}_1 \in R^+(M,s',\bar{v}/\bar{v}',\bar{x}/\bar{y},V'')}} B(M_I,\Omega,\bar{x}_1/\bar{y}_1) \big) \Big|
\end{aligned}
$$

Figure 5.6: Functions for state counting

To these, I add a new helper function *noFailure* that takes two sets $\mathcal{O}_I, \mathcal{O} \subseteq (X/Y)^*$ of IO sequences as inputs and returns whether $\mathcal{O}_I$ is a subset of $\mathcal{O}$:

$$
noFailure(\mathcal{O}_I, \mathcal{O}) = \begin{cases} true & \text{if } \mathcal{O}_I \subseteq \mathcal{O} \\ false & \text{otherwise} \end{cases}
$$

In testing, the sets $\mathcal{O}_I$ and $\mathcal{O}$ contain the reactions (in the form of IO sequences) of M_I and M, respectively, to some set of input sequences. Then, if $\mathcal{O}_I \not\subseteq \mathcal{O}$ holds, a failure must have been observed.

5.6.2 Implementation

Using these functions, I implement the adaptive state counting algorithm as given in Algorithm 5.1.

Algorithm 5.1 Adaptive state counting algorithm

1: **function** *performAdaptiveStateCounting*(
  in $M = (S, s_1, X, Y, h) : FSM$, ▷ Specification
  in $M_I = (T, t_1, X, Y, h_I) : FSM$, ▷ SUT
  in $V : \mathbb{P}(X^*)$, ▷ Deterministic state cover of M
  in $\Omega : \mathbb{P}(\Upsilon(X, Y))$, ▷ Adaptive characterizing set of M
  in $m : \mathbb{N}$, ▷ Upper bound on $|T|$
) : $Bool \times \mathbb{P}(X^*)$ ▷ Conformance result and test suite
2: $\mathcal{T}, \mathcal{C}, \mathcal{R} : \mathbb{P}(X^*)$;
3: $\mathcal{T} \leftarrow V$; ▷ Sequences followed by Ω in this or previous iterations
4: $\mathcal{C} \leftarrow V$; ▷ Elements of $\mathcal{T}$ considered for extension in this iteration
5: $\mathcal{R} \leftarrow \emptyset$; ▷ Sequences of $\mathcal{C}$ to be removed
6: $\mathcal{O}, \mathcal{O}_I : \mathbb{P}((X/Y)^*)$; ▷ Observed responses of M, M_I to $\mathcal{T}.\Omega$
7: $\mathcal{O} \leftarrow \bigcup_{\bar{x} \in \mathcal{C}} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\}$;
8: $\mathcal{O}_I \leftarrow \bigcup_{\bar{x} \in \mathcal{C}} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x}) \right\}$;
9: $V' : \mathbb{P}(\mathbb{P}((X/Y)^*))$; ▷ Possible responses of the SUT to V
10: $V' \leftarrow Perm(V, M_I)$;
11: **while** $\mathcal{C} \neq \emptyset \wedge \mathcal{O}_I \subseteq \mathcal{O}$ **do**
12: $\mathcal{R} \leftarrow \{\bar{x}' \in \mathcal{C} \mid$
$\forall \bar{y}' \in h_I^{out}(t_1, \bar{x}') :$
$\exists S_1 \subseteq S, V'' \in V', \bar{v}/\bar{v}' \in V'', \bar{x}/\bar{y} \in (X/Y)^* :$
$\bar{v}\bar{x}/\bar{v}'\bar{y} = \bar{x}'/\bar{y}'$
$\wedge\ \bar{v}/\bar{v}' = mcp(\bar{x}'/\bar{y}', V'')$
$\wedge\ \forall s_1, s_2 \in S_1, s_1 \neq s_2 :$
$\forall \bar{x}_1/\bar{y}_1 \in R^+(M, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') :$
$\forall \bar{x}_2/\bar{y}_2 \in R^+(M, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') :$
$B(M_I, \Omega, \bar{x}_1/\bar{y}_1) \neq B(M_I, \Omega, \bar{x}_2/\bar{y}_2)$
$\wedge\ LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}, S_1, \Omega, V'') > m\}$;
13: $\mathcal{C} \leftarrow ((\mathcal{C} \setminus \mathcal{R}).X) \setminus \mathcal{T}$;
14: $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{C}$;
15: $\mathcal{O} \leftarrow \mathcal{O} \cup \bigcup_{\bar{x} \in \mathcal{C}} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\}$;
16: $\mathcal{O}_I \leftarrow \mathcal{O}_I \cup \bigcup_{\bar{x} \in \mathcal{C}} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x}) \right\}$
17: **end while**
18: **return** $(noFailure(\mathcal{O}_I, \mathcal{O}), \mathcal{T})$
19: **end function**

This algorithm then represents the calculation of the $\mathcal{T}$, $\mathcal{C}$ and $\mathcal{R}$ sets described in the previous section as follows:

- Lines 3 to 5 initialize the $\mathcal{T}$ and $\mathcal{C}$ sets to their values for the first iteration. Furthermore, $\mathcal{R}$ is set to $\mathcal{R}_0 = \emptyset$.
- In line 6, the sets $\mathcal{O}$ and $\mathcal{O}_I$ are introduced, which are used to contain the IO sequences observed while applying $\mathcal{T}.\Omega$ to M and M_I, respectively.

- Lines 7 and 8 then initialize these sets for the first iteration (i.e., assign to them the observations of the application of $V.\Omega$).
- Lines 9 and 10 introduce and calculate V'.
- Line 11 then begins a **while** loop that finalizes the current iteration and begins the next. Here, a new iteration is entered as long as $\mathcal{C} \neq \emptyset \wedge \mathcal{O}_I \subseteq \mathcal{O}$ holds. Therefore, the loop is exited only if no sequences are left to extend, which indicates that the final iteration has been performed, or if a failure has been observed while applying $\mathcal{T}.\Omega$, that is, if there exists some reaction $\bar{x}/\bar{y} \in \mathcal{O}_I \setminus \mathcal{O}$, where, by construction, $\mathcal{O} \subseteq L(M)$ and $\mathcal{O}_I \subseteq L(M_I)$.

  Note that terminating the loop as soon as a failure has been observed constitutes another application of adaptivity, as after observing a failure it is no longer necessary to extend the test suite.
- Line 12 calculates the set $\mathcal{R}$ of sequences of $\mathcal{C}$ not to be extended. Here, in iteration i of the loop, the same criteria for removal as in the calculation of $\mathcal{R}_i$ are used.
- Lines 13 and 14 update $\mathcal{C}$ and $\mathcal{T}$ based on the new value of $\mathcal{R}$ analogously to the calculation of $\mathcal{C}_{i+1}$ and $\mathcal{T}_{i+1}$ from some $\mathcal{C}_i$, $\mathcal{T}_i$ and $\mathcal{R}_i$.
- Lines 15 and 16 apply $\mathcal{C}.\Omega$ to M and M_I, respectively, and accordingly update the sets of observations $\mathcal{O}$ and $\mathcal{O}_I$.
- Finally, if the **while** loop has been exited, then by line 18 the algorithm returns a pair of two values: First the truth value of $\mathcal{O}_I \subseteq \mathcal{O}$, which indicates whether a failure has been observed, and second the set $\mathcal{T}$ of sequences which have been followed by Ω during execution.

I omit the proof that if the algorithm returns $(\mathit{true}, \mathcal{T})$, then $\mathcal{T} = \mathcal{T}_i = \mathcal{T}_{i+1}$ for some $i \in \mathbb{N}$, that is, that the algorithm calculates exactly the result described in the previous section, as long as it did not observe a failure during execution. The analogous result for the modified algorithm is proven in Section 7.5.

Chapter 6

Incorrectness of the Original Algorithm

Let $M = (S, s_1, X, Y, h)$ be some FSM representing a specification with $V \subseteq X^*$ and $\Omega \subseteq \Upsilon(X, Y)$ respectively being a deterministic state cover and an adaptive characterizing set of M. Furthermore, let $M_I = (T, t_1, X, Y, h_I)$ be an FSM with at most $m \in \mathbb{N}$ states that represents the behaviour of the SUT. Suppose that

$$\mathit{performAdaptiveStateCounting}(M, M_I, V, \Omega, m) = (\mathit{true}, \mathcal{T})$$

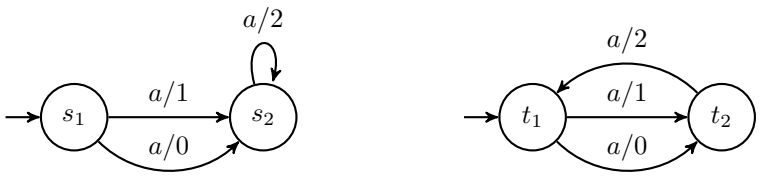
That is, according to the algorithm $M_I \preceq M$ holds. As only tests in $\mathcal{T}.\Omega$ have been applied to both M and M_I, this result can only be correct if the following holds:

$$IO(M_I, t_1, \mathcal{T}.\Omega) \subseteq IO(M, s_1, \mathcal{T}.\Omega) \implies L(M_I) \subseteq L(M)$$

In this chapter, I prove that the test suite generated by the adaptive state counting algorithm is not always sufficient to find failures when testing whether M_I is the reduction of M. In other words, I show that the above implication does not hold. To prove this claim, I provide specific FSMs M^F and M_I^F such that $L(M_I^F) \not\subseteq L(M^F)$, whereas the result of the algorithm implies the opposite.

6.1 Counterexample

Using input alphabet $X = \{a\}$ and output alphabet $Y = \{0, 1, 2\}$, let M^F and M_I^F be defined by the graphical representations presented in Figure 6.1.



(a) Specification FSM M^F (b) SUT FSM M_I^F

Figure 6.1: Counterexample FSMs

For these machines, $L(M_I^F) \not\subseteq L(M^F)$ holds, as, for example, $aaa/020 \in L(M_I^F) \setminus L(M^F)$. This is also evident from the product machine $P(M^F, M_I^F)$, illustrated in Figure 6.2, where any input sequence of length at least 3 reaches the state $Fail$.

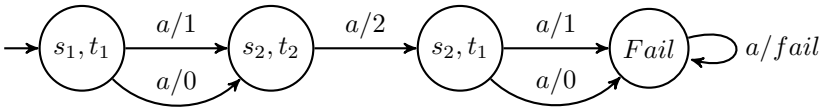


Figure 6.2: Product machine $P(M^F, M_I^F)$

Examining this product machine, it also follows that no input sequence of length less than 3 can reach a failure.

Let $V = \{\epsilon, a\}$. Then, V is a deterministic state cover of M^F, as s_1 is d-reached via ϵ and s_2 via a. Since M_I^F can respond to the input sequence a with either 0 or 1, it follows that $V' = \{V_0'', V_1''\}$ with

$$V_0'' := \{\epsilon, a/0\}$$
$$V_1'' := \{\epsilon, a/1\}$$

That is, for $i \in \{0, 1\}$, the set V_i'' denotes the permutation of responses of M_I^F to V that contains a/i.

Input a also r-distinguishes s_1 and s_2. Thus, I use $\Omega = \{\bar{\sigma}_a^Y\}$ as an adaptive characterizing set of M^F, where $\bar{\sigma}_a^Y$ is the adaptive test case for a in Y. That is, $\bar{\sigma}_a^Y$ is the adaptive test case represented by Figure 6.3.

