A Case Study: Verification of Specifications of an Embedded System and Generation of Verification Items using Pairwise Testing

Toshifusa Sekizawa*　Tsugu Kotorii**

Ensuring the reliability of embedded systems has become very important. Reliability may be ensured by a number of formal methods. We study one such verification technique by applying it to an in-house development product in Mitsubishi Space Software Co., Ltd. This is a practical industrial case study, we describe our approaches and present verification results. Our aim is to check the correctness of specifications which include a set of constraints on parameters individually called an evaluation item. To that end, we adopt model checking and satisfiability checking. In our study, we set conversion rules from specifications to formal models. Part of the conversion is done by hand in this study. Manual generation limits the preparation of individual evaluation items. To overcome this limitation we present an approach for automatically generating combinations of parameters for verification by applying the pairwise testing method. Finally, we present experimental results. Note that the application of formal techniques, in this setting, is still in its preliminary stages. It is intended to develop formal techniques to the point where products may be automatically verified.

Keywords-embedded system, verification, model checking, generation of verification items, case study

Ⅰ. INTRODUCTION

Embedded systems have become important in our society.Software has come to control almost everything from safety critical systems to home appliances. Consequently, verifying the properties of embedded systems has become increasingly important. Formal methods are mathematical based techniques for verification and development. Formal methods are widely used in order to ensure properties. Several successful applications have been reported, these include verification of protocol specifications[1], internal groupware systems[2], and avionics runway safety monitoring[3]. The appearance of such formal and semi-formal methods has led to the introduction of international standards such as ISO/IEC 61508 and RTCA（Radio Technical Commission for Aeronautics）DO-178C. Therefore, it is crucial to accommodate these standards by introducing formal methods to the product development process.

Model checking[4] is one formal method which has been successfully applied in the verification of many systems. A model of the system is verified by exhaustively searching its state space to ensure that properties are satisfied. In this approach models may be represented by a transition system, such as a Kripke structure, and properties are described by using a logic such as Linear Temporal Logic（LTL）or Computation Tree Logic（CTL）[5].

This study reports on the application of formal methods in the development of an embedded control system by Mitsubishi Space Software Co., Ltd.（MSS）. Note that the application of model checking in the product development process is still in its early stages. This study, therefore, includes investigation into the applicability of formal methods, model checking in

*Faculty of Informatics,Osaka Gakuin University, Japan　**Quality Assurance Department, Kansai Division, Mitsubishi Space Software Co., Ltd., Japan

particular, to the development process. To do this we chose an in-house developed embedded system which had its main functions defined by two specification methods, a functional specification and a combination table of signals. The following three approaches were applied.

1）Reachability checking using model checking,
2）Satisfiability checking using an SMT（Satisfiability Modulo Theories）solver, and
3）Generation of combinations of variables for verification using pairwise testing methods.

In approach 1, a state transition system is constructed from functional specifications and formulas for reachability are extracted from the combination table. These are then verified using model checking. Approach 2 is a kind of bounded model checking where satisfiability checking is only applied to properties associated with no transitions. Approach 3 differs from the above in that it verifies properties generated from a manually produced combination table. Because of the limited number of combinations that can be produced manually we adopt pairwise testing to generate combination of variables.

Through experiments, we have found some shortcomings in the descriptions of specifications which have passed the review process. The contributions of this study are; i）to show the translation of product specifications into formal models, ii）to show the verification results of an industry case study, and iii）to describe the generation of sets of verification parameters by applying pairwise testing. Note that the names of systems and variables have been generalized so as to protect trade secrets.

The layout of this paper is as follows. In Section Ⅱ, we describe the system and specifications. Then we present our approach for applying model checking to investigate correctness of the target system and specifications in SectionⅢ. SectionⅣ presents verification results, and SectionⅤ offers some discussion of these results. SectionⅥ provides a concluding summary and outlines our future work.

