MEDICI: Combinatorial Test Generation Based on Multivalued Decision Diagrams

Andrea Bombarda
Department of Engineering
University of Bergamo
Bergamo, Italy
andrea.bombarda@unibg.it

Angelo Gargantini
Department of Engineering
University of Bergamo
Bergamo, Italy
angelo.gargantini@unibg.it

I. Tool Description

With this document, we submit to the 2nd edition of the CT-Competition MEDICI [3], a C++ tool exploiting Multivalued Decision Diagrams (MDDs), through the library `meddly`[1], to generate test suites with the desired combinatorial coverage.

The tool source code, together with other documents and tests is available at https://github.com/garganti/medici.

A. Input format

MEDICI accepts inputs expressed in CASA format to describe a combinatorial model. Additionally, the tool defines its own input format, which offers greater flexibility in defining constraints, including the ability to use free-form expressions (not necessarily in CNF) and Polish Inverse Notation. An example of input in the tool's format is shown in Listing 1. The first row indicates the strength, the second row denotes the number of parameters, and the third row specifies the number of values for each parameter. The fourth line indicates the number of constraints, and the following lines report all the constraints, one per line, expressed using the Polish Inverse Notation.

```
2
4
6 4 3 3
1
0 6 * -
```

Listing 1: Example of a MEDICI model

In the constraints, the symbols *, + and - can be used, representing, resplictively, the and, the or and not operators. Moreover, the numbers used in the constraints indicates the values we are considering (0 is assigned to the first value of the first parameter, and so on). For example `0 6 * -` is equivalent to `NOT(0 AND 6)`, which means `NOT(Par1.Value1 AND Par2.Value1)` in the example above.

To participate in the 2nd edition of the CT-Competition, we expanded MEDICI's capabilities to accept input expressed in the CTWedge [2] format. We achieved this by incorporating a Boost Spirit parser into the MEDICI tool. It's important to note that the parser is still in its early stages and may not accurately handle all input models.

[1]https://meddly.sourceforge.io/index.html

B. Tool algorithm

MEDICI generates combinatorial tests using MDDs. It creates an MDD M_{VS}, which represents the intersection between the model domain (i.e., all the parameters declared in the combinatorial model along with their values) and the constraints. It then builds one test at a time until all the testing requirements are achieved. The tool utilizes a greedy approach to select the optimal parameter that is not already set in the test, along with its optimal value, according to the weighting criteria, and adds this assignment to the test being built. The greedy algorithm initially populates the list of all tuples T_{TC}, which includes all the combinations to be covered, given the strength t. However, some tuples may be infeasible due to the constraints. To filter only valid tuples, among those in T_{TC}, MEDICI keeps only those that have a non-empty intersection with the MDD M_{VS}, i.e., those that satisfy the constraints.

For each iteration, MEDICI generates a test case M_{nc}, starting from M_{VS} and appending new tuples until a final cardinality of 1 is reached. In the single iteration, M_{nc} is initialized to M_{VS}. Then, all the parameters are sorted by counting for each parameter p, the number of tuples in T_{TC} that contain p. Next, MEDICI assigns the best value to each parameter p, by taking the value that produces an assignment compatible with M_{nc} and that maximizes the coverage of tuples in T_{TC}. This process is iteratively repeated until T_{TC} becomes empty.

To optimize the result in terms of test suite size, MEDICI uses several optimizations as described in [3]. For example, the tool orders the tuples to be covered based on their *compatibility*, taking into account the constraints. Also, since the results of MEDICI are not deterministic, the tool can be configured to repeat the generation part using different starting seeds to find the best solution.

