Test Selection, Trace Distance and Heuristics [*]

L.M.G. Feijs, N. Goga, S. Mauw [†] J. Tretmans [‡]
feijs, goga, sjouke@win.tue.nl tretmans@cs.utwente.nl

September 12, 2001, version 0.1

Abstract: Since exhaustive testing is in general impossible, an important step in the testing process is the development of a carefully selected test suite. Selection of test cases is not a trivial task. We propose to base the selection process on a well–defined strategy. For this purpose, we formulate two heuristic principles: *the reduction heuristic* and *the cycling heuristic*. The first assumes that few outgoing transitions of certain states show essentially different behaviour. The second assumes that the probability to detect erroneous behaviour in a loop decreases after each correct execution of the loop behaviour. We formalize these heuristic principles and we define a coverage function which serves as a measure for the error–detecting capability of a test suite. For this purpose we introduce the notion of a *marked trace* and a distance function on such marked traces.

Keywords: test selection, test coverage, trace distance, test selection heuristics, edit distance.

1 Introduction

Systematic testing is an important technique to check and control the quality of software systems. Testing consists of systematically developing a set of experiments or test cases, then running these experiments on the software system that has to be tested, also referred to as the IUT (the implementation under test), and subsequently concluding from the observations made during the execution whether the IUT behaved as expected leading to a verdict about the IUT's correctness. Because of time and resource limitations, any form of testing can only exercise a small subset of all possible system behaviour. Therefore, testing can never give certainty about the correctness of a system; it can only increase confidence.

Since in practice exhaustive testing is impossible, an important step in the testing process is the development of a carefully selected test suite, i.e., a set of test cases. Such a test suite should have a large potential of revealing errors in the implementation. Moreover, we would like to be able to compare different test suites in order to select the best one, and to quantify their error-detecting capability.

The selection of an appropriate set of tests from all possible ones (usually infinitely many test cases), is not a trivial task. We refer to this task as *test selection*. Traditionally, test selection is based on a number of heuristic criteria. Well-known heuristics include equivalence partitioning, boundary value analysis, and use of code-coverage criteria like statement-, decision- and path-coverage [Mye79]. Although these criteria provide some heuristics for selecting test cases, they are rather informal and they do not allow to measure the error-detecting capability of a test suite.

If test cases are derived from a formal specification, in particular if it is done algorithmically using tools for automatic test generation, e.g., AUTOLINK [SKGH97], TGV [JM99]

[*]This research was supported by the Dutch Technology Foundation STW under project STW TIF.4111: *Côte de Resyste* – COnformance TEsting of REactive SYSTEms; URL: http://fmt.cs.utwente.nl/CdR.

[†]Eindhoven University of Technology, P.O. Box 513, NL–5600 MB Eindhoven, The Netherlands

[‡]University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

or TorX [BFV+99], then the test selection problem is even more apparent. These test tools can generate a large number of test cases, when given a specification in the appropriate formalism, without much user intervention. All these generated test cases can detect potential errors in implementations, and errors detected with these test cases indeed indicate that an implementation is not correct with respect to its specification. However, the number of potentially generated test cases may be very large, or even infinite. In order to control and get insight in the selection of the tests, and by that get confidence in the correctness of an IUT that passes the tests, it is important that the selection process is formally described and based on a well-defined strategy.

It should be noted, however, that test selection is an activity that in principle cannot be based solely on a formal specification of a system. In order to decide which test cases are more valuable than others, either extra information outside the realm of the specification formalism is necessary, or assumptions about the occurrence of errors in the implementation must be made. Such extra information may include knowledge about which errors are frequently made by implementers, which kind of errors are important, e.g., in the sense of having catastrophic consequences, what functionality is difficult to implement, which functionality is crucial for the well functioning of the system, etc. An approach to formalizing this extra information was given in [BTV91]. On the other hand, assumptions can be made about the occurrence of errors in implementations, e.g., that errors will not occur in isolation, i.e., if some behaviour is erroneous then there is a large probability that some other behaviour close to it is also erroneous. So we only have to test one of these behaviours (equivalence partitioning: the behaviours are equivalent with respect to the occurrence of errors). Another often used assumption is that errors are most likely to occur on the boundaries of valid data intervals (boundary value analysis).

This paper approaches the problem of test selection by making assumptions in an automata-based, or labelled transition system-based formalism. Section 2 introduces the labeled transition systems and automata. Two different kinds of assumptions are introduced and expressed as *heuristic principles* in Section 3 starting with the ideas of [CG96]. The first one, called *reduction heuristic*, assumes that few outgoing transitions of certain states show essentially different behaviour. The second one, referred to as *cycling heuristic*, assumes that the probability to detect erroneous behaviour in a loop decreases after each correct execution of the loop behaviour. After that we propose a mathematical framework, defining a heuristic as a function on the set of behaviours (traces). This is done in Section 4. When we want to make the two heuristics more precise, defining them as functions according to the definition from Section 4, we observe that an appropriate behaviour representation for them is needed. Therefore in Section 5 we define the *marked trace* representation. After these preparations the definitions of the heuristics as functions on marked traces are straight forward (Section 6). Subsequently, the notion of isolation and closeness of errors is formalized in Section 7 by defining a *distance* function between behaviours. This idea is taken from [ACV93, ACV97] and extended on *marked traces*. The trace distance implements the considered heuristics in the sense that the traces which are selected by the heuristics are remote from each other. Every trace which is excluded by the heuristics is close to one of the selected traces. A *coverage function* which may serve as a measure for the error-detecting capability of a test suite is defined based on the maximum distance between selected and non-selected behaviours and a formula for approximating the coverage is given in Section 8.

2 Preliminaries

The basic formalism for our discussion about test selection is the labelled transition system, or the automaton. A labelled transition system provides means to specify, model, analyze and reason about (concurrent) system behaviour. A labelled transition system is defined in terms of states and labelled transitions between states. In this section we recall some basic definitions.

Definition 2.1 A *labelled transition system* is a 4-tuple $\langle Q, L, T, q_0 \rangle$, where Q is a non-empty set of *states*, L is a set of *labels*, $T \subseteq Q \times L \times Q$ is the *transition relation*, and $q_0 \in Q$ is the *initial state.*

The labels in L represent the actions of a system. An action $a \in L$ is executable in state $q \in Q$ if $(q, a, q') \in T$ for some state $q' \in Q$, which is said to be the new state after execution of a; we also write $q \xrightarrow{a} q'$. A finite sequence of pairs $\langle$*state, action*$\rangle$ ending into a state is called a *path*. Similarly, a finite sequence of actions is called a *trace*. The set of all traces over L is denoted by L^*, with ϵ denoting the empty sequence. Abusing notation, we will use p to denote both the labelled transition system and the current (or initial) state of the system.

The traces of a labelled transition system p are all sequences of action that p can execute from its initial state q_0: $traces(p) =_{\text{def}} \{ \sigma \in L^* \mid q_0 \xrightarrow{\sigma} \}$. Here we use the following additional definitions ($n \in \mathbb{N}$, $i \leq n$, q, q', $q_i \in Q$, $a_i \in L$, $\sigma \in L^*$):

$$\begin{array}{lll} q \xrightarrow{a_1 \cdot \ldots \cdot a_n} q' & =_{\text{def}} & \exists q_0, \ldots, q_n : \; q = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n = q' \\ q \xrightarrow{\sigma} & =_{\text{def}} & \exists q' : \; q \xrightarrow{\sigma} q' \end{array}$$

For our presentation and formalization we use *minimal*, *deterministic*, *finite-state* transition systems. A finite-state labelled transition system has a finite number of states, i.e., Q is finite. A transition system is deterministic if for any state $q \in Q$ and action $a \in L$ there is at most one successor state, i.e., $T : Q \times L \to Q$ is a (partial) function. A transition system is minimal if there are no equivalent states, i.e., no two states with exactly the same traces, which means: $\nexists q, q' \in Q : traces(q) = traces(q')$. We (ab)use the word *automaton* for these minimal, deterministic, finite-state transition systems.

Although it may seem a severe limitation to restrict to automata an important formal test theory, viz. **ioco**-testing [Tre96], can be expressed, for the larger part, in terms of automata. So the test selection approach which is presented in this paper can be integrated with **ioco**-testing.

In testing, the traces of the minimal, deterministic, finite automata are used. A complete (maximal) test suite for an automaton specification s is expressed as $traces(s)$. However, even if s is finite-state, its set of traces will usually be infinite and contain traces of unbounded length. Hence, a complete test suite will have infinitely many tests of unbounded length. Such a test suite can never be executed within any reasonable limits of time and resources. Consequently, the problem of *test selection* consists of selecting a finite subset $T \subseteq traces(s)$, such that we end up with a reasonably sized set of bounded-length test cases.

The challenge of test selection now is to choose T such that the resulting test suite keeps a large error-detecting capability. Moreover, we wish to quantify this capability in order to compare and select test suites. The next sections will present and formalize an approach to selection and quantification.