TABLE Ⅰ　TRANSITION CONDITIONS OF THE SYSTEM

state	No.	condition	transition
S_1	#S1-1	AND {(σ_1=ON), (σ_5=OFF)}	S_2
S_2	#S2-1	OR {(σ_1=OFF), (σ_5=ON)}	S_5
:	:	:	:
S_7	#S7-1	OR {(σ_1=OFF), (σ_2=OFF), (σ_5=ON) }	S_5
S_7	#S7-3	AND {(σ_9=ON), (σ_2=ON)}	S_3

Ⅱ. SYSTEM AND SPECIFICATIONS

The Behavior of the target systems in this study are defined by the following specifications.

1）Functional specifications of a control system
2）Combination table of signals of state flow

The Functional specifications describe the core parts of inhouse embedded products at MSS' s development site. Various projects use this specification either as a foundation, or to add feature expansions. Ensuring the properties of this core specification is crucial. Note that, this specification has been reviewed, but additional reliability checking has been requested.

Functional specifications define signals（variables）, processes and transition conditions for every system state. There are fifteen boolean signals $\sigma_1, \sigma_2, \ldots, \sigma_{15}$ which include both normal signals and error signals. Each signal represents a sensor state in the target system. The target system also has seven system states, $S_1, \ldots, S_7$, depending on values of the signals. Each system state represents a mode, such as initial condition diagnostics. As might be expected, transition conditions are also defined in the functional specification. Table I shows a part of these transition conditions. There are 16 transition conditions for system states. The table lists the current state, the condition number, the transition conditions, and transition destination for each entry. For example, the first row states that if the current state is S_1 and if transition condition #S1-1 holds, *i.e.*, (σ_1=ON) ∧ (σ_5=OFF) holds, then the system makes a transition to state S_2.

The functional specification also describes the whole system which consists of the following four processes, $P_1, \ldots, P_4$.

1）P_1 represents the main controller,
2）P_2 simulates human manipulations,

3) P_3 is an external environment, and
4) P_4 is an external environment different from P_3.

These four processes run in parallel and all except P_2 are continuous. Process P_1 is the main controller which handles the system state according to the behaviors defined by transition conditions in Table I. This main control process is the verification target. Process P_2 simulates human manipulations and handles only one global signal σ_1 which represents the main system switch. Note that the main switch does not interrupt the power supply and all processes will continue to run if the main switch is turned off. Process P_3 represents an external environment. It detects and aggregates changes in sensor values then makes them available to P_1. Process P_4 is also an external environment and is similar to P_3.

Here, we describe data flow between processes. Process P_2 controls a global switch σ_1 which affects the condition of the system. For example, signal σ_2 never becomes ON if σ_1 is OFF, but σ_2 becomes ON if σ_1 = ON and some other conditions hold. However, hardware sensors may be activated by conditions outside of the specification, even if σ_1 is OFF. This is because we are handling the core part of the embedded system, and the sensors are affected by external conditions. Processes P_3 and P_4 detect changes in the sensors, and asynchronously send the sensor values to P_1. The main controller P_1 periodically polls and collects these sensor values. According to the combination of the sensor values, P_1 judges whether or not the conditions for state transitions hold. If a transition condition is satisfied for the current state then P_1 makes a transition to another system state.

Before describing specification 2, let an *evaluation item* (EI) be a combination of the values of fifteen signals $\sigma_1, \ldots, \sigma_{15}$, and let an *evaluation table* (ET) be a list of EIs. Specification 2 provides an ET which contains 35 EIs. These EIs are extracted from review results by hand. These may be described as necessary combinations of signals for ensuring the reliability of P_1. Note that the ET is used to generate test cases for the testing phase of this product.

Table II shows part of an ET which contains 35 EIs. The three header rows show the process number, signal names, and a signal classification, *i.e.*, normal signal (N) or error signal (E). Each of the data rows corresponds to an EI which lists the item number, constraints on signals, the current state, and the expected transition. Signal constraint cells may either have a value or be blank. If the cell has a constraint value then the signal must take that value. If the cell is blank then the signal may, non deterministically, take any value in its range. The "state" column identifies the current system state as defined in specification 1. Lastly, the transition column lists the expected property which will either be a transition from state S_i to state S_j, listed as "$S_i \rightarrow S_j$", or no transition, listed as "Status Quo", which means that the controller stays in its current state.For example, entry No.1 indicates that for P_2 normal signal σ_1=ON

