3

Transformation of Estelle modules aiming at test case generation

O. Henniger[a], A. Ulrich[b], and H. König[c]

[a] *GMD – German National Research Center for Information Technology*
Rheinstr. 75, 64295 Darmstadt, Germany
e-mail: henniger@darmstadt.gmd.de

[b] *Dept. of Computer Science, University of Science and Technology*
Clear Water Bay, Kowloon, Hong Kong
e-mail: ulrich@cs.ust.hk

[c] *Dept. of Computer Science, Technical University of Cottbus*
P.O. Box 101344, 03013 Cottbus, Germany
e-mail: koenig@informatik.tu-cottbus.de

Abstract

This paper presents a method for transforming an extended finite state machine (EFSM) given as an Estelle normal form module into an equivalent expanded EFSM without control variables, i.e. an Estelle normal form module free of provided-clauses. The transformed EFSM allows to apply methods based on the finite state machine (FSM) model for test case generation. Using this approach, it is possible to cope with test sequence generation for control and data flow and with test data selection. The transformation is feasible if the variables that occur in provided-clauses have finite, countable domains. For realistic protocol specifications, this condition is fulfilled most of the time.

Keywords

Conformance testing, formal specifications, Estelle

1 INTRODUCTION

Early work on automatic test sequence generation for communication protocols has been based on the model of finite state machines (FSMs) (e.g. [NT81, SD88, ADLU88, SLD89, CVI89]). A main problem of these test sequence generation methods is that FSMs usually model only the control aspect of a system. To model the data aspect as well as the control aspect, extended finite state machine (EFSM) models, which are based on an FSM extended by variables, are applied in many cases. EFSM models form the basis of

A. Cavalli et al. (eds.), *Protocol Test Systems VIII*

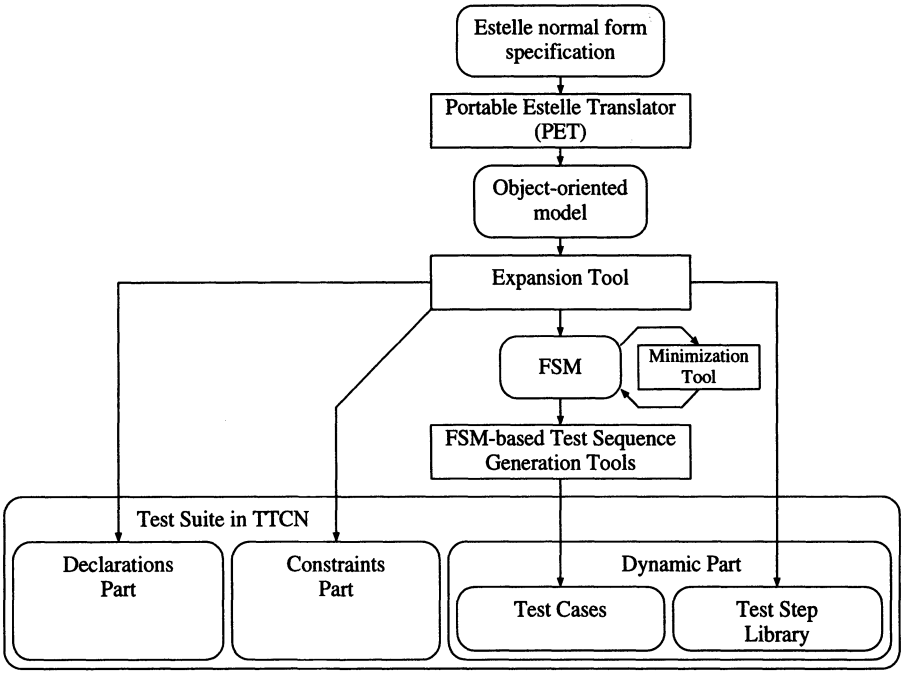

Figure 1 Test generation approach.

the standardized formal description techniques (FDTs) Estelle [ISO89] and SDL [ITU92] that increasingly come into use.

Theoretically, provided that all variables of an EFSM have a finite, countable domain, an EFSM can be transformed into an equivalent FSM. The transformation leads to the removal of variables from the state machine and an increase in the number of states. Practically, since the number of states of the resulting FSM may get very large, the transformation of an EFSM into an equivalent FSM is not feasible.

This paper discusses an approach to bridge the gap between an EFSM and an FSM avoiding inordinate increase in the number of states. Our aim is to make FSM based methods for test generation and for validation applicable to EFSM. The approach is discussed on the basis of an one-module Estelle normal form specification [Sar93] representing an EFSM.

Our approach is based on the observation that variables influencing the control flow usually have a small, finite domain (Boolean type, enumerated type, or subrange type), and that only these variables cause serious problems in applying FSM based methods. We transform an EFSM into an equivalent expanded EFSM where enabling of transitions depends only on the current state and the input, i.e. variables do not influence the control flow. That means we obtain an Estelle normal form specification without provided-clauses. The resulting specification still contains variables which, however, are not used

in provided-clauses. Therefore, it still represents an EFSM and not a pure FSM. The expanded EFSM can be interpreted in terms of FSM.

