Assignment for VDS Class Project

Lucas Deutschmann, Philipp Schmitz

1 Part 2

In the second part of the Class Project, you are going to implement several performance improvements. The goal is to improve your implementation regarding both runtime and memory efficiency.
You can start by following these steps:

- Integrate all files in your project repository.
  The source files should be located in *src/bench/* and *src/verify/* respectively. The benchmarks should not be in the source folder and can be added to the project directory directly (*project_directory/benchmarks/*). How to run the tools is described further below.
- Read the provided paper (*BDD_Paper.pdf*) on "Efficient Implementation of a BDD Package"
- Adjust your *CMakeLists.txt* in order to compile the benchmark and verification tool (see below)
- Verify your implementation with the provided results for the c432 and (if possible) c3540 benchmark
- Make sure the benchmark results are **not** added to your git repository. The easiest way to achieve this is to edit your .gitignore accordingly.
- Your main task is to implement the computed table as presented in the paper, using a hash function.
  If you're not familiar with the concept of hashing or how it works in C++, you should start by researching it.
- In order to achieve the required performance, you're also going to need a *second* hash table other than the computed table. It will be up to you to find out, where look-ups can also be greatly enhanced using a hash table. Investigate the performance of your implementation again and try to identify the bottleneck.
- *Optional:* Feel free to implement any additional optimizations from the paper, such as the *Standard Triples*.
- To pass the second part, your implementation should run the c3540 benchmark in *roughly 10 seconds* while keeping the VM requirement *below* 10^6. If you think your implementation is fast enough, let us know and we will run the final estimation on our machine.
  Hint: When benchmarking with CLion, make sure you compile using *Release* mode!

2 Running a Benchmark

To use the benchmarking tool, you first have to add the source files to *project_directory/src/bench/* and add the sub directory to the CMake list:

include_directories($CMAKE_SOURCE_DIR/src/bench/)
link_directories($CMAKE_SOURCE_DIR/src/bench/)

To run the tool, the path to the bench file should be passed as an argument to the binary.
The benchmarking tool receives a logic circuit described in the bench format and generates an ROBDD for each output function using your implementation of the Manager class. For each output function in the bench file, it will provide a graphical (dot format) and text representation of the generated ROBDD. Furthermore, information about the total runtime and memory usage of the ROBDD generation procedure is reported.
After running the tool, the performance of your implementation is reported as follows:

Runtime: The amount of time spent executing user code. It should be noted that this is different from wall time, as the time spent executing system functions or other OS activities is not considered.

VM: The virtual memory size consumed by the program.

RSS: The portion of memory occupied by the process that is held in main memory (RAM).

The computed ROBDD will be stored in a subfolder named *results_$benchFileName$* in the same directory.

3 Verifying the Result

To verify your result, a simple program is given to check the equivalence between your generated ROBDD and the correct graph generated by our implementation. You first have to add the source files to *project_directory/src/verify/* and add the sub directory to the CMake list:

include_directories($CMAKE_SOURCE_DIR/src/verify/)
link_directories($CMAKE_SOURCE_DIR/src/verify/)

This program takes two ROBDDs in text representation and checks if the two graphs are isomorphic. The text representation of the ROBDD can be found in the following path after running the benchmark tool:

results_$benchFileName$/txt/$outputName$.txt