Implementation and Evaluation of BVZot Presented in "Efficient Scalable Verification of LTL Specifications"

Mohammad Mehdi Pourhashem Kallehbasti
Politecnico di Milano – Dipartimento di Elettronica, Informazione e Bioingegneria
Via Golgi 42 – 20133 Milano, Italy
Email: mohammadmehdi.pourhashem@polimi.it

0.1 Implementation

The bit-vector-based encoding has been implemented as a plugin in the $\mathbb{Z}$ot [1] tool, called ***bvzot***.

$\mathbb{Z}$ot is an extensible Bounded Model/Satisfiability Checker written in Common Lisp. More precisely, $\mathbb{Z}$ot is capable of performing bounded satisfiability checking of formulae written both in LTL (with past operators) and in the propositional, discrete-time fragment of the metric temporal logic TRIO [2], which is equivalent to LTL, but more concise. In fact, TRIO formulae can straightforwardly be translated into LTL formulae, so we use the two temporal logics interchangeably.

The verification process in $\mathbb{Z}$ot goes through the following steps: (i) the user writes the specification to be checked as a set of temporal logic formulae (these formulae could also be produced automatically as in [3]), and selects the plugin and the time bound (i.e., the value of bound k) to be used to perform the verification; (ii) depending on the input temporal logic (TRIO or LTL) and the selected plugin, $\mathbb{Z}$ot encodes the received specification in a target logic (e.g., propositional logic, or bit-vector logic); (iii) $\mathbb{Z}$ot feeds the encoded specification to a solver that is capable of handling the target logic; (iv) the result obtained by the solver is parsed back and presented to the user in a textual representation.

$\mathbb{Z}$ot supports two kinds of solvers: SAT solvers (e.g., MiniSat [4]) for propositional logic, and SMT solvers, such as Z3 [5], for bit-vector logic and decidable fragments of first-order logic.

To evaluate the bit-vector-based encoding we compared it against three other encodings available in the literature: the classic bounded encoding presented in [6]; the optimized encoding presented in [7], which has been further improved in [8] and made incremental in [9]; and the encoding optimized for metric temporal logic presented in [10]. The first two encodings are implemented in the well-known NuSMV model checker [11] (in fact, NuSMV implements an optimized, incremental version of the classic encoding of [6]), whereas the third is implemented in the *meezot* plugin of the $\mathbb{Z}$ot tool.

We label the experiments carried out with the classic encoding implemented in NuSMV as *bmc*, those performed with the optimized encoding of [8] as *smbc*, those with the incremental version of *sbmc* presented in [9] as *sbmc_ inc*[1], and those performed with the metric encoding implemented in $\mathbb{Z}$ot as *meezot* (all these labels come from the commands used in NuSMV and $\mathbb{Z}$ot to select the encodings). Note that both NuSMV and $\mathbb{Z}$ot support other encodings for LTL/TRIO; we have chosen those mentioned above because further experiments, not reported here, have shown them to be, on average, the most efficient ones for the two tools.

To test the relative efficiency of the four encodings, we applied them to the verification of three case studies, two from the literature and one from previous work of ours. The case studies were chosen mostly for their complexity to

[1]While conductiong the *sbmc_ inc* experiments we did not activate the completeness checking option since it often slows the verification down, as shown in [9].

highlight the relative strengths and weaknesses of each tool. These three case studies employ a BSC approach, that is, they use temporal logic to describe both the system under verification and the properties to be checked. In all three cases we performed two kinds of checks. First, we took the temporal logic formula ϕ_S describing the system, and we simply checked for its satisfiability. This allowed us to determine whether the specification is realizable or not. As a second type of check, we also provided a logic formula ϕ_P capturing the property that the system should have satisfied, and we fed the BSC algorithm with formula $\phi_S \wedge \neg\phi_P$ to determine whether the property holds for the system or not. We also experimented with different bounds k to analyze how the tools behave when k is increased.

0.2 Evaluation

We briefly introduce the three case studies; interested readers can refer to the cited literature for their details.

