Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions, volume B-2014-2 of Department of Computer Science Series of Publications B, University of Helsinki 2014. ISBN 978-951-51-0043-6.

ntusat and ntusatbreak

Hsiao-Lun Wang
National Taiwan University
Taipei, Taiwan
b98901096@ntu.edu.tw

Jie-Hong Roland. Jiang
National Taiwan University
Taipei, Taiwan
jhjiang@ntu.edu.tw

***Abstract*—This document is brief description of solver ntusat and ntusatbreak entering the SAT Competition 2014.**

I. Introduction

ntusat is based on Simpsat in SAT Competition 2012 which is an enhanced version of CRYPTOMINISAT 2.9.2 [1] with new technique for gaussian elimination described in [2]. ntusatbreak is ntusat with preprocessing technique described in [3] and [4] of SAT Competition 2013. The additional technique compared to Simpat in 2012 is pure literal detection and learning [5]. Adding pure literal implication clauses from time to time could make the theory not functional equivalent but reduce the search space to help the solver solve faster. The other technique is to find more XOR-clauses by improving XOR-recovery. If no XOR-clauses found , do the clause resolution procedure to find more information. The detail technique would be described in algorithm and datastructure part.

II. Main Techniques

ntusat and ntusatbreak are CDCL solver with simplification techniques subsumption, block literal, and inprocessing technique literal probing , hyper-binary resolution applied by CRYPTOMINISAT 2.9.2. Since there are XORclauses extraction and equivalent variable replacement , part of the pure literal detection technique is different from MINIPURE [5] last year. If any variable of literal β is in the XORclause , β would not be detected as pure literal. If literal α is replaced by literal β, then all the clause ID in vector litcontain [5] of α should be moved to litcontain(β). For other inprocessing techniques which causes variable elimination , clause subsumption and clause elimination, the clause watch and litcontain datastructure should also be checked.

III. Main Parameters

Most of parameters are the same as SIMPSAT in SAT Competition 2012. Since improved XOR-recovery from time to time is time consuming, NTUSAT do not apply that technique after 700 seconds. The pure literal detection and learning is applied every eight restarts. Emphirically, for some cases , adding to much pure literal implication clauses would make it harder to find solution for SAT cases. If the difference between max and min decision level of variables in pure literal implication clauses is less than 25 , the clauses is added to the database as irredundant clauses . Otherwise, the solver drop the clause.

Add-XORs(add new learnt xor for each restart)
input: Solver data: watch , original clauses ϕ
output: NEWXOR and LEARNTCLAUSE
begin
01 **for** each original and learnt clause C size¡5 and C not found
02 match-clauses := ϕ;
03 **for** each literal L in C
04 candidate-clauses := *watch*($\neg L$)+*watch*(L)
05 +*cache*($\neg L$)+*cache*(L);
06 match-clauses := match-clauses+
07 *check-all-literal*(candidate-clauses);
08 **end**
09 **if** *foundallxor*(C, match-clauses);
10 newxor.push(C);
11 **else**
12 learntclause:=*Getmoreinfo*(C,match-clauses);
13 **end**
end

