Local Search for Boolean Relations on the Basis of Unit Propagation

Yakov Novikov
The United Institute of Informatics Problems, National Academy of Sciences of Belarus
Email:nov@newman.bas-net.by

Abstract

We propose a method for local search of Boolean relations relating variables of a CNF formula. The method is to branch on small subsets of the set of CNF variables and to analyze results of unit propagation. By taking into account variable value assignments deduced during the unit propagation procedure the method is able to justify any relation represented by a Boolean expression. The proposed technique is based on bitwise logical operations over ternary vectors. We implement a restricted version of the method used for unit clause derivation and equivalent-literal identification in a preprocessor engine for a SAT-solver. The experiments show that the proposed technique is useful for solving real-world instances of the formal verification domain.

1: Introduction

The satisfiability problem (SAT) attracts researchers due to its numerous applications and theoretical importance. In the last decade substantial progress has been made in the development of practical complete SAT algorithms [1,2,5,6,8,11-15,18]. All of them, except [5], are descendants of the DPLL-algorithm [7].

The DPLL-algorithm can be considered as a special case of general resolution that is called tree-like resolution. It was shown in [3] that there is an exponential gap between the performance of tree-like resolution and that of general resolution. Clause recording [2,13-15,18], restarts [1,10,13,14], and deduction of "strong" relations (unit clauses, equivalences and others) [5,6,16] are the steps made in the modern state-of-the-art SAT-solvers towards general resolution.

This paper belongs to the direction of research [9,19,20,23] meant to deduce and effectively use strong relations. We develop Le Berre's recent idea [19] to deduce relations by analyzing results of unit propagation. Our method consists of the following steps. Given a CNF, we select a few variables and branch on them performing unit propagation. On the basis of analysis of the values assigned to variables during unit propagation we deduce new relations relating the assigned variables. From a practical standpoint, an important feature of our method is the use of bitwise logical operations over ternary vectors that speeds up the calculations.

Our method outperforms Le Berre's one at least for the following three reasons. First, Le Berre's method can be viewed as a restricted version of our method with branching on two variables at most. (In the current implementation we limit the set of branching variables to five variables to cut down to 32 the length of the used ternary vectors). Second, our method is able to justify not only the unit clauses and literal equivalences as in Le Berre's method but also deduce any relation of Boolean variables. Third, our method is able to deduce the relations in the presence of don't-cares when the values of the considered variables are known partly.

The proposed method was implemented in a preprocessing engine named P_EQ. In the current implementation we identify unit clauses and literal equivalences only, because these relations are stronger and easier to check on. Besides, there is a great deal of such relations in the instances we consider. In this paper we compare the performance of BerkMin [14] with that of BerkMin enhanced by P_EQ on publicly available benchmarks from the formal verification domain. The experiments show that the proposed technique is useful. Almost all the obtained results are superior to the best known so far.

The paper is organized as follows. First we describe the basic idea. Second, we introduce inductive and deductive stages of our method. Then we outline a CNF formula traversing strategy meant to identify unit clauses and literal equivalence relations. Finally, we present experimental results.

2: Basic idea

Given a conjunctive normal form (CNF) F specified on the set of variables $\{x_1,\ldots,x_n\}$, the satisfiability problem (SAT) is to satisfy (set to *1*) all the disjunctions of F by some assignment of values to the variables from $\{x_1,\ldots,x_n\}$ or to prove that such a satisfying assignment called solution does not exist. A disjunction of variables

from $\{x_1,\ldots,x_n\}$ is also called a clause. A clause containing only one literal is called a unit clause.

A Boolean function g is called an implicate of a Boolean function f if $f \to g = 1$. Implicate g can be added to f conjunctively, i.e. $f = f \wedge g$. If g is an implicate of f no solution exists within the part of Boolean space specified by $\neg g$. Adding implicates to CNF F representing Boolean function f, especially adding short implicates (depending on a small number of variables), simplify solving SAT-instances.

