On Complexity of Equivalence Checking

Cadence Berkeley Labs
1995 University Ave.,Suite 460, Berkeley, California,94704
phone: (510)-647-2825, fax: (510)-486-0205

CDNL-TR-2003-0826
August 2003

Eugene Goldberg (Cadence Berkeley Labs), egold@cadence.com
Yakov Novikov (National Academy of Science, Belarus), yakov_nov@tut.by

Abstract

We introduce the notion of a common specification (CS) that is the key to understanding the complexity of equivalence checking. A CS S of functionally equivalent Boolean circuits N_1 and N_2 is a circuit of multi-valued blocks where N_1 and N_2 can be obtained from this CS by encoding the values of multi-valued variables of S. We show that the performance of an equivalence checking algorithm heavily depends on whether a non-trivial CS S is known. If it is, then there exists an algorithm we describe whose run time is linear in the number of blocks in S. However, there are good reasons to believe that for any algorithm that does not have information about such a CS, equivalence checking is hard (if not infeasible). We experimentally show that even equivalence checking of circuits with a very fine CS is hard for a representative collection of methods while CS driven equivalence checking takes only a few seconds.

1. Introduction

In electronic CAD one has to face computationally hard problems very often. One of the examples of such problems is equivalence checking of combinational circuits that is a coNP-complete problem. In complexity theory an algorithm solving a coNP-complete problem has to return an easily checkable proof that is a “solution” of this problem. In terms of equivalence checking, an algorithm testing the equivalence of Boolean circuits N_1 and N_2 has to return a proof that N_1 and N_2 are equivalent. Currently, researchers in complexity theory study two types of complexity. In terms of equivalence checking, a class T of equivalence checking problems has complexity of the first type in a proof system D , if in D there is no polynomial size proof for the problems of T. A class T of equivalence checking problems has complexity of the second type in a proof system D, if for the problems of T there is no polynomial time algorithm to find polynomial size proofs even such proofs exist in D. (In that case the proof system D is called non-automatizable [2].) Complexity of the second type is “weaker” than that of the first type because it suggests that the problems of T can be efficiently solved if the proof system D is supplied with some extra information about the “structure” of short proofs.

In this paper we show that there are good reasons to believe that equivalence checking problems occurring in practice have complexity of the second type in proof systems like general resolution or BDDs [4]. These are proof systems that are commonly used in existing equivalence checkers. (The general resolution system is the basis of existing SAT-solvers.). This means that these problems have short proofs of equivalence that cannot be found by existing (and probably any possible) equivalence checkers unless some extra information about the structure about short proofs is given.

Let N_1 and N_2 be two Boolean circuits to be checked for equivalence. The extra information mentioned above is a common specification (CS) of circuits N_1 and N_2. A CS S is just a circuit of multi-valued gates further referred to as blocks such that N_1 (or N_2) can be obtained from S by replacing each block G of S with its implementation $I_1(G)$ (or $I_2(G)$). The circuit $I_1(G)$ (or $I_2(G)$) implements a multi-output Boolean function obtained from the truth table of G after encoding the values of multi-valued variables with binary codes.

Any pair of functionally equivalent circuits N_1,N_2 has a CS S. If N_1,N_2 are identical copies of a circuit N^*, then N^* can be considered as their CS. Each "block" of the specification N^* is "implemented" with only one gate of N_1 or N_2. So N^* is the "finest" possible CS. If N_1 and N_2 are completely structurally dissimilar, they have a trivial CS consisting of only one block where N_1 and N_2 are implementations of this block. If circuits N_1 and N_2 are structurally similar they have a set of non-trivial CSs (i.e. ones different from a trivial single block CS mentioned above).

Let S be a CS of circuits N_1 and N_2 consisting of n blocks. We describe a procedure of equivalence checking of N_1 and N_2 whose run time is $n*Compl(p)$. Here p is the maximal number of gates used in the implementation of a block of S. The value of p characterizes the "granularity" of S. (Henceforth, when we say that a CS if fine we mean that the value of p is small.) The function $Compl(p)$ describes the complexity of computing filtering and correlation functions for the block of S that has the largest implementation (of p gates). The procedure computes these functions for each block of S proceeding in topological order from inputs to outputs. In this paper we use general resolution as the "instrumental" proof system. In particular, for general resolution we give an upper bound (a very conservative one) on the value of function $Compl(p)$ which is 3^{6p} .

The result above leads to the following two conclusions. First, if there is a non-trivial CS S of circuits N_1 and N_2 consisting of n blocks, then equivalence checking of N_1 and N_2 splits into n subproblems. (Henceforth, when we say that N_1 and N_2 have a CS S we mean that S is non-trivial.) If the truth table of each block G of S is known along with the codes of values of multi-valued variables used when obtaining N_1 and N_2 from S, such problem decomposition is obvious. One just needs to check that each block G of S is correctly implemented by subcircuits $I_1(G)$ and $I_2(G)$ of N_1 and N_2 respectively. The key point however is that the procedure we introduce needs neither any knowledge of the functionality of blocks nor the codes of values of multi-valued variables. It only needs information about the topology of S (i.e. how blocks are connected to each other) and the correspondence between the gates of N_1 and N_2 and the blocks of S.

The second conclusion is that finer CSs mean more efficient equivalence checking. In particular, if one needs to solve equivalence checking problems from a class where each pair of circuits N_1,N_2 has a CS with the granularity p bounded by a constant, equivalence checking is linear in the size of circuits N_1 and N_2 no matter how large they are.

A natural question is whether equivalence checking of circuits N_1 and N_2 having a fine CS S is easy if the latter is unknown. (Henceforth, if we say that a CS of N_1 and N_2 is known or unknown, we mean the finest possible CS of N_1 and N_2, see Definition 9, or a "good" approximation of it.) The theory that addresses these issues has only started [13] and so cannot answer our question yet. However, there is a very good reason to believe that no procedure can efficiently check equivalence of circuits having even a very fine CS, if the latter is not known. On the one hand, the problem of finding the finest CS S of circuits N_1 and N_2 (or a good approximation of S) is most likely NP-complete. On the other hand, given a short proof of equivalence of N_1 and N_2 one could recover a "good" CS from this proof. So the existence of an efficient procedure for finding a short proof of equivalence would mean that there is an efficient algorithm for solving an NP-complete problem.

