# Test generation methodology based on symbolic execution for the Common Criteria higher levels

**Alain Faivre, Christophe Gaston**

CEA/LIST, Saclay F-91191, Gif-sur-Yvette Cedex, France
Email : alain.faivre@cea.fr, christophe.gaston@cea.fr

## 1 Introduction

In the field of security software, the Common Criteria (CC) constitute an ISO standard for the evaluation of products and systems from Information Technologies. The international recognition of the Common Criteria justifies the investment undertaken by the manufacturers to obtain the certification of their products. The evaluation criteria are defined according to the Evaluation Assurance Level (EAL). There are seven EALs: EAL1 to EAL7, in an increasing order of security demand. For the upper levels of evaluation, the use of formal methods is mandatory. In that case, supplies intended to realize evaluation activities must contain components associated to modelling, proof and test. This contribution proposes a methodology and a tool (AGATHA [1,2]) which allow to cover the requirements associated to test generation for the upper levels of the Common Criteria. In that case, the criterion used to stop the test generation activity is defined as follows: the generated test case set covers all functions of the reference model. Each function must be covered "complete" way (although the term complete remains ambiguous in CC definitions). The strategy presented in this paper provides a formal meaning to this criterion and associated test generation techniques.

## 2 AGATHA tool

The AGATHA tool allows to simulate specifications based on communicating automata (SDL, Statecharts (STATEMATE), a subset of UML, ESTELLE and IF) and to automatically generate test case sets in accordance with several coverage criteria. AGATHA mainly relies on symbolic execution techniques. Now, we present the mechanisms linked to test generation from specifications expressed in the input language called IOSTS (Input Output Symbolic Transition System) [3]. This language is based on communicating automata with inputs and outputs. Communication between automata is a synchronous one. The example of the Figure 1 shows two of these automata As we can see, the following information can be associated with each IOSTS transitions:

- *guards* which are formula on variables used by the corresponding IOSTS. These guards are conditions to trigger the transitions. In the IOSTS on the right, the transition between $q'_1$ and $q'_2$ can be trigger only if $z$ is not equal to zero.
- *expressions* which denote communication. Outputs correspond to expressions of the form $c!t$ (where $t$ is a term) and inputs to expressions of the form $c?x$. In the IOSTS on the right, the transition between $q'_0$ and $q'_1$ can be trigger only if a value is received through the channel $c_i$. This

value is then assigned to the variable *z* ($c_i?z$). In the example of Figure 1 we introduce two channels: $c_i$ which is an internal channel used as a communication point between the two automata and $c_e$ which is an external one used to communicate with the external environment.

- *assigments* which modelized updating of variables. In the IOSTS on the right of Figure 1, the trigger of the transition between $q'_1$ and $q'_2$ has the effect of updating the variable *z* with its current value divided by two.