Unit clauses (of the form x, $\neg x$), binary clauses ($x \wedge y$, $\neg x \wedge y$, $x \wedge \neg y$, $\neg x \wedge \neg y$), and equivalences ($x \leftrightarrow y$, $\neg x \leftrightarrow y$, $x \leftrightarrow \neg y$, $\neg x \leftrightarrow \neg y$) are known to be very useful kinds of implicates used to simplify solving SAT-instances that arise in formal verification and test generation domain [9,25-29]. Special relations (of the form $x \to (y \leftrightarrow z)$, $x \leftrightarrow y \leftrightarrow z$) were proven to be useful for solving parity-32 problem [6]. This list may grow longer in the future.

Consider CNF $F = (a \vee \neg b) \wedge (b \vee \neg c) \wedge (\neg a \vee \neg d) \wedge (d \vee \neg c)$. Suppose, a solution exists and contains $a = 0$. When making assignment $a = 0$, the first clause becomes unit consisting of single literal $\neg b$. We can satisfy the clause only by assigning $b = 0$. This assignment is called *deduced*. Then we can deduce $c = 0$ from the second clause. Such procedure of deducing assignments based on appearance of unit clauses in the formula is called *unit propagation* or Boolean constraint propagation. We can conclude for our example, if a solution exists and contains $a = 0$, then $c = 0$ is also in the solution.

Similarly, based on the third and fourth clauses we can deduce, if a solution exists and contains $a = 1$, then again $c = 0$ is in the solution. So, if F is satisfiable, then $c = 0$ is in any solution. Hence, $\neg c$ is an implicate of F. In other words, if branching on variable x of CNF F we deduce the same value for the same variable y, say $y = 0$, in both branches $x = 0$ and $x = 1$, then we can conclude that $\neg y$ is an implicate of F.

Now we are ready to extend this observation to the general case. Suppose, we have a partition of the Boolean space (where the CNF F is specified) and we know that some blocks of the partition (marked in dark in fig. 1) have no solution whereas a solution can exist in any other block of the partition (marked in white). Moreover, it is known, if a solution exists in a white block, it satisfies a Boolean relation $R(a,..,b)$. Then we can conclude that $R(a,\ldots,b)$ is an implicate of F.

We can divide the Boolean space into parts by branching on a subset of variables. For instance, by branching on *3* variables, one partitions the space into *8* blocks. By making assignments to branching variables and performing unit propagation we can deduce values of some variables $a,\ldots,b$ in each block of the partition. Suppose, unit propagation leads to appearance of complementary unit clauses x and $\neg x$ in F. Such situation is called a *conflict*. A conflict means that there is no

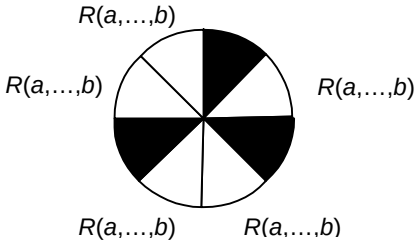

Fig. 1: Partition of Boolean space

solution in the block specified by the current assignment to branching variables. So, conflicting blocks can be marked in dark and conflict-free blocks can be marked in white (fig. 1). We come to the *basic idea*: if values of $a,\ldots,b$ satisfy a relation $R(a,..,b)$ in each conflict-free block of the partition, then $R(a,..,b)$ is an implicate of F.

In the case of two-block partitions, this idea leads to a "partial contraposition" of the basic rule $(A \to B) \wedge (A \to \neg B) \Rightarrow \neg A$ of Socrates algorithm [26], namely to $(\neg B \to \neg A) \wedge (B \to \neg A) \Rightarrow \neg A$. The case of four-block partitions was partly considered in [19]. The general case is closely related to Stalmark's method [16]. The difference is that Stalmark's method with its saturation procedure is global. When branching on variable x_1, it tries to branch on x_2, x_3, and so on in breadth first manner in each block of the partition determined by branching on x_1. In contrast, our method is local, and its locality allows one to arrange search for arbitrary relations in parallel and in the presence of don't-cares as shown below.