3 Introduction to heuristics, distance and coverage

In this section we introduce the concepts of heuristics and coverage. Two specific heuristics will be proposed in Section 3.1. They are illustrated by an example in Section 3.2.

3.1 The heuristics principles for the test selection

As motivated in Section 2, the specification is seen as a minimal finite-state automaton. The specification has a set of traces which usually is too large; for this reason, we want to obtain a smaller set of traces. As we explained in Section 1, this goal can be reached by making assumptions on automata, assumptions which are expressed as heuristic principles. The heuristic principles with which we are working in this paper are:

- *Reduction:* if the specification automaton contains for certain states a large number of outgoing transitions, only a small number of these transitions need to be selected;
- *Cycling:* each cycle in the automaton needs to be traversed only a limited number of times by every single trace.

3.2 A first illustration of the test selection

Now we will give an example on which we will apply our heuristics for test selection.

Let s be the specification automaton from the left–hand side of Figure 1. The specification has four states. The labelset is $L = \{b, c, d, e, f\} \cup \{a_i \mid i \in \mathbb{N}\}$ and the initial state is the state *I*. This state has infinitely many outgoing transitions ($\{a_i \mid i \in \mathbb{N}\}$). Via a transition a_i from the initial state, one arrives at *II*. This state contains a cycle which goes via *III* using the transitions b and d; the state *III* contains another cycle, via the transition c. From *II* one arrives at *IV* using e or f. For simplicity, we will consider only the traces of s that end in *IV*. Let T be the set of traces of s.

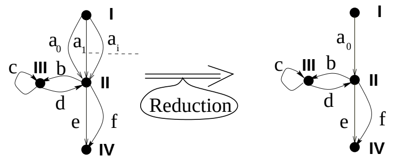

Figure 1: A minimal automaton of a specification

Now let us consider a_0 as being the representative transition in the initial state; by choosing this transition the *Reduction* heuristic is applied in this state. By this application we reduce the labelset to a finite one $L' = \{a_0, b, c, d, e, f\}$. This labelset L' corresponds to the reduced automaton from the right–hand side of Figure 1. Now our initial set of traces becomes $T^{\textit{Reduction}}$ and it contains all the traces of the automaton which are starting from state *I*, arriving in state *IV* and going through transition a_0 in *I* ($T^{\textit{Reduction}}$ equals also the set of traces of the reduced automaton). In this example one representative is selected; in a more general example it could also be two or more of the a_i.

The following heuristic to be applied is the *Cycling* heuristic. The traces are cycling via the states *II* and *III* of the automaton. In this automaton the state *II* has a cycle via the sequence of transitions bd and the state *III* has another cycle via the transition c. We can fix the cycle limit number to 1 for the cycling states *II* and *III*. So the transitions c from *III* and bd from *II* can be traversed only once by every single trace of $T^{\textit{Reduction}}$. The traces which respect this condition and form the set of traces $T^{\textit{Reduction, Cycling}}$ are:

$$T^{\textit{Reduction, Cycling}} = \{a_0e, a_0bde, a_0bcde, a_0f, a_0bdf, a_0bcdf\}$$

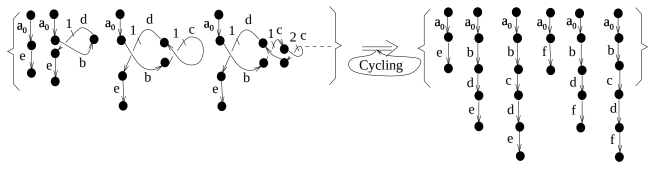

Figure 2: An application of the *Cycling* heuristic

The application of the *Cycling* heuristic is represented graphically in Figure 2. The full set is represented at the left–hand side and the reduced set at the right. As it can be seen the set $T^{\textit{Reduction, Cycling}}$ is a finite set and all its traces have a length of at most 5 (finite length).

Our method deals with bigger cycles as well, such as going via the transitions bd several times; the technique of [ACV93] deals only with simple cycles – such as going via transition

c several times. Another advantage is that with the proposed test selection technique one can deal with an infinite branching of transitions (see the initial state of this automaton). As we saw in our example, limiting the cycle number implicitly limits the length and therefore a length heuristic, which is considered by [ACV93], is not necessary for us.

Now we are going to express the heuristic principles in terms of distances among traces. A distance is a measure which expresses how far apart two traces are. A particular way to compute such a trace distance is given in Section 7.2. For getting a feeling of how the trace distance is related to the heuristic principles, let us take as an example the distance between the traces a_0bdf and a_0bdbdf. In Figure 3, it can be seen that the distance between the traces a_0bdf and a_0bdbdf is smaller than the distance for example between a_0f and a_0bdbdf. This happens because the trace a_0bdf cycles one time via the state II, a_0bdbdf cycles two times and a_0f cycles zero times. Therefore intuitively, the trace a_0bdf should be closer to a_0bdbdf than to the other traces (exactly as we assume in the *Cycling* heuristic that the later cycles are less important, so the distance between two traces which are cycling more often through a state will decrease).

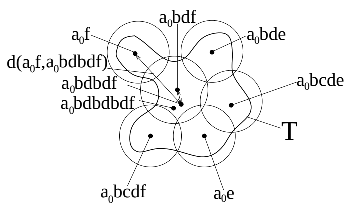

Figure 3: A covering of the initial set using trace distance

In Figure 3, every trace from the reduced set $T^{Reduction,\ Cycling}$ is the center of a sphere. The initial set T is covered by the reduced set $T^{Reduction,\ Cycling}$, such that every trace from T has a corresponding trace in $T^{Reduction,\ Cycling}$ to which the distance is smaller than a given limit ε (ε is the radius of the spheres). This process of selecting one representative for each sphere leads to a notion of coverage. When taking big spheres only few representatives are selected and the error detection capability is low. Small spheres, on the other hand give a large coverage. If we scale things in such a way that $0 \leq \varepsilon \leq 1$, then the coverage can be expressed as $1 - \varepsilon$. The coverage of the reduction from T to $T^{Reduction,\ Cycling}$ is denoted as $cov(T, T^{Reduction,\ Cycling})$. Therefore we express $cov(T, T^{Reduction,\ Cycling}) = 1 - \varepsilon$.

This gave some intuition about how the heuristics and the trace distance are used in the test selection and computation of the coverage. Now we are going to be more formal.

4 The trace distance and the test heuristics

In our test selection method we use heuristics which are applied on traces and distances between traces. This section describes the formal definitions of these notions.

In a formal way a trace heuristic is a function between two sets of traces such that the range is a proper subset of the domain (so the heuristic reduces the size of the initial set).

Definition 4.1 A trace heuristic h is a function $h : T \longrightarrow T$, where T is a set of traces and $Ran(h) \subset T$.

Definition 4.2 Let T be a set. Then a function $d : T \times T \longrightarrow \mathbb{R}_{\geq 0}$ is a distance iff: 1) $d(x,x) = 0$; 2) $d(x,y) = d(y,x)$; 3) $d(x,y) \leq d(x,z) + d(z,y)$; for all $x,y,z \in T$.

In particular we use Definition 4.2 for sets of traces and such distances are called trace distances. The pair (T,d) is a metric space. It is customary to express coverages by numbers in the range $[0,1]$ and therefore we restrict ourselves to distance functions such that $0 \leq d(x,y) \leq 1$ for all x,y. This can be done without loss of generality (suppose we would have distances in $[0,\infty)$ and ε numbers in the range $[0,\infty]$ then we could scale them back to $[0,1]$ using a suitable monotonic and continuous bijection $b : [0,\infty] \longrightarrow [0,1]$). In order to use a trace distance for test selection the concept of ε–cover is useful.

Definition 4.3 A set T' is an ε–cover of T ($T' \subseteq T, \varepsilon \geq 0$) if for every $t \in T$ there exists $t' \in T'$ such that $d(t,t') \leq \varepsilon$.

The property of ε–cover gives rise to the property of total boundedness for a metric space.

Definition 4.4 A metric space (T,d) is totally bounded if for every $\varepsilon > 0$ it is possible to find a finite set $T_\varepsilon \subseteq T$ such that T_ε is an ε–cover of T with respect to distance d.

Now a link between a heuristic and a trace distance is established: if for that heuristic the subset obtained by the application of that heuristic is an ε–cover of the original set, then the trace distance implements the heuristic.

Definition 4.5 Let T be a set of traces and h be a trace heuristic such that $h : T \longrightarrow T$. Let d be a trace distance defined on T. Then d implements the heuristic h iff: $\exists \varepsilon_h \geq 0$: $Ran(h)$ is an ε_h–cover of T with respect to the distance d.

The following definition shows how to obtain the coverage.

