

Validating Software Reliability Early through Statistical Model Checking

Youngjoo Kim, S-Core

Okjoo Choi, Moonzoo Kim, and Jongmoon Baik, Korea Advanced Institute of Science and Technology

Tai-Hyo Kim, FormalWorks

// *A proposed framework employs statistical model checking to validate software reliability at an early stage. This can prevent the propagation of reliability allocation errors and design errors at later stages, thereby achieving safer, cheaper, and faster development of safety-critical systems.* //



DURING THE PAST few decades, the proportion of software in safety-critical systems has significantly increased. So, to ensure high-level safety, it's essential to improve software reliability. Consequently, it has become important to implement and acquire highly reliable software and to satisfy the safety requirements imposed by

functional-safety standards, such as IEC 61508 and ISO 26262.¹⁻³ These standards define *safety integrity level* (SIL) and automobile SIL (ASIL) as measures of a system's quality or dependability.

To develop a highly reliable software-intensive system, developers allocate a reliability goal for a target system according to a target SIL or ASIL

after hazard analysis and risk assessment.⁴ Then, they allocate reliability goals to each software component early in the life cycle. Each component's reliability goal is usually validated through failure detection during software testing, which can result in high costs to correct defects.

We propose a framework to validate the reliability goals of safety-critical systems at an early stage by using *statistical model checking* (SMC) to obtain safety certification. SMC validates a target system's reliability by computing the probabilities that an executable model of a target system satisfies given functional-safety requirements. (For more information, see the "Statistical Model Checking" sidebar.)

The Framework

Our framework (see Figure 1) extends IEEE Standard 1633, which covers software reliability practices. (For more information, see the "Software Reliability Engineering" sidebar.) It employs the following process.

Specify the Functional-Safety Requirement

This step uses hazard analysis methods such as FTA (fault tree analysis), FMEA (failure mode and effects analysis), and Fracas (failure reporting, analysis, and corrective action system) to identify safety-related functions for each component C_i .⁴ It then converts functional-safety requirements for those functions into bounded linear temporal logic (BLTL) requirements req_{ij} of C_i .

Allocate the Reliability Requirement

On the basis of the results of the "Specify the reliability requirement" step of IEEE Standard 1633, this step allocates a reliability goal R_i to C_i .

Validate the Reliability Requirement

This is the step we added that extends IEEE Standard 1633. Here, SMC

STATISTICAL MODEL CHECKING

Statistical model checking (SMC) uses randomly sampled simulation traces to compute the probabilities that a target model will satisfy given requirement properties.¹ Figure A gives an overview of SMC, which consists of a simulator, a bounded linear temporal logic (BLTL) model checker, and a statistical analyzer. It receives

- a stochastic target model M , which is an executable simulation model;
- a BLTL formula ϕ , which formally represents a functional-safety requirement of the target system; and
- precision parameters with which to determine a calculated probability's accuracy.

The simulator executes M and generates a sample execution trace σ_i . The model checker determines whether σ_i satisfies ϕ and sends the result (success or failure) to the statistical analyzer. The statistical analyzer calculates the probability p that M satisfies ϕ by checking whether σ_i satisfies ϕ . The statistical analyzer then asks the simulator to generate σ_i repeatedly until the number of successful results of σ_i over the total

number of σ_i is distributed within a given precision boundary.

Unlike conventional formal verification techniques such as model checking, SMC doesn't analyze a target system's internal logic. So, it can validate complex safety-critical systems without the state explosion problems caused by those systems' complex hybrid (continuous dynamics plus discrete computation) characteristics.

Reference

1. P. Zuliani, A. Platzer, and E.M. Clarke, "Bayesian Statistical Model Checking with Application to Stateflow/Simulink Verification," *Proc. 13th ACM Int'l Conf. Hybrid Systems: Computation and Control (HSCC 10)*, ACM, 2010, pp. 243–252.