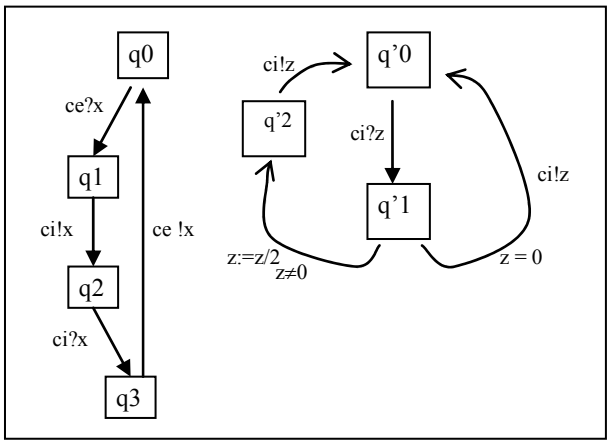


**Figure 1.**

Treatments carried out by AGATHA aim at characterizing the symbolic behaviours associated to an IOSTS in an exhaustive manner. For that, the tool determines necessary and sufficient constraints corresponding to each behaviour specified by the IOSTS. Now we show how the AGATHA tool characterizes these constraints on the example described in Figure 1. Treatments made by AGATHA are described in Figure 2. In fact in this case we want to determine constraints associated to behaviours resulting of a system made up of two synchronously communicating IOSTS. Initially, the system is in the state $(q_0,q'_0)$ corresponding to initial states of the two IOSTS. For the moment the variables x and y, manipulated by these IOSTS, are associated with no constraint. Thus, we assign two symbolic constants, a and b, respectively to x and y, associated with no constraint (which is denoted by the true expression). The IOSTS on the right of Figure 1 can not evolve since it is waiting for an input through the internal channel $c_i$ and the IOSTS on the left is not in a state which allows to send an output through this channel. On the other hand, the IOSTS on the left can evolve if it receives a value from the environment through the channel $c_e$. We represent this value by a symbolic constant c. Then, the system evolves in the state $(q_1,q'_0)$ in which c is assigned to x. In this new state, an internal communication can occur (denoted by $\tau$). This communication has the effect of updating the variable z with the value of x: then z is assigned to c and the system evolves in the state $(q_2,q'_1)$. Let us consider the arrow which enables the system to evolve in the state $(q_2,q'_2)$: this corresponds to trigger the transition between $q'_1$ et $q'_2$ in the IOSTS on the right. This evolution is possible only if $z \neq 0$. Since z is assigned with the symbolic value c, we introduce the constraint $c \neq 0$. Now the value of z must be divided by 2: z is assigned with c/2. The other steps of calculus represented in Figure 2 are based on the same principle. Now, for each behaviour (i.e. path) contained in the graph in Figure 2, we are able to associate a test case. Let us consider the behaviour depicted by the path on the left in Figure 2: the associated calculated

constraint is c≠0. The tool generates an arbitrary value verifying this constraint: for example c=6.Two other symbolic constants a and b must be associated with any numerical values, for example a=2 and b=3 (these values have no consequence on the generated test script). Then the tool works out the corresponding numerical trace: ce?6 ce!3. The corresponding test case is: enter the value 6 and compare the output value with 3. If the value sent by the system is effectively 3, the verdict is success, otherwise the verdict is failure.

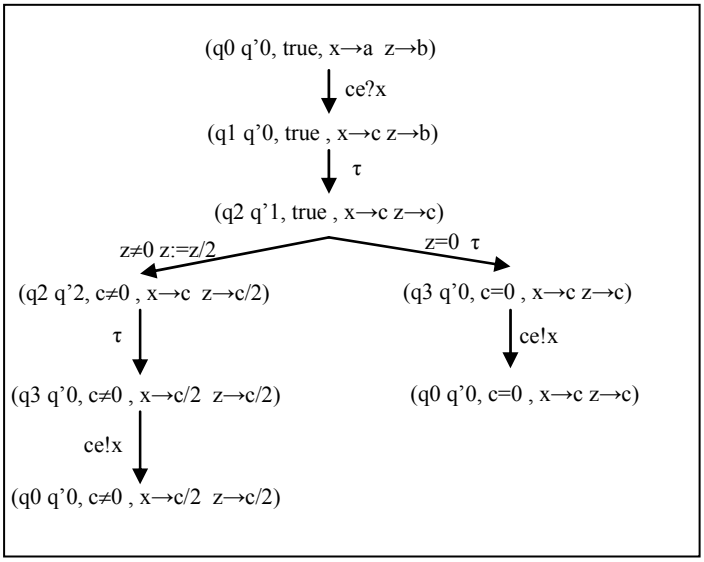


**Figure 2.**

Of course, in case of reactive systems (which continuously interact with the environment), there is no reason to stop the symbolic execution process. Then, in order to generate test cases, we characterize stopping criteria applied to symbolic execution. The simplest is to cover all system transitions. The advantage is to simplify calculation and to obtain a weak-size set of numerical tests. From the testing point of view, such criterion can be too weak. We propose a more sophisticated criterion: the inclusion criterion, based on the inclusion of symbolic states calculated by the tool. This criterion allows the exhaustive calculus of the system symbolic behaviours. Its advantage is to supply a test set associated with a high confidence level. Its disadvantage is to cause a combinatory explosion during analyse of complex systems with high level of parallelism. Now, we present this criterion with the help of the example of Figure 1 with its associated symbolic execution in Figure 2. In this figure each node of the graph is associated to a 3-uplet, named symbolic state, which contains: 1) the reached states by the IOSTS of the system, 2) the constraint on symbolic inputs, named path condition, necessary to reach the node, 3) the symbolic value of each variable associated to each IOSTS (e.g. in symbolic root node the assignment is: x → a et z → b).