Definition 4.6 Let T be a set of traces and $T' \subseteq T$ be an ε–cover of T with respect to a trace distance d. Let $\varepsilon_m = inf\{\varepsilon \geq 0 \mid T' \text{ is an } \varepsilon\text{–cover of } T\}$ be the inferior minimum of the ε values. Then the coverage of T' with respect to T is $cov(T',T) = 1 - \varepsilon_m$.

5 The marked trace representation

When we want to make the two heuristics more precise, defining them as functions according to Definition 4.1, we observe that an appropriate trace representation for them is needed. When we apply the *Cycling* heuristic on a trace, we observe that the trace does not have enough information regarding how it was generated, what states it has been going through and how often it went through them. As a result, we will represent the trace in such a way that the information regarding its generation from the automaton will be included. This leads us to a concept called marked traces, which will be developed in Section 5.1. In general a given trace can be interpreted in several ways as being the result of running through cycles in the automaton. This introduces a problem of ambiguity which is addressed in Section 5.2.

5.1 The marked traces

The first example in this section will explain why a new representation for the traces is needed.

Example Let us consider the automaton from Figure 4 and one of its traces *abcbcd*. This trace is traversing (cycling) twice via the state *II*. But this information is not present in the trace. Therefore this representation is not appropriate for working with cycles. Now let us transform it into a path, which is $IaIIbIIIcIIbIIIcIIdIV$. We can observe that the path contains extra information which is not needed for cycles: for example it contains the states I and IV which are not part of any cycle. Summing up the observations, we arrive at the conclusion that a new representation is needed. An intuitive one is $a[\langle bc\rangle\langle bc\rangle]^{2,II}d$ where $[\langle bc\rangle\langle bc\rangle]^{2,II}$ indicates that two cycles of the transitions bc are performed through the state II.

As we saw in the introductory example, we associate the cycles with how many times a trace is traversing a state. The name of the state, which is seen as a mark, will represent the identifier of the cycle. Also we will include the number of cycles through a state. We

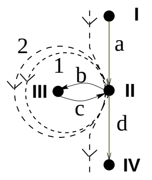

Figure 4: A trace which cycles through an automaton

call such an extended trace a *marked trace*. Now we have all the ingredients to define a marked representation of a trace.

Definition 5.1 Let L be a labelset and Q a set of states. Then a marked trace is inductively defined by

1. $a \in L$, ϵ and $[\ ]^{0,q}$ $(q \in Q)$ are marked traces;
2. if u and v are marked traces then uv is a marked trace;
3. if u and $[\sigma]^{n,q}$ $(n \in \mathbb{N}, q \in Q)$ are marked traces then $[\sigma\langle u\rangle]^{n+1,q}$ is a marked trace.

In the definition from above σ is a sequence of type $\langle\sigma_1\rangle ... \langle\sigma_n\rangle$ where σ_i are marked traces $(i = 1, ...n)$.
Example Some examples of marked traces are: $a[\langle bc\rangle]^{1,II}d$ and $a[\langle bc\rangle\langle bc\rangle]^{2,II}d$ with $II \in Q$.
We will denote the set of all the marked traces over a labelset L and a set of marks Q as L^*_Q. The transformation between the marked representation of a trace and a normal representation of a trace can be made easily by eliminating all the parentheses which occur in the marked representation. For example the marked trace $a[\langle bc\rangle\langle bc\rangle]^{2,II}d$ is transformed in the trace *abcbcd*. We will call this transformation *unfold*.

Definition 5.2 Let L be a labelset, let Q be the set of marks and let L^*_Q be the set of marked traces. Then the function $unfold : L^*_Q \longrightarrow L^*$ which transforms a marked trace into a trace is

1. if $a \in L, q \in Q$ then $unfold(a) = a$, $unfold(\epsilon) = \epsilon$, $unfold([\ ]^{0,q}) = \epsilon$;
2. if u and v are marked traces then $unfold(uv) = unfold(u)unfold(v)$;
3. if u and $[\sigma]^{n,q}$ $(q \in Q, n \in \mathbb{N})$ are marked traces then

   $unfold([\sigma\langle u\rangle]^{n+1,q}) = unfold([\sigma]^{n,q})unfold(u)$.

In the following example, we will illustrate a way in which a trace can be transformed into a marked trace. In general, this transformation is not unique. To illustrate this, we need a more complex example.
Example Consider the automaton from Figure 1. The states of the automaton are marked with *I, II, III, IV*. The cycling states are the states *II* and *III*. Consider the trace $a_0bcdbcde$. Adding boxes to reflect nesting structure, the corresponding path is $Ia_0\boxed{II}b\boxed{\boxed{III}}c\boxed{\boxed{III}}d\boxed{II}b\boxed{\boxed{III}}c\boxed{\boxed{III}}d\boxed{II}eIV$. The state *II* (surrounded with a box in the path) is repeated three times. Between two occurrences of state *II* in the path, the state *III* (surrounded with two boxes) appears twice. If we match every new occurrence of state *II* in the path with its first occurrence (we will call this way of matching the states *first state matching*), the path will be divided in 4 component paths: $\underbrace{Ia_0II}_{1}[\langle\underbrace{IIbIIIcIIIdII}_{2}\rangle\langle\underbrace{IIbIIIcIIIdII}_{3}\rangle]^{2,II}\underbrace{IIeIV}_{4}$.
If we do the same for *III* in the paths 2) *IIbIII cIIIdII* and 3) *IIbIIIcIIIdII* and eliminate all the states, we obtain the marked trace $a_0[\langle b[\langle c\rangle]^{1,III}d\rangle\langle b[\langle c\rangle]^{1,III}d\rangle]^{2,II}e$. This marked trace corresponds to the initial trace $a_0bcdbcde$. However, there are also other ways of transforming it into a marked trace. For example, the states of the same trace can be

grouped in another way as $IaII b\boxed{III}c\boxed{III}dIIb\boxed{III}c\boxed{III}dIIeIV$ and the same trace has another correspondent marked trace which is $a_0b[\langle c\rangle\langle db\rangle\langle c\rangle]^{3,III}de$.

For this example we see that there is not a unique way of transforming a trace in a marked trace. Therefore we leave it as an option to the implementer (the user of our theory) to choose the way by which he transforms a trace into a marked trace (one algorithm will be given in the next subsection – Subsection 5.2). We will only define the set of marked traces of an automaton which will be the set of all possible marked traces which can be derived from the traces of an automaton. After that we make a partition on the set of marked traces (an equivalence class of marked traces contains all the marked traces which *unfold* to the same trace) and we assume that the implementer chooses a set of representatives from the equivalence classes of marked traces. Below we will define the set of marked traces of an automaton. In this definition, we use $C[\![x]\!]$ to denote a term with an occurrence of a substring x. $C[\![\ ]\!]$ is called the context in which x occurs.

Definition 5.3 Let s be an automaton. Let $path(s)$ be the set of its paths; let Q be the set of its states. Then

1. the set of marked paths $MPath(s)$ of s is:
   (a) if $p \in path(s)$ then $p \in MPath(s)$;
   (b) if $m \in MPath(s)$, $m = C[\![p]\!]$, $p \in path(s)$ such that $p = p_1qp_2q...qp_n$, then $C[\![p_1q[\langle qp_2q\rangle...\langle qp_{n-1}q\rangle]^{n-2,q}qp_n]\!] \in MPath(s)$ with $q \in Q$, $n \in \mathbb{N}$, p_1q, $qp_iq, qp_n \in path(s)$, $i = 2, ..., n-1$;
2. the set of marked traces of an automaton is: $MTraces(s) = \{m\ |_{trace}|\ m \in MPath(s)\}$, where $|_{trace}$ transforms a path to a trace by eliminating all the states which appear in the path and keeping all the labels.

At this point it is not necessary to demand that the procedure from step 1.b) is applied until it cannot be done further; we come back to this in Section 5.2. Considering the way we constructed $MTraces(s)$, it is evident that every trace has at least one corresponding marked trace. Two marked traces will be equivalent if their *unfold* will give the same trace.

Definition 5.4 Let m_1, m_2 be two marked traces. Then m_1 is equivalent with m_2, denoted as $m_1 \sim m_2$, iff: $unfold(m_1) = unfold(m_2)$.

The equivalence relation gives a partition on $MTraces(s)$ in a set of equivalence classes. By choosing a representative for every class, the implementer builds the set of representatives $traces^m(s)$. A way to obtain such a set is given in the beginning of the next subsection. For the remainder of this paper we will work only with the set of representatives $traces^m(s)$ and we will abuse the words *marked trace* for such representative marked trace.

5.2 An algorithm for obtaining a set of representatives

In the beginning of this subsection we give a way to implement the transformation of a trace in a representative marked trace and to obtain the set of representatives.