3: Inductive stage

Suppose, we are going to find relations of a type R (for example binary clauses or equivalences) relating variables from a subset Z of variables of CNF F. We call the set Z *observable*. Let Y be a subset of Z selected in a way. Suppose, we have a procedure which branches on variables from Y, performs unit propagations, and records values assigned to the observable variables. Let assigned values be stored in a ternary matrix $T(Z)$.

As an example, consider a ternary matrix (fig. 2). Here, the observable set is $Z = \{y_1,y_2,y_3,a,d,e,f,g\}$ and the set of branching variables is $Y = \{y_1,y_2,y_3\}$. Each column of the matrix corresponds to a branch in our procedure. For instance, column C_2 contains values assigned to observable variables on the branch $y_1 = 0$, $y_2 = 1$, $y_3 = 0$. Each row of the matrix corresponds to an observable variable.

Matrix $T(Z)$ is called *observable*. Let the observable matrix consist of dashes in the initial state, and our procedure specifies values of the matrix elements by substituting values *0* or *1* for dashes.

The part of the Boolean space determined by assignments $y_1 = 0$, $y_2 = 1$, $y_3 = 0$ is represented by cube

		C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	
		0	0	0	0	1	1	1	1	y_1
		0	0	1	1	0	0	1	1	y_2
		0	1	0	1	0	1	0	1	y_3
$T(Z)$	=	1	1	1	-	-	0	-	1	a
		0	0	-	-	1	-	-	-	d
		1	0	0	1	1	1	0	0	e
		0	0	-	0	-	1	1	-	f
		0	-	0	-	0	0	0	1	g
b	=	0	1	0	1	0	0	0	1	

Fig. 2: Observable matrix

a	b	$\neg a$	$a \wedge b$	$a \vee b$	$a \rightarrow b$	$a \leftrightarrow b$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1
0	-	1	0	-	1	-
-	0	-	0	-	-	-
1	-	0	-	1	-	-
-	1	-	-	1	1	-
-	-	-	-	-	-	-

Table 1: Logical operations

$\neg y_1\, y_2\, \neg y_3$. The column C_2 shows that if a solution exists in the cube $\neg y_1\, y_2\, \neg y_3$ then it contains $a = 1$, $e = 0$ and $g = 0$, but values of d and f are unknown (fig. 2).

Also consider a binary vector b which will be used by our procedure to record encountered conflicts as follows. The i-th component b_i of b is equal to 1, if making the assignment to the variables of Y in accordance with i-th column of the observable matrix leads to a conflict. Otherwise $b_i = 0$. In our example, $b_3 = 1$ as a consequence of a conflict imposed by assignments $y_1 = 0$, $y_2 = 1$, $y_3 = 1$ (fig. 2). Vector b is called a *vector of conflicts*.

The procedure filling in the observable matrix and the vector of conflicts implements an inductive stage of the method. As a prototype of the procedure, we can use the DPLL algorithm [7] slightly modified to record conflicts and value assignments as described above.

4: Deductive stage

Consider the column C_2 of the observable matrix represented on fig. 2. If a solution exists in the cube $\neg y_1 y_2 \neg y_3$, the value of variable d is unknown in the solution. To clarify the situation we have to branch within the cube $\neg y_1 y_2 \neg y_3$. It is important to underscore that we can solve the problem of insufficient information in spite of partial uncertainty of the observable matrix.

Consider table 1 describing commonly used logical operations. We see that for some operations the amount of uncertainty is reduced. For example, 1 dominates 0 for the disjunction operation, which means $1 = - \vee 1 = 1 \vee -$. Similarly, relation $R = (\neg a \rightarrow d) \vee f$ holds in the cube $\neg y_1 y_2 \neg y_3$ corresponding to column 2, since $a = 1$, $d = f = -$ in the column, and $(\neg 1 \rightarrow -) \vee - = 1$ (fig. 2). We can see that values of a, d, f satisfy the relation R for each column marked by 0 in the vector of conflicts b. So, R is an implicate of F.