Fig. 1. Algorithm: ADDXOR

IV. Special Algorithms, Data Structures, and Other Features

During the simplification procedure , NTUSAT would detect XORclauses by Figure 1 . If there is no XOR-clauses found, the solver would apply the resolution method to find more information. For example, the clause set Ω = { $(x_1 \vee x_2 \vee x_3)$, $(x_1 \vee \neg x_2 \vee x_3)$, $(x_1 \vee x_2 \vee \neg x_3)$, $(x_1 \vee \neg x_2 \vee \neg x_3)$ } these clauses can not form a XOR clause, but these clauses imply x_1=1 directly. The other clause set Γ= { $(x_1 \vee x_2 \vee \neg x_3)$, $(x_1 \vee \neg x_2 \vee x_3)$, $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$, $(\neg x_1 \vee \neg x_2 \vee x_3)$ } can neither form a XOR clause , but $x_2 = x_3$ can be implied from the set. The detail of the algorithm is in Figure 2. The solver generates clauses by resolution which would reduce the clause size by 1. In line 2, the solver implements this process until size is equal to three and in this loop the generating clauses would be binary clauses. In line 15, the vector candidate-learnt will store the binary clauses of the same variables, e.g. { $(\neg a \vee b)$, $(\neg a \vee \neg b)$ } would be push into candidate-learnt (a,b) ={ 1,0 } . $(\neg a, b)$ the sign of the variables in binary bits form is equal to 1 and $(\neg a, \neg b)$ is equal to 0.In line 22, k.size() means how many binary clauses with the same variables. k.size() equal to 4 means conflict, k.size( ) equal to 3 means the two variables are assigned . k.size() equal to 2 means the two variable are equivalent or one variable is assigned. The solver could get more information even there is no XOR-clauses found.

```
Getmoreinfo
  input: C, match-clauses
  output: learntclauses
begin
01  even:= φ, odd := φ, candidate_learnt := φ;
02  for sz= match-clauses.size down to 3
03    for each clause i in match-clauses and C
04    match-clauses.popup( ) ;
05      if xor all the literal in i is true
06        even.pushback(i);
07      else
08        odd.pushback(i);
09      end
10      for each i in even
11        for each j in odd
12        if sign of literals in i , j has only one difference
13          learnt:= resolve(i,j);
14          if sz equal 3
15            candidate-learnt[learnt].pushback(signs of learnt);
16          else
17            match-clauses.pushback(learnt);
18        end
19      end
20    end
21  for each k in candidate-learnt and k.size() > 0
22    learntclause.pushback(learn clauses according to k.size())
23  end
end
```

Fig. 2. Algorithm: GETMOREINFO

V. IMPLEMENTATION DETAILS

ntusat and ntusatbreak are implemented in C++. Both of them are based on SIMPSAT and CRYPTOMINISAT2.9.2. ntusatbreak applies saucy and breakID as preprocessor before solving.

VI. SAT COMPETITION 2013 SPECIFICS

Both of the solveres are submitted to hard combinatorial SAT and UNSAT , Application SAT and UNSAT , Random SAT and UNSAT. The compiler version is 4.7.1. , O3 used for compiling and 64-bit binary. There is no command-line option. All the parameters are locked in the solver. More information about parameters is in Main Parameters part.

VII. AVAILABILITY

Solver NTUSAT is available upon request for research purpose. Send email to the authors. For preprocessor of BreakID [3] and Saucy http://vlsicad.eecs.umich.edu/BK/SAUCY/

ACKNOWLEDGMENT

The author thanks to Paul T. Darga, Mark Lifton and Hadi Katebi for the symmetry detection tool Saucy and embedded breaking clause tool Shatter in Saucy 3.0. Also thanks to K.N.M. Soos who is the author of CRYPTOMINISAT series and the discussion with C.-S. Han for new ideas , who is the author of SIMPSAT.

REFERENCE

[1] K. N. M. Soos and C. Castelluccia., "Extending sat solvers tocryptographic problems." *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing*, pp. 244–257, 2009.
[2] C.-S. Han and J.-H. R. Jiang., "When boolean satisfiability meets gaussian elimination in a simplex way." *Proc.Int'l Conf. on Computer Aided Verification*, pp. 410–426, 2012.
[3] J. Devriendt and B. Bogaerts, "Shatterglucose and breakidglucose," *Proc. of SAT-13*, pp. 12–13, 2013.
[4] K. A. S. H. Katebi and I. L. Markov, "Graph symmetry detection and canonical labeling: Differences and synergies," *Proc. Turing-100*, 2012.
[5] H.-S. Wang, "minipure," *Proc. of SAT-13*, pp. 57–58, 2013.