From there we are able to characterize constraints on variables manipulated by the IOSTS to reach any symbolic state. For each symbolic state, we are able to characterize what we name "symbolic

state semantics": a couple which contains the reached states and the constraints associated to the variables manipulated by the IOSTS. Thus, the semantics of the symbolic root node ($q_0q'_0$,, true ,x→a and z→b) is

Sem ($q_0q'_0$,, true ,x → a and z → b) = ($q_0q'_0$,, true).

Similarly, if we consider the "leaf" symbolic state (at the bottom and on the left of Figure 2)

($q_0q'_0$,, c ≠ 0, x → c/2 and z → c/2),

its associated semantics is

Sem ($q_0q'_0$, c ≠ 0, x→c/2 and z→c/2) = ($q_0q'_0$ , x ≠ 0).

AGATHA offers a mechanism to compare the semantics of symbolic states: we say that the semantics $Sem_1$ of a symbolic state $S_1$ is included in the semantics $Sem_2$ of a symbolic state $S_2$ (or more simply $S_1$ is included in $S_2$) iff: 1) the first component of $Sem_1$ is equal to the first component of $Sem_2$ (the two symbolic states characterize the same reached states), 2) the second component $f_1$ de $Sem_1$ characterizes a constraint "stronger" than the second component $f_2$ of $Sem_2$ (in other words $f_1 \Rightarrow f_2$ is a tautology). For example in Figure 2, the symbolic state ($q_0q'_0$,, c ≠ 0, x → c/2 and z → c/2) is included in the root symbolic state. Actually their semantics denote the same reached states $q_0q'_0$ and the formula x≠0 ⇒ true is a tautology.

When the AGATHA tool carries out a symbolic execution with the aim to satisfy the inclusion criterion, the process is the following:

- As soon as a new symbolic state Q is calculated, its semantics is compared to the semantics of the previous symbolic states.
- If the semantics of Q is included in no semantics of the previous symbolic states, all possibly triggered transitions from Q are executed in a symbolic manner. With each executed transition, a new symbolic state is defined and again the same process is executed.
- If the semantics of Q is included in a previous symbolic state, no transition from Q is executed.

Let us note that the symbolic execution tree of the Figure 2 satisfies the inclusion criteria.

## 3 Cover criteria consistent with CC

In the framework of the CC, we propose a mechanism with two phases explained on the example of Figure 1. Let us suppose that we would like to cover all the behaviours of the calculus function represented by the IOSTS on the right side. The first phase consists in characterizing the behaviours to cover. The Figure 3 represents the generated symbolic execution tree associated to the IOSTS on the right of Figure 1. The resulting tree covers all transitions of the IOSTS and covers all paths of some size N (in the example N=2) given as parameter to the tool. Of course, the ending criterion for the symbolic execution used in the first phase can be more or less sophisticated, the stronger being the inclusion criterion.

The two generated paths in Figure 3 are respectively named $Exec_1$ and $Exec_2$. For these two behaviours we would like to generate a test case. This point is the subject of the second phase. Again the tool carries out a symbolic execution of the complete system. In that case, this symbolic execution

must satisfy the following second criterion: the behaviour taken as argument (in our example $Exec_1$ or $Exec_2$) is covered by at least one of the paths of the symbolic execution tree and it exists one path in this tree which covers the behaviour and ends by an output to the environment.