From the expanded EFSM, test sequences can be generated using classic methods, such as the transition tour method [NT81] or a UIO method [SD88, ADLU88, SLD89, CVI89]. The transformation algorithm does not cause loss of information. This allows to generate test sequences that cover control and data aspects of the original EFSM.

Beside test generation, the expanded EFSM is also useful for analyzing the original EFSM and for detecting specification errors.

The transformation algorithm has been implemented as a prototype expansion tool based on the PET&DINGO tool set [SS90]. Input to the expansion tool is the object-oriented model of an one-module specification produced by the Portable Estelle Translator (PET). Output of the expansion tool is an FSM representation of the source specification, a test step library and parts of the declarations and constraints parts in the test notation TTCN.MP [ISO92]. Figure 1 shows a scheme of our test generation tool set.

The rest of this paper is organized as follows: Section 2 defines the prerequisites necessary for the transformation algorithm. It gives formal definitions of an FSM and an EFSM, a definition of an Estelle normal form specification, its link to EFSMs, and a classification of EFSM variables. Section 3 introduces the algorithm for transforming a given EFSM into an expanded EFSM, discusses the interpretation of the expanded EFSM in terms of FSM and FSM based test sequence generation methods. Section 4 discusses issues arising from test generation in the context of multi-module specifications. In Section 5 our approach is demonstrated for the Inres protocol specification. Section 6 shortly reviews related work dealing with test generation from EFSM, and Section 7 gives some concluding remarks.

2 PRELIMINARIES

2.1 Finite state machines and extended finite state machines

Definition 1 A *finite state machine (FSM)* is a tuple $\langle S, I, O, T, s_0 \rangle$, where S is a non-empty finite set of states, I is a non-empty finite set of inputs, O is a non-empty finite set of outputs, $T \subseteq S \times I \times O \times S$ is the transition relation, and $s_0 \in S$ is the initial state of the FSM. $\diamond$

A transition $t \in T$ of an FSM is a tuple $\langle s, i, o, s' \rangle$ where $s \in S$ is a current state, $i \in I$ is an input, $o \in O$ is an output related to s and i, and $s' \in S$ is a next state related to s and i.

To enhance the descriptive power, additional variables are introduced into the mathematical model. These variables are used in programming language constructs specifying conditions for the execution of transitions and calculations carried out during transitions. For extending a conventional finite state machine by variables, such model is called extended finite state machine.

Definition 2 An *extended finite state machine (EFSM)* is a tuple $\langle S, C, I, O, T, s_0, c_0 \rangle$ where S is a non-empty finite set of main states, $C = \mathrm{dom}(v_1) \times \ldots \times \mathrm{dom}(v_n)$ is a non-empty countable set of contexts with $v_i \in V$, V is a non-empty finite set of variables and

$\mathrm{dom}(v_i)$ is a non-empty countable set which is referred to as the domain of v_i, I is a non-empty finite set of inputs, O is a non-empty finite set of outputs, $T \subseteq S \times C \times I \times O \times S \times C$ is the transition relation, $s_0 \in S$ is the initial main state, and $c_0 \in C$ is the initial context of the EFSM. ◇

A context is a concrete assignment of values to the variables. A transition $t \in T$ of an EFSM is a tuple $\langle s, c, i, o, s', c' \rangle$ where $s \in S$ is a current main state, $c \in C$ is a current context, $i \in I$ is an input, $o \in O$ is an output, $s' \in S$ is a next main state, and $c' \in C$ is a next context.

The mathematical structure EFSM can be expressed using different syntactic constructions, e.g. using modules or processes of the FDTs Estelle [ISO89] or SDL [ITU92].

2.2 Estelle normal form specification

Estelle uses a subset of ISO Pascal which is complemented by special constructs for expressing the elements of EFSM transitions, for structuring, and for expressing communication concepts. In Estelle, a system is specified as a hierarchy of module instances that communicate with each other via FIFO channels. The behavior of a single module instance is characterized in terms of an EFSM.

Starting point of our test generation approach is a specification in a specification style that makes the interpretation of module instances in terms of EFSM easy. A specification of this style is referred to as a normal form specification. In certain cases, specifications based on other specification styles may be transformed into normal form specifications by means of syntax-directed transformation rules [SB85, Sar93]. Main characteristics of a normal form specification are:

- The influence of variables on the control flow is specified only in provided-clauses, i.e. the specification contains no conditional-statements ("if", "case", or "forone" statements) and no repetitive-statements ("repeat", "while", "for", or "all" statements).
- The specification is complete, i.e. it contains no "any" constant-definitions, no "..." type-definitions, and no "external" and "primitive" directives.
- The specification has a static structure, i.e. it contains "init", "connect", "attach", "release", "terminate", "disconnect", and "detach" statements only in the initialization-part of module definitions.
- Shorthand notations, like nested transitions, "provided otherwise", state sets etc., are expanded.