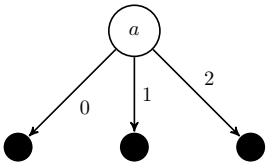


Figure 6.3: Adaptive test case $\bar{\sigma}_a^Y$

Finally, let $m = |T| = |S| = 2$. Having established these parameters, I now calculate the result of $performAdaptiveStateCounting(M^F, M_I^F, V, \Omega, m)$. By definition, the following holds in the first iteration:

$$\mathcal{T} = \mathcal{C} = V = \{\epsilon, a\}$$

Then, for $\mathcal{T}.\Omega = V.\Omega$, both machines behave identically, as no applied sequence is of length at least 3.

Next, the new value of $\mathcal{R}$ is calculated. Since $X = \{a\}$, the sequence $\epsilon \in \mathcal{C}$ can only be extended to a, which is already contained in $\mathcal{T}$ and hence cannot be contained in the next value of $\mathcal{C}$. Thus, it is only necessary to check whether a is removed, that is, whether for both responses of M_I^F to a, those being 0 and 1, suitable $S_1 \subseteq S$ and $V'' \in V'$ can be found to satisfy the criteria for removal.

I begin by examining response 0 and choose $S_1 = S$ and $V'' = V_1''$. Then, the longest prefix of $a/0$ in V_1'' is ϵ. Thus, the following sets of sequences need

to be distinguished via Ω:

$$R^+(M^F, s_1, \epsilon, a/0, V_1'') = R(M^F, s_1, \epsilon, a/0) \cup \{\epsilon\} \quad = \{\epsilon\}$$
$$R^+(M^F, s_2, \epsilon, a/0, V_1'') = R(M^F, s_2, \epsilon, a/0) \cup \{a/1\} = \{a/0, a/1\}$$

These are then distinguished by Ω due to the following behaviour:

$$B(M_I^F, \Omega, \epsilon) \qquad\qquad = \{a/0, a/1\}$$
$$B(M_I^F, \Omega, a/0) = B(M_I^F, \Omega, a/1) = \{a/2\}$$

Finally, the lower bound is calculated as follows:

$$\begin{aligned} LB(M^F, M_I^F, \epsilon, a/0, \mathcal{T}, S_1, \Omega, V_1'') &\geq \sum_{s \in S_1} \left| R^+(M^F, s, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V_1'') \right| \\ &= |R^+(M^F, s_1, \epsilon, a/0, V_1'')| \\ &\quad + |R^+(M^F, s_2, \epsilon, a/0, V_1'')| \\ &= 3 \end{aligned}$$

Similarly, for response 1, I again select $S_1 = S$ but choose $V'' = V_0''$. Thus, the longest prefix of $a/1$ in V_0'' is again ϵ. By reasoning analogous to that for response 0, the following holds:

$$LB(M^F, M_I^F, \epsilon, a/1, \mathcal{T}, S_1, \Omega, V_0'') \geq 3$$

Therefore, for each response of M_I to a, the removal criteria can be met and hence $a \in \mathcal{R}$. Then, in turn, the next value of $\mathcal{C}$ is $\emptyset$, and $\mathcal{T}$ remains unchanged, which causes the **while** loop to exit with $\mathcal{T}.\Omega = V.\Omega$ still being the only tests applied. Since for these tests no failure is observed, the algorithm returns *true* as its first return value, and $\mathcal{T}$ as the second, having essentially only applied the sequences $V.\{a\} = \{a, aa\}$.

Thus, the generated test suite and the tests applied are insufficient to show that $L(M_I^F) \not\subseteq L(M^F)$.

6.2 Analysis of the Observed Error

Recall that, by intention, the $\mathcal{R}$ sets should only contain sequences that do not form prefixes of minimal sequences to failures extending V. Furthermore, aaa is a minimal sequence to a failure extending V. More specifically, it extends $a \in V$ by aa. Hence, as $a \in pref(aaa)$, the input sequence a is a prefix of a minimal sequence to a failure extending V and should therefore not be removed.

Subsequently, as $a \in V = \{\epsilon, a\}$, the input sequence a can only be a prefix of sequences of minimal length extending V of the form $\bar{x} = a\bar{x}'$, where a is the maximum length prefix of $\bar{x}$ in V. However, for each of the two responses of M_I^F observed for a, it is possible to select a $V'' \in V'$ containing the respective other response to a, such that ϵ is in each case the maximum length prefix in V'':

$$mcp(a/0, \{\epsilon, a/1\}) = \epsilon \qquad \text{(choosing } V_1'' \text{ for } a/0)$$
$$mcp(a/1, \{\epsilon, a/0\}) = \epsilon \qquad \text{(choosing } V_0'' \text{ for } a/1)$$

It is only for these selections of V'' that the lower bound exceeds m, as for all $S_1 \subseteq S$ the following holds:

$$LB(M^F, M_I^F, a/0, \epsilon, \mathcal{T}, S_1, \Omega, V_0'') = 2$$
$$LB(M^F, M_I^F, a/1, \epsilon, \mathcal{T}, S_1, \Omega, V_1'') = 2$$

Generally speaking, there exist instances where, for an input sequence $\bar{x}$ and a response $\bar{y}$ of M_I^F to it, the algorithm has to consider some $V'' \in V'$ such that for $\bar{x}/\bar{y} = \bar{v}\bar{x}'/\bar{v}'\bar{y}'$ it holds that $\bar{v}/\bar{v}'$ is the maximum length prefix of $\bar{x}/\bar{y}$ in V'', but $\bar{v}$ is not the maximum length prefix of $\bar{x}$ in V. Then, as shown by the counterexample above, if the lower bound calculated for $\bar{x}/\bar{y}$ and such V'' exceeds m, this is not sufficient to prove that $\bar{x}/\bar{y}$ cannot form a prefix to a minimum length sequence to a failure extending V.

In the next chapter, I present a modification of the adaptive state counting algorithm that is based on checking the lower bound for some $\bar{x}/\bar{y}$ only against a subset of V' that depends on $\bar{x}/\bar{y}$, avoiding such choices of V'' as in the above counterexample.

Chapter 7

Refined Adaptive State Counting

This chapter provides the main contribution of this work, a refinement of the adaptive state counting algorithm with an accompanying proof that the test suite it generates is sufficient when testing for reduction. Furthermore, I give an algorithmic representation and prove that this algorithm behaves as desired.

To accomplish this, I use the same objects as in previous chapters: Let $M = (S, s_1, X, Y, h)$ be the reference model, and let $M_I = (T, t_1, X, Y, h_I)$ represent the SUT. Additionally, suppose that M_I has at most $m \in \mathbb{N}$ states for some $m \geq |S|$. Finally, let V and Ω be a deterministic state cover and an adaptive characterizing set of M, respectively.

7.1 Narrowing the Choice of Permutations

In Section 6.2, I have illustrated how the adaptive state counting algorithm stops extending some sequences too early because it computes an invalid lower bound for some special $V'' \in V'$. More specifically, some IO sequences $\bar{x}/\bar{y}$ are checked against $V'' \in V'$ such that the longest prefix of $\bar{x}$ in V is some $\bar{v}_1$, but the longest prefix of $\bar{x}/\bar{y}$ in V'' has an input portion $\bar{v}_2 \neq \bar{v}_1$. Then, by definition, also $|\bar{v}_2| < |\bar{v}_1|$ holds.

The proposed modification consists of constraining which $V'' \in V'$ are used to calculate lower bounds. As it is desirable to avoid testing against such V'' as above, for some IO sequence I then select only those *relevant* $V'' \in V'$ where the maximum prefixes with respect to V and V'' share the same input portion. This is to be performed by the function N.

Definition 7.1.1. For a sequence $\bar{x}/\bar{y}$ and deterministic state cover V, the function N *narrows* V' to the relevant subset $N(\bar{x}/\bar{y}, V')$ defined as follows:

$$N(\bar{x}/\bar{y}, V') := \Big\{ V'' \in V' \;\Big|\; \exists \bar{v} \in \mathit{pref}(\bar{x}) \;:\; (\forall \bar{w} \in \mathit{pref}(\bar{x}) \;:\; |\bar{w}| > |\bar{v}| \implies \bar{w} \notin V) \quad (1)$$
$$\wedge\, (\exists \bar{v}' \in Y^* \;:\; \bar{v}/\bar{v}' \in V'' \cap \mathit{pref}(\bar{x}/\bar{y})) \Big\} \quad (2)$$

Here, (1) requires $\bar{v}$ to be the maximum length prefix of $\bar{x}$ in V and (2) requires V'' to respond to $\bar{v}$ in the same way as observed for $\bar{x}/\bar{y}$.

To shorten notation, I write $V'_{\bar{x}/\bar{y}}$ to denote the set $N(\bar{x}/\bar{y}, V')$. ⊣

Following from the definition of V' and the fact that $\epsilon \in V$, any such narrowed set is nonempty. Furthermore, the following property follows directly from the definition of $V'_{\bar{x}/\bar{y}}$.

Corollary 7.1.2. *Let $\bar{x}/\bar{y} = \bar{v}\bar{x}'/\bar{v}'\bar{y}' \in L(M) \cap L(M_I)$ be some IO sequence such that $\bar{v}$ is the maximum length prefix of $\bar{x}$ in V. Then, for any $V'' \in V'_{\bar{x}/\bar{y}}$, it holds that V'' contains $\bar{v}/\bar{v}'$ and also $\bar{v}/\bar{v}' = mcp(\bar{x}/\bar{y}, V'')$.*

By checking $\bar{x}/\bar{y}$ for removal only against such V'' as contained in $V'_{\bar{x}/\bar{y}}$, the scenario described in the previous chapter can be avoided.

7.2 Generated Test Suite

Using the concept introduced in the previous section, I define in this section the result $\mathcal{T}^N \subseteq X^*$ (and the corresponding test suite $\mathcal{T}^N.\Omega$) generated by a refined adaptive state counting algorithm. This refined version modifies the original algorithm, as presented in Chapter 5, by employing the technique of narrowing V' when checking for the removal of a sequence.

I use sets $\mathcal{T}_i^N$, $\mathcal{C}_i^N$, $\mathcal{R}_i^N$ and $\mathcal{E}_i^N$ to respectively denote the sets of sequences *tested*, *current*, *removed* and *extended* in some iteration i, distinguishing them from those of the original adaptive state counting algorithm by the superscript N, indicating the use of the function N.

