Discrete Modelling Optimisation and Search: Assignment 1

Olivier Delattre - 0555358
Elias De Deken - 0555196

January 2023

1 Introduction

In this paper we present a veriPB proof-logging extension for *cryptominisat*. *Cryptominisat* is an extension of *Minisat*, made to be better suited to solving cryptographic problems. This is done by extending the solver with the ability to reason about xor-clauses, which are common in these kinds of problems.

Our contribution is adding proof-logging in veriPB format to *cryptominisat*.

2 Restrictions

In order to keep the scope of this project to a manageable size, we had to impose some restrictions on our work.

First, we decided to implement veriPB proof-logging to an older version of *cryptominisat* (version 1.2.8[1]). The reason for this is that the implementation of this version lies closer to the original *minisat* implementation. As a result, the extensions made by *cryptominisat* are simpler to find in the code, allowing us to put our focus on adding proof logging to these extensions. This version is built on the assumption that xor-clauses are provided in the input "cnf"-file in DIMACS[2] format. Xor-clauses take the form of a normal clause preceded by an x. For example, the line (x1 -2 3 4 0) represents the xor-clause:

$$x_1 \oplus \neg x_2 \oplus x_3 \oplus x_4 \oplus 0$$

By adding support for xor-clauses to the DIMACS format, these cardinality constraints can be represented in a more concise format[3], reducing the number of clauses the solver has to store in memory.

Second, our implementation currently only allows clauses with up to 20 variables. This restriction was made since we had to extend the inner representation

[1] https://github.com/msoos/cryptominisat/tree/marijn-1.2.8
[2] https://logic.pdmi.ras.ru/ basolver/dimacs.html
[3] https://www.msoos.org/xor-clauses/

of xor-clauses such that they keep a reference to their original form as well as their simplified form. This restriction could be avoided by assigning each clause memory proportionate to twice their original clause size, rather than using a fixed value as is currently the case.

3 Discussion

We will now discuss several important aspects of our extension:

Since the veriPB proof checker does not natively support reasoning with xor-clauses, it would not be able to read the original cnf-file containing the solved formula. As a solution, we have decided in our current implementation to simply make the solver create an extended input cnf-file. This extended file contains all the clauses from the original formula, as well as cnf-encodings of xor-clauses that were used to propagate literals. E.g. if clause $(x_1 \oplus \neg x_2 \oplus x_3)$ is used under current assignment $x_1 = 1, x_2 = 0$ to propagate x_3, the solver would add the clause $(\neg x_1 \vee x_2 \vee \neg x_3)$ to the extended cnf-file. As a result, only a subset of all encodings of xor-clauses are actually added to the cnf-file, reducing the initial amount of constraints veriPB has to take into account. The proof checker can then use this extended cnf-file to verify the proof.

When running the solver, 2 files should be created. The first file is named "*cryptominisat_cnf_log_final.cnf*" and contains this extended cnf-file. The second file is named "*cryptominisat_proof_log_final.cnf*" and contains the veriPB proof.

4 Future work

The following paragraph discusses some potential future work that could be done concerning our extensions. First of all, instead of printing the relevant xor-encodings to the cnf-file, veriPB could be modified such that it natively supports xor-clauses. Secondly, the proofs generated by our extension only make use of reverse unit propagation statements to infer new clauses. This could be extended such that shorter, more efficient proofs are found. Finally, our extensions could be mapped to the newer releases of *cryptominisat*.