Furthermore, we assume that the specification is deterministic and does not contain priority-clauses.

2.3 Classification of variables in an Estelle normal form specification

The variables occurring in a module definition contained in an Estelle normal form specification can be classified as context variables and interaction variables according to the place of their declaration.

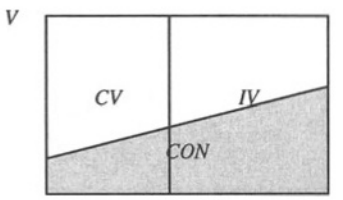

Figure 2 Visualization of the variable classes.

Definition 3 A variable that is declared within the declaration-part of a body-definition is called *context variable.* ◇

Definition 4 A variable that is declared within a value-parameter-specification of an interaction-definition contained in a channel-definition is called *interaction variable.* ◇

The set of all context variables and the set of all interaction variables of an Estelle module are called CV and IV respectively. Each variable belongs either to CV or to IV; i.e., CV and IV together form the set of all variables V of an Estelle module: $V = CV \cup IV$.

Furthermore, context variables as well as interaction variables can belong to the class of control variables. Control variables influence the selection of transitions.

Definition 5 A variable that occurs in a provided-clause is a *control variable.* A variable that is used to assign a value to a *control variable* is a *control variable* itself. ◇

The set of all control variables of an Estelle module is called CON. The set of all control variables can be found using a recursive algorithm based on the given recursive definition of a control variable. CON is a subset of the set of all variables: $CON \subseteq CV \cup IV$. Figure 2 depicts the relationship between the classes of variables.

2.4 Interpretation of an Estelle module in terms of an EFSM

The EFSM described by a module definition in an Estelle normal form specification is the tuple $\langle S, C, I, O, T, s_0, c_0 \rangle$ with $S = SID$ where SID is the set of state-identifiers introduced within the declaration-part of the body-definition, $C = \mathrm{dom}(v_1) \times \ldots \times \mathrm{dom}(v_n)$ with $v_i \in CV \cup IV$ where CV and IV are the sets of context variables and interaction variables as explained in Section 2.3, $I = \{iref \in IREF \mid iref \text{ is used in a when-clause}\}$ and $O = \{iref \in IREF \mid iref \text{ is used in an output-statement}\}$ where $IREF$ is the set of interaction-references, T represented by the set of transition-declarations TR, $s_0 = initial$ where *initial* is the state-identifier specified in the from-clause contained in the initialization-part, and c_0 represented by the values assigned in the transition-block contained in the initialization-part.

An interaction-reference $iref_i \in IREF$ may be associated with interaction variables as parameters: $iref_i(iv_{i,1}, \ldots, iv_{i,m_i})$. The interaction variables associated with $iref_i$ make up the set $IV_i \subseteq IV$.

A transition-declaration $tr \in TR$ has the following form:

trans from *current* **to** *next* **when** *input* **provided** *predicate*
begin *assignments*; *outputs* **end;**

where

- *current* $\in$ *SID* and *next* $\in$ *SID* are state-identifiers specifying the start state and the end state of a transition;
- *input* $\in$ *IREF* is an interaction-reference (possibly associated with interaction variables as parameters) specifying the input of a transition; alternatively to the when-clause, a delay-clause could be given to designate a time-out event as input;
- *predicate*$(v_1, \ldots, v_k)$ is a Boolean expression of the variables $v_1, \ldots, v_k$ specifying the enabling condition for a transition;
- *assignments* is a (possibly empty) sequence of assignment-statements specifying that variables are set to new values during execution of a transition;
- *outputs* is a (possibly empty) sequence of output-statements specifying the output caused by execution of a transition.

Since the Boolean expression in the provided-clause may be true for several contexts, a single Estelle transition-declaration comprises in general several EFSM transitions.

3 EXPANSION OF EFSM

3.1 Outline of the transformation approach

The FSM based test sequence generation methods are not easily applicable to EFSMs. In contrast to the transitions of an FSM, the transitions of an EFSM depend not only on the input and the current state, but also on the actual values of variables, which may depend on the whole record of previous inputs. Application of FSM based test sequence generation methods taking into consideration only the main states, but neglecting provided-clauses and the variables contained in them would lead to infeasible subtours.

In order to apply FSM based methods, the EFSM should be transformed into an equivalent FSM. An EFSM $\langle S, C, I, O, T, s_0, c_0 \rangle$ could theoretically be transformed into an equivalent FSM $\langle S', I, O, T', s_0' \rangle$ by means of Cartesian multiplication, i.e. $S' = S \times C$. Carrying out this transformation, one is faced with the state-explosion problem, i.e. with a very large or even infinite set of states. If all variables have a finite domain, S' is a finite set, and an equivalent FSM exists. If any of the variables does not have a finite domain, S' is an infinite set, and the EFSM can not be transformed into an equivalent FSM.