Now, we transform this observation into a vector form. Denote by t^w the row that correspondents to the observable variable w in the observable matrix. Let R be a logical expression connecting observable variables. For our example $R = (\neg a \rightarrow d) \vee f$. Replacing each observable variable w with corresponding ternary vector t^w in the expression we form the **vector** expression R^* where the logical operations over vectors are **bitwise**. We have $R^* = (\neg t^a \rightarrow t^d) \vee t^f$ in our example.

Theorem 1. $R \Leftarrow F$ if $R^* \vee b = 1^*$.

Here F is the initial CNF, $\Leftarrow$ denotes $R \leftarrow F = 1$, b is a binary vector recording conflicts, 1^* is the vector containing only 1 values and having the same length as b.

As a consequence we have

$x \Leftarrow F$ if $t^x \vee b = 1^*$,

$x \leftrightarrow y \Leftarrow F$ if $(t^x \leftrightarrow t^y) \vee b = 1^*$,

$x \vee \neg y \Leftarrow F$ if $t^x \vee \neg t^y \vee b = 1^*$, and so on.

It is easy to check that $\neg t^g \vee b = 1^*$, $(t^{y_2} \leftrightarrow \neg t^e) \vee b = 1^*$, $(\neg t^a \rightarrow t^d) \vee t^f \vee b = 1^*$ for the observable matrix $T(Z)$ (fig. 2). The first expression means that the unit clause $\neg g$ can be added to the CNF F for which the matrix $T(Z)$ was constructed. The second expression shows that literals y_2 and $\neg e$ (as well as $\neg y_2$ and e) are equivalent and can be identified in F. The third expression means that $R = (\neg a \rightarrow d) \vee f$ is implied by F.

Note that we have got the equivalence $y_2 \leftrightarrow \neg e$ due to the fact that $b_1 = b_3 = 1$ in spite of $y_2 \neq \neg e$ in columns C_1, C_3. So, conflicts give a "masking" effect, which can be used to resolve situations known as "false negatives" and "impossibilities to propagate inequality to primary outputs" in equivalence checking [28] (we omit the discussion of the question for lack of space).

Note that we proved some Boolean relations in spite of a number of uncertainties in the observable matrix. Nevertheless, to exploit Theorem 1 it is necessary to eliminate uncertainty completely, forming vector 1^* when expression R^* is calculated. Consider how the last requirement can be relaxed and how relations can be deduced without eliminating the uncertainty completely.

Let h be a conjunction depending on variables of the set Y. Suppose $h = \neg y_1 y_2$. Since $h = \neg y_1\, y_2\, \neg y_3 \vee \neg y_1\, y_2\, y_3$, we can say that h corresponds to columns C_2, C_3 (fig. 2). Consider a ternary vector $\neg h^*$ that differs from the Boolean vector 1^* only in that it can take any value from the set $\{0,-,1\}$ in the components which correspond to the conjunction h.

Theorem 2. $h \vee R \Leftarrow F$ if $R^* \vee b = \neg h^*$.

The theorem can be interpreted as follows. The cube

h "masks" the white blocks where R doesn't hold (fig. 1).

It is easy to see that it is impossible to use theorem 1 to prove that $e \vee f \Leftarrow F$ for our example, since the component of $t = t^e \vee t^f \vee b$ corresponding to column C_2 is equal to -. At the same time, according to theorem 2 expression $h \vee e \vee f \Leftarrow F$ holds where $h = \neg y_1 y_2 \neg y_3$.

Deducing a strong relation like unit clause or literal equivalence allows one to reduce search space by a factor of 2. Note that relations like $h \vee R$ where h is a conjunction can be strong as well. For instance, the relation $\neg y_1 y_2 \neg y_3 \vee f$ allows one to reduce search space by 1/2 - 1/8 = 3/8. Such relations are attractive from the viewpoint of unit propagation. Indeed, when $R=0$ we have to set conjunction h to *1* by assigning appropriate values to all its variables. So, given relation $\neg y_1 y_2 \neg y_3 \vee f$ and $f = 0$, we deduce value assignments $y_1 = 0$, $y_2 = 1$, $y_3 = 0$.