This set is obtained by applying the following function (ALG) on each trace (path) of a finite-state minimal deterministic automaton s. The function builds a marked trace from a trace using a first state match technique like the one we used for the trace $a_0bcdbcde$ at the beginning of the previous example. In ALG we use the following function and procedure: 1) the function $NotRepetitivesState(p, Q)$, p a path, Q a set of states, returns true if every state of p which is contained in Q occurs only once in p and 2) the procedure $Divide(p,\ Q,\ q,\ p_1, ...,\ p_n)$ finds $q \in Q$ and splits p in $p_1, ..., p_n$ ($n \in \mathbb{N}$, $i = 2, ..., n-1$, p_1q,

qp_iq, qp_n paths) such that: i) $q \in Q$ is the first repetitive state in p, ii) $p = p_1qp_2q...qp_n$ and iii) the set of states of p_j does not contain q $(j = 1, ..., n)$.

```
function $ALG$ ($p$ : $Path$, $Q$ : $SetStates$) : $MarkedTrace$;
var $q$ : $State$;
    $p_1, ..., p_n$ : $(\epsilon + Label)(State\ Label)^*(\epsilon + State)$;
begin
   if ($NotRepetitiveState(p, Q)$) then
      return $p\ |_{trace}$;
   else
      $Divide(p, Q, q, p_1, ..., p_n)$;
      $Q = Q \setminus \{q\}$;
      return $ALG(p_1q, Q)[\langle ALG(qp_2q, Q)\rangle ... \langle ALG(qp_{n-1}q, Q)\rangle]^{n-2,q} ALG(qp_n, Q)$;
end
```

Initially, the function ALG is applied on a path p and on the set of states Q of the automaton. When a) p does not contain states from Q which are repetitive, ALG returns the trace corresponding to p ($p\ |_{trace}$). When a) does not hold, ALG finds the first repetitive state $q \in Q$ in p and divides p in n parts $p_1, ..., p_n$. Every p_i $(i = 2, ..., n-1)$ lies between two occurrences of q in p; p_1 and p_n are the initial part (ending with q) and respectively the last part (starting with q) of p. After this the state q is deleted from Q, which becomes $Q \setminus \{q\}$. Making so, terms as $[\_[\_]^{-,q}\_]^{-,q}$ in which q is interpreted twice as a cycle are avoided. Without deletion, the repetition of q can be reinterpreted as a cycle by a later call of ALG, when it is applied on a component path qp_iq. After the transformation of Q, ALG returns the concatenations of the marked traces obtained by recursively applying the algorithm to the components $p_1, ..., p_n$, which is $ALG(p_1q, Q)[\langle ALG(qp_2q, Q)\rangle ... \langle ALG(qp_{n-1}q, Q)\rangle]^{n-2,q} ALG(qp_n, Q)$.

The set of representatives is $traces^m(s) = \{ALG(p, Q) \mid p \in path(s)\}$. In the remainder of this paper, in the examples which we use, we assume that the marked traces are generated with ALG from the traces of an automaton.

As one can see, our way of building the set of representatives is rather complex. One can imagine trivial solutions as for example: every marked trace is the trace itself. But the marked traces built with ALG have nice properties which are required for the application of our test selection theory. For example the width of such marked traces is uniformly bounded (Lemma 5.6), a property which is used in the theorem of total boundedness (Theorem 8.2). The marked traces generated with the trivial solution do not have this property.

In conclusion, once we have the set of representatives, we want to know if it has some specific properties. So for a marked trace of an automaton we want to know if the width of it is uniformly bounded, and if the nesting depth of it is also bounded. Here uniformly bounded means that the same upperbound applies at all nesting levels. But first let us define these terms.

Definition 5.5 Let s be an automaton. Let L be the labelset and Q the set of states of s. Then the function $width : traces^m(s) \longrightarrow \mathbb{N}$ is

1. if $a \in L, q \in Q$ then $width(a) = 1$, $width(\epsilon) = 0$, $width([\ ]^{0,q}) = 1$;
2. if u and v are marked traces then $width(uv) = width(u) + width(v)$;
3. if u and $[\sigma]^{n,q}$ $(q \in Q, n \in \mathbb{N})$ are marked traces then $width([\sigma\langle u\rangle]^{n+1,q}) = 1$.

In the definition above the terms $[\_]^{-,-}$ are counted as single terms of the marked trace.

Example Let us take the trace $a_0[\langle bd\rangle\langle bd\rangle]^{2,II}f$. Then

$$width(a_0[\langle bd\rangle\langle bd\rangle]^{2,II}f) = width(a_0) + width([\langle bd\rangle\langle bd\rangle]^{2,II}) + width(f) = 1 + 1 + 1 = 3$$

The following lemma shows that the width of every marked trace generated with ALG is uniformly bounded.

Lemma 5.6 *The width of a marked trace generated with ALG from an automaton and the widths of all its component marked traces are less than or equal to $2m-1$, where m is the number of the states of the automaton. (Without proof)*

Definition 5.7 Let s be an automaton. Let L be the labelset and Q the set of states of s.

Then the function $nesting : traces^m(s) \longrightarrow \mathbb{N}$ is

1. if $a \in L, q \in Q$ then $nesting(a) = 0$, $nesting(\epsilon) = 0$, $nesting([\ ]^{0,q}) = 1$;
2. if u and v are marked traces then $nesting(uv) = max(nesting(u), nesting(v))$;
3. if u and $[\sigma]^{n,q}$ $(q \in Q, n \in \mathbb{N})$ are marked traces then

   $nesting([\sigma\langle u\rangle]^{n+1,q}) = 1 + max(nesting(\sigma), nesting(u))$.

Example Let us take the trace $a_0[\langle bd\rangle\langle b[\langle c\rangle]^{1,II}d\rangle]^{2,II}f$. Then

$$nesting(a_0[\langle bd\rangle\langle b[\langle c\rangle]^{1,III}d\rangle]^{2,II}f) = max(nesting(a_0), nesting([\langle bd\rangle\langle b[\langle c\rangle]^{1,II}d\rangle]^{2,II}),$$

$$nesting(f)) = max(0, 1 + max(nesting(bd), nesting(b[\langle c\rangle]^{1,III}d)), 0) = max(0, 2, 0) = 2$$

The following lemma shows that the nesting depth of every marked trace generated with ALG is bounded.

Lemma 5.8 *The nesting depth of a marked trace generated with ALG from an automaton is less than or equal to the number of the states of the automaton. (Without proof)*

As we motivated before, for applying our theory of test selection we need some specific properties to be owned by the set of representatives. So we require for the set of (representative) marked traces of an automaton that the width of every marked trace, the widths of all its component marked traces, and its nesting depth to be uniformly bounded. In this subsection we showed that the marked traces generated with ALG have these properties (Lemma 5.6, Lemma 5.8). Certain algorithms works as well. For example, similarly as we did in this subsection, one can prove that the marked traces obtained with a *last state matching* technique (the last repetitive state of the path is matched) have also these properties. Independent of the way in which the set of marked traces is obtained, once it has the required properties, our test selection theory can be applied on it.

Now we have an algorithm that makes sure that every trace of the automaton has a unique correspondent representative marked trace, we will work with marked traces in place of traces throughout the remainder of this paper.

6 The heuristics defined for marked traces

Below we will define the heuristics in a formal way. As we presented in Section 3.1, the intuition behind the heuristics *Reduction* and *Cycling* is that they take into account two aspects: the finiteness of 1) the number of outgoing transitions of certain states and of 2) the number of times each cycle can be traversed by every single trace.

When *Reduction* is applied, the labelset L is split in two parts: the selected labels which form a finite set $L' \subseteq L$ and the set of unselected labels which is $L \setminus L'$. This application can be seen as the application of a mapping function *trans*: $L \longrightarrow L'$ which maps every unselected label to a selected label from L' and every selected label to itself. One practical way to make the selection and to obtain L' and *trans* is by defining a distance d_L between labels, such that the metric space (L, d_L) is totally bounded. Let us fix a positive real number $\varepsilon_L \geq 0$. Now L' will be a labelset which is an ε_L–cover of L. The labels which are remote from each other (their distance is greater than ε_L) are selected and the labels from $L \setminus L'$ remain unselected. The function *trans*: $L \longrightarrow L'$ can be defined in this case such that $trans(a) = b$ with $a \in L, b \in L'$ and $d_L(a, b)$ minimum.

For the *Cycling* heuristic we relate the cycles of the automaton to the marked representation of the trace; limiting the numbers of times of traversing the cycles means limiting the powers of the marked symbols in the marked traces. Now, let us define these heuristics in a formal way.

