MINIPURE

Dept. of Electrical Engineering
National Taiwan University
Taipei, Taiwan ,Hsiao-Lun Wang
B98901096@ntu.edu.tw

***Abstract*—This document is a description of the sat solver — minipure of SAT competition 2013. The implementation embedded the pure literal detection in MINISAT which is a well known conflict based solver.**

I. INSTRUCTION OF MINIPURE

This is first version of MINIPURE. If a literal b is not in CNF formula $\mathcal{F}$ while b' is then b is called pure literal, and the clause contain b' can be eliminated for we can set b=False. Iteratively, we can eliminate all the pure literal thus reduce the number of clauses and variable of $\mathcal{F}$. But Chaff [1] and other kinds of conflict based solver sacrifice this heuristic for the efficiency of unit propagation. MINIPURE is the implementation of combining pure literal implication and unit propagation together to get better performance.

The solver is choose to embedded this idea in is MINISAT [2] for its simplicity and one of most well known conflict based solvers. The version of the MINISAT embedded is 2.2.0 . The implementation detail would be introduced later.

II. MAIN TECHNIQUES

The algorithms paradigm based on is CDCL.The further solving techniques used are: preprocessing, restart, phase saving , learning by conflict clauses. The above are common technique used in CDCL. Here are special techniques for MINIPURE: pure literal detection and pure literal learning (in part 4).

III. MAIN PARAMETERS

Beside the parameters used by MINISAT [2],there three parameters used for MINIPURE to tune.

1) additionan three paramters:

A . freq_pure : this paramters decide after how many retarts we can do pure literal detection again after restart level of stop_pure.

B. stop_pure: when the restart level is is bigger than the stop_pure *stop the pure literal detection and learning.*

C. *slowdown_pure: after which restart level should minipure slow down the pure literal detection. This number should be smaller than the stop_pure.*

For example, if slowdown_pure=2, stop_pure=3 and freq_pure=5, then at restart level 0~2 MINIPURE would do pure literal detection and learning and level 3 it would slow down the pure literal detection and after level 3 only level 8,13,...,3+5n would do pure literal detection again.

2) The paramters MINIPURE use for sat2013 competition is slowdown=0, stop_pure=1 and freq_pure=2. These three parameters are choose by the experiment result(t uned by hand) ,they have good performance on average. But for some special case these paramters should be changed.
Other parameters are the same as the default value of MINISAT.

IV. SPECIAL ALGORITHMS, DATASTRUCTURES, AND OTHER FETURES

The procedure and data structure explained below assumes the reader knowing about algorithms of Chaff and implementation of MINISAT. For more detail about the complete algorithm about Chaff and MINISAT please refer to [1] and [2].

1) Data Structure and initialize :

A. After parsing and preprocessing, MINIPURE of the original clauses. For each clauses, we assign an varible representing the clauses,e.g. $\mathcal{F}$=(a+b+..)(a'+b+..).., then c_1 represent (a+b+..), $c_2$represent(a+b+..). c_i is true iff at least one of literal is true, unassign iff c_i is not true and the literals in it are not all false. c_i is false means conflict happens need backtrack. Call them clause-variable.

Like procedure of assigning variable, if c_i is true push it into the vector clause_trail and for each decision level remember the size of clause_trail vector in another vector clause_trail_lim.

B. For each variable of positive and negative phase contruct a new vector that contains the clause-variable in positive phase if the the literal is contains in that clause,e.g.

$\mathcal{F}$=(a+b')(a'+b')(a+c)(a'+d).. then

a : | c_1 | c_3 | a': | c_2 | c_4 | b′: | c_1 | c_2 |

And there is a vector called lit_cla collect pointer to these kinds of vector storing the *clause-variable.*

C. For each literal MINIPURE, add one watch for it (in Chaff and MINISAT for each clause add two watches for unit propagation) ,the adding literal is clause-variable literal which is the first place of the vector descirbed in B. for literal a, c_1 is added , for literal a', c_2 is added.

Then watch(c1) returns a, b' while watch(c2) returns a' in the example of B. Here MINIPURE only do one clause-variable literal watch for clause-variable literal only has two state true

or unassigned. If c1 is assigned true check watch(c1) would get a, b' then check lit_cla[a] if c1=c3=true and a is unassigned implied by pure literal a' is true.

