How Good Can a Resolution Based SAT-Solver Be?

Eugene Goldberg[1], Yakov Novikov[2]

[1] Cadence Berkeley Labs, USA, egold@cadence.com
[2] The United Institute of Informatics Problems, National Academy of Sciences, Belarus, nov@newman.bas-net.by

Abstract. We introduce a parameterized class $M(p)$ of unsatisfiable formulas that specify equivalence checking of Boolean circuits. If the parameter p is fixed, a formula of $M(p)$ can be solved in general resolution in a linear number of resolutions. On the other hand, even though there is a polynomial time deterministic algorithm that solves formulas from $M(p)$, the order of the polynomial is a monotone increasing function of parameter p. We give reasons why resolution based SAT-algorithms should have poor performance on this very "easy" class of formulas and provide experimental evidence that this is indeed the case.

1 Introduction

In the long run, studying the complexity of proofs in various proof systems of propositional logic is aimed at establishing lower bounds on performance of deterministic algorithms for solving SAT. The first result on complexity of resolution proofs was obtained by Tseitin in 1966 [12]. He proved exponential lower bounds for proofs in a restricted version of general resolution called regular resolution. Two decades later Haken established exponential lower bounds for general resolution [7]. Some proof systems (like extended resolution) are so powerful that all attempts to find a class of formulas that admit only proofs of exponential length have failed so far. For that reason, along with trying to break non-deterministic proof systems, a new line of research has started that studies the automatizability of proof systems [4]. A proof system P is automatizable if there is a deterministic algorithm that for any unsatisfiable CNF formula F finds a proof whose length is bounded by a polynomial of the length of the shortest proof in P (that establishes the unsatisfiability of F). In [9] it was shown that general resolution is non-automatizable under an assumption.

In this paper, we introduce a parameterized class $M(p)$ of CNF formulas. This class should be very helpful in studying the complexity of finding short resolution proofs by a deterministic algorithm, that is in studying the automatizability of resolution. A formula F of $M(p)$ represents an instance of checking the equivalence of two Boolean circuits N_1,N_2 obtained from a common "specification" S. Here, by specification we mean a circuit consisting of multi-valued gates and so computing a multi-valued function of multi-valued arguments. (A

multi-valued gate will be further referred to as a block). Boolean circuits N_1,N_2 are obtained from S by encoding values of variables of S with binary codes. After encoding the values of the input and output variables of a block G, the obtained multi-output Boolean function can be implemented by a Boolean sub-circuit. Circuits N_1,N_2 are obtained from S by replacing each block G of S with Boolean sub-circuits $I_1(G)$ and $I_2(G)$ respectively. The formula F is satisfiable if and only if N_1 and N_2 are not equivalent. The parameter p bounds the number of gates in implementations of a block G of S. (I.e. each implementation of a block has at most p gates.)

We show that the unsatisfiability of a CNF formula F of $M(p)$ describing equivalence checking of circuits N_1 and N_2 can be proven in general resolution in no more than $d * n * 3^{6p}$ resolution steps. Here, d is a constant and n is the number of blocks in a common specification S of N_1 and N_2. So, if the value of p is fixed, the formulas of $M(p)$ have linear size proofs in the formula length. (We will further refer to these proofs as specification guided ones.) On the other hand, the maximal length (number of literals) of resolvents one has to deduce in those proofs is bounded by $6 * p$. So if p is fixed, the length of resolvents is bounded by a constant and there is a trivial deterministic algorithm that solves all the formulas from $M(p)$ in polynomial time. This algorithm just derives all the possible resolvents whose length does not exceed a threshold. If no empty clause is derived the algorithm just increases the threshold by 1 and repeats derivation of resolvents. We show that given a value r, there is always p' such that in a specification guided proof of the unsatisfiability of a formula from $M(p')$ one has to produce a clause of length r. This means (see Section 4) that the order of the polynomial bounding the runtime of this trivial deterministic algorithm is a monotonic increasing function of p. So the gap between the complexity of nondeterministic proofs and that of this trivial deterministic algorithm widens as the value of p grows.

Since formulas of $M(p)$ have linear complexity in general resolution and appear to be hard for deterministic algorithms they can be used to gauge the performance of resolution based SAT-solvers. Of course, in this paper we just show that formulas of $M(p)$ are hard (from a practical point of view) for a trivial deterministic algorithm. However, there is a reason to believe that these formulas are hard for any deterministic algorithm. Let us assume that specification guided resolution proofs are "significantly" shorter (at least for some classes of specifications) than any other resolution proofs. On the one hand, a specification guided proof of the unsatisfiability of a formula F from $M(p)$ describing equivalence checking of N_1 and N_2 closely follows the topology of a common specification S of N_1 and N_2. So knowing a short proof of unsatisfiability for F one could recover S from N_1 and N_2 . On the other hand, the problem of finding a common specification of N_1 and N_2 appears to be hard (most likely NP-hard).

In this paper we give some experimental evidence that the class $M(p)$ is indeed hard for existing SAT-solvers. Namely, we test the performance of state-of-the-art SAT-solvers 2clseq [1], Zchaff [8], BerkMin [6] and show that their performance quickly degrades as the size of parameter p grows.

2 Formal definition of class $M(p)$

In this section, we specify the class of formulas we consider in this paper.

2.1 Definition of Specification and Implementation

In this subsection, we introduce the notion of a specification and its implementation that play a key role in the following exposition.

