# A uniform test derivation method for Estelle, LOTOS and SDL

*Hans van der Schoot and Hasan Ural*

*University of Ottawa*
*Department of Computer Science*
*Ottawa, Ontario, Canada, K1N 6N5*
*E-mail: ural@csi.uottawa.ca*

## 1 Name of the method

The test derivation method outlined here is called the *dfc-method* (data-flow-chains-method).

## 2 Purpose of the method

The dfc-method is intended for the *selective* derivation of test cases from a formal description of the system under test (SUT). The targetted selective derivation is based on the analysis of the *input-output (I/O) relations* in a given specification for exposing the effects of each input interaction on those output interactions which are influenced by the input interaction. Such effects are identified by tracking variables through the specification of the SUT, following them as they are modified, until they are ultimately used to compute values for other variables. The method requires that each of these effect is examined at least once during testing.

The intuition behind the selective derivation of test cases based on the coverage of effects of inputs on outputs (i.e. I/O-functionality) is that faults in a system may lead to incorrect values and, as the result of propagation through computations, an error may show up at the output of the SUT.

## 3 FDTs supported

The dfc-method is a uniform test derivation method for all three standardized FDTs, namely Estelle, LOTOS, and SDL.

## 4 Test notation supported

The dfc-method does not encompass a particular test description language, such as TTCN, for expressing its final output. The output of the dfc-method consists of sequences of actions in LOTOS and sequences of input/output interactions in Estelle and SDL, which must be further processed to be expressed in a specific format of a test description language.

## 5 Restrictions on the FDTs

Some syntactic restrictions must be imposed on each FDT to make the dfc-method applicable to the FDT. We enumerate these restrictions below for each FDT individually.

### 5.1 Estelle

The dfc-method assumes an Estelle specification given as the description of a collection of modules and procedures, where

a) there are no WITH statements;

b) every transition is a simple transition, i.e. a transition which is not nested and does not contain ANY clauses, but consists of an optional WHEN clause, a FROM clause containing a specific identifier for the present state, an optional PROVIDED clause containing a boolean expression, a TO clause containing a specific identifier for the next state, an optional DELAY clause, an optional PRIORITY clause, and a transition block;
c) every simple transition is named and the output statement(s) are considered as the last statement(s) in the statement part of each simple transition.

These restrictions do not limit the applicability of the dfc-method to Estelle in practice since
a) each WITH statement is eliminated by concatenating the record name to the beginning of the identifier of each variable that stands for a field of the record within the scope of the WITH statement;
b) section 7.5.2.4 of the Estelle Standard defines two mappings that can be used to obtain a set of simple transitions from a transition that is either nested or contains ANY clauses. Moreover, one of these mappings (i.e. the mapping from a nested transition to a set of simple transitions) also replaces a PROVIDED clause containing "otherwise" by a PROVIDED clause containing the negation of an appropriate boolean expression;
c) implicit references to state identifiers (e.g. "same" in a TO clause or state-set-identifier in a FROM clause) are eliminated by replicating the simple transition as many times as necessary.

### 5.2 LOTOS
The following syntactic restrictions must be imposed on LOTOS specifications in order to apply the dfc-method:
a) process instantiations must have relabelling function *Id*, i.e. the identity function;
b) recursive process instantiations must be tail recursive, i.e. recursive instantiations may not occur on the left-hand-side of the enable operator and the disable operator;
c) no generalized parallel constructs (par expressions) may occur;
d) no generalized choice constructs (sum expressions) over gates may occur.

These restrictions are necessary to ensure termination of the flowgraph construction algorithm and, consequently, the construction of a finite flowgraph from a LOTOS specification.
It should be noted that the above restrictions do not necessarily limit the application of the dfc-method to LOTOS in practice. Violations of the above cases can be easily avoided in the definition of a new specification. As for the INRES specification in LOTOS, all the constraints listed above are indeed satisfied.

### 5.3 SDL
The dfc-method assumes that an SDL specification is given as the description of a collection of processes and procedures, where each process or procedure is described as a collection of state transitions defined in terms of the constructs of basic SDL. However, since section 4 of the SDL standard can be used for the transformation of SDL shorthand notations into constructs of basic SDL, this assumption does not place any restriction on the application of the dfc-method to SDL in practice.

## 6 Definition of the method
The dfc-method consists of four steps. The first three steps are concerned with the input-output relation analysis, which is facilitated by tracing the flow of both control and data in a given specification. The final step deals with the selection of the (executable) test sequences which capture the identified IO-functionality.