Since not all variables must have a finite domain, the transformation of an EFSM into an equivalent FSM is in general not applicable. However, taking into consideration the different classes of variables in Estelle, an approach can be established which limits the growth of the number of states.

We studied example specifications (e.g. [Hog92]) and realistic protocol specifications in Estelle (e.g. [HHP93, GHLP93, Häh94]) and observed that in many cases

- only a subset of all variables are control variables, and

- the control variables have small finite domains, i.e. they have the Boolean type, an enumerated type, or a subrange type with few values.

Our transformation approach is based on this observation, and the transformation algorithm is applicable only for specifications that satisfy the prerequisite that each control variable $c \in CON$ has a finite domain. Though not all variables must have a finite domain, at least the variables from the set of control variables usually have a finite domain.

Our approach is to eliminate only the control variables since the existence of control variables and provided-clauses is the main problem that hinders us from applying FSM based methods. Variables that do not influence the control flow can still remain in the transformed state machine; they do not bother us. The control variables are eliminated by shifting them to a set of new states S' or to a set of new inputs I'.

For determining the new set of states S', only the context variables that are control variables need to be taken into consideration. A new state is a combination of a main state of the original EFSM and of a concrete assignment of values to all context variables that are control variables.

For determining the new set of inputs I', interaction variables that are control variables are taken into consideration. A new input is a combination of the original input and of a concrete assignment of values to all parameters associated with this input that are control variables.

Once S' and I' are determined, we work out the final states and outputs of the new transitions by symbolic evaluation of the original transitions. The output parameters are partly replaced by concrete values during symbolic evaluation.

After carrying out the transformation, the transformed EFSM can be interpreted in terms of an FSM.

3.2 Transformation algorithm

This section describes the transformation algorithm in detail. Input to the algorithm is a module definition contained in an Estelle normal form specification and representing an EFSM as defined in Section 2.4. Output is an Estelle module definition without provided-clauses representing an expanded EFSM without control variables. The expanded EFSM is semantically equivalent to the original EFSM; no information is lost during the transformation.

Algorithm *Expansion*

Input: Estelle module definition representing an EFSM M with the set of state-identifiers SID, the set of interaction-references $IREF$, the subsets of variables CV, IV, and CON, and the set of transition-declarations TR.

Output: Estelle module definition without provided-clauses representing the expanded EFSM M'.

begin

{Computation of new states}

$ST = SID \times \mathrm{dom}(cv_1) \times \ldots \times \mathrm{dom}(cv_k)$ such that $cv_i \in CV \cap CON$ for $1 \leq i \leq k$;

{Computation of new input interactions}
$INT = INT_1 \cup \ldots \cup INT_l$ such that
$INT_i = \{iref_i\} \times \text{dom}(iv_{i,1}) \times \ldots \times \text{dom}(iv_{i,m_i})$, $iref_i \in IREF$ for $1 \leq i \leq l$, and
$iv_{i,j} \in IV_i \cap CON$ for $1 \leq j \leq m_i$;

{Computation of new transitions}
for each $tr \in TR$ **do**
  for each $st = \langle sid, \text{value}(cv_1), \ldots, \text{value}(cv_k)\rangle \in ST$ **do**
    for each $int = \langle iref_i, \text{value}(iv_{i,1}), \ldots, \text{value}(iv_{i,m_i})\rangle \in INT$ **do**
      if $((sid = current) \wedge (iref_i = input) \wedge$
        $(predicate(\text{value}(cv_1), \ldots, \text{value}(cv_k), \text{value}(iv_{i,1}), \ldots, \text{value}(iv_{i,m_i})) = \text{true}))$
      then
        begin
          Create a new transition-declaration $tr' \in TR'$ such that
            - $current' = \text{stateid}(st)$;
              {stateid(st) is a unique state-identifier assigned to st}
            - $input' = \text{intref}(int)$;
              {intref(int) is a unique name assigned to int}
            - $predicate' = \text{true}$;
              {i.e., the provided-clause can be omitted}
            - $next' = \text{stateid}(\langle next, \text{newvalue}(cv_1), \ldots, \text{newvalue}(cv_k)\rangle)$;
              {newvalue(cv_i) is the value of cv_i after evaluation of *assignments*}
            - $outputs' = outputs$ with actual parameters replaced by their symbolic values after evaluation of *assignments*;
            - $assignments' = assignments$ with assignment-statements having a control variable on their left-hand side omitted
        end;
end.

ST is the generalized Cartesian product of the set of state-identifiers SID and of the domains of all context variables that are control variables. Each ordered n-tuple $st \in ST$ consists of a state-identifier $sid \in SID$ and of a concrete value from the domain of each variable $cv_i \in CV \cap CON$. Because of the prerequisite that each control variable has a finite domain, the cardinality of ST (i.e. the number of states of the expanded EFSM) is a finite number: $|ST| = |SID| \cdot |\text{dom}(cv_1)| \cdot \ldots \cdot |\text{dom}(cv_k)|$.