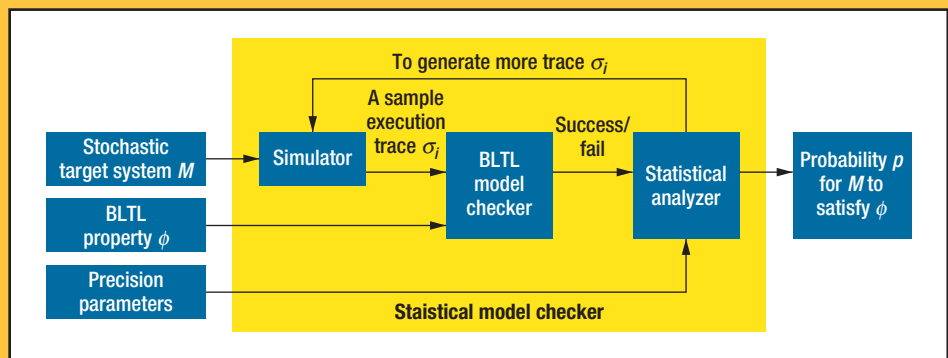


FIGURE A. Statistical model checking uses randomly sampled simulation traces to compute the probabilities that a target model satisfies given requirement properties.

generates random sample execution traces σ_i repeatedly until the number of the traces is enough to calculate the probability that C_i satisfies req_{ij} (that is, $P(req_{ij})$). If not, SMC simulates C_i again to generate more sample traces.

Validate the Reliability Goal

This step validates R_i by comparing it with the calculated reliability R'_i , obtained on the basis of $P(req_{ij})$ and the corresponding weight values for req_{ij} .

Continue Validation or Reallocate

If R'_i satisfies R_i (that is, $R'_i \geq R_i$), validation continues for the next component C_{i+1} regarding R_{i+1} . If the calculated reliabilities of all the components satisfy the allocated reliability goals, software reliability assessment continues.

If R'_i doesn't satisfy R_i , this step reallocates all the components' reliability goals. If the reallocation continues to fail,

this could indicate that the target component was designed incorrectly. If this is the case, after several trials of the reliability reallocation, the component should be redesigned to improve its reliability.

Employing the Framework: A Case Study

The top part of Figure 2 diagrams a fault-tolerant fuel control system (FFCS),⁵ a safety-critical component of an automobile's engine controller. The FFCS receives input from sensors for throttle angle, speed, exhaust gas oxygen (EGO), and manifold absolute pressure (MAP). It then generates a proper fuel injection rate and air-to-fuel ratio. It also detects sensor faults and shuts down the engine for safety if necessary. It has three components: a sensor failure detector and estimator (SFDE), an airflow calculator, and a fuel calculator.

The SFDE consists of a sensor failure detector and a sensor data estimator. The detector receives all the sensor