TABLE Ⅱ EVALUATION TABLE

	P_2	P_3			P_4												
No.	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_{14}	σ_{15}	state	transition
Type	N	N	N	E	E	N	N	N	N	N	N	N	E	E	N		
1	ON				OFF											S_1	$S_1 \rightarrow S_2$
2	OFF				ON											S_1	Status Quo
:							:										:
21	ON	ON			OFF									ON		S_4	$S_4 \rightarrow S_6$
22	ON	ON			OFF								ON			S_4	$S_4 \rightarrow S_7$
:							:										:
31	ON	ON			OFF									ON		S_6	Status Quo
:							:										:
35	ON	ON			OFF				ON				OFF	OFF		S_7	$S_7 \rightarrow S_3$

and for P_4 error signal σ_5=OFF. All other signals have non-deterministic settings, either ON or OFF. Then if the current system state is S_1, the target process P_1 is expected to make a transition into state S_2.

Ⅲ. OUR APPROACH

Our approach consists of two parts; verifying properties defined by an ET, and generating EIs using a pairwise testing method, Fig. 1 illustrates this approach and it is described in detail below.

A. Verification of Properties

Verification is carried out by using the SPIN[6] model checker and satisfiability checking uses the Yices[7] SMT solver. These procedures are detailed here.

1) *Model Checking:* To verify reachability properties, which are expressed as "$S_i \rightarrow S_j$" in an EI, we use the SPIN model checker. SPIN takes a model written in a specification language, Promela, and properties described in LTL. There are a number of EIs in the ET but because some of them conflict with others we verify the EIs one by one.

The Promela input model for SPIN consists of variable declarations, four processes, and some inline functions. The model is constructed from the specification in a straightforward manner. In this study, we draw up conversion rules from specifications to Promela for future automated verification. Our conversion rules mainly consists of two parts, a conversion rule for generating P_1 from transition conditions, and conversion rules for generating other processes from an EI.

First, we describe the conversion rule from specification 1. As shown in Table I, transition conditions for the system are expressed in predicate logic where signals are atomic propositions. Therefore, it is easy to convert from the formula to a Promela model. For example, condition No.#S1-1 is converted into Promela model as shown in Fig. 2. Note that, P_1 is the verification target and is shared in all verifications. Therefore, this conversion is used only once.

Next, we describe the conversion rules from an EI to Promela. Conversions are done automatically according to conversion rules. Process P_2 only handles one global variable. This process can be modeled in a straightforward manner as shown in Fig. 3.

Processes P_3 and P_4 have essentially the same behaviors. The reason for division into two processes is based on the design of the target system, and is outside the scope of this study. Both processes are non-stop processes which continuously send signals to P_1. Therefore, the whole process is expressed as an infinite loop. One execution of the do loop corresponds to the sending of a set of signals. Values of signals depend on the values of the elements in the EI. If a value of signal is defined with a constraint, then the signal is expressed deterministically. Otherwise, if no constraints are defined then the value of signal is handled by non-deterministic choice. The Promela code shown in Fig. 4 expresses the conversion rules. Note that if there are no deterministic signals then conversion for the signals are skipped.

In addition, we need at least one property for verification. A property is extracted from the cells in the EI which describe the target state and expected property. If an expected property is written as "$S_i \rightarrow S_j$" in a cell of the ET, it means "from the state S_i, eventually reaches to S_j". This reachability property is translated into an LTL formula $\Box$ (state==$S_i \rightarrow \Diamond$ (state==S_j)) , where state is a variable in Promela representing a system state.

