Test Case Generation for Context Testing of Embedded Systems

Qi-Ping Yang and Tae-Hyong Kim

School of Computer and Software Engineering, Kumoh National Institute of Technology, Gumi, 730-701 Korea

Summary
The context of a modular system through which its embedded components interact with the user is the main development target of the modular system because developers usually purchase embedded components on the market. Therefore, context testing is necessary for the development of a reliable modular system. Test case generation for context testing may be complicated as the tester cannot directly control the interfaces between the context and the embedded components. This paper first shows a basic solution approach and its incompleteness. Then it investigates the conditions for avoiding nondeterminism in context testing. A graph conversion algorithm is also proposed which constructs safer context specifications for test generation of context testing without nondeterminism.

Key words:
Embedded System, *Context Testing*, *Test Case Generation*

1. Introduction

As communication protocols are getting more complex, their architectures have been changed from monolithic to modular with a lot of components. Embedded systems are currently popular examples of modular systems where a couple of components are crucial control ones. An embedded system consists of two parts: (1) the components embedded in the system, and (2) the context through which embedded components interact with the environment. In order to develop a reliable modular system, there have been several researches on testing embedded components [1-5]. However, when developing a practical embedded system, designers tend to directly utilize components that have been verified faultless and can be purchased from the market as modules for reducing the developing and testing time of an embedded system. These modules are embedded in the system and operated by interactions with the user through the context. Therefore, how to design a dependable context is a practical goal in the development of embedded systems. In fact, testing the context, what we call 'context testing', may be more important than testing the components in embedded system testing. In order to generate test cases for context testing, testers should be able to generate every possible input that is defined in the context. In addition, all outputs generated by the context should be observable by the testers.

Unfortunately, these two requirements are often unattainable. Testers may have two different types of interfaces on the context: accessible interfaces and inaccessible interfaces. Fig.1 shows a context of an embedded system which has inaccessible interfaces such as I_1 and I_2. Testers can directly apply inputs to the context through accessible interfaces but cannot apply to the interfaces between the components and the context. Such inaccessible interfaces are similar to the semi-controllable interfaces that were presented in [6]. Due to the controllability problem, nondeterminism and race conditions may occur during testing, which reduces the testability of the context.

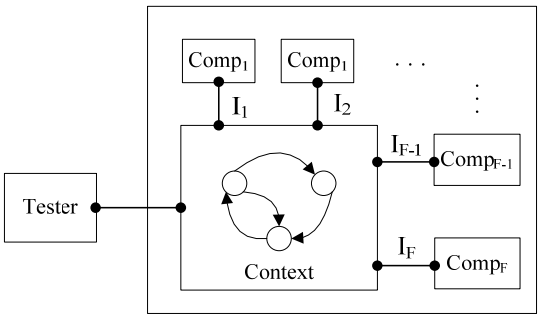

Fig. 1 The concept of context testing

Interactions between embedded components and the context through multiple inaccessible interfaces may sometimes cause nondeterminism. If the context, for example, moves into a state at which several inputs from different interfaces are pending, choosing which input is consumed first may lead to nondeterminism. Another important issue in context testing is the buffer types of the interfaces between the context and components. Although those interfaces can be assumed as FIFO-type buffers, in practice this assumption may not be correct for all implementations. Test sequences generated under that assumption may not be applicable due to the different buffer type.

Fecko, et al. presented a test generation algorithm for embedded systems which tried to handle safely

Manuscript received November 5, 2007
Manuscript revised November 20, 2007

semi-controllable interfaces of the system [6]. This paper first tries to utilize that algorithm in test generation for context testing. Then the test generation conditions are discussed for context testing which can avoid the possible race condition and nondeterminism. We also propose a new algorithm for checking and modifying the context to generate such safe test sequences.

The rest of this paper is organized as follows. After the preliminaries of Section 2, Section 3 introduces how to use the existing algorithm to construct a test sequence for the context testing. A new algorithm is proposed in section 4 to check and modify the context graph for handling the inaccessible interfaces without the race conditions. Finally, section 5 concludes the paper and presents ideas for future work.

2. Preliminaries

In this paper, an FSM model, which is sufficient to model protocols with finite state space and deterministic behavior, is used to represent the context and the components.

2.1 FSM and its graphical representation

Definition 1. A finite state machine (FSM) is a 6-tuple $M = (S, X, Y, \delta, \lambda, s_0)$, where S is a finite set of states of M and $s_0 \in S$ is the initial state of M, X is a finite nonempty set of input symbols, Y is a finite nonempty set of output symbols, δ is a state transition function that maps $S \times X$ to S, and λ is an output function that maps $S \times X$ to Y.

State s_i is *equivalent* to state s_j if the inputs defined for s_i are a subset of those for s_j and their corresponding outputs and next states are identical. An FSM M is said to be *minimal* if its specification has no equivalent states.