Kernel Railway Crossing (KRC). The KRC problem is frequently used for comparing real-time notations and tools [12]. A railway crossing system prevents crossing of the railway by vehicles during passage of a train, by controlling a gate. A temporal logic-based version of the KRC was developed in [10] for benchmarking purposes. It describes one track and one direction of movement of the trains, and it considers an interlocking system. We experimented with two sets of time constants that allow different degrees of nondeterminism, hereafter denoted as `krc1` and `krc2`. The level of nondeterminism is increased by using bigger time constants, e.g., the time for a train to go through the railway crossing, which increase the number of possible combinations of events in the system. We also carried out formal verification with two properties of interest: a safety property that says that as long as a train is in the critical region the gate is closed (`P1`); and a utility property that states that the gate must be open when it is safe to do so (i.e., the gate should not be closed when unnecessary), where the notion of "safe" is captured through suitable time constants (`P2`).

Fischer's Protocol. This is a classic algorithm for granting exclusive access to a resource that is shared among many processes. Fischer's protocol is a typical benchmark for verification tools capable of dealing with real-time constraints. The version we used for our tests, where the specification of the system is described through temporal logic formulae, is taken from [10]; it includes 4 processes, and the delay that a process waits after sending a request, which is the key parameter in Fischer's protocol, is 5 time instants. We performed formal verification of a safety property that states that it is never the case that two processes are simultaneously in their critical sections (`P1`).

Verification of UML Diagrams. *Corretto*[2] is the toolset we developed to perform formal verification of UML models [3]. *Corretto* takes as input a set of UML diagrams and produces their formal representation through formulae of temporal logic. In our tests we used the example diagrams introduced in [3],

[2] `https://github.com/deib-polimi/corretto`

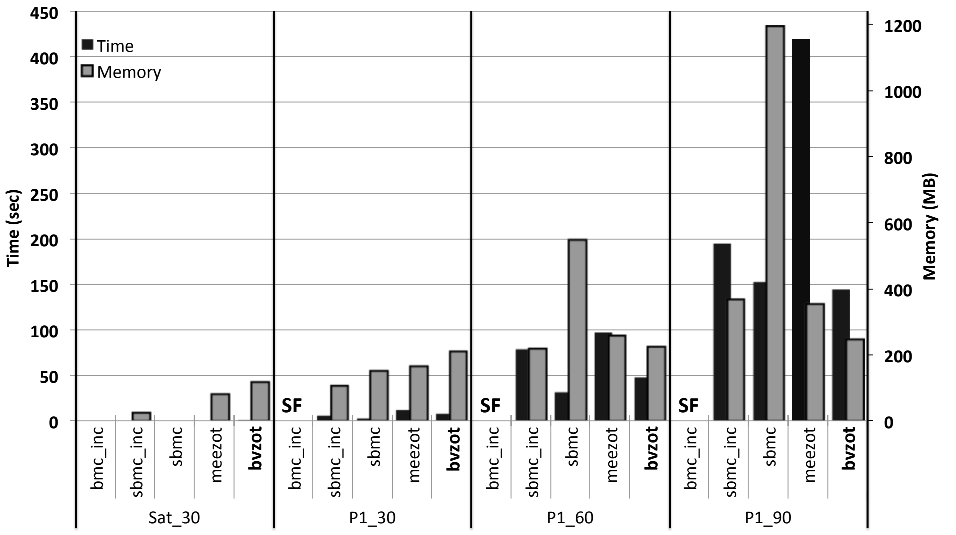

Figure 1: Time/Memory Comparison for KRC1 (SAT and P1).

which describe the behavior of an application that pings two servers, and then sends queries to the server that responds first. The model comprises a loop, and we performed tests on two versions of the system, called `sdserverl2`, and `sdserverl3`, where the number of iterations in the loops is 2 and 3, respectively. We also performed formal verification on the example system using property `P1` defined in [3], to which we refer the reader for further details on this case study.

To compare the performances of the *bmc*, *sbmc*, *sbmc_ inc*, *meezot* and *bvzot* encodings, we built a simple translation tool that converts specifications written in the ℤot input language such as those used in [10] and [3] into the input language of NuSMV.

