AGENT: An advanced test-case generation system for functional testing

by ZENGO FURUKAWA and KENROKU NOGI
Systems Development Laboratory, Hitachi Ltd.

and
KENJI TOKUNAGA
Software Works, Hitachi Ltd.
Kawasaki, Japan

ABSTRACT

This paper presents the AGENT method, a systematic test-case generation method for functional testing. AGENT consists of two steps. In the first step, a functional specification of a program is described with a formalized notation called a function diagram (FD). An FD consists of two components, a state transition which is described with a state transition diagram, and a set of boolean functions which are described with a cause-effect graph or a decision table. In the second step, test cases are mechanically generated from the FD.

Test cases generated by this method satisfy the following conditions: (1) They validate input conditions and output conditions in all states, and (2) They pass all transitions at least once and include a case of bypassing and getting through each loop in a structured state transition.
INTRODUCTION

There are two important subjects for software testing techniques, the improvement of testing efficiency, and the improvement of testing precision. The latter is more important because the purpose of software testing is the detection of errors. There are two types of testing, functional and structural. Functional testing is thought to be more basic than structural testing because errors in a program are defined as behavior discordant with the functional specification of the program. Nevertheless, functional testing has not been studied sufficiently. One conventional approach to functional testing is to have people read a functional specification and pick out test cases intuitively. However, high software reliability cannot be achieved by this method.

We have developed AGENT (Automated GENeration method of Test-cases), a systematic test-case generation method for functional testing. AGENT is composed of the following components:

1. A function diagram (FD) which formally expresses the functional specification of a program.
2. A mechanical procedure for generating test cases from the FD.

An FD model is composed of a state transition model and a boolean function model. A state transition expresses input data sequences and corresponding output data. A boolean function in a state expresses the correspondence between conditions on input and output data. A test case constitutes a sequence of states passed through in testing and a pair of conditions in each state which must be satisfied by the input and output data. AGENT automatically generates test cases from an FD.

AGENT has the following characteristics:

1. Description of a function specification with an FD is easier than with a boolean function alone because an order of input data need not to be changed into the constraints needed for a boolean function.
2. Criteria for test case generation are clear, and the number of test cases generated is reasonable.
3. Test cases are automatically generated from an FD.

We had previously developed the AGENT-I program, which generates test-cases from a cause-effect graph (CEG). However, since it was difficult to describe a functional specification with CEG, we later developed the FD notation and the AGENT-II program (usually called merely AGENT). This paper primarily discusses the method used for generating test cases. It explains FD notation and the generation algorithm, and provides a concise outline of the AGENT program.

FUNCTION DIAGRAM (FD)

Some studies have been done on the description of a functional specification as a dependency between input and output data. A cause-effect graph expresses the correspondence between input and output data based on boolean function, and the interdependency among input data owing to constraint conditions.

A functional specification of a program usually consists of a dynamic and static specification (see Figure 1). The former expresses the order of input data (or translation order); the latter, a correspondence between input and output conditions. A specification described only by a static specification is impractical because of the large number of possible combinations. A dynamic specification must be used to simplify the functional specification. A state transition model is suitable for describing a dynamic specification. In a state transition model, the output data and subsequent state are determined by the input data and the present state. A boolean logic model is suitable for describing a static specification. In a boolean logic model, the output data are determined only by the input data.

Function Diagram Notation

The notations shown in Figure 2 are used to describe a function diagram as follows:

1. **State transition.** A state transition is described with states and transitions.
   - **State.** A state indicates a place (or time) wherein data are input. An initial state is the starting point of activity and final states are possible ending places of activity.
   - **Transition.** A transition designates a change of a state. A state prior to transition is called a tail state, and after transition, a head state. The boxed T symbol over an arrow expresses a boolean function with decision table or cause-effect graph. If boolean function is simple, then the input and output condition is written directly on the arrow.

2. **Boolean function.** A boolean function in each state is described with a cause-effect graph or decision table. Interdependencies among input conditions are expressed with constraint conditions.
An Example of a Function Diagram

Figure 3 is a sample function diagram representing the following functional specification of a simplified Automatic Teller Machine (ATM). State S3 represents the display insert card.