An FSM M may be represented by a directed graph (digraph) $G = (V, E)$ where a set of vertices V represents the set S of states of M, and a set of directed edges E represents the set of transitions of M. An edge e represents a specific transition t of M from state s_i to state s_j with input $x \in X$ and output $y \in Y$. Thus, an edge e is defined by a 3-tuple $(v_i, v_j, x/y)$ in which v_i is the initial vertex, v_j is the final vertex and $l = x/y$ is its label. Vertices v_i and v_j which represent respectively state s_i and s_j are called the head and the tail of e, denoted $head(e)$ and $tail(e)$.

The *indegree* and *outdegree* of a vertex are the number of edges coming toward and directly away from it, respectively. If, for any given vertex, its indegree is equal to its outdegree, the graph is said to be *symmetric*. A *tour* of a graph G is a sequence of consecutive edges that starts and ends at the same vertex. One special kind of tour is an *Euler tour*, which contains every edge of G exactly once.

A digraph $G = (V, E)$ is said to be *strongly connected*, if, for every pair of vertices v_j and v_k, there exists a path from v_j to v_k. G is *weakly connected* if the undirected graph, generated by removing the direction from each edge, is connected. If a graph is symmetric and strongly-connected, an Euler tour exists [8,9].

2.2 Modeling testing embedded systems

Since the structure of embedded systems is similar to the multi-layer testing environment that is presented in [6], we directly use the same definitions for modeling the embedded systems in this paper. Given a graph $G(V, E)$ representing an FSM model of the context with multiple interfaces with components, the following parameters are defined: $|V|$ is the number of nodes in G, F is the number of multiple interfaces between the context and the components, b_i is the buffer size (maximum number of inputs buffered) at the i-th interface I_i, A_i is the set of inputs from i-th component triggering transitions, O_i is the set of outputs of the context that force inputs in A_i to the context from the i-th component, T_ϕ is the subset of edges in G whose input and output symbols are not in A_i and O_i respectively, and c_i is the number of different transition classes in the context triggered by inputs from i-th component. Two transitions t_1 and t_2 belong to the same transition class if and only if they both become fireable by the same input, $T_{i,j}$ ($\subset E$) is the subset of edges in G triggered by the inputs $a_{i,j}$ from the i-th component, and $U_{i,j}$ is the set of transitions in the context with output $o_{i,j}$ such that, in response to $o_{i,j}$, an input $a_{i,j}$ is buffered at I_i. An example is presented for those definitions, which was shown in [6]. We use that example for easier comparison afterwards.

Example 1. Consider the context of Fig. 2 which is interacting with components M_1 and M_2 through inaccessible interfaces I_1 and I_2, respectively. The FSM of the context is described in Table 1. Transition e_1, triggered by input x_1 from the tester, generates output $o_{1,1}$ to M_1. In response, M_1 sends back input $a_{1,1}$ which triggers transition e_3. Note that $a_{i,j}$ denotes the expected response to $o_{i,j}$. e_3, when traversed, outputs $o_{2,1}$ to M_2, which responds with input $a_{2,1}$ triggering e_4 or e_9. $o_{2,1}$ is also output to M_2 by e_{12}, which is fired by the tester's input $x_{1,2}$. Transitions e_7 and e_8, after being triggered by the tester's inputs x_7 and x_8, respectively, generate output $o_{1,2}$ to M_1. M_1 sends back input $a_{1,2}$, which triggers either e_2 or e_{11}. e_2 outputs $o_{1,2}$ to M_1. Again, M_1 responds with input $a_{1,2}$. On the other hand, transitions e_5, e_6, e_{10}, e_{13}, and e_{14}, can be triggered directly by the tester and do not generate outputs to the inaccessible interfaces. In this example, we have:

- $|V| = 3$; $F = 2$; $c_1 = 2$, $c_2 = 1$
- $A_1 = \{ a_{1,1}, a_{1,2} \}$, $A_2 = \{ a_{2,1} \}$
- $O_1 = \{ o_{1,1}, o_{1,2} \}$, $O_2 = \{o_{2,1} \}$

- $T_{1,1} = \{ e_3 \}$, $T_{1,2} = \{ e_2, e_{11} \}$, $T_{2,1} = \{ e_4, e_9 \}$
- $U_{1,1} = \{ e_1 \}$, $U_{1,2} = \{ e_2, e_7, e_8 \}$, $U_{2,1} = \{ e_3, e_{12} \}$

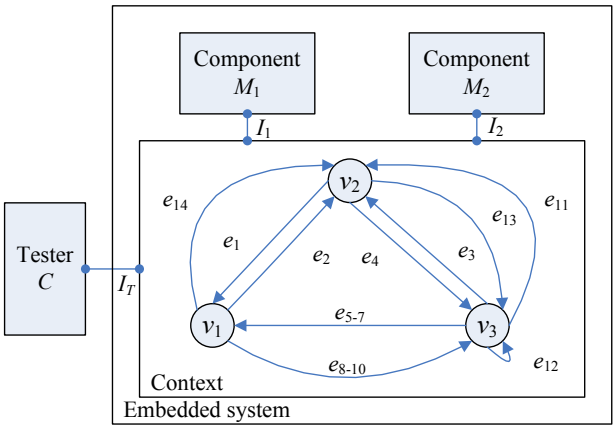

Fig. 2 FSM model of the context in Example 1 [6]

