Generating Reduced Finite State Machine from Concurrent Scenarios using Static Partial Order Method

Nam Hee Lee and Sung Deok Cha

Division of Computer Science and AITrc/SPIC/IIRTRC‡
Department of Electrical Engineering and Computer Science, KAIST
373-1, Gusung-Dong, Yusung-Gu, Daejon, Korea
{nhlee,cha}@salmosa.kaist.ac.kr

Finite state machine (FSM) representation is widely used to perform behavioural analysis and generate test cases from a set of hierarchically organized scenarios written in Message Sequence Charts (MSCs). Brute-force approach of translating MSCs into FSM is impractical, especially when scenarios are executed concurrently. In this paper, we describe how to identify a sequence of message exchanges that are semantically equivalent and apply partial order method to reduce the number of transitions in the FSM. We demonstrate that the proposed technique is scalable by describing the results of a case study in which reduced FSM was automatically generated from a partial specification of digital TV software.

ACM Classification: D.2.4: (Software-Software, Engineering-Software/Program, Verification-Formal methods)

1. INTRODUCTION

Modern electronic devices such as digital TV, web phones or cellular phones heavily depend on software. As a product's life-cycle is usually short, testing of embedded software must be automated as much as possible. Finite state machine (FSM) has long been a popular notation in capturing intended system behaviour, identifying potential flaws in the model, and automatically generating test cases. However, modelling of large and complex system behaviour with primitive notations like FSM is impractical, and high-level specification languages are often used. Message Sequence Charts (MSCs) (Z.120, 2000) is a popular choice in industry because: (1) it is an internationally standardized and formal language whose graphical notations are easily understood; and (2) there are various constructs, including high-level MSC and in-line expressions (e.g., iterative, conditional, or concurrent execution), which allow complex behaviour to be hierarchically organized.

Test cases can be obtained from a formal specification by separately modelling required behaviour of each subsystem (or component), translating individual models into FSM, constructing a global FSM (GFSM) as a product of component FSMs, and generating test cases from GFSM based on chosen coverage criteria. Communicating Extended FSM (CEFSM) supports extended notations for expressing communication mechanisms and data manipulation. SDL (Z.100, 1992), Estelle (Budkowski and Dembinski, 1987) and Statecharts (Harel, 1987) are examples of

‡ Partially supported by the Advanced Technology Research Centre (AITrc)/SPIC/IIRTRC

Manuscript received: 25 March 2003
Communicating Editor: Amos Omondi

specifications that can be translated to CEFSM. In the case of SDL or Estelle, several CEFSM are composed to form GFSM (Luo *et al*, 1994), and unfolding techniques are used to generate GFSM from Statecharts (Hong *et al*, 2000).

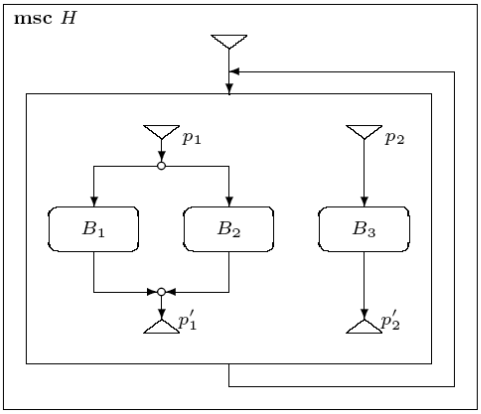
In scenario-based software development, however, it is common to specify and maintain scenarios as separate and partial specifications. Scenario-based specification languages include MSC (Z.120, 2000), UML Sequence Diagram (Booch *et al*, 1998), and LSC (Harel and Kugler, 2002). Scenario usually involves interaction among multiple components, and multiple scenarios need to be combined to perform behavioural analysis or generate test cases. When several scenarios can be executed concurrently (e.g., processing events generated by physically distinct input devices), GFSM must take all possible interleaving of input events into consideration. However, little work has been done on how to combine concurrent scenarios and generate compact GFSM. Lee and Cha (2003) described how a set of scenarios can be translated into GFSM. Their approach is based on the assumption that configurations of embedded systems do not change unless the values of state variables change and that each scenario is activated only when guarding conditions are met. They demonstrated that the degree of the state explosion problem encountered when translating MSCs into GFSM can be controlled. While the technique work is effective when scenarios are composed only sequentially, the number of transitions grows too quickly when some scenarios are executed concurrently.

Partial order methods (Valmari, 1992; Katz and Peled, 1992; Godefroid and Wolper, 1993; Godefroid, 1996) refer to a collection of state exploration techniques intended to relieve the state explosion problem often encountered when verifying concurrent programs. All classical state space search techniques are similar in that only the set of enabled transitions T are selectively explored at each state. While partial order methods are similar in concept, difference between the two is that properties to be verified influence how T is computed. Conceptually distinct states are combined into one if they are irrelevant as far as the property is concerned. Therefore, application of partial order methods on the same model with respect to different properties results in different T and different state spaces. Example approaches include stubborn set (Valmari, 1992), ample set (Katz and Peled, 1992), or persistent set (Godefroid and Wolper, 1993).