Due to lack of theory, we substantiate our claim experimentally. Namely, we show that if a CS S of circuits N_1,N_2 is unknown then their verification is very difficult. However, if S is known, N_1 and N_2 can be checked for equivalence in a few seconds using the procedure we introduce in this paper.

2. Some Background and Possible Applications of Our Theory

The most successful equivalence checkers try to make full use of structural similarity of circuits N_1,N_2 to be compared. In particular, they try to establish some strong relationships (like equivalence or implication) between internal points of N_1,N_2 [3], [7],[8]. These relationships are deduced in topological order proceeding from inputs to outputs until the equivalence of corresponding primary outputs of N_1 and N_2 is deduced. Circuits are considered to be structurally similar and so easy for equivalence checking if they have many internal points that are related by these strong relationships. Our theory generalizes the notion of structural similarity that has been used so far. Namely, we show that if N_1 and N_2 have a CS S of small granularity, then the equivalence checking of N_1 and N_2 can be very easy even though no internal points are related by strong relationships.

A new approach developed in [9],[11] is to implicitly compute the set $R(C)$ of all the satisfiable combinations of values for the variables of a cut C. (C is a cut in the miter of N_1,N_2 [3]). This cut gradually moves from inputs to outputs until it reaches the output of the miter. Every move of the cut is accompanied by recompilation of $R(C)$. The main flaw of this approach is that one has to compute the set $R(C)$ exactly which may be infeasible even if this set is represented implicitly. The techniques of [9],[11] allows one to simplify the BDDs representing the functionality of cut points. This is done by grouping a set of cut points and introducing new variables in BDDs representing the functionality of these points so that the set of satisfiable combinations for this set of cut points does not change. The problem here is that it is hard (if not impossible) to find a good variable grouping not knowing a CS of N_1 and N_2.

The procedure we introduce can also be considered as computation of a set R of satisfiable combinations for a cut C which gradually moves from inputs to outputs. The difference is that the knowledge of a CS S allows to approximate the set R(C) by a set of filtering and correlation functions. Besides, knowing S, one can "optimally" group variables by putting together ones that correspond to the implementation of a block of S.

Of course, the current state-of-the-art tools due to the creativity and ingenuity of their developers are able to check equivalence of the majority of circuits produced by the existing synthesis tools. This can be attributed to the fact that synthesis procedures tend to introduce only local changes because a non-local change is hard to verify. So the structure of the synthesized circuit is very similar to the original one. However, the theory we develop suggests that one can efficiently verify even non-local changes.

Suppose that N_1 is a circuit and N_2 is another circuit obtained from N_1 by a synthesis step. As long as N_1 and N_2 share a predefined CS S, they can be efficiently checked for equivalence.

(A natural candidate for such a predefined specification is the initial high-level description of the circuit.) An example of specification preserving synthesis steps are transformations that correspond to re-encoding values of a variable C associated with the output of a block G of S. This transformation is non-local because all the circuit in the fan-out cone of the implementation of G is affected up to the primary outputs. A synthesis tool making such transformations would generate various implementations of the same specification. Then equivalence checking of the initial circuit N_1 and the resulting circuit N_2 would be impossible without any knowledge of S.

Probably, the best way to describe the relation between N_1,N_2 and S is as follows. The initial circuit N_1 can be viewed as the original "message" and the synthesis procedure as an encryption process. The circuit N_2 is the result of encryption where S is a key. Then equivalence checking is just restoring the original message N_1 from N_2 i.e. a decryption process. If the key is known, then decryption can be performed very efficiently. Otherwise, equivalence checking is nothing else but code breaking. So no matter how powerful an equivalence checker is, without the key it can decrypt N_2 only if the latter is very similar to N_1.

The notion of a CS can also be helpful in reducing the complexity of equivalence checking of the circuits produced by existing synthesis tools. The idea is to keep a CS S of the initial circuit N_1 and the current circuit N_2 and recompute S incrementally after each change of N_2. Initially N_1,N_2 and S are identical (since N_1 and N_2 are identical they have a CS that is a copy of N_1 or N_2). Each synthesis step changing N_2 is accompanied by recomputation of S. The synthesis procedure produces a synthesized circuit N_2 and a CS S of N_1 and N_2. If the current specification S becomes too coarse during synthesis one can output S and the circuit synthesized so far and start computing a CS anew considering the current circuit as "initial". Then equivalence checking of the initial circuit and the synthesized one is performed in a number of steps. At each step one verifies the equivalence of some intermediate circuits N_i and N_{i+1} using their CS $S_{i,i+1}$ output in the middle of synthesis.

Of course, there is much to be done to realize ideas sketched above. In this paper we just make a first step providing an efficient procedure for equivalence checking of circuits with a known CS.

3. Common Specification of Boolean Circuits

In this section, we introduce the notion of a common specification of Boolean circuits. Let S be a combinational circuit of multi-valued blocks (further referred to as a ***specification***) specified by a directed acyclic graph H. The sources and sinks of H correspond to primary inputs and outputs of S. Each non-source node of H corresponds to a multi-valued block computing a multi-valued function of multi-valued arguments. Each node of n of H is associated with a ***multi-valued variable*** V. If n is a source of H , then the corresponding variable specifies values taken by the corresponding primary input of S. If n is a non-source node of S then the corresponding variable describes the values taken by the output of the block specified by n. If n is a source (respectively a sink), then the corresponding variable is called a ***primary input variable (***respectively ***primary output variable)***. We will use the notation C=G(A,B) to indicate that a) the output of a 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 and these nodes are associated with variables A and B.