5: Formula traversing strategy

Strong relations usually simplify testing the satisfiability of large real-life CNF formulas, especially when such relations are in abundance. Suppose it is known that a large number of relations can be deduced by branching on subsets that consist of two variables. If the number of variables in a formula is large (at present, "large" means up to a few millions), then even a simple quadratic algorithm examining all pairs of variables is impractical. Besides, deduction of relations is order dependent, that is deducing of some relations may make it much easier to deduce others. So, an efficient strategy of formula traversing is necessary to select subsets of variables to branch on.

In this paper, we simulate the well-known strategy used by commercial BDD based tools for equivalence checking of combinative circuits. In accordance with the strategy, equivalence relations between intermediate points of the compared circuit are propagated from inputs to outputs. The strategy can be explained by the following example.

Consider two AND gates with identical input variables: $u = ab$, $v = ab$. A consistent value assignment to the variables of the first gate has to satisfy CNF $(\neg a \vee \neg b \vee u) \wedge (a \vee \neg u) \wedge (b \vee \neg u)$. A consistent value assignment to the variables of the two gates has to satisfy CNF $N = (\neg a \vee \neg b \vee u) \wedge (a \vee \neg u) \wedge (b \vee \neg u) \wedge (\neg a \vee \neg b \vee v) \wedge (a \vee \neg v) \wedge (b \vee \neg v)$. Let us assume that the modified DPLL procedure is used and it branches on the set of variables $V = \{ a, b \}$. It is easy to see that the method proposed above allows one to prove $u \leftrightarrow v \Leftarrow F$. The result will not change, if the gates have different input variables, for instance a, b and a', b', but $a \leftrightarrow a'$ and $b \leftrightarrow b'$. So, branching on the input variables of a gate it is possible to propagate an equivalence relation from the gate input to its output.

Suppose, it is unknown which variables of $\{a,b,u\}$ are input ones for the considered AND gate. In this case, we can deduce the equivalence relation $u \leftrightarrow v$ branching on the longest clause from the CNF description of the gate, namely $\neg a \vee \neg b \vee u$, since the clause contains all the variables relating to the gate.

We have to use this heuristic since later on we will consider the situation when one has to deal with a CNF formula describing some verification problem without knowledge of the structure of the underlying circuit. We assume that variables of the CNF are correctly numbered i. e. if a node described by variable x_i has a lower level number than a node described by variable x_j , then $i < j$. The level of a node in a circuit is equal to the maximal length of a path leading to the node from an input of the circuit. (In other words, variable numbering follows the topological ordering of nodes.) We also assume that CNF description follows the topological ordering as well. That is if clauses specifying gate g are located closer to the beginning of the formula then clauses specifying gate g' then the level of gate g' cannot be smaller than that of g. Such assumptions are not necessarily true for publicly available benchmarks. But even if the topological ordering is preserved partially it may allow one to deduce a lot of strong relations.

We traverse the formula twice. To perform the first traversal we mark a set of variables having the smallest numbers. Ideally, input variables should be marked only (but in our experiments input variables were unknown and the first *150* variables were marked). Then the set of variables is examined in the ascending order of variable numbers. For each variable x_i we select the clauses having size $l \in \{3,4,5,6\}$ and containing x_i itself. If the clause contains l-1 marked variables, the modified DPLL procedure is applied to deduce strong relations by branching on the marked variables. If the equivalence of two literals is proven, the literals are marked. In the second traversal we examine the formula from the beginning to the end and the clauses of length l are picked again. But now we branch on all variables of the clause except the case when $l = 6$. In this case we branch on the first *5* variables of the clause. If a unit clause is deduced, unit propagation procedure is run to deduce the additional unit clauses. Periodically, the discovered equivalent literals are identified i.e. all literals of one variable are replaced with the corresponding literals of the equivalent variable.

State-of-the-art algorithms of combinational equivalence checking [9,28,29] make full use of structural properties of processed circuits. The procedure described in this section uses structure partially and indirectly. Our purpose is only to demonstrate the potential of the technique proposed in the paper. We can see from the next section that new technique allows one to improve performance of SAT-tools significantly on some well known benchmarks from formal verification domain.

