VIS Analyzer: A Visual Assistant for VIS Verification and Analysis

Sehun Jeong
College of Info. and Comm.
Korea University
Seoul, South Korea
gifaranga@korea.ac.kr

Junbeom Yoo
College of Info. and Comm.
Konkuk University
Seoul, South Korea
jbyoo@konkuk.ac.kr

Sungdeok Cha
College of Info. and Comm.
Korea University
Seoul, South Korea
scha@korea.ac.kr

***Abstract*—Formal verification plays an important role in demonstrating the quality of safety-critical [1] systems such as nuclear power plants. We have used the VIS verification system [2] to determine behavioral equivalence between two successive revisions in developing the KNICS RPS (Reactor Protection System) [3] in Korea. The VIS accepts a high-level programming language Verilog [4] as input, and its verification results contain valuable information about one reason of the failure. However the VIS offers no graphical interface, and partially displays relevant information necessary to understand the full verification scenario accurately. Many nuclear engineers and verification experts found the information insufficient, and it makes hard to the wide use of the VIS verification system in industry. This paper proposes the VIS Analyzer, a visual assistant for VIS verification and analysis, which can help nuclear engineers take full benefits of VIS without being overwhelmed by incomplete and low-level details. The VIS Analyzer automates the VIS verification processes such as equivalence checking and model checking, and displays the verification results in visual formats. We used a recent case study introduced in [5] to demonstrate its effectiveness and usefulness.**

I. Introduction

Benefits of formal methods [6] cannot be emphasized enough. Regulatory bodies, such as KINS (Korea Institute of Nuclear Safety), often mandate that developers or SQA (Software Quality Assurance) teams apply formal methods as a means of safety demonstration. Among many tools and techniques for formal methods, model checking techniques [7] have received the most attention of research communities as well as practitioners. Model checker performs exhaustive search of systems behavior without any users intervention in the checking process, and also generates useful output called a counterexample for the failed case. Engineers can get insights into the system behavior which is difficult to get through other methods (e.g. inspection and testing). We can also get a high level of safety assurance if a model checker reports success, indicating the system behavior satisfies the required properties.

Outputs from formal verification tools such as the VIS and the SMV [10] are often quite difficult for practitioners to understand. The reasons are as follows. First, semantic gap between domain knowledge of practitioners and model checkers' output is too significant for practitioners to understand them sufficiently. Most model checkers have their own programming languages as input, and practitioners have to understand the input program translated from its requirements specifications or models. Second, counterexamples may contain too excessive or redundant information to reason about the source of failures. Efficient visualization of the counterexamples is a key to take full advantage of model checking. Third, counterexamples from such model checkers as the VIS often contain partial information on states and values. The information partiality includes omission of unchanged values between two successive steps in counterexamples. Automatic reorganization of the partial information into complete one is also crucial for its wide use. Fourth, verification processes of formal verification tools are overly long and detail for domain engineers. For example, the single VIS model checking task requires six to seven commands inputting in a row. Carefully abstracted verification process provides benefits of formal verification without too much efforts for retraining.

The VIS (Verification Interacting with Synthesis) is a widely used tool that integrates formal verifications, simulation, and synthesis of finite states hardware systems. It uses Verilog as a front-end and supports fair CTL (Computational Tree Logic) model checking, language emptiness checking, combinational and sequential equivalence checking, cycle-base simulation, and hierarchical synthesis. We have used the VIS to formally verify the Verilog programs translated from FBD (Function Block Diagram) programs in our verification framework proposed in [5].

This paper proposes an assistant tool, VIS Analyzer (ver. 3.0), to automate the VIS verification processes (equivalence checking and model checking) and support visual analysis of the verification results. In case of equivalence checking, the VIS Analyzer reads two Verilog programs and executes sequential or combinational equivalence checking seamlessly. It also reorganizes omitted information in the verification results through the VIS simulation on background, and displays them in intuitive tabular and flowchart forms. When performing model checking, it reads one Verilog program and CTL properties, executes model checking, and seamlessly shows the verification result visually to aid understanding. We

1555-0885/10 $26.00 © 2010 IEEE
DOI 10.1109/ISORC.2010.41