INT is a set of ordered n-tuples that consist of an interaction-reference $iref_i \in IREF$ and of concrete values from the domains of all control variables iv_j that are declared in the interaction-definition for $iref_i$.

Since concrete values are given for all control variables, it is no problem to compute a concrete value for $next'$. If variables that are not control variables are used in a transition-block, the computation of actual output parameters and of new values for these variables will lead to symbolic values, i.e. expressions with variable names, arithmetic operators, and constants instead of concrete values.

All control variables are taken into account while computing the sets of new states and of new interactions; therefore, they are removed from the expanded EFSM. The transition-declaration-part TR' of the expanded EFSM consists only of transition-declarations that

do not contain a provided-clause, and that contain assignments, if any, only to variables that do not influence the control flow.

3.3 Interpretation of the expanded EFSM in terms of FSM

A transition-declaration of the transformed module definition $tr' \in TR'$ has the following form:

trans from *current'* **to** *next'* **when** *input'*
begin *assignments'*; *outputs'* **end;**

Only remaining variables and the statement-part between "begin" and "end" offend against the definition of an FSM. Because there are no provided-clauses, the output of the transformation algorithm can easily be interpreted in terms of an FSM $\langle S, I, O, T, s_0 \rangle$.

The state-identifiers correspond to the states of the FSM. Input interaction-references, possibly still associated with interaction parameters that were not used in the provided-clauses of the original specification, are mapped to FSM inputs by disregarding the parameters. The statement-parts of transition-blocks are mapped to FSM outputs as a whole. Statement-parts with equivalent sequences of output-statements correspond to the same FSM output.

Since no information should be lost by interpreting the expanded EFSM as an FSM, our expansion tool generates a declarations part, a constraints part, and a test step library at the same time as the FSM is generated.

3.4 Minimization of FSM

Given an FSM $\langle S, I, O, T, s_0 \rangle$, two states $s_1 \in S$ and $s_2 \in S$ are referred to as equivalent states if each sequence of inputs produces identical sequences of outputs in s_1 and s_2. An FSM without equivalent states is a minimal FSM. Each FSM can be reduced to a minimal FSM by applying minimization algorithms merging equivalent states to one state [HU79].

The FSM obtained by interpreting the expanded EFSM as an FSM may contain equivalent states. Minimization reduces the number of states and makes the state-explosion problem less severe.

An FSM $\langle S, I, O, T, s_0 \rangle$ is called a complete FSM if the transition relation is defined for each pair $\langle s, i \rangle \in S \times I$, otherwise it is a partial FSM. In case of partial FSMs, the classic minimization algorithms fix a next state and an output for pairs $\langle s, i \rangle$ for which no next state and no output have been defined originally. In the context of conformance testing, this means that additional conformance requirements are introduced that have not been given in the original specification. Since this must be avoided, minimization is carried out only for complete FSMs.

3.5 Test generation based on FSM

The different test sequence generation methods based on FSMs have a common basic idea [BU91]: A test sequence is a preferably short sequence of consecutive transitions that contains every transition of the FSM at least once and allows to check whether every transition is implemented as defined. To test a transition, one has to apply the input

for the transition in the starting state of the transition, to check whether the correct output occurs, and to check whether the correct next state has been reached after the transition. Checking the next state might be omitted (transition tour method [NT81]) or be carried out by means of distinguishing sequences (checking experiments method), characterizing sequences (W-method), or unique input/output sequences (UIO methods [SD88, ADLU88, SLD89, CVI89]).

Different FSM based test sequence generation methods may be applied to the FSM interpretation of the expanded EFSM. In case of large expanded EFSM, the transition tour method is most appropriate.

4 MULTI-MODULE SPECIFICATIONS

So far, our presented approach is able to deal with single-module specifications only. On the other hand, a system may be specified as a set of communicating modules. To apply our approach in this case, it is necessary to combine the communicating modules together. The composition is normally carried out by assuming synchronous communication between modules [SLU89]. If we use multi-module Estelle specifications as input to our approach, we slightly change the semantics of Estelle, which uses asynchronous communication over unbounded buffers. The changes in the semantics may be accepted if only modules are combined which will be implemented locally on a single processor system.

While combining modules to a composite machine, we are faced with another kind of state space explosion caused by interleaving events. To alleviate this problem, two different approaches are suggested in the literature: The first approach takes advantage of the hierarchical structure of the specification [SLU89], and the second one tries to prune the modules of a set of communicating modules before combining them according to a certain test purpose [LSKP93]. Whereas the first approach can not guarantee a feasible size of the composite module, in the second approach the module composed from a set of pruned machines may not represent the correct behavior under some circumstances.

