21

SELEXPERT – a knowledge-based tool for test case selection

Abdelaziz Guerrouat[a], *Hartmut König*[b], *Andreas Ulrich*[a]
[a] *Otto von Guericke University, Department of Computer Science, P O Box 4120, 39016 Magdeburg / Germany*
e-mail: {guer, ulrich}@cs.uni-magdeburg.de
[b] *Brandenburg University of Technology, Institute of Computer Science, P O Box 101344, 03013 Cottbus / Germany*
e-mail: koenig@informatik.tu-cottbus.de

Abstract

In this paper, we present the knowledge-based tool SELEXPERT to support the selection of test cases derived from LOTOS specifications. SELEXPERT provides information to the test engineers about the quality of the test cases to estimate the consequences of their selection. The selection criteria are based on fault coverage and cost of the test case. We discuss how this knowledge is represented and describe the selection procedure for test cases derived from Basic and Full LOTOS. For the latter, a probabilistic approach to address the non-reachability issue is proposed. Examples from the INRES protocol are used to illustrate the test case selection procedure.

Keywords

Test case selection, conformance testing, knowledge-based tool, formal specification, LOTOS.

1 INTRODUCTION

In the communication protocol area, the conformance test aims to detect any abnormal behavior of an implementation with respect to its reference specification. Testing is carried out by applying test suites that are derived from the protocol specification. Many methods for test suite generation have been developed [Dudu91] [Fuji91] [Vuon89]. Most of them are based on finite state machines (FSMs). Since a protocol specification usually describes an infinite behavior, the generated test suite may be very large or even infinite. In practice, it is very hard or even impossible to apply a complete test suite for checking the conformance of an implementation. To hold the cost of testing in reasonable bounds, the execution of a test suite should lead

G. V. Bochmann et al. (eds.), *Formal Description Techniques VIII*

to a verdict in limited time. Therefore, a selection strategy for test cases to limit the number of tests to be executed is of large interest.

Test case selection depends mainly on two factors: the faults that can be detected by executing the test case and the cost for its execution [Brin91] [FMCT95]. The first factor reflects the quality of the test case with respect to its error-detecting capabilities. The second factor expresses the different efforts made in the test process for test case derivation, execution, and test result analysis.

The information needed to evaluate these factors usually requires a detailed knowledge about the conformance test process. However, the knowledge related to the type of faults frequently made in testing or to the estimation of the cost for executing specific actions cannot be only derived from the specification formalism. Additional heuristic knowledge about the specification context, e.g. the specific character of certain interactions of a protocol with its environment, is needed which has to be provided by human experts. A suitable way to capture such sort of knowledge is by means of a knowledge-based system.

In this paper, we present a knowledge-based tool, called SELEXPERT, to support the selection of test cases. SELEXPERT accumulates knowledge about the protocol specification, the test suite, the test cases, and the cost. It provides information about the quality of a test case to the test engineers in order to support them in estimating the consequences of their selection. The current implementation of the knowledge base of SELEXPERT is dedicated to protocol specifications in LOTOS. The test suites fed into SELEXPERT are derived by means of the interactive LOTOS simulator *smile* [Eijk91] (see Figure 1). They are deduced from the behavior tree that represents the output of the simulator. With the help of a human expert, properties of test cases related to their testing capability are evaluated. Finally, appropriate test cases that fulfill given selection criteria are offered by the tool.

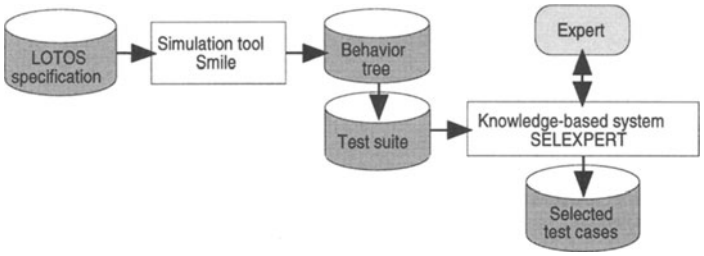

Figure 1 Overview of the test case selection environment.

The paper is organized as follows. Section 2 introduces the basic concepts of test case selection. Section 3 outlines the architecture of the knowledge-based system SELEXPERT and gives an overview about the knowledge representation formalisms. In Section 4, we describe the application of SELEXPERT for selecting test cases derived from specifications in Basic LOTOS. We use the INRES protocol to demonstrate the principle. Section 5 discusses the test case selection procedure based on Full LOTOS specifications. We present an approach to convert the specification into an intermediate form that allows the application of the selection principles applied for Basic LOTOS. Section 6 summarizes the main contributions of the paper.

2 BASIC CONCEPTS OF TEST CASE SELECTION

2.1 Preliminaries

Conformance testing is used to determine the conformance of an implementation to a standard test suite. The meaning of conformance depends on the implementation relation adopted [Boch91] [Tret93]. The relation defines what a correct implementation is. For deterministic specifications, the relation is usually given in terms of *trace equivalence*. In this case, the same traces of interaction sequences have to be provided by a valid implementation as defined in the specification. For non-deterministic specifications, many implementation relations have been defined, e.g. observation equivalence, bisimulation equivalences [Miln89], failure preorder [Brin88], generalized failure preorder [Lang89] and many others. We consider only deterministic LOTOS specifications in this paper. Therefore, we use the trace equivalence as implementation relation. We speak about a faulty implementation when this relation does not hold for the implementation with respect to its reference specification.

