The NUXMV Symbolic Model Checker*

Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio,
Alessandro Mariotti, Andrea Micheli, Sergio Mover,
Marco Roveri, and Stefano Tonetta

Fondazione Bruno Kessler

Abstract. This paper describes the NUXMV symbolic model checker for finite-
and infinite-state synchronous transition systems. NUXMV is the evolution of the
NUSMV open source model checker. It builds on and extends NUSMV along
two main directions. For finite-state systems it complements the basic verification
techniques of NUSMV with state-of-the-art verification algorithms. For infinite-
state systems, it extends the NUSMV language with new data types, namely Inte-
gers and Reals, and it provides advanced SMT-based model checking techniques.

Besides extended functionalities, NUXMV has been optimized in terms of per-
formance to be competitive with the state of the art. NUXMV has been used in
several industrial projects as verification back-end, and it is the basis for sev-
eral extensions to cope with requirements analysis, contract based design, model
checking of hybrid systems, safety assessment, and software model checking.

1 Introduction

NUSMV [1] is a symbolic model checker for finite state fair transition systems. It has
been developed jointly by Carnegie Mellon University, the University of Trento and
Fondazione Bruno Kessler (FBK) since 1999. It is distributed as open source under
the LGPL license, and it integrates some of the most successful BDD and SAT based
symbolic model checking algorithms up to 2011 (the year of its last official release).

Since its first release in 1999, the public available version of NUSMV has been
complemented and extended, internally at FBK, with multiple functionalities. This was
done in order to facilitate its deployment in several operational settings, and to take into
account the needs resulting from industrial and research projects. In particular, we in-
tegrated functionalities for requirements engineering, safety assessment, contract based
design, and techniques for the analysis of hybrid systems; we also interfaced NUXMV
with SMT engines. The whole set of new functionalities were developed as a single
code-base, referred in several papers as NuSMT and NuSMV3. To better maintain such
features, and to facilitate the deployment, we started a re-engineering process where we
separated all of them in several different tools. In this view, NUSMV is the code-base
that provides basic functionalities and common data structures to all the other tools (e.g.
symbol table, handling of expressions, interface with the CUDD [2] BDD package, in-
terface with the SAT solvers (e.g. MINISAT [3]). NUSMV also provides all the basic
model checking algorithms for the pure Boolean case.

In this paper we describe NUXMV, a new symbolic model checker for finite- and infinite-state synchronous fair transition systems. NUXMV is the evolution of NUSMV, as such it builds on NUSMV and extends it along two main directions. For finite-state systems, it complements NUSMV basic verification techniques with a family of new state-of-the-art verification algorithms. For infinite-state systems, it extends the NUSMV language with new data types, namely Integers and Reals, and it provides advanced SMT-based model checking techniques. NUXMV participated in the 2013 hardware model checking competition (HWMCC'13) positioning among the first four in the single and multiple tracks. NUXMV also compares well with other model checkers for infinite-state systems. Finally, NUXMV has been successfully used in several application domains and in industrial settings. It is currently the core verification engine for many other tools (also industrial ones). The tool is distributed in binary code, free to be used for academic research and for non-commercial uses. The latest version of NUXMV can be downloaded from https://nuxmv.fbk.eu.

2 Functionalities

NUXMV inherits, and thus provides to the user, all the functionalities of NUSMV [1]. In this section we describe all the new features distinguishing them in those for the analysis of finite-state domains, those for the analysis of infinite-state domains, and other generic features.

2.1 Analysis of Finite-State Domains

NUXMV complements the NUSMV language with the AIGER [4] format. AIGER is the language adopted in the hardware model checking competition. Once the AIGER file is read, the internal data structures of NUXMV are populated, and it is possible to verify the properties (if any) with any of the available verification algorithms, or specify new properties interactively "playing" with the design.