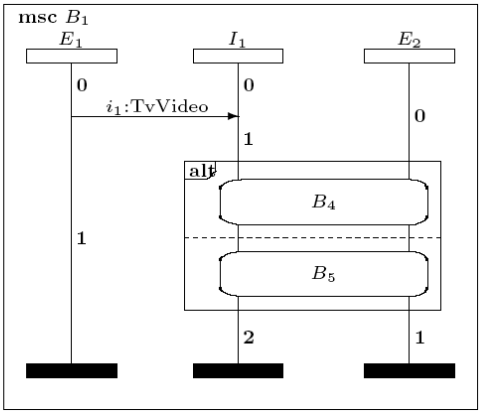
Many state-based automatic verification tools (Godefroid *et al*, 1996; Holzmann, 1997) support partial order reduction methods because one can analyze large and complex problems. A large part of the calculations necessary for applying partial order reduction is performed at run-time, and Kurshan *et al*, (1998) proposed a cycle closing condition to be checked at compile time to enhance efficiency. Their approach is based on the notion of sticky transitions that break all cycles of the reduced state graph. Thus, local cycles that change some resource in a monotonic way can be exempted from the search for sticky transitions. Such a set can be found by applying static analysis on control flow graph of each system process. This approach successfully identified dependency between loops of different processes.

In this paper, we propose the notion of an independent region as a means of reducing the number of transitions in GFSM generated from concurrent scenarios. Independent regions are identified via static analysis on a set of hierarchical and concurrent MSCs. We demonstrate effectiveness of the proposed approach by modelling a part of digital TV software whose specification consists of 34 MSCs. It is quite complex in that there are more than 800,000 lines of Java and C code.

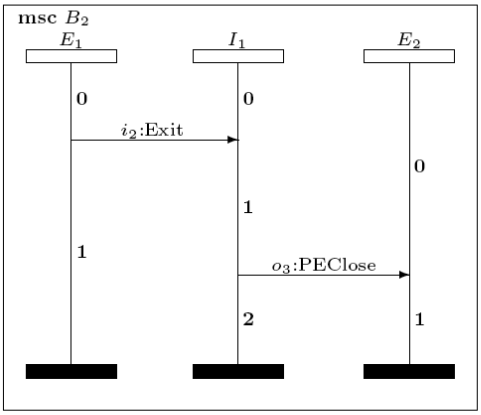
The rest of this paper is organized as follows. In Section 2, an algorithm to transform a set of hierarchically organized MSCs into a GFSM is explained, and we explain heuristics useful in identifying independent regions. We report results obtained from our case study in Section 3. Finally, Section 4 concludes the paper and discusses topics worthy of further research.