C. Tool execution

Since MEDICI requires several parameters to be set while generating test suites, we have developed an additional bash wrapper which sets all the required parameter and then ex-

TABLE I: Average of the results obtained in our experiments with MEDICI

Model	Size	Tot. time $[s]$	Model	Size	Tot. time $[s]$
UNIFORM_BOOLEAN_0	9.0	0.08	UNIFORM_ALL_0	66.7	2.40
UNIFORM_BOOLEAN_1	9.0	0.22	UNIFORM_ALL_1	167.7	17.87
UNIFORM_BOOLEAN_2	8.0	0.04	UNIFORM_ALL_2	101.0	10.2
UNIFORM_BOOLEAN_3	9.0	0.11	UNIFORM_ALL_3	208.7	21.19
UNIFORM_BOOLEAN_4	7.0	0.02	UNIFORM_ALL_4	7.0	0.02
MCA_0	164.0	25.54	BOOLC_0	9.0	0.11
MCA_1	82.0	1.53	BOOLC_1	11.0	0.20
MCA_2	147.7	3.36	BOOLC_2	11.0	0.20
MCA_3	188.0	29.72	BOOLC_3	8.0	0.02
MCA_4	197.7	32.51	BOOLC_4	5.0	0.02
MCAC_0	383.0	64.71	NUMC_0	–	–
MCAC_1	320.7	43.40	NUMC_1	–	–
MCAC_2	127.7	7.13	NUMC_2	–	–
MCAC_3	85.0	6.04	NUMC_3	–	–
MCAC_4	255.3	40.00	NUMC_4	–	–
FM_0	12.0	1.34	CNF_0	196.0	33.37
FM_1	9.0	0.04	CNF_1	115.7	11.06
FM_2	13.0	1.08	CNF_2	171.0	22.47
FM_3	12.0	1.07	CNF_3	242.3	30.96
FM_4	10.0	0.07	CNF_4	533.7	54.35
INDUSTRIAL_0	18.0	1.88	HIGHLY_CONSTRAINED_0	276.0	21.44
INDUSTRIAL_1	22.3	14.39	HIGHLY_CONSTRAINED_1	268.7	40.33
INDUSTRIAL_2	19.0	0.16	HIGHLY_CONSTRAINED_2	67.0	0.20
INDUSTRIAL_3	60.0	0.54	HIGHLY_CONSTRAINED_3	173.7	15.32
INDUSTRIAL_4	18.0	0.04	HIGHLY_CONSTRAINED_4	182.7	7.23

ecutes MEDICI. The whole process is executed through the following command:

```
./medici.sh [strength] [ctwedgeModel]
```

II. MEDICI PERFORMANCE ON EXAMPLE BENCHMARKS

In this section, we report the results obtained with MEDICI on the example benchmarks[2] given by the organizers of the second edition of the CT-Competition [1] at IWCT 2023 on a machine using a Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz (16 physical cores, 32 logical cores) with 256 GB RAM. As per the CT-Competition rules, we have set a timeout of 300 seconds. Moreover, the experiments have been executed 3 times for each model, and the average of the size and generation time is reported in Tab. I.

III. CONCLUSION

With this document, we have briefly introduced the tool MEDICI, previously presented in [3] and we have shown its applicability to the benchmarks given by the CT-Competition organizers. MEDICI has shown to be able to deal with all the example benchmarks, except for the NUMC instances, and to always produce test suites without having timeouts. Moreover, thanks to this competition, we have been able to extend MEDICI functionalities, by including CTWedge as a supported input format.

[2]https://github.com/fmselab/CIT_Benchmark_Generator/tree/main/Benchmarks_CITCompetition_2023

REFERENCES

[1] A. Bombarda, E. Crippa, and A. Gargantini. An environment for benchmarking combinatorial test suite generators. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, apr 2021.

[2] A. Gargantini and M. Radavelli. Migrating combinatorial interaction test modeling and generation to the web. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 308–317, April 2018.

[3] A. Gargantini and P. Vavassori. Efficient combinatorial test generation based on multivalued decision diagrams. In *Hardware and Software: Verification and Testing*, pages 220–235. Springer International Publishing, 2014.