STEP 1: *Constructing a flowgraph*
For ease of tracing the flow of both control and data, a given Estelle, LOTOS, or SDL specification is transformed into a *flow graph* as the first step of the dfc-method. A flow graph is a digraph in which a node represents a collection of the syntactic elements (possibly a single element) in the specification and an edge represents the possible flow of control between nodes.

The algorithmic construction of a flowgraph from a formal description in Estelle, LOTOS and SDL is described in [4, 5], [2, 3] and [7], respectively. Each transformation preserves those aspects of the original description that are necessary for test derivation purposes. For instance, a large part of the syntactic information is retained by using different types of nodes and edges to identify the various syntactic elements in a specification.

STEP 2: *Classifying variable occurrences*
Data flow in a specification can be traced through associations among variables in the specification. To achieve this, each occurrence of every variable in a specification is first classified as being a definition or a use. A *definition* (*def*) of a variable is an occurrence of the variable by which it takes a value. A *use* of a variable is an occurrence of the variable by which its value is referenced. In addition to this classification, the relative positions of the defs and uses within the specification is determined by associating each def and use with the appropriate node or edge in the flowgraph of the specification.
The precise details of classifying variable occurrences as defs and uses for each FDT are reported in [2, 4, 5, 7]. It should be noted that, prior to their classification, variables in a specification may have to be renamed to avoid global scope conflicts that might arise.

STEP 3: *Establishing data flow associations*
In order to establish the associations among defs and uses of variables in a specification, two types of relation between variable occurrences are identified, based on the classification performed in step 2. The first type of relation exists between a def of a variable and each use of that same variable. Such a relation is represented as a 2-tuple, called *def-use pair*. The second type indicates relations between a use of a variable and a def of a different variable. These so-called causal relations appear when a variable is referenced (a use) to define another variable (a def), for instance in an assignment statement.
Tracing a sequence of computations which capture particular data flow associations among variables is now carried out by forming *data-flow-chains* (*df-chains*). A df-chain is a sequence of du-pairs in the flowgraph of the specification linked by means of the existing causal relations among the various defs and uses. Each df-chain exposes a data flow association between a def of a variable and a use of a possibly different variable which is influenced by the def.
The particular df-chains of interest are those that start with *inputs* and terminate with *outputs*. The rationale here is to identify the functionality of the SUT in terms of the effects of data accepted from its environment on data offered to its environment. Inputs and outputs are pointed out as particular kinds of defs and uses, respectively [3, 5, 7]. Accordingly, associations are established between each input and every output that is influenced by that input. Each such IO-association, called an *input-output-data-flow-chain* (*IO-df-chain*), thus is a chain of def-use pairs which starts with an input and ends with an output influenced by that input.

STEP 4: *Selecting test cases*
As the last step of the dfc-method, test cases are selected to cover each IO-association in a specification at least once. It is important to stress that the number of such IO-associations in a specification is finite due to some extra requirements we put on forming df-chains [2, 3, 4, 5, 7]. As a consequence, the generated test suite is finite.
Selecting a test case to cover an IO-association in an Estelle or SDL specification is rather straightforward. There exists a one-to-one mapping from an IO-association, or IO-df-chain, in these specifications to a sequence of input and output interactions. This sequence is generated by extracting the appropriate input/output interactions, associated with the nodes and edges in the flowgraph in terms of defs and uses, from the path in the flowgraph that covers the IO-df-chain.
A more sophisticated approach is needed to select test cases from a LOTOS specification, as LOTOS is based on a synchronous communication model. The technique we use borrows its main concepts from [1] and is discussed in detail in [3]. Each IO-association constitutes guidance information for a goal-driven inference system. This guidance information is used to direct the (dynamic) derivation of a test case which covers the IO-association.

## 7 Results of application to the INRES protocol

In accordance with the proposed test architecture for the IUT (i.e. the initiator side of the INRES protocol), we have removed the responder side of the INRES protocol in each of the given protocol specifications, although this is not necessary for the dfc-method to be applicable. Furthermore, we have performed a hiding operation on the resulting specifications in agreement with the suggested PCOs, namely the user/initiator interface and the medium/responder interface. In LOTOS this was straightforward due to the availability of the hide operator. In Estelle and SDL it was also easily achieved by projecting the sequences of input/output interactions on the two PCOs.

From both the Estelle specification and the SDL specification we obtained a test suite consisting of 12 test cases. From the LOTOS specification we obtained nine test cases. This difference is explained by the (obvious) distinct syntactic structures of the languages. The number of defs and uses and their relative positions in a flowgraph may differ per FDT, causing variations in the relations that exist between variable occurrences (i.e. def-use pairs and causal variable relations). The equal number of test cases for Estelle and SDL is, therefore, merely a coincidence.