Table 1 Inputs and outputs for the edges of Fig.2. [6]

Edge	*Input*	*Output*	*Edge*	*Input*	*Output*
e_1	$C?x_1$	$M_1!o_{1,1}$	e_8	$C?x_8$	$M_1!o_{1,2}$
e_2	$M_1?a_{1,2}$	$M_1!o_{1,2}$	e_9	$M_2?a_{2,1}$	$C!y_9$
e_3	$M_1?a_{1,1}$	$M_2!o_{2,1}$	e_{10}	$C?x_{10}$	$C!y_{10}$
e_4	$M_2?a_{2,1}$	$C!y_4$	e_{11}	$M_1?a_{1,2}$	$C!y_{11}$
e_5	$C?x_5$	$C!y_5$	e_{12}	$C?x_{12}$	$M_2!o_{2,1}$
e_6	$C?x_6$	$C!y_6$	e_{13}	$C?x_{13}$	$C!y_{13}$
e_7	$C?x_7$	$M_1!o_{1,2}$	e_{14}	$C?x_{14}$	$C!y_{14}$

* $A?x$ and $B!y$ denote receiving input x from A, and sending output y to B, respectively.

3. Basic Approach

In this section, we present a basic approach for generating minimum-cost test sequences for context testing while avoiding race conditions and nondeterminism. In this approach we directly use the test generation algorithm for multiple semi-controllable interfaces which was presented in [6].

3.1 Controllability problem

Consider the embedded system shown in Fig. 2 and Table 1. In order to execute the transition e_{11} of the context, input $a_{1,2}$ is to be applied from M_1 to the context which is in state v_3. The tester has to force the context to generate output $o_{1,2}$ to M_1 through the accessible interface I_T. We can use transition e_7 for that purpose. When the input x_7 is applied, the context generates output $o_{1,2}$ to the M_1 and moves to state v_1. Then the tester applies input x_{10} to the context, which executes transition e_{10} and the state is changed to v_3 in the context. Finally transition e_{11} can be triggered by input $a_{1,2}$ from M_1 which may be buffered in I_1 before.

Unfortunately, this solution may have a race condition. When the context is in state v_1, after traversing e_7, M_1 may produce $a_{1,2}$ as a response to $o_{1,2}$ before the tester applies x_{10} for executing e_{10}. Then the $a_{1,2}$ may be consumed by transition e_2 before transition e_{10} fires. This problem occurs because the tester cannot directly control the interface between the context and the components. The input $a_{1,2}$ from M_1 may arrive at the context before, after, or at the same time input x_{10} does.

As shown in this example, the context may move into a state where the context is forced to consume a previously buffered input. This situation may create a race condition if the test sequence requires that another input be sent to the context immediately by the tester. Therefore, a test sequence producing such a race condition should be avoided which may bring the context to a state where multiple inputs are pending, one from the tester, and others from the buffers. Test sequences should be generated so that they may traverse the transitions of the context without such a race condition.

3.2 Algorithm for graph transformation

The algorithm we use in this approach is described in Algorithm 1, which was presented in [6]. That algorithm constructs a new graph $G' = (V', E')$ from the original graph G for generating risk-free test sequences. It creates a new state $v' \in V'$ by multiplying the original state $v \in V$ by buffer configurations. In this process, all possible buffer configurations with up to b_i inputs in buffer B_i at I_i are constructed by examining all outgoing edges of v in a breadth-first-search manner, where b_i is the buffer size of B_i. Several copies may be generated in E' for each edge $e \in E$, based on the class of transition e. In general, each edge in E belongs to one of the four classes defined as follows [6].

- *Class 1*: e is triggered by an input from and generates output(s) to the tester.
- *Class 2*: e is triggered by an input from the tester and generates an output $o_{q,l}$ at I_q, which is buffered in B_q to create a new configuration.
- *Class 3*: e is triggered by $a_{p,k}$ from I_p and generates output(s) to the tester, which is extracted from B_p to create a new configuration.
- *Class 4*: e is triggered by an input $a_{p,k}$ from I_p and generates an output $o_{q,l}$ at I_q.

Algorithm 1. Graph conversion from G to G'
Step.1: initialize r', the root of G', as $(r, \phi, \ldots, \phi)$.
Step.2: initialize E' as empty set, and V' as $\{r'\}$.
Step.3: initialize Q, queue of vertices, as V'.
Step.4: repeat until Q is empty.