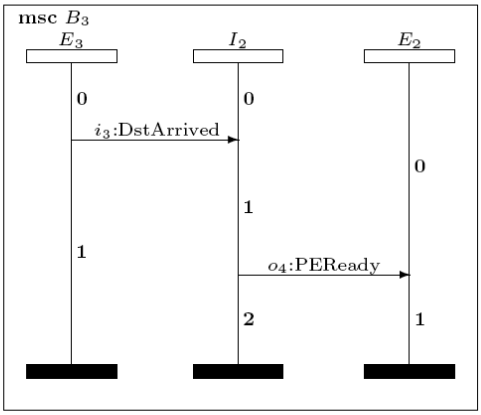
(a) H



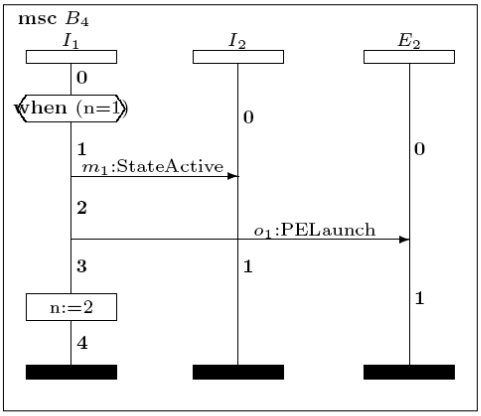
(b) B_1



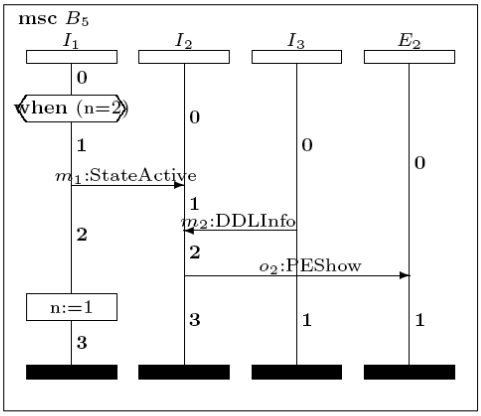
(c) B_2



(d) B_3



(e) B_4



(f) B_5

Figure 1: Example

2. INDEPENDENT REGIONS AND GFSM GENERATION

When specifying required behaviour of embedded software, basic MSC (bMSC) separately specifies how the system is to react to an input event, and high-level MSCs (hMSCs) describe how bMSCs are composed: sequentially, alternatively, or concurrently. Figure 1 indicates that bMSCs B_1 and B_2 are executed alternatively and that bMSC B_3 operates independent of B_1 and B_2. In further specifying B_1, either bMSC B_4 or B_5 is invoked depending on the value of the state variable 'n'. Each bMSCs specifies the required message exchanges, and the effect of executing a scenario on the system variable is indicated. For example, B_4 is invoked when the guarding condition, '**when** (n=1)', is met, and after sending messages m_1 and o_1, variable 'n' is changed to two.

When translating a set of MSCs into GFSM, one may translate each bMSC into separate behaviour automata (BA), as shown in Figure 2, by assigning monotonically increasing integer values to denote progress. Each message is expressed as a transition, and all possible interleaving of concurrent automata results in GFSM. While straightforward in concept, such a brute-force approach quickly becomes impractical due to state and transition explosion problems. When generating GFSM from concurrent scenarios, the transition explosion problem must be addressed. Given two concurrent behaviour automata whose path length is m and n, respectively, there are ${}_{(m+n)}C_n$ possible interleaving sequences, and the number of transitions increases exponentially as either m or n becomes large. For example, the number of transitions in GFSM grows to 59 when bMSCs B_3, containing only two message exchanges, is included.

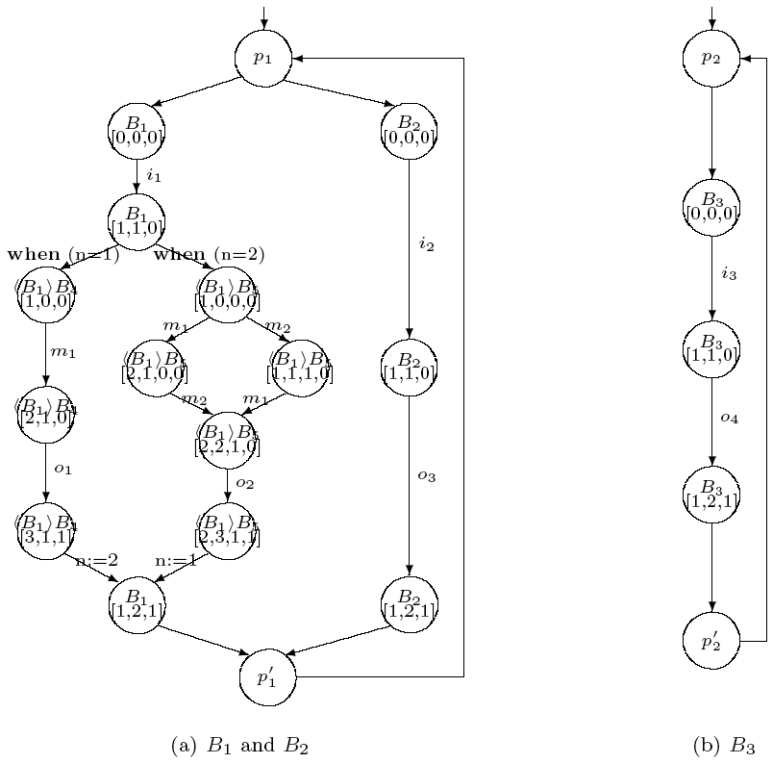


Figure 2: Behaviour automata for each concurrent scenario

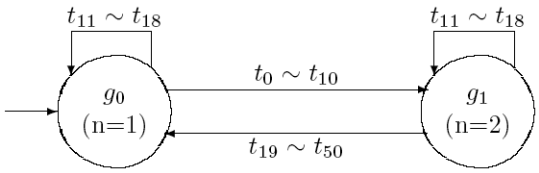


Figure 3: GFSM for concurrent scenarios, B1, B2 and B3

While the proposed approach to addressing the transition explosion problem is similar in concept to the technique proposed by Kurshan *et al* (1998), we focus on the changes made to scenario guarding conditions and state variables instead of cycles. Intuitively speaking, transitions are considered independent if the order of their occurrence is irrelevant to the eventual outcome. Consider the two transitions, t_0 and t_1, shown in Figure 3. Transition t_0 and t_1 represent the message sequences $\langle i_1; i_3; o_4; m_1; o_1\rangle$ and $\langle i_3; i_1; o_4; m_1; o_1\rangle$, respectively. The messages i_1 of the scenario B_1 and i_3 of the scenario B_3 are independent because the execution of two transitions in any order leads to the same state, 'n = 2', and both t_0 and t_1 need not be included in the GFSM. Independency among transitions appearing in different behaviour automata can be formalized by the following definition (adapted from Katz and Peled (1992)).

