NuSMV: a new Symbolic Model Verifier

A. Cimatti[1] E. Clarke[2] F. Giunchiglia[1] M. Roveri[1,3]

[1]ITC-IRST, Via Sommarive 18, 38055 Povo, Trento, Italy
{cimatti,fausto,roveri}@irst.itc.it
[2]SCS, Carnegie-Mellon University, 5000 Forbes Avenue, Pittsburgh, PA
15213-3891, USA, Edmund.Clarke@cs.cmu.edu
[3]DSI, University of Milano, Via Comelico 39, 20135 Milano, Italy

1 Introduction

This paper describes NuSMV, a new symbolic model checker developed as a joint project between Carnegie Mellon University (CMU) and Istituto per la Ricerca Scientifica e Tecnolgica (IRST). NuSMV is designed to be a well structured, open, flexible and documented platform for model checking. In order to make NuSMV applicable in technology transfer projects, it was designed to be very robust, close to the standards required by industry, and to allow for expressive specification languages.

NuSMV is the result of the reengineering, reimplementation and extension of SMV [6], version 2.4.4 (SMV from now on). With respect to SMV, NuSMV has been extended and upgraded along three dimensions. First, from the point of view of the system functionalities, NuSMV features a textual interaction shell and a graphical interface, extended model partitioning techniques, and allows for LTL model checking. Second, the system architecture of NuSMV has been designed to be highly modular and open. The interdependencies between different modules have been separated, and an external, state of the art BDD package [8] has been integrated in the system kernel. Third, the quality of the implementation has been strongly enhanced. This makes of NuSMV a robust, maintainable and well documented system, with a relatively easy to modify source code. NuSMV is available at `http://nusmv.irst.itc.it/`.

2 System Functionalities

NuSMV can process files written in SMV language [6], and allows for the construction of the model with different modalities, reachability analysis, fair CTL model checking, computation of quantitative characteristics of the model, and generation of counterexamples. In addition, NuSMV features an enhanced partitioning method for synchronous models based on [7], and allows for disjunctive partitioning of asynchronous models, and for the verification of invariant properties in combination with

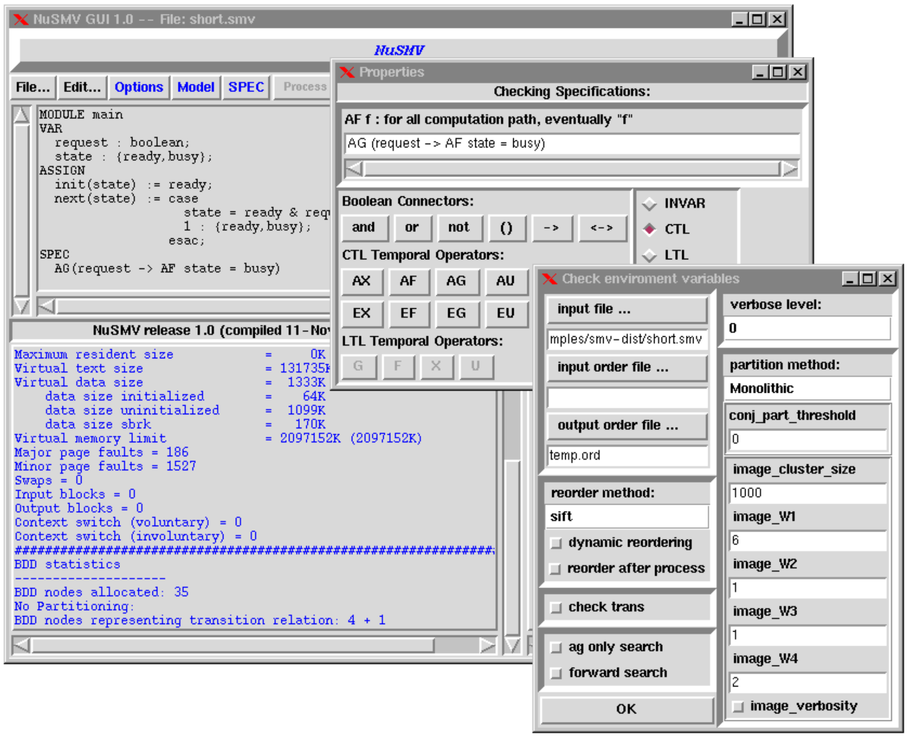

Figure 1: A snapshot of the NuSMV GUI.

reachability analysis. Furthermore, NUSMV supports LTL model checking. The algorithm is based on the combination of a tableau constructor for the LTL formula with standard CTL model checking, along the lines described in [5].