First, the protocol specification is simulated by means of the interactive LOTOS tool *smile* - a symbolic simulator of LOTOS specifications that proves their dynamic semantics [Eijk91]. In order to handle infinite behavior, a maximum unfold depth M is given before starting the simulation. The restriction of the unfold depth leads to a partial behavior tree, the output of *smile*, that reflects the correct behavior of the specification up to a sequence of M events in one trace of the tree. It is up to the user of *smile* to which unfold depth M he will generate the behavior tree.

The aim of the simulation should be a behavior tree that covers the simulated behavior almost completely. Here we meet the problem that real protocols due to the recursion of protocol parts mostly describe infinite behavior. To restrict the extent of the behavior tree we apply two different approaches: (1) a behavior tree is generated from a selected partial protocol specification or (2) the simulation continues at least until a recursive call to the same process happens that is just simulated.

The disadvantage of approach (1) is that a selected partial behavior expression can interact with other protocol parts. In this case, the generated behavior tree represents behavior parts that are not intended for simulation. Therefore, attention is required to choose an appropriate behavior expression. Approach (2) aims at the fact that it is almost exhausting to consider the complete control flow of the protocol once. Usually, both approaches together will be applied to generate a behavior tree that covers the simulated behavior expression sufficiently. The problems related to pruning a specification by simulation and their impact on test case derivation are discussed in a broader extent in [Ulri93].

After simulation, the behavior tree is interpreted as a test suite of the simulated LOTOS specification. Because only observable events can be executed in a test run, internal events obtained during the simulation process are omitted. This simplification is allowed if deterministic specifications are considered.

Definition (1): A test suite *TS* is a non-empty, finite set of traces σ of observable events corresponding to the behavior tree *B(S)* that is obtained from a given deterministic protocol specification *S* in LOTOS. The length of the traces in *TS*, is limited to the unfold depth M.

A *trace* is a branch of the test suite that always starts from the initial event and ends at a leaf-event of the tree. A test suite *TS* can be seen as a finite set of test cases *TC*. A test case describes a subtree in *TS*, i.e. it consists of a subset of traces from *TS*.

Definition (2): A test case *TC* of a test suite *TS* is a non-empty set of traces. The traces are the same as in the corresponding test suite *TS*. The number of traces in *TC* is less or equal to those in *TS*.

The maximum number of test cases of a test suite *TS* is described by all possible combinations of different traces in *TS* and each combination represents one possible test case. That seems to be realistic since all possible test cases that can be derived from the specification have to be treated. The maximum number of test cases that can be generated is equal to $2^{|B(S)|}-2$ where $|B(S)|$ represents the number of traces in the behavior tree $B(S)$.

2.2 Determining fault coverage

The capability of a test case to detect faults in faulty implementations is called *fault coverage* (or coverage for brevity). A faulty implementation may comprise one or several faults. A fault in an implementation is recognized by refusing an expected observable behavior. The fault model can be generated by modifying the correct behavior of the reference specification. The modified specifications are called *mutants* of the specification. Such mutants have been usually used for determining the fault coverage of test suites and for analyzing and comparing the fault coverage of various FSM based test selection methods [Prob90].

In our approach, the generation of mutants can be assisted by a human expert. Based on his experience, the expert is able to assess whether the implementation under test may behave as a specific mutant or not. Furthermore, expert knowledge is also used to assist the evaluation of the occurrence probability of faulty implementations. Figure 2 illustrates the process. Mutant generation is based on a finite, deterministic behavior tree in contrast to an FSM as suggested in [Boch91]. However, if the behavior tree represents all transitions in the specification exactly once, both approaches are interchangeable. In this case, our approach based on behavior trees is justified. Furthermore, it allows even a more flexible generation of mutants if transitions occur more than once.

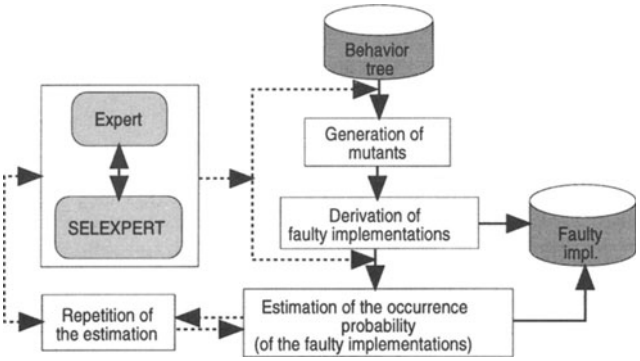

Figure 2 Mutant generation.

In literature, several methods for coverage measures are proposed [FMCT95]. In most of these methods, the *coverage* is based on the percentage of faults that a test suite can detect with respect to a certain fault model. In other works [Vuon91], the coverage is considered as a measure how good a test suite approximates an infinite set of tests that fully cover a given protocol specification. In this case, the coverage is defined based on the concept of distance in a metric space of execution sequences.

In SELEXPERT, the following formula based on the concept of the percentage of faults that can be detected [Boch91] [FMCT95] is used to measure the coverage of a test case.

$$COV_{TC} = \frac{\sum_{I \in I_F} P_{FAIL}(TC,I) * P(I)}{\sum_{I \in I_F} P(I)} \text{ with} \tag{1}$$

- $P_{FAIL}(TC,I)$ represents the probability that the execution of the test case TC for the implementation I produces the result $FAIL$, i.e. TC detects a fault in I. $P_{FAIL}(TC,I)$ is equal to 1 if the faulty implementation I contains a fault that can be detected by TC, otherwise it is equal to 0.
- I_F is a finite set of faulty implementations. The number of the faulty implementations that is considered here depends on the number of generated mutants. Their maximal number $|I_F|$ is equal to $2^{|F|}-1$ where $|F|$ represents the number of faults.
- $P(I)$ is the probability that the faulty implementation I occurs.