Figures 1-7 and tables 1-4 show the time (in seconds) and memory (in MBs) consumed in each of the experiments we performed. Note that if no bar is visible, and no error tag is reported, this means that the number is very small[3]

For example, Figure 1 shows the time/memory consumption for each encoding (*bmc*, *sbmc*, *sbmc_ inc*, *meezot*, and *bvzot*) in the various checks on example `krc1`: simple satisfiability checking with maximum bound $k = 30$ (*sat_ 30*), verification of property `P1` with maximum bound $k = 30$ (*P1_ 30*), $k = 60$ (*P1_ 60*) and $k = 90$ (*P1_ 90*), respectively. The role of the "maximum bound" is the following: for a given maximum bound k, the tools iteratively (possibly incrementally) try to find an ultimately periodic model $\alpha\beta^\omega$ where the length of $\alpha\beta$ is $1, 2, \ldots, k$. As soon as a model is found, the search stops, and the model is output; if no model is found for any bound up to k, the search stops at k and the formula is declared unsatisfiable.

[3]Interested readers can refer to `http://home.deib.polimi.it/pourhashem.kallehbasti/icse-2015.php` for the complete and detailed data about the experiments.

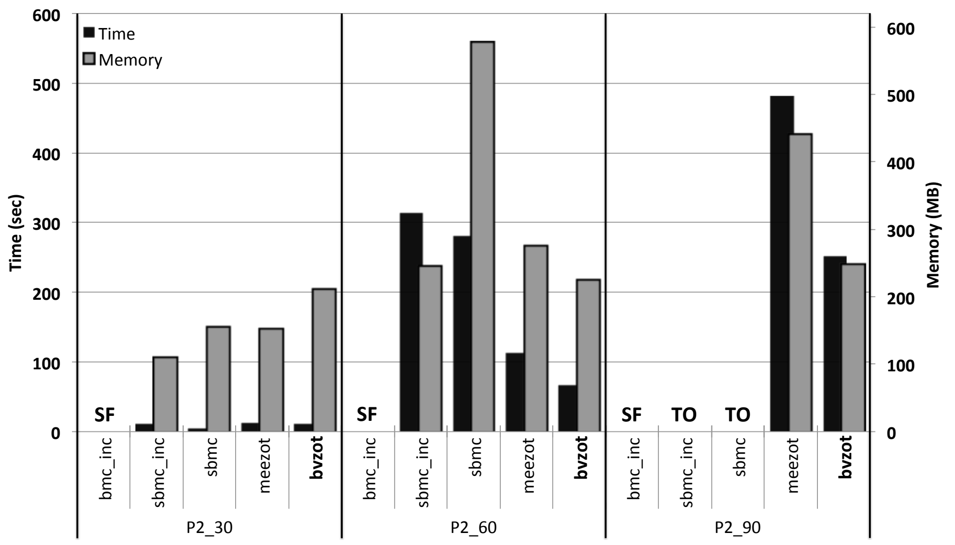

Figure 2: Time/Memory Comparison for KRC1 (P2).

All the runs reported in Figures 1-7 had a time limit of 1 hour and a memory limit of 4GB RAM; that is, if the verification took longer than 1 hour or occupied more than 4GB of RAM, it was stopped. Hence, the possible outcomes of a run are **satisfiable**, **unsatisfiable**, **out of time** (**TO**), and **out of memory** (**MO**). In addition, in some cases the tool stopped with a **segmentation fault** (**SF**) error, and in others with **heap exhausted** (**HE**) while pre-processing the specification to produce the encoding.

All the experiments were carried out on a Linux desktop machine with a 3.4 GHz Intel® Core™ i7-4770 CPU and 8 GB RAM[4]. The NuSMV version was 2.5.4. The SAT and SMT solvers used with $\mathbb{Z}$ot were, respectively, MiniSat version 2.2 and Z3 version 4.3.2.

As the figures show, *sbmc_inc* is the most memory efficient and *sbmc* is the fastest algorithm implemented in NuSMV. There are six models that *sbmc_inc* can afford to verify, while *sbmc* fails. However, for the models affordable for both algorithms, *sbmc* is faster than *sbmc_inc*.

When performing checks that require small bounds, such as the satisfiability checks, *bvzot* is only occasionally more efficient than the other tools. However, as the models and bounds[5] grow in size, *bvzot* demonstrates its strengths. For example, when proving properties for the KRC version with the highest level of nondeterminism, i.e., `krc2`, *bvzot* is the only tool able to explore all the

[4] *bvzot*, along with the code for all the experiments, is available at `http://home.deib.polimi.it/pourhashem.kallehbasti/icse-2015.html`

[5] For each experiment, the length of the smallest model found was the same for each tool: 1 for the `krc` examples (the execution in which nothing happens, that is, no train enters the crossing, is admissible), 29 for Fischer's protocol, and 10 for `sdserverl2` and 13 for `sdserverl3`.