Let S be a combinational circuit of multi-valued gates specified by a directed acyclic graph H. The circuit S will be further referred to as a ***specification***. The sources and sinks of H correspond to ***primary inputs and outputs*** of S. Each internal node of H corresponds to a multi-valued gate further referred to as ***a block***. A block is a device computing a multi-valued function of multi-valued arguments (like a "regular" gate computes a Boolean function of Boolean arguments.) Let n_1 and n_2 be nodes of H connected by an edge directed from n_1 to n_2. Then in the specification S the output of the block G_1 corresponding to n_1 (or the primary input corresponding to n_1 if n_1 is a source of H) is connected to an input of the block G_2 corresponding to n_2. Each node n of H is associated with ***a multi-valued variable*** V. If n is a source (respectively a sink) of H then the corresponding variable specifies values taken by the corresponding primary input (respectively primary output) of S. If n is an internal node of H (i.e. it is neither a source nor a sink) than the corresponding variable (also called an ***internal variable***) specifies values taken by the output of the block specified by n. Let G be the block of S specified by a node n of H. We will use the notation $C = G(A, B)$ to indicate that a) the output of the block G is associated with a variable C; b) the function computed by the block G is $G(A, B)$; c) only two nodes of H are connected to the node n in H by edges directed to n and these nodes are associated with variables A and B.

Denote by $D(A)$ the ***domain*** of the variable A associated with a node of H. The number of values taken by A i.e. the value $|D(A)|$ is called the ***multiplicity*** of A. If the multiplicity of each variable A associated with a node of H is equal to 2 then S is a ***Boolean circuit***.

Henceforth, by a variable A of a specification S we will mean the variable associated with a node of the graph H. Let $D(A) = \{a_1, \ldots, a_t\}$ be the domain of a variable A of S. Denote by $q(A)$ ***a Boolean encoding*** of the values of $D(A)$ that is a mapping $q : D(A) \to \{0,1\}^m$. Denote by $length(q(A))$ the number of bits in the encoding q (that is the value of m). The value $q(a_i)$, $a_i \in D(A)$ is called the ***code*** of a_i. Given an encoding q of length m of a variable A, denote by $v(A)$ the set of all m ***coding Boolean variables*** that is the variables specifying the Boolean space of codes for the values of A.

In the following exposition we make the assumptions below.

Assumption 1 *Each block of a multi-valued and each gate of a Boolean circuit has two inputs and one output.*

Assumption 2 *For each variable A of any specification, $|D(A)|$ is a power of 2.*

Assumption 3 *For each variable A of any specification S, a minimal length encoding is used. That is $length(q(A)) = log_2(|D(A)|)$.*

Assumption 4 *Any two different values of a variable A of S have different codes. That is if $a_i, a_j \in D(A)$ and $a_i \neq a_j$ then $q(a_i) \neq q(a_j)$.*

Remark 1. From Assumption 2 and Assumption 3 it follows that for each variable A of any specification, any encoding $q : D(A) \rightarrow \{0,1\}^m$ is bijective. In particular, any assignment to the variables of $v(A)$ is a code of some $a_i \in D(A)$.

Now we describe how a Boolean circuit N is obtained from a specification S.

Definition 1. *Denote by $Inp(I)$ (respectively $Out(I)$) the set of primary input (respectively primary output) variables of a Boolean circuit I.*

Definition 2. *Let X_1 and X_2 be sets of Boolean variables and $X_2 \subseteq X_1$. Let y be an assignment to the variables of X_1. Denote by $proj(y, X_2)$* ***the projection*** *of y on X_2 i.e. the part of y that consists of assignments to the variables of X_2.*

Definition 3. *Let $C = G(A, B)$ be a block of specification S. Let $q(A)$,$q(B)$,$q(C)$ be encodings of variables A,B, and C respectively. A Boolean circuit I is said to* ***implement the block G*** *if the following three conditions hold:*

1. *The set $Inp(I)$ is a subset of $v(A) \cup v(B)$.*
2. *The set $Out(I)$ is equal to $v(C)$.*
3. *If the set of values assigned to $v(A)$ and $v(B)$ form codes $q(a)$ and $q(b)$ respectively where $a \in D(A)$, $b \in D(B)$, then $I(z') = q(c)$ where $c = G(a, b)$. Here, z' is the projection of assignment $z = q(a), q(b)$ on $Inp(I)$ and $I(z')$ is the value taken by I at z'.*

Remark 2. If $Inp(I) = v(A) \cup v(B)$, then the third condition above just says $I(q(a), q(b)) = q(c)$. However, in general, one can implement a block G of S with a circuit having fewer input variables than $|v(A)| + |v(B)|$ (because the output of the block G may take the same value for different assignments to variables A,B). This is why we used the notion of projection to define block implementation.

Definition 4. *Let S be a specification. A Boolean circuit N is said to* ***implement a specification S****, if it is built according to the following two rules.*

1. *Each block G of S is replaced with an implementation I of G.*
2. *Let the output of block $C = G_1(A, B)$ of S be connected to the input C of a block $G_2(C, D)$. Then the outputs of the circuit I_1 implementing G_1 are properly connected to inputs of circuit I_2 implementing G_2. Namely, the primary output of I_1 specified by a Boolean variable $v_i \in v(C)$ is connected to the input of I_2 specified by the same variable, if $v_i \in Inp(I_2)$.*

Let N be an implementation of a specification S. From Remark 1 it follows that for any value assignment y to the primary input variables of N there is a unique set of values $(x_1,\ldots,x_k)$, where $x_i \in D(X_i)$ such that y=$(q(x_1),\ldots,q(x_k))$. In other words there is one-to-one correspondence between assignments to primary inputs of S and N.

