A Model Checker for AADL
Experimental Results

Marco Bozzano[2], Alessandro Cimatti[2], Joost-Pieter Katoen[1],
Viet Yen Nguyen[1], Thomas Noll[1], Marco Roveri[2], and Ralf Wimmer[3]

[1] RWTH Aachen University, Germany
[2] Fondazione Bruno Kessler, Italy
[3] Albert-Ludwigs-University Freiburg, Germany

1 Introduction

This document contains some experimental data for the COMPASS toolset, described in the paper "A Model Checker for AADL".

2 Description of the Experiments

The following tests have been carried out, and the results analyzed:

- BDD-based model checking
- SAT-based model checking
- fault tree generation
- performability evaluation
- dynamic fault tree verification

The tests have been chosen as representatives of the different analysis tasks and techniques (qualitative and quantitative, BDD versus SAT) available in the COMPASS toolset.

3 Models

The following models have been used to run the analyses (all the models are parametric):

Adder A parametric version of the adder example in the COMPASS toolset distribution. The adder has one layer with N generators and N bit modules (where N is a parameter) and successive layers with $N/2$, $N/4$, ... 1 adders. The following properties have been tested. For model checking, no faults were injected. Four properties were verified. The first and second property state that the inputs being all equal implies that the output of the outermost adder is, respectively, zero or one; these properties may be true or false depending on the number of input bits. The third and fourth property state that if it is not the case that the inputs are all equal, then the output of the outermost

adder is, respectively, zero or one; these properties are always false. For fault tree analysis, faults of type inverted for the bit modules were injected. The following propositional property was used as top level event: all inputs are equal and at least one bit module in the innermost layer has output one. The generated fault trees are not empty (faults are needed to trigger the top level event).

SensorFilter A parametric version of the sensorfilter example in the COMPASS toolset distribution. For performability evaluation, the number N indicates the degree of redundancy, so for example for $N = 4$, it means the example describes four redundant sensors and four redundant filters. An injection is made on each sensor's output, by setting it to 15 when the Dead state is entered. A second injection is made on each filter's output, by setting it to 0 when the Dead state is entered. On each parameterised example, we tested the property that computes the probability that the last sensor dies, i.e., all sensors have died. For dynamic fault tree evaluation, N indicates the amount of cut sets in the fault tree. An increasing number indicates a bigger fault tree. On each parameterised example, we computed the probability that the top-level event is triggered.

4 Experimental Results

We present the corresponding analysis results, respectively, in Tables 1, 2, 3, 4 and 5. In the tables, the first column show the name of the model, the second column its complexity[4], and the third column the time needed to complete the test (in seconds; note that 24 hours corresponds to 86400 seconds).

The timeout and memory limits were set to, respectively, 24 hours and 3 GB.

In the first two tables, for brevity, only some results are presented in tabular form; moreover, for more clarity plots graph are presented in Figure 1 and 2 that show how the computation times scale depending on the problem dimension. The plots show the running time for each model, for increasing numbers of the parameter N. The complexity of the adder models can be computed with the following formula: $12 * N + 5$.

[4] Appropriate notions of complexity are described for each table.

Model	Complexity	Time
adder_2	21	1.05
adder_5	55	1.78
adder_10	115	4.74
adder_15	175	8.91
adder_20	235	17.75
adder_25	295	32.03
adder_30	355	75.96
adder_35	415	571.66
adder_40	475	1786.30
adder_42	499	11297.04

Table 1. Test results for BDD-based model checking. The complexity is measured as the number of Boolean variables in the encoding.

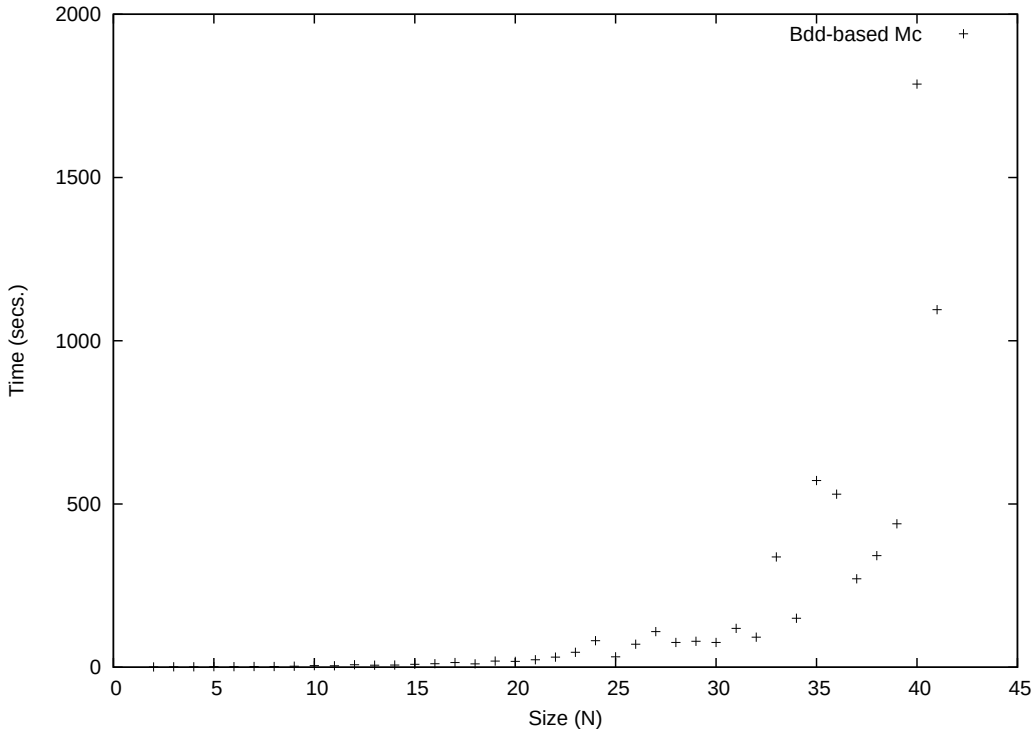

Fig. 1. Plot of results for BDD-based model checking. The size N is measured as the number of Boolean variables in the encoding.