6: Experimental results

The proposed method was implemented in a preprocessing engine P_EQ for SAT-solver BerkMin [14]. Both programs were written in Microsoft's Visual C++ under Windows-95. The experiments were run on a computer with Pentium-III-700 and 640Mb of RAM. We considered all instances from benchmark class Joao [22] which encodes equivalence checking of combinational circuits, and the hardest instances from class FVP-UNSAT2.0 [17] describing verification of pipelined microprocessors, and from the BMC benchmark class encoding bounded model checking problems [4].

The results of the experiments are shown in Tables 2-3. Names of instances are given in the column "Name". The column "BerkMin" describes BerkMin's performance. The column "eq_lit" gives the number of deduced pairs of equivalent literals. Note that since the equivalent literals were not immediately identified, the number of deduced pairs of equivalent literals forming an equivalence class may exceed the number of elements in this class. The column "Unit" gives the number of derived unit clauses. The column P_EQ+BerkMin describes the performance of BerkMin enhanced by P_EQ. The column "the best at SAT-EX" shows the best results among 23 sat-solvers tested by L. Simon at the web site SAT-EX [21] (class FVP-UNSAT2.0 has not been tested there).

Table 3 shows that using P_EQ substantially reduces runtimes on the instances of classes Barrel, Longmult and Joao in comparison with running BerkMin alone. The explanation is that a lot of strong relations were discovered for these instances. Frequently, the instances were solved by P_EQ alone. Note that using P_EQ for satisfiable instances (marked by suffix "bug") does not give any speed-up. The instances were solved quickly by BerkMin without any preprocessing.

Instances c6288, c6288_s, c3540_s, c5315, c5315_s, c7552, c7552_s from the class Joao are classified as quasi-challenging for current Sat-solvers [21]. Each of the instances is solved by at most two SAT-solvers tested at the site SAT-EX (within the timeout limit of 10000 s.). On the other hand, after applying P_EQ these instances are easily solved by BerkMin.

When solving the instances encoding microprocessor verification problems, P_EQ revealed few strong relations. Nevertheless, using P_EQ allows BerkMin to solve the hardest instances of the class FVP-UNSAT2.0 faster (except for one instance). Note that this class is hard for current SAT-solvers [14,24]. Web site [30] says that state-of-the-art SAT-solver Zchaff [13] solved instances 6pipe and 7pipe in 18,439.4 s. and 54,928.9 s., correspondingly, on SUNW, Ultra-80 system with clock frequency of 450MHz. We can see from Table 2 that BerkMin solved the instances much faster, and using P_EQ reduces runtime for the instances nearly two times.

Name	BerkMin	P_EQ			P_EQ+ BerkMin
	Time	Eq_lit	Units	Time	Time
5pipe_4	294.1	112	806	29.2	**226.3**
6pipe	701.7	72	1763	89.7	**455.9**
6pipe_6_	**481.9**	212	1523	84.1	507.6
7pipe	2237.5	76	3067	220.8	**1201.5**
7pipe_bg	479.5	72	2987	214.8	**243.9**
Total	4194.7			638.6	**2635.2**

Table 2: Representatives of class 8FVP-UNSAT2.0

The comparison of the results given in columns "P_EQ+BerkMin" and "the best at site SAT-EX" shows that enhancing BerkMin by P_EQ allows one to solve the instances considered in this paper a few times faster (up two orders of magnitude) than using the best program listed at the web site for each instance (with the exception of easy satisfiable instances for which results are comparable).