Definition 6.1 Let s be an automaton. Let L be the labelset and Q the set of states of s. Let $L' \subseteq L$ be a finite subset of L and let *trans*: $L \longrightarrow L'$ be the mapping function. Then the heuristic *Reduction* : $\mathit{traces}^m(s) \longrightarrow \mathit{traces}^m(s)$ is

1. if $a \in L, q \in Q$ then $\mathit{Reduction}(a) = \mathit{trans}(a)$, $\mathit{Reduction}(\epsilon) = \epsilon$, $\mathit{Reduction}([\ ]^{0,q}) = [\ ]^{0,q}$;
2. if u and v are marked traces then $\mathit{Reduction}(uv) = \mathit{Reduction}(u)\mathit{Reduction}(v)$;
3. if u and $[\sigma]^{n,q}$ $(q \in Q, n \in \mathbb{N})$ are marked traces then

   $\mathit{Reduction}([\sigma\langle u\rangle]^{n+1,q}) = [\mathit{Reduction}(\sigma),\mathit{Reduction}(u)]^{n+1,q}$.

Example Let us consider the automaton from Figure 1. For this automaton the set of labels is $L = \{c, b, d, e, f\} \cup \{a_i \mid i = 0, 1, ...\}$.

Let $L' = \{a_0, c, b, d, e, f\}$ be a finite subset of L and *trans*: $L \longrightarrow L'$

$$\mathit{trans}(x) = \begin{cases} a_0 & x = a_i, i \in \mathbb{N} \\ x & \text{otherwise} \end{cases}$$

Then $\mathit{Reduction}(a_3 e) = \mathit{Reduction}(a_3)\mathit{Reduction}(e) = \mathit{trans}(a_3)\mathit{trans}(e) = a_0 e$.

Definition 6.2 Let s be an automaton. Let L be the labelset and Q the set of states of s. Let l_c be the cycle limit. Then the heuristic *Cycling* : $\mathit{traces}^m(s) \longrightarrow \mathit{traces}^m(s)$ is

1. if $a \in L, q \in Q$ then $\mathit{Cycling}(a) = a$, $\mathit{Cycling}(\epsilon) = \epsilon$, $\mathit{Cycling}([\ ]^{0,q}) = [\ ]^{0,q}$;
2. if u and v are marked traces then $\mathit{Cycling}(uv) = \mathit{Cycling}(u)\mathit{Cycling}(v)$;
3. if u and $[\sigma]^{n,q}$ $(q \in Q, n \in \mathbb{N})$ are marked traces then
   (a) $\mathit{Cycling}([\sigma\langle u\rangle]^{n+1,q}) = [\mathit{Cycling}(\sigma)\mathit{Cycling}(u)]^{n+1,q}$, for $l_c > n$;
   (b) $\mathit{Cycling}([\sigma\langle u\rangle]^{n+1,q}) = [\mathit{Cycling}(\sigma')]^{l_c,q}$, for $l_c \leq n$
   where $\sigma = \langle\sigma_1\rangle...\langle\sigma_n\rangle$ and $\sigma' = \langle\sigma_1\rangle...\langle\sigma_{l_c}\rangle$ is obtained by cutting σ after l_c symbols.

Example Let us consider the automaton from Figure 1. Let us fix l_c to 2. Then
$\mathit{Cycling}(a_0[\langle bd\rangle\langle bd\rangle\langle bd\rangle]^{3,II}e) = \mathit{Cycling}(a_0)\mathit{Cycling}([\langle bd\rangle\langle bd\rangle\langle bd\rangle]^{3,II})\mathit{Cycling}(e) =$
$a_0[\langle bd\rangle\langle bd\rangle]^{2,II}e$

Lemma 6.3 *Reduction(Cycling(x))=Cycling(Reduction(x)) (Without proof)*

7 The trace distance for marked traces

In this section we make the trace distance more precise, defining it as a distance function according to Definition 4.2. As explained in Section 4, this gives us an alternative formalization of the ideas behind the heuristics (they will be compared in Section 8). We will combine these ideas with another well–known idea, viz. the edit distance. Section 7.1 introduces the edit distance. After this preparation, the definition of the trace distance function can be given (Section 7.2).

7.1 The edit distance between strings

Because in our trace distance we use the concept of edit distance we shall present this first. The concept is applied in problems such as string search, words substitution using dictionaries, etc. Informally the edit distance is defined as the minimum number of insertions, deletions and substitutions required to transform one string into another.

Levenshtein ([Ste92]) defined the edit distance $d(x, y)$ between two strings x and y as the minimum of the cost of editing x to transform it into y. The cost of editing is the sum of the costs of a number of atomic edit actions. According to Levenshtein the costs are as follows: inserting a symbol costs 1, deleting a symbol costs 1 and changing an a into a b costs 1 too.

Wagner and Fisher ([Ste92]) generalized the definition of Levenshtein by adopting different costs for the various atomic edit actions. According to Wagner–Fisher transforming a into a b costs $w(a,b)$. Extending this notation, $w(a,\epsilon)$ is the cost of deleting a and $w(\epsilon,b)$ is the cost of inserting b. Again, the cost of editing is the sum of the costs of the atomic edit actions, and $d(x,y)$ is the minimum cost over all possible edit sequences that transform x into y.

Definition 7.1 Let $w(a,b)$ be the weighting for the cost of transforming symbol a in symbol b, $w(a,\epsilon)$ be the cost of deleting a and $w(\epsilon,b)$ be the cost of inserting b. Of course $w(a,a)=0$. Then the edit distance between the strings x and y is denoted as $ED(x,y)$ and it is computed as

1. $ED(au,bv) = min(w(a,b)+ED(u,v), w(a,\epsilon)+ED(u,bv), w(\epsilon,b)+ED(au,v))$;
2. $ED(au,\epsilon) = w(a,\epsilon)+ED(u,\epsilon)$; $ED(\epsilon,bv) = w(\epsilon,b)+ED(\epsilon,v)$; $ED(\epsilon,\epsilon)=0$;

   where a,b are symbols and u,v are strings.

This definition will be used throughout the paper.

Example Let us take the labelset $L=\{a,b,c\}$ with the cost 1 for insertion, deletion, and for transforming a symbol in another symbol. The edit distance between a and ba is computed as

$$ED(a,ba) = min(w(a,b)+ED(\epsilon,a), w(a,\epsilon)+ED(\epsilon,ba), w(\epsilon,b)+ED(a,a)) = min(1+w(\epsilon,a)$$
$$+w(\epsilon,\epsilon), 1+w(\epsilon,b)+w(\epsilon,a)+w(\epsilon,\epsilon), 1+min(w(a,a)+ED(\epsilon,\epsilon), w(a,\epsilon)+ED(\epsilon,a),$$
$$w(\epsilon,a)+ED(a,\epsilon))) = min(1+1, 1+2, 1+0) = 1$$

So the edit distance between a and ba is 1 which corresponds to the deletion of b.

7.2 Defining the trace distance

Our test selection technique uses two heuristics. For expressing these heuristics in the trace distance, it is important to remember that in the formalization of the *Reduction* heuristic a label distance was used. The incorporation of this heuristic in the trace distance is achieved in a simple way by using the label distance in the formula of the trace distance. Now a solution should be found for the *Cycling* heuristic.

For the *Cycling* heuristic we simply weight every level k of a cycling symbol (a marked trace of type $[\_]^{n,q}, n \in \mathbb{N}, q \in Q$) with a weight from a series of positive numbers p_k. This series has the property that $\sum_{k=1}^{\infty} p_k = 1$. The logic behind this weighting is that summing the weights after a given limit (which is the cycle limit) will contribute with a small number reflecting our assumption that the first cycles are more important than the later cycles.

We will define the trace distance for all the possible combinations of the points (1), (2), (3) of Definition 5.1 (which are generating marked traces). We summarize these combinations below

- between the marked traces generated with point (1) (such as $[\ ]^{0,q}, q \in Q$ and $a \in L$) we will define a distance function called *AtomicDistance* because these are the atomic elements which form the marked trace; of course the *AtomicDistance* between two labels will be given by d_L, the distance between these labels; between a label and a marked trace such as $[\ ]^{0,q}$ it will be maximum (one) and between two marked traces such as $[\ ]^{0,q}$,$[\ ]^{0,q'}$ $(q,q' \in Q, q \neq q')$ it will also be one;
- between the marked traces generated with point (2) (such as af or ae) we will use a distance function called *EditDistance*; we took this option because these traces are generated in a similar style as the strings are formed and it is quite natural to use it because it compares in a good way the terms which form the marked traces (for example in the traces a_0e and $a_0[\langle bd\rangle]^{1,II}e$ the edit distance will recognize that the labels a_0 and e from the first trace are present in the second trace);

- between the marked traces generated with point (3) (such as building $[\langle bd\rangle]^{1,II}$ once we know that $[\ ]^{0,II}$ is a marked trace) we employ the principle that cycles of different marks are very remote and hence have the maximum distance, i.e, 1; when dealing with cycles of the same mark we employ weighting factors p_k with the effect that the later iterations are considered less important than e.g. the first iteration; this can be done by using a function *EditDistanceWeighted* which is an edit distance for which the formula of Definition 7.1 is modified in such a way to take into account the weights.

The rest of the possible combinations such as (1) with (2), (2) with (3) etc. are defined in a similar style by using one of the techniques mentioned above (*EditDistance* or *AtomicDistance*).