Denote by $D(V)$ the ***domain*** of a variable V associated with a node of H. The value of $|D(V)|$ is called the ***multiplicity*** of V. If the multiplicity of every variable V of S is equal to 2 then S is a ***Boolean circuit***.

Now we describe how a Boolean circuit N can be produced from a specification S by encoding the multi-valued variables. Let $D(V)=\{v_1,\ldots,v_t\}$ be the domain of a variable V of S. Denote by $q(V)$ a Boolean encoding of the values of $D(V)$ that is a mapping $q{:}D(V)\rightarrow\{0,1\}^m$. Denote by $length(q(V))$ the number of bits in q that is the value of m. The value of $q(v_i)$, $v_i \in D(V)$ is called the ***code*** of v_i. Given an encoding q of length m of a variable V associated with a block of S, denote by $v(V)$ the set of m ***coding Boolean variables***.

In the following exposition we make the assumptions below.

Assumption 1. Each gate of a Boolean circuit and each block of a specification has two inputs and one output.

Assumption 2. The multiplicity of each primary input (or output) variable of a specification is a power of 2.

Assumption 3. If V is a primary input (or output) variable of a specification, then $length(q(V))=log_2(|D(V)|)$

Assumption 4. If v_1 and v_2 are values of a variable V of a specification and $v_1 \neq v_2$, then $q(v_1) \neq q(v_2)$.

Assumption 5. If A and B are two different variables of a specification , then $v(A) \cap v(B) = \emptyset$.

Remark 1. From Assumption 2 , Assumption 3 and Assumption 4 it follows that if A is a primary input (or output) variable, a mapping $q{:}D(A)\rightarrow\{0,1\}^m$ is bijective. In particular, any assignment to the variables of $v(A)$ is a code of some value $a \in D(A)$.

Definition 1. Given a Boolean circuit I, denote by $Inp(I)$ (respectively $Out(I)$) the set of variables associated with primary inputs (respectively primary outputs) of 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 the 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:

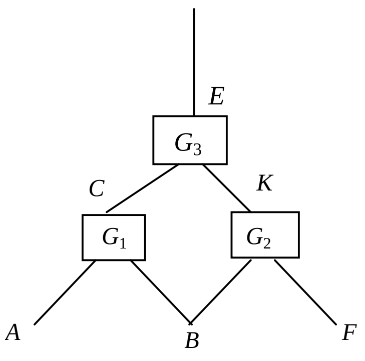
1) The set $Inp(I)$ is a subset of $v(A) \cup v(B)$.
2) The set $Out(I)$ is equal to v(C).
3) If the set of values assigned to $v(A)$ and $v(B)$ form codes $q(a)$ and $q(b)$ respectively where $a \in D(A)$, $b \in D(B)$, then $I(z')=q(c)$ where z' is the projection of the assignment $z=(q(a),q(b))$ on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

Remark 2. The reason why $Inp(I)$ may not include all the variables of $v(A)$ and/or $v(B)$ is that the function $G(A,B)$ may not distinguish some values of A or B. ($G(A,B)$ does not distinguish, say, values $a_1,a_2 \in D(A)$, if for any $b \in D(B)$, $G(a_1,b)=G(a_2,b)$.) So to implement $G(A,B)$ the circuit I may need only a subset of variables of $v(A) \cup v(B)$. This said, for the sake of simplicity, we will write $I(q(a),q(b))$ meaning $I(q'(a),q'(b))$, $q'(a)=proj(q(a),Inp(I))$ and $q'(b)=proj(q(b),Inp(I))$.

Definition 4. Let S be a multi-valued circuit. A Boolean circuit N is said to ***implement the 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 G_1 (specified by variable R) be connected to an input of block G_2 (specified by the same variable R) in S. 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 $q_i \in v(R)$ is connected to the input of I_2 specified by the same variable of $v(R)$ if $q_i \in Inp(I_2)$.



(a)

$C=G_1(A,B)$

A	B	C
a_0	b_0	c_0
a_0	b_1	c_1
a_0	b_2	c_1
a_0	b_3	c_0
a_1	b_0	c_1
a_1	b_1	c_2
a_1	b_2	c_2
a_1	b_3	c_0

(b)

$I_1(q_1(A),q_1(B))$

$q_1(A)$	$q_1(B)$		$q_1(C)$	
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	0

(c)

$I_2(q_2(A),q_2(B))$

$q_2(A)$	$q_2(B)$		$q_2(C)$		
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	1	0	0

(d)

Figure 1. A specification and the functionality of two implementations of a block

In Fig. 1*a* a specification of three blocks if shown. The functionality of two different implementations of the block $C=G_1(A,B)$ (Fig. 1*b*) are shown in Fig. 1*c* and 1*d*. Here $D(A)=\{a_0,a_1\}$, $D(B)=\{b_0,b_1,b_2,b_3\}$ and $D(C)=\{c_0,c_1,c_2\}$. Since A and B are primary input variables they are encoded with a minimum code length and $q_1(A)=q_2(A)$ and $q_1(B)=q_2(B)$ where $q_1(a_0)=0$, $q_1(a_1)=1$, $q_1(b_0)=00$, $q_1(b_1)=01$, $q_1(b_2)=10$, $q_1(b_3)=11$. Finally, the encodings $q_1(C)$ and $q_2(C)$ are $q_1(c_0)=00$, $q_1(c_1)=10$, $q_1(c_2)=01$ and $q_2(c_0)=100$, $q_2(c_1)=010$, $q_2(c_2)=001$.

Remark 3. Let N be an implementation of a specification S. Let p be the largest number of gates used in an implementation of a multi-valued block of S in N. We will say that S is a specification of ***granularity*** p for N.

Definition 5. The ***topological level*** of a block G in a specification S is the length of the longest path from a primary input of S to G. (The length of a path is measured in the number of blocks on it. The topological level of a primary input is assumed to be 0.) Denote by ***level(G)*** the topological level of G in S.