The probability of occurrence $P(I)$ of a faulty implementation I depends on the probability of the occurrence of faults contained in I. Suppose $F=\{F_1,F_2, \ldots ,F_n\}$ symbolizes the finite set of all possible faults that can independently occur in a faulty implementation, then $P(I)$ is determined as follows:

$$P(I) = \prod_{j=1}^{n} \delta_j{}^I \text{ with } \delta_j{}^I = \begin{cases} P(F_j) & \text{if } I \text{ contains } F_j \\ 1 - P(F_j) & \text{otherwise} \end{cases} \tag{2}$$

The probability of the occurrence of a fault F_j in a faulty implementation I can be computed by the following formula, assuming that the faults appear randomly. Note, n is the number of all possible faults, and m is the number of traces σ_j in the specification.

$$P(F_i) = \frac{\sum_{j=1}^{m} \alpha_{ij}}{\sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{ij}} \text{ with } \alpha_{ij} = \begin{cases} 1 \text{ if } F_i \in \sigma_j \\ 0 \text{ if } F_i \notin \sigma_j \end{cases} \tag{3}$$

A fault F_i is obtained by modification of an observable event in a trace. Observable events shared between different traces lead to the same faults. Thus, the occurrence probability of a fault depends on its occurrence in different traces.

2.3 Cost estimation

The cost of a test suite expresses a quantification of efforts needed to generate, maintain, implement, and execute a test suite. Hence, cost is a measure for the quality of a test suite; a low

cost expresses low efforts [FMCT95]. To reduce the cost of test execution, the number of test cases selected should be as minimal as possible.

The cost of a test case is determined by different factors, such as

- the cost of the equipment used for testing,
- the number of traces a test case is composed of,
- the average length of each trace,
- the time needed to execute the test case,
- the effort needed to implement a tool for executing the test case.

We assume that the cost of each test case related to these factors can be quantified by a human expert and that the cost consists of two parts: *fixed cost* and *execution cost*. The *fixed cost* includes the cost estimated for the implementation of the test equipment, its use, etc. To estimate the fixed cost, we use a top-down method [Boeh84] for general software development. It is based on deducing the total cost of a software product starting from its global features and properties. Then, the total cost is splitted into components of the product. However, the estimation usually does not deal with difficult technical aspects on the lower system levels.

On the other side, the bottom-up method is used to estimate the execution cost, which typically consists of (1) the cost of preamble execution, (2) the cost of postamble execution, and (3) the cost of test body execution. This method evaluates each component of a software product separately. The results are accumulated to deduce the cost of the whole product. This method is more precise but it supposes that all components are evaluated. In our approach, we apply both techniques to estimate the cost of a test suite.

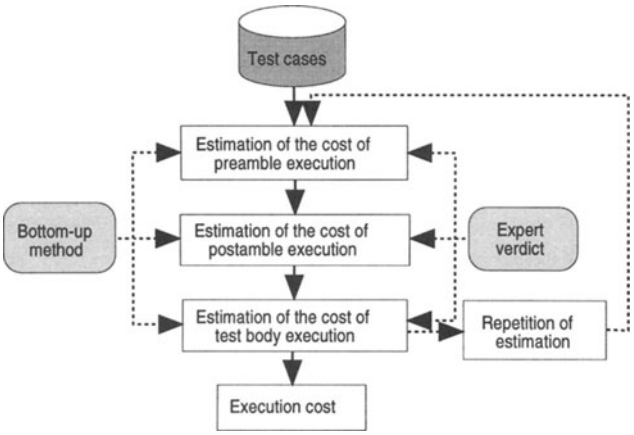

Figure 3 Estimation of execution cost using the bottom-up method.

Figure 3 illustrates the principle of applying the bottom-up method. The repetition of the estimation process assisted by one or more experts gives a more precise estimation. This principle is known as *groups agreement technique* [Boeh84]. It allows to deduce an average value for the cost from various intermediate values obtained by individual estimations of different experts or by repeating the estimation.

The total cost C_{TC} of a test case is the sum of the fixed cost C_{fix} and the execution cost C_{exec}.

$$C_{TC} = C_{fix} + C_{exec} \text{ with } C_{exec} = \sum_{\sigma \in TC} C_\sigma \tag{4}$$

The cost of a trace C_σ is defined as the sum of the cost of all observable events t.

$$C_\sigma = \sum_{t \in \sigma} C_t \tag{5}$$

The estimation of the cost of observable events requires specific knowledge from the expert about the difficulty of their testing and depends on various factors, e.g. the time needed to execute an event. The effort related to test events may differ from one event to another. For example, the analysis of a receive event consumes more time than the generation of a send event.

3 KNOWLEDGE-BASED SYSTEM SELEXPERT

3.1 System architecture

SELEXPERT is a tool built by means of *Nexpert Object*, a general-purpose knowledge-based development environment, which supports a large range of knowledge representation features [NeDa91]. The knowledge domain is modeled in terms of objects and production rules. Figure 4 gives an overview of the system.