We observe also that this trace distance is to be used in the computation of coverage which should be in the range $[0, 1]$. For simplifying the computation of coverage, we want the trace distance values to be in the range $[0, 1]$. This can be done by dividing all the above mentioned values (generated with an *EditDistance* or *AtomicDistance*) by the maximum width of the marked traces from $traces^m(s)$ (the maximum width is finite, see Section 5.2). For completing the picture it is necessary to add that the trace distance between a null trace (ϵ) and any other marked trace is maximum (1).

Now we have all the ingredients to define a trace distance on marked traces. We will call it d. In the definition, the distances already mentioned (*EditDistance* and *AtomicDistance*) will be used; also it is implicitly assumed that the definition is symmetric in the sense that $d(x, y) = d(y, x)$, x and y being marked traces and that $d(x, x) = 0$.

As explained above (first bullet), the function *AtomicDistance* deals with the cases ϵ, $a \in L$ and $[\ ]^{0,-}$. We generalize it to marked traces of the form $[\_]^{-,-}$ as well.

Definition 7.2 Let s be an automaton. Let L be the labelset of s, d_L the label distance defined on it and Q the set of states of s. The metric space (L, d_L) is totally bounded and d_L has all its values in the range $[0, 1]$. Let l_m be the maximum of the width of the marked traces from $traces^m(s)$. Let p_k ($k = 1, 2, ...$) be a series of positive numbers such that $\sum_{k=1}^{\infty} p_k = 1$. The trace distance d is symmetric in the sense that $d(x, y) = d(y, x)$, x and y being marked traces and that $d(x, x) = 0$. Then

1. $d(a, b) = \frac{AtomicDistance(a,b)}{l_m}$;

   $d(a, \epsilon) = d(\epsilon, [\ ]^{0,q}) = 1$;

   $d(a, [\ ]^{0,q}) = \frac{AtomicDistance(a,[\ ]^{0,q})}{l_m}$;

   $d([\ ]^{0,q}, [\ ]^{0,q'}) = \frac{AtomicDistance([\ ]^{0,q},[\ ]^{0,q'})}{l_m}$

   with $a, b \in L, q, q' \in Q$;
2. $d(a, uv) = \frac{EditDistance(a,uv)}{l_m}$;

   $d(\epsilon, uv) = 1$;

   $d([\ ]^{0,q}, uv) = \frac{EditDistance([\ ]^{0,q},uv)}{l_m}$

   with u, v marked traces and $a \in L, q \in Q$;
3. $d(a, [u\langle v\rangle]^{n+1,q}) = \frac{AtomicDistance(a,[u\langle v\rangle]^{n+1,q})}{l_m}$;

   $d(\epsilon, [u\langle v\rangle]^{n+1,q}) = 1$;

   $d([\ ]^{0,q}, [u\langle v\rangle]^{n+1,q}) = \frac{AtomicDistance([\ ]^{0,q},[u\langle v\rangle]^{n+1,q'})}{l_m}$

   with v and $[u]^{n,q'}$ marked traces ($n \in \mathbb{N}, q' \in Q$) and $a \in L, q' \in Q$;
4. $d(uv, rt) = \frac{EditDistance(uv,rt)}{l_m}$

   with u, v, r, t marked traces;

5. $d(uv, [r\langle t\rangle]^{n+1,q}) = \frac{EditDistance(uv,[r\langle t\rangle]^{n+1,q})}{l_m}$

   with $u, v, t, [r]^{n,q}$ marked traces ($n \in \mathbb{N}, q \in Q$);

6. $d([u\langle v\rangle]^{n+1,q}, [r\langle t\rangle]^{n'+1,q'}) = \frac{AtomicDistance([u\langle v\rangle]^{n+1,q},[r\langle t\rangle]^{n'+1,q'})}{l_m}$

   with $v, t, [u]^{n,q}, [r]^{n',q'}$ marked traces ($n, n' \in \mathbb{N}, q, q' \in Q$);

where

- $AtomicDistance(x, y) = \begin{cases} d_L(x, y) & x, y \in L \\ EditDistanceWeighted(x, y) & x = [\_]^{n,q}, y = [\_]^{n',q}, q \in Q \\ & n, n' \in \mathbb{N}, n \neq 0, n' \neq 0 \\ 1 & \text{otherwise} \end{cases}$
- $EditDistance(uv, rt) = min(AtomicDistance(u, r) + EditDistance(v, t),$
  $AtomicDistance(u, \epsilon) + EditDistance(v, rt),$
  $AtomicDistance(\epsilon, r) + EditDistance(uv, t));$

  $EditDistance(uv, \epsilon) = AtomicDistance(u, \epsilon) + EditDistance(v, \epsilon);$
  $EditDistance(\epsilon, rt) = AtomicDistance(\epsilon, r) + EditDistance\ (\epsilon, t);$
  $EditDistance(\epsilon, \epsilon) = 0$

  with u, v, r, t marked traces;
- $EditDistanceWeighted([\langle u_1\rangle ... \langle u_n\rangle]^{n,q}, [\langle v_1\rangle ... \langle v_p\rangle]^{p,q}) = EDW^1([\langle u_1\rangle ... \langle u_n\rangle]^{n,q}, [\langle v_1\rangle ... \langle v_p\rangle]^{p,q})$

  with u_i and v_j marked traces ($i = 1, ..., n,\ j = 1, ..., p,\ n, p \in \mathbb{N},\ q \in Q$).

We add some explanation. It is easy to check that the definition of *EditDistance* and *EditDistanceWeighted* are copied from Definition 7.1 except for the fact that suitable weighting factors p_k have been incorporated. To complete the definition we only have to give the auxiliary EDW

$$EDW^k([\langle u\rangle\sigma]^{h,q}, [\langle v\rangle\sigma']^{g,q}) = min(p_k \times d(u, v) + EDW^{k+1}([\sigma]^{h-1,q}, [\sigma']^{g-1,q}),$$
$$p_k \times d(u, \epsilon) + EDW^{k+1}([\sigma]^{h-1,q}, [\langle v\rangle\sigma']^{g,q}),$$
$$p_k \times d(\epsilon, v) + EDW^{k+1}([\langle u\rangle\sigma]^{h,q}, [\sigma']^{g-1,q}));$$
$$EDW^k([\langle u\rangle\sigma]^{h,q}, [\ ]^{0,q}) = p_k \times d(u, \epsilon) + EDW^{k+1}([\sigma]^{h-1,q}, [\ ]^{0,q});$$
$$EDW^k([\ ]^{0,q}, [\langle v\rangle\sigma']^{g,q}) = p_k \times d(\epsilon, v) + EDW^{k+1}([\ ]^{0,q}, [\sigma']^{g-1,q});$$
$$EDW^k([\ ]^{0,q}, [\ ]^{0,q}) = 0$$

with $k, h, g \in \mathbb{N}$, $q \in Q$ and $u, v, [\sigma]^{h-1,q}, [\sigma']^{g-1,q}$ marked traces;

The parameter k in EDW^k indicates the position at which the next edit action takes place. Please note that the recursive definition of EDW^k is well defined because at least one of the right–hand sides of the equation is one symbol shorter than the corresponding left–hand side (therefore, the fact that k is increasing causes no problem). The base case is $EDW^k([\ ]^{0,q}, [\ ]^{0,q}) = 0$. We will illustrate how the *EditDistanceWeighted* function is working. Let us consider two strings ab and c, (although *EditDistanceWeighted* is defined on marked traces, for making the example more understandable, let us show how it works on common strings). For transforming one string into another, there are five possible combinations: 1)$(ab, c\epsilon)$, 2)$(ab, \epsilon c)$, 3)$(\epsilon ab, c\epsilon\epsilon)$, 4)$(a\epsilon b, \epsilon c\epsilon)$, 5)$(ab\epsilon, \epsilon\epsilon c)$. The *EditDistanceWeighted* will be the minimum from the costs of the five combinations. The cost for every combination is computed as the sum of the edit actions multiplied with the corresponding weights. For example, the combination $(ab\epsilon, \epsilon\epsilon c)$, which means two insertions followed by one deletion, we encounter weighting factors for $d(a, \varepsilon)$, $d(b, \varepsilon)$ and $d(\epsilon, c)$, which are p_1, p_2 and p_3, respectively. The cost of this combination will be $p_1 \times d(a, \varepsilon) + p_2 \times d(b, \varepsilon) + p_3 \times d(\varepsilon, c)$. In a similar style the cost of the other combinations is computed and the minimum is chosen for *EditDistanceWeighted*.