Let N be an implementation of a specification S. From Remark 1 it follows that for any value assignment h to the 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 $h=(q(x_1),\ldots,q(x_k))$. That is there is one-to-one correspondence between assignments to primary inputs of S and N. The same applies to primary outputs of S and N.

Definition 6. Let N be an implementation of S. Given a Boolean vector y of assignments to the primary inputs of N, the corresponding vector $Y=(x_1,..,x_k)$ such that $y=(q(x_1),\ldots,q(x_k))$ is called the ***pre-image*** of y.

Proposition 1. Let N be a circuit implementing specification S. Let $I(G)$ be the implementation of a block $C=G(A,B)$ of S in N. Let y be a value assignment to the primary input variables of N and Y be the pre-image of y. Then the values of primary outputs of $I(G)$ form the code $q(c)$ where c is the value taken by the output of G when the inputs of S take the values specified by Y.

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

Proof. The proofs of Proposition 1 and Proposition 2 are simple and so we omit them to save space.

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

Remark 4. Let S be a CS of N_1,N_2 and C be a variable of S. We will assume that $v_1(C)=v_2(C)$ if C is a primary input variable and $v_1(C) \cap v_2(C) = \varnothing$ otherwise.

Definition 8. Let S be a CS of N_1,N_2. Let p_1 (respectively p_2) be the granularity of S with respect to N_1 (respectively N_2). Then we will say that S is a CS of N_1,N_2 of ***granularity*** $p = max(p_1,p_2)$.

Definition 9. Given two functionally equivalent Boolean circuits N_1, N_2, S is called the ***finest common specification*** if it has the smallest granularity p among all the CSs of N_1 and N_2.

4. Equivalence Checking as SAT

Since in this paper we formulate the complexity of equivalence checking in terms of resolution proofs, we recall a common way of reducing equivalence checking to the satisfiability problem.

Definition 10. A disjunction of literals of Boolean variables not containing two literals of the same variable is called a ***clause***. A conjunction of clauses is called a ***conjunctive normal form*** (CNF).

Definition 11. 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 K of F is said to be ***satisfied*** by a value assignment y if $K(y)$=1.

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 Boolean circuits to be checked for equivalence. At the first step of this conversion, a circuit M called ***a miter*** [3] 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) XORing each pair of corresponding primary outputs of N_1 and N_2; 3) ORing the outputs of the added 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 evaluate to 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.

At 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,..,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. The CNF F_i specifies the i-th gate g_i of M. Any assignment to the variables of F_i that is inconsistent with the functionality of g_i falsifies a clause of F_i (and vice versa, a consistent assignment satisfies all the clauses of F_i.) For instance, the AND gate $y=x_1x_2$ is specified by the following three clauses $\sim x_1 \vee \sim x_2 \vee y$, $x_1 \vee \sim y$, $x_2 \vee \sim y$.

5. Equivalence Checking in General Resolution

In this section, we prove some results about the complexity of equivalence checking of circuits with a CS of granularity p. The main idea of the proof is that if S is a CS of N_1 and N_2, then their equivalence checking reduces to computing filtering and correlation functions for each variable of S. The two main properties of these functions are that

- They can be built based only on the information about the topology of S and about "assignment" of gates of N_1 and N_2 to blocks of S.
- Filtering and correlation functions for a variable C specifying the output of a block $G(A,B)$ can be computed "locally" from filtering and correlation functions of variables A and B and CNFs specifying implementations $I_1(G)$ and $I_2(G)$. So these functions can be computed in topological order starting with inputs and proceeding to outputs.

In this paper we consider computation of filtering and correlation functions as CNFs in the general resolution system. However, these functions can be computed in many other ways, for example, by using BDDs (see Remark 10).

Definition 12. 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.

- F is the CNF formula (obtained by the procedure described in Section 4) specifying the miter of a pair of functionally equivalent circuits N_1,N_2.
- N_1,N_2 has a CS of granularity p.

Definition 13. Let K and K' be clauses having opposite literals of a variable (say variable x) and there is only one such variable. The ***resolvent*** of K , K' in variable x is the clause that contains all the literals of K and K' but the positive (i.e. literal x) and negative (i.e. literal $\sim x$) literals of x. The operation of producing the resolvent of K and K' is called ***resolution***.

Definition 14. ***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, a proof $L(F)$ of unsatisfiability of F in the general resolution system consists of a sequence of resolutions resulting in the derivation of an ***empty clause*** (i.e. a clause without literals).

General resolution is complete, which means that given an unsatisfiable formula F there is always a proof $L(F)$ that derives an empty clause.

Definition 15. Let $F(X_1,X_2)$ be a Boolean function where X_1 and X_2 are sets of Boolean variables. The function $H(X_2)$ is called ***existentially implied*** by F if

- $F(X_1,X_2) \rightarrow H(X_2)$
- if $H(z)$=1 where z is an assignment to the variables of X_2, then there is an assignment y to the variables of X_1 such that $F(y,z)$=1.

Remark 5. Given a function $F(X_1,X_2)$, the function $H(X_2)$ existentially implied by F is unique. It can be obtained by from F by existentially quantifying away the variables of X_1. The definition of existential implication is important for understanding how filtering and correlation functions are obtained.

Proposition 3. Let $F(X_1,X_2)$ and $H(X_2)$ be CNF formulas where $H(X_2)$ consists of all the clauses depending only on variables from X_2 that can be derived from $F(X_1,X_2)$ by general resolution. Then $H(X_2)$ is existentially implied by $F(X_1,X_2)$.

Proof. The CNF $F(X_1,X_2)$ implies $H(X_2)$ because each clause of H is implied by F since it is derived by resolution. Assume that H is not existentially implied by F. Then there is an assignment z to the variables of X_2 such that $H(z)$=1 and for any assignment y to the variables of X_1, $F(y,z)$=0. However, this means that F implies a clause K depending only on variables of X_2 such that $K(z)$=0. Since K should be in H, then $H(z)$ should be equal to 0, which leads to a contradiction. □

Definition 16. Let F be a set of clauses. Denote by ***supp(F)*** the set of variables whose literals occur in clauses of F.

