# Formal Modeling and Verification of Safety-Critical Software

**Junbeom Yoo,** *Konkuk University*
**Eunkyoung Jee,** *Korea Advanced Institute of Science and Technology*
**Sungdeok Cha,** *Korea University*

A formal-methods-based process for developing safety-critical software supports development, verification and validation, and safety analysis and has proven to be effective and easy to apply.

Rigorous quality demonstration is important when developing safety-critical software such as a nuclear power plant's reactor protection system (RPS). Although stakeholders strongly recommend using formal modeling and verification, domain experts often reject such methods because the candidate techniques are overabundant, the notations appear complex, the tools often work only in isolation, and the output is frequently too difficult for domain experts to understand and to extract meaningful information.

To overcome such obstacles, we developed a formal-methods-based process that supports development, verification, and safety analysis. We also developed CASE tools to let nuclear engineers apply formal methods without having to know the underlying formalism in depth. In this article, we describe more than seven years' experience working with nuclear engineers in developing RPS software and applying formal methods. Nuclear engineers and regulatory personnel found the process effective and easy to apply with our integrated tool support.

## Developing a Digital Control System

When developing and verifying safety-critical software, formal methods are important for increasing safety assurance and demonstrating compliance with strict regulations. In 2001, the Korean Nuclear Instrumentation and Control System consortium (KNICS; www.knics.re.kr) began developing a digital control system for the APR-1400 reactor. At the project's start, project managers made two decisions that strongly influenced our process:

- When developing safety-critical components such as an RPS, we would use formal methods whenever it was practical to do so.
- Software development would be based on the programmable logic controller (PLC), using function block diagram (FBD) as the implementation language.

As a software engineering research group in computer science, we began working with nuclear engineers to produce a formal requirements specification, develop necessary CASE tools, and conduct formal verification during software development. Figure 1 describes the overall process we developed, which covers three essential aspects of safety-critical software: development, formal verification, and safety demonstration.