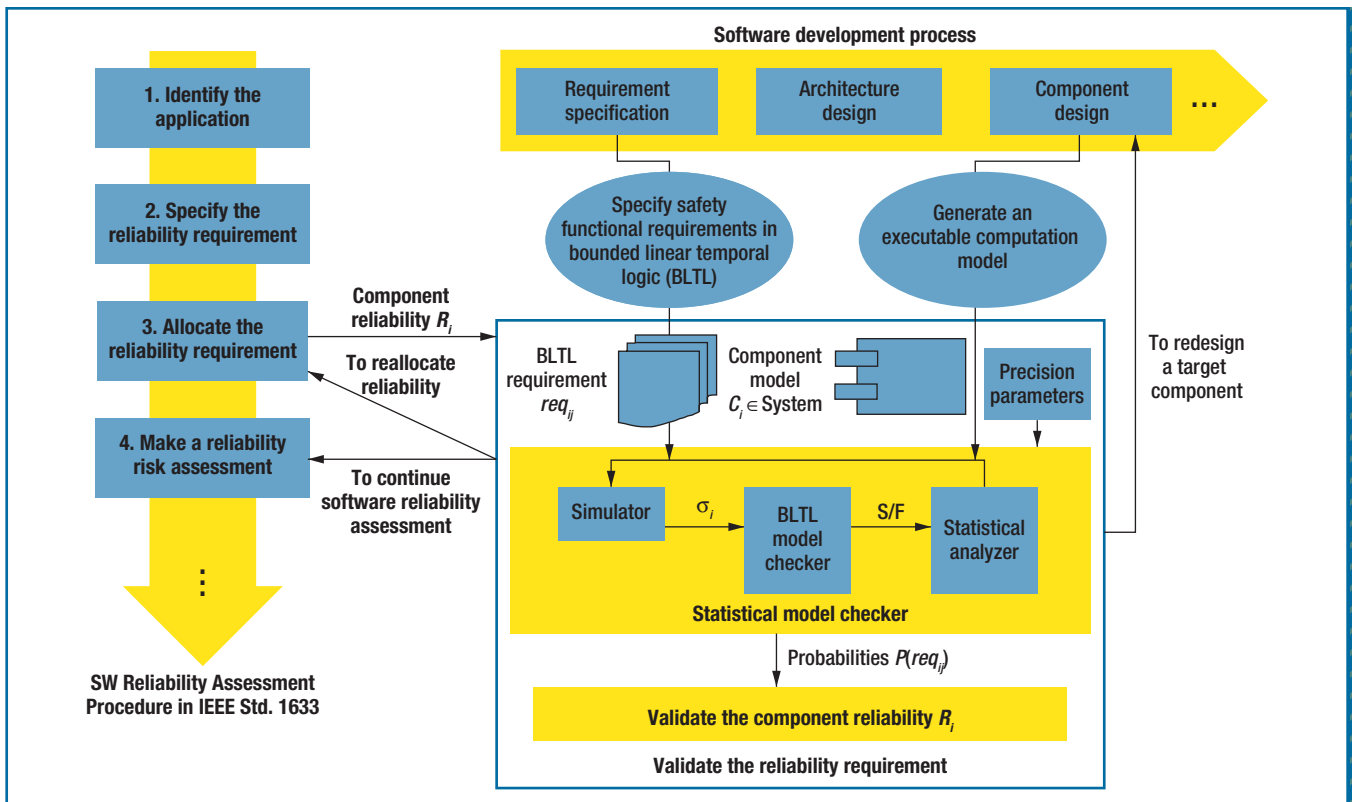
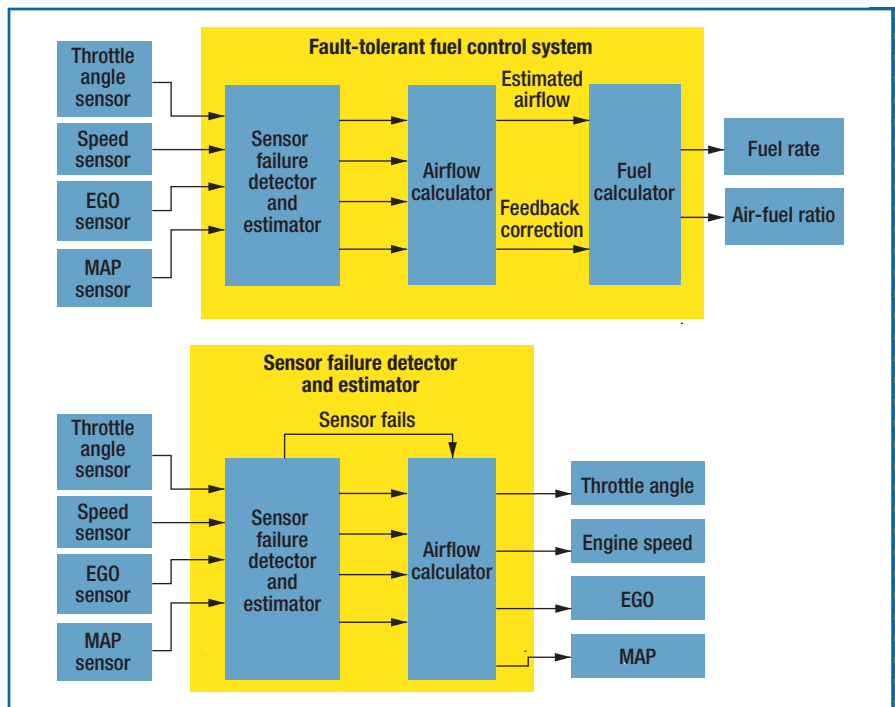


FIGURE 1. Our software reliability validation framework extends IEEE Standard 1633 by adding the step “Validate the reliability requirement” after the “Allocate the reliability requirement” step during software reliability assessment.

FIGURE 2. A fault-tolerant fuel control system (FFCS). Using input from sensors for throttle angle, speed, exhaust gas oxygen (EGO), and manifold absolute pressure (MAP), the FFCS generates a proper fuel injection rate and air-fuel ratio. It also detects sensor faults and shuts down an engine for safety if multiple sensor failures occur.



data and decides whether a sensor has failed. It delivers all the data to the estimator; if a sensor fails, it notifies the estimator of the failure. If multiple sensors fail, the detector shuts down the engine because the air-fuel ratio is uncontrollable.