Definition 5. *Let N be an implementation of S. Given a Boolean vector y of assignments to the primary inputs of N, the corresponding vector $Y = (x_1, \ldots, x_k)$ such that $y = (q(x_1), \ldots, q(x_k))$ is called the* ***pre-image*** *of y.*

Definition 6. *The* ***topological level*** *of a block G in a circuit S is the length of the longest path going from a primary input of S to G. (The length of a path is measured in the number of blocks on it. So, the topological level of a primary input is 0.) Denote by* ***level(G)*** *the topological level of G in S.*

Proposition 1. *Let N be a circuit implementing specification S. Let y be a value assignment to the primary input variables of N and Y be the pre-image of y. Then, for any variable C of S, the values assigned to variables $v(C)$ in N form the code $q(c)$ where c is the value taken by variable C when the inputs of S take the values specified by Y.*

Proof. The proposition can be proven by induction in topological levels of variables of the specification S.

Base step. The correctness of the theorem for the variables with topological level 0 (that is primary input variables of S) follows from Remark 1.

Induction step. Let C be a variable of S and $level(C) = n$, $n > 1$. The induction hypothesis is that the proposition holds for all the variables of S with topological level less than n. Let $G(A, B)$ be the block of S whose output is associated with C. Let $I(G)$ be the implementation of G in N. Obviously, the topological level of A and B is less than n. Then under the assignment y to the primary inputs of N the variables $v(A)$ and $v(B)$ take values $q(a)$ and $q(b)$. Here, a and b are the values taken by A and B under the assignment Y to primary inputs of S. Then from Definition 3 it follows that the value taken by the outputs of $I(G)$ is $q(c)$ where $c = G(a, b)$. □

Proposition 2. *Let N_1, N_2 be circuits implementing a specification S. Let each primary input (or output) variable Q have the same encoding in N_1 and N_2. Then Boolean circuits N_1 and N_2 are functionally equivalent.*

Proof. Let y be an arbitrary assignment of values to the variables of N_1 and N_2. Let us prove that all the corresponding primary outputs of N_1 and N_2 take identical values. Denote by Y_1 and Y_2 the pre-images of y with respect to encodings of variables of S used when obtaining implementations N_1 and N_2. Since N_1 and N_2 have identical encodings of corresponding primary input variables, then $Y_1 = Y_2$. Let C be the primary output variable of S associated with a block G. Denote by c the value taken by the output of G under the input assignment Y_1 (or Y_2). From Definition 3 it follows that the value assignment taken by the outputs of the implementation I_1 (respectively I_2) of the block G in the circuit N_1 (respectively N_2) is $q_1(c)$ (respectively $q_2(c)$). Since all the primary output variables have the same encoding, then $q_1(c) = q_2(c)$. □

Definition 7. *Let N_1, N_2 be two functionally equivalent Boolean circuits implementing a specification S. Let for each primary input and output variable X of S, encodings $q_1(X)$ and $q_2(X)$ (used when producing N_1 and N_2 respectively) be identical. Then S is called* ***a common specification (CS)*** *of N_1 and N_2.*

Definition 8. *Let S be a CS of N_1 and N_2. Let the maximal number of gates in an implementation of a block from S in N_1 and N_2 be equal to p. Then we will say that S is a CS of N_1 and N_2 of* ***granularity*** *p.*

2.2 Equivalence checking as satisfiability problem

In this subsection, we introduce the class $M(p)$ that is considered in this paper

Definition 9. *A disjunction of literals of Boolean variables not containing more than one literal of the same variable is called a* ***clause****. A conjunction of clauses is called a* ***conjunctive normal form*** *(CNF).*

Definition 10. *Given a CNF F,* ***the satisfiability problem*** *(SAT) is to find a value assignment to the variables of F for which F evaluates to 1 (also called a* ***satisfying assignment****) or to prove that such an assignment does not exist. A clause C of F is said to be* ***satisfied*** *by a value assignment if a literal of C is set to 1 by this assignment.*

The standard conversion of an equivalence checking problem into an instance of SAT is performed in two steps. Let N_1 and N_2 be two Boolean circuits to be checked for equivalence. In the first step, a circuit M called ***a miter*** [5] is formed from N_1 and N_2. The miter M is obtained by 1) identifying the corresponding primary inputs of N_1 and N_2; 2) adding d two-input XOR gates (where d is the number of primary outputs in N_1 and N_2), each XOR gate being fed by the j-th primary outputs of N_1 and N_2, $1 \leq j \leq d$; 3) adding a d-input OR gate, inputs of which are connected to the outputs of the d XOR gates. So the miter of N_1 and N_2 evaluates to 1 if and only if for some input assignment a primary output of N_1 and the corresponding output of N_2 take different values. Therefore, the problem of checking the equivalence of N_1 and N_2 is equivalent to testing the satisfiability of the miter of N_1 and N_2.

In the second step of conversion, the satisfiability of the miter is ***reduced*** to that of a CNF formula F. This formula is a conjunction of CNF formulas $F_1, \ldots, F_n$ specifying the functionality of the gates of M and a one-literal clause that is satisfied only if the output of M is set to 1. A CNF F_i specifies a gate g_i of M if and only if each assignment satisfying F_i is consistent with the functionality of g_i and each assignment falsifying a clause of F_i is inconsistent with the functionality of g_i. For instance, the AND gate $y{=}x_1x_2$ is specified by the following three clauses $\overline{x_1} \vee \overline{x_2} \vee y$, $x_1 \vee \overline{y}$, $x_2 \vee \overline{y}$.