2) *Satisfiability Checking:* If the transition cell for an EI

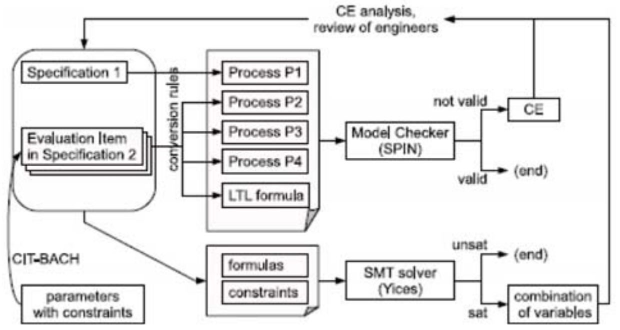

Fig. 1 Our Approaches

```
 if
   state==S1 ->
     if
     ::(s1==true&&s5==false)->atomic{state=S2}
     ::else->skip
     fi;
```

Fig. 2 Example of Conversion of Transition Conditions

```
proctype P2() {
  (s1 == false) -> s1 true;
}
```

Fig. 3 Conversion Rule for Process P_2

```
proctype processname () {
  do
  :: true ->
    (constraint)*
    (if
    (:: non-deterministic choice)*
    fi;)?
  od
}
```

Fig. 4　Conversion Rules for Evaluation Item

has "Status Quo" as its expected value then there is no state transition. In this section, we describe our approach to EI including "Status Quo" . For this expected property, it is possible to write a corresponding LTL formula, but verification using SPIN is hard because of the state explosion problem caused by the number of signals. As described above, transition conditions in $\boldsymbol{P}_1$ can be expressed as a formula in predicate logic. Constraints on signals in an EI can also be expressed as a formula. Therefore, it is possible to check whether or not the transition condition is met by satisfiability checking.

Specifically, we first obtain the transition condition φ associated with systems state $\boldsymbol{S}_i$ and the constraints χ defined by the appropriate EI. Then we check the satisfiability of φ with constraints χ. If φ is unsatisfiable, there will be no combination of signals which makes a transition from state $\boldsymbol{S}_i$. Otherwise, if the result is satisfiable, there will be at least one combination of signals, and the expected property does not hold. In this case, Yices shows the combination of signals which satisfies the checked formula and this provides a counter example.

This approach is a kind of bounded model checking[8] in which the boundary is limited to the current state. Note that an SAT solver is sufficient for checking the properties handled in this study, because all signals are two-valued. However, enhanced versions of specifications will include real values and functions on a range of signals. Therefore, we adopt an SMT solver for future extensions.

B. Generation of Evaluation Items

Manual generation of EIs limits the number of properties that can be checked. Therefore, it is necessary to have automatic generation of EIs. We adopt pairwise testing which is a combinatorial method of generating test cases from a pair of parameters. EIs are thus generated using pairwise testing a properties are verified by model checking the generated EIs. Because pairwise testing only generates pairs of parameters it is then necessary to associate a property with each EI. We use CIT-BACH (combinatorial interaction testing tool with a BDD-assisted constraint handler) [9] as a pairwise test case generator. One feature of CIT-BACH is that it allows constraints to be applied to parameters. If one gives constraints on input parameters then generated pairs of parameters will satisfy these constraints.

Now we consider the construction of a model using the results from CIT-BACH. We can construct an input model for SPIN based on specification 1 and pairs of signals considered as constraints in an EI. However, CIT-BACH only generates pairs of signals and there is no property. To solve this problem, we focus on the two classifications of system states, normal states and error states. That is, system states $\boldsymbol{S}_5$, $\boldsymbol{S}_6$ and $\boldsymbol{S}_7$ are assigned to error states. According to the specification, if at least one error signal is ON the system must transit to one of its error states. This may be an indirect transition, for example, $\boldsymbol{S}_1$ cannot directly reach one of the error states and must indirect via some other state. This specification can be applied to set a property. Specifically, We set at least one error signal to ON in the pairwise testing input parameters, and set the property for reachability checking to state $\boldsymbol{S}_5$, $\boldsymbol{S}_6$, or $\boldsymbol{S}_7$.

In this study, the approach described above is carried out by hand. However, this approach can be automated. It allows us to carry out verifications while computation environments have idle time. We believe that such additional verification will contribute to improve reliability.

Ⅳ. VERIFICATION RESULTS

In this section, we present results of verifications and satisfiability checking. We implemented a prototype tool for generating input files from ET based on the conversion rules described in Sec. III. Note that the prototype tool only generates input files for SPIN leaving the input files for Yices to be constructed by hand. However, input files for Yices are not complicated, and there is no difficulty.