To estimate the complexity of obtaining the function existentially implied by F in general resolution, we need the following proposition.

Proposition 4. Let F be a set of clauses that implies a clause K. Then there is a sequence of at most $3^{|supp(F)|}$ resolution steps that results in the derivation of the clause K or a clause that implies K.

Proof. Denote by F' the formula that is obtained from F by making the assignments that set the literals of K to 0 (and removing the satisfied clauses and the literals set to 0). It is not hard to see that F' is unsatisfiable since it implies an empty clause. So there is a resolution proof $L(F')$ that results in deducing an empty clause. Then by replacing each clause of F' involved in $L(F')$ with its "parent" clause from F we get a sequence of resolutions resulting in deducing either the clause K or a clause that implies K. The number of resolvents in $L(F')$ cannot be more than $3^{|supp(F')|}$ (i.e. the total number of clauses of $|supp(F')|$ variables) and so it cannot be more than $3^{|supp(F)|}$. □

Remark 6. From Proposition 3 and Proposition 4 it follows that given a CNF $F(X_1,X_2)$ one can obtain the function $H(X_2)$ existentially implied by F in no more than $3^{|\mathrm{supp}(F')|}$ resolution steps.

Definition 17. Let N be an implementation of a specification S. Let C be a variable of S. A function Ff is called ***a filtering function*** if:

- $\mathrm{supp}(Ff) \subseteq v(C)$.
- If an assignment z to the variables of $v(C)$ is a code $q(c)$, $c \in D(C)$, then $Ff(z)=1$. Otherwise, $Ff(z)=0$.

Remark 7. If C is a primary input variable of S , then $Ff(v(C))\equiv 1$. Indeed, as it follows from Remark 1 any assignment to C is the code of a value $c \in D(C)$.

Proposition 5. Let N be an implementation of a specification S. Let $C=G(A,B)$ be a block of S. Let F be the CNF formula specifying N built as described in Section 4 and $F(I(G))$ be the part of F specifying the implementation $I(G)$ of G in N. Then P existentially implies $Ff(v(C))$ where $P=Ff(v(A)) \wedge Ff(v(B)) \wedge F(I(G))$.

Proof. To prove that $P \rightarrow Ff(v(C))$ one needs to show that any assignment that sets P to 1 also sets $Ff(v(C))$ to 1. It is not hard to see that the support of all the functions of the expression $P \rightarrow Ff(v(C))$ is a subset of $supp(F(I(G)))$. Let $h=(x,y,z)$ be an assignment that sets P to 1 where x,y,z are assignments to the variables from $v(A),v(B)$ and $v(C)$ respectively. Then h has to set to 1 the functions $Ff(v(A))$, $Ff(v(B)$, $F(I(G))$. Since h sets $Ff(v(A))$ to 1, then $x=q(a)$, $a \in D(A)$. Since h sets $Ff(v(B))$ to 1, then $y=q(b)$, $b \in D(B)$. So $h=(q(a),q(b),z)$. To set to 1 $F(I(G))$ assignment z has to be equal to $q(c)$, where $c=G(a,b)$. Then h sets $Ff(v(C))$ to 1.

Assume that $Ff(v(C))$ is not existentially implied by P. Then there exists an assignment $z=q(c)$, $c \in D(C)$ such that $Ff(z)=1$ and for any assignments x and y to the variables of $v(A)$ and $v(B)$ respectively, $P(x,y,z)=0$. However, $P(q(a), q(b), z) = 1$ where a and b are values of A and B such that $G(a,b)=c$, which leads to a contradiction. □

Definition 18. Let S be a CS of circuits N_1 and N_2 and C be a variable of S. A function Cf is called ***a correlation function*** for encodings q_1 and q_2 of the values of C (used when producing N_1 and N_2) if :

- $supp(Cf) \subseteq v_1(C) \cup v_2(C)$.
- $Cf(z_1, z_2)=1$ for any assignment z_1 to $v_1(C)$ and z_2 to $v_2(C)$ such that $z_1=q_1(c)$ and $z_2=q_2(c)$ where $c \in D(C)$. Otherwise $Cf(z_1, z_2)=0$.

Remark 8. If C is a primary input variable of S, then $Cf(v_1(C),v_2(C)) \equiv 1$. Indeed, as it follows from Remark 1 any assignment to $v_1(C)$ or $v_2(C)$ is the code of a value $c \in D(C)$. Besides, from the definition of CS it follows that $q_1(C)=q_2(C)$. Finally, from Remark 4 it follows that $v_1(C)=v_2(C)$. So any assignment (x,y) to the variables of $v_1(C),v_2(C)$ can be represented as $(q_1(c),q_2(c))$, $c \in D(C)$.

Proposition 6. 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 built as described in Section 4. Let $F(I_1(G))$ and $F(I_2(G))$ be the part of F specifying the implementation $I_1(G)$ and $I_2(G)$ of G in N_1 and N_2 respectively. Then P existentially implies $Cf(v_1(C),v_2(C))$. Here P = *Filtering* $\wedge$ *Correlation* $\wedge$ *Implementation* and *Filtering* = $Ff(v_1(A)) \wedge Ff(v_1(B)) \wedge Ff(v_2(A)) \wedge Ff(v_2(B))$, *Correlation* = $Cf(v_1(A),v_2(A)) \wedge Cf(v_1(B),v_2(B))$, *Implementation* = $F(I_1(G)) \wedge F(I_2(G))$.