2) Algorithms:

A. *Pure literal detection:*

During the process of unit propagation, assume now propagating c' , after c' is propagated the original of MINISAT is to propagate the next literal in the unchecked queue if there is no conflict. Here MINIPURE change to see which clause-variable literal is assigned to be true after c' is true according to the vector lit_cla[c'].

For those clause-variable in lit_cla[c'] check whether it is unassigned or not, if unassigned set it to be true and push it into clause_trail , e.g. c': c1,c2,c3 and c1 is unassigned push it into clause_trail then check watch(c1) .Assume the propagation order is a,b,c' ,and watch(c1)=c',b,d only d is unassigned now check lit_cla[d]=(c1,c4,c5). For the clause-variable literals c4 and c5 if both are true , then by pure literal implication d' should be put into the uncheck queue, else without losing generality assume that c4 is unassigned, MINIPURE remove d from watch(c1) and push d into watch(c4).

Note that the implication of pure literal should only be done when it's unassigned otherwise some SAT problem would be UNSAT. If $\mathcal{F}$=(a+b')(a'+b) , then in decision level 1 push a in uncheck queue. a->c1->a', here c1->a' is implied by pure literal detection without checking whether variable is assigned or not which turns a SAT problem into UNSAT.

B. Pure literal clause learninig

When the conflict happens ,we need to backtrack and add a learning clause. In implication graph the clause-variable is implied to be true when one of its literal in their corresponding clause is true. This is a little different from the traditional unit propagation. However, these wouldn't affect the construction of implication just replaced the clause-variable literal with the original literal which implied it to be true. Literal b' is implied by c1,c2,c3 where b is contained by all of corresponding clauses c1,c2,c3 while c1 is put to clause_trail by literal x set true ,and c2 by y,c3 by z ,then fig.1 shows how it's reduced.

After construction the implication graph, MINIPURE can know which level to backtrack to. The remain problem is to unassign the clause-variable according to clause_trail_lim.

C. Detect Pure literal lazily

Since after restart again and again the variable order would change greatly, and MINIPURE wouldn't do pure literal detection all the time for the algorithm above should be done once every time a literal propagating. As a result, after restart for certain times(defined by user), MINIPURE would stop the pure literal detection. Also, restart the pure literal detection from time to time(defined by user) leaving the learnt clauses in the database to help us implied the pure literal while the pure literal detection mode is closed.

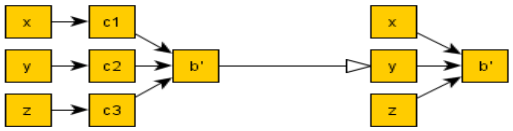

Fig 1

V. Implementation Details

1) The solver implement in language of C++.

2) The solver is modified from MINISAT 2.2.0 .

VI. SAT competition 2013 Specifics

1) The solver participate in SAT2013 competition of track 1,3,4,6,7,9,13, this solver is not joining MINISAT hack track for the difference is more than 1000 non-space characters.

2) The compiler is GCC4.5.2 with O3 optimization flag , 32bit.

3) Command line option: ./minipure <cnf file> -freq_pure=2 -stop_pure=0 -slowdown_pure=1 , and for other parameters the values are as default of MINISAT. Note that "=" must be placed.

VII. Availability

The code of MINIPURE is open and free for any who use for research and educational purpose. The license and code (including the license of MINISAT2.2.0) is available in [3].

VIII. Acknowledgement

The author thank to Jie-Hong Rolan Jiang's discussion about the implementation detail of MINIPURE and the idea of preprocessing with pure literal by Valeriy Balabanov.

IX. Refrerence

[1] M.W. Moskewicz, C.F. Madigan, Y.Zhao, L.Zhan,S. Malik **"Chaff: Engineering an Efficinet SAT Solver"** in Proc. of the 38th Design Automation Conference, 2001

[2] Niklas Sörensson, Niklas Een"**A SAT Solver with Conflict-Clause Minimzatin**",2005

[3] http://freakshare.com/files/pshpyy13/minipure.zip.html