All 35 EIs listed in specification 2 are verified, 25 EIs are verified using SPIN and 10 EIs are checked using Yices. Results indicated that three EIs, No.21, No.22, and No.31, are not sufficient to satisfy transition conditions. Two of these, No.21 and No.22, were identified by model checking, and the third by satisfiability checking. The verification results were unexpected, because specifications are reviewed and all properties are expected to hold.

In response to the verification results, we investigated the reasons for the failures. First, we checked the conversion rules. Particular attention was given to the examination of P_1 and its associated conversion rules. This is because P_1 is generated only once and shared by all verifications, *i.e.*, if P_1 has error, verification results make no sense. As a result of examination, we conclude that the P_1 accurately reflected its specification. Hereafter, assume that P_1 is correct.

Next we investigated transition conditions for EIs No.21 and 22. For No.21, the expected property is reachability, $\Box((\text{state}=S_4) \rightarrow \Diamond(\text{state}=S_6))$. On analyzing the counter example it was found that a necessary constraint was missing. The authors handle verification from the viewpoint of quality assurance without the perspective of engineers working on system development. Engineers in charge of the product were asked to review these verification results. It was confirmed that there were missing constraints. In more concrete terms, signal σ_{13} is defined as a non-deterministic variable in EI No.21, but σ_{13} must be fixed to false. After the problem was solved, we set the signal to be false, and verified the property again. This verification confirmed that the property satisfactory holds. The problem with No.22 was similar to that of No.21, *i.e.*, a missing constraint. This was also confirmed after appropriate constraints were set. Note that the prototype conversion tool is used again for these re-verifications. In addition, we examine the effects of these problems because EIs are used as a foundation for constructing test cases for this product. These missing constraints are handled as implicit knowledge in the test phase, and errors are avoided. Obviously, such occurrences decrease reliability in software development.

The problem found in EI No.31 was then investigated. The expected result was unsatisfiable, but the solver indicated that it was satisfiable. Yices returns a combination of variables for satisfiable transition conditions and this was analyzed to identify the problem. This led to the identification of inconsistencies in variables and the conclusion that conditions for EI No.31 were not sufficient. As with the cases of EIs of No.21 and No.22, the results were checked by engineers. It transpired that, EI No.31 was intentionally inserted into the ET by engineers in order to investigate power of formal methods. According to the engineers, EI No.31 simulates a situation in which a variable is misread and a constraint is incorrectly described. EI No.31 does not exist in the real specification and the authors had not be notified of the addition of this test item. Even though EI No.31 is an experimental item, we believe that detection of its inconsistency shows the applicability of satisfiability checking.

Finally, we describe our trial verification results for generating EIs using CIT-BACH. To confirm validity, we generated a set of EIs using CIT-BACH according to the approach described in Section III. In applying the approach, it is required to determine which error signal is set. Here, we set error signal σ_5 to be true as a constraint. Normal signal σ_1 is also set to be true, because if this signal is false then many other signals are affected. We constructed an input file for CIT-BACH which described these constraints. CIT-BACH then generated nine EIs for two constraints and thirteen signals not specified. In addition, it is required to determine the target system state and at least one property for verification. Here, the target state is set to S_3, and a property is expressed as reachability to one of the system error states. The reachability property is $\Box(\mathit{targetstate} \rightarrow \Diamond \mathit{errorstates})$, where $\mathit{targetstate}$ is $\text{state}=S_3$ and $\mathit{errorstates}$ is $\text{state}=S_5 \vee \text{state}=S_6 \vee \text{state}=S_7$. The reason that *errorstates* consists of three system states is because an error state cannot be absolutely specified for the constraints. The generated EIs were then converted into Promela models using our prototype conversion tool, and verified using SPIN. The results were as expected and no inconsistencies were found. Note that, all signal values were fixed in the setting described above. It is, of course, possible to set some of of the signals to be non-deterministic values.

Verifications were performed using SPIN 6.2.2, Yices 1.0.38, and CIT-BACH 1.01 on Windows 7 64bit, running on an Intel Core i5-2400 3.10GHz, with 8GB memory. According to the verification log of SPIN, the number of system states stored is approximately 3×10^7 and the time for a verification when properties hold is about 450 seconds. It is easy to understand how Yices returns the result in about one second, because unused variables are omitted.