(a) extract $v' = (v_{start}, \mathring{B}_1, \ldots, \mathring{B}_F)$ as the first element from Q, where $(\mathring{B}_1, \ldots, \mathring{B}_F)$ is the current configuration.
(b) for each outgoing edge $e = (v_{start}, v_{end}) \in E$ do
  i. identify the class k of e.
  ii. given $(\mathring{B}_1, \ldots, \mathring{B}_F)$ and class k definition, construct:
  - new configuration $(\mathring{B}_1, \ldots, \mathring{B}_F)$
  - new vertex $v'_{new} = (v_{end}, \mathring{B}_1, \ldots, \mathring{B}_F) \in V'$
  - new edge $e'_{new} = (v', v'_{new}) \in E'$

(c) add new edges in E' if and only if inputs in $(\mathring{B}_1, \ldots, \mathring{B}_F)$ cannot trigger other edges outgoing from v_{start}.
(d) append to Q the end vertices $v'_{new} \in V'$ of new edges included in E'.

Step.5: remove from V' all vertices from which r' cannot be reached.
Step.6: remove from E' all edges incident to such vertices.

As mentioned in the introduction, the model presented in section 2 is constructed under the assumption that an inaccessible interface consists of FIFO-type buffers. In practice, in addition to (or instead of) FIFO-type buffers, interrupt-driven mechanism may be used as an interaction technique, which can be considered also an inaccessible interface. There may be various choices to implement interfaces, Therefore, test sequences generated for a system with the assumption that all interfaces is composed of FIFO-type buffers, may result in nondeterminism when applied in real testing if that assumption is not valid. To eliminate this type of nondeterminism in test sequences, the system model should have a buffer size $b_i = 1$. Another issue is the number of inputs that can be buffered simultaneously at all inaccessible interfaces of the context. When inputs are allowed to be buffered simultaneously at several interfaces, even a buffer size equal to 1 may not prevent nondeterministic behavior during testing. To avoid this type of nondeterminism during testing, the model presented in section 2 should be used to generate tests with the restriction that, at any time, inputs may be buffered in only one inaccessible interface of the context. According the above consideration, a refinement of the graph conversion algorithm is also given [6]: "at most one input at only one inaccessible interface utilized at any time."

By the graph transform algorithm and the refinement, the new graph $G'(V',E')$ is illustrated as shown in Fig.3. A minimum-cost tour of G' where each original edge from G is covered at least once can be generated by the Rural Chinese Postman (RCP) method [10]. There are two sets of edges of E': E'_c, the set of *mandatory* edges, and $(E' - E'_c)$, the set of *optional* edges. The minimum-cost tour should include each transition in the mandatory set at least once and each transition in the optional set zero or more times. The sets E, E' and E'_c are as follows [6]:

$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}\}$

$E' = \{e_{1.0}, e_{2.2}, e_{3.1}, e_{4.3}, e_{5.0}, e_{5.3}, e_{6.0}, e_{6.3}, e_{7.0}, e_{8.0}, e_{9.3}, e_{10.0}, e_{10.1}, e_{12.0}, e_{13.0}, e_{13.1}, e_{13.2}, e_{14.0}, e_{14.1}\}$

$E'_c = \{e_{1.0}, e_{2.2}, e_{3.1}, e_{4.3}, e_{5.0}, e_{6.0}, e_{7.0}, e_{8.0}, e_{9.3}, e_{10.0}, e_{11.2}, e_{12.0}, e_{13.0}, e_{14.0}\}$

Given the above sets E' and E'_c, the minimum-length test sequence for G' is shown in Table 2. Note that, for simplicity, the Unique Input Output (UIO) sequences [11] for state verification are not included in this sequence and a step with '→' indicates that the corresponding edge is tested in that step.

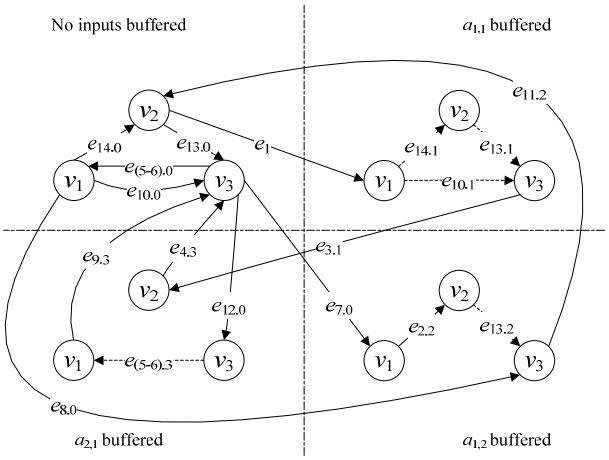

Fig. 3 Graph Transformation applied to the graph of Fig.2 [6]

Table 2 Minimum-length test sequence for the context of Fig.2 [6]