Example Let us consider the automaton from Figure 1. For this automaton the maximum width of the marked trace is 3. Let $p_k = \frac{1}{2^k}, k = 1, 2, \ldots$. Let d_L be the following label distance

$$d_L(x,y) = \begin{cases} 0 & x = y \\ \mid \frac{1}{4^{i+1}} - \frac{1}{4^{j+1}} \mid & x = a_i, y = a_j, i,j \in \mathbb{N} \\ 1 & \text{otherwise} \end{cases}$$

1. Let us consider the traces a_0e and $a_0[\langle bd\rangle]^{1,II}e$.

   $d(a_0e, a_0[\langle bd\rangle]^{1,II}e) = \frac{1}{3} \times (EditDistance(a_0e, a_0[\langle bd\rangle]^{1,II}e)) = \frac{1}{3} \times (AtomicDistance(\epsilon, [\langle bd\rangle]^{1,II})) = \frac{1}{3}$.

   The trace distance recognizes that the symbols a_0 and e from the first trace are present in the second trace.

2. *The cycling effect:*

   Let us take the traces $a_0[\langle bd\rangle]^{1,II}e$, $a_0[\langle bd\rangle\langle bd\rangle]^{2,II}e$ and $a_0[\langle bd\rangle\langle bd\rangle\langle bd\rangle]^{3,II}e$;

   $d(a_0[\langle bd\rangle]^{1,II}e, a_0[\langle bd\rangle\langle bd\rangle]^{2,II}e) = \frac{1}{3} \times (EditDistance(a_0[\langle bd\rangle]^{1,II}e, a_0[\langle bd\rangle\langle bd\rangle]^{2,II}e)) = \frac{1}{3} \times (EditDistanceWeighted([\langle bd\rangle]^{1,II}, [\langle bd\rangle\langle bd\rangle]^{2,II})) = \frac{p_2}{3} = \frac{1}{12}$;

   $d(a_0[\langle bd\rangle\langle bd\rangle]^{2,II}e, a_0[\langle bd\rangle\langle bd\rangle\langle bd\rangle]^{3,II}e) = \frac{1}{3} \times (EditDistance(a_0[\langle bd\rangle\langle bd\rangle]^{2,II}e, a_0[\langle bd\rangle\langle bd\rangle\langle bd\rangle]^{3,II}e)) = \frac{1}{3} \times (EditDistanceWeighted([\langle bd\rangle\langle bd\rangle]^{2,II}, [\langle bd\rangle\langle bd\rangle\langle bd\rangle]^{3,II})) = \frac{p_3}{3} = \frac{1}{24}$.

   When two marked traces are cycling more times through the same cycle, the values of the trace distance start to decrease.

3. *The reduction effect:*

   Let us take the traces a_0e, a_1e and a_2e

   $d(a_0e, a_2e) = \frac{1}{3} \times (EditDistance(a_0e, a_2e)) = \frac{d_L(a_0,a_2)}{3} = \frac{0.23}{3}$;

   $d(a_1e, a_2e) = \frac{1}{3} \times (EditDistance(a_1e, a_2e)) = \frac{d_L(a_1,a_2)}{3} = \frac{0.06}{3}$.

   When the label distance between the labels (which compose the marked traces) is decreasing, the trace distance is also decreasing.

8 Transforming the heuristics into a coverage

The trace distance formula depends on the label distance d_L which implements the *Reduction* heuristic and the weights p_k which implement the *Cycling* heuristic. On the other hand, by choosing for each automaton s a finite set $L_\varepsilon \subseteq L$ which is an ε_L–cover of L with respect to d_L and a cycling limit l_c, a finite set of marked traces $T \subseteq traces^m(s)$ can be obtained. This is done by the application of the *Cycling* and the *Reduction* heuristics on $traces^m(s)$ by taking $T =$*Ran(Cycling* $\circ$ *Reduction)*. Now for this T and using d we want to compute its ε–cover of $traces^m(s)$ so that we can compute the coverage $cov(T, traces^m(s))$ – see Definition 4.6. Intuitively ε should depend on ε_L and l_c. Its formula is given by Theorem 8.1.

The next theorems are about other properties of the trace distance d: Theorem 8.2 shows that for any desired ε it is possible to obtain an ε–cover by choosing a suitable cycle limit and a suitable label approximation, with other words that the metric space $(traces^m(s), d)$ is a metric space that is totally bounded. Theorem 8.3 shows that the distance d implements the *Reduction* and *Cycling* heuristics.

Theorem 8.1 *Let s be an automaton. Let L be the labelset of s and d_L the label distance defined on it. The metric space (L, d_L) is totally bounded and d_L has all its values* *in the range $[0,1]$. Let l_c be the cycle limit. Let p_k ($k = 1, 2, ...$) be a series of positive numbers such that $\sum_{k=1}^{\infty} p_k = 1$. Let l_m be the maximum of the width and z the maximum of the nesting depth of the marked traces from $traces^m(s)$. Let $L_\varepsilon \subseteq L$ be an ε_L–cover of L. Then the finite set T =Ran(Reduction $\circ$ Cycling) of traces obtained by the application of the two heuristics on $traces^m(s)$ is an ε–cover of $traces^m(s)$ with $\varepsilon = \varepsilon^z$ and $\varepsilon^0 = \varepsilon_L$; for $i = 1, ..., z$: $\varepsilon_c^i = \sum_{k=1}^{l_c} p_k \times (\max_{j=0,...,i-1}(\varepsilon^j)) + \sum_{k=l_c+1}^{\infty} p_k$; $\varepsilon^i = \max_{\text{cycles}=0,...,l_m}(\frac{\text{cycles}\times\varepsilon_c^i+(l_m-\text{cycles})\times\varepsilon_L}{l_m})$. (Without proof)*

The following theorem shows that the metric space $(traces^m(s), d)$ is a totally bounded metric space.

Theorem 8.2 *Let s be an automaton. Let L be the labelset of s and d_L the label distance defined on it. The metric space (L, d_L) is totally bounded and d_L has all its values in the range $[0,1]$. Let p_k ($k = 1, 2, ...$) be a series of positive numbers such that $\sum_{k=1}^{\infty} p_k = 1$. Let l_m be the maximum of the width and z the maximum of the nesting depth for the marked traces from $traces^m(s)$. Then for every ε a positive real number in the range $[0,1]$, there exists a cycling limit l_c and a label approximation ε_L with $\varepsilon_L = \sum_{k=l_c+1}^{\infty} p_k \leq \frac{\varepsilon}{2^z}$ such that the finite set T =Ran(Reduction $\circ$ Cycling) of traces obtained by the application of the two heuristics on* traces^m *is an ε–cover of $traces^m(s)$ and the metric space $(traces^m(s), d)$ is totally bounded. (Without proof)*

The following theorem shows that the trace distance d implements the *Cycling* and the *Reduction* heuristics, in the sense of Definition 4.5.

Theorem 8.3 *Let s be an automaton. Let L be the labelset of s and d_L the label distance defined on it. The metric space (L, d_L) is totally bounded and d_L has all its values in the range $[0,1]$. Let l_c be the cycle limit. Let p_k ($k = 1, 2, ...$) be a series of positive numbers such that $\sum_{k=1}^{\infty} p_k = 1$. Let l_m be the maximum of the width and z the maximum of the nesting depth for the marked traces from $traces^m(s)$. Then the distance d implements the Reduction and the Cycling heuristics. (Without proof)*

For the computation of the coverage we approximate the minimum ε_m from Definition 4.6 with the ε^z computed in Theorem 8.1. We will illustrate the computation of the coverage in the following example.

Example Consider the automaton from Figure 1. Let us fix the final state to be *IV*. Let us consider the reduced set $L_\varepsilon = \{a_0, b, c, d, e, f\}$ which is an ε_L–cover of the labelset L with $\varepsilon_L = 0.25$ (ε_L is computed with respect to the d_L defined in the example from Section 7.2). For this automaton the maximum width is $l_m = 3$ and the maximum nesting depth is $z = 2$. Let us fix the series $p_k = \frac{1}{2^k} (k \in \mathbb{N})$ and in the beginning $l_c = 1$.

Then the set of traces T which is obtained by the application of the heuristics *Reduction* and *Cycling* is an ε–cover of the whole set of traces $traces^m(s)$ with ε computed with the formula from Theorem 8.1 as

1. $l_c = 1$, $L_\varepsilon = \{a_0, b, c, d, e, f\}$
   $\varepsilon^0 = \varepsilon_L = 0.25$; $\varepsilon_c^1 = \sum_{k=1}^{1} p_k \times \varepsilon^0 + \sum_{k=l_c+1}^{\infty} p_k = \frac{0.25}{2} + \sum_{k=l_c+1}^{\infty} \frac{1}{2^k} = 0.63$;
   $\varepsilon^1 = max_{\text{cycles}=0,...,3}(\frac{\text{cycles}\times\varepsilon_c^1+(l_m-\text{cycles})\times\varepsilon_L}{l_m}) = 0.63$; $\varepsilon = \varepsilon^2 = 0.81$;
   The coverage is computed via Definition 4.6 and it is $cov(T, traces^m(s)) = 1 - \varepsilon = 0.19$;
2. $l_c = 2$, $L_\varepsilon = \{a_0, b, c, d, e, f\}$
   When we enlarge the set T to T' for $l_c = 2$ we find that $cov(T', traces^m(s)) = 0.51$;
3. $l_c = 1$, $L_{\varepsilon'} = \{a_0, a_1, b, c, d, e, f\}$
   When we enlarge the set T to T'' for $L_{\varepsilon''} = \{a_0, a_1, b, c, d, e, f\}$ we find that $\varepsilon_L'' = 0.06$ and that $cov(T'', traces^m(s)) = 0.29$.