Ⅴ. DISCUSSION

Here, we discuss our experiments and consider related work in the field of control engineering and embedded systems.

A. Discussions on the Experiments

In this section, we discuss our verification experiments and consider effectiveness of our approaches to development processes.

First, we consider the missing but necessary signal values in EIs. As mentioned, the evaluation table had been reviewed entirely save for the intentionally inserted experimental EI. Even if these omissions are avoided by implicit knowledge, their detection during the design phase is valuable for product development because EIs are the basis for test cases. The detection of an intentionally inserted EI also shows possibility of formal methods.

We have not found any inconsistencies in the verification of EIs generated using CIT-BACH. Even so, these generated EIs specify new combinations of signals allowing us to verify additional items that were not extracted from the review process. These results indicate both the applicability and the effectiveness of our approach. Recall that one of the purposes of this study was an investigation of the applicability of formal methods. It is widely said that formal methods have the power to reduce problems in the earlier stage of development. Our results confirm that formal methods are useful and that this saying expresses a valid point of view.

Though the verification results are positive we believe that there are still some problems to overcome and one of these is automation. Automation is an important consideration for applying formal methods to product development. In this study, conversion rules are applied to generate formal models. We believe that this approach enables automation. Counter examples are, however, analyzed by hand and this is not an ideal situation for the application of formal methods. This is because counter examples do not necessarily present the shortest path and the manual analysis of counter examples is requires skill and is costly. Therefore, computer aided counter example analysis will be required. Another problem is the state explosion problem. This is widely considered as a major problem in model checking. We constructed models from specifications in a straightforward manner. This approach improves the readability of models, but causes state explosion problems. The size of state space for this study was within SPIN' s capabilities but it was close to the state space size limit. If we apply our approach to the enhanced versions of the product specification then the state space explosion will potentially cause problems. Abstraction seems to be a promising technique for reducing the size of the state space. Data mapping and predicate abstraction seem to be especially, efficient in this regard.

Now we consider function enhancement. Specifications were converted into Promela models for conventional model checking. This approach was sufficient for the specifications handled in this study, because all signals are two-valued. However, these specifications are a core part of our product and in the enhanced version signals are not limited to twovalue ranges. Additionally, various types of properties are expected to be verified. For example, one important behavior aspect of embedded system is time-related properties. Several models have been proposed to deal with such real-time systems. One such model is the timed automaton[10] which is being considered as a candidate if we extend our approach to verify time-related properties. Another possibility is the verification of hybrid systems[11] in which continuous and discrete dynamics are mixed with time progression. Embedded systems sometimes control continuous systems. Therefore, hybrid systems also seem to promise models which reflect this target architecture more closely.

B. Related Work

Here we briefly describe related work on conversion, verification of applications in control engineering. When one considers applying formal techniques to ensure the quality of software products, it is necessary to consider the construction of models and the description of properties. There have been a number of studies on the conversion of specification into formal models. For example, the verification of realtime control programs[12] describes the construction of a timed automaton model using automatic translation from the control program to properties. The automatic conversion from a specification language, such as SysML and UML, into formal models used in model checking is described in[13]. A survey of model checking of robotics control systems[14] is of interest because the authors summarize various techniques for verification and also describe verification results for safety and liveness properties. The process of technology transfer from the academic domain into industrial practice is described well in[15].

Ⅵ. CONCLUSION

We have studied verification techniques to ensure the reliability of specifications for an embedded system which is an in-house development product of Mitsubishi Space Software Co., Ltd. One of the purposes of this study was investigation of applicability of formal methods. To this end, we took two approaches, the conversion from specification into formal models for model checking or satisfiability checking, and the generation of combinations of parameters using the pairwise testing method. We first set conversion rules which translated specifications into formal models. Then we constructed models of the target system by applying the conversion rules and verified them using a model checker, SPIN, or an SMT solver, Yices. Results led to us finding deficiencies in the descriptions of parameters in specifications. We have also described an approach to the generation of evaluation items using pairwise testing. This approach enables additional verifications and contributes to improving the reliability of specifications. These results strongly convince us of value of formal methods.