Definition 11. *Given a constant p, a CNF formula F is a member of the* ***class M(p)*** *if and only if it satisfies the following two conditions.*

1. *F is the CNF formula (obtained by the procedure described above) specifying the miter of a pair of funcionally equivalent circuits N_1,N_2.*
2. *N_1,N_2 have a CS of granularity p', where $p' \leq p$.*

3 Solving formulas of $M(p)$ in general resolution

In this section, we show that the unsatisfiability of formulas from $M(p)$ can be proven in general resolution in a linear number of resolution steps.

Definition 12. *Let K and K' be clauses having opposite literals of a variable x and there is only one such variable. The* ***resolvent of K , K' in x*** *is the clause that contains all the literals of K and K' but the positive and negative literals of x. The operation of producing the resolvent of K and K' is called* ***resolution.***

Definition 13. ***General resolution*** *is a proof system of propositional logic that has only one inference rule. This rule is to resolve two existing clauses to produce a new one. Given a CNF formula F, the proof of unsatisfiability of F in the general resolution system consists of a sequence of resolution steps resulting in the derivation of an empty clause (i.e. a clause without literals).*

Definition 14. *Let F be a set of clauses. Denote by* ***supp(F)*** *the support of F i.e. the set of variables whose literals occur in F.*

The following three propositions are used in the proof of Proposition 6.

Proposition 3. *Let A,B,C be Boolean functions and $A \wedge B \rightarrow C$. (The symbol $\rightarrow$ means implication.) Then for any function A' such that $A' \rightarrow A$, it is true that $A' \wedge B \rightarrow C$.*

Proof. The proof follows from the definition of implication.

Proposition 4. *Let X_1 and X_2 be sets of Boolean variables, $F(X_1, X_2)$ and $H(X_2)$ be CNF formulas, and F imply H. Then one can derive a CNF formula H' implying H, where $supp(H') \subseteq supp(H)$, from clauses of F in at most $3^{|supp(F)|}$ resolution steps.*

Proof. Let $H = C_1 \wedge \ldots \wedge C_k$. We show that for any $C_i, 1 \leq i \leq k$ one can derive a clause C_i' (by resolving clauses of F) such that C_i' implies C_i. Then the CNF formula $H' = C_1' \wedge \ldots \wedge C_k'$ implies H.

Since C_i is a clause of H and F implies H, then F implies C_i. Then from the completeness of resolution it follows that by resolving clauses of F one can derive a clause C_i' implying C_i. Indeed, by making the assignments setting the literals of C_i to 0, we turn F into an unsatisfiable CNF formula F'. Then an empty clause can be derived from clauses of F'. From a derivation of an empty clause from F', one can easily construct the derivation of a clause C_i' (from clauses of F) implying C_i. The total number of resolvents that can be obtained from clauses of F is bounded by $3^{|supp(F)|}$. So all the clauses of H' can be obtained in no more than in $3^{|supp(F)|}$ resolution steps.

Definition 15. *Let G be a block of S, I_1 and I_2 be the implementations of G in N_1 and N_2 and C be the multi-valued variable associated with the output of G. A set of clauses $R(v_1(C), v_2(C))$ is called* ***a restricting set*** *for the block G if the following two conditions hold:*

1. *The support of $R(v_1(C), v_2(C))$ is a subset of $v_1(C) \cup v_2(C)$ (here, $v_1(C)$, $v_2(C)$ are the coding variables of encodings q_1 and q_2 for the values of C).*
2. *Let z_1, z_2 be assignments to the variables of $v_1(C)$ and $v_2(C)$ respectively. An assignment (z_1, z_2) satisfies $R(v_1(C), v_2(C))$ if and only if $z_1 = q_1(c)$ and $z_2 = q_2(c)$ where $c \in D(C)$).*

Proposition 5. *Let S be a CS of circuits N_1, N_2. Let $C = G(A, B)$ be a block of S. Let F be the CNF formula specifying the miter of N_1, N_2. Let $F(I_1(G))$ and $F(I_2(G))$ be the parts of F specifying the implementation $I_1(G)$ and $I_2(G)$ respectively. Then the CNF $P =$ Restriction $\wedge$ Implementation implies $R(v_1(C), v_2(C))$. Here, Restriction $= R(v_1(A), v_2(A)) \wedge R(v_1(B), v_2(B))$ and Implementation $= F(I_1(G)) \wedge F(I_2(G))$.*

Proof. To prove that P implies $R(v_1(C), v_2(C))$ one needs to show that any assignment satisfying P satisfies $R(v_1(C), v_2(C))$ as well. It is not hard to see that the support of all CNFs of the expression $P \rightarrow R(v_1(C), v_2(C))$ is a subset of $supp(F(I_1(G))) \cup supp(F(I_2(G)))$. Let $h = (x_1, x_2, y_1, y_2, z_1, z_2)$ be an assignment satisfying P where $x_1, x_2, y_1, y_2, z_1, z_2$ are assignments to $v_1(A)$, $v_2(A), v_1(B), v_2(B), v_1(C), v_2(C)$ respectively. Since h satisfies P it has to satisfy *Restriction* and *Implementation*. Since h satisfies *Restriction*, then $x_1 = q_1(a)$, $x_2 = q_2(a)$ and $y_1 = q_1(b)$, $y_2 = q_2(b)$ where $a \in D(A)$ and $b \in D(B)$. So h can be represented as $(q_1(a), q_2(a), q_1(b), q_2(b), z_1, z_2)$. Since h satisfies *Implementation* then z_1 has to be equal to $q_1(c)$, $c = G(a, b)$ and z_2 has to be equal to $q_2(c)$. Hence h satisfies $R(v_1(C), v_2(C))$. □