Nevertheless, after carefully investigating the test purpose of each test case it is observed that the test suites obtained from the three different versions of the INRES protocol strongly coincide. In fact, they are identical up to nine test cases, i.e. the LOTOS test suite is included in both the Estelle and SDL test suites. It is also observed that all the test cases are related to the data transmission phase of the INRES protocol. However, this was to be expected because the connection establishment phase and the disconnection phase of the protocol are specified without the explicit use of interaction parameters (the PDUs in these phases appear as constants). We argue that a larger test coverage would have been obtained in case of a more complete version of the INRES protocol. For instance, a version in which at least a calling address and called address appear as parameters of the abstract SPs and PDUs.

The detailed intermediate and final results of applying the dfc-method to the INRES protocol are included in the Appendix.

## 8 Limitations of the method

Our test derivation method assumes that infeasible paths in the flowgraph of a specification are detected while an attempt is made to select test cases that cover the IO-associations. Otherwise, coverage of all identified IO-associations by feasible paths would not be assured. However, static information is generally insufficient to anticipate accurately the dynamic evolution of a system behaviour. For instance, it may not be possible to find a particular selection of values which satisfies the predicates on some paths. It is, therefore, not always possible to find out whether a path is infeasible. In fact, it can be easily shown from the decidability theory that the generation of infeasible paths cannot be uniformly avoided. This is the price paid by *any test method* working on the basis of static information for the derivation of test cases.

The dfc-method is a *selective* test derivation method focusing on the I/O-aspects of an IUT. That is, the derivation of test cases is directed to a part of the conformance requirements only. Accordingly, complete test coverage is generally not obtained and, as a result, not every deviation of an IUT from the corresponding specification is detected. In other words, if the IUT implements the generated test suite, only the conformance of the IUT to the test suite can be concluded. In this context it is important to note that the test cases constructed from a specification by the dfc-method, or more general by data flow oriented test generation methods, are considered complementary to those that are obtained by control flow oriented test generation methods. Although the former may also examine certain control flow aspects of an IUT, it is not guaranteed that, in practice, a reasonable control flow coverage is achieved by these test cases.

The above limitations are, of course, quite natural within the realm of practical testing. As mentioned before, the detection of infeasible paths is an undecidable problem in the genral case. Furthermore, the activity of test *selection* limits itself (by definition) to selecting the most interesting and promising test purposes. Of course, this makes complete coverage impossible in practice. In spite of this, we have practically experienced that the dfc-method results in a proper selection of test cases, i.e. the test purposes of these test cases are highly representative of an important part of the functionality of the SUT.

## 9 Tools supporting the method

The main parts of the method have been, and are currently implemented partly in C and partly in Prolog under Unix. Specifications in Estelle and SDL are automatically transformed into corresponding flowgraphs. The generation of IO-df-chains from the flowgraph of a specification has been automated. Furthermore, the guided inference system for the dynamic derivation of test cases for LOTOS, based on the IO-df-chains has been implemented. Unfortunately, the implemented components are not yet (as of September 1993) integrated into a single tool. However, this integration will be realized in the near future.

## 10 Characterisation of implementation relation that is tested

Due to the selective nature of the proposed method, conformance of an IUT to a specification cannot be concluded when the IUT passes the generated test suite. Nonetheless, trace equivalence can be characterised as the implementation relation that is tested as far as the generated test suite is concerned. If the IUT does not implement the test suite it is certainly not trace equivalent with the specification. That is, the set of traces in the test suite shall definetely be included in the set of execution traces of the IUT.

## References

[1] M. Haj-Hussein, L. Logrippo, and J. Sincennes, “Goal oriented execution for LOTOS,” in *Proc. FORTE '92*, Lannion, France, M. Diaz and R. Groz, Eds., pp. 305-320, 1992.

[2] H. van der Schoot and H. Ural, “Validation activities based on static data flow analysis of system specifications in LOTOS”, Technical Report TR-92-42, Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada, 1992.

[3] H. van der Schoot and H. Ural, “Data flow oriented test selection for LOTOS”, Technical Report TR-93-12, Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada, 1993.

[4] H. Ural, “Test sequence selection based on static data flow analysis”, *Comput. Commun.*, vol. 10, no. 5, pp. 234-242, 1987.

[5] H. Ural and B. Yang, “A test sequence selection method for protocol testing”, *IEEE Trans. Commun.*, vol.□39, no. 4, pp. 514-523, 1991.

[6] H. Ural and B. Yang, “Test path selection via IP/O-chains coverage criteria”, Technical Report TR-91-32, Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada, 1991.

[7] H. Ural and A. Williams, “Test generation by exposing control and data dependencies within system specifications in SDL”, in *Proc. FORTE '93*, Boston, MA, 1993.