Proof. To prove that P implies $Cf(v_1(C),v_2(C))$ one needs to show that any assignment that sets P to 1 also sets $Cf(v_1(C),v_2(C))$ to 1. It is not hard to see that the support of all the functions of the expression $P \rightarrow Cf(v_1(C),v_2(C))$ 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 that sets P to 1 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. Then h has to set to 1 all the functions the conjunction of which forms P. Since h has to set the function *Filtering* to 1, then $x_1=q_1(a_1)$, $x_2=q_2(a_2)$ where $a_1,a_2 \in D(A)$ and $y_1=q_1(b_1)$, $y_2=q_2(b_2)$, where $b_1,b_2 \in D(B)$. So $h=(q_1(a_1),q_2(a_2), q_1(b_1),q_2(b_2), z_1, z_2)$. Since h sets the function *Correlation* to 1 then a_1 has to be equal to a_2 and b_1 has to be equal to b_2. So h can be represented as $(q_1(a),q_2(a), q_1(b),q_2(b), z_1, z_2)$ where $a \in D(A)$ and $b \in D(B)$. Since h sets the function *Implementation* to 1, then z_1 has to be equal to $q_1(c)$, $c=G(a,b)$ and z_2 has to be equal to $q_2(c)$. So h is equal to $(q_1(a),q_2(a),q_1(b),q_2(b),q_1(c),q_2(c))$ and hence it sets the correlation function $Cf(v_1(C),v_2(C))$ to 1.

Assume that $Cf(v_1(C),v_2(C))$ is not existentially implied by P. Then there exists an assignment $z_1=q_1(c)$, $z_2=q_2(c)$ to the variables of $v_1(C)$ and $v_2(C)$ respectively such that $Cf(z_1, z_2)=1$ and for any assignment x_1, x_2, y_1, y_2 to the variables of $v_1(A)$, $v_2(A)$, $v_1(B)$, $v_2(B)$ respectively, $P(x_1, x_2, y_1, y_2, z_1, z_2)=0$. However, $P(q_1(a), q_2(a), q_1(b), q_2(b), z_1, z_2)=1$ where a, b are the values of A and B respectively for which $c=G(a,b)$. This leads to a contradiction. □

Proposition 7. 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. 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.

Proof. From Proposition 5 and Proposition 6 it follows that one can deduce correlation and filtering functions for all the variables of S starting with blocks of topological level 1 and proceeding in topological order. Indeed, let $C=G(A,B)$ be a block of topological level 1. Then A and B are primary input variables and the filtering and correlation functions for them are known (they are tautologies). Then $Ff(v_1(C))$ and $Ff(v_2(C))$ are existentially implied by $F(I_1(G))$ and $F(I_2(G))$ respectively. According to Proposition 5 , $Ff(v_1(C))$ (respectively $Ff(v_2(C))$) can be derived by resolving clauses of $F(I_1(G))$ (respectively $F(I_2(G))$). Similarly, the correlation function $Cf(v_1(C),v_2(C))$ is existentially implied by $F(I_1(G)) \wedge F(I_2(G))$. So it can be derived from the latter by resolution. After filtering and correlation functions are computed for all the variables of level 1, the same procedure can be applied to variables of topological level 2 and so on. If S consists of n blocks, then in n steps one can deduce correlation functions for the primary output variables of S. At each step two filtering and one correlation function are computed for a variable $C=G(A,B)$ of S. The complexity of this step is no more than 3^{6p}. Indeed, the support of all functions mentioned in Proposition 5 and Proposition 6 needed for computing $Ff(v_1(C))$, $Ff(v_2(C))$ and $Cf(v_1(C),v_2(C))$ is a subset of $A=supp(F(I_1(G))) \cup supp(F(I_2(G)))$.

The total number of gates in $I_1(G)$ and $I_2(G)$ is bounded by $2p$, each gate having 2 inputs and 1 output. So the total number of variables in A cannot be more than $6p$. Then according to Remark 6, in no more than 3^{6p} steps one can deduce CNFs $Ff(v_1(C))$, $Ff(v_2(C))$ and $Cf(v_1(C),v_2(C))$. Then the total number of resolution steps one needs to deduce correlation functions for primary output variables of S is bounded by $n*3^{6p}$.

Now we show that from the correlation functions for primary output variables of S one can deduce an empty clause in the number of resolution steps linear in $n*p$. Let C be a primary output variable specifying the output of a block G of N. Let $I_1(G)$ and $I_2(G)$ be the implementations of G in N_1 and N_2 respectively. Let $|D(\mathrm{C})|=2^k$ (By Assumption 2 the multiplicity of C is a power of 2.) Then $length(q_1(C))= length(q_2(C))=k$. (By Assumption 3, values of S are encoded by a minimal length encoding.)

Now we show that there is always a correlation function $Cf(v_1(C),v_2(C))$ specified by the CNF consisting of k pairs of two literal clauses specifying the equivalence of corresponding outputs of $I_1(G)$ and $I_2(G)$. Let f_1 and f_2 be two Boolean variables of $v_1(\mathrm{C})$ and $v_2(\mathrm{C})$ respectively that specify corresponding outputs of N_1 and N_2. Since S is a CS of N_1 and N_2, then $q_1(\mathrm{C})=q_2(\mathrm{C})$. So any assignment $q_1(c),q_2(c)$ to $v_1(\mathrm{C})$ and $v_2(\mathrm{C})$ that satisfies $Cf(v_1(\mathrm{C}),v_2(\mathrm{C}))$ also satisfies clauses $K'=f_1 \vee \sim f_2$ and $K''=\sim f_1 \vee f_2$. So K' and K'' are implied by $Cf(v_1(C),v_2(C))$ and can be deduced by the procedure described in the proof of Proposition 6. (The resolution steps one needs to deduce equivalence clauses are already counted in the expression $n*3^{6p}$)

Using each pair of equivalence clauses K' and K'' and the clauses specifying the gate $g=\mathrm{XOR}(f_1,f_2)$ of the miter, one can deduce a single literal clause $\sim g$. This clause requires setting the output of this XOR gate to 0. Each such a clause can be deduced in the number of resolutions bounded by a constant and the total number of such clauses cannot be more than $n*p$. Finally, from these unit clauses and the clauses specifying the final OR gate of the miter, the empty clause can be deduced in the number of resolutions bounded by $n*p$. So the empty clause is deduced in no more than $n*3^{6p} + d'*n*p$ steps where d' is a constant. Finally, one can pick a constant d such $n*3^{6p} + d'*n*p \leq d*n*3^{6p}$ □

Remark 9. In Proposition 7 we give a very conservative estimate of the complexity of deducing filtering and correlation functions. In practice this complexity can be much lower. In a sense, the best way to interpret the theory developed in this section is that the problem of equivalence checking of circuits N_1,N_2 with a CS S of n blocks can be partitioned into n subproblems of computing filtering and correlation functions for each variable of S.