Step	*Edge*	*Input*	*Output*	*Step*	*Edge*	*Input*	*Output*
→1	e_{14}	$C?x_{14}$	$C!y_{14}$	→10	e_{12}	$C?x_{12}$	$M_2!o_{2,1}$
→2	e_{13}	$C?x_{13}$	$C!y_{13}$	11	e_5	$C?x_5$	$C!y_5$
→3	e_5	$C?x_5$	$C!y_5$	→12	e_9	$M_2?a_{2,1}$	$C!y_9$
→4	e_8	$C?x_8$	$M_1!o_{1,2}$	→13	e_7	$C?x_7$	$M_1!o_{1,2}$
→5	e_{11}	$M_1?a_{1,2}$	$C!y_{11}$	→14	e_2	$M_1?a_{1,2}$	$M_1!o_{1,2}$
→6	e_1	$C?x_1$	$M_1!o_{1,1}$	15	e_{13}	$C?x_{13}$	$C!y_{13}$
→7	e_{10}	$C?x_{10}$	$C!y_{10}$	16	e_{11}	$M_1?a_{1,2}$	$C!y_{11}$
→8	e_3	$M_1?a_{1,1}$	$M_2!o_{2,1}$	17	e_{13}	$C?x_{13}$	$C!y_{13}$
→9	e_4	$M_2?a_{2,1}$	$C!y_4$	→18	e_6	$C?x_6$	$C!y_6$

3.3 Nondeterminism in context testing

The graph conversion of algorithm 1 may be directly used for generating a minimum-cost test sequence for context testing, and the refinement of algorithm 1 established to eliminate nondeterminism may be also necessary. However, that refinement cannot be applied for test sequence generation in some particular context specifications. In such specifications, therefore, test sequences would most likely have nondeterminism.

Example 2. Consider the following possible test sequence for the context of Fig.2. The edge information of this context is described in Table 3. Note that Table 3 is identical to Table 1 except the edges e_1, e_6, and e_{13}.

e_{10}, e_7, $\underline{e_2}$, $\underline{e_1}$, $\underline{e_2}$, $\underline{e_4}$, e_{11}, e_1, e_9, e_6, e_{14}, e_{13}, e_3, e_4, e_{12}, e_5, e_9

Table 3 The edge information of Fig.2 for Example 2

Edge	*Input*	*Output*	*Edge*	*Input*	*Output*
e_1	$C?x_1$	$M_2!o_{2,1}$	e_8	$C?x_8$	$M_1!o_{1,2}$
e_2	$M_1?a_{1,2}$	$M_1!o_{1,2}$	e_9	$M_2?a_{2,1}$	$C!y_9$
e_3	$M_1?a_{1,1}$	$M_2!o_{2,1}$	e_{10}	$C?x_{10}$	$C!y_{10}$
e_4	$M_2?a_{2,1}$	$C!y_4$	e_{11}	$M_1?a_{1,2}$	$C!y_{11}$
e_5	$C?x_5$	$C!y_5$	e_{12}	$C?x_{12}$	$M_2!o_{2,1}$
e_6	$C?x_6$	$M_1!o_{1,1}$	e_{13}	$M_1?a_{1,1}$	$M_1!o_{1,1}$
e_7	$C?x_7$	$M_1!o_{1,2}$	e_{14}	$C?x_{14}$	$C!y_{14}$

The underlined portion of the above test sequence first traverses e_2, which causes input $a_{1,2}$ to be buffered at I_1 and the context to move to state v_2 where that buffered input $a_{1,2}$ cannot be consumed. Since transitions e_4 and e_{13} requires inputs $a_{2,1}$ and $a_{1,1}$ to be applied to the context from M_1 and M_2 respectively, transition e_1 can be uniquely traversed from the current state v_2. Transition e_1 moves the state of the context into v_1 with input $a_{2,1}$ buffered at I_2. As $a_{1,2}$ was generated earlier than $a_{2,1}$, the second e_2 is expected to be triggered ahead of e_9 . Actually, due to the unknown response time of the interfaces, $a_{1,2}$ may be applied to the context at an arbitrary time: earlier than, later than, or simultaneously with $a_{2,1}$. In this situation, the refinement condition "at most one input at only one of the context's inaccessible interface utilized at any time" is not fulfilled and nondeterministic behaviors may happen, which may result in that test sequence invalid.

4. Extension for Context Testing

4.1 Condition for avoiding nondeterminism

In order to eliminate nondeterminism in context testing, the specification of the context in an embedded system should satisfy the refinement condition. For presenting our modified method for context testing to avoid nondeterminism, we introduce the following concept.

Definition 2. An edge e_i ($\in E$) is an *independent edge* if $e_i \in T_\phi$, and a path $p = e_1e_2 \ldots e_m$, is an *independent path* if and only if e_i is an independent edge for all i ($1 \le i \le m$).

Example 3. In Example 1, e_5 is an independent edge, and $e_5e_{14}e_{13}$ is an independent path.