0740-7459/09/$25.00 © 2009 IEEE

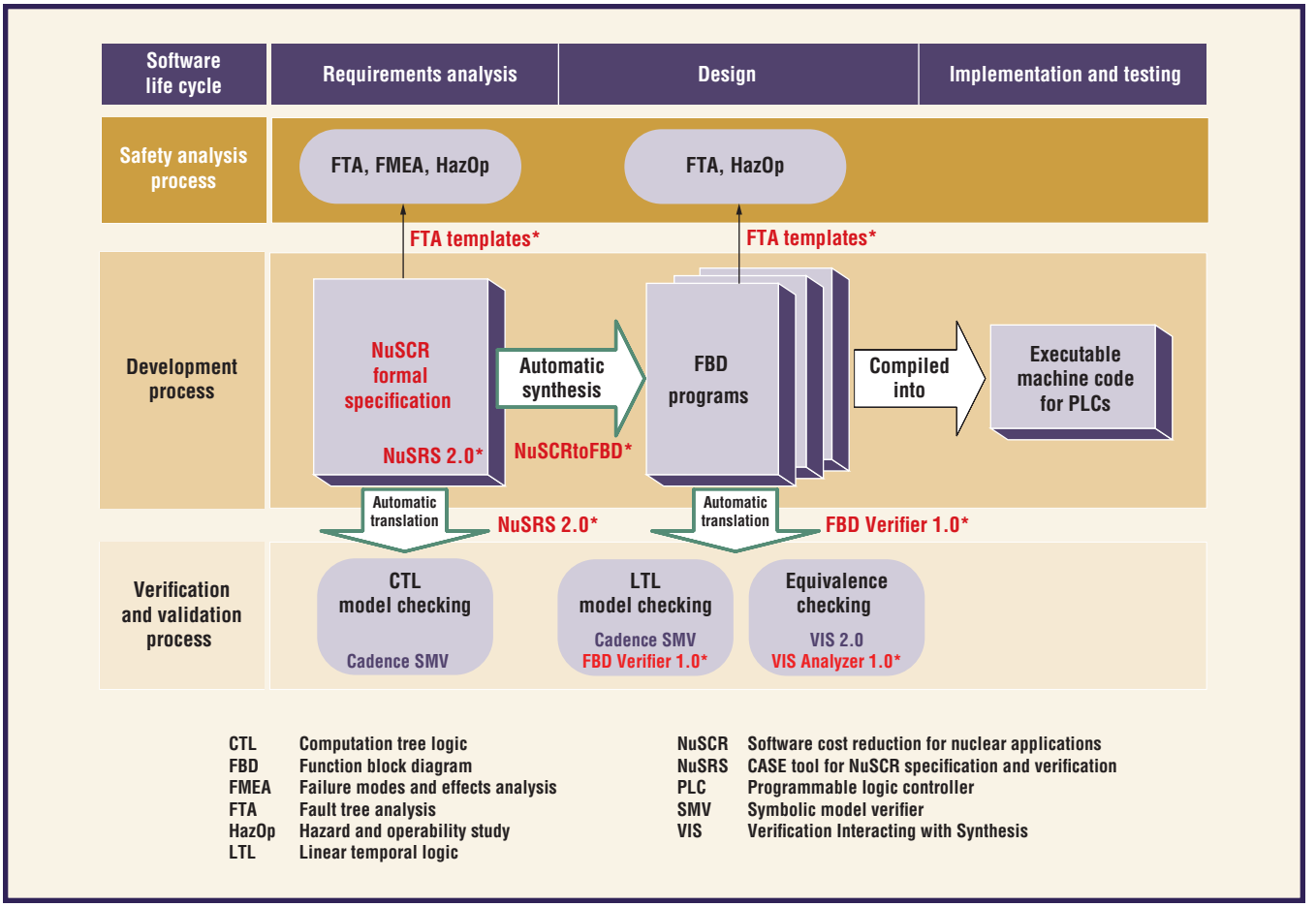


**Figure 1. Software development, verification, and safety analysis for the Korean Nuclear Instrumentation and Control System (KNICS) consortium's reactor protection system software. CASE tools marked with an asterisk were developed by the authors.**

Our group developed the CASE tools marked with an asterisk in the figure.

Although the process is similar to that of a typical software project, little development activity occurs beyond the design phase. Once the final FBD design has successfully completed safety analysis and verification and has been officially approved, a compiler provided by the PLC vendor automatically generates executable code. So, it's fair to treat an FBD as an implementation language, too. Also, a different group tested the software extensively at various levels, according to the standard practices.

In developing the formal-methods-based process in Figure 1, we insisted on three core principles. First, we tried to honor the end users' and stakeholders' opinions whenever practical. In our project, nuclear engineers developing a plant instrumentation and control system were the most important user group. The government's regulatory personnel were also important stakeholders. They needed to review and approve all software requirements, designs, and associated documents for the system to be certified for operation. Because the regulatory agency had experience reviewing a similar system (currently used in a Wolsung plant in Korea), many of the project personnel were familiar with software cost reduction (SCR) and SCR-like tabular notations.[1] However, they felt uneasy about exclusively using the tabular notations. So, we chose the syntax of NuSCR,[2] an SCR-like formal specification language that we customized for nuclear applications, to address the domain experts' concerns. We chose to retain tabular notations with relaxed rules on expressions while introducing automata-like notations for specifying timing behavior. Once the notations became fixed, we defined formal semantics so that we could perform automated analysis and develop the CASE tool. Choosing notations that the domain experts will accept is the first step toward successfully applying formal methods in industrial environments.

Second, we were determined not to reinvent the wheel, by using techniques and tools already proven effective. For example, model-checking theory and tools are mature enough, and we wanted seamless integration with our process. Because FBD was our implementation language, we chose model checkers that would work well with it. For example, because we expected the initial