NUXMV implements a vast portfolio of algorithms for invariant checking. We currently provide an implementation for the McMillan interpolation-based approach [5] and for the interpolation sequence approach [6]. Interpolation based algorithms are complemented with k-induction algorithms [7] and a family of algorithms based on IC3 [8,9,10]. The IC3 algorithm using abstraction refinement [10] comes in two variant depending on the approach to refinement: the original one based on IC3, and a new variant based on BMC. All these techniques, benefit from the use of temporal decomposition [11] and from techniques to discover equivalences to simplify the problem. We remark that, to implement the interpolation based algorithms we extended MINISAT [3] to build a resolution proof.

Still related to the verification of invariants, we also improved the BDD based invariant checking algorithms by allowing the user to specify hints in the spirit of guided reachability [12]. The hints are specified using a restricted fragment of the PSL SERE [13]. The hints can also be used to compute the full set of the reachable states.

For LTL SAT based model checking, we complemented the BMC based algorithms of NUSMV [14,15] with k-liveness [16] integrated within an IC3 framework. K-liveness is based on counting and bounding the number of times a fairness constraint can become

true. This is used in conjunction with the construction of a monitor for LTL properties, for which we use the LTL2SMV [17] as provided by NuSMV.

2.2 Analysis of Infinite-State Domains

In order to allow the user to specify infinite-state systems, we extended the language of NuSMV with two new data types, namely Reals and unbounded Integers. This, for instance, enables to specify domains with infinite data types (like e.g. the example in Fig. 1).

To analyze such kind of designs, we integrated in nuXmv several new verification algorithms based on Satisfiability Modulo Theory (SMT) [18] and on abstraction, or combination of abstraction with other techniques.

We lifted Simple Bounded Model Checking (SBMC) [15] from the pure Boolean case to the SMT case. The encoding is the same as that of SBMC, but instead of using a SAT solver we use an SMT solver. The SBMC SMT based approach for LTL verification is comple-

```
1   MODULE main
2   IVAR
3     d : Real;
4   VAR
5     state : {s0, s1};
6     res   : Real;
7   ASSIGN
8     init(state) := s0;
9     next(state) := case
10      state = s0 & res >= 0.10 : s1;
11      state = s1 & res >= 0.20 : s0;
12      TRUE                     : state;
13    esac;
14    next(t) := case
15      state = s0 & res < 0.10 : res + d;
16      state = s1 & res < 0.20 : res + d;
17      TRUE                    : 0.0;
18    esac;
19  INIT
20    res >= 0.0
21  TRANS
22    (state = s0 -> (d >= 0 & d <= 0.01)) &
23    (state = s1 -> (d >= 0 & d <= 0.02))
24  INVARSPEC res <= 0.3;
```

Fig. 1. Example of the nuXmv language

mented with k-liveness combined with IC3 extended to the infinite-state case [19]. This approach relies on recent results on applying an IC3-based approach to the verification of infinite-state systems [20]. We remark that, although these approaches are in general incomplete, if a lazo-shaped counterexample exists, it is guaranteed to be eventually found. Moreover, for certain designs, they are able to conclude that the property hold.

As far as invariant checking is concerned, we lifted the pure Boolean approaches like BMC, k-induction, interpolation, and IC3 to the infinite-state systems case. Intuitively, we use an SMT solver in place of the SAT solver. Similarly to the finite case, we provide an SMT based implementation for the McMillan approach [5], the interpolation sequence approach [6], k-induction [7] and for algorithms based on IC3 [20,21].

nuXmv also implements several approaches based on abstraction refinement [22]. We provide new algorithms combining abstraction with BMC and k-induction [23]. The algorithms do not rely on quantifier elimination techniques to compute the abstraction, but encode the model checking problem over the abstract state space into SMT problems. The advantage, is that they avoid the possible bottleneck of abstraction computation. The very same approach has been recently lifted and tightly integrated within the IC3 framework [21], with very good results. All these techniques complement the "classical" counterexample guided (predicate) abstraction refinement (CEGAR) [22], also implemented in nuXmv. The CEGAR approach requires the computation of a quantifier-free formula that is equivalent to the abstract transition relation w.r.t. a given set of predicates. This, in turn, requires the solving of an AllSAT problem [24]. For this step, nuXmv implements different techniques: a combination of BDD and SMT [25,26], where BDDs are used as compact Boolean model enumerators within an AllSMT approach; a technique that exploits the structure of the system under verification, to partition the abstraction problem into the combination of several smaller

abstraction problems [27]. For the refinement step to discard the spurious counterexample, NUXMV implements three approaches based on the analysis of the unsatisfiable core, on the analysis of the interpolants, and on the weakest preconditions.