Definition 3. $T\_SET(i, j)$ ($\subseteq V$) and $U\_SET(i, j)$ ($\subseteq V$) are defined as $\{ head(e) \mid e \in T_{i,j} \}$ and $\{ tail(e) \mid e \in U_{i,j} \}$ respectively. $B\_SET(i, j)$ ($\subseteq V$) and $C\_SET(i, j)$ ($\subseteq V$) are defined as $\{ v \mid v \in U\_SET(i,j) \wedge v \notin T\_SET(i,j) \}$ and $\{ v \mid v \in T\_SET(i,j) \wedge v \notin U\_SET(i,j) \}$ respectively.

Example 4. In Example 1, $T\_SET(1,1) = \{ v_3 \}$, $T\_SET(1,2) = \{ v_1, v_3 \}$, and $T\_SET(2,1) = \{ v_2, v_1 \}$. $U\_SET(1,1) = \{ v_1 \}$, $U\_SET(1,2) = \{ v_2, v_1, v_3 \}$, and $U\_SET(2,1) = \{ v_2, v_3 \}$. $B\_SET(1,1) = \{ v_1 \}$, $B\_SET(1,2) = \{ v_2 \}$, and $B\_SET(2,1) = \{ v_3 \}$. $C\_SET(1,1) = \{ v_3 \}$, $C\_SET(1,2) = \phi$, and $C\_SET(2,1) = \{ v_1 \}$.

Proposition 1. A test sequence can be generated which does not cause multiple inputs pending in interface buffers during testing if the following two conditions are satisfied:

- *Condition A*: for each input $a_{i,j} \in A_i$, if $B\_SET(i, j)$ is not empty, then $T\_SET(i, j)$ is also nonempty; and for each $v \in B\_SET(i, j)$, $\exists v' \in T\_SET(i, j)$ such that there is at least one independent path from v to v'.
- *Condition B*: for each input $a_{i,j} \in A_i$, if $C\_SET(i, j)$ is not empty, then $B\_SET(i, j)$ is also nonempty; and for each $v \in C\_SET(i, j)$, $\exists v' \in B\_SET(i, j)$ such that there is at least one independent path from v' to v.

Proof) For each $v \in V$, there are four classes related to $U\_SET(i, j)$ and $T\_SET(i, j)$.

- *Class 1*: $\{ v \mid v \notin U\_SET(i, j)$ and $v \notin T\_SET(i, j) \}$. For each state v in this class, no outgoing transitions can trigger the input $a_{i,j}$ to be buffered, so it is impossible to occur multiple inputs pending in buffers for $a_{i,j}$.
- *Class 2*: $\{ v \mid v \in U\_SET(i, j)$ and $v \notin T\_SET(i, j) \}$. For each state v in this class, $v \in B\_SET(i, j)$. According to the condition A, there is an independent path from v' to v. Therefore, in any state v, $a_{i,j}$ is consumed before another input is buffered.
- *Class 3*: $\{ v \mid v \notin U\_SET(i, j)$ and $v \in T\_SET(i, j) \}$. For each state v in this class, $v \in C\_SET(i, j)$. The condition B makes sure that the transition $T_{i,j}$ for state v can be traversed without multiple inputs pending in buffers for $a_{i,j}$.
- *Class 4*: $\{ v \mid v \in U\_SET(i, j)$ and $v \in T\_SET(i, j) \}$. For each state v in this class, obviously, input $a_{i,j}$ is consumed immediately; no multiple inputs can be pending in buffers for $a_{i,j}$.■

Example 5. Consider the context of Fig.2 with its specification in Table 3. We have:

- $T\_SET(1,1) = \{ v_2, v_3 \}$, $T\_SET(1,2) = \{ v_1, v_3 \}$, $T\_SET(2,1) = \{ v_1, v_2 \}$.
- $U\_SET(1,1) = \{ v_1, v_3 \}$, $U\_SET(1,2) = \{ v_1, v_2, v_3 \}$, $U\_SET(2,1) = \{ v_1, v_2, v_3 \}$.
- $B\_SET(1,1) = \{ v_1 \}$, $B\_SET(1,2) = \{ v_2 \}$, $B\_SET(2,1) = \{ v_3 \}$.
- $C\_SET(1,1) = \{ v_2 \}$, $C\_SET(1,2) = \phi$, $C\_SET(2,1) = \phi$.

For input $a_{1,2}$, *B_SET*(1,2) has an element v_2 and *T_SET*(1,2) has v_1 and v_3, however, there are no independent path from v_2 to v_1 nor to v_3. Therefore, the proposition 1 cannot be satisfied for that context and accordingly the multiple pending inputs cannot be avoided during testing.