Definition 7.2.1. Let $\mathcal{T}_i^N$, $\mathcal{C}_i^N$, $\mathcal{R}_i^N$ and $\mathcal{E}_i^N$ for some iteration $i \in \mathbb{N}$ with $i > 0$ be given by the following definitions:

$$
\begin{aligned}
\mathcal{R}_0^N &:= \emptyset \\
\mathcal{C}_1^N &:= V \\
\mathcal{T}_1^N &:= V \\
\mathcal{E}_n^N &:= (\mathcal{C}_n^N \setminus \mathcal{R}_n^N).X && \text{for } n \geq 1 \\
\mathcal{C}_{n+1}^N &:= \mathcal{E}_n^N \setminus \mathcal{T}_n^N && \text{for } n \geq 1 \\
\mathcal{T}_{n+1}^N &:= \mathcal{T}_n^N \cup \mathcal{C}_{n+1}^N && \text{for } n \geq 1
\end{aligned}
$$

where R_{n+1} for $n \geq 0$ is defined as follows:

$$\begin{aligned}
\mathcal{R}^N_{n+1} := \Big\{ \bar{x}' \in \mathcal{C}^N_{n+1} \;\Big|\; & \forall \bar{y}' \in h^{out}_I(t_1, \bar{x}') : \exists S_1 \subseteq S, \bar{x}/\bar{y} \in (X/Y)^* : \\
& \exists V'' \in V'_{\bar{x}'/\bar{y}'}, \bar{v}/\bar{v}' \in V'' : && \text{(N)} \\
& \bar{v}\bar{x}/\bar{v}'\bar{y} = \bar{x}'/\bar{y}' && (1) \\
& \wedge\, \bar{v}/\bar{v}' = mcp(\bar{x}'/\bar{y}', V'') && (2) \\
& \wedge\, \forall s_1, s_2 \in S_1, s_1 \neq s_2 : && (3) \\
& \quad \forall \bar{x}_1/\bar{y}_1 \in R^+(M, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\
& \quad\quad \forall \bar{x}_2/\bar{y}_2 \in R^+(M, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\
& \quad\quad\quad B(M_I, \Omega, \bar{x}_1/\bar{y}_1) \neq B(M_I, \Omega, \bar{x}_2/\bar{y}_2) \\
& \wedge\, LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}^N_{n+1}, S_1, \Omega, V'') > m \Big\} && (4)
\end{aligned}$$

Finally, a sequence $\bar{x} \in \mathcal{R}^N_i$ is again said to be *removed* in iteration i, whereas a sequence $\bar{x} \in \mathcal{C}^N_i \setminus \mathcal{R}^N_i$ is said to be *extended* in iteration i. ⊣

That is, the sets calculated for the refined adaptive state counting algorithm differ from those given in Definition 5.5.1 only in the way the removal sets are calculated, while the other sets merely propagate this change. $\mathcal{R}^N_i$ then differs from $\mathcal{R}_i$ by (N), and hence, by requiring that V'' is contained in $V'_{\bar{x}'/\bar{y}'}$ instead of using all $V'' \in V'$.

Following from the difference between both algorithms being localized to the difference between $\mathcal{R}$ and $\mathcal{R}^N$, all results for the relationships between the sets of Definition 5.5.1 described in Section 5.5 can be trivially transferred to hold for the sets of Definition 7.2.1.

7.3 Proof of a Sufficient Criterion for Reduction

In this section, I prove that the result of the refined state counting algorithm is sufficient to test for reduction. More specifically, for some generated test suite $\mathcal{T}^N_i = \mathcal{T}^N_{i+1}$ with $i \in \mathbb{N}$, I show that M_I is a reduction of M if and only if no failure is observed while applying $\mathcal{T}^N_i.\Omega$.

First, I prove that the absence of failures observed while applying $\mathcal{T}^N_i$ implies reduction.

Lemma 7.3.1. *Let $\mathcal{T}^N_i = \mathcal{T}^N_{i+1}$ be the test suite generated by the refined adaptive state counting algorithm for specification FSM M, SUT FSM M_I with at most $m \geq |S|$ states, and a deterministic state cover V and adaptive characterizing set Ω of M. If no failure is observed applying the tests in $\mathcal{T}^N_i.\Omega$, then M_I is a reduction of M. That is,*

$$IO(M_I, t_1, \mathcal{T}^N_i.\Omega) \subseteq IO(M, s_1, \mathcal{T}^N_i.\Omega) \implies M_I \preceq M$$

holds.

Proof. Proof by contradiction: Assume that $M_I \not\preceq M$. Therefore, it is required to show that a failure is observed while applying $\mathcal{T}^N_i.\Omega$.

By the assumption, $L(M_I) \setminus L(M) \neq \emptyset$ and hence, by Lemma 5.4.13, there exists a minimal sequence $\bar{x}_f/\bar{y}_f = \bar{v}\bar{x}_1/\bar{v}'\bar{y}_1$ to a failure extending V such that $\bar{v}$ is the maximum length prefix of $\bar{x}_f$ in V.

Consider the case where a failure is observed when applying the tests in $(\mathcal{T}_i^N \setminus \{\bar{x}_f\}).\Omega$. Then, the failure is trivially also observed when applying $\mathcal{T}_i^N.\Omega$. Thus, for the following, suppose that the opposite holds:

$$IO(M_I, t_1, (\mathcal{T}_i^N \setminus \{\bar{x}_f\}).\Omega) \subseteq IO(M, s_1, (\mathcal{T}_i^N \setminus \{\bar{x}_f\}).\Omega)$$

Next, either $\bar{x}_f \in \mathcal{T}_i^N$ or $\bar{x}_f \notin \mathcal{T}_i^N$ must hold:

Case 1) Assume that $\mathcal{T}_i^N$ contains $\bar{x}_f$. Then, as $\bar{x}_f$ is a sequence to a failure, $h_I^{out}(t_1, \bar{x}_f) \not\subseteq h^{out}(s_1, \bar{x}_f)$ and a failure is already observed when applying $\mathcal{T}_i^N$.

Case 2) Assume that $\bar{x}_f \notin \mathcal{T}_i^N$. Then, by construction of $\mathcal{T}_i^N$, there must exist some proper prefix of $\bar{x}_f$ that has been removed. Using Corollary 5.5.10, there must also exist some $\bar{x}_2\bar{x}_3/\bar{y}_2\bar{y}_3$ such that $\bar{x}_f/\bar{y}_f = \bar{v}\bar{x}_1/\bar{v}'\bar{y}_1 = \bar{v}\bar{x}_2\bar{x}_3/\bar{v}'\bar{y}_2\bar{y}_3$ and both $\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2 \in \mathcal{R}_j^N$ for some $0 < j \leq i$ and $|\bar{x}_3/\bar{y}_3| > 0$ hold.

From $\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2 \in \mathcal{R}_j^N$ it follows that $\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2$ must satisfy all criteria for removal. That is, there must exist some $\bar{x}_r/\bar{y}_r \in (X/Y)^*$, $S_1 \subseteq S$, $V'' \in V'_{\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2}$ and $\bar{v}_r/\bar{v}'_r \in V''$ for which the following holds:

$$\begin{aligned}
&\bar{v}_r\bar{x}_r/\bar{v}'_r\bar{y}_r = \bar{v}\bar{x}_2/\bar{v}'\bar{y}_2 \\
&\wedge\, \bar{v}_r/\bar{v'}_r = mcp(\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2, V'') \\
&\wedge\, \forall s_1, s_2 \in S_1, s_1 \neq s_2 : \\
&\quad \forall \bar{x}_{s_1}/\bar{y}_{s_1} \in R^+(M, s_1, \bar{v}_r/\bar{v}'_r, \bar{x}_r/\bar{y}_r, V'') : \\
&\quad \forall \bar{x}_{s_2}/\bar{y}_{s_2} \in R^+(M, s_2, \bar{v}_r/\bar{v}'_r, \bar{x}_r/\bar{y}_r, V'') : \\
&\quad\quad B(M_I, \Omega, \bar{x}_{s_1}/\bar{y}_{s_1}) \neq B(M_I, \Omega, \bar{x}_{s_2}/\bar{y}_{s_2}) \\
&\wedge\, LB(M, M_I, \bar{v}_r/\bar{v}'_r, \bar{x}_r/\bar{y}_r, \mathcal{T}_j^N, S_1, \Omega, V'') > m
\end{aligned}$$

Here, since $\bar{v}$ is the maximal length prefix of $\bar{x}_f$, by Corollary 7.1.2 it holds that $\bar{v}_r/\bar{v}'_r = \bar{v}/\bar{v}' = mcp(\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2, V'')$ and hence also $\bar{x}_r/\bar{y}_r = \bar{x}_2/\bar{y}_2$.

As the calculated lower bound exceeds $m \geq |T|$, it is invalid. That is, the following holds:

$$\neg Size_m(M, M_I, V'', S_1, \Omega, \mathcal{T}_j^N, \bar{v}/\bar{v}', \bar{x}_2/\bar{y}_2)$$

This enables the application of Lemma 5.4.11, implying that the prerequisites are not met or that a repetition of any kind occurs. Therefore, the following must hold:

$$\begin{aligned}
&\neg Prereq(M, M_I, V, V'', S_1, \Omega, \mathcal{T}_j^N, \bar{v}/\bar{v}', \bar{x}_2/\bar{y}_2) \\
&\vee\, Rep_{Pre}(M, M_I, \bar{v}/\bar{v}', \bar{x}_2/\bar{y}_2) \\
&\vee\, Rep_{Cov}(M, M_I, \bar{v}/\bar{v}', V'', \bar{x}_2/\bar{y}_2)
\end{aligned}$$

Of these, I first show that $Prereq$ must hold. Recall the definition of $Prereq$ as given in Definition 5.4.10. Then, condition (1) is satisfied by $\bar{v}/\bar{v}' = mcp(\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2, V'')$ as illustrated above. Next, condition (2) follows from the complete testing assumption. As it has been assumed that no failure is observed applying tests in $(\mathcal{T}_j^N \setminus \{\bar{x}_f\}).\Omega$ and that $\bar{x}_f \notin \mathcal{T}_j^N$, condition (3) is met. Next, condition (4) is satisfied by Corollary 5.5.6. Finally, condition (5) follows from meeting the removal criteria.

Thus, at least one of Rep_{Pre} and Rep_{Cov} must hold. Assume first that Rep_{Pre} holds. Then, for some prefix $\bar{v}\bar{x}_p/\bar{v}'\bar{y}_p \in \{\bar{v}/\bar{v}'\}.(pref(\bar{x}_2/\bar{y}_2) \setminus \{\epsilon\})$

that reaches state q of $P(M, M_I)$, there must exist some prefix $\bar{v}\bar{x}'_p/\bar{v}'\bar{y}'_p \in \{\bar{v}/\bar{v}'\}.pref(\bar{x}_2/\bar{y}_2)$ with $\bar{x}'_p \neq \bar{x}_p$ that also reaches q. Without loss of generality, let $|\bar{x}'_p| < |\bar{x}_p|$. Let $\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2 = \bar{v}\bar{x}_p\bar{x}_s/\bar{v}'\bar{y}_p\bar{y}_s$. Then, $\bar{v}\bar{x}_p\bar{x}_s\bar{x}_3/\bar{v}'\bar{y}_p\bar{y}_s\bar{y}_3$ and $\bar{v}\bar{x}'_p\bar{x}_s\bar{x}_3/\bar{v}'\bar{y}'_p\bar{y}_s\bar{y}_3$ both reach $Fail$ in $P(M, M_I)$ and hence $\bar{v}\bar{x}'_p\bar{x}_s\bar{x}_3/\bar{v}'\bar{y}'_p\bar{y}_s\bar{y}_3$ is a minimal sequence to a failure extending V. From $|\bar{x}'_p| < |\bar{x}_p|$, it immediately follows that $\bar{x}'_p\bar{x}_s\bar{x}_3/\bar{y}'_p\bar{y}_s\bar{y}_3$ is shorter than $\bar{x}_p\bar{x}_s\bar{x}_3/\bar{y}_p\bar{y}_s\bar{y}_3$, contradicting the assumption that no input shorter than $\bar{x}_1 = \bar{x}_p\bar{x}_s\bar{x}_3$ can be used after some sequence in V to observe a failure.

Therefore, Rep_{Cov} is required to hold. Let the repetition be caused by some $\bar{v}_2/\bar{v}'_2 \in V''$, that is, let there exist some $\bar{x}_p/\bar{y}_p \in pref(\bar{x}_2/\bar{y}_2) \setminus \{\epsilon\}$ such that both $\bar{v}_2/\bar{v}'_2$ and $\bar{v}\bar{x}_p/\bar{v}'\bar{y}_p$ reach the same state q of $P(M, M_I)$. Analogous to the previous case, by extending from $\bar{v}_2/\bar{v}'_2$ instead of extending from $\bar{v}/\bar{v}'$ a sequence to a failure extending V can be created that contradicts the minimality of the length of $\bar{x}_1$.