1. The ATM displays key in pass code when a cash card is inserted.
2. The ATM checks the correspondence between the keyed-in pass code and the code on file. If they are same, the ATM displays key in sum; if not, the ATM checks whether the number of times that the correct code has not been entered is equal to three. If so, the ATM displays stop process, cancels the card registration, and displays insert card for the next customer. If not, the ATM displays key in pass code again.
3. When an amount is keyed in, the ATM checks whether it is less than or equal to the balance. If the amount is greater than the balance, the ATM displays key in sum and waits for the amount to be keyed in again. If the amount is less than or equal to the balance, the ATM pays the money requested, reports the balance and displays insert card.
(1) State
   - Initial State
   - Final State
(2) Transition

(a) state transition notation

(1) Cause-effect graph
A \rightarrow C if A and B then C
B \rightarrow C if A or B then C
A \rightarrow B if not A then B
A \rightarrow S if A \land B then goto S

(2) Decision table

<table>
<thead>
<tr>
<th>input</th>
<th>C</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>1</td>
<td>2</td>
<td>*</td>
</tr>
<tr>
<td>state</td>
<td>S1</td>
<td>*</td>
<td>S2</td>
</tr>
</tbody>
</table>

* Y: input condition is true
* N: input condition is false
* *: true
* true
* false

(3) Constraint conditions
E A : exclusive
F B : inclusive
G A : one and only one
R B : require

(b) boolean function notation

Figure 2—Function diagram notation

METHOD OF TEST CASE GENERATION

In generating test cases from an FD, it is important that the number of test cases be practical and that the criteria for test case generation be clear. An FD is composed of a state transition and boolean functions. Myers described a test case generation method for boolean functions in which test cases are generated from a cause-effect graph. The AGENT-I program was developed based on this method. The criteria considered for a state transition is the path testing strategy of programs where a state is substituted for a node and a transition is substituted for a branch.

Using clearly defined criteria, a practical number of test cases are generated from an FD by combining test cases of a state transition (testing paths) with test cases of boolean functions (partial test cases).

Test Case Generation Criteria

Partial test cases which are generated from the boolean functions of an FD include true and false cases of input data conditions, and are nearly minimum. The number of test cases increases linearly with the number of input data conditions.

There are many criteria for state transitions; among these, passes all states (C0 coverage) and gets through all transitions (C1 coverage) are well known. For the state transition shown in Figure 4(a), testing paths (i.e., sequences of states) of (b) and (c) are example sequences for all states passing and all transitions getting through, respectively.

In programming, a case of bypassing a loop is apt to be overlooked or a mistake made concerning a loop termination condition. Testing paths should include both a case of bypassing a loop and one of getting through a loop in a state transition. A set of testing paths may satisfy the criteria for all transitions getting through, but not include a case of bypassing a loop. For example, in Figure 4, testing paths (c) get through all transitions, but a case of bypassing the loops at state S2 is not included. It is then necessary to include such a case for testing criteria as in (d).

It is difficult to recognize all of the loops of a typical state transition, but easy to recognize them in a structured state transition which is composed of only three kinds of forms (e.g., sequence, selection and iteration, as shown in Figure 5.)

In Figure 6, (b) is an example of a structured state transition and of the state transition in (a). There are two loops in Figure 4(a): (S4, S5, S4) and (S4, S5, S3, S2, S4).

For testing path generation, the test cases must pass all transitions of a structured state transition (SST) at least once, and include the cases of bypassing and getting through loop. Figure 4(c) is an example of testing paths which satisfy this criteria. Were the foregoing loop requirements not included in our criteria, they would be satisfied by path 1 and path 2. These do not include the case of bypassing the loops at state S4.

Test Case Generation Procedure

Test cases are generated from an FD by the following procedure.

1. Generation of partial test cases. In each state, partial test cases are generated from a cause-effect graph in which causes are input data conditions and effects are output data or head states of the boolean function. If a decision
Table 1: A Decision Table

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>pass code=registration</td>
<td>Y N N</td>
</tr>
<tr>
<td>mistaken=three times</td>
<td>N Y N</td>
</tr>
<tr>
<td>M2</td>
<td>*</td>
</tr>
<tr>
<td>M3</td>
<td>*</td>
</tr>
<tr>
<td>M4</td>
<td>*</td>
</tr>
<tr>
<td>cancel card</td>
<td>*</td>
</tr>
</tbody>
</table>

(a) State transition part

(b) Boolean function part

Figure 3—An example of FD (ATM)

(a) State transition

(b) Testing Path for passing all states

(c) Testing Paths for getting through all transitions

(d) Testing Paths for loop criteria

Figure 4—An example of testing paths

Figure 5—Notation for a structured state transition
sized from the testing paths and partial test cases in each state of the FD. They are made up of sequences of states from initial to final and combinations of input data conditions and combinations of output data in each state. In the synthesis algorithm, a partial test case is assigned to a state of a testing path in which a head state of the partial test case is the same as the next state of the assigned state in the testing path.