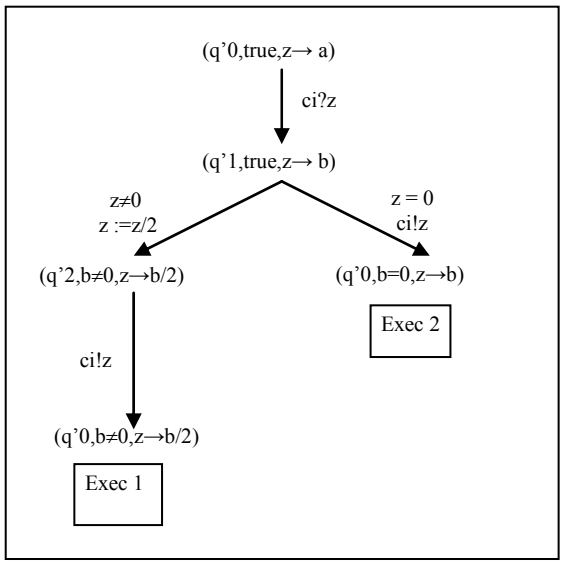


**Figure 3.**

If we consider the path $Exec_1$, any symbolic execution of the system described in Figure 1 and including the path of Figure 4 satisfies the test criterion. The phase 2 of our mechanism is then finished.

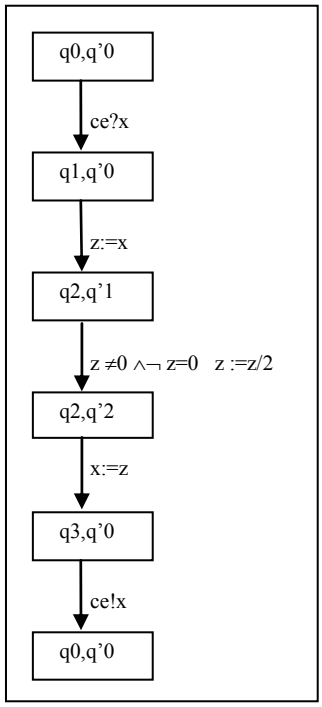


**Figure 4.**

More precisely, we have put a heuristic in place in order to constraint the symbolic execution mechanisms to avoid an explosion combinatory which is inherent to this type of criterion.This heuristic is about a symbolic adaptation of the « hit or jump » ([4]) algorithm. Let Sys the system made up of all communicating automata, ch the path to cover and N an integer given as parameter to the algorithm. The algorithm executes the following phases:

- Symbolic execution of Sys by constructing all paths of size N.
- If one of the paths (named ch1) contains ch (i.e. to construct $ch_1$ it has been necessary to trigger all transitions of ch, respecting the order implied by ch) then $ch_1$ is a candidate to characterize the test case.
- Otherwise let $ch_1 \ldots ch_n$ the set of all paths which contain the longer prefix of ch among all executed paths. We consider only one of these paths randomly chosen: we named it $ch_i$. Then the algorithm is again executed from the end state of $ch_i$ and considering the prefix of ch not still cover (i.e. this prefix becomes the path to cover).
- The test case will be extracted from a path prefixed by $ch_i$ and suffixed by the result of recursive execution of the algorithm.

## 4 Conclusion

It is interesting to notice that this testing approach had been approved for the higher levels of the Common Criteria by a CESTI which is an organisation in charge to certify methodologies and tools in regards of CC.

At the present time, we perfect the AGATHA tool by adding more subtle strategies associated to the "hit or jump" algorithm and we plan to evaluate them on industrial size specifications.

## References

[1] C. Bigot, A. Faivre, J.-P. Gallois, A. Lapitre, D. Lugato, J.-Y. Pierron and N. Rapin, "Automatic Test Generation with AGATHA", Proc. TACAS, 2003.

[2] N. Rapin, C. Gaston, A. Lapitre and J.-P. Gallois, "Behavioural Unfolding of Formal Specifications Based on Communicating extended automata", Proc. ATVA 2003.

[3] V. Rusu, L. du Bousquet, and T. Jéron, "An approach to symbolic test generation", in IFM '00: Proceedings of the Second International Conference on Integrated Formal Methods. London, UK: Springer- Verlag, 2000, pp. 338--357.

[4] A. Cavalli, D. Lee, C. Rinderknecht and F. Zaidi, "HIT-OR-JUMP: an Algorithm for Embedded Testing with Applications to IN Services", Proc. FORTE/PSTV, 1999.