4.2 Transition addition for better testability

A well-designed context of an embedded system would utilize safe and sound inaccessible interfaces between the context and the embedded components. We expect that, in such well-designed contexts, proposition 1 will be satisfied and thus a safe test sequence can be found for the context without nondeterminism. On the contrary, there may be some contexts where proposition 1 is not fulfilled like the context of example 5.

In order to solve that problem, we propose an intuitive digraph transform algorithm which checks the given context whether it satisfies proposition 1, and then attempts to add special transitions for test in the context if proposition 1 is not satisfied. Owing to those additional transitions a safe test sequence can be generated without the nondeterminism problem. Algorithm 2 shows how to transform an original graph $G(V, E)$ to a new graph $G''(V'', E'')$.

Algorithm 2. Graph conversion from G to G''

Step.1: For each vertex v_j ($\in V$), create a set of vertices $v_j^{\phi}, v_j^{\{a_{i,1}\}}, ..., v_j^{\{a_{i,n_i}\}}$ in V''.

Step.2: For each edge $e_{j,k} = (v_j, v_k, x/y)$ ($\in E$), create the following edge(s) in E'':

- $(v_j^{\phi}, v_k^{\phi}, x/y)$ and a set of dashed edges, $(v_j^{\{a_{i,1}\}}, v_k^{\{a_{i,1}\}}, x/y), ..., (v_j^{\{a_{i,n_i}\}}, v_k^{\{a_{i,n_i}\}}, x/y)$, if $e_{j,k} \in T_{\phi}$
- $(v'_j, v'_k, x/y)$, if $e_{j,k} \notin T_{\phi}$, such that:
  - $v'_j = v_j^{\{a_{i,n_i}\}}$, if $x \in A_i$ and $x = a_{i,n_i}$; otherwise, $v'_j = v_j^{\phi}$.
  - $v'_k = v_k^{\{a_{i,m_i}\}}$, if $y \in O_i$ and y = trigger input a_{i,m_i}; otherwise, $v'_k = v_k^{\phi}$.

Step.3: For each vertex $v_k^{\{a_{i,j}\}}$ ($\in V''$) which has solid incoming edges, add a set of transitions $e_{\phi} = (v_k^{\{a_{i,j}\}}, v_l^{\{a_{i,j}\}}, x_{\phi}/y_{\phi})$ such that $v_l \in T\_SET(i, j)$, if $v_k^{\{a_{i,j}\}}$ has no outgoing edges.

Step.4: For each vertex v ($\in V''$) which has solid outgoing edges, remove from E'' all dashed outgoing edges of v.

Step.5: For each vertex v ($\in V''$) which has no outgoing edges, remove all incoming edges of v both from V'' and from E''.

Since algorithm 2 requires to modify the context by adding some transitions $e_{\phi} = (v_j, v_k, x_{\phi}/y_{\phi})$, it is recommended to be applied during the design phase of the context when its modification will not cost high.

Example 6. Consider the context of Fig.2 with its specification in Table 3. When algorithm 2 is applied so as to solve the nondeterminism problem, a new digraph G'' are generated as shown in Fig.4. In that figure, the solid edges are *mandatory* edges and the dashed edges are *optional*. The minimum-length test sequence for G'' are shown in Table 4.

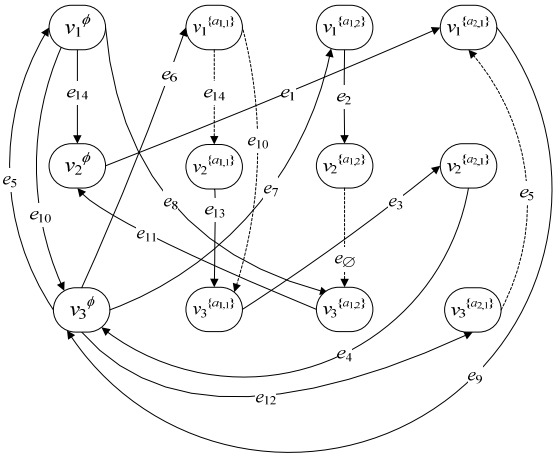

Fig. 4 New digraph generated by the algorithm 2 for example 6

Table 4 Minimum-length test sequence for example 6

Step	*Edge*	*Input*	*Output*	*Step*	*Edge*	*Input*	*Output*
→1	e_{10}	$C?x_{10}$	$C!y_{10}$	→10	e_{11}	$M_1?a_{1,2}$	$C!y_{11}$
→2	e_6	$C?x_6$	$M_1!o_{1,1}$	→11	e_1	$C?x_1$	$M_2!o_{2,1}$
→3	e_{14}	$C?x_{14}$	$C!y_{14}$	→12	e_9	$M_2?a_{2,1}$	$C!y_9$
→4	e_{13}	$M_1?a_{1,1}$	$M_1!o_{1,1}$	→13	e_{12}	$C?x_{12}$	$M_2!o_{2,1}$
→5	e_3	$M_1?a_{1,1}$	$M_2!o_{2,1}$	→14	e_5	$C?x_5$	$C!y_5$
→6	e_4	$M_2?a_{2,1}$	$C!y_4$	15	e_9	$M_2?a_{2,1}$	$C!y_9$
→7	e_7	$C?x_7$	$M_1!o_{1,2}$	16	e_5	$C?x_5$	$C!y_5$
→8	e_2	$M_1?a_{1,2}$	$M_1!o_{1,2}$	→17	e_8	$C?x_8$	$M_1!o_{1,2}$
9	e_{ϕ}	$C?x_{\phi}$	$C!y_{\phi}$	18	e_{11}	$M_1?a_{1,2}$	$C!y_{11}$