**Synthesis Algorithm of Test Cases**

An SST is expressed as described by the tree expression shown in Figure 5(c) for test case synthesis. A tree expression of an SST is called a condition structure tree (CST). The CST of the Automatic Teller Machine (ATM) is presented in Figure 7.

Test cases are synthesized according to the following steps:

1. The number of partial test cases which have the same head state is counted in each state, and the sum is set for the corresponding leaf of the CST.
2. Each node of the CST is retrieved in post order, and the number of test cases in each node is calculated with the rules shown in Figure 8.
3. Test cases are synthesized in numerical order from 1 to \( n \) where \( n \) is the number of test cases in the root node of the CST. The \( n \)th test case in each node is made up from children's test cases by the following rules (where \( c(i) \) is the number of test cases of the \( i \)th child node.

(a) **Sequence.** In the \( i \)th child node, the \( j \)th test case is

\[
j = \max(i_1, i_2, \ldots, i_n)
\]

(b) **Selection.**

\[
j = i_1 + i_2 + \ldots + i_n
\]

(c) **Iteration.**

\[j = i + 1\]
Test case 1
  (empty)

Test case 2
  (insert card)
  not (pass code=registration code) and (mistaken=three times)
  (*) - uncondition

Test case 3
  (insert card)
  not (pass code=registration code) and not (mistaken=three times)
  (pass code=registration code) and not (mistaken=three times)
  (sum balance)
  (*)

Test case 4
  (insert card)
  (pass code=registration code) and not (mistaken=three times)
  (sum balance)
  (sum balance)
  (*)

Figure 9—An example of test cases

extracted and test cases for all children are connected. \( j \) satisfies (1-1)

\[
j(i) = \text{mod}(n - 1, c(i)) + 1 \quad \text{(1-1)}
\]

where \( 1 \leq i \) the number of children.

(b) Selection. The \( k \)th test case of the \( j \)th child node is extracted. \( j \) and \( k \) satisfy (1-2) and (1-3) respectively.

\[
\sum_{m=1}^{i} c(m) < n \leq \sum_{m=1}^{i} c(m) \quad \text{(1-2)}
\]
\[
k = n - \sum_{m=1}^{i} c(m) \quad \text{(1-3)}
\]

(c) Iteration. The \( j \)th test case of the child node is extracted. If \( j \) equals zero, then the test case is empty.

\[
j = \text{mod}(n - 1, c(1)) \quad \text{(1-4)}
\]
(d) **Leaf.** A tail state of a transition which corresponds to a leaf and the \( j \)th partial test case is extracted.

\[
j = \text{mod}(n - 1, \text{the number of partial test cases}) + 1
\]

\[(1-5)\]

A test case is composed of a sequence of tail states which are extracted at the leaves, and combinations of input conditions and output data which are parts of the partial test cases at the leaves.

The numerals on the right side of the CST node box in Figure 8 represent the number of test cases of each node. The ATM example has four test cases. Each test case is described in Figure 9.

**THE AGENT PROGRAM**

The AGENT program, which automatically generates test cases from an FD, was written in PL/I and put into operation in 1983. The functions of the AGENT program and some experiences with it are briefly described below.

**Input and Output**

The source list, test case table and test case sheet of the ATM example are presented in Figures 10 and 11. The input to the AGENT program is an FD which is written in function diagram language. A function diagram is composed of a title statement (TITLE), state statements (STATE), an initial state statement (INITIAL), a final state statement (FINAL) and an end statement (END). In each state statement, a boolean function is described with condition definitions (NODE), boolean expression (RELATE) or decision table definitions (DECISION), and constraint condition definitions (CONST).