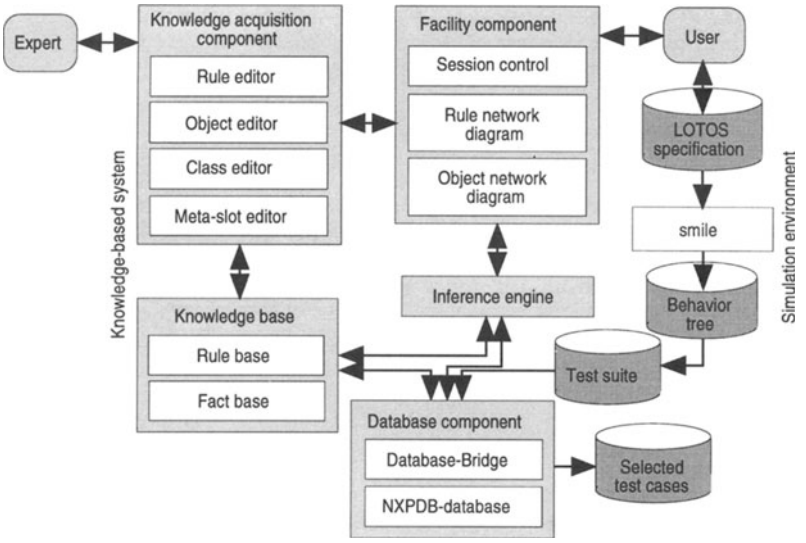

Figure 4 Knowledge-based system architecture and simulation environment.

SELEXPERT is functionally composed of five principal components: a knowledge base component, an inference engine, a knowledge acquisition component, a facility component and a database component. The *knowledge base component* consists of two units, the rule base and the fact base. The *rule base* contains all production rules that the inference engine uses during

the knowledge processing. The *fact base* consists of all elements modeled by the object concept that can be temporarily or permanently stored. The *inference engine* interprets the expert knowledge disposed in the knowledge base by applying an inference strategy. The inference strategy defines the order in which the rules have to be evaluated. Two inference techniques, *backward chaining* and *forward chaining*, can be used. The *knowledge acquisition component* consists of four basic modules: the *rule editor* for creating rules, the *object editor* for describing object structures, the *class editor* for creating a class and its relatives, and the *meta-slot editor* to define methods for the properties. The *facility component* is composed of three interfaces: the *rule network diagram* that gives a graphic representation of the growing rule base, the *object diagram* that shows the object structures in form of a hierarchical network, and the *session control* that lets the user interactively control the knowledge processing. The *database component* consists of a *database* that can be supported by the development environment and the database bridge. The *database bridge* guarantees the data transfer from external data sources and its transformation into the object structure within the fact base, and vice versa.

3.2 Knowledge representation

The knowledge representation in SELEXPERT is based on objects and production rules. The *object concept* represents the knowledge domain in terms of objects, classes and properties [Rumb91]. An *object* represents an instance of a class. A *class* defines the common characteristics shared by a family of related objects. A *property* is an attribute that can be associated with an object or class. The property values and their *methods* are stored in the so-called *slots* and can be inherited from classes and objects. A method consists of a sequence of actions that is executed under certain conditions during knowledge processing [Lunz94]. For computing the execution cost of a test case, for instance, it is assumed that an expert is asked first to supply the system with knowledge about the cost of single observable events. If the expert cannot accomplish the task, it is completed by the system itself. This is done by attaching methods that implement formula (4) and (5) to the property *cost* in the class of test cases.

The *production rules* capture the knowledge needed to solve particular domain problems in form of relations, heuristics, and procedural knowledge. A rule consists of (1) one or more conditions under which the rule is treated or *fired*, (2) one hypothesis that is inferred to be true if all conditions are satisfied, and (3) zero or more actions that are executed if the conditions are satisfied. Additionally, a production rule may have an inference priority that is used to solve conflicts between rules. The selection of a test case, for instance, can be performed by applying two different rules according to the importance of one of the properties coverage or cost. The rule corresponding of the most important property receives the highest inference priority. If the hypothesis is not confirmed by this rule, the rule with the lower priority is processed.

3.3 Gradual growth of the knowledge base

The benefit of applying a knowledge-based system in the test case selection process consists in its ability to exploit facts accumulated in previous knowledge processing sessions. This means, that a run of the knowledge-based system does not monotonously regenerate the same data obtained in former phases of the knowledge processing. A new session only generates and stores facts that were unknown in the previous ones.

SELEXPERT supports the gradual growth of the knowledge base by storing the actual facts in a database. In a later knowledge processing session for the same protocol, it is sufficient to

recall the required facts from the database. The knowledge in the database is retransformed into classes and objects which can be now used to produce new facts. Thus, the knowledge base can be progressively enriched with new facts about protocols and test suites.

4 TEST CASE SELECTION FOR BASIC LOTOS

4.1 Principle of test case selection

Using the object structure, a test case can be described as a class in SELEXPERT. The properties associated with the class describe the relevant properties of objects. The main properties are test case identifier, test purpose, behavior description, detected faults, coverage, fixed and execution cost.

The selection process is illustrated in Figure 5. It composes a subclass of test cases *exec_set* that meets the requirements related to fault detection capability and cost. The test suite is represented in the knowledge base by a set of objects of the class *test_case*. Each member TC_i of this class represents a test case. The test cases TC_i inherit the properties from their parent class *test_case*.

The selection procedure consists of two steps. In the first step, test cases that meet the coverage criterion are assigned to the temporal subclass *tmp_class*. In the second step, test cases of the subclass *tmp_class* that fulfill the cost criterion are linked to the final class *exec_set* which is a subclass of *tmp_class*.