Definition 1 *Two transitions $t_1 \in BA_1$ and $t_2 \in BA_2$ found in different behaviour automata are independent if for every state g in GFSM the following two conditions hold:*

1. *if t_1 and t_2 are enabled in g and $g \xrightarrow{t_1} g_0$ (or $g \xrightarrow{t_2} g_0$), then t_2 (or t_1) remains enabled in g_0; and*
2. *if t_1 and t_2 are enabled in g, then there is a unique state g_0 such that $g \xrightarrow{t_1 t_2} g_0$ and $g \xrightarrow{t_2 t_1} g_0$*

A previous approach (Lee and Cha, 2003) required the whole state spaces of the GFSM be generated to find independent transitions. Unfortunately, the technique is impractical when applied to large and complex systems due to a transition explosion problem. In this paper, as an improvement, we propose a heuristic search algorithm where only the state spaces of individual behaviour automaton, not that of the GFSM, are analyzed. We first introduce the concept of an atomic region.

Definition 2 *An atomic region is a sequence of transitions in a behaviour automaton which can be completed without interruption by any transitions found in other concurrent behaviour automata.*

When generating GFSM by computing all possible concurrent events, atomic regions can reduce the number of transitions. For example, only considering bMSCs B_3 and B_4, six interleaving sequences, $\langle m_1; o_1; i_3; o_4\rangle$, $\langle m_1; i_3; o_1; o_4\rangle$, $\langle m_1; i_3; o_4; o_1\rangle$, $\langle i_3; m_1; o_1; o_4\rangle$, $\langle i_3; m_1; o_4; o_1\rangle$, and $\langle i_3; o_4; m_1; o_1\rangle$, are possible. However, if two messages, m_1 and o_1, were grouped as an atomic region, GFSM contains only three interleaving sequences, $\langle m_1; o_1; i_3; o_4\rangle$, $\langle i_3; m_1; o_1; o_4\rangle$, and $\langle i_3; o_4; m_1; o_1\rangle$. Needless to say, as transitions included in atomic regions grow, GFSM would contain a smaller number of transitions.

If a transition resets some state variables which are used in guarding the condition of concurrent behaviour automata, some transitions may become enabled (or disabled) as the result of executing

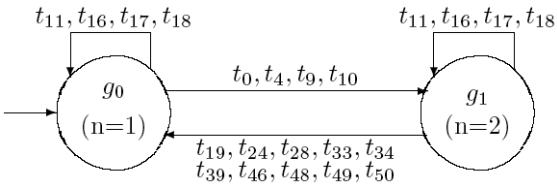


Figure 4: Reduced GFSM applying the independent region

the transition. Consequently, two MSCs' constructs, 'action' and 'guarding condition', determine whether or not two transitions are related. That is, message exchanges between these constructs are considered independent of the other if they do not affect guarding conditions and state variables.

Definition 3 *An independent region is a sequence of message exchanges in a bMSCs which are located between the 'action' and 'guarding condition' constructs.*

It must be emphasized that the above assumption is justified because GFSM generated by our approach is characterized by the different configurations the system can be in and that not all message exchanges initially included in the independent regions are truly independent. They are just candidate sets. For example, m_1 and o_1 of the bMSC B_4 form an independent region because these messages are located between '**when** (n = 1)' and 'n := 2', and i_3 and o_4 of the bMSC B_3 form another independent region. However, i_1 is not included in the independent region of $\langle m_1, o_1 \rangle$ because the guarding condition, '**when** (n = 1)', is affected by other concurrent behaviour automata. Consequently, GFSM includes interleaved execution of three events: i_1, $\langle m_1, o_1 \rangle$, and $\langle i_3, o_4 \rangle$, and only three transitions, $\langle i_1; m_1; o_1; i_3; o_4 \rangle$, $\langle i_1; i_3; o_4; m_1; o_1 \rangle$, and $\langle i_3; o_4; i_1; m_1; o_1 \rangle$ are generated. Likewise, Figure 1 contains independent regions, $\langle i1 \rangle$, $\langle m_1, o_1 \rangle$, $\langle m_1, m_2, o_2 \rangle$, $\langle m_2, m_1, o_2 \rangle$, $\langle i_2, o_3 \rangle$, and $\langle i_3, o_4 \rangle$, and there are only 22 transitions, as opposed to 59, in GFSM (See Figure 4).