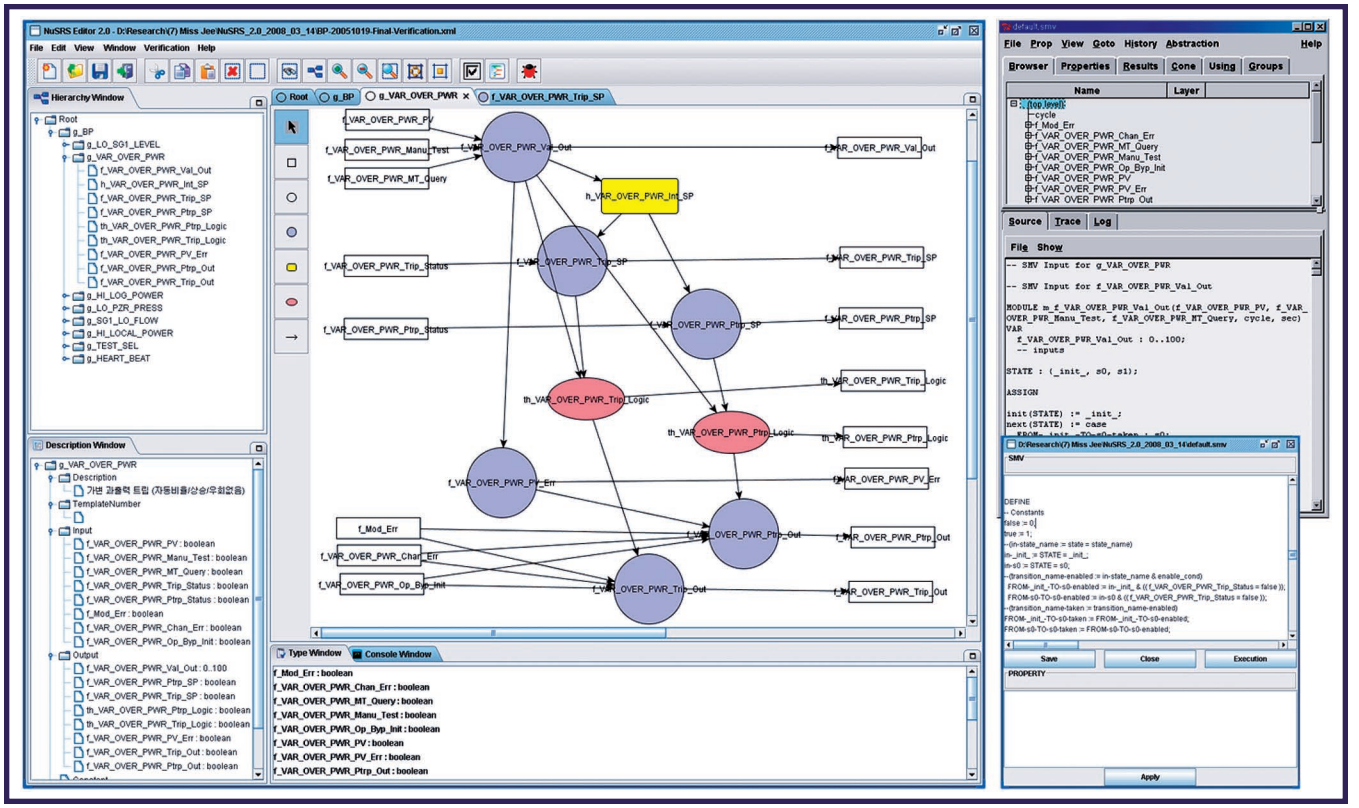

**Figure 2. NuSRS 2.0: A CASE tool for NuSCR specification and verification. The formal modeling and verification tool significantly improves software development productivity as well as safety assurance because it can detect many errors early and automatically.**

FBD design to go through multiple revisions and releases, we were interested in an equivalence verification feature of the VIS (Verification Interacting with Synthesis; http://embedded.eecs.berkeley.edu/research/vis) model checker. Another important factor was a relatively small semantic gap between FBD and Verilog. Because the VIS and Cadence SMV (Symbolic Model Verifier; www.kenmcmil.com/smv.html) model checkers can process Verilog as their inputs, we developed FBD-to-Verilog translation rules and proved semantic equivalence to utilize Cadence SMV and VIS. It's essential to choose "industrial strength" formal methods proven effective in similar applications rather than preaching the pet formalism of formal-method experts.[3]

Third, we provided proper tool support to make formal methods as easy and intuitive as possible. Although VIS equivalence checking was apparently highly useful, nuclear engineers couldn't accept a text-based tool interface. It's nearly impossible for them to understand the output or understand why two designs behave differently. The nuclear engineers didn't have the time for or interest in learning VIS technical details to investigate why two designs revealed different behavior after seven states. Formal-method experts shouldn't blame nuclear engineers for this attitude. Likewise, understanding a Cadence SMV counterexample is really daunting to most people who aren't formal-methods experts. To bridge such semantic gaps, we developed visualization tools so that domain experts could focus on semantic analysis in familiar notations without being overwhelmed by low-level, often partial, and sometimes redundant raw data. For formal methods to be successfully applied in industry, there must be a reasonable interpretation of the results using the terms domain experts understand. Visualization is often the most effective approach.

## Development

To begin requirements analysis, the domain experts prepared a natural-language specification, and we worked with nuclear engineers to prepare formal specification in NuSCR. NuSCR refers to a specification language and the approach we developed, not a document. Hands-on tutorial sessions helped them better understand NuSCR's syntax and semantics. As the language was defined—in close consultation with an expert who understood both domains—most developers accepted NuSCR without much difficulty or resistance. However, at their request, we relaxed rules on expressions on structured decision tables (SDTs), compared to the SCR method, in that NuSCR allowed relational and range expressions. Domain experts insisted that those equations they would be forced

to break into multiple subexpressions actually represent accurate "atomic" domain knowledge, and that automated analysis of completeness and consistency is unnecessary. However, they had trouble understanding specification of timing-related behavior in tabular notation, and they clearly preferred automata-like diagrams.

Our group worked with domain experts in developing a formal specification for two of the four major subsystems, whose natural-language specification was nearly 200 pages long. We developed the formal specification in NuSCR notation, following a process similar to the one Nancy Leveson and her graduate students used to develop a formal specification for the TCAS (Traffic Alert and Collision Avoidance System) for aviation.[4] Nearly 200 NuSCR nodes (for example, SDTs and automata) were scattered in nearly 20 group nodes organized hierarchically. We used the NuSRS CASE tool we developed (see Figure 2).