The Simulink/Stateflow FFCS model’s size and complexity in terms of the Halstead metrics⁶ are as follows. The model has 65 operator blocks, 111 operands, 35 distinct operators, and 95

SOFTWARE RELIABILITY ENGINEERING



Software reliability engineering (SRE) deals with predicting, estimating, and evaluating a target software system's reliability.¹ To apply statistical SRE techniques, developers collect reliability-related metrics throughout the development life cycle by testing the system on the basis of its operational profile.² So, SRE is essentially a quantitative study of software development regarding the given reliability goal. This activity repeats until it achieves the reliability goal. IEEE Standard 1633 provides guidelines with which to evaluate reliability by applying software reliability models.³

Recently, researchers have developed several software reliability prediction models to quantitatively manage software reliability at early development phases (the architecture and design phases), on the basis of system structure and the system usage profile.⁴ However, these models are unrealistic owing to a lack of empirical data, especially for the early development phases. Also, they assume

that each target component's reliability is known, which isn't true for real-world software components. On the other hand, our proposed software reliability validation framework—based on statistical model checking (see the main article and the other sidebar)—validates reliability at an early stage without such limitations.

References

1. M.R. Lyu, "Software Reliability Engineering: A Roadmap," *Proc. Future of Software Eng. Conf. (FOSE 07)*, IEEE CS, 2007, pp. 153–170.
2. J.D. Musa, "Operational Profiles in Software-Reliability Engineering," *IEEE Software*, vol. 10, no. 2, 1993, pp. 14–32.
3. *IEEE Std. 1633, Recommended Practice on Software Reliability*, IEEE CS, 2008.
4. L. Cheung et al., "Early Prediction of Software Component Reliability," *Proc. ACM/IEEE 30th Int'l Conf. Software Eng. (ICSE 08)*, IEEE CS, 2008, pp. 111–120.

distinct operands. So, the calculated program volume V , representing the model's size, is 1,234, and the program difficulty D , representing the model's complexity, is 20.7. The automatically generated C code from the model has 222 functions in 8,266 SLOC. More information on the FFCS model is at www.mathworks.co.kr/products/simulink/examples.html?file=/products/demos/shipping/simulink/sldemo_fuelsys.html.

FFCS Software Reliability Validation

An FFCS requires the ASIL D safety goal, and ASIL D in ISO 26262 requires a $1 - 10^{-3}$ to $1 - 10^{-9}$ reliability goal. So, we specify an FFCS's reliability goal as 0.9999. To determine the reliability goals for each component (the SFDE, airflow calculator, and fuel calculator) and the weight values for the functional-safety requirements, we consulted field experts from FormalWorks. This company produces software tools to test automobile software and conducts consulting for ISO 26262 certification. To obtain the reliability goals and the weight values more accurately, we can use Wideband Delphi estimation⁷ with several iterations of experts' evaluations. We can also use Probe (proxy-based estimation),⁸ another effective technique.

Specifying the functional-safety requirement. Through discussion with the FormalWorks experts who performed hazard analysis, we decided to specify functional-safety

requirements for each of the component's output values. (For example, we specify four requirements for the SFDE, each corresponding to the output values for throttle angle, speed, EGO, and MAP.) So, we specified four safety-critical requirements for the SFDE, two requirements for the airflow calculator, and two requirements for the fuel calculator. During the entire execution period, the SFDE has these requirements:

- $req_{throttle}$. The throttle output shouldn't be out of the throttle opening range (from 3 to 90 percent) for 1 second.
- req_{speed} . The engine speed output shouldn't exceed 628 radians per second (6,000 rpm) for 1 second.
- req_{EGO} . During the initial warm-up period (25 seconds), the EGO output should not be out of the range [0, 1] for 1 second. After the warm-up, the EGO output should be between 0.03 and 0.97.
- req_{MAP} . The MAP output shouldn't exceed one atmosphere.