The main output of AGENT is a table of test cases. A source list, decision tables, and a transition matrix can also be output. In a test case table, a transition is composed of a tail state (TAIL), head state (HEAD). Conditions (NODE) in

```plaintext
1 TITLE CD = 'AUTOMATIC TELLERS MACHINEM
2 STATE (S0)
3 NODE CAUSE CO = 'INSERT CARD'
4 EFFECT M1 = 'KEY IN PASS CODE'
5 DECISION
6 C01 = (CO)/M1->S1
7 STATE (S1)
8 NODE CAUSE AA = 'PASS CODE = REGISTRATION CODE'
9 AB = 'MISTAKE = THREE TIMES'
10 EFFECT M2 = 'KEY IN SUM'
11 M3 = 'KEY IN PASS CODE AGAIN'
12 M4 = 'STOP PROCESS'
13 OC1 = 'CANCEL CARD'
14 DECISION
15 C11 = (AA NOT AB)/M2->S2
16 C12 = (NOT AA AB)/M4 OC1->S3
17 C13 = (NOT AA NOT AB)/M3->S1
18 CONST
19 U11 = AA AB EXCLUSIVE
20 STATE (S2)
21 NODE CAUSE KA = 'SUM <= BALANCE'
22 KB = 'SUM > BALANCE'
23 EFFECT M21 = 'KEY IN SUM'
24 FR = 'REPORT BALANCE'
25 FR1 = 'BALANCE <= BALANCE-SUM'
26 DM = 'PAYMENT MONEY'
27 DM1 = 'MONEY=SUM'
28 RELATE
29 R1 = KA SIMP FR
30 R2 = KA SIMP FR1
31 R3 = KA SIMP DM
32 R4 = KA SIMP DM1
33 R5 = KA SIMP ->S3
34 R6 = KB SIMP M21
35 R7 = KB SIMP ->S2
36 CONST
37 U21 = KA KB ONLY
38 STATE (S3)
39 NODE EFFECT M7 = 'INSERT CARD'
40 DECISION
41 C31 = (*)/M7->S0
42 INITIAL S0
43 FINAL S0
44 END
```

Figure 10—An example of a source list

From the collection of the Computer History Museum (www.computerhistory.org)
the tail state are divided into those for input data (I) and output data (O). Each test case is expressed in one column of a test case table. A blank indicates that a test case did not get through that transition, an 'O' indicates that a condition is true, and an 'X' indicates that a condition is false.

The first test case in Figure 11 did not get through any transition. Since the ATM example has the same initial and final state (S0), the first test case is a case of bypassing the loops at S0. The second test case gets through a sequence of states (S0, S1, S3, S0) where at each state, input conditions are set and output data are checked. For example, at state S1, the former are to key in error pass code and iterate three times to
error and the latter are to certify that the message key in pass code again is not output, the message stop process is output, and the card registration is cancelled.

Experiences with AGENT

Table I is a summary of AGENT applications. Numbers 1 to 5 are experimental specifications for FD descriptions; numbers 6 to 11 are parts of practical software. The following points have been deduced from the data and user suggestions.

1. Size of FD. As shown in Table I, the number of states is less than fifteen. It is thought that this is due to the conventional notation for describing a functional specification (natural language or state transition matrix, etc.). Thus, the limitation on the number of states may be changed by using an FD. In our experience, when the number of states is over twenty, the FD cannot be readily grasped by human operators.

The mean number of input conditions varies widely depending on the software. In our experience with cause-effect graphs, the number of conditions (input conditions, output data, and head states in an FD) is limited to thirty or forty. There is a problem in the decision table (which is induced to an FD to describe boolean functions) in that the combination of conditions increases exponentially in proportion to the number of input conditions. Thus, the number of input conditions at each state should be kept as small as possible.

2. Remarks on writing an FD. It is noticeably easier to describe the ordered logic of a functional specification in an FD than in a cause-effect graph. However, when states are made without giving them much thought, non-executable test cases are apt to be generated by AGENT. Non-executable test cases are caused by inadequate attention to state setting. Such cases can be solved by state decomposition.

3. Effective use of the AGENT method. The AGENT method is particularly effective because it allows detection of design errors in a functional specification by translating it to an FD. Even inexperienced people are able to generate test cases with AGENT. The following factors enhance the effectiveness of the AGENT program.

(a) Description of an FD at the design stage. This is effective for early detection of design errors. The FD review ensures that development people will have a common understanding of a functional specification.

(b) Use of inexperienced people for generation and execution of basic test cases in functional testing. This allows experienced people to concentrate on cases not covered by the AGENT method or on very special cases which are deduced by experience and human intuition.

SUMMARY

The AGENT method has been proposed for systematic test case generation for functional testing. Generated test cases satisfy the following criteria for a function diagram which expresses a functional specification:

1. Each state includes true and false cases of input and output conditions.
2. All transitions of a state transition are retrieved at least once, and bypassing and getting through all loops in a state transition are included.

The AGENT program automatically generates test cases from a function diagram. While further evaluation is required, experience with the AGENT method and program have shown it to be effective for standardizing test case generation and simplifying test case management.

In future work, it will be necessary to evaluate our testing criteria by using error data gathered in the field. In order to widely propagate the AGENT method, much related research is needed. The first task is to extend the function diagram. For example, a function diagram should be able to describe parallel processing. The next task is to strengthen the testing criteria. To accomplish more effective testing, the AGENT method must be capable of generating not only test cases, but also test data for executing a program.

REFERENCES