Model	Complexity	Time
adder_2	21	0.93
adder_5	55	1.44
adder_10	115	1.40
adder_15	175	1.79
adder_20	235	2.06
adder_25	295	2.73
adder_30	355	4.01
adder_35	415	4.39
adder_40	475	5.68
adder_45	535	6.77
adder_50	595	8.96
adder_55	655	9.88
adder_60	715	12.98
adder_65	775	15.17
adder_70	835	17.43
adder_75	895	26.67
adder_80	955	44.78
adder_85	1015	36.19
adder_90	1075	59.85
adder_95	1135	56.56
adder_100	1195	45.41
adder_105	1255	58.57
adder_110	1315	89.72
adder_115	1375	61.10
adder_120	1435	66.93
adder_125	1495	103.66
adder_130	1555	75.12
adder_135	1615	80.24
adder_140	1675	84.64
adder_145	1735	83.96
adder_150	1795	144.27
adder_155	1855	100.71
adder_160	1915	160.54
adder_165	1975	108.00
adder_170	2035	106.64
adder_175	2095	159.51
adder_180	2155	153.68

Table 2. Test results for SAT-based model checking. The complexity is measured as the number of Boolean variables in the encoding.

Model	Complexity	Time
adder_2	19	0.48
adder_3	31	1.06
adder_4	43	1.82
adder_5	55	3.99
adder_6	67	7.84
adder_7	79	105.85
adder_8	91	335.96
adder_9	103	1671.63

Table 3. Test results for fault tree generation. The complexity is measured as the number of Boolean variables in the encoding.

Model	Complexity	Time
sensorfilter_2	2	4.05
sensorfilter_3	3	16.96
sensorfilter_4	4	74.21
sensorfilter_5	5	273.43
sensorfilter_6	6	861.69
sensorfilter_7	7	2677.01

Table 4. Test results for performability evaluation. The complexity is measured as the amount of redundant sensors and filters.

Model	Complexity	Time
sensorfilter_ft_1	1	0.50
sensorfilter_ft_2	2	1.01
sensorfilter_ft_3	3	10.53
sensorfilter_ft_4	4	92.29
sensorfilter_ft_5	5	2486.07
sensorfilter_ft_6	6	38337.98

Table 5. Test results for dynamic fault tree verification. The complexity is measured as the amount cut sets in the fault tree.

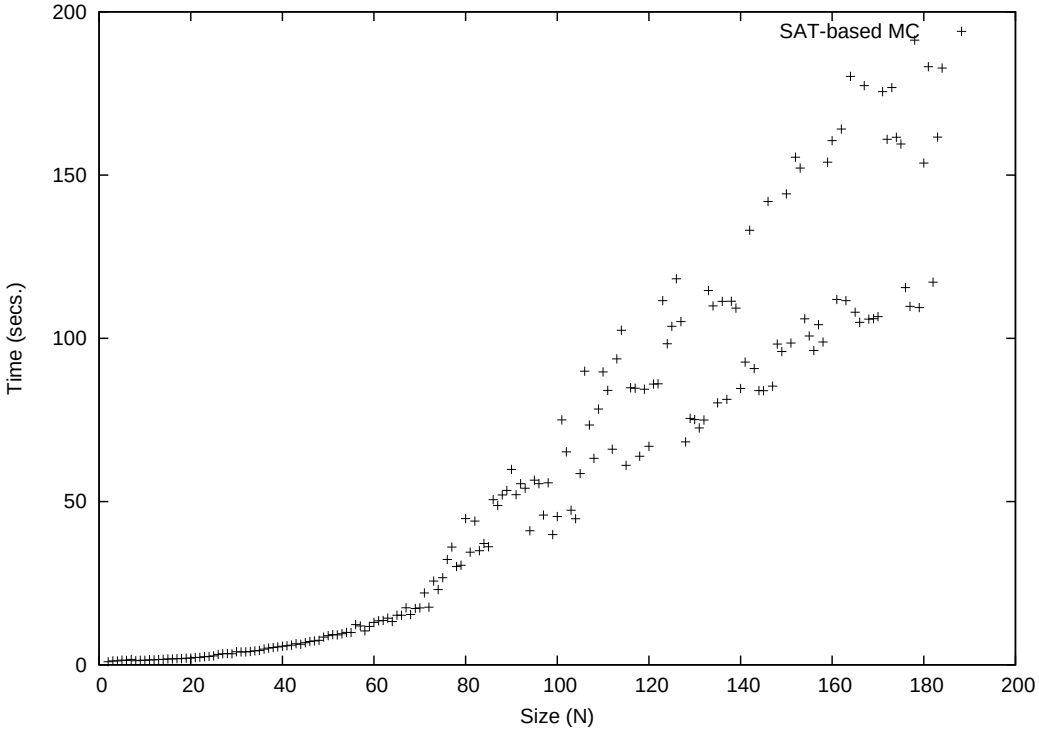

Fig. 2. Plot of results for SAT-based model checking. The size N is measured as the number of Boolean variables in the encoding.