Following from these results, there can exist no proper prefix of $\bar{x}_f = \bar{v}\bar{x}_1$ extending $\bar{v}$ that has been removed (i.e., that is contained in any $\mathcal{R}_j^N$ for $0 < j \leq i$). This in turn contradicts the assumption that $\bar{x}_f \notin \mathcal{T}_i^N$ and then, as $\bar{x}_f \in \mathcal{T}_i^N$, a failure must be observed while applying $\mathcal{T}_i^N.\Omega$. □

Note that this proof relies on the fact that $\bar{v}_r/\bar{v}'_r = mcp(\bar{v}\bar{x}_2/\bar{v}'\bar{y}_2)$ to provide the contradiction for Rep_{Cov}. If $\bar{v}_r/\bar{v}'_r$ were shorter than $\bar{v}/\bar{v}'$, then there would exist some $\bar{v}_e/\bar{v}_e \neq \epsilon$ such that $\bar{v}/\bar{v}' = \bar{v}_r\bar{v}_e/\bar{v}'_r\bar{v}'_e$. The repetition caused by $\bar{v}/\bar{v}'$ would then not allow for the construction of a shorter sequence to a failure extending V, resulting instead only in $\bar{v}\bar{x}_2\bar{x}_3/\bar{v}\bar{y}_2\bar{y}_3$, which is identical to $\bar{v}_r\bar{v}_e\bar{x}_2\bar{x}_3/\bar{v}'_r\bar{v}'_e\bar{y}_2\bar{y}_3$.

The reverse of the property just proven is given by the following formula:

$$IO(M_I, t_1, \mathcal{T}_i^N.\Omega) \subseteq IO(M, s_1, \mathcal{T}_i^N.\Omega) \impliedby M_I \preceq M$$

The truth of this formula follows immediately from the definition of reduction for any $\mathcal{T}_i^N \subseteq X^*$ and $\Omega \in \Upsilon(X, Y)$. Thus, I have proven that the refined adaptive state counting algorithm generates a test suite that is sufficient to test for reduction.

Theorem 7.3.2. *Let $\mathcal{T}_i^N = \mathcal{T}_{i+1}^N$ be the test suite generated by the refined adaptive state counting algorithm for FSM M representing the specification, FSM M_I representing the SUT with at most $m \geq |S|$ states, and a deterministic state cover V and adaptive characterizing set Ω of M. Then, the following holds:*

$$M_I \preceq_{\mathcal{T}_i^N.\Omega} M \Leftrightarrow M_I \preceq M$$

That is, M_I is a reduction of M on $\mathcal{T}_i^N.\Omega$ if and only if it is a reduction of M.

7.4 A Refined Adaptive State Counting Algorithm

Having proven the correctness of the generated test suite, I present next an implementation of the modified algorithm in Algorithm 7.1.

This algorithm differs from Algorithm 5.1 on page 52 (i.e., the adaptive state counting algorithm as presented in [**hierons**]) in only two points:

Algorithm 7.1 Refined adaptive state counting algorithm

1: **function** *performRefinedAdaptiveStateCounting*(
  in $M = (S, s_1, X, Y, h) : FSM$, ▷ Specification
  in $M_I = (T, t_1, X, Y, h_I) : FSM$, ▷ SUT
  in $V : \mathbb{P}(X^*)$, ▷ Deterministic state cover of M
  in $\Omega : \mathbb{P}(\Upsilon(X, Y))$, ▷ Adaptive characterizing set of M
  in $m : \mathbb{N}$, ▷ Upper bound on $|T|$
) : $Bool \times \mathbb{P}(X^*)$ ▷ Conformance result and test suite
2: $\mathcal{T}^N, \mathcal{C}^N, \mathcal{R}^N : \mathbb{P}(X^*)$;
3: $\mathcal{T}^N \leftarrow V$; ▷ Sequences followed by Ω in this or previous iterations
4: $\mathcal{C}^N \leftarrow V$; ▷ Elements of $\mathcal{T}^N$ considered for extension
5: $\mathcal{R}^N \leftarrow \emptyset$; ▷ Sequences of $\mathcal{C}^N$ to be removed
6: $\mathcal{O}^N, \mathcal{O}_I^N : \mathbb{P}((X/Y)^*)$; ▷ Observed responses of M, M_I to $\mathcal{C}^N.\Omega$
7: $\mathcal{O}^N \leftarrow \bigcup_{\bar{x} \in \mathcal{C}^N} \{\{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x})\}$;
8: $\mathcal{O}_I^N \leftarrow \bigcup_{\bar{x} \in \mathcal{C}^N} \{\{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x})\}$;
9: $V' : \mathbb{P}(\mathbb{P}((X/Y)^*))$; ▷ Possible responses of the SUT to V
10: $V' \leftarrow Perm(V, M_I)$;
11: $iter : \mathbb{N}$; ▷ Iteration counter
12: $iter \leftarrow 1$;
13: **while** $\mathcal{C}^N \neq \emptyset \wedge \mathcal{O}_I^N \subseteq \mathcal{O}^N$ **do**
14: $iter \leftarrow iter + 1$;
15: $\mathcal{R}^N \leftarrow \{\bar{x}' \in \mathcal{C}^N \mid$
$\forall \bar{y}' \in h_I^{out}(t_1, \bar{x}') : \exists S_1 \subseteq S, \bar{x}/\bar{y} \in (X/Y)^* :$
$\exists V'' \in N(\bar{x}'/\bar{y}', V'), \bar{v}/\bar{v}' \in V'' :$
$\bar{v}\bar{x}/\bar{v}'\bar{y} = \bar{x}'/\bar{y}'$
$\wedge\ \bar{v}/\bar{v}' = mcp(\bar{x}'/\bar{y}', V'')$
$\wedge\ \forall s_1, s_2 \in S_1, s_1 \neq s_2 :$
$\forall \bar{x}_1/\bar{y}_1 \in R^+(M, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') :$
$\forall \bar{x}_2/\bar{y}_2 \in R^+(M, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') :$
$B(M_I, \Omega, \bar{x}_1/\bar{y}_1) \neq B(M_I, \Omega, \bar{x}_2/\bar{y}_2)$
$\wedge\ LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}^N, S_1, \Omega, V'') > m\}$;
16: $\mathcal{C}^N \leftarrow ((\mathcal{C}^N \setminus \mathcal{R}^N).X) \setminus \mathcal{T}^N$;
17: $\mathcal{T}^N \leftarrow \mathcal{T}^N \cup \mathcal{C}^N$;
18: $\mathcal{O}^N \leftarrow \mathcal{O}^N \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \{\{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x})\}$;
19: $\mathcal{O}_I^N \leftarrow \mathcal{O}_I^N \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \{\{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x})\}$
20: **end while**
21: **return** $(noFailure(\mathcal{O}_I^N, \mathcal{O}^N), \mathcal{T}^N)$
22: **end function**

- An iteration counter *iter* is introduced in lines 11 and 12. This counter is then increased by 1 in line 14 each time the program inside the **while** loop is executed. This counter was added only to simplify the proof in Section 7.5 and, serving no other purpose, can be omitted in any practical implementation.
- The calculation of sequences to remove in line 15 is now based on the narrowed subset of V' and for some $\bar{x}' \in X^*$ with response $\bar{y}' \in Y^*$ in M_I considers only $V'' \in N(\bar{x}'/\bar{y}', V') = V'_{\bar{x}'/\bar{y}'}$ instead of all $V'' \in V'$.

The restriction to consider only those V'' contained in $N(\bar{x}'/\bar{y}', V')$ has a twofold effect on the complexity of the algorithm:

1. It is necessary to calculate $N(\bar{x}'/\bar{y}', V')$. This amounts to finding the maximum length element $\bar{v}$ of V that is a prefix of $\bar{x}'$ and then collecting all $V'' \in V'$ that contain some $\bar{v}/\bar{v}' \in \mathit{pref}(\bar{x}'/\bar{y}')$. The first part can be performed using at most $\mathcal{O}(|V| \cdot |\bar{x}'|)$ operations, as for each $\bar{v} \in V$ at most $\mathcal{O}(|\bar{x}'|)$ operations are needed to check whether it is a prefix of $\bar{x}'$ and if it is, then only a constant number of additional operations is required to check whether it is longer than the previously found prefix. Next, let $\bar{v}_p/\bar{v}'_p$ be the prefix of $\bar{x}'/\bar{y}'$ such that $\bar{v}_p$ is the maximum length prefix of $\bar{x}'$ in V. Then, the collection of V'' to consider requires at most $\mathcal{O}(|V'| \cdot |V| \cdot |\bar{x}'|)$ operations for checking each $V'' \in V'$, where $|V''| = |V|$, and each $\bar{v}/\bar{v}' \in V''$ whether $\bar{v}/\bar{v}' = \bar{v}_p/\bar{v}'_p$, where also $|\bar{v}_p| \leq |\bar{x}'|$.
2. By not considering any $V'' \in (V' \setminus N(\bar{x}'/\bar{y}', V'))$, the lower bound for such V'' does not need to be calculated.

7.5 Correctness of the Refined Algorithm

In this section, I prove that the implementation of the refined adaptive state counting algorithm given with Algorithm 7.1 generates the test suite $\mathcal{T}_i^N = \mathcal{T}_{i+1}^N$ described in Section 7.2 or observes some failure during execution. More specifically, I prove that the algorithm returns $(\mathit{true}, \mathcal{T}^N)$ if and only if M_I is a reduction of M, and that in this case $\mathcal{T}^N = \mathcal{T}_i^N = \mathcal{T}_{i+1}^N$.

To perform this proof, it is necessary to reference specific lines of the algorithm. Let PL_k denote line k of Algorithm 7.1, excluding any semicolons. For example, PL_{12} refers to the statement $i \leftarrow 1$. Furthermore, all axioms and techniques used to prove correctness of the program are found in to Chapter 4. Particularly, the notion of substitutions adopted in this work is still as follows: For some expression x and formula p possibly containing some variable y, let $p[x/y]$ denote the formula created by replacing each occurrence of y in p by x. Thus, for example, $(a = b + 3)[(c + 2)/b] \equiv (a = c + 5)$.

I proceed by first discussing the pre- and postconditions to use. After that, I provide a proof for the correct behaviour of the **while** loop, which I then extend to a proof for the entire algorithm.

7.5.1 Input Constraints

I only prove the correctness of the algorithm over sensible inputs for M, M_I, V, Ω and m. The formula $\mathit{pre}(M, M_I, V, \Omega, m)$ then holds if and only if all the

following hold:

- The FSMs $M = (S, s_1, X, Y, h)$ and $M_I = (T, t_1, X, Y, h_I)$ are both completely specified and observable and share an input alphabet X and an output alphabet Y.
- $m \geq |S|$ and M_I has at most m states (i.e., $m \geq |T|$).
- V is a deterministic state cover of M.
- Ω is an adaptive characterizing set of M.

In the following sections, I then use $pre(M, M_I, V, \Omega, m)$ as a precondition to limit the possible choices of inputs to sensible combinations. In doing so, I generally only write *pre*, as the parameters used are always M, M_I, V, Ω and m.

7.5.2 Sketch of a Postcondition

The postcondition is to encode that for the return value $(f, \mathcal{T}^N)$ of the algorithm at least one of the following must hold:

a) The final test suite has been calculated and hence there exists some $i \in \mathbb{N}$ such that both $\mathcal{T}^N = \mathcal{T}_i^N$ and $\mathcal{T}_i^N = \mathcal{T}_{i+1}^N$.

b) A failure has been observed and hence $\mathcal{O}_I^N \not\subseteq \mathcal{O}^N$. That is, the value of $f = noFailure(\mathcal{O}_I^N, \mathcal{O}^N)$ is *false*.