2.3 Miscellaneous Functionalities

NUXMV provides novel functionalities that aim at facilitating the modeling and the understanding of complex designs. For instance, it allows for the generation of an explicit state representation (subject to the projection over a set of user specified predicates) in XMI format of the design under verification. The generated XMI can be visualized in any UML based viewer supporting the import from XMI.

LTL and invariant properties have been extended to allow for the use of input signals and next values of state variables. This does not add any expressive power to the language, but facilitates the writing of properties from the user's point of view. Internally, each state formula containing a reference to an input or next signal is replaced with a corresponding monitor allowing for the reuse of off-the-shelf verification engines.

NUXMV also provides several model transformation techniques aiming to reduce the state space of the design. It uses static analysis techniques to extract possible values for variables, and then re-encode the design using such information (e.g. using a word at 32 bit to store 2 values can be re-encoded with just one Boolean variable). These techniques are complemented with others aiming at simplifying the model through constants and free inputs propagation [28].

Finally, in NUXMV we removed the NUSMV limitation that restricted the support to bit vectors with less than 64 bits only.

3 Architecture

NUXMV extends the NUSMV [1] architecture as described here after. NUXMV shares with NUSMV all the basic functionalities, e.g. the symbol table, the flattening of the design, the Boolean encoding of scalar variables, the representation of the finite-state machines at the different abstraction levels (e.g., scalar, BDD). Moreover, it inherits from NUSMV all the basic model checking algorithms for finite domains both using BDDs (using the CUDD [2]) and SAT (e.g. MINISAT [3]). To implement the new functionalities, we added new Boolean reasoning engines. We extended MINISAT with the construction of the resolution proof. On top of this, we built an interface to extract interpolants. In this respect, we extended the standard API in NUSMV to also provide API for extracting and manipulating interpolants. This also enables for the use of different SAT engines in a transparent way.

For IC3 based algorithms, NUXMV provides two modes: execution as a library, or call of an external executable (this was also done to participate to the hardware model checking competition). The use of an external executable also opens to experiment with other engines, and to reuse the results within NUXMV, provided the I/O interface is respected. The model checking problem is dumped into AIGER format, and for violated properties the resulting AIGER trace is converted back into a NUXMV trace.

To reason over infinite-state systems, we created an interface towards SMT engines. We instantiated this interface on the MATHSAT5 [29] SMT solver (although in
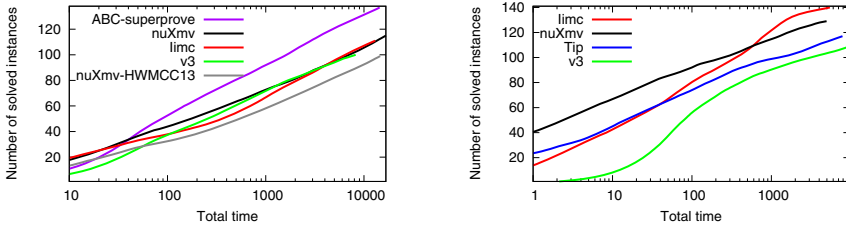
Fig. 2. Results for finite domains on the HWMCC13 samples

principle other SMT solvers could be plugged in). The interface enables for a wide range of queries: satisfiability checking and AllSMT queries; extraction of unsatisfiable cores and of interpolants; and, quantifier elimination. This interface is the basis on top of which the majority of the SMT based algorithms are built. For IC3 based algorithms, a more tight integration with the SMT solver is required, therefore we implemented them directly on top of MATHSAT5. We then interfaced this new engine with NUXMV.