NuSCR uses a finite-state machine (FSM) to specify state-dependent operations and a timed transition system (TTS), a variant of automata, to specify timing-related requirements. We made various nodes different shapes and colors so that we could easily see their roles. Naming conventions (such as *f_* for functions, *h_* for history variables, and *th_* for timed history variables) also indicate the role. In addition, the function overview diagram (see the middle of Figure 2) illustrates which NuSCR nodes are included in a group node whose prefix is *g_*. Furthermore, all the externally visible inputs and outputs are organized in groups and shown on the left along with their attributes. NuSRS 2.0 supports XML-based interfaces and includes menus to perform automated translation to inputs accepted by Cadence SMV.

Once the domain experts understood NuSCR notation and a reasonably stable CASE tool became available, nuclear engineers could specify most of the formal specification (although they needed our help occasionally). During requirements analysis, domain experts inspected the NuSCR models. Our research group also used Cadence SMV to see whether the NuSCR specification preserved required properties.

Once the experts approved and baselined the formal specification in NuSCR, we could synthesize semantically equivalent FBD designs using rules from our previous research.[5] We emphasized semantic-preserving and correct synthesis rather than an optimal generated FBD design. For example, when some expressions appeared several times in the specification, the synthesized FBD contained redundancy although it was semantically correct. Our experiment revealed that synthesized FBD programs often contain more than twice the number of FBD blocks than manually coded and optimized designs. In the nuclear application, correctness and safety are the most important quality criteria because regulator personnel must rigorously review the design. In addition, there are only a few installations at most, and cost saving through optimal design is rarely a practical concern. However, in different application domains (automotive control systems in particular), an optimal design would become a critical requirement owing to the sheer number of systems to be produced in a highly competitive market.

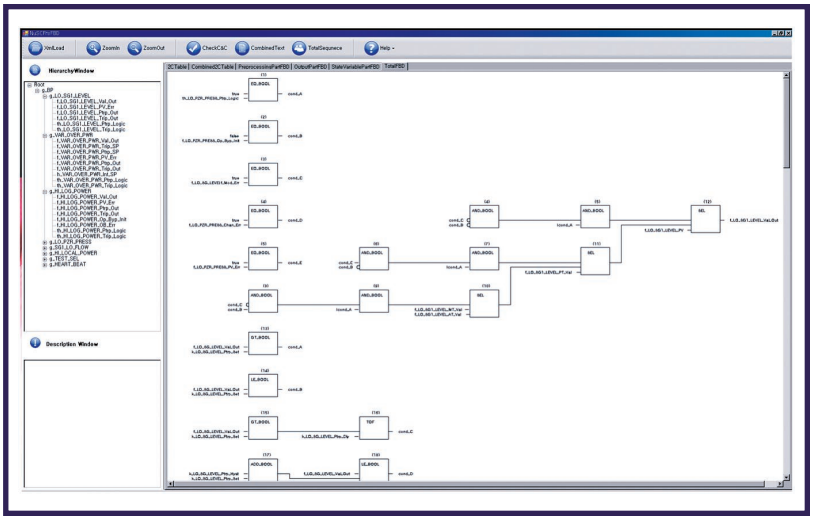

**Figure 3. NuSCRtoFBD: A CASE tool for automatic FBD synthesis from NuSCR. Integrated support for formal methods is critical. If one is forced to manually develop an FBD design from a formal specification, such an approach is unlikely to win acceptance in industry.**

Synthesized design is a useful starting point for engineers to revise and develop official FBD design. Many FBD engineers felt that manual FBD programming, regardless of the specification notations used, was the most error-prone activity. Unfortunately, we couldn't use our FBD synthesis tool, NuSCRtoFBD (see Figure 3), because our team couldn't develop all the necessary CASE tools in time. However, when we consulted the nuclear engineers after the tool development, they felt that such a tool would have significantly improved their productivity.

## Verification

For safety-critical software such as RPS, engineers must perform verification after each phase.[6] Although inspecting the requirements document and FBD design is useful, it's insufficient for meeting rigorous regulatory requirements. During requirements analysis, we developed rules to translate the NuSCR specification into language the Cadence SMV model checker could accept. We implemented automatic translation and seamless