Proposition 6. *Let F be a formula of $M(p)$ specifying the miter of circuits N_1, N_2 obtained from a CS S of granularity p' where $p' \leq p$. The unsatisfiability of F can be proven by a resolution proof of no more than $d * n * 3^{6p}$ resolution steps where n is the number of blocks in S and d is a constant. (These proofs will be further referred to as **specification guided proofs**).*

Proof. From Proposition 5 it follows that one can deduce a restricting set of clauses for each variable of S in topological order starting with blocks of topological level 1 and proceeding to outputs. Indeed, if a variable A is a primary input variable of S, then $R(v_1(A), v_2(A))$ is empty. (The latter is true because, since S is a CS of N_1 and N_2, then $q_1(A)$ and $q_2(A)$ are identical and, besides, variables $v_1(A)$ and $v_2(A)$ in a miter are identified.) Let $C = G(A, B)$ be a block of topological level 1. Then A and B are primary input variables and restricting sets of clauses for them are known (namely, they are empty). From Proposition 5 it follows that $R(v_1(C), v_2(C))$ is implied by $F(I_1(G)) \cup F(I_2(G))$. From Proposition 4 it follows that a CNF R' formula implying $R(v_1(C), v_2(C))$ such that $supp(R') \subseteq supp(R)$ can be derived by resolving clauses of $F(I_1(G)) \cup F(I_2(G))$. From Proposition 3 it follows that replacing CNF formula $R(v_1(C), v_2(C))$ with a CNF R' that implies R does not break the inductive scheme of deducing restricting sets in topological order. After restricting CNFs are computed for the variables of topological level 1, the same procedure can be applied to variables of topological level 2 and so on.

The complexity of the derivation of a restricting set $R(v_1(A), v_2(A))$ is 3^{6p}. Indeed, the support of all CNFs forming the expression of Proposition 5 is a subset of $supp(F(I_1(G))) \cup supp(F(I_2(G)))$ where G is the block whose output is associated with the variable A. The number of gates in each implementation is bounded by p. Besides, each gate has two inputs and one output. So, the total number of variables in the clauses used when deriving $R(v_1(A), v_2(A))$ is at most $6 * p$. Hence the complexity of derivation is bounded by 3^{6p}. Since the total number of blocks in S in n, then the complexity of deriving restricting sets for all the variables of S is bounded by $n * 3^{6p}$.

Now we show that after restricting sets are deduced for all the primary output variables of S, an empty clause can be derived in the number of resolution steps that is linear in $n * p$. Let G be a block of S whose output (associated with variable C) is a primary output of S. From the definition of the class $M(p)$ it follows that the encodings $q_1(C)$ and $q_2(C)$ are identical and have minimal length. According to Remark 1 any assignment to the variables of $v_1(C)$ (or $v_2(C)$) is a code of a value $c \in D(C)$. Then any assignment h satisfying $R(v_1(C), v_2(C))$ has to assign the same values to corresponding variables of $v_1(C)$ and $v_2(C)$. This means that $R(v_1(C), v_2(C))$ implies 2-literal clauses specifying the equivalence of corresponding variables of $v_1(C)$ and $v_2(C)$. Then these 2-literal equivalence clauses can be deduced from $R(v_1(C), v_2(C))$ in the number of resolution steps bounded by 3^{6p}. (These steps are already counted in the expression $n * 3^{6p}$). Having deduced these equivalence clauses one can derive $|v_1(C))|$ single literal clauses (each variable representing the output of a gate XORing a pair of corresponding outputs of I_1 and I_2) that can be satisfied only by setting outputs of XOR gates to 0. The number of such clauses is bounded by $n * p$ and the complexity of the derivation of such a clause is bounded by a constant. Finally, in the number of resolution steps bounded by $n * p$ we can derive the empty clause. It is deduced from the single-literal clause that requires setting the output of the miter to 1, the clauses specifying the last OR gate, and the single-literal clauses that require setting the outputs of the XOR gates to 0. □

Remark 3. If the value of p is fixed, then the expression 3^{6p} is just a constant and so the formulas of $M(p)$ can be proven to be unsatisfiable in a linear number of resolution steps.

Proposition 7. *Given a number r, there is always p such that in a specification driven resolution proof of unsatisfiability of a formula $F \in M(p)$ one has to produce a clause of at least r literals.*

Proof. Let $C = G(A, B)$ be a block of a specification S and N_1 and N_2 be circuits with a CS S. Let $q_1(C)$, $q_2(C)$ be the binary encodings of the variable C used when obtaining N_1 and N_2. Let F be the formula specifying the miter of N_1 and N_2. The key feature of a specification guided resolution proof is that it deduces restricting sets of clauses. To prove the required proposition it suffices to show that for any r there are always binary encodings $q_1(C)$, $q_2(C)$ such that any restricting set of clauses $R(v_1(C), v_2(C))$ has to contain a clause of at least r literals. Then in any specification guided resolution proof of the unsatisfiability

of F one has to produce a clause of at least r literals. Since S is finite, there exists p such that F is in $M(p)$.