5. Concluding Remarks

In an embedded system, where the embedded components interact with the environment through the context, a tester cannot directly control the interfaces between the context and the embedded components. Therefore, race conditions and/or nondeterministic behaviors may happen during testing. This paper first showed a basic solution technique generated for handling semi-controllable interface and also its incompleteness due to no considerations on the nondeterminism condition. This paper then presented a complimentary technique to identify the possibility of nondeterminism in the context and to add some transitions to eliminate possible nondeterminism in test sequences. This technique might be also very useful to refine context specifications during the context design phase for the development of safer embedded systems.

How to relax the refinement condition for avoiding nondeterminism is our major interest for further study of this work in order to apply to various types of embedded systems. Another interesting issue is test generation for the embedded systems under multiple testers for easier testing, which may address synchronization problems [12].

Acknowledgment

This paper was supported by Research Fund, Kumoh National Institute of Technology.

References

[1] L. P. Lima Jr. and A. R. Cavalli, "A pragmatic approach to generating test sequence for embedded systems," Proc. IFIP Int'l Workshop on Testing of Communicating Systems(IWTCS), Cheju Island, Korea, Sep 1997.

[2] A. F. Petrenko and N. Yevtushenko, " Fault detection in embedded components," Proc. IFIP Int'l Workshop on Testing of Communicating Systems(IWTCS), Cheju Island, Korea, Sep 1997.

[3] A. F. Petrenko and N. Yevtushenko, and G. von Bochmann, "Fault models for testing in context," Proc. IFIP Joint Int'l Conf. FORTE/PSTV, Kaiserslautern, Germany, Oct 1996.

[4] N. Yevtushenko, A. R. Cavalli, and L. P. Lima Jr., "Test suite minimization for testing in context," Proc. IFIP Int'l Workshop on Testing of Communicating Systems (IWTCS)}, pages 127-145, Tomsk, Russia, Sep 1998.

[5] A. F. Petrenko, N. Yevtushenko, G von Bochmann, and R. Dssouli, "Testing in context: Framework and test derivation," Computer Communications, 19(14):1236-1249,1996.

[6] M. A. Fecko, M. U. Uyar, A. S. Sethi, and P. D. Amer, "Issues in conformance testing: Multiple semicontrollable interfaces," Proc. IFIP Joint Int'l Conf. FORTE/PSTV, pages 111-126, Paris, France, Nov 1998.

[7] The Department of Defense (DoD), "Military Standard--Interoperability Standard for Digital Message Device Subsystems (MIL-STD 188-220B)," Jan. 1998.

[8] A. Gibbons, "Algorithmic Graph Theory," Cambridge University Press, Cambridge, MA, 1985.

[9] M. Gondran, M. Minoux, and S. Vajda, "Graphs and Algorithms," Wiley-Interscisence, Series in Discrete Mathematics, New York, NY, 1984.

[10] 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," IEEE Trans. on Communications, 39(11):1604-1615, Nov 1991.

[11] K. Sabnani and A. Dahbura, "A protocol test generation procedure," Computer Networks and ISDN Systems, Vol.15, pp.285-297, 1988.

[12] B. Sarikaya, G.v. Bochmann, "Synchronization and specification issues in protocol testing," IEEE Trans. Comm}. 32 (1984) pp.389-395.

Qi-Ping Yang received the B.S. degree in communication engineering from Jilin University, Changchun, China, in 2001, and the M.S. degree in computer engineering from Kumoh National Institute of Technology (KIT), Gumi, Korea, in 2006. He is currently a Ph.D. candidate in the School of Computer and Software Engineering at KIT. His current research interest is software testing techniques and next generation mobile networks.

Tae-Hyong Kim received the B.S. and M.S. degrees, from Yonsei University in 1992 and 1995, respectively, and a Ph.D. degree in electrical and electronic engineering from the same university in 2001. He was a postdoctoral fellow at the School of Information Technology and Engineering (SITE) in University of Ottawa from 2001 to 2002. He is currently an assistant professor at the School of Computer and Software Engineering (SCSE) in Kumoh National Institute of Technology (KIT), Korea. His current research interests include software and protocol specification, verification and testing techniques, communication protocols, and next generation mobile networks. He is an IEEE member and also a member of the SDL Forum Society.