In both cases, the result is meaningless unless it is also required that $\mathcal{O}_I^N$ and $\mathcal{O}^N$ actually represent responses to $\mathcal{T}^N.\Omega$ and contain nothing else. Then, let *post* be the basic postcondition such that

$$post \equiv (q_{final} \vee q_{fail}) \wedge q_{\mathcal{O}_I^N} \wedge q_{\mathcal{O}^N}$$

where

$$\begin{aligned}
q_{final} &\equiv \exists i \in \mathbb{N} : \mathcal{T}^N = \mathcal{T}_i^N = \mathcal{T}_{i+1}^N \\
q_{fail} &\equiv \mathcal{O}_I^N \not\subseteq \mathcal{O}^N \\
q_{\mathcal{O}_I^N} &\equiv \mathcal{O}_I^N = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x}) \right\} \\
q_{\mathcal{O}^N} &\equiv \mathcal{O}^N = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\}
\end{aligned}$$

Here, $(q_{final} \vee q_{fail})$ ensures that (a) or (b) of the requirements above hold, while $(q_{\mathcal{O}_I^N} \wedge q_{\mathcal{O}^N})$ ensures that $\mathcal{O}_I^N$ and $\mathcal{O}^N$ contain exactly the respective responses of M_I and M to $\mathcal{T}^N.\Omega$.

Thus, if it can be proven that these postconditions hold for any input that satisfies *pre*, then the desired test suite has been calculated and applied, or during the execution of the algorithm an actual failure has been observed, possibly for a test suite smaller than $\mathcal{T}_i^N = \mathcal{T}_{i+1}^N$.

7.5.3 Loop Correctness

Before proving the correctness of the algorithm as a whole, I first show that the **while** loop calculates the correct sets for iteration *iter*. Furthermore, I prove that it does not loop indefinitely, and instead terminates when the final iteration has been calculated or a failure has been observed.

The main instrument in this proof is the Loop axiom for total correctness (see Section 4.2), which is defined as follows:

$$\frac{\begin{array}{c}\{p \wedge B\}\ S\ \{p\} \\ \{p \wedge B \wedge t = z\}\ S\ \{t < z\} \\ p \implies t \geq 0\end{array}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{end while}\ \{p \wedge \neg B\}}$$

For the **while** loop in Algorithm 7.1, the formula B is given by the following expression:

$$B \equiv \mathcal{C}^N \neq \emptyset \wedge \mathcal{O}_I^N \subseteq \mathcal{O}^N$$

Therefore, if B holds, then another iteration of the program inside the loop is executed. In establishing conditions for this inner program, the Assignment axiom is used to propagate backwards the desired postcondition, beginning from PL_{19}, the last statement inside the loop. To ease referencing the programs inside the loop, I here introduce some further notation: For some $14 \leq i \leq 19$, let W_i denote the sequential composition of statements in the **while** loop from line i to line 19:

$$W_i \equiv PL_i\ ;\ PL_{i+1}\ ;\ \ldots\ ;\ PL_{19}$$

Hence, $S \equiv W_{14}$. The loop can thus also be written as

$$\textbf{while}\ B\ \textbf{do}\ W_{14}\ \textbf{end while}$$

The formula p is then to ensure that the sets $\mathcal{T}^N$, $\mathcal{C}^N$, and $\mathcal{R}^N$ are equal to those given in Section 7.2 with respect to the iteration *iter*. That is, p must contain the following:

$$(\mathcal{T}^N = \mathcal{T}_{iter}^N) \wedge (\mathcal{C}^N = \mathcal{C}_{iter}^N) \wedge (\mathcal{R}^N = \mathcal{R}_{iter-1}^N)$$

where $\mathcal{R}^N$ is required to correspond to the removal set for the previous iteration, from which $\mathcal{T}^N$ and $\mathcal{C}^N$ are then constructed.

Furthermore, p is to encode that the inputs are valid and remain so. This extends to the calculation of V' from V, which is performed in the algorithm before executing the **while** loop. The following subformula of p achieves this goal:

$$pre \wedge V' = Perm(V, M_I)$$

Next, the formula p must ensure that the sets $\mathcal{O}_I^N$ and $\mathcal{O}^N$ are correctly maintained, which is readily achieved using the following subformula:

$$q_{\mathcal{O}_I^N} \wedge q_{\mathcal{O}^N}$$

Finally, the following expression is added to p:

$$(iter \leq (m^2 + 2))$$

This expression limits the iteration counter $iter$ to be at most $m^2 + 2$, which I have proven in Lemma 5.5.9 to be an upper bound on the number of iterations to calculate the test suite.

Thus, the following formulas have been established:

$$\begin{aligned} B &\equiv \mathcal{C}^N \neq \emptyset \wedge \mathcal{O}_I^N \subseteq \mathcal{O}^N \\ p &\equiv (\mathcal{T}^N = \mathcal{T}_{iter}^N) \wedge (\mathcal{C}^N = \mathcal{C}_{iter}^N) \wedge (\mathcal{R}^N = \mathcal{R}_{iter-1}^N) \\ &\quad \wedge pre \wedge V' = Perm(V, M_I) \\ &\quad \wedge q_{\mathcal{O}_I^N} \wedge q_{\mathcal{O}^N} \\ &\quad \wedge (iter \leq (m^2 + 2)) \end{aligned}$$

Therefore, all expressions necessary in the first premise of the Loop axiom have been assembled:

$$\{p \wedge B\}\ W_{14}\ \{p\}$$

Before introducing any expressions to use in place of t or z, I first prove this statement.

To accomplish this, I begin by repeatedly applying the Assignment axiom. As p is to be established as the postcondition of W_{14}, I initially set the postcondition of PL_{19} to p. Then, as

$$PL_{19} \equiv \mathcal{O}_I^N \leftarrow \mathcal{O}_I^N \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x}) \right\}$$

the following precondition can be created by using the Assignment axiom and substituting in p the new value of $\mathcal{O}_I^N$ for the old:

$$p^{19} \equiv p \left[\left(\mathcal{O}_I^N \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x}) \right\} \right) \ / \ \mathcal{O}_I^N \right]$$

By the axiom, $\{p^{19}\}\ PL_{19}\ \{p\}$ then holds. To enable the later application of the Composition axiom, in the next step I use p^{19} as a postcondition for PL_{18}.

More generally, for $14 \leq i \leq 19$, let p^i describe the precondition of PL_i constructed using the Assignment axiom from the postcondition p^{i+1} with $p^{20} = p$. The following formulas are constructed using this method:

$$\begin{aligned} p^{19} &\equiv p^{20} \left[\left(\mathcal{O}_I^N \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x}) \right\} \right) \ / \ \mathcal{O}_I^N \right] \\ p^{18} &\equiv p^{19} \left[\left(\mathcal{O}^N \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \right) \ / \ \mathcal{O}^N \right] \\ p^{17} &\equiv p^{18} \left[(\mathcal{T}^N \cup \mathcal{C}^N) / \mathcal{T}^N \right] \\ p^{16} &\equiv p^{17} \left[(((\mathcal{C}^N \setminus \mathcal{R}^N).X) \setminus \mathcal{T}^N) / \mathcal{C}^N \right] \\ p^{15} &\equiv p^{16} \left[\mathcal{R}' / \mathcal{R}^N \right] \\ p^{14} &\equiv p^{15} \left[(iter + 1) / iter \right] \end{aligned}$$

where $\mathcal{R}'$ is defined as follows:

$$\begin{aligned}\mathcal{R}' := \Big\{ \bar{x}' \in \mathcal{C}^N \;\Big|\; & \forall \bar{y}' \in h_I^{out}(t_1, \bar{x}') : \exists S_1 \subseteq S, \bar{x}/\bar{y} \in (X/Y)^* : \\ & \exists V'' \in N(\bar{x}'/\bar{y}', V'), \bar{v}/\bar{v}' \in V'' : \\ & \bar{v}\bar{x}/\bar{v}'\bar{y} = \bar{x}'/\bar{y}' \\ & \wedge\, \bar{v}/\bar{v}' = mcp(\bar{x}'/\bar{y}', V'') \\ & \wedge\, \forall s_1, s_2 \in S_1, s_1 \neq s_2 : \\ & \quad \forall \bar{x}_1/\bar{y}_1 \in R^+(M, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\ & \quad\quad \forall \bar{x}_2/\bar{y}_2 \in R^+(M, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\ & \quad\quad\quad B(M_I, \Omega, \bar{x}_1/\bar{y}_1) \neq B(M_I, \Omega, \bar{x}_2/\bar{y}_2) \\ & \wedge\, LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}^N, S_1, \Omega, V'') > m \Big\}\end{aligned}$$

I omit performing these substitutions in detail at this point: all intermediate results can be found in Appendix A. The final result p^{14}, which is to be used as a precondition for W_{14}, then is the following formula:

$$\begin{aligned}p^{14} \equiv\; & ((\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N)) = \mathcal{T}^N_{(iter+1)}) \\ & \wedge\, ((((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N) = \mathcal{C}^N_{(iter+1)}) \\ & \wedge\, (\mathcal{R}' = \mathcal{R}^N_{iter}) \\ & \wedge\, pre \\ & \wedge\, V' = Perm(V, M_I) \\ & \wedge \left(\mathcal{O}_I^N \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N)} \big\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x}) \big\} \right) \\ & \quad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N))} \big\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x}) \big\} \\ & \wedge \left(\mathcal{O}^N \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N)} \big\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \big\} \right) \\ & \quad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N))} \big\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \big\} \\ & \wedge\, ((iter + 1) \leq (m^2 + 2))\end{aligned}$$

With the above applications of the Assignment axiom, the following property has been established for all $14 \leq i \leq 19$:

$$\{p^i\}\ PL_i\ \{p^{i+1}\}$$

This result, together with the Composition axiom, the associativity of sequential composition, and the definitions that $W_{19} = PL_{19}$ and $p^{20} = p$, in turn enables the following inferences for all $14 \leq i \leq 18$:

$$\frac{\{p^i\}\ PL_i\ \{p^{i+1}\}\ ,\ \{p^{i+1}\}\ W_{i+1}\ \{p\}}{\{p^i\}\ PL_i\ ;\ W_{i+1}\ \{p\}}$$

In particular, this establishes that $\{p^{14}\}\ W_{14}\ \{p\}$ holds, as $W_{14} \equiv PL_{14};W_{15}$.

Later in this section, in Lemma 7.5.2, I use the Consequence axiom in the following way to establish that $\{p \wedge B\}\ W_{14}\ \{p\}$ holds:

$$\frac{p \wedge B \implies p^{14}, \{p^{14}\}\ W_{14}\ \{p\}, p \implies p}{\{p \wedge B\}\ W_{14}\ \{p\}}$$

To simplify this proof, I prove the following Lemma separately:

Lemma 7.5.1. *The formula*

$$p \wedge B \implies p^{14}$$

holds for p, p^{14} and B as defined above.

