AIGEN tool user manual

June 2, 2021

AIGEN is an open source tool for the generation of transition systems in a symbolic representation "AIGER". To ensure diversity, it employs a uniform random sampling over the space of all Boolean functions with a given number of variables. AIGEN relies on reduced ordered binary decision diagrams (ROBDDs) and canonical disjunctive normal form (CDNF) as canonical representations that allow us to enumerate Boolean functions, in the former case with an encoding that is inspired by data structures used to implement ROBDDs. Several parameters allow the user to restrict generation to Boolean functions or transition systems with certain properties, which are then output in AIGER format.

Contents

1 Testing the Tool

AIGEN is pre-installed on this virtual machine in folder "AIGEN-AE-Submission". To test its basic functionality, you can run a simple command like: python3 AIGEN-AE-Submission/aigen.py -bdd -output ReviewerTest.aag -c 1 -u 1 -l 5 -o 4 -noABC

where:

- **FileName.aag** is the file name to be generated.
- **-bdd** means that use a bdd based random generation. When replaced by **-dnf** the tool will use canonical DNF based random generation.

- **-c 1** means 1 controllable input
- **-u 1** means 1 uncontrollable inputs
- **-l 10** means 10 latches (i.e. the tool will generate 12 Boolean functions over 15 variables)
- **-o 9** means that 9 latches variable influences the output function (i.e. the output function will assign true to (l - o) variables and in this case (10-9)).
- **-noABC** is an optional parameter. A user has to omit it if he wants to run the ABC tool minimization procedure that reduces the size of the generated AIGER file.

After the call terminates, AIGEN will have generated the file "FileName.aag", including the resulting transition system in AIGER format and comments at the end of the file that include the parameters of its generation, namely the seeds that has been used to generate the random numbers for each latch, and the seed for choosing random 9 variables out of 10 to be used in the output function.

To replicate an existing file (an AIGER file that was generated previously by AIGEN tool), use as arguments the same model (bdd or dnf), number of uncontrollable inputs, number of controllable inputs, number of latches, the same number of output variables, and the same seeds. All these info can be found as comments at the end of the file. Note that the number of seeds must be equal to l + 1.

The following is a sample command to generate a replica of the AIGER file **testFile.aag**:

```
python3 AIGEN-AE-Submission/aigen.py -bdd -output testFileReplica.aag -c
1 -u 1 -l 5 -o 4 -seeds 1619008565150552 1619008565157898 1619008565162168
1619008565166834 1619008565170687 1619008565175030 -oseed 1619008565177500
-noABC
```

The numbers after "-seeds" are the seeds for the latches and the number after "-oseed" is the seed for the output function.

2 Tables and figures replication

2.1 Figures Replication

We provide three commands for replication of the figures: the first runs quickly (in 20-30 min) and replicates part of Figures 2 and 3, the second replicates Figures 2 and 3 completely, and the third replicates Figures 2, 3, and 4 (but will run tens of hours).

- The below command generates results for Figures 2 and 3, which compare the behaviour of the BDD-based and the DNF-based approach without using the ABC tool (-noABC). However, in order to replicate a meaningful subset of the results in an acceptable amount of time, for every configuration, it generates 3 benchmarks instead of 9, and we omit the following configurations: 9.9.4 and 10.11.2.
  - Run the command: AIGEN-AE-Submission/experiment-noABC.sh
- The below command generates results for Figures 2 and 3. Similar to the above command, for every configuration, it generates 3 benchmarks instead of 9. That means this command will generate whole figures however the averages will be computed from 3 instances instead of 9.
  - Run the command: AIGEN-AE-Submission/experiment-noABC-2.sh
- The below command replicates Figures 2, 3, and 4 completely, in exactly the same way that we used for the paper. That is, it includes all configurations, and computes the average of 9 random instances per configuration, except for the very large configuration 10.11.2, which is averaged over 5 instances:
  - Run the command: AIGEN-AE-Submission/experiment.sh (WARNING: this will take tens of hours...)

These commands will generate AIGER files in folder "AIGEN-AE-Submission/experiments", "plots.pdf" file that contains the plots, and "results.txt" which is the log file.

2.2 Table 2 Replication

Note that the results from Table 2 can not be replicated in this VM, as they were generated by the competition chairs of SYNTCOMP, using additional synthesis tools that are not available here. These results are reported in the StarExec instance of SYNTCOMP, which can be found at the following URL:
https://www.starexec.org/starexec/secure/details/job.jsp?id=35621.
The file : simpleBDDSolver.csv contains logs of the results of the tool "SimpleBDDSolver" extracted from the above link.

3 Log File

The log file of the paper results is "Paper-Results.txt". Moreover, after running the replication commands the user will find the logs in file "results.txt". A log file contains for each generated AIGER file the following: The generation model (dnf or bdd), the file name, the number of AND gate, and the creation Date time.

4 The Code

The code is split into five files:

- **aigen.py**. This is the main file to run the tool, it reads first the inputs from the user and then execute the code accordingly. The main arguments are "-bdd" or "-dnf", once one is chosen the corresponding files are loaded and executed.
- **DNFtoAIG.py**. This file implements the DNF approach. First it generates for each latch a random truth table, i.e. a bit vector that determines the satisfying assignments. Then it converts the truth table to a set of AND gates, that are in turn written in an AIGER file. **The most important function** here is: $generate\_DNF\_latch\_fct$ which converts a truth table represented in a bitvector to a Boolean formuala with only AND and NOT operators (gates).
- **GenerateBDDs.py**. This file generates for each latch a random number and generates for this number the equivalent ROBDD as explained in the paper. The ROBDD is saved in a table called *bdd_tuple* which contains all the triples $(bdd\_ID, high, low)$. **The most important functions** here are:
  - *getChildren*. Given a bdd_ID this function extracts its low and high children as explained in the paper (they are extracted from the VRT table).
  - $generate\_latches\_fct$. This function creates for every latch an AIG, and it creates an AIG for the output function.
- **convertToAiger.py**. Given the ROBDDs table *bdd_tuple* this function will create the corresponding AIGs making use of the fact that a BDD can be seen as a network of multiplexers. Finally, these AIGs are written in an AIGER file.
- **ABCMinimization.py**. This file contains a function that minimizes an AIGER file by first converting it to AIG format using the aigtoaig tool, then it uses the ABC tool to minimize the AIG file, then uses again aigtoaig tool to convert the resized AIG to AIGER.