

Vallst

Daniel Vallstrom

Abstract

A boolean constraint solver, *Vallst*, is briefly and spottily described. Vallst is a solver in the Chaff tradition [3, 4] with various extensions and novelties.

The solver has support for more formula types than just disjunctions, for example equivalences and multiplicative constraints $c < \sum_i c_i p_i$. You can tell the solver to maximize or minimize constraint constants. There are also “global” formula types.

We will discuss to some extent one semi-novelty, *proof improvement*, where you take time trying to improve upon a (sub-) proof instead of moving on directly. This is somewhat similar to Multiple Individual Conflicts (MIC) in [2].

1 Introduction

The solver is open source and available both as an API or library, and as a standalone, command line based program or script.

2 Global formula types

A global formula type says that a set of literals must have some property. If it is not possible to say in a concise and efficient manner that a set of literals must have the property, then introducing a global formula type might be a good solution.

For example, implemented in the solver is a global tour formula type that says that a solution corresponding to nodes and edges in a complete directed irreflexive graph must be a tour (i.e. a Hamiltonian cycle).

3 Proof improvement

Suppose that the solver, given some assumptions, has found a conflict by closing under some rules using some boolean constraint propagation, bcp_0 . It is conceivable that closing under

the same rules but applying them in some different order could yield a different, perhaps shorter and better proof.

Proof improvement is implemented in the solver as follows. Let bcp_1 differ from bcp_0 only in the order that the rules are applied. When a conflict, using bcp_0 is found, the solver resets to a state with assumptions that is known to be conflicting. Then the solver closes again using bcp_1 this time. Then the proofs are compared. If the first, bcp_0 proof is better than the second, bcp_1 proof, the solver again resets to a known conflicting state, closes using bcp_0 and continues from there. Otherwise, the solver just continues, using the second, bcp_1 proof.

Although largely untested, initial indications suggest that proof improvement, as implemented here, isn’t too beneficial generally. This is consistent with [2] where it was found that MIC usually wasn’t beneficial.

4 Scripts

Apart from a usual standalone solver program available, there are solver scripts containing functionality not available in the standalone solver program alone.

The scripts repeatedly calls the standalone program. After each call to the standalone program, the state and progress is saved and used in the next call of the program. Between calls there are syntactical simplifications made. In particular, if it has been proved that some variable must have a certain truth value, that variable will be simplified away.

The repeated calls also give opportunities to try different high level heuristics. For example, occasionally the scripts call the standalone program in a simplification heavy mode, telling it to spend more time on simplifications and proof improvement than otherwise.

A new invocation of the standalone program will be made if enough variables have attained a truth value, or if enough time has passed.

5 Conflict clauses without watched literals

An other (untested) feature of the solver is that you can opt not to have any watched literals in some or all conflict clauses. This means that the search won't be slowed down by all the accumulated conflict clauses. On the other hand it also means that the boolean constraint propagation, BCP, will be weaker. The conflict clauses are still added for the conflict analysis and for Berkmin [1] style branching heuristics where an open variable in the last open formula is branched on.

References

- [1] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Design, Automation, and Test in Europe (DATE '02)*, pages 142–149, March 2002.
- [2] Marc Herbstritt. zchaff: Modifications and extensions. Technical Report 188, Institute of Computer Science, Albert-Ludwigs-University, July 2003.
- [3] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.
- [4] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285, November 2001.