In case of truly concurrent modules, execution of test cases generated from a composite machine becomes difficult because of weak controllability of the implementation under test. The test suite may require to test a certain order of interleaving events, but this order depends on a concrete test run and is in general non-deterministic. Thus, the entire concept of testing concurrent systems must be reconsidered. Currently, we investigate a general approach to test generation and execution of concurrent systems. The test generation will be based on partial order semantics [UC95].

5 RESULTS OF THE APPLICATION TO THE INRES PROTOCOL

We demonstrate the transformation algorithm on the Estelle specification of the Inres protocol [Hog92]. The Inres protocol is used as demonstration example for many FDT based methods for test case generation [FMC95]. The Inres protocol provides a simple data transfer service over an unreliable medium. Only the initiator side of the Inres protocol is considered. The initiator side consists of two modules, "Initiator" and "Coder". The two module definitions were merged into one, and transformation rules for obtaining a normal form specification were applied.

In the declaration-part of the module definition three context variables are introduced:

```
var olddata : ISDUType;
    counter : 0..4;
    number : Sequencenumber;
```

The context variables `counter` and `number` are used in provided-clauses. Searching for variables that are used to assign values to these two variables, only `counter` and `number` themselves are found. Thus, these two context variables are control variables and will be eliminated by the algorithm. `olddata` is used for buffering data and will not be eliminated. It does not matter whether or not such a variable has a finite domain.

In the channel-definitions, some interaction variables are introduced:

```
channel ISAPchn(User, Station);
    by User :
        ICONreq; ICONresp; IDATreq(ISDU : ISDUType); IDISreq;
    by Station :
        ICONconf; ICONind; IDATind(ISDU : ISDUType); IDISind;
channel MSAPchn(Station, Medium_Service);
    by Station :
        MDATreq(id : PduType; num : Sequencenumber; data : ISDUType);
    by Medium_Service :
        MDATind(id : PduType; num : Sequencenumber; data : ISDUType);
```

Only the interaction variables `id` and `num` belonging to the interaction `MDATind` are used in provided-clauses. Thus, these two interaction variables are control variables and will be eliminated as well.

With the following type-definitions, all control variables have a finite domain:

```
type PduType = (CR, CC, DT, AK, DR);
type Sequencenumber = 0..1;
```

Thus, the prerequisite for the applicability of our transformation algorithm is satisfied for the Inres example.

The first step of the algorithm delivers a set of new states by combining the original states with concrete values for all context variables that are control variables. The following set of states is given:

```
state DISCONNECTED, WAIT, CONNECTED, SENDING;
```

Combining the original states with concrete value assignments for `counter` and `number`, we obtain the following new set of states*:

```
state
    SENDING_0_0, SENDING_1_0, SENDING_2_0, SENDING_3_0, SENDING_4_0,
```

*The notation `SENDING_0_0` refers to `SENDING` combined with `counter` = 0 and `number` = 0, etc.

```
SENDING_0_1, SENDING_1_1, SENDING_2_1, SENDING_3_1, SENDING_4_1,
CONNECTED_0_0, ..., CONNECTED_4_1,
WAIT_0_0, ..., WAIT_4_1,
DISCONNECTED_0_0, ..., DISCONNECTED_4_1;
```

The next step of the algorithm delivers new interaction-definitions for interactions associated with control variables as parameters by combining these interactions with concrete value assignments for the control variables. So, `MDATind` is modified as follows[†]:

```
MDATind_CR_0(data : ISDUType);
MDATind_CR_1(data : ISDUType);
..
MDATind_DR_1(data : ISDUType);
```

In the last step of the transformation algorithm, the new states and the new interactions are utilized to replace each transition-declaration by a number of new transition-declarations without provided-clauses. Let us consider the following transition-declaration as an example:

```
trans from SENDING to SENDING
    when MSAP.MDATind
        provided (id = AK) and (num <> number) and (counter < 4)
            begin
                counter := counter + 1;
                output MSAP.MDATreq(DT, number, olddata)
            end;
```

This transition-declaration is replaced by 8 new transition-declarations without provided-clauses:

```
trans from SENDING_0_0 to SENDING_1_0
    when MSAP.MDATind_AK_1(data)
        begin output MSAP.MDATreq(DT, 0, olddata) end;
trans from SENDING_0_1 to SENDING_1_1
    when MSAP.MDATind_AK_0(data)
        begin output MSAP.MDATreq(DT, 1, olddata) end;
...
trans from SENDING_3_1 to SENDING_4_1
    when MSAP.MDATind_AK_0(data)
        begin output MSAP.MDATreq(DT, 1, olddata) end;
```

The variables `data` and `olddata`, which do not influence the enabling of transitions, remain in the newly generated transitions. `data` is an interaction variable, whereas `olddata` is a context variable for buffering the last data package sent. Since there are no provided-clauses, the result of the transformation is suitable for the application of FSM based test

[†]The notation `MDATind_CR_0` refers to `MDATind` combined with `id` = `CR` and `num` = `0`, etc.

