Lab 1: Implementing CDCL

September 20th, 2023

The goal of this lab (and mini-project) is to code some simple CDCL solver and, if possible, some of its improvements. You are free to use any programming language. You could work with at most one partner. The due date is 2023-10-27 at 1pm.

1 Input format: DIMACS

The standard way to represent a Boolean formula in CNF is the DIMACS format. The format is used as input to modern SAT-solvers. A file in the DIMACS format is a text file. It starts with a line specifying that the formula is in normal form, the number of variables in the formula, and how many clauses it contains. For example, `p cnf 4 3` specifies that the file contains a formula in conjunctive normal form with 4 variables and 3 clauses. Then, the next lines describe the clauses, one on each line. Each line contains a list of positive or negative integers, and ends with a zero. A positive integer i indicates that the ith variable appears in the clause, whereas a negative integer $-i$ indicates that the negation of the ith variable appears in the clause. For example, here's a formula, followed by its encoding using the DIMACS format: $(x_1 \vee x_3 \vee x_4) \wedge (x_4) \wedge (x_2 \vee x_3)$;

```
p cnf 4 3
1 3 -4 0
4 0
2 -3 0
```

An OCaml parser for the DIMACS format is available on eCampus.

2 CDCL

You have to implement the basic algorithm for CDCL given on the lecture notes. The start may be the DPLL coded on OCaml which is available on eCampus.

3 Optimizations

3.1 Optimal Storage

Since CDCL is making big use of memory, it could be useful to compact the representation of variables and clauses. Here are some ideas:

- Variables are represented using one byte/int/long; positive literals are represented by positive values, negative literals by negative values.

- Clauses may be encoded by a record with a header storing some information about the clause (e.g., how much literals unassigned) to avoid repeated computation of this information in the algorithm. The set of literals (integers) may be stored in a list or some (see below watched literals) stored in an array and the remainder in a list.
- Partial models are good to be stored as a stack since the main operations are pus/pop.

3.2 Special cases

Before launching CDCL, your solver could identify if the set of clauses form a Horn formula or a two-variable formula. In these cases, a simpler solving algorithm exists, so it can be applied instead of CDCL.

Could this optimization be applied on a sub-set of clauses? That means, if a subset of clauses is in the Horn format (for instance) could we start the CDCL from the partial assignment given for these clauses? If yes, is this a complete decision procedure?

3.3 Two-watched Literals (2WL)

To applying the unit propagation, all literals in a clause except one need to be false with respect to the current partial model. For detecting a conflict, all literals need to be false with respect to the current trail. Thus as long as at least two literals of a clause are unassigned by the current assignment (partial model) or at least one literal is true with respect to the current assignment the clause can be disregarded by the CDCL algorithm.

This results in the well-known 2-watched literals idea where only two literals of a clause are indexed. Indexing of literals (i.e., in which clauses they are used) is needed in general, in order to efficiently access relevant clauses. After a decision or propagation of some literal ℓ, all clauses containing ℓ (or its complementation) need to be visited in order to check for unit propagate and conflict. Simply traversing all clauses is too inefficient. A complete indexing of clauses assigns to every literal the sequences of clauses containing the literal. The 2-watched literal index reduces this to indexing only the first two literals of a clause. The invariant for the 2-watched literals with respect to the current run is:

> If one of the watched literals is false and the other watched literal is not true, then all other literals of the clause are false.

There are two major advantages of indexing a clause by 2WL:

- when a literal of the clause that is not watched changes its truth value nothing needs to be done,
- at backtracking, there is no update on the 2WL data structure, because literals are only changed to status undefined.