Let C=$G(A,B)$ be a block of a specification S and $level(C) = 1$. Assume that the multiplicity of variables A,B and C is 2^r. Let $I_1(G)$ and $I_2(G)$ be two implementations of G where $q_1(A) = q_2(A)$ and $q_1(B) = q_2(B)$ and the variable C is encoded with two different r-bit encodings. Now we show that there is a pair of encodings $q_1(C)$ and $q_2(C)$ such that any restricting set of clauses $R(v_1(C), v_2(C))$ has to include a clause of at least r literals

Denote by Z the vector whose all r components are equal to 0. Denote by B_i the vector of r components i-th component of which is equal to 1 and the rest of the components are equal to 0. Let $c_0,\ldots,\ c_m$ be the values of C where $m = 2^r - 1$. Let $q_1(C)$ and $q_2(C)$ be picked in such a way that the following two conditions hold:

1) The codes for $c_1,\ldots,c_r$ are chosen as follows: $q_1(c_1)$=$q_2(c_1)$=B_1, $\ldots$, $q_1(c_i)$=$q_2(c_i)$=B_i, $\ldots$, $q_1(c_r)$=$q_2(c_r)$=B_r.

2) Encodings q_1 and q_2 assign the code Z to different values of C. In other words, there are $c_j, c_p \in D(C)$, where $j, p > r$ such that $q_1(c_j) = Z$, $q_2(c_j) \neq Z$ and $q_2(c_p) = Z$, $q_1(c_p) \neq Z$ and $j \neq p$.

Let us show that among the clauses of $R(v_1(C), v_2(C))$ there has to be a clause of at least r literals. When the output of $I_1(G)$ is equal to Z the output of $I_2(G)$ cannot be equal to Z. So in any restricting set $R(v_1(C), v_2(C))$ there must be a clause K falsified by the vector Z,Z (obtained by the concatenation of two vectors Z.) Let us show that any clause of $R(v_1(C), v_2(C))$ that is falsified by Z,Z has to contain at least r literals. The idea is that if a clause K contains less than r literals, then it falsifies one of the allowed combinations B_i,B_i where $i = 1, \ldots, k$. Indeed, to be falsified by Z,Z the clause should contain only positive literals of variables $q_1(C)$, $q_2(C)$. Let x be a positive literal contained in K. Any two allowed combinations B_i,B_i and B_j,B_j where $i \neq j$ have different components that are equal to 1. Hence only one of the allowed combinations B_i,B_i can set the literal x to 1. So if the clause C contains less than r literals than there exists an allowed combination B_i,B_i that falsifies K. □

4 Class $M(p)$ and non-automatizability of general resolution

It is not hard to show that if p is fixed, the unsatisfiability of a formula F from $M(p)$ can be proven by a polynomial time deterministic algorithm. Indeed, from the proof of Proposition 6 it follows that the maximal length of a resolvent one needs to derive when producing an empty clause is bounded by $6*p$. So an empty clause can be obtained by the following trivial algorithm. This algorithm derives all the resolvents whose length does not exceed a threshold value. (Initially, this value is equal to 1.) If an empty clause is not derived, the algorithm increases the threshold value by 1 and derives all the resolvents whose length does not exceed the new threshold value. As soon as the threshold value reaches $6 * p$, the algorithm deduces an empty clause, and since the value of p is fixed, the

algorithm completes in polynomial time (in formula length). From Proposition 7 it follows that the order of the polynomial bounding the performance of the described algorithm is a monotonic increasing function of p. (Indeed, according to Definition 11 if $p' < p''$, then $M(p') \subset M(p'')$. Denote by $L(p)$ the length of the longest clause one has to deduce in a specification guided proof of a formula from $M(p)$. Then $L(p') \leq L(p'')$. From Proposition 7 it follows that the value of $L(p)$ actually increases as p grows.) So even though the class $M(p)$ is polynomially tractable for every fixed p, the complexity of the described algorithm increases as p grows, which makes it impractical.

Due to polynomial tractability, formulas from $M(p)$ cannot be used for proving general resolution to be non-automatizable. (Recall that a proof system is non-automatizable if there is no algorithm for finding proofs whose length is a polynomial of the length of the shortest proof.) However, one can form harder classes of formulas by letting the parameter p be a function of formula length L. Denote by $M(\infty)$ the union of classes $M(p)$ for all p. Let $M(p \leq t * log_2(L))$ be a class of formulas from $M(\infty)$ for which the value of p is bounded by the value of $t * log_2(L)$ where t is a constant. (So the value of p "slowly" grows as the length of the formula increases.) Since for a formula from $M(p)$ there is a resolution proof whose length is bounded by $d*n*3^{6p}$, then the size of resolution proofs for the formulas from $M(p \leq t*log_2(L))$ is bounded by L^{12t}. That is these formulas have polynomial size resolution proofs of unsatisfiability. On the other hand, since the value of p is not fixed, the length of resolvents one has to derive for the formulas of $M(p \leq t * log_2(L))$ is not bounded any more. So there is no trivial polynomial time deterministic algorithm like the one described above for solving formulas from $M(p \leq t * log_2(L))$.

A natural question is whether there is a better deterministic algorithm for solving formulas from a class $M(p)$ or harder classes that can be derived from $M(\infty)$. We believe that the answer is negative unless at least one of the following two assumptions does not hold. The first assumption is that, given circuits N_1, N_2 having a CS S, finding S is hard. (In particular, the problem of testing if a pair of circuits has a CS of granularity less or equal to p is most probably NP-complete.) The second assumption is that specification guided proofs are the only short resolution proofs for a class of formulas from $M(p)$. Then, for a formula of this class, the only way to find a short resolution proof is to construct a specification guided proof. But knowing such a proof one could recover the underlying specification. On the other hand, according to our first assumption finding a CS should be hard. So, for this class of formulas there should not be an efficient algorithm for finding short resolution proofs.