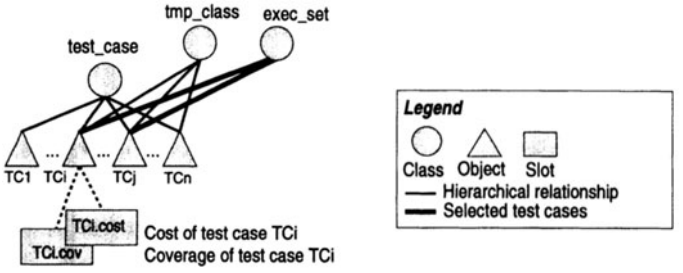

Figure 5 Test case selection based on coverage and cost.

Each test case indicated in *exec-set* fulfills the given selection criteria for coverage and cost. But it is possible that the set of selected test cases *exec_set* remains empty. In this case, we introduce a second principle that ranks all test cases according to coverage and cost. Then, the test cases are selected one after another and assigned to the subclass *exec_set* until the expected criterion for the coverage is obtained for this subclass. If the total cost of the selected test cases in *exec_set* is less or equal to the permitted cost, the complete set of test cases fulfills the selection criteria. Otherwise, the assignment of test cases to *exec_set* is repeated until the sum of permitted cost is reached. If the fault coverage of the test cases in *exec_set* provides the required fault detection capability, the indicated test cases are a possible solution of the selection process. If both attempts fail, there is no set of test cases that satisfies the given selection criteria.

4.2 Application to INRES protocol

To demonstrate the principle of test case selection, we take the INRES protocol [Hogr92] as an example. The INRES protocol provides a simple data transfer service over an unreliable medium. The Basic LOTOS specification of the protocol is given in [Hogr92]. The test case selection procedure for this protocol is outlined as follows.

Step 1 - Selection of a behavior expression: First, a behavior expression of the specification is picked out and simulated. We focus on the process *disconnection* of the responder part here.

```
process disconnection[IDISreq,DR,ICONind,IDATind,CR,DT,ICONresp]:noexit:=
        IDISreq;DR;
        responder[ICONind,IDATind,IDISreq,CR,DT,DR,CC,ICONresp]
endproc (* disconnection *)
...
```

Step 2 - Transforming the behavior tree into a test suite: In this step, the test suite is derived from the behavior tree generated by *smile*. (The unfold depth was set to 10.) The test suite is represented in a database format that can be manipulated by SELEXPERT. To make the example easier to understand, the test suite is pruned to four traces. The first column of the table below that represents the database indicates the trace identifier. The events that define a trace are quoted in the second column.

trace_identifier	event_sequence
tr1	IDISreq;DR;IDISreq
tr2	IDISreq;DR;CR;IDISreq
tr3	IDISreq;DR;CR;ICONind;IDISreq
tr4	IDISreq;DR;CR;ICONind;ICONresp;IDISreq

Step 3 - Determining the faults of the specification: The output of this step consists of two tables: the table of the faults where each fault corresponds to a mutant, and the table of the faulty implementations. These tables are not given here for lack of space. (The number of generated faulty implementations is 511.)

Step 4 - Derivation of test cases: The table shows the test cases which are automatically derived from the test suite. The first column indicates the test case identifier. In the second column, only the trace identifiers that compose the test case are given. The third column lists the faults that can be discovered by the test case.

testcase_ident	comprised_traces	detected_faults
tc1	tr1	F3,F2,F1
tc2	tr2	F5,F4,F2,F1
tc3	tr1,tr2	F5,F4,F3,F2,F1
tc4	tr3	F7,F6,F4,F2,F1
tc5	tr1,tr3	F7,F6,F4,F3,F2,F1
tc6	tr2,tr3	F7,F6,F5,F4,F2,F1
tc7	tr1,tr2,tr3	F7,F6,F5,F4,F3,F2,F1
tc8	tr4	F9,F8,F6,F4,F2,F1
tc9	tr1,tr4	F9,F8,F6,F4,F3,F2,F1
tc10	tr2,tr4	F9,F8,F6,F5,F4,F2,F1
tc11	tr1,tr2,tr4	F9,F8,F6,F5,F4,F3,F2,F1
tc12	tr3,tr4	F9,F8,F7,F6,F4,F2,F1
tc13	tr1,tr3,tr4	F9,F8,F7,F6,F4,F3,F2,F1
tc14	tr2,tr3,tr4	F9,F8,F7,F6,F5,F4,F2,F1

Step 5 - Selection of test cases: The selection process is supposed to cover all possible errors in an implementation. The total cost of all selected test cases should not exceed the factor of 30 cost units assuming equal cost for all events. Provided with this information, the knowledge-based system generates the following output.

```
testcase_ident|                test_purpose| coverage| cost|
*************************************************************
         tc13| test of traces: tr1,tr3,tr4 |    0.97 | 14.0|
         tc14| test of traces: tr2,tr3,tr4 |    0.97 | 15.0|
*************************************************************
```

There are two test cases selected. The first test case *tc13* is capable to detect all faults except fault *f5*, whereas test case *tc14* detects all faults except fault *f3*. These two test cases provide the highest coverage, and both test cases together are qualified to find all possible faults. The total cost of the two test cases is 29 and thus it is less then the given cost factor. That means, *tc13* and *tc14* fulfill the given selection criteria and represent a possible solution of the test case selection process.