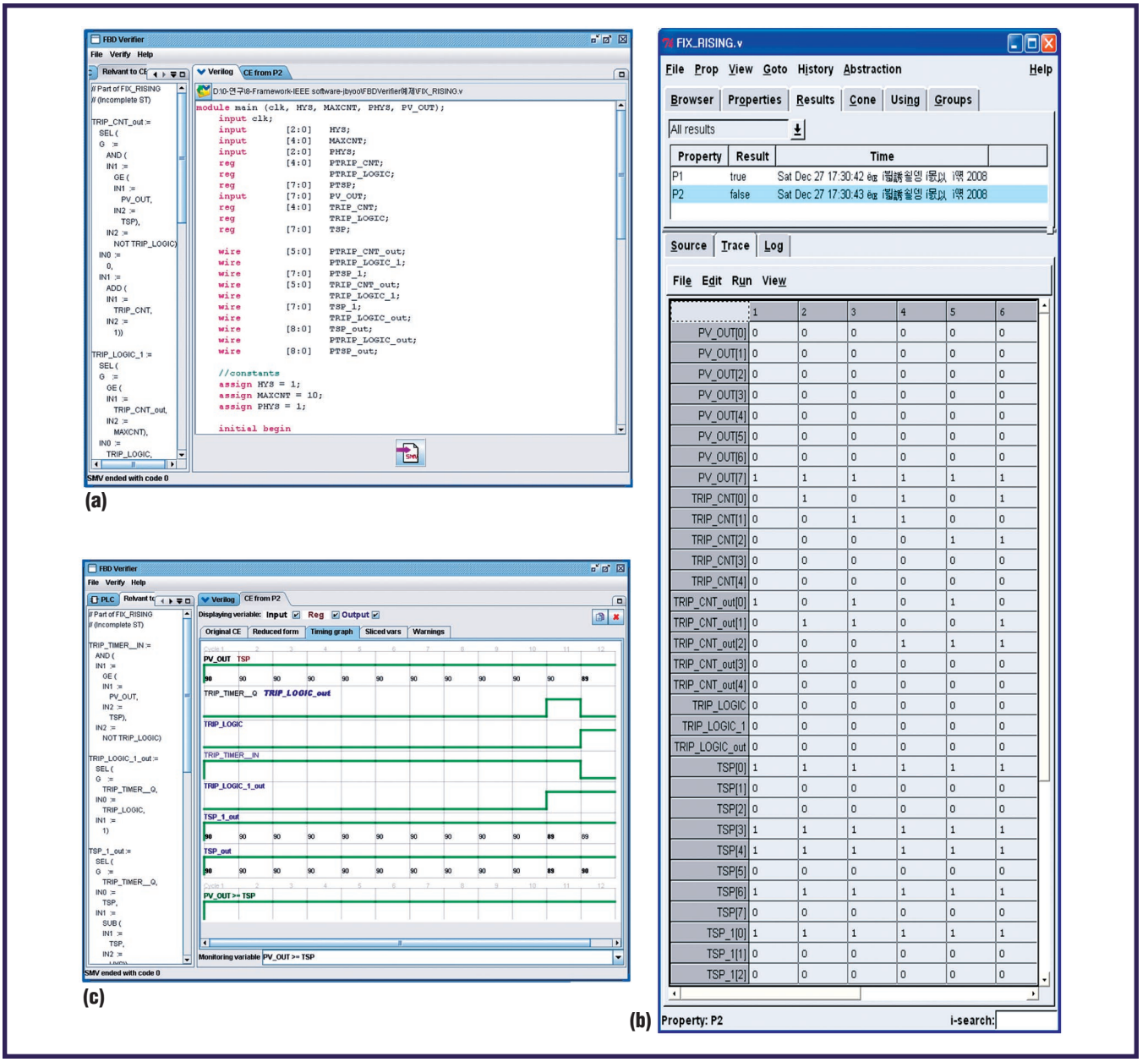


**Figure 4. FBD Verifier: A CASE tool for automatically translating FBD into Verilog programs: (a) automatic translation of a Verilog program, (b) SMV verification results—a counterexample, and (c) the counterexample's timing graph. Without proper visual support, counterexample analysis is boring and potentially error-prone, even for people with technical knowledge of model checking.**

execution of Cadence SMV as one of NuSRS 2.0's pop-up verification menus. In Figure 2, the windows on the right illustrate that domain experts can perform all the verification activities without having to know separate commands for invoking Cadence SMV. The properties to be proved, however, couldn't have been automatically generated, so we encoded temporal-logic formulas in close consultation with nuclear engineers. Such properties, however, rarely change, although details on requirements or FBD design might change.

We also performed Cadence SMV model checking on two subsystems: BP (Bi-stable Processor) and CP (Coincidence Processor). We detected 25 errors, including an incorrect specification in BP. Most errors were omissions in the specification. Although a few mistakes were introduced during formal specification, we discovered and fixed them relatively quickly. We report on the official verification results for a preliminary version of BP in previous work.[7]

During design, we applied Cadence SMV

model checking and VIS equivalence checking on FBD programs. Whereas the former examines whether the FBD programs meet required properties, the latter determines behavioral equivalence between two FBD revisions. For Cadence SMV model checking, we first defined FBD's semantics as a state transition system and developed rules to generate semantically equivalent Verilog programs. Using model-checking techniques, we identified 13 distinct types of errors in the FBD programs and detected several incidents of incorrect FBD logic.[8]

To assist FBD design verification, we developed the FBD Verifier (see Figure 4a). This tool reads FBD programs in standard XML format from the PLC vendor's engineering tools (Figure 4a, left) and translates them into equivalent Verilog programs (Figure 4a, right). We didn't want to overwhelm domain experts with unnecessary details; nevertheless, the tool allows line-by-line comparison to give regulatory personnel and domain experts confidence that the translation is correct. Most domain experts would simply click buttons at the bottom to perform model checking without bothering with FBD and Verilog syntax details. Unfortunately, Cadence SMV often generates counterexamples with excessively primitive details (see Figure 4b). However, the Verilog code in the main window was highly useful for analyzing such counterexamples.