Similarly, the more concurrent scenarios there are in the specification, the greater reduction is accomplished by the proposed approach. Typical embedded systems must process inputs arriving concurrently from many sensors and input devices, and our approach of finding independent regions via static analysis plays a key role in determining whether or not GFSM can be practically generated. GFSM generated by the independent region preserves the original behaviour of the system described by concurrent MSCs. The proof is given in Appendix A. However, it does not necessarily include the maximum set of independent transitions. For example, only two transitions t_0 and t_{10} are needed for the transition from g_0 to g_1 in our example, but our approach results in four transitions. In this paper, we analyze each bMSC separately to find independent regions, and an optimal algorithm requires multiple bMSCs be searched simultaneously. Such algorithm has to deal with a state explosion problem of another kind, and it is beyond the scope of this paper.

Once independent regions are identified, concurrent bMSCs can be combined and the corresponding GFSM generated by applying the Algorithm 1 which traverses all concurrent behaviour automata. All paths in each behaviour automata must be executed with guarding conditions and all transitions in concurrent behaviour automata must be interleaved except those belonging to the independent regions. s_0 indicates the initial state of the GFSM. The vector *pnode* points to the current location of each concurrent behaviour automata, and stack includes all interleaving points.

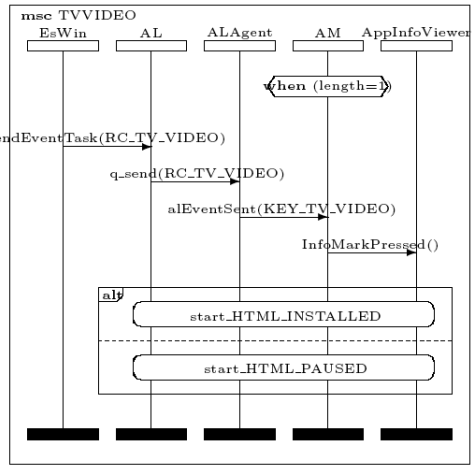
Algorithm 1

```
1:  initialize pnode to the start nodes of each behaviour automata;
2:  push s_0 and pnode onto Stack;
3:  while Stack ≠ φ {
4:    pop s and pnode from Stack;
5:    if all p ∈ pnode is visited then {
6:      if s is NOT visited then {
7:        add s to the state of GFSM;
8:      }
9:     add path to the transition of GFSM;
10:   }
12:   path = φ;
13:   for all p ∈ pnode do {
14:     T = out transitions of p;
15:     for all t ∈ T do {
16:       if t satisfies the guarding condition then {
17:         if t is in independent region then {
18:           path = path + t;
19:           update s and p;
20:           continue;
21:         }
22:         path = path + t;
23:         update s and p;
24:         push s and p onto Stack;
25:       }
26:     }
27:   }
28: }
```

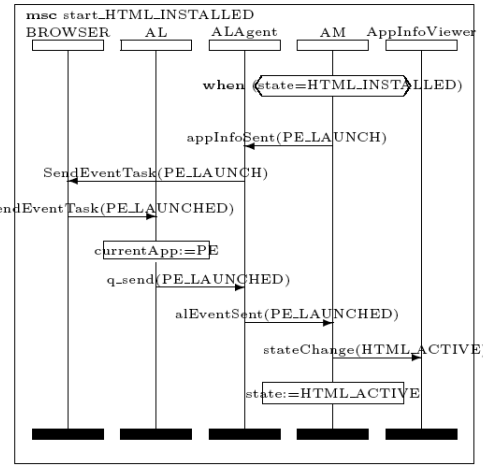
3. EXPERIMENTAL RESULTS

We implemented a toolset including MSCs editor and GFSM generator, and applied the proposed approach to partial specification of embedded software running on a digital TV (DTV) developed by Samsung Electronics (SEC). It implements the DASE (Digital television Application Software Environment) standards defined by the ATSC (Advanced Television Systems Committee). The DASE includes software modules that allow decoding and execution of application programs that deliver interactive and data broadcast services. DTV software consists of Java modules and native applications written in C. The former provides DASE application programming interface, and the latter provides the vendor specific functionality such as Web browser and e-mail, etc. It is quite complex in that it has to process 25 input events generated by the remote control unit and 42 events embedded in the transport stream.

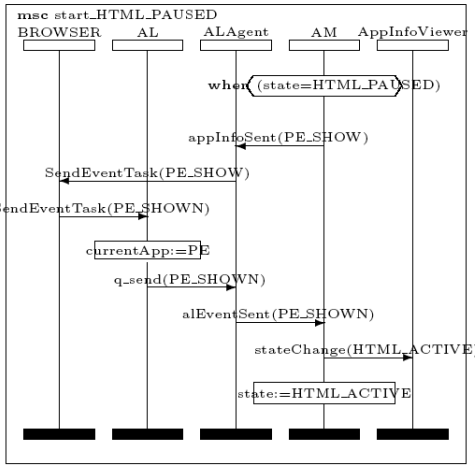
The bMSC 'TVVIDEO' (Figure 5a) illustrates a sequence of required interactions when an external event 'RC_TV_VIDEO' occurs and only one HTML application is running. Upon completing the required message exchanges, either the bMSCs 'start_HTML_INSTALLED' (Figure 5b) or 'start_HTML_PAUSED' (Figure 5c) is selectively activated. The exact conditions under which bMSC is activated appear in the referenced bMSCs. In Figure 5b, upon completing the required message exchange, the state of the HTML application is set to 'HTML_ACTIVE'.