NUSMV can work in batch mode, just like SMV, processing an input file according to the specified command line options. In addition, NUSMV has an interactive mode: it enters a shell performing a read-eval-print loop, and the user can activate the various computation steps (e.g. parsing, model construction, reachability analysis, model checking) as system commands with different options. (This interaction mode is largely inspired by the VIS interaction mode [2].) These steps can therefore be invoked separately, possibly undone or repeated under different modalities. Each command is associated with an on-line help. Furthermore, the internal parameters of the system can be inspected and modified to tune the verification process. For instance, the NUSMV interactive shell provides full access to the configuration options of the underlying BDD package. Thus, it is possible to investigate the effect of different choices (e.g. whether and how to partition the model, the impact of different cache configurations) on the verification process. For instance, it is possible to control the application of BDD variable orderings in a particular phase of the verification (e.g. after the model is built).

On top of the interactive shell, a graphical user interface (GUI from now on) has been developed (Figure 1). The GUI provides an integrated environment to edit and verify the file containing the model description. It provides graphical access to all the commands interpreted by the textual shell of NuSMV, and allows for the modification of the options in a menu driven way. Moreover, the GUI offers a formula editor which helps the user in writing new specifications. Depending on the kind of formula being edited (e.g. propositional, CTL, LTL), various buttons corresponding to modalities and/or boolean connectors are activated and deactivated.

3 System Architecture

Model checking is often referred to as "push-button" technology. However, it is very important to be able to customize the model checker according to the system being verified. This is particularly true in technology transfer, when the model checker may act as the kernel for a custom verification tool, to be used for a very specific class of applications. This may require the development of a translator or a compiler for a (possibly proprietary) specification language, and the effective integration of decomposition techniques to tackle the state explosion.

NuSMV has been explicitly designed to be an open system, which can be easily modified, customized or extended. The system architecture of NuSMV has been structured and organized in modules. Each module implements a set of functionalities and communicates with the others via a precisely defined interface. A clear distinction between the system back-end and front-end has been enforced, in order to make it possible to reuse the internal components independently of the input language being used to describe the model.

The architecture of NuSMV (see Figure 2) is composed of the following modules:

Kernel. The kernel provides the low level functionalities such as dynamic memory allocation, and manipulation of basic data structures (e.g. cons cells, hash tables). The kernel also provides all the basic BDD primitives, directly taken from the CUDD [8] BDD package. The integration of the CUDD package hides the details of the garbage collection. The NuSMV kernel can be used as a black box, following coding standards which have been precisely defined.

Parser. This module implements the routines to process a file written in NuSMV language, check its syntactic correctness, and build a parse tree representing the internal format of the input file.

Compiler. This module is responsible for the compilation of the parsed model into BDDs. The *Instantiation* submodule processes the parse tree, and performs the instantiation of the declared modules, building a description of the finite state machine (FSM) representing the model. The *Encoding* submodule performs the encoding of data types and finite ranges into boolean domains. Having separated this module makes it possible to have different encoding policies which can be more appropriate for different kind of variables (e.g. data path, control path). The *FSM Compiler* submodule provides the routines for constructing and manipulating FSM's at the BDD level. It is responsible of all the necessary semantic checks on the read model, such as the absence of circular definitions. The FSM's can be represented in monolithic or partitioned form [3]. The heuristics used to perform the conjunctive partitioning of the transition relation and reordering of the clusters [7] have been developed to work at the BDD level, independently of the input language. The interface to other modules is given by the primitives

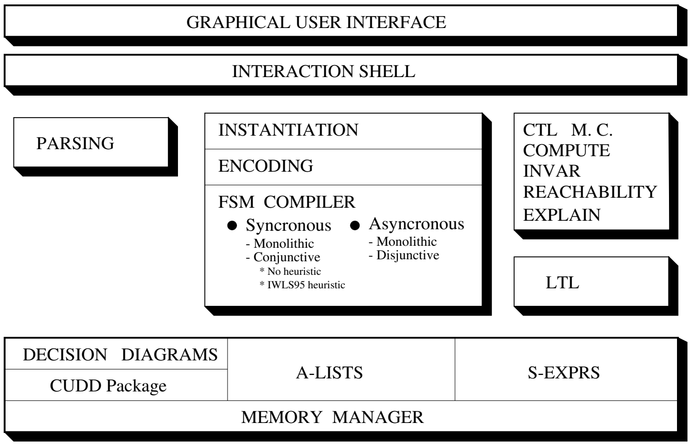

Figure 2: The NuSMV system architecture.

for the computation of the image and counter-image of a set of states. These primitives are independent of the method used to represent the transition relation.

Model Checking. This module provides the functionalities for reachability, fair CTL model checking, invariant checking, and computation of quantitative characteristics. Moreover, this module provides the routines for counterexample generation and inspection. Counterexamples can be produced with different levels of verbosity, in the form of reusable data structures, and can subsequently be inspected and navigated. All these routines are independent of the particular method used to represent the FSM.