6. A Procedure of Equivalence Checking for Circuits with a Known CS

In Section 5 we considered equivalence checking in general resolution that is a non-deterministic proof system. This means that the proof is guided by an oracle that points to the next pair of clauses to be resolved. In this section, we summarize the results of Section 5 as a deterministic procedure of equivalence checking of circuits with a known CS.

Let S be a CS of granularity p of Boolean circuits N_1 and N_2. Let F be the CNF formula specifying the miter of N_1 and N_2. Our procedure of equivalence checking consists of two stages:

1. For each variable C of S compute filtering functions $Ff(v_1(C))$, $Ff(v_2(C))$ and the correlation function $Cf(v_1(C), v_2(C))$ proceeding in topological order of variables. If C is a primary input variable, then $Ff(v_1(C))$, $Ff(v_2(C))$ and $Cf(v_1(C), v_2(C))$ are tautologies. Let $C=G(A,B)$. Then $Ff(v_1(C))$ is built by computing the function existentially implied (see Definition 15) by $Ff(v_1(A)) \vee Ff(v_1(B)) \vee F(I_1(G))$. ($F(I_1(G))$ is a subset of F specifying the implementation of G in N_1. The function $Ff(v_2(C)$ is built similarly to $Ff(v_1(C))$.) The function $Cf(v_1(C),v_2(C))$ is built by computing the function existentially implied by $Ff(v_1(A)) \vee Ff(v_1(B)) \vee Ff(v_2(A)) \vee Ff(v_2(B)) \vee Cf(v_1(A), v_2(A)) \vee Cf(v_1(B), v_2(B)) \vee F(I_1(G)) \vee F(I_2(G))$.
2. Once the correlation functions are computed for every primary output variable of S, finish the proof of unsatisfiabiby of F by invoking a SAT-solver like [6],[12]. (This SAT-solver is applied to the CNF consisting of the clauses describing the correlation functions for the primary output variables of S, the clauses specifying the gates XORing primary outputs of N_1 and N_2 and the final OR gate of the miter.)

The complexity of this procedure is the same as in general resolution i.e. $d*n*3^{6p}$ where d is a constant. This procedure is flexible with respect to the method of computing existentially implied functions. Below we describe a few options.

Let F be a CNF and $supp(F) = X_1 \cup X_2$. Suppose one needs to compute a CNF $H(X_2)$ that is existentially implied by F. If the value of $|X_2|$ is small, one can compute $H(X_2)$ by running 2^k SAT-checks where $k=|X_2|$. For every assignment z to the variables of X_2 one needs to check if there is an assignment y to the variables of X_1 such that (y,z) satisfies F. If such an assignment exists then the next assignment is checked. Otherwise, a clause consisting of literals of variables from X_2 that is falsified by the assignment z is added to the clauses of $H(X_2)$.

If the size of X_2 is large, one can compute filtering and correlation functions by existential quantification of the variables of X_1. In terms of SAT, existential quantification of a CNF F in a variable w of X_1 means adding to F all the resolvents that can be produced by resolving clauses of F in w. Of course, existential quantification in all the variables of X_1 is very expensive in SAT and so it works only for blocks of a small size. However, less expensive methods for computing $G(X_2)$ in terms of SAT can be and should be developed.

Remark 10. For the sake of completeness, we should mention that nothing prevents one from computing filtering and correlation functions using BDDs. This especially makes sense when existential quantification by BDDs is more efficient that in SAT.

7. Experimental Results

The objective of experiments was to show that equivalence checking of circuits with a fine CS S is easy if S is known and is hard otherwise. To produce circuits having a fine CS we used the following procedure. To get multi-valued specifications with realistic topologies we “borrowed” them from MCNC-91 benchmark circuits as follows. First, all the benchmarks were technology mapped using SIS [14] to get circuits consisting only of two-input AND gates. Then from each obtained circuit N a multi-valued specification S was produced by replacing each two-input binary gate with a two-input single output block of four-valued variables. (In other words, S changes the functionality of N while preserving its topology.) Then from S two functionally equivalent Boolean circuits N_1, N_2 implementing S were produced using two different sets of two-bit encodings of four-valued values. The encodings were picked in such a way that the two different implementations of the same four-valued block in N_1 and N_2 had no functionally equivalent outputs. This way we guaranteed that internal functionally equivalent points in N_1 and N_2 may occur only by accident.

Note that after encoding, the number of inputs and outputs in N_1 and N_2 is twice the number of inputs and outputs in the original Boolean circuit N. For instance, the two circuits produced from C6288 used as a "specification" have the topology of a 16-bit multiplier and the number of inputs and outputs of a 32-bit multiplier.