Table 1: $\frac{Time(s)}{Memory(MB)}$ Comparison for KRC1.

Tool / Model	bmc_inc	sbmc_inc	sbmc	meezot	bvzot
Sat_30	$\frac{\mathbf{0}}{\mathbf{0}}$	$\frac{\mathbf{0}}{\mathbf{0}}$	$\frac{\mathbf{0}}{\mathbf{0}}$	$\frac{\mathbf{0}}{83}$	$\frac{1}{117}$
P1_30	SF	$\frac{6}{\mathbf{107}}$	$\frac{\mathbf{3}}{152}$	$\frac{12}{165}$	$\frac{8}{212}$
P1_60	SF	$\frac{79}{\mathbf{219}}$	$\frac{\mathbf{32}}{546}$	$\frac{97}{258}$	$\frac{48}{225}$
P1_90	SF	$\frac{194}{368}$	$\frac{152}{1193}$	$\frac{419}{353}$	$\frac{\mathbf{144}}{\mathbf{247}}$
P2_30	SF	$\frac{11}{\mathbf{112}}$	$\frac{\mathbf{5}}{157}$	$\frac{13}{154}$	$\frac{11}{212}$
P2_60	SF	$\frac{314}{246}$	$\frac{281}{578}$	$\frac{113}{276}$	$\frac{\mathbf{67}}{\mathbf{226}}$
P2_90	SF	TO	TO	$\frac{481}{442}$	$\frac{\mathbf{252}}{\mathbf{249}}$

bounds up to 90, and it is faster than the others when the time bound is kept smaller (30 or 60). Similar results hold for the verification of properties on the UML diagrams, whose formalization in temporal logic is in fact the biggest specification that we have tested due to the necessity of capturing all the possible sequences of events in the Sequence Diagram.

However, we must highlight that in the case of Fischer's protocol, *sbmc* is the most efficient encoding time-wise, whereas *bvzot* is often the one with the least memory consumption.

All in all, we can conclude that the experimental results show a promising ability by *bvzot* to scale up as the size of the specification and of the time bound increase. Further gains could also be obtained by adapting some of the optimizations presented in [8] in *bvzot*.

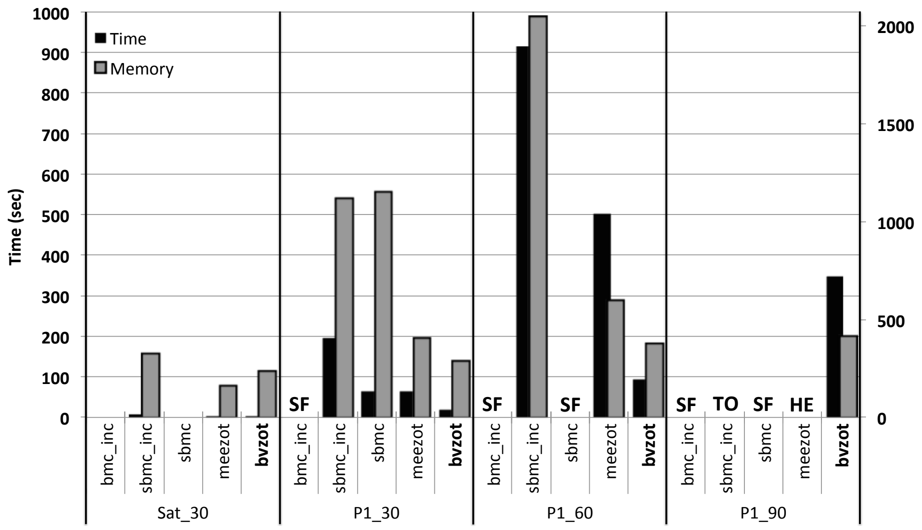

Figure 3: Time/Memory Comparison for KRC2 (SAT and P1).