(a) TVVIDEO



(b) HTML_INSTALLED

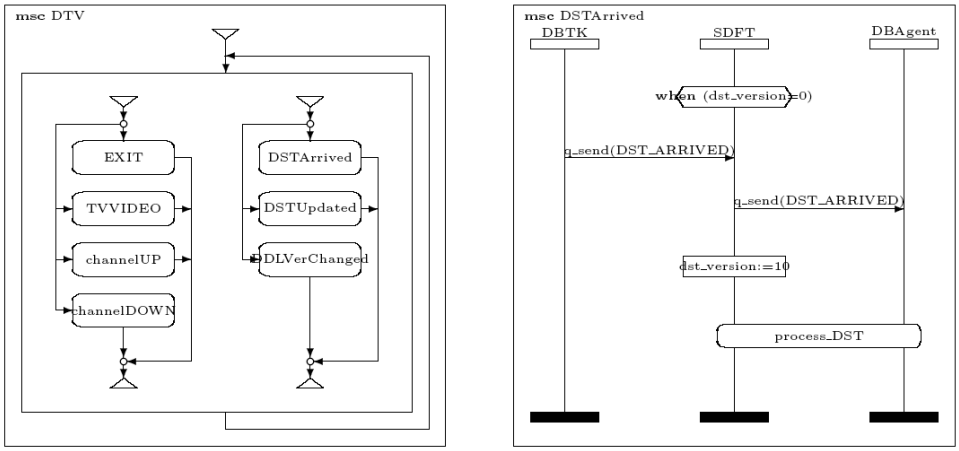


(c) HTML_PAUSED

Figure 5: Execution of HTML applications

Because many input events (e.g., channel numbers) are processed only with one or two message exchanges, we needed 34 bMSCs to specify required behaviour for seven external input events (See Table 1 for details). The relations among events, either alternative or concurrent, are described in Figure 6a. The bMSC 'DSTArrived' (Figure 6b) which is concurrently executed with the bMSC 'TVVIDEO' shows a scenario for an input event 'DST_ARRIVED'.

The first row of Table 2 describes how many states and transitions GFSM contains when generated by combining concurrent bMSCs processing one stream input (DSTArrived) and four control inputs. The experiment was run on the Sun Blade 1000 system with Ultra Sparc III 750 MHz CPU and 1Gbytes M/M. Our technique reduced the number of transitions by about 98% (from 7.892,866 to 17,376). The second row of Table 2 shows the result of combining four remote control inputs and three stream inputs. Whereas the brute-force approach was unable to generate GFSM due to transition explosion, application of the independent region successfully generated GFSM relatively quickly.



(a) DTV (b) DST parsing

Figure 6: Concurrent scenarios

Input Device	Input Event	# of bMSCs	# of Messages in bMSCs	# of Transitions in behavior automata
Remote Control Unit	EXIT	3	7	11
	TVVIDEO	3	16	23
	channelUP	5	14	23
	channelDOWN	5	14	23
Stream	DSTArrived	7	23	38
	DSTUpdated	10	33	53
	DDLVerChanged	1	2	4

Table 1: The size of each scenario

Input Event		Reduction Method	GFSM		
Stream	Remote Control Unit		State	Transition	Time
DSTArrived	EXIT, TVVIDEO, channelUP, channelDOWN	No Reduction Independent Region	24 24	7,892,866 17,376	5,220s 27s
DSTArrived, DSTUpdated, DDLVerChanged	EXIT, TVVIDEO, channelUP, channelDOWN	No Reduction Independent Region	× 60	× 236,648	× 360s

Table 2: Reduced GFSM using the Independent Region

4. CONCLUSIONS

Message Sequence Charts (MSCs) formalism is useful when describing reactive behaviour of large and complex embedded software, and MSCs can be algorithmically translated to GFSM. However, the brute-force approach is impractical due to state and transition explosion problems. State variable was used, as suggested in Lee and Cha (2003), to reduce the number of states included in GFSM. In this paper, we introduced the concept of an independent region to reduce the number of transitions. We proved that independent regions preserve the original behaviour of GFSM and that static analysis is sufficient – although not optimal – in finding independent transitions.

There are a couple of topics worthy of further research. First, independent region, as discussed in this paper, includes only minimal message interactions, and further improvements on static analysis methods to identify independent transitions are needed. Second, standard semantics of MSCs interprets alternative scenarios as sequential composition. However, such interpretation does not accurately model actual system behaviour where a scenario may become activated prior to completing the interactions of previous scenarios. MSC semantics must be extended to properly model such behaviour.