More important, domain experts will likely refuse to use formal methods if they have to manually perform such analysis. So, we developed a feature where users could enter arbitrary expressions and visually display how values change in a manner similar to the timing graph in Figure 4c. Users can choose to display only their items of interest. They can combine existing entries in the timing graph and display how those entries change values in the counterexample. Using the visualization tool, verification personnel can easily understand why model checking failed.

We also used the VIS verification system to determine behavioral equivalence between the two successive FBD revisions. Although VIS accepts Verilog as input, it offers no graphical interface, and the results only partially display relevant information. Existing VIS output is similar to that in Figure 5b (see the next page). To many who are not experts on formal methods, it simply says two Verilog programs exhibited different behavior after seven states but offers no useful insights as to why. It displays only partial information necessary to accurately understand the full scenario. Even a system output display, shown in the sixth state, isn't intuitive. Most nuclear engineers found the information in Figure 5b totally inadequate. However, with the VIS Analyzer tool we developed, engineers not only can compare two Verilog designs side-by-side (see Figure 5a) but also can click the Result Table tab to display the equivalence-checking results in an easily understandable format (see Figure 5c). A recent case study demonstrated that the behavioral-equivalence checking was effective.[9]

## Safety Analysis

Fault tree analysis (FTA) is the most common safety analysis technique; the theory is mature, and the practice is well established for sequential source codes such as C. Developers usually perform FTA manually, using fault tree templates that illustrate potential failure modes. Unfortunately, no FTA template existed for NuSCR specification and FBD nodes, so we developed a set of templates to capture potential failure modes of the NuSCR language constructs[10] and FBD blocks.[11] Figure 6 (see page 49) shows a fault tree template for the AND function block. This template consists of fault events and cause/effect events. The cause/effect events denote fault propagation and help analysts understand the logical operation. Our project partner, an instrumentation and control research group in a nuclear-engineering department, performed fault tree analysis, improved template definitions, and published the final FTA results.[12] Although safety analysis is mandatory in nuclear applications, this application could become optional for other domains.

We developed many of the tool prototypes as the need arose, while the project was in full swing. So, we plan to integrate all the tools, from requirements analysis to design. Most domain experts strongly prefer an integrated development and verification environment where analysis happens behind the scenes and results are displayed, visually if possible, in a language familiar to them. Such languages might vary from one domain to another.

Finally, regardless of technical advances in formal methods, testing will likely always remain an essential component. FBD testing technique is relatively undeveloped. We're developing theories on FBD testing measures so that developers and regulatory bodies can assess the adequacy of testing FBD quantitatively with proper tool support.