In experiments we used the best tools that were available to us. Namely, we used the SAT-solver BerkMin downloaded from [1], the program Nanotrav built on top of the Colorado University Decision Diagram (CUDD) package [5] and a SAT-based equivalence checker CSAT [10] (courtesy of Prof. Li of UCSB). We also tried the SAT-solver Zchaff [12], but BerkMin was up to three orders of magnitude faster on our formulas. In the experiments we used the special mode of BerkMin designed for equivalence checking that is described at [1]. BerkMin was run on the formula specifying the miter M of N_1 and N_2 as described in Section 4. Nanotrav was used to build a BDD for the miter M and CSAT checked the satisfiabilty of the miter's output. To check the suitability of the tools for equivalence checking, we first ran them on "regular" MCNC benchmarks to verify optimized versus non-optimized circuits. (We do not report these results). The tools showed quite decent performance. For example, BerkMin was able to quickly verify all the instances including the multiplier C6288. The same kind of performance was shown by CSAT. Nanotrav was able to build BDDs for all the miters except C6288 very quickly (in a few seconds). In all the experiments we ran Nanotrav using settings suggested by Fabio Somenzi (private communication). In particular, the variable sifting option was on. In Table 1 we give runtimes of the three programs shown in our experiments. All the programs 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 results of the best out of the three programs is shown in black. In the last column we report run times of a trivial CS driven procedure. This procedure computes filtering and correlation function of blocks in terms of SAT by existentially quantifying variables (as it was described in Section 6) and eventually deduces an empty clause.

It is not hard to see that run times of the CS driven procedure are linear in the size of circuits to be checked for equivalence. This is due to the fact that the size of specification blocks is fixed (and very small). On the other hand, the instances we generated turned out to be hard for the three chosen tools. Even if one compares the best run times with run times of the CS driven procedure, it is not hard to see that the former quickly increased as the size of the instances grew.

Table 1. Equivalence checking of circuits with a fine CS

Name of "specification"	CSAT (sec.)	Nanotrav (BDDs) (sec.)	BerkMin (sec.)	CS driven (sec.)
C880	162.8	60,000	**3.7**	1.1
ttt2	281.0	**1.0**	11.7	1.3
x4	284.3	**4.7**	17.3	1.8
i9	75.3	**1.5**	32.7	2.1
term1	1,604.6	40.9	**35.9**	1.6
c7552	282.0	60,000	**52.8**	3.6
c3540	34,905.8	60,000	**64.1**	2.3
rot	163.6	19,315.6	**72.2**	2.1
9symml	31.07	**1.9**	113.2	0.5
frg2	13,610.4	**22.6**	131.4	2.9
frg1	**265.8**	60,000	330.3	1.7
i10	60,000	60,000	**445.0**	4.8
des	12,520.3	**9.7**	451.7	12.1
dalu	17,496.9	60,000	**518.6**	3.1
x1	13,580.3	13,009.6	**950.2**	2.8
alu4	8,020.4	**135.1**	992.6	2.0
i8	60,000	**98.0**	1,051.5	5.1
c6288	60,000	60,000	**1,955.1**	5.2
k2	60,000	59,392.9	**5,121.5**	4.3
too_large	60,000	60,000	60,000	15.2
t481	60,000	60,000	60,000	6.3

It is unlikely that an industrial strength equivalence checker would do much better on the circuits we generated because they have no functionally equivalent points. Besides, one can always produce much harder equivalence checking problems by using a ***slightly*** more coarse specification (Recall that in the experiments we used a very fine CS S consisting of four-valued blocks. That is the circuits produced from S were "almost" identical.) As we mentioned in the introduction, the problem of finding a short proof of equivalence of N_1,N_2 if a CS is not known, comes down to recovering this CS from the description of N_1,N_2 which is computationally very hard (if not infeasible).

8. Conclusions

We introduce the notion of a common specification (CS) S of Boolean circuits N_1, N_2 as a measure of complexity for equivalence checking of N_1,N_2. We show that if a CS S of n blocks is known, the problem of equivalence checking of N_1,N_2 reduces to n much simpler subproblems. The complexity of each subproblem (that is to compute filtering and correlation functions for a block of S) heavily depends on the granularity p

of *S*. If *p* is small then each subproblem is very simple and so equivalence checking of N_1 and N_2 is extremely fast. On the other hand, equivalence checking of N_1 and N_2 without any knowledge of a CS is most likely hard for any equivalence checker. We experimentally show that circuits having a fine CS *S* can be checked for equivalence in a few seconds if *S* is known, while they cannot be verified in many hours by a representative set of tools.

These results suggest that the granularity of CSs can be considered as the backbone of complexity of equivalence checking. Hence the notion of a CS should be taken into account when designing efficient synthesis and verification procedures.

References

[1] BerkMin web page. http://eigold.tripod.com/BerkMin.html

[2] Bonet M. e.a. *On interpolation and automatization for Frege systems.* SIAM Journal on Computing, 29(6):1939-1967, 2000.

[3] Brand D. *Verification of large synthesized designs*. Proceedings of ICCAD-1993,pp 534-537.

[4] Bryant R. *Graph based algorithms for Boolean function manipulation.* IEEE Trans. on Computers, C(35):677-691.

[5] CUDD web page. http://vlsi.colorado.edu/~fabio/

[6] Goldberg E.,Novikov,Y. *BerkMin: A fast and robust SAT-solver*. Design, Automation, and Test in Europe (DATE '02), pages 142-149, March 2002..

[7] Kuehlmann A., Krohm, F. *Equivalence checking using cuts and heaps.* Proceedings of DAC-1997.

[8] Kunz W., Pradhan, D., *Recursive learning: a new implication technique for efficient solutions to cad problems - test, verification and optimization*. IEEE Tran. on CAD 13(9) Sep. 1994

[9] Kwak H.H e.a. *Combinational equivalence checking through function transformation.* Proc. of ICCAD,1992, pp. 526-534.

[10] Lu F. e.a. *A circuit SAT solver with signal correlation guided learning*,.DATE-2003, pp. 892-898.

[11] Moondanos J. e.a. *Clever: Divide and conquer combinational logic equivalence verification with false negative elimination.* CAV-2001, pp.131-143.

[12] Moskewicz M. e.a. *Chaff: Engineering an efficient SAT-solver.* Proceedings of DAC-2001.

[13] Razborov A., Alekhnovich M. *Resolution is not automatizable unless W[p] is tractable.* Proc. of the 42nd IEEE FOCS-2001, pages 210-219.

[14] Sentovich E. e.a. *Sequential circuit design using synthesis and optimization*. Proceedings of ICCAD, pp 328-333, October 1992.