5 Experimental results

The goal of experiments was to prove that formulas of $M(p)$ are hard for existing SAT-solvers and that the hardness of formulas increases as p grows. Namely, we would like to test for satisfiability formulas from classes $M(p')$ and $M(p'')$ where $p' < p''$ and show that formulas from $M(p'')$ are much harder. The main

problem here is to guarantee that the formulas of $M(p'')$ we build are not in $M(p')$. (That is one needs to make sure that circuits with a CS of granularity $p \leq p''$ whose equivalence checking a formula from $M(p'')$ describes, do not have a "finer" CS whose granularity is less or equal to p'.) We solve this problem by constructing formulas of classes $M(1)$ and $M(p), p > 1$ and using the fact that proving that a formula of $M(p), p > 1$ is not in $M(1)$ is easy. (Of course, generating formulas with a greater variety of values of p will make experimental results more convincing. We hope that such experiments will be carried out in the near future.)

To construct formulas of $M(1)$ we just form the miter of two identical copies N_1,N_2 of a Boolean circuit N. These two copies can be considered as obtained from the "specification" N where each two-valued variable X was encoded using the trivial encoding $q(0) = 0$, $q(1) = 1$. Since each block G of specification is replaced with a "sub-circuit" consisting of one gate (identical to G), then the formula F specifying the miter is in $M(1)$.

To construct formulas from a class $M(p')$, $p' > 1$ that are not in $M(1)$ we used the following observation. The fact that a formula F is in $M(1)$ means that F specifies equivalence checking of circuits N_1,N_2 that are identical modulo negation of some gate pins. That is if g is a gate of N_1 then its counterpart in N_2 is identical to g modulo negation of an input (or both inputs) and/or the output. This means that each internal point y_1 of N_1 has a corresponding point y_2 of N_2 that is functionally equivalent to y_1 (modulo negation) in terms of primary input variables. So one can obtain a formula F that is in $M(p')$, $p' > 1$ and not in $M(1)$ by producing two circuits from a specification S that have no functionally equivalent internal points (modulo negation). This can be achieved by taking a specification S and picking two sets of "substantially different" encodings of the internal multi-valued variables of S

In the experiments we tested the performance of 2clseq (downloaded from [15], Zchaff (downloaded from [13]), BerkMin (version 561 that can be downloaded from [3]). BerkMin and Zchaff are the best representatives of the conflict driven learning approach. The program 2clseq was used in experiments because it employs preprocessing to learn short clauses and uses a special form of resolution called hyperresolution. The experiments were run on a SUNW Ultra-80 system with clock frequency 450MHz. In all the experiments the time limit was set to 60,000 sec. (16.6 hours). The best result is shown in bold.

In Table 1 we test the performance of the three SAT-solvers on formulas from $M(1)$. Each formula describes equivalence checking of two copies of an industrial circuit from the LGSynth-91 benchmark suite [16]. First, a circuit N of LGSynth-91 was transformed by a logic optimization system SIS [10] into a circuit N' consisting only of two-input AND gates and invertors. Then the formula specifying the miter of two copies of N' was tested for satisfiability. BerkMin has the best performance on almost all the instances except four. On the other hand, 2clseq is the only program to have finished all the instances within the time limit. In particular, it is able to solve the C6288 instance (a 16-bit multiplier) in about one second while BerkMin and Zchaff fail to solve it

Table 1. Equivalence checking of circuits from $M(1)$

Name of specification	Number of var.	Number of clauses	2clseq (sec.)	Zchaff (sec.)	BerkMin (sec.)
9symml	480	1,410	0.2	0.1	**0.04**
C880	806	2,237	**0.1**	5.4	0.5
ttt2	1,385	4,069	1.3	0.3	**0.1**
frg1	1,615	4,759	1.3	0.6	**0.3**
term1	1,752	5,157	2.0	2.0	**0.6**
x4	2,083	5,929	1.5	1.1	**0.3**
alu4	2,368	7,057	29.9	2.0	**0.5**
i9	2,477	7,105	14.3	1.0	**0.3**
c3540	2,624	7,723	**3.0**	71.0	13.4
rot	2,990	8,497	2.1	6.1	**0.9**
x1	4,380	12,955	27.8	3.8	**0.9**
dalu	4,713	13,899	36.8	5.7	**1.9**
c6288	4,770	14,245	**1.2**	*	*
frg2	5,158	14,967	17.5	6.5	**1.3**
c7552	5,781	16,621	**4.6**	327.5	83.6
k2	5,848	17,369	406.8	2.1	**1.1**
i10	6,499	18,653	31.2	237.4	**24.4**
i8	7,262	21,307	207.7	8.1	**2.7**
t481	9,521	28,512	3,250.4	13.1	**8.7**
des	14,451	42,343	164.2	271.7	**15.1**
too_large	29,027	86,965	8,838.9	384.47	**162.6**

'*' means that the program was aborted after 60,000 seconds

in 60,000 seconds. This should be attributed to successful formula preprocessing that results in deducing many short clauses describing equivalences of variables.

In Table 2 we test the performance of 2clseq, Zchaff, BerkMin on formulas that are in a class $M(p'), p' > 1$ and not in $M(1)$. These circuits were obtained by the following technique. First, we formed multi-valued specifications. Each specification S was obtained from a circuit N of the LGSynth-91 benchmark suite by replacing each binary gate of N with a four-valued gate. (In other words, the obtained specification S had different functionality while inheriting the "topology" of N.) Then two functionally equivalent Boolean circuits N_1, N_2 were produced from S using two "substantially different" sets of two-bit encodings of four-valued values. This guaranteed that the internal points of N_1 had no (or had very few) functionally equivalent (modulo negation) counterparts in N_2 and vice versa. So each produced formula was not in $M(1)$.

