gluH: Modified Version of gluclose 2.1

Chanseok Oh
New York University
New York, NY, USA

***Abstract*—This document describes the SAT solver "gluH" submitted to SAT Competition 2013, a modification of glucose 2.1, whose main feature is an addition of the generational management of learnt clauses.**

I. Introduction

gluH is a modified version of glucose [1], which is in turn based on the well-known MiniSat [2], a typical CDCL solver renowned for its compactness yet decent performance. The main addition of gluH is its generational management of learnt clauses, which periodically relegates dormant learnt clauses to the second class, separating them from first-class learnts recently created or actively participating in unit propagation. The aim is to reduce the size of the database of the first-class learnts to facilitate rapid propagations, and the idea is based on the observation that often, clauses that have not been involved at all in unit propagation for a long time have a fair chance that they will be rarely useful for a while, if at all.

II. Main Techniques

The major difference between gluH and glucose is the addition of the generational management of learnt clauses, despite the implementation and its evaluation being still in a primitive stage.

Aside from the normal learnt clause management, clauses inactive for a long period of time are further classified as dormants and demoted into the second class where they receive less attention from propagation. Note that only dormant learnts can ever be demoted; gluH still discards learnts immediately and permanently during the regular reduction.

Basically, clauses classified as dormants are not always inspected for possible unit propagation or conflict but on a random basis. However, if they become involved in propagation or conflict, they are promoted back to the first class promptly.

As one can expect, the database of dormant learnts will grow over time, and gluH uses the exact same logic to reduce its size, i.e., based mainly on LBD [1] values.

III. Main Parameters

The ratio of discarding learnts when performing database reduction in original glucose, as is the case with MiniSat, is roughly half. gluH reduces this ratio to prevent removing too many useful clauses, since a considerable portion of learnts will be relegated and excluded as dormants. For the dormant learnt database, the ratio is half.

The criterion for classification of a dormant clause is whether it has ever been involved in propagation or conflict within a fixed number of last conflicts; the number is static throughout the entire execution.

IV. Implementation Details

Separate watcher lists are maintained for dormant learnts. The normal watcher lists are accessed whenever a variable is assigned, as usual, whereas the watcher lists for dormants are accessed with a relatively small and fixed probability.

The task of classifying dormants is carried out periodically, in synchronization with the regular database reduction. For the purpose of classifying dormants, each learnt clause is assigned a timestamp when created, which will be updated if the clause is involved in unit propagation or conflict; the timestamp is the accumulated number of conflicts so far.

If a clause is deemed to be dormant according to the above-mentioned criterion, it is detached from the normal watcher lists and added to the dormant lists.

V. SAT Competition 2013 Specifics

Two instances with different configurations have been submitted to SAT Competition 2013.

1) Using SatELite [3] as a front-end CNF preprocessor, ratio of normal learnt reduction: 1/4, probability of checking dormant watcher lists: 0.25, dormant learnt criterion: silent for 20000 conflicts
2) Based on "simp" version of MiniSat, ratio of normal learnt reduction: 1/4, probability of checking dormant watcher lists: 0.20, dormant learnt criterion: silent for 30000 conflicts

VI. Availability

gluH adds no additional license to that of glucose.

References

[1] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *IJCAI*, C. Boutilier, Ed., 2009, pp. 399–404.
[2] N. Eén and N. Sörensson, "An extensible sat-solver," in *SAT*, ser. Lecture Notes in Computer Science, E. Giunchiglia and A. Tacchella, Eds., vol. 2919. Springer, 2003, pp. 502–518.
[3] N. Eén and A. Biere, "Effective preprocessing in sat through variable and clause elimination," in *SAT*, ser. Lecture Notes in Computer Science, F. Bacchus and T. Walsh, Eds., vol. 3569. Springer, 2005, pp. 61–75.