Proof. Following from the definition of p, $\mathcal{T}^N = \mathcal{T}^N_{iter}$ and $\mathcal{C}^N = \mathcal{C}^N_{iter}$ hold. By substituting $\mathcal{T}^N$ and $\mathcal{C}^N$ in $\mathcal{R}'$, an expression $\mathcal{R}'' = (\mathcal{R}'[\mathcal{T}^N_{iter}/\mathcal{T}^N])[\mathcal{C}^N_{iter}/\mathcal{C}^N]$ can be created such that

$$\begin{aligned}
\mathcal{R}'' \equiv \Big\{ \bar{x}' \in \mathcal{C}^N_{iter} \ \Big| \ & \forall \bar{y}' \in h^{out}_I(t_1, \bar{x}') : \exists S_1 \subseteq S, \bar{x}/\bar{y} \in (X/Y)^* : \\
& \exists V'' \in N(\bar{x}'/\bar{y}', V'), \bar{v}/\bar{v}' \in V'' : \\
& \quad \bar{v}\bar{x}/\bar{v}'\bar{y} = \bar{x}'/\bar{y}' \\
& \quad \wedge\, \bar{v}/\bar{v}' = mcp(\bar{x}'/\bar{y}', V'') \\
& \quad \wedge\, \forall s_1, s_2 \in S_1, s_1 \neq s_2 : \\
& \qquad \forall \bar{x}_1/\bar{y}_1 \in R^+(M, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\
& \qquad\quad \forall \bar{x}_2/\bar{y}_2 \in R^+(M, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\
& \qquad\qquad B(M_I, \Omega, \bar{x}_1/\bar{y}_1) \neq B(M_I, \Omega, \bar{x}_2/\bar{y}_2) \\
& \quad \wedge\, LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}^N_{iter}, S_1, \Omega, V'') > m \Big\}
\end{aligned}$$

Furthermore, as p is assumed to hold, $pre \wedge V' = Perm(V, M_I)$ follows, and therefore the expression $\mathcal{R}''$, and hence also $\mathcal{R}'$, is equivalent to the definition of $\mathcal{R}^N_{iter}$.

This allows the replacement of $\mathcal{T}^N$, $\mathcal{C}^N$ and $\mathcal{R}'$ in p^{14} by $\mathcal{T}^N_{iter}$, $\mathcal{C}^N_{iter}$ and $\mathcal{R}^N_{iter}$, respectively, as expressed in the following formula p':

$$p' := ((p^{14}[\mathcal{T}^N_{iter}/\mathcal{T}^N])[\mathcal{C}^N_{iter}/\mathcal{C}^N])[\mathcal{R}^N_{iter}/\mathcal{R}']$$

Applying these substitutions, p' is defined as follows:

$$
\begin{aligned}
p' \equiv\ & ((\mathcal{T}^N_{iter} \cup (((\mathcal{C}^N_{iter} \setminus \mathcal{R}^N_{iter}).X) \setminus \mathcal{T}^N_{iter})) = \mathcal{T}^N_{(iter+1)}) \\
& \wedge ((((\mathcal{C}^N_{iter} \setminus \mathcal{R}^N_{iter}).X) \setminus \mathcal{T}^N_{iter}) = \mathcal{C}^N_{(iter+1)}) \\
& \wedge (\mathcal{R}^N_{iter} = \mathcal{R}^N_{iter}) \\
& \wedge \mathit{pre} \\
& \wedge V' = \mathit{Perm}(V, M_I) \\
& \wedge \left(\mathcal{O}^N_I \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N_{iter} \setminus \mathcal{R}^N_{iter}).X) \setminus \mathcal{T}^N_{iter})} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \right) \\
& \quad = \bigcup_{\bar{x} \in (\mathcal{T}^N_{iter} \cup (((\mathcal{C}^N_{iter} \setminus \mathcal{R}^N_{iter}).X) \setminus \mathcal{T}^N_{iter}))} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \\
& \wedge \left(\mathcal{O}^N \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N_{iter} \setminus \mathcal{R}^N_{iter}).X) \setminus \mathcal{T}^N_{iter})} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \right) \\
& \quad = \bigcup_{\bar{x} \in (\mathcal{T}^N_{iter} \cup (((\mathcal{C}^N_{iter} \setminus \mathcal{R}^N_{iter}).X) \setminus \mathcal{T}^N_{iter}))} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \\
& \wedge ((\mathit{iter} + 1) \leq (m^2 + 2))
\end{aligned}
$$

I now show that each clause in p' is implied by $p \wedge B$. The first two clauses of p' hold trivially, as they follow the definitions of $\mathcal{T}^N_{(iter+1)}$ and $\mathcal{C}^N_{(iter+1)}$. Next, the third clause is a tautology. Furthermore, *pre* and $V' = \mathit{Perm}(V, M_I)$ are both contained in p and not contradicted in B, which allows their containment in p'.

Using the equalities described in the first two clauses, the sixth clause is equivalent to the following formula:

$$
\begin{aligned}
& \left(\mathcal{O}^N_I \cup \bigcup_{\bar{x} \in \mathcal{C}^N_{iter+1}} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \right) \\
& \quad = \bigcup_{\bar{x} \in \mathcal{T}^N_{iter+1}} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\}
\end{aligned}
$$

By the assumption p, the truth of $q_{\mathcal{O}^N_I}$ is implied, such that the following holds:

$$
\mathcal{O}^N_I = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\}
$$

Then, from $q_{\mathcal{O}^N_I}$ and the previously established $\mathcal{T}^N = \mathcal{T}^N_{iter}$, the desired equality

follows:

$$\begin{aligned}
&\left(\mathcal{O}_I^N \cup \bigcup_{\bar{x}\in\mathcal{C}_{iter+1}^N} \left\{\{\bar{x}/\bar{y}\}.B(M_I,\Omega,\bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1,\bar{x})\right\}\right)\\
&= \bigcup_{\bar{x}\in\mathcal{T}_{iter}^N} \left\{\{\bar{x}/\bar{y}\}.B(M_I,\Omega,\bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1,\bar{x})\right\}\\
&\quad \cup \bigcup_{\bar{x}\in\mathcal{C}_{iter+1}^N} \left\{\{\bar{x}/\bar{y}\}.B(M_I,\Omega,\bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1,\bar{x})\right\}\\
&= \bigcup_{\bar{x}\in(\mathcal{T}_{iter}^N \cup \mathcal{C}_{iter+1}^N)} \left\{\{\bar{x}/\bar{y}\}.B(M_I,\Omega,\bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1,\bar{x})\right\}\\
&= \bigcup_{\bar{x}\in\mathcal{T}_{iter+1}^N} \left\{\{\bar{x}/\bar{y}\}.B(M_I,\Omega,\bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1,\bar{x})\right\}
\end{aligned}$$

This proves that the sixth clause of p' is satisfied. The seventh clause is implied by an analogous argument using $q_{\mathcal{O}}^N$.

Finally, from B, it follows that no failure has been observed for $\mathcal{T}_{iter}^N$ and that $\mathcal{C}_{iter}^N \neq \emptyset$. The formula p also implies $iter \leq (m^2+2)$. Furthermore, using Lemma 5.5.9, $\mathcal{T}_{m^2+2}^N = \mathcal{T}_{m^2+3}^N$ holds, and hence, by definition, $\mathcal{C}_{m^2+2}^N = \emptyset$. Thus, $iter$ must be smaller than m^2+2, which, in turn, satisfies $(iter+1) \leq (m^2+2)$, the final clause of p'.

Therefore, I have proven that $p \wedge B \implies p'$ holds, which, following from the method of constructing p' from p^{14}, extends to a proof of $p \wedge B \implies p^{14}$. □

This result then leads to the correctness of the **while** loop.

Lemma 7.5.2. *The formula p is a loop invariant of the **while** loop in Algorithm 7.1 with respect to total correctness. That is, the following statement is true:*

$$\{p\}\ \textbf{\textit{while}}\ B\ \textbf{\textit{do}}\ W_{14}\ \textbf{\textit{end while}}\ \{p \wedge \neg B\}$$

Proof. As previously noted, I employ the Loop axiom to prove this Lemma. That is, I show that there exist t and z such that

$$\frac{\begin{array}{c}\{p \wedge B\}\ W_{14}\ \{p\}\\ \{p \wedge B \wedge t = z\}\ W_{14}\ \{t < z\}\\ p \implies t \geq 0\end{array}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ W_{14}\ \textbf{end while}\ \{p \wedge \neg B\}}$$

is a valid inference. The first premise, $\{p \wedge B\}\ S\ \{p\}$, follows directly from application of Lemma 7.5.1.

The choice of t is based on the established upper bound of the number of iterations required to calculate the test suite of the final iteration. Let z be some variable not used in Algorithm 7.1. I choose as t the following expression:

$$t := (m^2 + 2 - iter)$$

The third premise then reads as $p \implies (m^2+2-iter) \geq 0$, which trivially holds since p contains the clause $iter \leq (m^2+2)$.

Thus, the second premise remains to be shown, which reads as follows for the above choice of t:

$$\{p \wedge B \wedge z = (m^2 + 2 - iter)\}\ W_{14}\ \{(m^2 + 2 - iter) < z\}$$

which is equivalent to the following, where I explicitly write the assignment to t as a part of the program:

$$\{p \wedge B\}\ z \leftarrow (m^2 + 2 - iter)\ ;\ W_{14}\ \{(m^2 + 2 - iter) < z\}$$

Here, by repeatedly applying the axioms of Assignment and Composition analogously to the proof of Lemma 7.5.1, the following can be established:

$$\{(m^2 + 2 - (iter + 1)) < z\}\ W_{14}\ \{(m^2 + 2 - iter) < z\}$$

Using this precondition as a postcondition for the assignment of z, by the Assignment axiom the following holds:

$$\{(m^2 + 2 - (iter + 1)) < (m^2 + 2 - iter)\}$$
$$z \leftarrow (m^2 + 2 - iter)\ \{(m^2 + 2 - (iter + 1)) < z\}$$

Thus, over the whole loop and assignment, this extends to the following:

$$\{(m^2 + 2 - (iter + 1)) < (m^2 + 2 - iter)\}$$
$$z \leftarrow (m^2 + 2 - iter)\ ;\ W_{14}\ \{(m^2 + 2 - iter)) < z\}$$

Then, I use the Consequence axiom with the following strengthening of the precondition:

$$p \wedge B \implies (m^2 + 2 - (iter + 1)) < (m^2 + 2 - iter)$$

The implication is valid, as I have shown in the proof of Lemma 7.5.1 that $p \wedge B$ implies $(iter + 1) \leq (m^2 + 2)$ and hence $iter < m^2 + 2$, which satisfies the inequality. Writing the assignment to z as part of the precondition again, the following property can thus be inferred:

$$\{p \wedge B \wedge z = (m^2 + 2 - iter)\}\ W_{14}\ \{(m^2 + 2 - iter) < z\}$$

In turn, the inference using the Loop axiom above is correct, proving that the following holds:

$$\{p\}\ \textbf{while}\ B\ \textbf{do}\ W_{14}\ \textbf{end while}\ \{p \wedge \neg B\}$$

This provides the desired result. □

7.5.4 Initial Correctness

Having established a loop invariant p for the **while** loop, I show next that for any input to the algorithm satisfying *pre*, the execution of the statements before the loop results in a state that satisfies p. That is, I prove that p is a postcondition of the initial part of the algorithm.

Let $INIT \equiv PL_2\ ;\ \ldots\ ;\ PL_{12}$ denote the program from lines 2 to 12. Then, as *INIT* is a composition of assignments such that each variable in *INIT*

is assigned a new value at most once, it holds that for any input, any state reached by executing *INIT* satisfies the following postcondition:

$$\begin{aligned} p_{init} \equiv\ & (\mathcal{T}^N = V) \\ & \wedge (\mathcal{C}^N = V) \\ & \wedge (\mathcal{R}^N = \emptyset) \\ & \wedge V' = Perm(V, M_I) \\ & \wedge \mathcal{O}_I^N = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h_I^{out}(t_1, \bar{x}) \right\} \\ & \wedge \mathcal{O}^N = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \\ & \wedge (iter = 1) \end{aligned}$$

That is, the following statement is true:

$$\{true\}\ INIT\ \{p_{init}\} \tag{P1}$$

This result can be refined as follows.

