ReconfAIGERation entering Core Challenge 2022

Nils Froleyks[1], Emily Yu[1], and Armin Biere[2]

[1]Johannes Kepler University, Linz, Austria
[2]Albert Ludwig University, Freiburg, Germany

March 31, 2022

Abstract

ReconfAIGERation encodes the Independent Set Reconfiguration problem into a circuit in AIGER format. The bad state property encodes that the target independent set is reached, without violating the token jump rule. A witness for the model checking problem is decoded into a reconfiguration sequence.

Encoding and Solving

To encode the ISR problem into AIGER (version 1.9 [1]) we define a circuit with one latch and two inputs per node. The latches encode if the node is in the current independent set. The inputs are divided into two groups: freed and marked. At each step, exactly one[1] node in the current IS is freed and one not in the current IS marked, thus enforcing the token jump rule. The independent set constraint is easily encoded by combining gates for each edge in the graph.

The tool aigtoaig is used to translate the human readable output from the previous step (.aag) into a binary encoded format (.aig).

For the existent-track the Aiger enconding is solved by the model checker ABC [4]. ABC runs multiple model checking algorithms in parallel. Among them is an implementation of IC3/PDR [3]. It can therefore prove the unsolvability of a problem.

For the shortest-track the model checking problem is solved with the Bounded Model Checker CaMiCaL [6]. In Bounded Model Checking (BMC)

[1]To encode the at-most-one part we used the product encoding [5] since a naive square encoding runs out of memory before the encoding is completed on some of the instances.

the transition function of a circuit is encoded into a SAT formula and copied a number of times (called makespan). The formula is satisfiable exactly if the target IS is reachable in at most makespan-many steps. Since the makespan is incremented step-wise, the first solution found is guaranteed to be the shortest. CaMiCaL is highly integrated with the incremental SAT solver CaDiCaL [2] to optimize the effectiveness of incremental inprocessing [6].

For the longest-track we skip the encoding to Aiger and use CaDiCaL directly. In addition to the usual BMC encoding we add a uniqueness constraint to the steps of the solution to disallow loops in the reconfiguration sequence. In theory the makespan is increased until it reaches the maximum possible unique solution length: $N = \binom{\text{number of nodes}}{\text{size IS}}$. The last found solution is then guaranteed to be the longest. In practice N exceeds the maximum integer value in most cases and the computation will not terminate in reasonable time or memory.

Implementation

Our solver is written in Nim and can be compiled with:

```
nim cpp -d:release --passL:cadical/build/libcadical.a reconfaigeration
```

It depends on other executables in the same directory. We cannot provide the source code for all of them.

We generated the log files in parallel on 32 nodes of our cluster. Each node has access to two 8-core Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz (turbo-mode disabled) and 128 GB main memory.

Since ABC spawns multiple processes we only allocated one instance per node. For CaMiCaL we ran 4 instances in parallel on each node. The memory was limited to 32 GB for each of the instances. The timeout was set to 10000 seconds for both. For the longest-track we used the same number of instances and memory limit and a timeout of 5000 seconds.

References

[1] Armin Biere, Keijo Heljanko, and Siert Wieringa. *AIGER 1.9 and Beyond*. 11/2. Altenbergerstr. 69, 4040 Linz, Austria: Institute for Formal Models and Verification, Johannes Kepler University, July 2011, 2011.

[2] Armin Biere et al. "Entering the SAT Competition 2020". In: (2020), p. 4.

[3] Aaron R. Bradley. "SAT-Based Model Checking without Unrolling". In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by Ranjit Jhala and David Schmidt. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 70–87. ISBN: 978-3-642-18275-4. DOI: 10.1007/978-3-642-18275-4_7.

[4] Robert Brayton and Alan Mishchenko. "ABC: An Academic Industrial-Strength Verification Tool". In: *Computer Aided Verification*. Ed. by Tayssir Touili, Byron Cook, and Paul Jackson. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 24–40. ISBN: 978-3-642-14295-6. DOI: 10.1007/978-3-642-14295-6_5.

[5] Jingchao Chen. "A New SAT Encoding of the At-Most-One Constraint". In: *Proc. Constraint Modelling and Reformulation* (2010), p. 8.

[6] Katalin Fazekas, Armin Biere, and Christoph Scholl. "Incremental Inprocessing in SAT Solving". In: *Theory and Applications of Satisfiability Testing – SAT 2019*. Ed. by Mikoláš Janota and Inês Lynce. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 136–154. ISBN: 978-3-030-24258-9. DOI: 10.1007/978-3-030-24258-9_9.