## Acknowledgments

**Konkuk University's faculty research fund supported this work in 2008.**

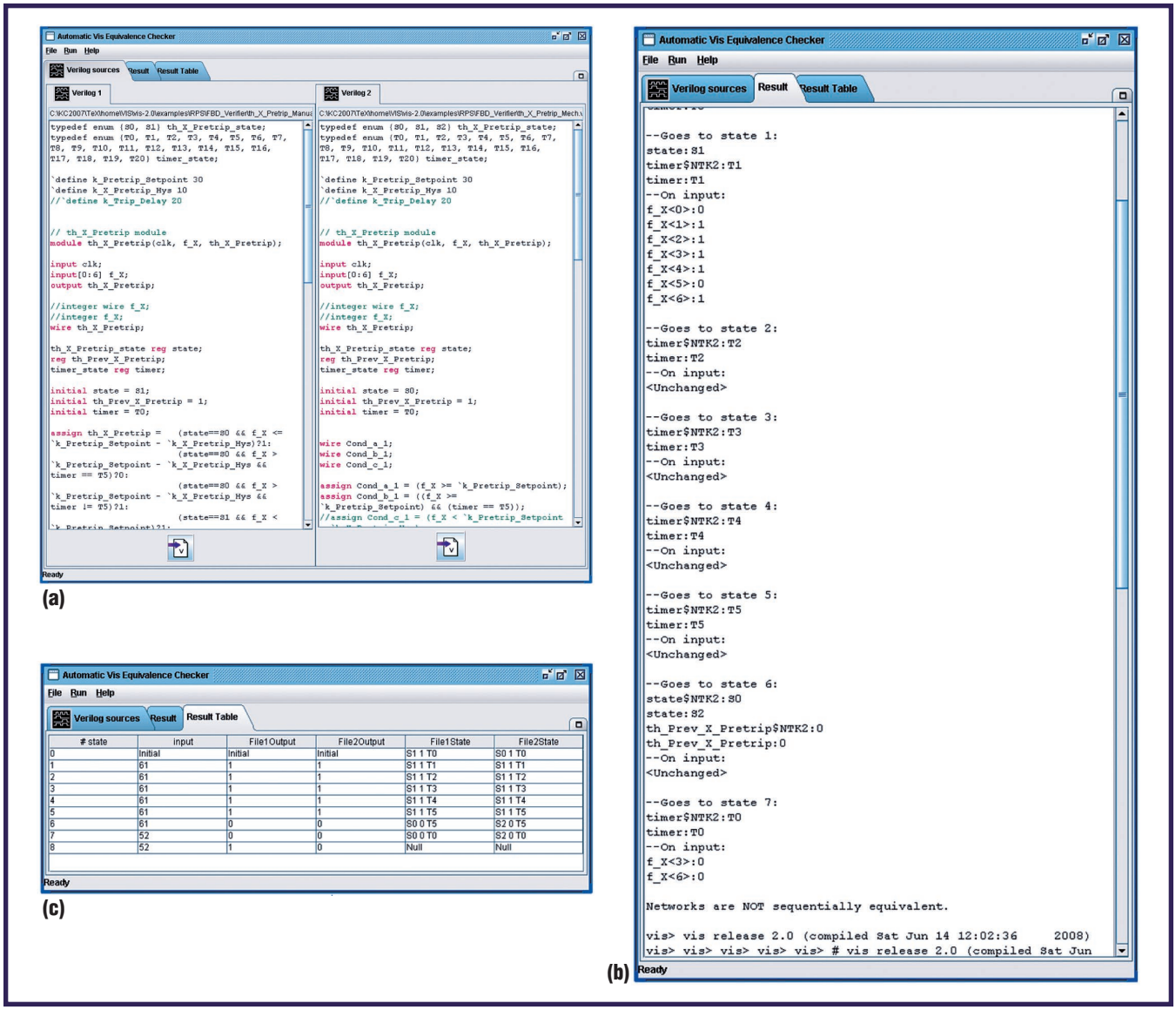


**Figure 5. VIS Analyzer: A CASE tool for seamless execution and visualization of VIS: (a) two Verilog programs, (b) VIS equivalence checking results, and (c) the result table. It's important to reduce the semantic gap between "raw data" and "domain knowledge" whenever possible.**

## References


1. K.L. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," *IEEE Trans. Software Eng.*, vol. 6, no. 1, 1980, pp. 2–13.
2. J. Yoo et al., "A Formal Software Requirements Specification Method for Digital Nuclear Plants Protection Systems," *J. Systems and Software*, vol. 74, no. 1, 2005, pp. 73–83.
3. S. Cha, "Pet Formalisms versus Industry-Proven Survivors: Issues on Formal Methods Education," *J. Research and Practice in Information Technology*, vol. 32, no. 1, 2000, pp. 39–46.
4. M.P.E. Heimdahl and N.G. Leveson, "Completeness and Consistency in Hierarchical State-Based Requirements," *IEEE Trans. Software Eng.*, vol. 22, no. 6, 1996, pp. 363–377.
5. J. Yoo et al., "Synthesis of FBD-Based PLC Design from NuSCR Formal Specification," *Reliability Eng. and System Safety*, vol. 87, no. 2, 2005, pp. 287–294.
6. US Nat'l Research Council, *Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues*, Nat'l Academy Press, 1997.
7. J. Cho, J. Yoo, and S. Cha, "NuEditor—a Tool Suite for Specification and Verification of NuSCR," *Proc. 2nd ACIS Int'l Conf. Software Eng. Research, Management, and Applications* (SERA 04), IEEE Press, 2004, pp. 298–304.
8. J. Yoo, S. Cha, and E. Jee, "A Verification Framework for FBD Based Software in Nuclear Power Plants," *Proc. 15th Asia Pacific Software Eng. Conf.*, IEEE Press, 2008, pp. 385–392.
9. J. Yoo, S. Cha, and E. Jee, "Verification of PLC Programs Written in FBD with VIS," *Nuclear Eng. and Technology*, Feb. 2009.
10. T. Kim, J. Yoo, and S. Cha, "A Synthesis Method of Software Fault Tree from NuSCR Formal Specification Using Templates," *J. Korea Inst. Information Scientists and Engineers*, vol. 32, no. 12, 2005, pp. 1178–1192 (in Korean).