Lemma 7.5.3. *For inputs satisfying pre, the initial part of Algorithm 7.1 reaches only states satisfying p. That is, the following statement is true:*

$$\{pre\}\ INIT\ \{p\}$$

Proof. First, the precondition *true* of (P1) can be trivially restricted to *pre*. Also, *INIT* does not modify any variable occurring in *pre*. Hence, the expression $\{pre\}\ INIT\ \{pre\}$ holds. From this and (P1), the following property can be derived:

$$\{pre\}\ INIT\ \{p_{init} \wedge pre\}$$

Any state that satisfies both p_{init} and *pre* then also satisfies p, as p_{init} describes the special case of $iter = 1$. Using the Consequence axiom, the following therefore is a valid inference:

$$\frac{pre \implies pre, \{pre\}\ INIT\ \{p_{init} \wedge pre\}, (p_{init} \wedge pre) \implies p}{\{pre\}\ INIT\ \{p\}}$$

The desired result follows. □

7.5.5 Overall Correctness

Using the results established above for the initial part of the algorithm and the **while** loop, I can now prove that the return value of the algorithm is correct with respect to checking for reduction.

Lemma 7.5.4. *Suppose that M, M_I, m, V and Ω satisfy the formula pre. Then, for these inputs, Algorithm 7.1 returns the pair*

$$performRefinedAdaptiveStateCounting(M, M_I, V, \Omega, m) = (true, \mathcal{T}^N)$$

if and only if $M_I \preceq M$ holds.

Proof. First, by line 21, the second value in the returned pair is always $\mathcal{T}^N$.

From the conditions established in Lemma 7.5.3 and Lemma 7.5.2, the following inference is gained by the Composition axiom:

$$\frac{\{pre\}\ INIT\ \{p\}\ ,\ \{p\}\ \textbf{while}\ B\ \textbf{do}\ W_{14}\ \textbf{end while}\ \{p \wedge \neg B\}}{\{pre\}\ INIT\ \textbf{;}\ \textbf{while}\ B\ \textbf{do}\ W_{14}\ \textbf{end while}\ \{p \wedge \neg B\}}$$

Notably,

$$INIT\ \textbf{;}\ \textbf{while}\ B\ \textbf{do}\ W_{14}\ \textbf{end while}$$

is the entire actual program in Algorithm 7.1 and is only followed by the **return** statement, which cannot modify any variable. Thus, if the algorithm is executed for inputs that satisfy *pre*, then the state reached when executing the **return** statement satisfies $p \wedge \neg B$.

Recall the postcondition *post* sketched in Subsection 7.5.2. If a state satisfies $p \wedge \neg B$, then it satisfies q_{final} or q_{fail}:

- If $\neg B$ is satisfied because $\mathcal{O}_I^N \not\subseteq \mathcal{O}^N$ holds, then trivially q_{fail} holds.
- Otherwise, $\neg B$ must hold because $\mathcal{C}^N$ contains no elements. By satisfying p, particularly $\mathcal{C}^N = \mathcal{C}_{iter}^N$ holds and thus $\mathcal{T}_{iter}^N = \mathcal{T}_{iter+1}^N$ follows by definition, which satisfies q_{final}.

Furthermore, $q_{\mathcal{O}_I^N}$ and $q_{\mathcal{O}^N}$ are contained as clauses in both p and *post*. Therefore, $p \wedge \neg B \implies post$ holds and hence

$$\{pre\}\ INIT\ \textbf{;}\ \textbf{while}\ B\ \textbf{do}\ W_{14}\ \textbf{end while}\ \{post\}$$

follows via use of the Composition axiom.

Next, I consider both implications expressed in the Lemma to prove: Case 1) $\implies$. For the algorithm to return *true* for $noFailure(\mathcal{O}_I^N, \mathcal{O}^N)$ while the postcondition *post* holds, it is not possible that a failure has been observed while applying $\mathcal{T}^N$. That is, q_{fail} cannot hold. From this and $(q_{\mathcal{O}_I^N} \wedge q_{\mathcal{O}^N})$, it follows that M_I is a reduction of M with respect to $\mathcal{T}^N$. Furthermore, q_{final} must hold and hence $\mathcal{T}^N = \mathcal{T}_i^N = \mathcal{T}_{i+1}^N$ for some $i \in \mathbb{N}$. This also implies that $\mathcal{T}_{iter}^N = \mathcal{T}_{iter+1}^N$. Finally, $M_I \preceq_{\mathcal{T}_{iter}^N.\Omega} M$ holds, which is sufficient to imply $M_I \preceq M$ by Theorem 7.3.2.

Case 2) $\impliedby$. If M_I is a reduction of M, then it is trivially also a reduction of M on any set of adaptive test cases. From the postcondition *post*, it follows that $\mathcal{T}^N.\Omega$ has been applied to both M and M_I. Therefore, q_{fail} cannot hold, and hence $noFailure(\mathcal{O}_I^N, \mathcal{O}^N)$ must be *true*. As discussed above, the algorithm always returns $\mathcal{T}^N$ as the second value in the returned pair. Thus, the return value of the algorithm can only be the desired $(true, \mathcal{T}^N)$.

□

7.6 Completeness

Let $(f, \mathcal{T}^N)$ be the result of executing the refined adaptive state counting algorithm for inputs M, M_I, V, Ω and m. Then, $\mathcal{T}^N.\Omega$ is a complete test suite with respect to the fault model $F(M, \preceq, \{M_I\})$:

- If $f = \mathit{true}$, then completeness follows from Theorem 7.3.2.
- If $f = \mathit{false}$, then M_I is neither a reduction of M nor does it pass $\mathcal{T}^N.\Omega$ with respect to M, which again implies completeness.

Due to the use of m as an upper limit on the number of states in M_I, the refined adaptive state counting algorithm can produce different test suites for M and M_I: Let $F(m)$ denote the set of all fault models $F(M, \preceq, \{M_I\})$ such that $M = (S, s_1, X, Y, h)$ and $M_I = (T, t_1, X, Y, h_I)$ are completely specified and observable FSMs that satisfy $m \geq |S|$ and $|T| \leq m$. Using some fixed $m \in \mathbb{N}$, the algorithm then is a complete testing theory for $F(m)$ that maps each $F(M, \preceq, \{M_I\}) \in F(m)$ to the test suite

$$\mathit{performRefinedAdaptiveStateCounting}(M, M_I, V, \Omega, m)$$

for corresponding choices of V and Ω. Depending on M, there might exist multiple choices for V and Ω.

Due to the adaptive techniques used in the algorithm, the generated test suite is not well-suited for use against FSMs other than M_I. For example, the test suite is not necessarily complete for any FSM where different response sets to Ω are observed after applying sequences in $\mathcal{T}^N$.

If $f = \mathit{true}$, then the test suite $\mathcal{T}^N$ is trivially complete for the fault model $F(M, \preceq, D(M_I))$ with the fault domain $D(M_I)$ defined as follows:

$$D(M_I) := \{(T', t_1, X, Y, h'_I) \mid T' \subseteq T \wedge h'_I \subseteq h_I \wedge \forall t' \in T', x \in X : |h'_I(t', x)| \geq 1\}$$

That is, $D(M_I)$ contains all FSMs M'_I that can be created from M_I by removing states and transitions such that M'_I remains completely specified. As M_I is observable, so is each $M'_I \in D(M_I)$. However, this result is only useful if it is known that the FSM to test is contained in $D(M_I)$, and in this case testing is unnecessary, as any FSM in $D(M_I)$ is by construction a reduction of M_I, which, by $f = \mathit{true}$, is a reduction of M.

In the absence of such deeper knowledge on the behaviour of the SUT, test suites generated by the presented algorithm can be meaningfully used for testing only with the SUT they have been created for. Thus, by reducing the number of sequences to apply, made possible with the application of adaptivity and described in [**hierons**], the reusability of the generated test suites is also reduced.

Chapter 8

Conclusion

The main goal of this work has been to analyse an error observed when applying the adaptive state counting algorithm originally presented in [**hierons**] and then to devise and prove correct a refinement of this algorithm that rectifies the error.

Following an introduction of the necessary definitions and concepts in Chapters 2 to 5, covering, in order, finite state machines, testing, program verification, and finally the adaptive state counting algorithm, the first part of this goal was achieved with a counterexample to this algorithm in Chapter 6. In the analysis in Section 6.2, I illustrated that the error was caused by terminating the extension of sequences too early, resulting in a test suite insufficient to prove reduction. This early termination was then shown to be the result of using invalid inputs when checking whether to stop extending some sequences.

This led to the modification proposed in Chapter 7, which consisted of not considering these invalid inputs. The resulting formulaic version of the algorithm was given in Section 7.2. In the subsequent section, I proved that the test suite generated using this refined algorithm is sufficient. Having thus achieved the second part of the goal, I also presented the algorithm as a program in Algorithm 7.1 and used Hoare-Logic to prove that this program produced the correct result when tasked to check whether some finite state machine is the reduction of another.

These modifications have been successfully incorporated into the work of Romero Früh in [**aro**]. Further work on adaptive methods of testing FSMs has been performed by Petrenko and Yevtushenko in [**petrenko2014adaptive**]. In their work, definitely reachable states are used instead of the deterministically reachable states employed in adaptive state counting.

8.1 Future Work

In this section, I briefly discuss two aspects of interest concerning future work based on the adaptive state counting algorithm. These are the use of proof assistants for program verification and variations on the algorithm that might speed up execution by applying fewer adaptive test cases, while also applying them less often.

8.1.1 Proof Assistants

As exhibited in Section 7.5 and considering in particular Appendix A, proving the correct behaviour of Algorithm 7.1, which is a simple program consisting of only a small number of statements, is a long and mechanical process. With respect to the repeated substitutions and minor modifications in the formulas, this process is also tedious and error-prone if performed manually. Thus, in future works, it would be preferable to employ proof assistant tools, like Isabelle/HOL[1], in which the theory of Hoare-Logic has already been implemented[2]. For an introduction to Isabelle/HOL and Hoare-Logic in its context, see [**nipkow2014concrete**].

8.1.2 Variations

Several ideas for further modifications of the adaptive state counting algorithm were introduced when discussing the example in Subsection 5.5.1. I conclude this work by outlining two such variations. Let $M = (S, s_1, X, Y, h)$ and $M_I = (T, t_1, X, Y, h_I)$ represent the specification and behaviour of the SUT, respectively, with $m > |S|$ and $|T| < m$ for some $m \in \mathbb{N}$. Furthermore, let V and Ω be a deterministic state cover and an adaptive characterizing set of M, respectively.

First, I consider an **IO-variation**: Subsection 5.5.1 explained that for some input sequence $\bar{x}x'$ that is to be applied and some response $\bar{y}y'$ of M_I to $\bar{x}x'$, the lower bound for IO sequence $\bar{x}/\bar{y}$ might already exceed m. In the IO-variation, the application of some $\bar{x}x'$ is not necessarily performed in full for each observed output, but instead stops if for the output observed so far, the lower bound is already exceeding m. In the example in Subsection 5.5.1 thus, when applying baa in the third iteration, using the IO-variation, effectively only $\{ba/02, baa/001\}$ instead of $\{baa/020, baa/022, baa/001\}$ is applied. Obviously, if some sequence is not applied fully for some observed response, then there are fewer sequences to apply Ω after. In the example, I only newly apply Ω after $baa/001$, as $ba/02$ was followed by Ω in the previous iteration. This illustrates the advantages and disadvantages of this variation: By applying Ω after fewer sequences, the number of sequences applied to M_I can be reduced to a degree dependent on the size of Ω. Similarly, by not applying sequences fully, the number of individual inputs applied to M_I is reduced. However, by applying Ω less often, it is possible that fewer distinct response sets to Ω are observed, which, in turn, decreases the second term of the lower bound calculation and hence might lead to some sequences requiring more extension until the lower bound calculated for them exceeds m. Finally, storing and retrieving which IO-sequences have already exceeded m with their lower bounds would require additional space and runtime.