REFERENCES

BOOCH, G., RUMBAUGH, J. and JACOBSON, I. (1998): The unified model language user guide, Addison-Wesley.

BUDKOWSKI, T. and DEMBINSKI, P. (1987): An introduction to Estelle: a specification language for distributed systems, *Computer Networks* and ISDN 14(1).

GODEFROID, P. (1996): Partial order methods for the verification of concurrent systems: an approach to the state explosion problem, Lecture Notes in Computer Science, Vol. 1032, Springer-Verlag, Berlin.

GODEFROID, P., PELED, D. and STASKAUSKAS, M. (1996): Using partial-order methods in the formal validation of industrial concurrent programs, *IEEE Transactions on Software Engineering* 22(7): 496–507.

GODEFROID, P. and WOLPER, P. (1993): Using partial orders for the efficient verification of deadlock freedom and safety properties, *Formal Methods in System Design* 2(2): 149–164.

HAREL, D. (1987): Statecharts: a visual formalism for complex systems, *Science of Computer Programming* 8: 231–274.

HAREL, D. and Kugler, H. (2002): Synthesizing state-based object systems from LSC specifications, *International Journal of Foundations of Computer Science* 13(1).

HOLZMANN, G. (1997): The model checker SPIN, *IEEE Transactions on Software Engineering* 23(5): 279–295.

HONG, H.S, KIM, Y.G., CHA, S.D. and BAE, D.H. (2000): A test sequence selection method for statecharts, *Journal of Software Testing, Verification, and Reliability* 10(4): 203–227.

KATZ, S. and PELED, D. (1992): Verification of distributed programs using representative interleaving sequences, *Distributed Computing* 6: 107–120.

KURSHAN, R., LEVIN, V., MINEA, M., PELED, D. and YENIGUN, H. (1998): Static partial order reduction, in *Proc. of 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS'98), LNCS 1384, Springer-Verlag, 345–357.

LEE, N.H. and CHA, S.D. (2003): Generating test sequences from a set of MSCs, *Computer Networks* 42(3): 405–417.

LUO, G., BOCHMANN, G. and PETRENKO, A. (1994): Test selection based on communication nondeterministic finite-state machines using a generalized Wp-method, *IEEE Transactions on Software Engineering* 20: 149–162.

VALMARI, A. (1992): A stubborn attack on state explosion, *Formal Methods in System Design* 1: 297–322.

RECOMMENDATION Z.100 (1992): SDL methodology guidelines, ITU-T, Geneva.

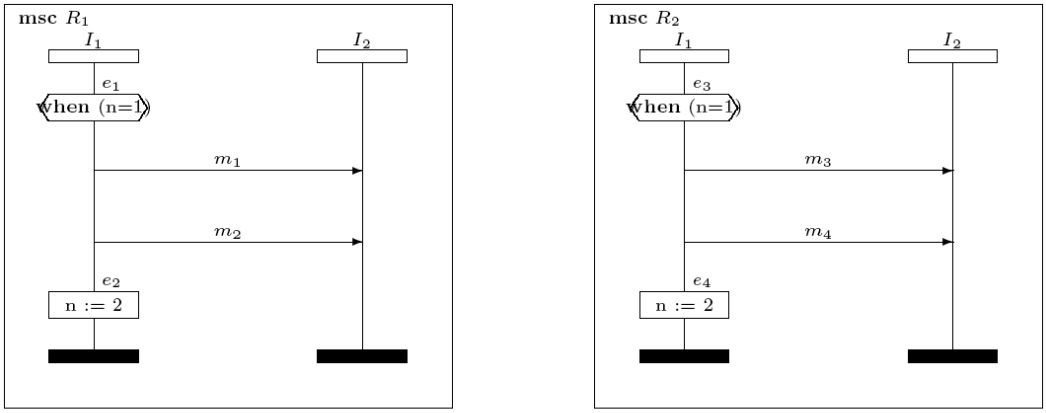
RECOMMENDATION Z.120 (2000): Message Sequence Chart (MSC'2000), ITU-T, Geneva.

APPENDIX A
A PROOF OF BEHAVIOUR PRESERVATION BY INDEPENDENT REGION

When combining two concurrent scenarios, there are five cases to consider for guarding conditions (e_1 and e_3 of Figure 7) and state changes (e_2 and e_4 of Figure 7). For all cases, the independent region preserves the system's behaviour as follows:

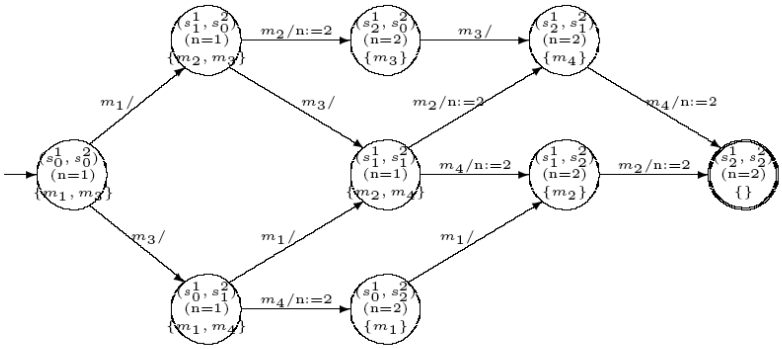
- **Case 1:** When two scenarios are activated by the same guarding condition and have the same destination state, all possible interleaved sequences and corresponding state changes are shown in Figure 8a. By the definition of GFSM given in Section 2, we can transform this FSM to GFSM as shown in Figure 8b, and the GFSM includes six message sequences and two global states. When independent region analysis is applied to GFSM, reduced GFSM including only two message sequences, but the number of state remains the same. Therefore, the independent region preserves the behaviour of two scenarios.

- **Case 2:** When two scenarios are activated by the same guarding conditions but have different destination states, Figure 9 captures all possible interleaved sequences and corresponding state



(a) R_1 (b) R_2

Figure 7: Two concurrent scenarios



(a) FSM for R_1 and R_2

(b) GFSM (c) Reduced GFSM

Figure 8: GFSM for Case 1

changes. GFSM shown in Figure 9b includes six message sequences and three global states, and reduced GFSM contain only two message sequences and three states. Consequently, the independent region preserves the behaviour of two scenarios shown in Case 2.

- **Case 3:** When two scenarios are activated by different guarding conditions but have the same destination state, this is the same as the Case 1 above.

- **Case 4:** When two scenarios are activated by different guarding conditions and have different destination states, this is the same as the Case 2 above.

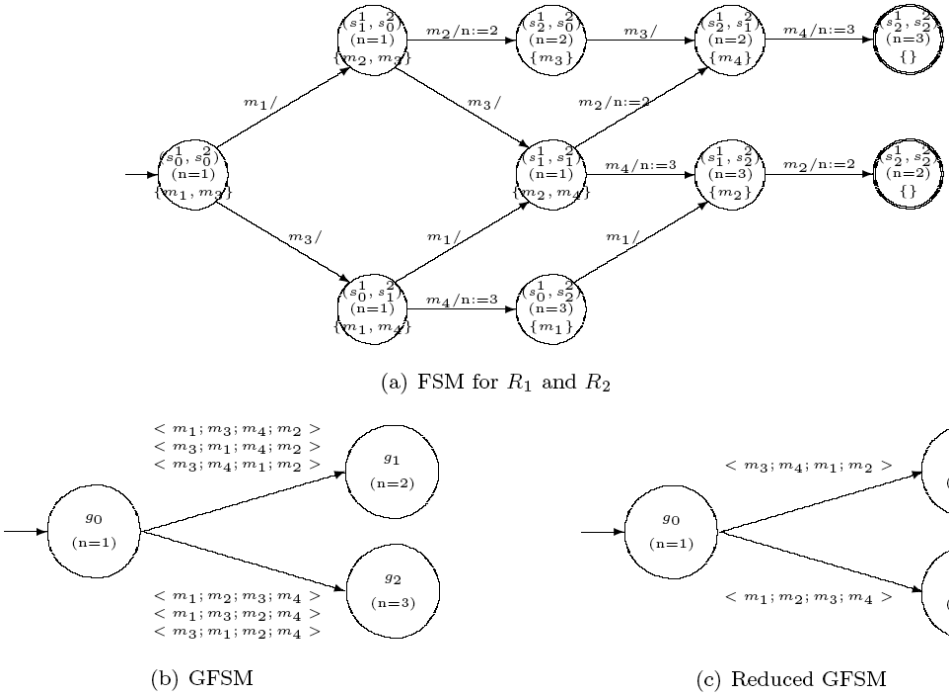


Figure 9: GFSM for Case 2

- **Case 5:** Two scenarios may be activated by different guarding conditions but have the same destination state involving two different variables. This is the same as the Case 1.

BIOGRAPHICAL NOTES

Nam Hee Lee

Nam Hee Lee received the BSc, MSc, and PhD degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 1991, 1998, and 2003, respectively. He is currently a member of the technical staff at Samsung SDS. His research interests include formal methods, software testing, and quality assurance.

Sung Deok Cha

Sung Deok Cha received the BSc, MSc, and PhD degrees in information and computer science from the University of California, Irvine, in 1983, 1986, and 1991, respectively. From 1990 to 1994, he was a member of the technical staff at Hughes Aircraft Company, Ground Systems Group, and the Aerospace Corporation, where he worked on various projects on software safety and computer security. In 1994, he became a faculty member of the Korea Advanced Institute of Science and Technology, Electrical Engineering and Computer Science Department. His research interest includes software safety, formal methods, and computer security.