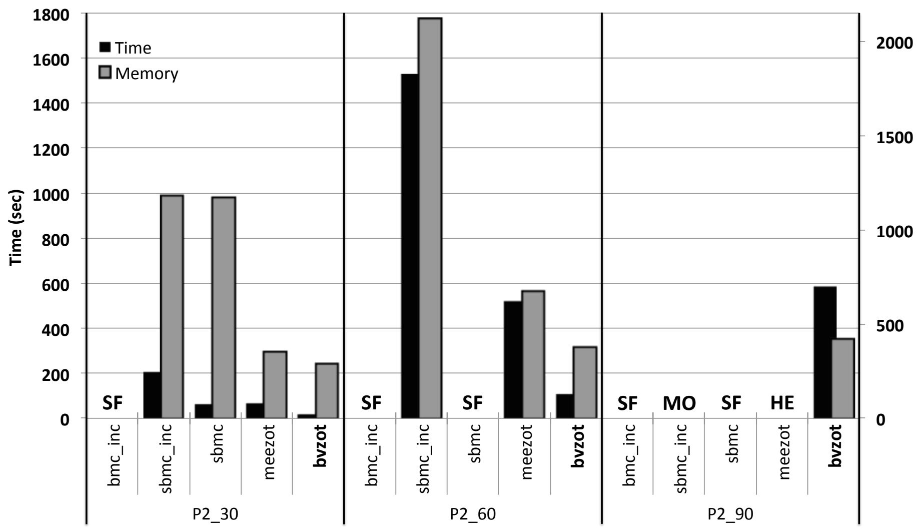

Figure 4: Time/Memory Comparison for KRC2 (P2).

Table 2: $\frac{Time(s)}{Memory(MB)}$ Comparison for KRC2.

Model \ Tool	bmc_inc	sbmc_inc	sbmc	meezot	bvzot
Sat_30	$\frac{\mathbf{0}}{\mathbf{0}}$	$\frac{10}{329}$	$\frac{\mathbf{0}}{\mathbf{0}}$	$\frac{5}{168}$	$\frac{5}{242}$
P1_30	SF	$\frac{198}{1122}$	$\frac{67}{1152}$	$\frac{67}{410}$	$\frac{\mathbf{22}}{\mathbf{292}}$
P1_60	SF	$\frac{914}{2045}$	SF	$\frac{502}{602}$	$\frac{\mathbf{97}}{\mathbf{382}}$
P1_90	SF	TO	SF	HE	$\frac{\mathbf{349}}{\mathbf{417}}$
P2_30	SF	$\frac{208}{1184}$	$\frac{68}{1173}$	$\frac{72}{359}$	$\frac{\mathbf{22}}{\mathbf{295}}$
P2_60	SF	$\frac{1527}{2123}$	SF	$\frac{521}{679}$	$\frac{\mathbf{113}}{\mathbf{382}}$
P2_90	SF	TO	SF	HE	$\frac{\mathbf{589}}{\mathbf{427}}$

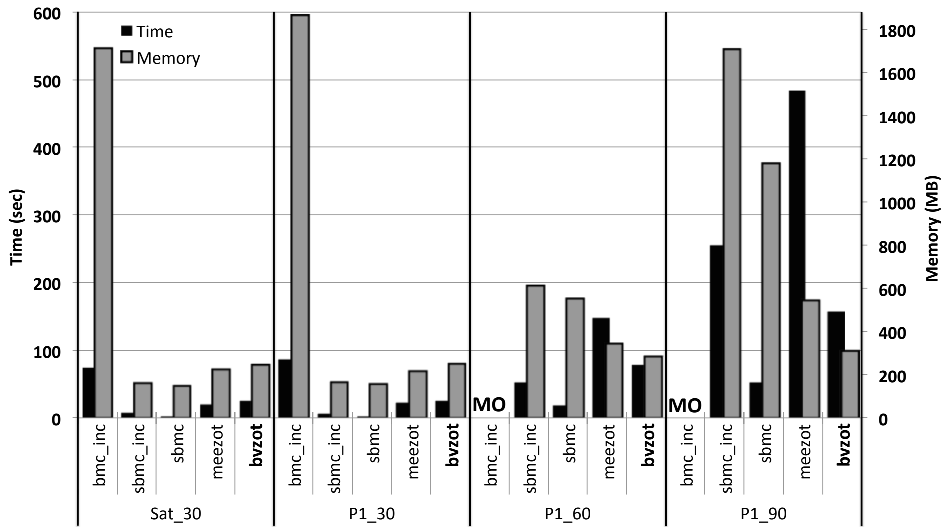

Figure 5: Time/Memory Comparison for Fischer (SAT and P1).

