CNF Encodings of Complete Pairwise Combinatorial Testing of our SAT Solver SATCH

Armin Biere
Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria

***Abstract*—This note describes the benchmarks we have submitted to the SAT Competition 2021 encoding the existence of a list of configurations of a given size k which covers pairwise all combinations of configurations of our SAT solver SATCH.**

The submitted DIMACS files encode feasibility of a list of configurations of a given size k for complete combinatorial pairwise testing [1], [2] of one internal version of our SAT solver SATCH available at https://github.com/arminbiere/satch. The encodings were generated by the tool GENCOMBI that comes with SATCH.

The `configure` script of SATCH has many different build options. Given a test suite (in case of SATCH the included test suite) the basic idea of two-way or pairwise testing is to run the suite on all combinations of all possible pairs of configuration options under the constraint that incompatible configuration options are avoided. Thus we want to produce a smallest possible list of configuration options satisfying these criteria. The DIMACS files encode the existence of such a list for a given size k. By default we also make sure that each pair of options is not used in at least one configuration.

Beside the size k of the test set, which corresponds to the number of different configurations, the instances vary in terms of dropping certain sorting constraints for symmetry breaking (option `--unsorted` and suffix '`u`') or dropping the requirement that all pairs of features should also not occur in a least one configuration (option `--weak` and suffix '`w`').

For the considered version of SATCH there does exist a list of configurations of size $k = 20$ satisfying all criteria. For size $k = 20$ all four instances are easy to satisfy including dropping inclusion of sorting constraints ('`u`') or dropping the requirement that pairs should also not occur ('`w`'). The smaller ones are getting hard at around $k = 14$ and are expected to be all unsatisfiable.

Note that in practical use GENCOMBI generates a new CNF for each considered k and first doubles k until an instance becomes satisfiable, where solving time is limited. Then the tool decreases k trying to reduce the upper bound or to find a lower bound where solving time takes 10 times more than for the upper bound. If a new upper bound is found the process repeats. The individual instances are actually kept in memory and solving is simply resumed if necessary (the only incremental way of solving supported for SATCH at this point).

Supported by the Austrian Science Fund (FWF) under projects W1255-N23, S11408-N23 and by the LIT AI Lab funded by the State of Upper Austria.

REFERENCES

[1] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatiorial design," *IEEE Trans. Software Eng.*, vol. 23, no. 7, pp. 437–444, 1997. [Online]. Available: https://doi.org/10.1109/32.605761

[2] A. Yamada, A. Biere, C. Artho, T. Kitamura, and E. Choi, "Greedy combinatorial test case generation using unsatisfiable cores," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, D. Lo, S. Apel, and S. Khurshid, Eds. ACM, 2016, pp. 614–624. [Online]. Available: https://doi.org/10.1145/2970276.2970335