Delayed Nondeterminism in Model Checking Embedded Systems Assembly Code

Thomas Noll\textsuperscript{1} and Bastian Schlich\textsuperscript{2}

\textsuperscript{1} Software Modeling and Verification Group
RWTH Aachen University
52074 Aachen, Germany
noll@cs.rwth-aachen.de
\textsuperscript{2} Embedded Software Laboratory
RWTH Aachen University
52074 Aachen, Germany
schlich@cs.rwth-aachen.de

Abstract. This paper presents an approach to the efficient verification of embedded systems. Such systems usually operate in uncertain environments, giving rise to a high degree of nondeterminism in the corresponding formal models, which in turn aggravates the state explosion problem. Careful handling of nondeterminism is therefore crucial for obtaining efficient model checking tools. Here, we support this goal by developing a formal computation model and an abstraction method, called delayed nondeterminism, which instantiates nondeterministic values only if and when this is required by the application code. It is shown how this technique can be integrated into our CTL model checking tool \texttt{/mc/square} by introducing symbolic abstract states which represent several concrete states. We also give a simulation relation between the concrete and the abstract state space, thus establishing the soundness of delayed nondeterminism with respect to “path-universal” logics such as ACTL and LTL. Furthermore, a case study is presented in which three different programs are used to demonstrate the effectiveness of our technique.

1 Introduction

Embedded systems are frequently used in safety critical systems. Full testing of these systems is often not possible due to fast time to market or uncertain environments. To address these problems, industry recognized model checking as a promising future tool for the analysis of such systems.

The first model checking tools available worked on proprietary models (e.g., SMV [1], Spin [2] and Uppaal [3]). To use them, the user had to remodel the system under consideration in the input language of the corresponding tool. Then, there were model checking tools which worked on higher level programming languages (e.g., C, C++ or Java). Nowadays, model checking of assembly language (machine level language) gets into focus of research, cf. [4,5,6,7,8].

Model checking assembly language has several advantages. Writing microcontroller programs in higher level languages usually involves direct hardware access...
or embedded assembly instructions, which is not supported by most of the tools. Moreover, the assembly code is the code that is actually deployed to the hardware. Hence, it is not an intermediate representation as the C code. Thus, all errors introduced during the complete development process can be found in the assembly code, including, for instance, errors in the compiler and errors in handling the hardware. The model checker does not have to consider the behavior of the compiler as when model checking C code. In contrast to C code, assembly code usually has a clean, formal and well documented semantics.

On the other side, model checking assembly code has two disadvantages. First, the created state spaces tend to be bigger than when model checking higher level languages as more details are involved. Second, the analysis is hardware dependent, and hence, model checking tools have to be adapted to every processor that should be supported.

In order to tackle this problem, we have developed [mc/square], which is a discrete, (mostly) explicit state, on-the-fly, Computation Tree Logic (CTL) model checker. It is capable of model checking assembly code written for certain microcontrollers (ATMEL ATmega and Infineon XC167). It was important for us not to restrict the set of supported constructs and to process arbitrary assembly code given by the user (including, e.g., direct and indirect memory access, recursions, and functions). Additionally, the user should not be forced to provide an environment. To address the disadvantage of being hardware-dependent, we developed an extensible architecture, which was described in [9]. To deal with the state explosion problem, we implemented different abstraction techniques in [mc/square]. One of these, which is described in this paper, is called delayed nondeterminism.

Delayed nondeterminism is an abstraction technique featuring two aspects which help to reduce the state space size. First, it tries to limit the number of bits (bytes) that have to be split up when determining nondeterministic values (e.g., when input is read from the environment and not all bits are needed for evaluation). Second, it tries to delay the split up as long as possible, i.e., nondeterministic values only have to be instantiated when the corresponding values are needed for evaluation. Delayed nondeterminism is implemented by introducing abstract states into [mc/square]. That is, a state in [mc/square] no longer represents a single concrete state, but may represent many concrete states.

This paper is structured as follows. We start with the presentation of related work. Then, a basic introduction to [mc/square] is given. In the following section, our formal approach to modeling microcontrollers is presented. As an example, the model of the ATMEL ATmega16 microcontroller is detailed. Then, the abstraction technique of delayed nondeterminism is introduced. It is shown that delayed nondeterminism induces a simulation relation between the concrete and the abstract state space. After that, a case study is presented which demonstrates the effect of delayed nondeterminism on the state space size of three different programs. In the end a conclusion is drawn and some potential directions for future improvements are shown.