NUXMV has been developed in C and in C++. It compiles and executes on the most widely used Operating Systems (OSs) and architectures; namely, Linux, MS Windows, and MacOS X. Porting to other OSs is also possible (although not tested yet).

4   Performance Evaluation

In order to see where NUXMV is positioned w.r.t. the state-of-the-art, we report in this section some results. In Fig. 2 we plot the comparison of NUXMV against the three best performers at the HWMCC13. The comparison is run on the same benchmarks used in the HWMCC13 in the single safety (left) and in the liveness (right) tracks. We compare two versions of NUXMV, the one that participated in the competition (NUXMV-HWMCC13) and the current one. The results show that, still ABC is the top performer, but the current version of NUXMV is able to solve more problems that the one submitted to the competition. Moreover, it solves more problems than iimc and v3 that are performing better than NUXMV-HWMCC13. Concerning the liveness track, NUXMV did not participate in the competition. Here iimc is still the winner, but NUXMV performs better than v3 and tip (positioned 2nd and 3rd resp. in the HWMCC13).

In Fig. 3 we report some results for the SMT cases. On the left we compare IC3 with implicit abstraction algorithm [21] as available in NUXMV against state-of-the-art engines on a set of bit-vector (BV) benchmarks (we took all the benchmarks used in [20], using BV as background theory instead of LRA, the instances of the bitvector set of the Software Verification Competition [30] and the instances from the test suite of InvGen [31]). For the other solvers, bit-blasting was performed. In Fig. 3 on the right, we compare NUXMV with other model checkers for Linear Integer Arithmetic (LIA). (The label nuXmv-IC3(IA) corresponds to the label IC3+IA(BV) of [21]). We used benchmarks taken from Lustre programs as available from the web page of Kind [32]. These results show that in both cases NUXMV can solve more problems than the other state-of-the-art tools.

These results, clearly show that NUXMV is well positioned in the space of formal verification engines, both for the finite domain case, and for the infinite domain one.
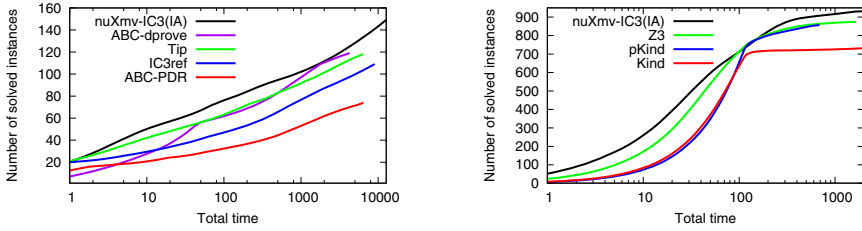
Fig. 3. Results on SMT domains: BV and LIA (taken from [21]

5   Applications

The NUXMV symbolic model checker has been used in a wide range of applications, both at academic and at industrial level.

As far as the industrial settings are concerned, we report that NUXMV is used at Rockwell-Collins as one of the available back-end verification engines [33]. NUXMV has been extended and is daily used by Ales s.r.l., also as a back-end verification engine, for their internal verification flow [34]. Moreover, Ansaldo STS integrated NUXMV within their development environment for the verification of railways interlocking software [35]. We also remark that NUXMV has been widely used in several industrial projects with the European Space Agency (ESA). For instance, it is the back-end of the COMPASS tool [36], developed within the COMPASS [37] and AUTOGEF [38] projects both funded by ESA. It has also been used in the EuRailCheck tool for the validation of a fragment of the ETCS requirements [39]. EuRailCheck was developed in response of an invitation to tender issued by the European Railway Agency.

NUXMV is also the back-end of several other tools. (We remark that, since the development of NUXMV started a long time ago, its functionalities were used already by other tools, often relying on intermediate, non-official versions of NUXMV itself.) It has been integrated in AutoFocus [40]. It is used in the KRATOS [41] software model checker, in RATSY [42] for temporal logic synthesis, and in OCRA [43] for contract based requirements analysis. Finally, it is the basis on top of which we built the safety assessment tool FSAP [44] and the HyCOMP [45] tool for the verification of hybrid systems.