5 TEST CASE SELECTION FOR FULL LOTOS

5.1 Basic principle

Now, we consider the test case selection for test suites derived from Full LOTOS specifications. The test case selection procedure becomes much more complicated because of the data flow to be considered [Higa92] [Verh91]. Additional problems are caused by non-reachable behavior parts. To guarantee the executability of the derived test cases, the non-reachable behavior parts have to be detected and eliminated before the test case selection process starts.

To find non-reachable parts, we transform the Full LOTOS specification into a structured labelled transition system. This transformation can be done either automatically by implementing the corresponding inference rules and axioms of the transition derivation system as defined in the LOTOS standard [ISO89] or by using LOTOS simulation tools.

Definition (3): A *structured labelled transition system SLTS* is defined as $<S,L\cup\{i\},A,T,s_0>$ with

- S is a set of *states* and $s_0 \in S$ the *initial state*.
- L is a set of *gates*.
- $i \notin L$ is the *internal event*.
- $A=<D,O>$ is a many-sorted algebra (D is a set of sorts and O is a set of operations);
- $T=\{t \mid t=s\text{—}\langle e,c\rangle\rightarrow s'\}$ is a *set of transitions* where
  - $s, s' \in S$;
  - e is a structured event that takes the form $g\alpha_1\alpha_2...\alpha_n$ with $g \in L$ and $\alpha_i \in D$;
  - c is a predicate associated to e.

Note that the *SLTS* is represented as a tree where the states S correspond to nodes in the tree, s_0 is the root node of the tree, and the transitions T, including the internal transitions, are the edges. Each edge represents a transition $t=s\text{—}\langle e,c\rangle\rightarrow s'$ and is labelled by an event e and a con-

dition c. We assume that transitions in the tree are symbolically represented, i.e. variables used in a transition are not resolved.

It is necessary that we have a specification whose behavior parts are reachable to deduce test cases whose execution is guaranteed. It is known from general software testing that the detection of non-reachable branches in a program is usually a non-decidable issue. In [Higa92] it has been shown, however, that with restriction to data type *integer* and to simple operations, this problem becomes decidable.

We propose here a probabilistic approach to evaluate the non-reachable behavior parts. The approach can be applied to all data types (*integer*, *real*, *boolean* and *string*) and their corresponding operations. Since exhaustive testing is not feasible for all possible values of variables, the number of values has to be confined. This is done in the following way (see Figure 6):

Suppose $V_S=\{v_1,v_2,...,v_n\}$ is the finite list of all variables in the specification and $R_S=\{R_1,R_2,...,R_n\}$ the corresponding list of their finite ranges. For each variable $v_i \in V_S$, a random value in the range R_i is generated. Now each boolean expression of a predicate is evaluated for the actual value of the variables and then the truth or falsity of the expression is stated. For the generated value, the execution of each transition in the *SLTS* is investigated. The execution of a transition t consists of the verification whether a trace that starts from the initial state s_0 and leads to the finale state of transition t exists for the given assignment of variables. The transition is reachable when all conditions associated to the predecessor transitions leading to transition t and the condition associated to t are fulfilled. This procedure is repeated until the number of values foreseen for the variable is reached and until all variables are treated. The operations for the non-reachability analysis are performed by rules in SELEXPERT.

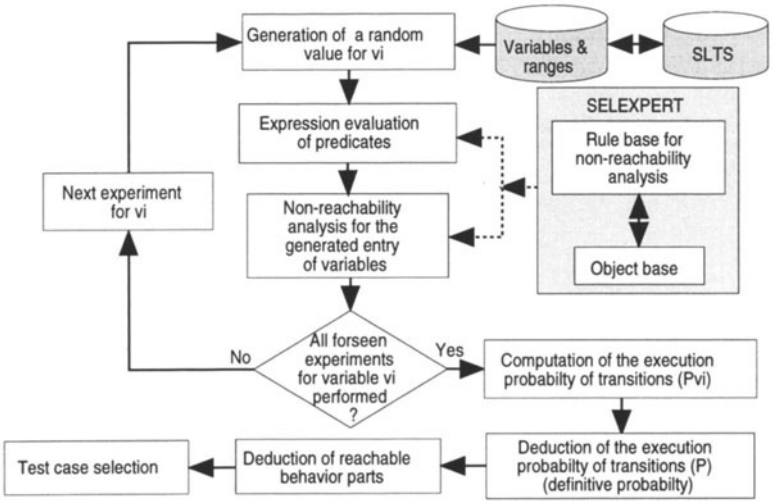

Figure 6 Procedure for detecting non-reachable behavior parts.

For a given variable v_i, the execution probability $P_{v_i}(t_{exec})$ of each transition $t \in T$ is

$$P_{v_i}(t_{exec}) = \frac{1}{n_{vi}} * \sum_{j=1}^{n_{vi}} m_{vi_j} \quad \text{where} \tag{7}$$

- n_{vi} represents the total number of values generated for variable v_i.
- m_{vi_j} indicates whether the transition t is executable for the value generated for variable v_i.

The sum of all m_{vi_j} indicates how frequent the execution of transition t has occurred.