References

1. L.Baptista, J.P.Marques-Silva. The interplay of randomization and learning on real-world instances of satisfiability//Proceedings of AAAI Workshop on Leveraging Probability and Uncertainty in Computation. - July 2000.
2. R.J.J.Bayardo, R.C. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT Instances// Proceeding of the Fourteenth National Conference on Artificial Intelligence (AAAI'97), Providence,Rhode Island. –1997. -P. 203-208.
3. E.Ben-Sasson, R. Impagliazzo, A.Wigderson. Near optimal separation of Treelike and General resolution// Proceedings of SAT-2000: Third Workshop on the Satisfiability Problem.-May 2000.-P.14-18.
4. A. Biere et al. Symbolic model checking using SAT procedures instead of BDDs// Proceedings of DAC'99. -1999.
5. J.F.Groote, J.P.Warners. The propositional formula checker HeerHugo//SAT2000. I.Gent et al. IOS Press. -2000. -P.261-281.
6. C.-M.Li. Integrating Equivalency reasoning into Davis-Putnam procedure//Proceedings of AAAI'2000. Austin.Texas. USA. -2000. -P. 291-296.
7. M.Davis, G.Longemann, D.Loveland. A Machine program for theorem proving//Comminications of the ACM. -1962. -V.5. -P.394-397.
8. O.Dubois, P.Andre, Y. Boufkhad, J.Carlier. SAT versus UNSAT//Johnson and Trick, Second DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society. –1996. -P.415-436.
9. E. Goldberg, M.Prasad. Using Sat for combinational equivalence checking// Proceedings of DATE'01. -P.114-121.
10. C.P.Gomes, B. Selman, H.Kautz. Boosting combinational search through randomization// Proceedings of International Conference on Principles and Practice of Constraint Programming. - 1997.
11. J.W. Freeman. Improvements to propositional satisfiability search algorithms//Ph.D. thesis, Department of computer and Information science, University of Pennsylvania, Philadelphia. - 1995.
12. C.M.Li. A constrained-based approach to narrow search for Satisfiability // Information processing letters.-1999. -V. 71. -P. 75-80.
13. M.W. Moskewicz et all. Chaff: Engineering an Efficient SAT Solver// Proceedings of DAC'01. -2001.
14. E. Goldberg, Ya. Novikov. BerkMin: a Fast and Robust SAT-solver// Proceedings of DATE'02. –P.142-147.
15. J.P.M.Silva, K.A.Sakallah. GRASP: A Search Algorithm for

Propositional Satisfiability//IEEE Trans. Comp.-1999.-V.48.-P.506-521.

16. G. Stalmarck. A Tutorial on Stalmarck's Proof Procedure for Propositional Logic//Formal Methods in System Design. -2000.-V.13. –P.23-58.

17. M.Velev. CMU benchmark suite. Available from http://www.ece.cmu.edu/~mvelev.

18. H.Zhang. SATO: An efficient propositional prover// Proceedings of the International Conference on Automated Deduction. -July 1997. -P.272-275.

19. D.L.Berre. Exploiting the real power of unit propagation look ahead//Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT2001).

20. J.P.M. Silva. Algebraic simplification techniques for propositional satisfiability//International Conference on Principles and Practice of Constraint Programming. -Sept. 2000. -P. 537-542.

21. http://www.lri.fr/~simon/satex/satex.php3.

22. ftp://algos.inesc.pt/pub/benchmarks/cnf/equiv-checking /MITERS.

23. I. Lynce, J.P.M.Silva. The Interaction Between Simplification and Search in Propositional Satisfiability // CP'01 Workshop on Modeling and Problem Formulation (Formal'01), November. 2001.

24. M.N. Velev, R.E. Bryant. Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors // Proceeding of DAC'01. -June 2001.-P.226-231.

25. W. Kunz, D.K. Pradhan. Recursive learning: A new implication technique for efficient solution to CAD-problems: Test, verification and optimization // IEEE Trans. CAD. -Vol.13. -No.9. -1994.

26. M.H. Schulz, E. Trischler, T.M. Sarfert. SOCRATES: A highly efficient automatic test pattern generation system // IEEE Trans. CAD. -Vol. 7. -Jun.1988. -P.126-137.

27. P. Stephan, R.K. Brayton, A.L. Sangiovanni-Vincentelli. Combinational test generation using satisfiability // IEEE Trans. CAD. –Vol. 15. –No. 9. -Sept. 1996. –P.1167-1176.

28. D. Brand. Verification of large synthesized designs // Digest of Technical Papers of the IEEE/ACM International conference on Computer-Aided Design, Santa Clara, CA. -Nov. 1993. –P.534-537.