6   Conclusions and Future Work

In this paper we presented NUXMV, a new symbolic model checker for finite- and infinite-state transition systems. We described its functionalities, and we reported some results that compare its performance with the state-of-the-art. The results show that it is well positioned w.r.t. the other possible competitors.

As future work, we would like to add support for arrays, their combination with bit vectors, and uninterpreted functions, and to interface with the BTOR format. We would like to exploit multi-core architecures to further speed-up the analyses. Finally, we would like to wrap the NUXMV functionalities in a scripting language to facilitate the experimentation of new algorithms and the customization w.r.t. user needs.

References

1. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
2. Somenzi, F.: CUDD: Colorado University Decision Diagram package — release 2.4.1
3. Eén, N., Sörensson, N.: An extensible sat-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
4. Biere, A., Heljanko, K., Wieringa, S.: AIGER (2011), http://fmv.jku.at/aiger/
5. McMillan, K.L.: Interpolation and sat-based model checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)
6. Vizel, Y., Grumberg, O.: Interpolation-sequence based model checking. In: FMCAD, pp. 1–8. IEEE (2009)
7. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a sat-solver. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 108–125. Springer, Heidelberg (2000)
8. Bradley, A.R.: Sat-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 70–87. Springer, Heidelberg (2011)
9. Hassan, Z., Bradley, A.R., Somenzi, F.: Better generalization in ic3. In: FMCAD, pp. 157–164. IEEE (2013)
10. Vizel, Y., Grumberg, O., Shoham, S.: Lazy abstraction and sat-based reachability in hardware model checking. In: Cabodi, G., Singh, S. (eds.) FMCAD, pp. 173–181. IEEE (2012)
11. Case, M.L., Mony, H., Baumgartner, J., Kanzelman, R.: Enhanced verification by temporal decomposition. In: FMCAD, pp. 17–24. IEEE (2009)
12. Thomas, D., Chakraborty, S., Pandya, P.K.: Efficient guided symbolic reachability using reachability expressions. STTT 10(2), 113–129 (2008)
13. Eisner, C., Fisman, D.: A Practical Introduction to PSL. Series on Integrated Circuits and Systems. Springer-Verlag New York, Inc., Secaucus (2006)
14. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without bdds. In: Cleaveland, W.R. (ed.) TACAS/ETAPS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
15. Biere, A., Heljanko, K., Junttila, T.A., Latvala, T., Schuppan, V.: Linear encodings of bounded ltl model checking. Logical Methods in Computer Science 2(5) (2006)
16. Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In: Cabodi, G., Singh, S. (eds.) FMCAD, pp. 52–59. IEEE (2012)
17. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another Look at LTL Model Checking. Formal Methods in System Design 10(1), 47–71 (1997)
18. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, pp. 825–885. IOS Press (2009)
19. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Verifying ltl properties of hybrid systems with k-liveness. Technical report. Under review (2014)
20. Cimatti, A., Griggio, A.: Software model checking via ic3. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 277–293. Springer, Heidelberg (2012)
21. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Ic3 modulo theories via implicit predicate abstraction. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 46–61. Springer, Heidelberg (2014)
22. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
23. Tonetta, S.: Abstract model checking without computing the abstraction. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 89–105. Springer, Heidelberg (2009)

24. Lahiri, S.K., Nieuwenhuis, R., Oliveras, A.: SMT Techniques for Fast Predicate Abstraction. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 424–437. Springer, Heidelberg (2006)

25. Cavada, R., Cimatti, A., Franzén, A., Kalyanasundaram, K., Roveri, M., Shyamasundar, R.K.: Computing Predicate Abstractions by Integrating BDDs and SMT Solvers. In: FMCAD, pp. 69–76. IEEE Computer Society (2007)

26. Cimatti, A., Franzén, A., Griggio, A., Kalyanasundaram, K., Roveri, M.: Tighter integration of BDDs and SMT for Predicate Abstraction. In: DATE, pp. 1707–1712. IEEE (2010)