Table 3: $\frac{Time(s)}{Memory(MB)}$ Comparison for Fischer.

Model \ Tool	bmc_inc	sbmc_inc	sbmc	meezot	bvzot
Sat_30	$\frac{75}{1712}$	$\frac{9}{164}$	$\frac{\mathbf{3}}{\mathbf{151}}$	$\frac{21}{227}$	$\frac{27}{250}$
P1_30	$\frac{68}{1864}$	$\frac{7}{168}$	$\frac{\mathbf{3}}{\mathbf{158}}$	$\frac{24}{218}$	$\frac{26}{251}$
P1_60	MO	$\frac{54}{613}$	$\frac{\mathbf{20}}{554}$	$\frac{149}{347}$	$\frac{79}{\mathbf{286}}$
P1_90	MO	$\frac{255}{1706}$	$\frac{\mathbf{53}}{1178}$	$\frac{484}{546}$	$\frac{158}{\mathbf{311}}$

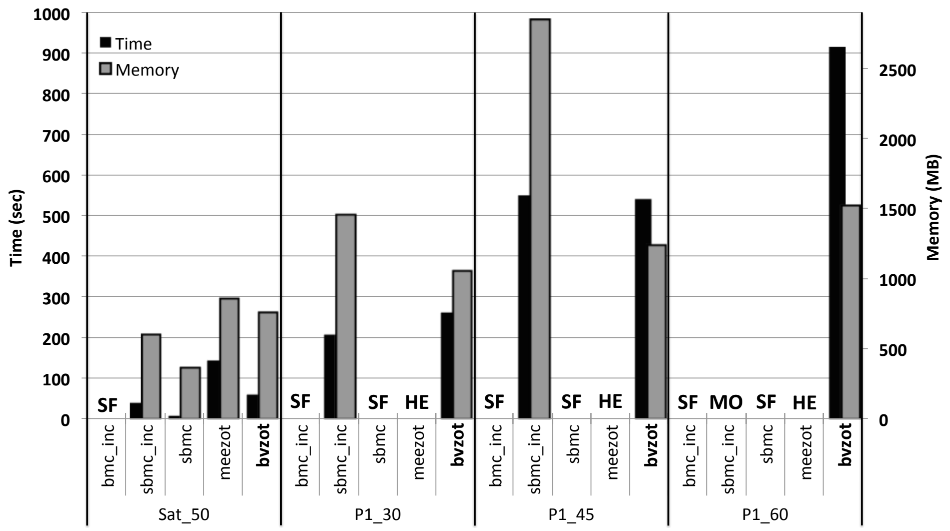

Figure 6: Time/Memory Comparison for SDServerl2 (SAT and P1).

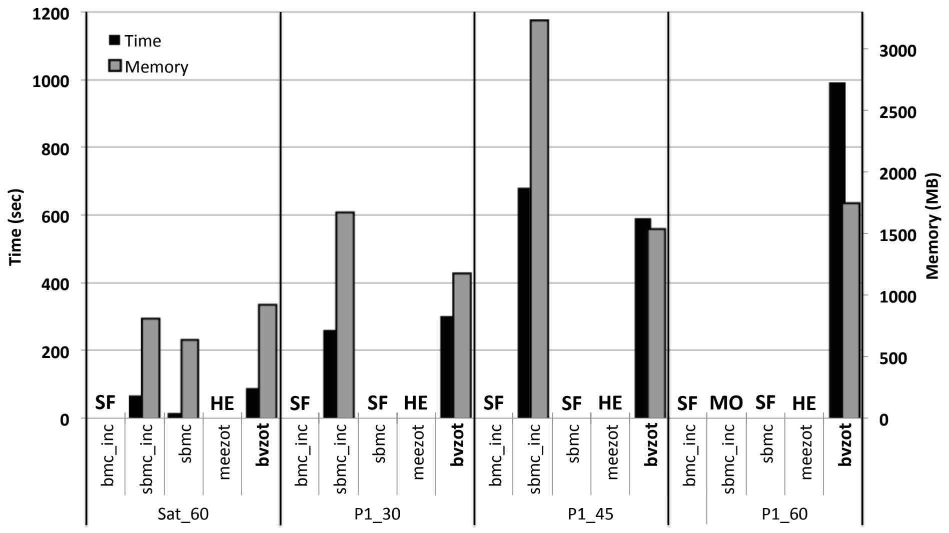

Figure 7: Time/Memory Comparison for SDServerl3 (SAT and P1).