We believe that certain of these results are a positive indication of the applicability of formal methods but this study has highlighted some problems. One of our future work areas includes the implementation of our approach. So far we have used a prototype conversion tool to construct input files for verification and satisfiability checking. The tool only converts a part of the input files, leaving some details to be processed manually. All conversions were, however, done by rote. This indicates that conversion from the evaluation table can be fully automated. These results lead us to believe that pragmatic expansion to the development site and into products will produce the desired effects. Another future task is introducing formal specification language such as B, Z, and VDM. In this study, we had to convert the main controller by hand because the specification was written in natural language. Obviously, this specification method leads to difficulties in automatic verification using formal methods.

ACKNOWLEDGMENT

The authors would express our sincere gratitude to Professor Tatsuhiro Tsuchiya, Osaka University, for his advise about pairwise testing and tool CIT-BACH.

The authors would like to thank Associate Professor R. D. Logie, Osaka Gakuin University, for his careful proofreading.

REFERENCES

[1] A. Schneider, T. Bluhm, T. Renner, U. Heinkel, J. Knablein, and R. Zavala, “Formal verification of abstract system and protocol specifications,” 2012 35th Annual IEEE Software Engineering Workshop, vol. 0, pp. 207–211, 2006.

[2] M. H. ter Beek, M. Massink, D. Latella, S. Gnesi, A. Forghieri, and M. Sebastianis, “A case study on the automated verification of groupware protocols,” in 27th International Conference on Software Engineering（ICSE 2005）, 15-21 May 2005, St. Louis, Missouri, USA, G.-C. Roman, W. G. Griswold, and B. Nuseibeh, Eds. ACM, 2005, pp. 596–603.

[3] R. I. Siminiceanu and G. Ciardo, Symbolic Model Checking for Avionics. John Wiley & Sons, Inc., 2012, pp. 85–112.

[4] E. M. Clarke, O. Grumberg, and D. Peled, Model Checking. MIT Press, 1999.

[5] E. A. Emerson, “Temporal and modal logic,” in Handbook of Theoretical Computer Science. Elsevier, 1990, vol. B, ch. 16, pp. 995–1072.

[6] G. J. Holzmann, The Spin Model Checker. Addison-Wesley.
[7] "The Yices SMT Solver," http://yices.csl.sri.com/.
[8] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," pp. 193–207.
[9] "CIT-BACH Website," http://www-ise4.ist.osaka-u.ac.jp/?t-tutiya/CIT/.
[10] R. Alur and D. L. Dill, "A theory of timed automata," Theoretical Computer Science, vol. 126, no. 2, pp. 183–235, 1994.
[11] J. Lunze and F. Lamnabhi-lagarrigue, Handbook of Hybrid Systems Control - Theory, Tools, Applications. Cambridge University Press, 2009.
[12] T. K. Iversen, K. J. Kristoffersen, K. G. Larsen, M. Laursen, R. G. Madsen, S. K. Mortensen, P. Pettersson, and C. B. Thomasen, "Modelchecking real-time control programs: verifying lego (R) mindstormsTM systems using UPPAAL," in 12th Euromicro Conference on Real-Time Systems. IEEE Computer Society, 2000, pp. 147–155.
[13] L. Alawneh, M. Debbabi, Y. Jarraya, A. Soeanu, and F. Hassaïne, "A unified approach for verification and validation of systems and software engineering models," in 13th Engineering of Computer Based Systems. IEEE Computer Society, 2006, pp. 409–418.
[14] N. Sharygina, J. C. Browne, F. Xie, R. P. Kurshan, and V. Levin, "Lessons learned from model checking a NASA robot controller," Formal Methods in System Design, vol. 25, no. 2-3, pp. 241–270, 2004.
[15] R. P. Kurshan, "Verification technology transfer," in 25 Years of Model Checking, ser. Lecture Notes in Computer Science, O. Grumberg and H. Veith, Eds., vol. 5000. Springer, 2008, pp. 46–64.