In this example, one can see that the coverage increases more by adopting a higher value for the cycling limit than by using a larger label subset. Consequently, one can conclude that it is better to increase the cycling limit for obtaining a better coverage. But this is not always true because we defined specific values for p_k and d_L (for other values, to increase the label subset will be better).

It can be seen from this example that the monotonicity property required in [BTV91] that $T \subseteq T' \Rightarrow cov(T) \leq cov(T')$ is respected by our coverage. From an intuitive point of view this property is reasonable: if one wants a better coverage, one needs to generate more tests.

We want to prove it in the general case. For this we will make an assumption which is quite natural: for a label set L when we have L_ε and $L_{\varepsilon'}$ such that these sets are an ε_L–cover and respectively an ${\varepsilon'}_L$–cover of L, $L_\varepsilon \subseteq L_{\varepsilon'}$ then $\varepsilon_L \geq {\varepsilon'}_L$.

Theorem 8.4 *Let s be a minimal finite deterministic automaton. Let L be the labelset of s and d_L the label distance defined on it. The metric space (L, d_L) is totally bounded and d_L has all its values in the range $[0,1]$. Let l_c and l'_c be two cycle limits ($l_c \leq l'_c$). Let l_m be the maximum of the width and z the maximum nesting depth of the marked traces from $traces^m(s)$. Let p_k ($k = 1, 2, ...$) be a series of positive numbers such that $\sum_{k=1}^{\infty} p_k = 1$. Let $L_\varepsilon \subseteq L$ be an ε_L–cover of L and $L_{\varepsilon'} \subseteq L$ be an ${\varepsilon'}_L$–cover of L such that $L_\varepsilon \subseteq L_{\varepsilon'}$ and $\varepsilon_L \geq {\varepsilon'}_L$. Let T =Ran(Reduction $\circ$ Cycling) and T' =Ran(Reduction $\circ$ Cycling) be the two finite sets of traces obtained by the application of the two heuristics on $traces^m(s)$ using L_ε, l_c and respectively $L_{\varepsilon'}$, l'_c. Then $cov(T, traces^m(s)) \leq cov(T', traces^m(s))$. (Without proof)*

9 Conclusions

In this paper we have studied two heuristics for reducing the number of traces in a test suite. The underlying assumption is that when automatically generating a set of traces, many traces will show similar behaviour. Test traces can be deleted without essentially reducing the error detection power of the test suite.

The first heuristic studied deals with restricting the branching degree of the nodes, when representing a set of test cases as a finite automaton. The basic idea is that in practice a high branching degree is generated because at the branching point an action is allowed which is parameterized by an element from a (large) data domain. The observation is that only a few values from such a data domain will show essentially different behaviour.

The second heuristic concerns the number of times a cycle in a finite automaton representation may be traversed. This is connected to the assumption that only for a few numbers of traversals the test cases will show essentially different behaviour.

The fact that we studied only these two heuristics in this paper, does not mean that these are the only interesting heuristics. More heuristics can be defined, e.g. with respect to the general length of a trace and with respect to the uniformity of the number of outgoing transitions from a state. We embedded the two heuristics chosen in a more general framework which allows the extension of our work with other heuristics.

A heuristic is a general guideline for reducing test suites, which must be made more precise to be practically applicable. Especially for the cycling heuristic we had to introduce additional notation. The reason is that the cycling structure of a trace through a finite automaton must be made explicit. We introduced *marked traces* for this purpose, which enabled us to extend the work on cycle reduction by Vuong [ACV93, ACV97].

In order to introduce a notion of *coverage* for the test suites reduced by means of the above mentioned heuristics, we defined a *trace distance* on marked traces. The results of our studies can be used to effectively calculate the coverage of a test suite reduced with our techniques.

Although our formal definitions of *Reduction* and *Cycling* work on large, sometimes even infinite sets this need not cause practical or algorithmic problems. For example, the practical generation of traces could be done in a similar way to [ACV97], starting with a first trace and then suppress the generation of a subsequent trace if it is close to an already generated trace. Other solutions could be based on a suitable transformation of the automaton, as in fact we did in Figure 1. A similar remark applies to the calculation of the ε value for a generated test suite. A solution is to choose l_c and ε_L and calculate ε arithmetically using the results of Section 8.

One issue deserves attention, viz. the choice of representatives embodied in the algorithms of Sections 5.1 and 5.2. At first sight, the reader may think that the distance d defined in Section 7.2 is independent of the choice of the representatives. However, this does not hold in general because of the essential ambiguity in the concept of *cycle* when using an automaton as a specification.

The proposed test selection technique can be compared to the existing theories in this area. In particular, these are the hypothesis theory developed by [CG97] and the trace distance theory of [ACV93, ACV97]. The hypothesis theory embodies the trace distance theory (see [CG97]), but the nice thing about trace distance theory is that it gives a measure for the degree to which a reduced set of traces approximates the original one. So we chose an approach which combines these two theories. In our view, first the heuristics (test hypotheses in the theory of [CG97]) are to be defined. After that, based on these heuristics a trace distance is built. This gives the possibility to make a test selection with a given ε approximation. The change of the heuristics leads to the change of the trace distance used in test selection.

We have started work on implementing our techniques in the TorX tool environment ([BFV$^+$99]). An assumption for implementing our work is that a label distance exists. Because the TorX tools support the input of finite automata defined in LOTOS [Bri88], we defined a label distance on LOTOS labels. This is not trivial because LOTOS labels may be parameterized by arbitrary data types.

Another restricting requirement is that we assume the specification to be given as a minimal finite deterministic automaton. Some test generation tools already provide such a format, but others support general finite automata. Determinizing a finite automata may cost exponential time. In this case it would be interesting to know whether the theoretical results achieved in this paper could be extended to non deterministic automata.

References

[ACV93] J. Alilovic-Curgus and S.T. Vuong. A metric based theory of test selection and coverage. In A. Danthine, G. Leduc, and P. Wolper, editors, *Protocol Specification, Testing, and Verification*, volume XIII, pages 289–304. North-Holland, 1993.

[ACV97] J. Alilovic-Curgus and S.T. Vuong. Sensitivity analysis of the metric based test selection. In M. Kim, S. Kang, and K. Hong, editors, *Int. Workshop on Testing of Communicating Systems*, volume X, pages 200–219. Chapman & Hall, 1997.

[BFV$^+$99] A. Belinfante, J. Feenstra, R.G. Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, *Intenational Workshop on Testing of Comunication Systems*, pages 179–196. Kluwer Academic, 1999.

[Bri88] E. Brinksma. On the design of extended lotos. Phd. Thesis, University of Twente, Netherland, 1988.

[BTV91] E. Brinksma, J. Tretmans, and L. Verhaard. A framework for test selection. In B. Jonsson, J. Parrow, and B. Pehrson, editors, *Protocol, Specification, Testing, and Verification*, volume XI, pages 233–248. North-Holland, 1991.

[CG96] O. Charles and R. Groz. Formalisation d'hypothèses pour l'evaluation de la couverture de test. In *Actes du Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'96)*, Editions Hermes, 1996.

[CG97] O. Charles and R. Groz. Basing test coverage on a formalization of test hypotheses. In M. Kim, S. Kang, and K. Hong, editors, *Int. Workshop on Testing of Communicating Systems*, volume X, pages 109–124. Chapman & Hall, 1997.

[JM99] T. Jéron and P. Morel. Test generation derived from model-checking. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 108–121. Springer-Verlag, 1999.

[Mye79] G.J. Myers. The art of software testing. John Wiley & Sons Inc, 1979.

[SKGH97] M. Schmitt, B. Koch, J. Grabowski, and D. Hogrefe. Autolink – a tool for the automatic and semi-automatic test generation. In A. Wolisz, I. Schieferdecker, and A. Rennoch, editors, *Formale Beschreibungstechniken für verteilte Systeme*, volume 315. GMD-Studien, St. Augustin, GI/ITG-Fachgespräch, GMD, 1997.

[Ste92] G.A. Stephen. String search. Technical Report TR-92-gas-01, School of Electronic Engineering Science, University College of North Wales, 1992.

[Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. Software–Concepts and Tools, 17(3):103–120, 1996. Also: Technical Report No. 96-26, Centre for Telematics and Information Technology, University of Twente, The Netherlands.