The second variation incorporates ideas of the H-Method (see Section 3.3) into the algorithm, in what I call **H-variation**: First, it is not always necessary to apply the full set Ω after some applied sequence. Second, if it is necessary to distinguish the states that two IO-sequences reach in M_I, then it is sufficient to only apply new adaptive test cases after them in the case that they are not already distinguished by tests that have been applied so far.

[1] See https://isabelle.in.tum.de/
[2] See https://isabelle.in.tum.de/library/HOL/HOL-Isar_Examples/Hoare.html

Let $S_1 \subseteq S$ be a set of pairwise r-distinguishable states of M. Furthermore, let $\bar{v}\bar{x}/\bar{v}'\bar{y} \in L(M_I)$, where $\bar{v}$ is the maximum length prefix of $\bar{v}\bar{x}$ in V. Then, consider some $V'' \in V'_{\bar{v}\bar{x}/\bar{v}'\bar{y}}$, states $s_1, s_2 \in S_1$ with $s_1 \neq s_2$ and some sequences $\bar{x}_1/\bar{y}_1 \in R^+(M, M_I, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$, $\bar{x}_2/\bar{y}_2 \in R^+(M, M_I, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'')$. To ensure that these sequences are followed with test sets distinguishing s_1 and s_2 in M_I, simply check whether any test set exists that r-distinguishes s_1 and s_2 in M and that has already been applied after both sequences. If so, then no further tests need to be applied after the sequences. Otherwise, both sequences are to be followed with some test set that r-distinguishes s_1 and s_2 in M. This set can usually be selected to be smaller than Ω and is sufficient to distinguish the states in M_I by Lemma 5.3.7. Then, if the algorithm is modified to consider only sets $S_1 \subseteq S$ of pairwise r-distinguishable states when calculating the lower bound, the above procedure ensures that all relevant sequences have been followed by test sets sufficient to distinguish the reached states, and it is no longer necessary to apply all tests in Ω after each sequence.

Thus, this variation too can reduce the number of sequences applied, especially if M contains r-distinguishable states that cannot be visited along a single sequence. However, this variation also requires substantial modification concerning the second term of calculating the lower bound, as there no longer exists a single test set that has been applied after every sequence and for which the different response sets can be collected. A possible solution is to perform this count individually for each set $\mathcal{T}$ of already applied sequences and each test set Ω' such that all sequences in $\mathcal{T}$ have been followed with Ω'. As there could be a large number of such sequence and test sets, this calculation might need to be guided by some heuristic.

Appendix A

Calculating Precondition p^{14}

In this appendix, I present the intermediate results constructed when calculating the precondition p^{14} in Section 7.5.

I start from the postcondition $p^{20} = p$ where p^{20} is given by the following:

$$
\begin{aligned}
p \equiv\ & (\mathcal{T}^N = \mathcal{T}^N_{iter}) \\
& \wedge (\mathcal{C}^N = \mathcal{C}^N_{iter}) \\
& \wedge (\mathcal{R}^N = \mathcal{R}^N_{iter-1}) \\
& \wedge pre \\
& \wedge V' = P(V, M_I) \\
& \wedge \mathcal{O}^N_I = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \\
& \wedge \mathcal{O}^N = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \\
& \wedge (iter \leq (m^2 + 2))
\end{aligned}
$$

Furthermore, the set $\mathcal{R}'$ to substitute for $\mathcal{R}^N$ is defined as follows:

$$
\begin{aligned}
\mathcal{R}' := \Big\{ \bar{x}' \in \mathcal{C}^N \ \Big| & \forall \bar{y}' \in h^{out}_I(t_1, \bar{x}') : \exists S_1 \subseteq S, \bar{x}/\bar{y} \in (X/Y)^* : \\
& \exists V'' \in N(\bar{x}'/\bar{y}', V'), \bar{v}/\bar{v}' \in V'' : \\
& \quad \bar{v}\bar{x}/\bar{v}'\bar{y} = \bar{x}'/\bar{y}' \\
& \quad \wedge\ \bar{v}/\bar{v}' = mcp(\bar{x}'/\bar{y}', V'') \\
& \quad \wedge\ \forall s_1, s_2 \in S_1, s_1 \neq s_2 : \\
& \qquad \forall \bar{x}_1/\bar{y}_1 \in R^+(M, M_I, s_1, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\
& \qquad\quad \forall \bar{x}_2/\bar{y}_2 \in R^+(M, M_I, s_2, \bar{v}/\bar{v}', \bar{x}/\bar{y}, V'') : \\
& \qquad\qquad B(M_I, \Omega, \bar{x}_1/\bar{y}_1) \neq B(M_I, \Omega, \bar{x}_2/\bar{y}_2) \\
& \quad \wedge\ LB(M, M_I, \bar{v}/\bar{v}', \bar{x}/\bar{y}, \mathcal{T}^N, S_1, \Omega, V'') > m \Big\}
\end{aligned}
$$

Then, the following formula expresses the first substitution:

$$
p^{19} \equiv p^{20} \left[\left(\mathcal{O}^N_I \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \right) \ / \ \mathcal{O}^N_I \right]
$$

Its result is equivalent to the following:

$$
\begin{aligned}
p^{19} \equiv\ & (\mathcal{T}^N = \mathcal{T}^N_{iter}) \\
& \wedge (\mathcal{C}^N = \mathcal{C}^N_{iter}) \\
& \wedge (\mathcal{R}^N = \mathcal{R}^N_{iter-1}) \\
& \wedge pre \\
& \wedge V' = P(V, M_I) \\
& \wedge \left(\mathcal{O}^N_I \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \\
& \wedge \mathcal{O}^N = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \\
& \wedge (iter \leq (m^2 + 2))
\end{aligned}
$$

Next, the following substitution is to be applied:

$$
p^{18} \equiv p^{19} \left[\left(\mathcal{O}^N \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \right) \ / \ \mathcal{O}^N \right]
$$

Its result is equivalent to the following:

$$
\begin{aligned}
p^{18} \equiv\ & (\mathcal{T}^N = \mathcal{T}^N_{iter}) \\
& \wedge (\mathcal{C}^N = \mathcal{C}^N_{iter}) \\
& \wedge (\mathcal{R}^N = \mathcal{R}^N_{iter-1}) \\
& \wedge pre \\
& \wedge V' = P(V, M_I) \\
& \wedge \left(\mathcal{O}^N_I \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \\
& \wedge \left(\mathcal{O}^N \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in \mathcal{T}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \\
& \wedge (iter \leq (m^2 + 2))
\end{aligned}
$$

The following substitution is next:

$$
p^{17} \equiv p^{18} \left[(\mathcal{T}^N \cup \mathcal{C}^N) / \mathcal{T}^N \right]
$$

Its result is equivalent to the following:

$$
\begin{aligned}
p^{17} \equiv\ & ((\mathcal{T}^N \cup \mathcal{C}^N) = \mathcal{T}^N_{iter}) \\
& \wedge (\mathcal{C}^N = \mathcal{C}^N_{iter}) \\
& \wedge (\mathcal{R}^N = \mathcal{R}^N_{iter-1}) \\
& \wedge pre \\
& \wedge V' = P(V, M_I) \\
& \wedge \left(\mathcal{O}^N_I \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup \mathcal{C}^N)} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \\
& \wedge \left(\mathcal{O}^N \cup \bigcup_{\bar{x} \in \mathcal{C}^N} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup \mathcal{C}^N)} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \\
& \wedge (iter \leq (m^2 + 2))
\end{aligned}
$$

The sequence of substitutions continues with the following formula:

$$p^{16} \equiv p^{17} \left[(((\mathcal{C}^N \setminus \mathcal{R}^N).X) \setminus \mathcal{T}^N) / \mathcal{C}^N \right]$$

Its result is equivalent to the following:

$$
\begin{aligned}
p^{16} \equiv\ & ((\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}^N).X) \setminus \mathcal{T}^N)) = \mathcal{T}^N_{iter}) \\
& \wedge ((((\mathcal{C}^N \setminus \mathcal{R}^N).X) \setminus \mathcal{T}^N) = \mathcal{C}^N_{iter}) \\
& \wedge (\mathcal{R}^N = \mathcal{R}^N_{iter-1}) \\
& \wedge pre \\
& \wedge V' = P(V, M_I) \\
& \wedge \left(\mathcal{O}^N_I \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N \setminus \mathcal{R}^N).X) \setminus \mathcal{T}^N)} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}^N).X) \setminus \mathcal{T}^N))} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \\
& \wedge \left(\mathcal{O}^N \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N \setminus \mathcal{R}^N).X) \setminus \mathcal{T}^N)} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}^N).X) \setminus \mathcal{T}^N))} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \\
& \wedge (iter \leq (m^2 + 2))
\end{aligned}
$$

Afterward, the following substitution is to be performed:

$$p^{15} \equiv p^{16} \left[\mathcal{R}' / \mathcal{R}^N \right]$$

Its result is equivalent to the following:

$$
\begin{aligned}
p^{15} \equiv\ & ((\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N)) = \mathcal{T}^N_{iter}) \\
& \wedge ((((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N) = \mathcal{C}^N_{iter}) \\
& \wedge (\mathcal{R}' = \mathcal{R}^N_{iter-1}) \\
& \wedge \mathit{pre} \\
& \wedge V' = P(V, M_I) \\
& \wedge \left(\mathcal{O}^N_I \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N)} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N))} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \\
& \wedge \left(\mathcal{O}^N \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N)} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N))} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \\
& \wedge (\mathit{iter} \leq (m^2 + 2))
\end{aligned}
$$

Finally, the desired precondition is given by the following substitution:

$$p^{14} \equiv p^{15}\,[(\mathit{iter} + 1)/\mathit{iter}]$$

Its result is equivalent to the following:

$$
\begin{aligned}
p^{14} \equiv\ & ((\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N)) = \mathcal{T}^N_{(iter+1)}) \\
& \wedge ((((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N) = \mathcal{C}^N_{(iter+1)}) \\
& \wedge (\mathcal{R}' = \mathcal{R}^N_{iter}) \\
& \wedge \mathit{pre} \\
& \wedge V' = P(V, M_I) \\
& \wedge \left(\mathcal{O}^N_I \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N)} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N))} \left\{ \{\bar{x}/\bar{y}\}.B(M_I, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}_I(t_1, \bar{x}) \right\} \\
& \wedge \left(\mathcal{O}^N \cup \bigcup_{\bar{x} \in (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N)} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \right) \\
& \qquad = \bigcup_{\bar{x} \in (\mathcal{T}^N \cup (((\mathcal{C}^N \setminus \mathcal{R}').X) \setminus \mathcal{T}^N))} \left\{ \{\bar{x}/\bar{y}\}.B(M, \Omega, \bar{x}/\bar{y}) \mid \bar{y} \in h^{out}(s_1, \bar{x}) \right\} \\
& \wedge ((\mathit{iter} + 1) \leq (m^2 + 2))
\end{aligned}
$$