LTL. The LTL module is a separated module which calls an external program that translates the LTL formula into a tableau suitable to be loaded into NuSMV. This program also generates a new CTL formula to be verified on the synchronous product of the original system and the generated tableau.

Interactive shell. From the interaction shell the user has full access to all the functionalities provided by the system.

Graphical user interface. The graphical user interface has been designed on top of the interactive shell. It allows the user to inspect and set the value of the environment variables of the system, and provides full access to all the functionalities.

4 Implementation

NuSMV has been designed to be robust, close to the standards required by industry and easy to maintain and modify. NuSMV is written in ANSI C and is POSIX compliant. This makes the system portable to any compliant platform. It has been throughly debugged with `Purify` (`http://www.pureatria.com`) to detect memory leaks and runtime memory corruptions errors.

The kernel of NuSMV provides low level functionalities, such as dynamic memory allocation, in a way independent from the underlying operating system. Moreover, it provides routines for the manipulation of basic data structures such as cons cells, hash tables, arrays of generic types, and encapsulates the CUDD BDD package [8].

In order to implement the architecture depicted in Section 3, the source code of NuSMV has been organized in different packages. NuSMV is composed of 11 packages. Each package exports a set of routines which manipulate the data structures defined in the package and which allow to modify the options associated to the functionalities provided by the package itself. Moreover, each package is associated with a set of commands which can be interpreted by the NuSMV interactive shell. We have packages for model checking, FSM compilation, BDD interface, LTL model checking and kernel functionalities. New packages can be added relatively easily, following precisely defined rules.

The GUI has been developed in Tcl/Tk. It runs as a separate process, synchronously communicating with NuSMV by issuing textual commands to the interactive shell, and processing the resulting output to display it graphically.

The code of NuSMV has been documented following the standards of the `ext` tool (`http://alumnus.caltech.edu/~sedwards/ext`), which allows for the automatic extraction of the programmer manual from the comments in the system source code. The programmer manual is available in TXT or HTML format, and can be browsed by an HTML viewer. This tool is also used to generate the help on line available through the interactive shell and via the graphical user interface.

The user manual has been written following the TEXINFO standard, from which different formats (i.e. POSTSCRIPT, PDF, DVI, INFO, HTML) can be automatically generated, and accessed via an HTML viewer or in hardcopy.

5 Results and Future Directions

NuSMV is a robust, well structured and flexible platform, designed to be applicable in technology transfer projects. The performance of NuSMV have been compared with those of SMV by running a number of SMV examples. Despite the fact that NuSMV gives up some of the optimizations of SMV to simplify the dependencies between modules, an improvement in computation time has been obtained. In most examples NuSMV performs better than SMV, in particular for larger examples. This enhancement in performance is mainly due to the use of CUDD BDD package.

The NuSMV architecture provides a precise distinction between the front-end, specific to the SMV input language, and the back-end (including the heuristics for model partitioning and model checking algorithms), which is independent of the input language. This separation has been used to develop on top of NuSMV the MBP system. MBP is a planner able to synthesize reactive controllers for achieving goals in nondeterministic domains [4].

Functionalities currently under development are a simulator, which is of paramount importance for the user to acquire confidence in the correctness of the model, and a compiler for an imperative style input language, which can often be very convenient in the modeling process. Further developments will include the integration of decomposition techniques (e.g. abstraction and compositional verification), and new and very promising techniques based on the use of efficient procedures for propositional satisfiability, following the ideas reported in [1].

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proc. TACAS'99*, March 1999. To appear.

[2] R. K. Brayton et al. VIS: A system for Verification and Synthesis. In *Proc. of CAV'96*. LNCS 1102, Springer-Verlag.

[3] J. Burch, E. Clarke, and D. Long. Representing Circuits More Efficiently in Symbolic Model Checking. In *Proc. of the 28th ACM/IEEE Design Automation Conference*, pages 403–407, Los Alamitos, CA, June 1991. IEEE Computer Society Press.

[4] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proc. of the 15th National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin, 1998. AAAI-Press.

[5] O. Grumberg E. Clarke and K. Hamaguchi. Another Look at LTL Model Checking. *Formal Methods in System Design*, 10(1):57–71, February 1997.

[6] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publ., 1993.

[7] R. K. Ranjan, A. Aziz, B. Plessier, C. Pixley, and R. K. Brayton. Efficient BDD algorithms for FSM synthesis and verification. In *IEEE/ACM Proceedings International Workshop on Logic Synthesis*, Lake Tahoe (NV), May 1995.

[8] F. Somenzi. CUDD: CU Decision Diagram package — release 2.1.2. Department of Electrical and Computer Engineering — University of Colorado at Boulder, April 1997. `ftp://vlsi.colorado.edu/pub/`