Assuming that the execution period is 60 seconds, the requirements become these BLTL formulas:

$$req_{throttle} : \neg(F^{60}G^1(throttle_{out} < 3 \parallel throttle_{out} > 90)),$$

$$req_{speed} : \neg(F^{60}G^1(enginespeed_{out} > 628)),$$

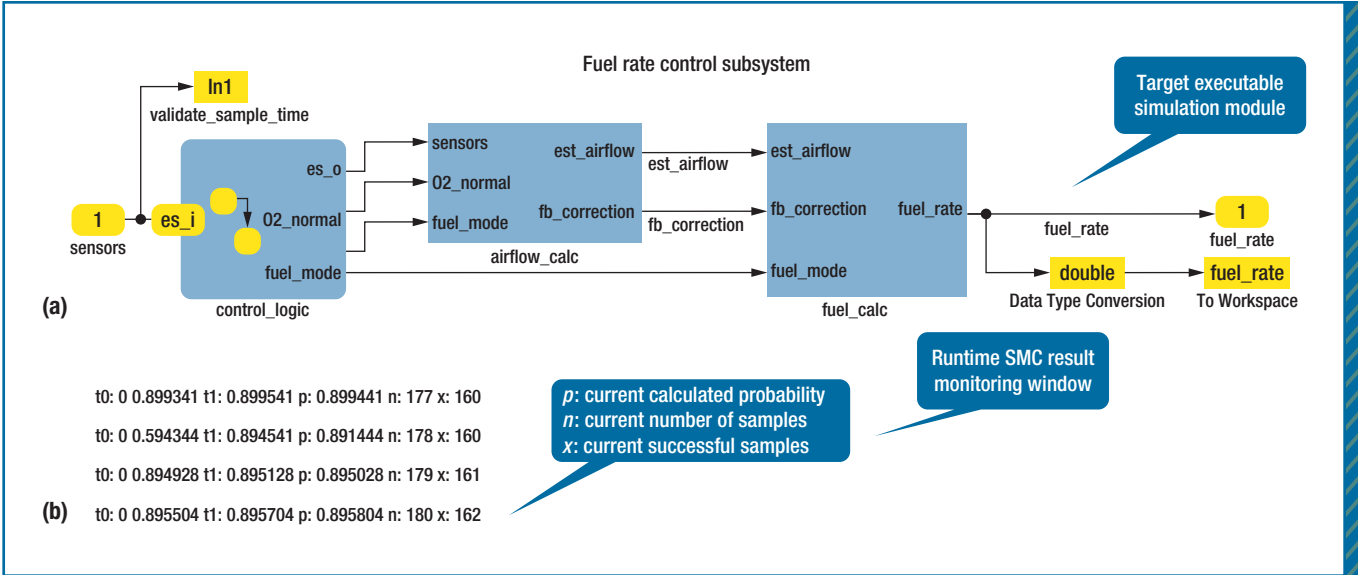


FIGURE 3. Screenshots of an SMC experiment on an FFCS. (a) A diagram of the fuel rate control subsystem. (b) Variable values related to the probability of the sensor failure detector and estimator (SFDE) satisfying $req_{throttle}$. The last line in Figure 3b indicates that 162 of the 180 generated sample traces satisfy $req_{throttle}$ so far. That line also indicates that the probability of the SFDE satisfying $req_{throttle}$ is 0.895604.

$$req_{EGO} : F^{60} \left(\begin{array}{l} \left(\text{warmup} = \text{true} \rightarrow \right. \\ \left. \neg G^1 (EGO_{out} < 0 \parallel EGO_{out} > 1) \right)^\wedge \\ \left(\text{warmup} = \text{false} \rightarrow \right. \\ \left. \neg G^{25} (EGO_{out} < 0.03 \parallel EGO_{out} > 0.97) \right) \end{array} \right),$$

$$req_{MAP} \neg (F^{60} G^{0.1} (MAP_{out} > 1)),$$

where $F^t f$ means that f eventually occurs in t seconds, and $G^t f$ means that f always occurs in t seconds.

Allocating the reliability requirement. Because all the FFCS components are combined sequentially, we can calculate the FFCS's reliability R_T by multiplying the reliabilities of the components of the target R'_i :

$$R_T = \prod_{i=1}^n R'_i,$$

where n is a total number of components.

To satisfy the FFCS's reliability (0.9999), we allocated the components' reliability goals via discussion with Formal-Works experts: 0.99997 for the SFDE, 0.99997 for the air-flow calculator, and 0.99997 for the fuel calculator.

Calculating each component's probability. To calculate probability, we use SMC. (We discuss this in more detail later.)

Validating each component's reliability. We can calculate the reliability of R'_i by assigning a weight to each requirement:

$$R'_i = \sum_{req_{ij} \in REQ} (w_{req_{ij}} \times P(req_{ij})),$$

where $w_{req_{ij}}$ is a weight value for req_{ij} .