It is not hard to see that the performance of all three SAT-solvers is much worse regardless of the fact that each formula of Table 2 is only a few times larger than its counterpart from Table 1. Namely, the number of variables in each formula of Table 2 is only two times and the number of clauses is only four

times that of its counterpart from Table 1. Such kind of performance degradation is not unusual for formulas having exponential complexity in general resolution. However, all the formulas we used in experiments are in $M(p')$ (class $M(1)$ is a subset of any class $M(p')$, $p' > 1$) that has linear complexity in general resolution.

2clseq is able to solve only three instances, which indicates that preprocessing that helped for the instances of $M(1)$ does not work for $M(p'), p' > 1$. The probable cause is that "useful" clauses are longer than for formulas of $M(1)$. BerkMin shows best performance on the instances of Table 2. Nevertheless it fails to solve five instances and is much slower on instances that it is able to complete. For example, BerkMin solves the instance dalu from $M(1)$ in 1.6 seconds while testing the satisfiability of its counterpart from $M(p'), p' > 1$ takes more than 20,000 seconds. These results suggest that indeed formulas $M(p)$ are hard for existing SAT-solvers and the "hardness" of formulas increases as p grows.

Table 2. Equivalence checking of circuits from $M(p'), p' > 1$

Name of specification	Number of var.	Number of clauses	2clseq (sec.)	Zchaff (sec.)	BerkMin (sec.)
9symml	960	6,105	206.3	210.6	**58.2**
C880	1,612	9,373	*	*	**200.1**
ttt2	2,770	17,337	*	799.4	**77.2**
frg1	3,230	20,575	*	*	**3,602.4**
term1	3,504	22,229	*	*	**1,183.6**
x4	4,166	24,733	*	769.5	**139.0**
alu4	4,736	30,465	45,653.5	25,503.6	**2,372.6**
i9	4,954	29,861	2,215.5	23.5	**11.5**
c3540	5,248	33,199	*	*	**4,172.0**
rot	5,980	35,229	*	*	**1,346.9**
x1	8,760	55,571	*	*	*
dalu	9,426	59,991	*	*	**20,234.4**
c6288	9,540	61,421	*	*	*
frg2	10,316	62,943	*	7,711.0	**1,552.9**
c7552	11,282	69,529	*	803.5	**74.5**
k2	11,680	74,581	*	*	*
i10	12,998	77,941	*	*	**38,244.6**
i8	14,524	91,139	*	35,721.5	**4,039.7**
t481	19,042	123,547	*	*	*
des	28,902	179,895	*	1,390.3	**331.1**
too_large	58,054	376,801	out of memory	*	*

'*' means that the program was aborted after 60,000 seconds

6 Conclusions

We introduced a class $M(p)$ of formulas specifying equivalence checking of Boolean circuits obtained from the same specification. If the value of p is fixed, the formulas of $M(p)$ can be proven to be unsatisfiable in a linear number of resolution steps. On the other hand, the formulas of $M(p)$ can be solved in a polynomial time by a trivial deterministic algorithm but the order of the polynomial increases as the value of p grows. We give reasons why formulas from $M(p)$ should be hard for any deterministic algorithm. We show that formulas of $M(p)$ are indeed hard for the state-of-the-art SAT-solvers we used in experiments.

References

1. F.Bacchus. *Exploring the computational tradeoff of more reasoning and less searching.* Fifth International Symposium on Theory and Applications of Satisfiability Testing, pages 7-16, 2002.
2. E.Ben-Sasson, R. Impagliazzo, A.Wigderson. *Near optimal separation of Treelike and General resolution* SAT-2000: Third Workshop on the Satisfiability Problem. - May 2000.
3. BerkMin web page. http://eigold.tripod.com/BerkMin.html
4. M.Bonet, T.Pitassi, R.Raz. On interpolation and automatization for Frege Systems. SIAM Journal on Computing, 29(6),pp 1939-1967,2000.
5. D. Brand. *Verification of large synthesized designs.* Proceedings of ICCAD-1993,pp 534-537.
6. E. Goldberg, Y. Novikov. *BerkMin: A fast and robust SAT-solver.* Design, Automation, and Test in Europe (DATE '02), pages 142-149, March 2002.
7. A. Haken. The intractability of resolution. Theor. Comput. Sci. 39 (1985),297-308.
8. M. Moskewicz, C. Madigan, Y. Zhao, L.Zhang, and S.Malik.*Chaff: Engineering an efficient SAT-solver.* Proceedings of DAC-2001.
9. M. Alekhnovich, A.Razborov. Resolution is not automatizable unless W[p] is tractable. Proceedings of FOCS,2001.
10. Sentovich, E. e.a. *Sequential circuit design using synthesis and optimization.* Proceedings of ICCAD, pp 328-333, October 1992.
11. J.P.M.Silva, K.A.Sakallah. GRASP: A Search *Algorithm for Propositional Satisfiability* . IEEE Transactions of Computers. -1999. -V. 48. -P. 506-521.
12. G.S.Tseitin. On the complexity of derivations in propositional calculus. Studies in Mathematics and Mathematical Logic. Part II (Consultants Bureau, New York/London, 1970) 115-125.
13. Zchaff web page. http://ee.princeton.edu/~chaff/zchaff.php
14. H.Zhang. SATO: *An efficient propositional prover.* Proceedings of the International Conference on Automated Deduction. -July 1997. -P.272-275.
15. 2clseq web page. http://www.cs.toronto.edu/~fbacchus/ 2clseq.html.
16. http://www.cbl.ncsu.edu/CBL_Docs/lgs91.html