Table 1 Size of the transformed specifications

	Number of main states	Number of transitions
Normalized module	4	34
Expanded EFSM	40	410
Minimized FSM	18	185

sequence generation methods. Table 1 illustrates the influence of the transformation on the size of a specification. We applied test generation tools to the minimized FSM for the Inres initiator part. A transition tour [NT81] derived from the specification of the initiator part consists of 400 transitions.

Analysis of the expanded EFSM led to the revelation of some errors in the original specification, like unintended non-determinism. So, the transformation is also useful for the validation of a specification.

Since the variable `data`, which is left over after the transformation, occurs in the send events of the test suite, the test suite designer has to choose a suitable value for it. What a "suitable" value means, is up to the test suite designer. The variable `olddata` remains in the test suite and is defined as a test suite variable.

6 RELATED WORK

Test sequence generation methods based on FSM have been widely studied. The extension by a data portion complicates the test generation from EFSM: both, the control aspect and the data aspect have to be tested, and besides the generation of a sequence of test events, one has to cope with the selection of test data.

The existing methods for test generation from EFSM can be roughly classified into methods with explicit test purposes and methods with implicit test purposes: methods with explicit test purposes require information about the test purpose or the fault model for the generated test cases as input in addition to the EFSM (e.g. [GHN93, WL93]); methods with implicit test purpose assume test purposes for the generated test cases implicitly and usually do not require supplementary inputs in addition to the EFSM (e.g. [Sar93, CA90, PG90, UY91, MP92, CZ93]).

The methods with explicit test purposes require the test designer to choose what to test. Then the methods ensure that test cases consistent with the specification and the test purposes are generated. These methods offer much flexibility, but on the other hand, they require considerable manual efforts and do not guarantee systematic fault coverage. For methods with implicit test purposes, the picture is reversed: While requiring less manual efforts, they offer less flexibility in choosing what faults to generate test cases for.

The methods with implicit test purposes take a normalized one-module specification as their starting point. Main problem of these approaches is to guarantee that the test sequences are executable. Some subtours may be not executable if the enabling predicates (also called constraints) of transitions along the test sequence can not be satisfied with any input parameter values. In general, the executability problem is undecidable. However,

on condition that variables have finite domains, constraint satisfaction techniques may be applied to ensure the executability of test sequences [CA90, CZ93].

Most of the methods base the test suite structure on the EFSM transitions. This leads to a dependence of the resulting test suite on the specification style: For the same system, very different test suites may be generated depending on whether the EFSM is specified in a more state-oriented or a more data-oriented style.

The approach presented in this paper is founded on experiences from earlier work on test generation from EFSM and extends the existing methods in certain aspects: The approach tackles the problem of test generation for both control and data flow by transforming an EFSM into an expanded EFSM representing both control and data aspect of the original EFSM and allowing to apply FSM based test generation methods. FSM based test generation methods assume an implicit fault model and generate shortest possible test sequences with a guaranteed fault coverage with respect to this fault model. The test sequence generated from the expanded EFSM is always executable since the enabling conditions of transitions have been considered in the course of the transformation. Furthermore, the approach copes with selection of test data by exhaustive enumeration of the control variables, which are required to have finite domains. High fault detection power is traded for considerable length of the test sequence. The generated test sequence is to a great extent independent from the specification style.

7 CONCLUSIONS

Although a formal specification is theoretically the best starting point for the automatic generation of conformance test suites, the way from a given formal specification to a conformance test suite is not straightforward in practice. Based on an Estelle normal form specification, we have proposed an algorithm to transform each module body into an equivalent EFSM without provided-clauses. The approach is limited to specifications where variables influencing the control flow range over a small domain. For many realistic protocol specifications, however, this limitation is fulfilled.

The described approach has been implemented as a prototype tool. The tool was applied to a number of example specifications and to realistic protocol specifications developed in the area of fieldbus systems and wireless telecommunication systems, and the feasibility of the method has been shown.

ACKNOWLEDGEMENTS

The authors are grateful to Karsten Nickoll for his assistance in programming the transformation algorithm.

REFERENCES

[ADLU88] A.V. Aho, A.T. Dahbura, D. Lee, and M.U. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification,*

Testing, and Verification, VIII, pages 75–86, Atlantic-City, New Jersey, USA, 1988. Elsevier Science B.V. (North-Holland).

[BSV89] E. Brinksma, G. Scollo, and C.A. Vissers, editors. *Protocol Specification, Testing, and Verification, IX*, Enschede, The Netherlands, 1989. Elsevier Science B.V. (North-Holland).

[BU91] B.S. Bosik and M.U. Uyar. Finite state machine based formal methods in protocol conformance testing: from theory to implementation. *Computer Networks and ISDN Systems*, 22(1):7–33, 1991.

[CA90] W. Chun and P.D. Amer. Test case generation for protocols specified in Estelle. In Quemada et al. [QMV90], pages 191–206.

[CVI89] W.Y.L. Chan, S.T. Voung, and M.R. Ito. On test sequence generation for protocols. In Brinksma et al. [BSV89], pages 119–130.

