GlucoRed

Siert Wieringa
Aalto University School of Science
Espoo, Finland

***Abstract*—This document describes the SAT solver GlucoRed as submitted the SAT Competition 2013. GlucoRed is an implementation of the solver/reducer architecture based on Glucose 2.1.**

I. Introduction

The solver GlucoRed discussed in this document implements the solver/reducer architecture [1], on top of the solver Glucose 2.1[1] [2]. It uses two concurrently executing threads, which are called the SOLVER and the REDUCER. The SOLVER acts just like Glucose would, except for its interaction with the REDUCER. The REDUCER's sole task is to strengthen the conflict clauses derived by the SOLVER. The interaction between the SOLVER and the REDUCER is handled by passing clauses through two shared-memory data structures called the *work set* and the *result queue*.

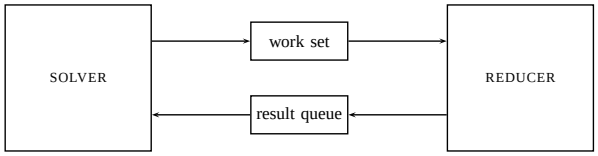

Fig. 1. The solver-reducer architecture

Whenever the SOLVER learns a clause it is pushed into the work set. The work set has a limited capacity. If a clause is pushed into the work set while it is full the new clause will replace the oldest clause in the set. The task of the REDUCER is to strengthen the clauses provided by the SOLVER through the work set. The reducer always picks the "best" clause from the work set as its next input. By setting the sorting metric for the work set the user may define the "best" clause as either the newest clause, the shortest clause or the clause with the smallest LBD [2]. The REDUCER tries to strengthen the clause by an algorithm based on unit propagation and conflict clause learning. If the REDUCER successfully reduces the length of a clause it places this new reduced clause in the result queue.

The SOLVER reads the clauses from the result queue, and adds them to its learnt clause database. The SOLVER can decide to do this at any decision level, hence the introduction of these "foreign" clauses may force the solver to backtrack.

II. Running GlucoRed

The solver GlucoRed inherits all parameters and magic constants from Glucose. Changing the value of any of these parameters affects both the SOLVER and the REDUCER, except for `-ccmin` which affects only the REDUCER. A new parameter `-solver-ccmin` controls the conflict clause minimization mode of the SOLVER. Both `-ccmin` and `-solver-ccmin` have default value "2=deep", which is the same as in MiniSAT and Glucose. There are two other new GlucoRed specific parameters. The first is the parameter `-work` which is an integer > 1, and represents the capacity of the work set. Its default value is the magic constant 1000. The second new parameter, `-rsort`, controls the sorting metric used for the work set. Legal values are "0=off" (newest first), "1=by size" (shortest first), "2=by LBD" (smallest LBD first). The default value is 2.

Although GlucoRed uses concurrency its performance remains decent when it is run on a single physical CPU core. This can be enforced for example by using the LINUX command `taskset`. We therefore submit our solver to both the sequential and parallel core solver tracks, for the benchmarks Application SAT+UNSAT, and Hard-combinatorial SAT+UNSAT.

III. Implementation details

GlucoRed is an extension of Glucose 2.1, which itself is based on MiniSAT 2.2.0[2]. All code is written in C++. The code that is unique to GlucoRed uses POSIX threads. The SOLVER and the REDUCER are both derived from Glucose's `Solver` class. GlucoRed was compiled to include MiniSAT's internal simplifier as implemented in the `SimpSolver` class. MiniSAT's original Makefile was used for compiling. Before submitting the code to the competition it has been tested after compiling it for a 64-bit architecture using `gcc` versions 4.4.7 and 4.6.3. It should also work correctly when compiled for a 32-bit architecture.

The version of GlucoRed submitted here differs from the version used for the experiments in [1] by the addition of MiniSAT's simplifier, and a minor clean-up of the source code.

IV. GlucoRed-Multi

GlucoRed-Multi is a simple multi-process portfolio of multiple instances of GlucoRed. There is no clause sharing between the different processes, but file parsing, initial iterative unit propagation, and optional simplification are only performed once. This is achieved by creating one instance of the solver, parsing the input file and performing preprocessing, and then forking the process multiple times.

[1] http://www.lri.fr/~simon

[2] http://www.minisat.se

Compared to GlucoRed the solver GlucoRed-Multi has two extra parameters, `-nc` and `-ns`. The parameter `-nc` controls the number of instances of GlucoRed to run directly on the input formula. The parameter `-ns` controls the number of instances of GlucoRed to run on the formula obtained by simplification using MiniSAT's internal simplifier. The GlucoRed-Multi solver was submitted to the same parallel core solver tracks as the basic GlucoRed solver, with parameter settings `-nc=1` and `-ns=3`. Given those settings and an input formula GlucoRed-Multi will do the following:

1) Create an instance of the 'GlucoRed' solver
2) Parse the input formula
3) Fork a copy of the process, run solver in child process.
4) Run the simplifier in the parent process.
5) Fork two copies of the parent process, run the solver in the parent process and both children.

The solver instance running in the parent process uses all the default GlucoRed settings. The solver instances running in the child processes also use the default settings, except from making 2% of their branching decisions at random and having a unique random seed based on their process id.

V. GlucoRed+March

GlucoRed+March is our submission to the *open track*. Even though the ranking in the open track is based on wall clock time this submission aims for a decent performance regarding CPU time. This is an experimental submission to see how simple heuristics compete with complex portfolios. It is not meant to be a serious contender for any awards.

GlucoRed+March runs a single copy of the solver march_rw[3] [3] if all clauses of the input formula have the same length, or if the formula contains clauses of exactly two different lengths and the diameter of the variable interaction graph (VIG) is at most 4. In all other cases GlucoRed+March runs a single copy of GlucoRed. Checking whether all clauses have the same length is a cheap way of determining that the formula is likely to be a random k-SAT formula. The use of the diameter of the VIG was inspired by [4].

VI. Availability

The source code for all submitted solvers is available from the author's web page[4]. The sources for MiniRed, a MiniSAT based solver/reducer implementation, are also provided through that same page. Both GlucoRed and MiniRed are licensed under MiniSAT's original non-restrictive license.

References

[1] S. Wieringa and K. Heljanko, "Concurrent clause strengthening," in *SAT*, ser. Lecture Notes in Computer Science, to appear 2013.

[2] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *IJCAI*, C. Boutilier, Ed., 2009, pp. 399–404.

[3] M. J. H. Heule, "Smart solving: Tools and techniques for satisfiability solvers," Ph.D. dissertation, Delft University of Technology, The Netherlands, 2008.

[4] P. Herwig, "Decomposing satisfiability problems," Master's thesis, Delft University of Technology, The Netherlands, October 2006.

[3] http://www.st.ewi.tudelft.nl/sat

[4] http://users.ics.aalto.fi/swiering/solver_reducer