Again, through discussion with the experts, we determined the weight values: $w_{throttle} = 0.11$, $w_{speed} = 0.45$, $w_{EGO} = 0.09$, and $w_{MAP} = 0.35$. This indicates that the speed and MAP sensors are more safety-critical than the throttle and EGO sensors. We will explain how to validate the reliability of the SFDE in the next section.

SMC Experiments

We performed all experiments on a 64-bit Windows 7 Professional machine with a 3.40-GHz Intel i5 and 8 Gbytes of memory. We used a Simulink/Stateflow FFCS model in Matlab R2010a. We simulated the model using the Matlab simulator to generate sample execution traces. To validate whether the model satisfies the reliability goal (0.9999), we applied Bayesian interval estimation testing (BIET), an SMC technique.⁹ To obtain a precise probability result (a goal of $1 - 10^{-4}$), we set the SMC precision parameters to $d = 0.00005$ and $c = 0.9999$ for BIET, where d is a half-size of an estimation interval that will contain the probability result and c is the coverage goal of the estimation interval.

Table 1. The statistical-model-checking results for validating the reliability of the sensor failure detector and estimator. The component’s reliability was 0.999973.

Requirement	Weight	Probability	No. of samples	No. of failed samples	Trace generation time (hrs.)	BLTL model-checking time (hrs.)*	BIET analysis time (hrs.)*	Total verification time (hrs.)
$req_{throttle}$	0.11	0.999889	776,747	85	317.17	0.75	0.99	318.91
req_{speed}	0.45	0.999989	92,098	0	37.99	0.19	0.26	38.44
req_{EGO}	0.09	0.999933	533,735	35	220.91	0.75	1.32	222.23
req_{MAP}	0.35	0.999989	92,098	0	38.01	0.20	0.26	38.47

* BLTL stands for bounded linear temporal logic; BIET stands for Bayesian interval estimation testing.

Figure 3 shows a snapshot of an FFCS simulation running with SMC. In Figure 3a, the three component blocks correspond to the FFCS components in Figure 2 (for example, the `control_logic` block corresponds to the SFDE). The `sensors` block represents all four sensor inputs; the `fuel_rate` block represents the fuel rate output.

In Figure 3b, the SMC tool displays variable values related to the probability that the SFDE satisfies $req_{throttle}$. Specifically, p is a calculated probability, n is the number of sample simulation traces so far, and x is the number of successful traces so far. For example, the last line in Figure 3b indicates that 162 of the 180 generated traces satisfy $req_{throttle}$. That line also indicates that the probability of the SFDE satisfying $req_{throttle}$ is 0.895604 so far.

We built a stochastic environment model that generates random faults at the sensors. We made a random-fault generator module and connected it to the sensors. The random faults are modeled by four independent Poisson processes with different arrival rates. The mean interarrival fault rate is 8 for the throttle sensor, 10 for the speed sensor, 9 for the EGO sensor, and 7 for the MAP sensor. For simplicity, we assume that all FFCS operations have the same occurrence rate. For a larger, more complex system, we would have to consider the operational profile so that the most frequently used operation would have the most testing.

We implemented the BLTL model checker (as a proof-of-concept prototype) in 500 lines of Matlab script to evaluate the eight functional-safety properties. In this case, it evaluates $req_{throttle}$, req_{speed} , req_{EGO} , and req_{MAP} over Matlab/Simulink simulation traces.

We implemented the BIET statistical analyzer (<http://pswlab.kaist.ac.kr/tools/SMC>) in 50 lines of Matlab script. The BIET analyzer is independent from the model checker and functional-safety requirements.

We plan to implement and publicly release a general

model checker that can evaluate arbitrary BLTL formulas over Matlab/Simulink simulation traces. The BLTL model checker and the BIET analyzer will be reusable for other target systems without modification.

Experiment Results

Table 1 lists the results of applying SMC to the SFDE. On the basis of the probabilities and weight values in the table, we calculate R'_i as

$$R'_i = 0.11 \times 0.999889 + 0.45 \times 0.999989 + 0.09 \times 0.999933 + 0.35 \times 0.999989 \doteq 0.999973$$

Because the calculated reliability is higher than the goal (0.99997), we conclude that the SFDE satisfies the goal. In total, the experiments consumed approximately 377 Mbytes for simulating the FFCS and 5 Mbytes for BLTL trace checking and BIET analysis.