By repeating the procedure for all variables $v_i \in V_S$, we get the following execution probability $P(t_{exec})$ for a given transition:

$$P(t_{exec}) = \frac{\sum_{v_i \in V_S} P_{v_i}(t_{exec})}{|V_S|} \quad \text{where } |V_S| \text{ represents the cardinal number of } V_S. \tag{8}$$

The non-execution probability of the transition $P(t_{\neg exec})$ is then:

$$P(t_{\neg exec}) = 1 - P(t_{exec}). \tag{9}$$

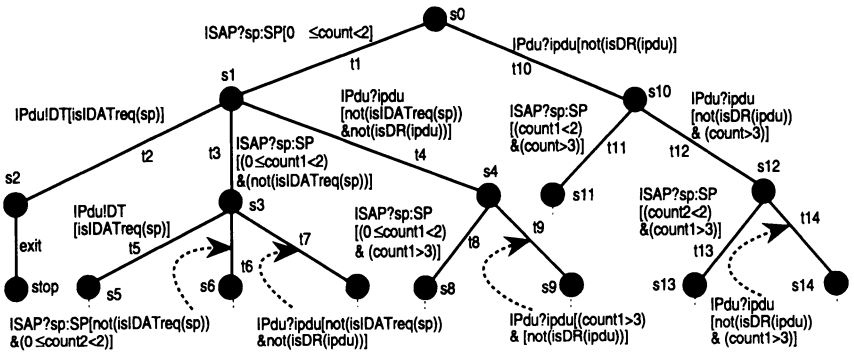

Figure 7 *SLTS* of the example.

5.2 Deriving a reduced specification

Based on the execution probability for each transition, a *reduced specification* can be deduced whose behavior parts are reachable. To obtain the reduced specification, we perform the following steps:

For a given ε (ε represents the smallest value of the execution probability indicating that a transition is considered to be executable.) with $0 \leq \varepsilon << 1$, eliminate transition t and all transition sequences outgoing from t in the *SLTS* of the specification if $P(t_{exec}) \leq \varepsilon$. The resulting behavior represents the reduced *SLTS* of the Full LOTOS specification whose behavior parts are reachable with a certain probability. Thus, all behavior parts of the obtained *SLTS* are executable.

Now, it is possible to derive a test suite that covers all behavior parts of the reduced *SLTS*. The principle of test case selection discussed in the previous sections for Basic LOTOS can be applied.

To illustrate the approach, we use the *readytosend* process of the INRES Full LOTOS specification [Hogr92]. The corresponding *SLTS* is depicted in Figure 7.

In Figure 7, the set of variables used in the behavior expression is V_S={*count,sp,ipdu*}. We assume R_{count}=[0,...,5], R_{sp}=[*IDISind,IDISreq,IDATreq,IDATind*] and R_{ipdu}={*CR,DT,DR*} as the ranges of *count*, *sp* and *ipdu*, respectively. To detect the non-reachable behavior parts of the behavior expression, 80 values for these variables were generated. The execution probability for all transitions is represented in Figure 8.

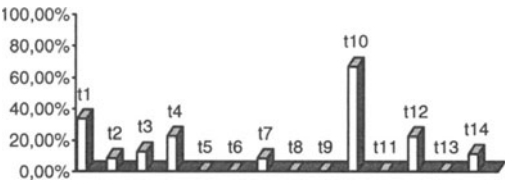

Figure 8 Execution probability of the transitions.

The evaluation shows that there are transitions whose execution probability is equal to zero. These are transitions *t5,t6,t8,t9,t11* and *t13*. To obtain the reduced specification, all non-executable transitions and their following transitions have to be eliminated. For ε=0, the reduced *SLTS* is depicted in Figure 9.

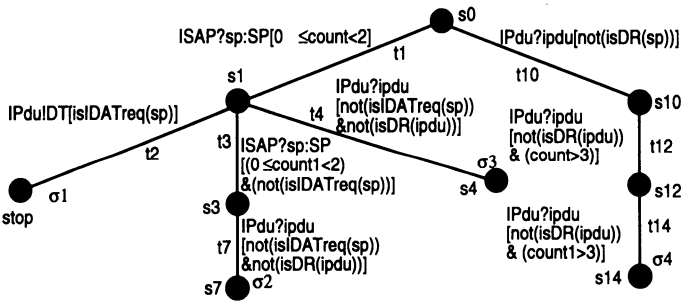

Figure 9 The reduced *SLTS*.

A test suite can be easily derived that covers all traces of the reduced *SLTS*. For this purpose, values for the variables can be deduced that allow to reach each final state of the reduced *SLTS*. These values exist because the probability of executing the transitions is greater than zero.

The following set of test cases *TS*={*TC1,TC2,TC3,TC4*} represents a test suite that covers all traces of the reduced *SLTS*:

- *TC1: ISAP?IDATreq;IPdu!DT* corresponds to trace σ1 (constraint *count*=0).
- *TC2: ISAP?IDISind;ISAP?IDATind;IPdu?DT* corresponds to trace σ2 (constraint *count*=1).
- *TC3: ISAP?IDATind;IPdu?DT* corresponds to trace σ3 (constraint *count*=1).

- *TC4: IPdu?CR;IPdu?DT;IPdu?DR* corresponds to trace σ4 (constraint *count*=5).

5.3 Assessment of the approach

The probabilistic approach allows to detect non-reachable behavior parts for the data types used in the specification, and to reduce the specification to derive an executable test suite. However, the process for detecting non-reachable parts is time-consuming because the ranges of variables and the number of transitions are usually very large. The efficiency and reliability of the approach depends on the number of values generated for the given variables. To make the approach practicable, the following assumptions have to be made:

(1) The *SLTS* is finite and the number of transitions is not very large. This assumption can be met by giving a maximum unfold depth for the *SLTS* in the simulation phase and by selecting only small behavior expressions, e.g. a process of the specification. In the latter case, we suppose that the initial state of the behavior expression is reachable through an executable transition sequence (a preamble).