27. Cimatti, A., Dubrovin, J., Junttila, T.A., Roveri, M.: Structure-aware computation of predicate abstraction. In: FMCAD, pp. 9–16. IEEE (2009)

28. Armoni, R., Fix, L., Fraer, R., Heyman, T., Vardi, M.Y., Vizel, Y., Zbar, Y.: Deeper Bound in BMC by Combining Constant Propagation and Abstraction. In: ASP-DAC, pp. 304–309. IEEE (2007)

29. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013)

30. Beyer, D.: Second Competition on Software Verification - (Summary of SV-COMP 2013. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 594–609. Springer, Heidelberg (2013)

31. Gupta, A., Rybalchenko, A.: InvGen: An efficient invariant generator. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 634–640. Springer, Heidelberg (2009)

32. Hagen, G., Tinelli, C.: Scaling up the formal verification of lustre programs with smt-based techniques. In: Cimatti, A., Jones, R.B. (eds.) FMCAD, pp. 1–9. IEEE (2008)

33. Miller, S.P., Whalen, M.W., Cofer, D.D.: Software model checking takes off. Commun. ACM 53(2), 58–64 (2010)

34. Ferrante, O., Benvenuti, L., Mangeruca, L., Sofronis, C., Ferrari, A.: Parallel NuSMV: A NuSMV Extension for the Verification of Complex Embedded Systems. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP Workshops 2012. LNCS, vol. 7613, pp. 409–416. Springer, Heidelberg (2012)

35. Cimatti, A., Corvino, R., Lazzaro, A., Narasamdya, I., Rizzo, T., Roveri, M., Sanseviero, A., Tchaltsev, A.: Formal Verification and Validation of ERTMS Industrial Railway Train Spacing System. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 378–393. Springer, Heidelberg (2012)

36. Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V.Y., Noll, T., Roveri, M., Wimmer, R.: A Model Checker for AADL. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 562–565. Springer, Heidelberg (2010)

37. Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V.Y., Noll, T., Roveri, M.: The compass approach: Correctness, modelling and performability of aerospace systems. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) SAFECOMP 2009. LNCS, vol. 5775, pp. 173–186. Springer, Heidelberg (2009)

38. Alaña, E., Naranjo, H., Yushtein, Y., Bozzano, M., Cimatti, A., Gario, M., de Ferluc, E., Garcia, G.: Automated generation of FDIR for the compass integrated toolset (AUTOGEF). DASIA 2012 (2012)

39. Chiappini, A., Cimatti, A., Macchi, L., Rebollo, O., Roveri, M., Susi, A., Tonetta, S., Vittorini, B.: Formalization and validation of a subset of the european train control system. In: Kramer, J., Bishop, J., Devanbu, P.T., Uchitel, S. (eds.) ICSE (2), pp. 109–118. ACM (2010)

40. Autofocus-Team: The AutoFOCUS tool, https://af3.fortiss.org

41. Cimatti, A., Griggio, A., Micheli, A., Narasamdya, I., Roveri, M.: Kratos - A Software Model Checker for SystemC. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 310–316. Springer, Heidelberg (2011)

42. Bloem, R., Cimatti, A., Greimel, K., Hofferek, G., Könighofer, R., Roveri, M., Schuppan, V., Seeber, R.: RATSY - A New Requirements Analysis Tool with Synthesis. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 425–429. Springer, Heidelberg (2010)
43. Cimatti, A., Dorigatti, M., Tonetta, S.: OCRA: A tool for checking the refinement of temporal contracts. In: ASE, pp. 702–705. IEEE (2013)
44. Bozzano, M., Villafiorita, A.: The FSAP/NuSMV-SA Safety Analysis Platform. STTT 9(1), 5–24 (2007)
45. Cimatti, A., Mover, S., Tonetta, S.: SMT-Based Verification of Hybrid Systems. In: Hoffmann, J., Selman, B. (eds.) AAAI. AAAI Press (2012)