Table 4: $\frac{Time(s)}{Memory(MB)}$ Comparison for SDServer{l2 , l3}.

Model \ Tool	bmc_inc	sbmc_inc	sbmc	meezot	bvzot
L2_Sat_50	SF	$\frac{41}{606}$	$\frac{\mathbf{9}}{\mathbf{37}}$	$\frac{146}{861}$	$\frac{62}{760}$
L2_P1_30	SF	$\frac{\mathbf{208}}{1458}$	SF	HE	$\frac{262}{\mathbf{1056}}$
L2_P1_45	SF	$\frac{549}{2847}$	SF	HE	$\frac{\mathbf{540}}{\mathbf{1239}}$
L2_P1_60	SF	MO	SF	HE	$\frac{\mathbf{913}}{\mathbf{1520}}$
L3_Sat_50	SF	$\frac{71}{815}$	$\frac{\mathbf{19}}{\mathbf{645}}$	HE	$\frac{93}{929}$
L3_P1_30	SF	$\frac{263}{1672}$	SF	HE	$\frac{\mathbf{304}}{\mathbf{1182}}$
L3_P1_45	SF	$\frac{681}{3228}$	SF	HE	$\frac{\mathbf{593}}{\mathbf{1536}}$
L3_P1_60	SF	MO	SF	HE	$\frac{\mathbf{990}}{\mathbf{1744}}$

Bibliography

[1] "The Zot bounded model/satisfiability checker," http://zot.googlecode.com.

[2] C. Ghezzi, D. Mandrioli, and A. Morzenti, "TRIO: A Logic Language for Executable Specifications of Real-time Systems ," *Journal of Systems and Software*, vol. 12, no. 2, pp. 107 – 123, 1990.

[3] L. Baresi, M. M. Pourhashem Kallehbasti, and M. Rossi, "Flexible Modular Formalization of UML Sequence Diagrams," in *Proc. of the 2nd FME Workshop on Formal Methods in Software Engineering*, ser. FormaliSE 2014, 2014, pp. 10–16.

[4] N. Een and N. Sorensson, "An extensible sat-solver," in *Proceedings of the 6th International Conference on Theory and Application of Satisfiability Testing (SAT'03)*, ser. Lecture Notes in Computer Science, 2003, no. 2919, p. 502–518.

[5] Microsoft Research, "Z3: An efficient SMT solver," http://research.microsoft.com/en-us/um/redmond/projects/z3/.

[6] A. Biere, K. Heljanko, T. A. Junttila, T. Latvala, and V. Schuppan, "Linear encodings of bounded LTL model checking," *Log. Meth. in Computer Science*, vol. 2, no. 5, pp. 1–64, 2006.

[7] T. Latvala, A. Biere, K. Heljanko, and T. Junttila, "Simple bounded LTL model checking," in *Formal Methods in Computer-Aided Design*, ser. Lecture Notes in Computer Science, 2004, vol. 3312, pp. 186–200.

[8] ——, "Simple is better: Efficient bounded model checking for past LTL," in *Verification, Model Checking, and Abstract Interpretation*, ser. Lecture Notes in Computer Science, 2005, vol. 3385, pp. 380–395.

[9] K. Heljanko, T. Junttila, and T. Latvala, "Incremental and complete bounded model checking for full PLTL," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, 2005, vol. 3576, pp. 98–111.

[10] M. Pradella, A. Morzenti, and P. San Pietro, "Bounded Satisfiability Checking of Metric Temporal Logic Specifications," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 3, pp. 20:1–20:54, 2013.

[11] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," in *Computer Aided Verification*, ser. LNCS, 2002, vol. 2404, pp. 359–364.

[12] C. Heitmeyer and D. Mandrioli, *Formal Methods for Real-Time Computing.* New York, NY, USA: John Wiley & Sons, Inc., 1996.