29. A.Kuehlmann et al. Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification // IEEE Trans. CAD. –No.12. –Des. 2002.

30. http://eigold.tripod.com/BerkMin.html.

Name	BerkMin	P_EQ			P_EQ+ BerkMin	The best at SAT-EX	
	Time	Eq_lit	Unit	Time	Time	Sat-solver	Time*
Representatives of BMC subclass Barrel							
Barrel5	2.09	1962	464	0.06	**0.06**	*Eqsatz*	0.67
Barrel6	15.17	2620	658	0.16	**0.16**	*Eqsatz*	1.39
Barrel7	80.3	5256	893	0.22	**0.22**	*Eqsatz*	1.82
Barrel8	480.87	7768	1301	0.28	**0.28**	*Satz-215*	2.72
Barrel9	378.76	13952	1870	0.55	**0.55**	*Eqsatz*	6.13
Representatives of BMC subclass Longmult							
Lngmlt9	99.20	4016	746	3.63	**64.48**	*Posit*	237.78
Lngmlt10	135.60	4558	670	5.22	**78.17**	*Posit*	347.20
Lngmlt11	195.04	5126	731	8.48	**58.11**	*Posit*	479.73
Lngmlt12	256.96	5720	706	12.30	**50.81**	*Zchaff*	602.05
Lngmlt13	296.10	6376	752	17.58	**33.01**	*Posit*	731.39
Lngmlt14	374.50	7090	862	24.06	**34.32**	*Zchaff*	682.91
Lngmlt15	270.01	7798	1027	31.97	**33.18**	*Heerhugo*	215.27
Class Joao							
C1355	0.77	1002	1	0.17	**0.28**	*Heerhugo*	0.86
C1355_s	0.6	772	380	0.05	**0.05**	*Heerhugo*	0.87
C1908	1.75	2836	81	0.28	**0.33**	*Zchaff*	11.74
C1908_bug	1.15	2838	81	0.28	0.33	*N_tab*	**0.16**
C1908_s	1.87	2856	81	0.27	**0.27**	*Zchaff*	13.36
C2670	1.75	3408	3111	0.22	**0.49**	*Zchaff*	6.15
C2670_bug	**0.06**	3338	294	0.22	0.33	*N_sat*	0.09
C2670_s	2.37	3800	367	0.17	**0.17**	*Zchaff*	6.17
C3540	30.54	4394	25	0.82	**1.7**	*Zchaff*	130.96
C3540_bug	0.22	4392	25	0.76	1.09	*Zchaff*	**0.14**
C3540_s	34.99	5000	47	0.77	**0.83**	*Zchaff*	202.44
C490	0.38	448	1	0.05	**0.11**	*Eqsatz*	0.18
C490_s	0.50	160	367	0.05	**0.05**	*Eqsatz*	0.14
C432	0.05	370	10	0.0	**0.0**	*Zchaff*	0.36
C432_s	0.06	372	10	0.0	**0.0**	*Zchaff*	0.26
C5315	20.21	6868	421	0.77	**1.59**	*Zchaff*	121.92
C5315_bug	0.71	6632	324	0.71	1.92	*N_sat*	**0.17**
C5315_s	22.91	6378	649	0.61	**0.61**	*Zchaff*	108.75
C6288	>5000	3082	111	0.16	**0.16**	*Heerhugo*	2.60
C6288_s	>5000	3082	108	0.17	**0.17**	*Heerhugo*	2.57
C7552	100.35	10422	303	1.32	**5.77**	*Zchaff*	243.78
C7552_bug	0.38	10154	304	1.37	4.33	*N_sat*	**0.24**
C7552_s	144.73	9596	519	0.87	**0.87**	*Zchaff*	380.95
C880	0.38	0	148	0.0	0.0	*Zchaff*	4.47
C880_s	0.38	0	148	0.0	0.0	*Zchaff*	4.45

*) Experiments were run on PII-400 under Linux [21].

Table 3: Representatives of BMC and Miters classes