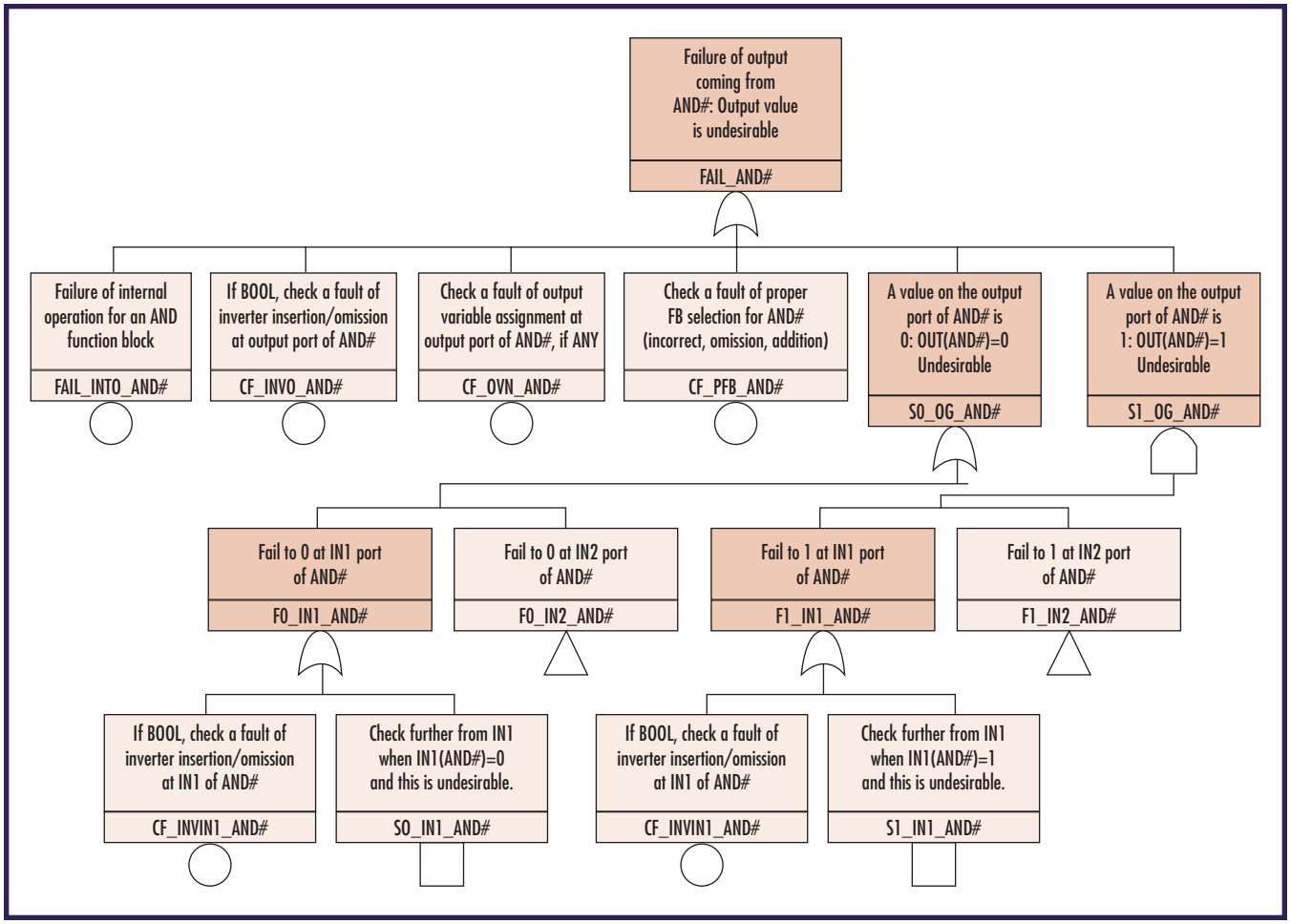


**Figure 6. A fault tree template for the AND function block. It illustrates how failure can occur and what dependencies exist among potential causes.**


11. Y. Oh et al., "Software Safety Analysis of Function Block Diagrams Using Fault Trees," *Reliability Eng. and System Safety*, vol. 88, no. 3, 2005, pp. 215–228.
12. G.-Y. Park et al., "Fault Tree Analysis of KNICS RPS Software," *Nuclear Eng. and Technology*, vol. 40, no. 5, 2008, pp. 397–408.






## About the Authors


**Junbeom Yoo** is an assistant professor in Konkuk University's Department of Computer Science and Engineering. His research interests include requirements engineering and formal methods. Yoo has a PhD in computer science from the Korea Advanced Institute of Science and Technology. Contact him at jbyoo@konkuk.ac.kr.

**Eunkyoung Jee** is a PhD candidate at the Korea Advanced Institute of Science and Technology. Her research interests include software testing and safety-critical software. Jee has an MS in computer science from the Korea Advanced Institute of Science and Technology. Contact her at ekjee@dependable.kaist.ac.kr.

**Sungdeok (Steve) Cha** is a professor in Korea University's Computer Science and Engineering Department. His research interests include software safety and computer security. Cha has a PhD in information and computer science from the University of California, Irvine. Contact him at scha@korea.ac.kr.