(2) The ranges of variables are finite and of reasonable size. This assumption is derived from heuristics developed in software testing [Myer79] that can be used here. They restrict the test to a limited number of values for a given range, e.g. to marginal values in a range and some other values chosen randomly.

6 CONCLUSIONS

We have presented the tool SELEXPERT to assist the test case selection process using knowledge-based techniques. It offers appropriate possibilities to gather all information required for test case selection from human experts and the specification itself. We have shown how this information can be represented using the basic knowledge representation paradigms production rules and objects. The knowledge base is dedicated to protocol specifications written in LOTOS. The principles of the approach, however can be also applied to specifications in other formal languages.

The selection procedure for test cases derived from Basic LOTOS specifications is based on fault coverage and cost. It supports the selection of test cases which are optimal related to both factors or only one of them. The knowledge-based system provides means to represent the knowledge about the fault model and mutants needed to evaluate the coverage of a test case. The cost estimation considers not only the length of a test case. It also allows to take into account other aspects like the use of test equipment. Nevertheless, it remains difficult to estimate the cost correctly. For practical applications, further experience is needed. SELEXPERT supports a stepwise knowledge acquisition which is very helpful in this process.

For Full LOTOS, the application of the outlined approach is more difficult because the interaction parameters enormously increase the number of possible faults. To apply the test selection approach, a specification must be simplified as much as possible. This simplification mainly requires to eliminate all non-executable parts from the specification and to consider small ranges of data values. For this purpose, we have proposed a method based on a probabilistic approach. After that, feasible test cases can be selected that cover all reachable behavior parts.

The main advantage of a knowledge-based system is its ability to support the acquisition as well as the representation of different kind of knowledge needed to evaluate the testing capability of test cases. Different knowledge representation paradigms and other components of the

knowledge-based system provide this feature. They make test selection more flexible and user-friendly. The knowledge base is built by accumulating new facts from external sources, e.g. directly from a human expert or from other databases, or by processing already existent knowledge. It can be exploited for different protocols that present common properties.

REFERENCES

[Boch91] G. v. Bochmann, et al.; *Fault Models in Testing*; Proc. 4th IWPTS, Leidschendam, 1991.

[Boeh84] B. W. Boehm; *Software Engineering Economics*; IEEE Transactions on Software Engineering, 10(1), 1984.

[Brin88] E. Brinksma; *A theory for the derivation of tests*; Proc. PSTV VIII, Leidschendam, 1988.

[Brin91] E. Brinksma, J. Tretmans, L. Verhaard; *A Framework for Test Selection*; Proc. PSTV XI, Stockholm, 1991.

[Dudu91] M. Duduc, R. Dssouli, G. v. Bochmann; TESTL: *An Environment for Incremental Test Suite Design Based on Finite-State Models*; Proc. 4th IWPTS, Leidschendam, 1991.

[Eijk91] P.H.J. v. Eijk; *Tools for LOTOS, a Lotosphere overview*; Memoranda informatica 91-25, University of Twente, 1991.

[FMCT95] ISO/IEC; *Working Document on Formal Methods in Conformance Testing*; Paris, 1995.

[Fuji91] S. Fujiwara, et al.; *Test Selection Based on Finite-State Models*; IEEE Transactions on Software Engineering, 17(6), 1991.

[Higa92] T. Higashino, et al.; *Automatic Analysis and Test Case Derivation for Restricted Class of LOTOS Expressions with Data Parameters*; Proc. 5th IWPTS, Montréal, 1992.

[Hogr92] D. Hogrefe; *OSI formal specification case study: the Inres protocol and service, revised*; Universität Bern, Institut für Informatik, 1992.

[ISO89] ISO/IEC; *LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior*; IS 8807, 1989.

[Lang89] R. Langerak; *A Testing Theory for LOTOS using Deadlock Detection*; Proc. PSTV IX, Amsterdam, 1989.

[Lunz94] J. Lunze; *Künstliche Intelligenz für Ingenieure* (in German); Oldenbourg Verlag, Munich, 1994.

[Miln89] R. Milner; *Communication and Concurrency*; Prentice-Hall; 1989.

[Myer79] G. Myers; *The art of software testing*; John Wiley & Sons Inc., 1979.

[NeDa91] NEURON DATA, Inc.; *Nexpert Object Version 2.0, User's Guide*; California, 1991.

[Prob90] R. Probert, F. Guo; *E-MPT Protocol Testing: Preliminary Experimental Results*; Proc. 3rd IWPTS, McLean, New Jersey, 1990.

[Rumb91] J. Rumbaugh et al.; *Object-Oriented Modeling and Design*; Prentice-Hall; New Jersey; 1991.

[Tret93] J. Tretmans; *A Formal Approach to Conformance Testing*; Proc. IWPTS VI, Pau, 1993.

[Ulri93] A. Ulrich, H. König; *Test Derivation from LOTOS using Structure Information*; Proc. IWPTS VI, Pau, 1993.

[Verh91] L. Verhaard; *Test Selection in Conformance Testing*; Memoranda Informatica, 91-52, University of Twente, 1991.

[Vuon89] S.T. Vuong, W. L. Chan, M. R. Ito; *The UIOv-Method for Protocol Test Sequence Generation*; Proc. 2nd IWPTS, Berlin, 1989.

[Vuon91] S.T. Vuong, J. Alilovic-Curgus; *On Test Coverage Metrics for Communication Protocols*; Proc. 4th IWPTS, Leidschendam, 1991.