[CZ93] S.T. Chanson and J.-S. Zhu. A unified approach to protocol test sequence generation. In IEEE INFOCOM'93 [IEE93], pages 106–114.

[FMC95] FMCT guidelines on "Test generation methods from formal descriptions", 1995. ITU-T Q.8/10 and ISO/JTC1/SC21/Project 54.2.

[GHLP93] F. Graner, O. Henniger, B. Lehnert, and A. Pöschmann. Estelle specification of the Lower Layer Interface (LLI) based on the ISP fieldbus specification. Technical report, Institute of Automation and Communication, Magdeburg, Germany, 1993.

[GHN93] J. Grabowski, D. Hogrefe, and R. Nahm. Test case generation with test purpose specification by MSCs. In O. Færgemand and A. Sarma, editors, *6th SDL Forum*, pages 253–266, Darmstadt, Germany, 1993. Elsevier Science B.V. (North-Holland).

[Häh94] J. Hähniche. Estelle specification of the PIA protocol. Technical report, Institute of Automation and Communication, Magdeburg, Germany, 1994.

[HHP93] J. Hähniche, E. Hintze, and A. Pöschmann. Portable implementation for PROFIBUS layer 7. Technical Report DFAM AIF No. 357D, University of Magdeburg, Magdeburg, Germany, 1993. In German.

[Hog92] D. Hogrefe. OSI formal specification case study: the Inres protocol and service, revised. Technical Report IAM-91-012, University of Bern, Bern, Switzerland, 1992. Update 1992.

[HU79] J.E. Hopcroft and J.D. Ullmann. *Introduction to automata theory, languages, and computation.* Addison-Wesley, Reading, Mass., USA, 1979.

[IEE93] *IEEE INFOCOM'93 Conference on Computer Communications*, volume 1, 1993.

[ISO89] Information processing systems – Open Systems Interconnection – Estelle: A formal description technique based on an extended state transition model. International Standard ISO 9074, 1989.

[ISO92] Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN). International Standard ISO/IEC 9646-3, 1992.

[ITU92] Specification and description language SDL '92. ITU-T Recommendation Z.100, 1992.

[LSKP93] D. Lee, K.K. Sabnani, D.M. Kristol, and S. Paul. Conformance testing of protocols specified as communicating FSMs. In IEEE INFOCOM'93 [IEE93], pages 115–127.

[MP92] R.E. Miller and S. Paul. Generating conformance test sequences for combined control and data flow of communication protocols. In R.J. Linn and M.U. Uyar, editors, *Protocol Specification, Testing, and Verification, XII*, Lake Buena Vista, Florida, USA, 1992. Elsevier Science B.V. (North-Holland).

[NT81] S. Naito and M. Tsunoyama. Fault detection for sequential machines by transition-tours. In *11th IEEE Fault Tolerant Computing Conference*, pages 238–243, 1981.

[PG90] M. Phalippou and R. Groz. From Estelle specifications to industrial test suites, using

an empirical approach. In Quemada et al. [QMV90], pages 175–190.

[QMV90] J. Quemada, J. Mañas, and E. Vázquez, editors. *Formal Description Techniques, III*, Madrid, Spain, 1990. Elsevier Science B.V. (North-Holland).

[Sar93] B. Sarikaya. *Principles of protocol engineering and conformance testing.* Ellis Horwood Ltd., Hemel Hempstead, Herts., G.B., 1993.

[SB85] B. Sarikaya and G. v. Bochmann. Obtaining normal form specifications for protocols. In L. Csaba, K. Tarnay, and T. Szentiványi, editors, *Conference on Computer Networking, COMNET'85*, pages 601–612, Budapest, Hungary, 1985. Elsevier Science B.V. (North-Holland).

[SD88] K.K. Sabnani and A.T. Dahbura. A protocol testing procedure. *Computer Networks and ISDN Systems*, 15(4):285–297, 1988.

[SLD89] Y.-N. Shen, F. Lombardi, and A.T. Dahbura. Protocol conformance testing using multiple UIO sequences. In Brinksma et al. [BSV89], pages 131–143.

[SLU89] K.K. Sabnani, A.M. Lapone, and M.U. Uyar. An algorithmic procedure for checking safety properties of protocols. *IEEE Transactions on Communications*, 37(9), 1989.

[SS90] R. Sijelmassi and B. Strausser. NIST integrated tool set for Estelle. In Quemada et al. [QMV90].

[UC95] A. Ulrich and S.T. Chanson. An approach for testing distributed software systems. In *Protocol Specification, Testing, and Verification, XV*, Warsaw, Poland, 1995.

[UY91] H. Ural and B. Yang. A test sequence selection method for protocol testing. *IEEE Transactions on Communications*, 39(4):514–523, 1991.

[WL93] C.-J. Wang and M.T. Liu. Automatic test case generation for Estelle. In *International Conference on Network Protocols*, pages 225–232, 1993.