Generating trace samples consumes 99 percent of the total verification time (for example, 317.17 out of 318.91 hrs. for $req_{throttle}$). So, we can significantly reduce the verification time by generating sample traces in parallel. Because the generated random samples are independent from each other (that is, Bernoulli-independent, identically distributed random samples), we can run multiple simulators on multiple machines to accelerate trace generation. This lets us assess a target component’s reliability within a modest time frame by running hundreds of simulators on a cloud computing platform such as Amazon EC2 (Elastic Compute Cloud). For example, with 100 machines, we can calculate a probability for $req_{throttle}$ in approximately five hours (317.17/100 + 0.75 + 0.99).

To further reduce verification time, we plan to apply hybrid SMC techniques that are faster than BIET.¹⁰



YOUNGJOO KIM is a full-time researcher at S-Core. Her research interests include automated software testing and statistical model checking. Kim received an MS in computer science from the Korea Advanced Institute of Science and Technology. Contact her at jerry88.kim@gmail.com.



OKJOO CHOI is a research assistant professor at the Korea Advanced Institute of Science and Technology's Department of Computer Science. Her research interests include software process, software safety, and reliability. Choi received a PhD in computer science from Sookmyung Women's University. Contact her at okjoo.choi@kaist.ac.kr.




MOONZOO KIM is an associate professor at the Korea Advanced Institute of Science and Technology's Department of Computer Science. His research interests include automated software testing and verification, concurrent program testing, and formal analysis of embedded software. Kim received a PhD in computer and information science from the University of Pennsylvania. He's a member of IEEE and ACM. Contact him at moonzoo@cs.kaist.ac.kr.



JONGMOON BAIK is an associate professor at the Korea Advanced Institute of Science and Technology's Department of Computer Science. His research interests include software process modeling, software economics, software reliability engineering, and software Six Sigma. Baik received a PhD in computer science from the University of Southern California. He's a member of IEEE and ACM. Contact him at jbaik@kaist.ac.kr.



TAI-HYO KIM is the CEO of FormalWorks. His research interests include formal methods and worst-case execution time analysis. Kim received a PhD in computer science from the Korea Advanced Institute of Science and Technology. Contact him at taihyo.kim@formalworks.com.

Many safety-critical system domains, such as the automotive or avionics domains, have adopted model-driven development. So, industries in those domains can incorporate our framework seamlessly. Adopting our framework will increase system reliability and decrease development costs through early detection of design faults or incorrect reliability allocation. 

Acknowledgments

National Research Foundation of Korea grants 2012046172 and 2010-0014375, Ministry of Knowledge Economy/Korea Evaluation Institute of Industrial Technology grant 10041752, and Dual-Use Technology Program grant UM11014RD1 in Korea supported this research.

References

1. D.S. Herrmann, *Software Safety and Reliability*, IEEE CS, 1999.
2. IEC 61508, *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*, Int'l Electrotechnical Commission, 2003.
3. ISO 26262, *Road Vehicles—Functional Safety*, Int'l Org. for Standardization, 2011.
4. *System Reliability Toolkit*, Reliability Information Analysis Center, 2005.
5. J. Lauber, T.M. Guerra, and M. Dambrine, "Air-Fuel Ratio Control in a Gasoline Engine," *Int'l J. Systems Science*, vol. 42, no. 2, 2011, pp. 277–286.
6. M.H. Halstead, *Elements of Software Science*, Elsevier, 1977.
7. A. Stellman and J. Greene, *Applied Software Project Management*, O'Reilly Media, 2005.
8. W.S. Humphrey, *PSP: A Self-Improvement Process for Software Engineers*, Addison-Wesley Professional, 2005.
9. P. Zuliani, A. Platzer, and E.M. Clarke, "Bayesian Statistical Model Checking with Application to Stateflow/Simulink Verification," *Proc. 13th ACM Int'l Conf. Hybrid Systems: Computation and Control (HSCC 10)*, ACM, 2010, pp. 243–252.
10. Y. Kim and M. Kim, "Hybrid Statistical Model Checking Technique for Reliable Safety Critical Systems," *Proc. IEEE Int'l Symp. Software Reliability Eng. (ISSRE 12)*, IEEE CS, 2012; http://pswlab.kaist.ac.kr/publications/issre2012_yjkim.pdf.

