Equivalence Checking of Dissimilar Circuits II

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

CDNL-TR-2004-0830
August 2004

Eugene Goldberg (Cadence Berkeley Labs), egold@cadence.com

Abstract

In this report we revisit the theory introduced in [2]. We formulate it in terms of correlation functions so showing that the introduction of filtering functions is not necessary. We also describe an algorithm of equivalence checking for circuits with a known specification that is based on computation of correlation functions only (no filtering functions are computed).

1. Introduction

This report is simply a rewrite of paper [2] that presented some results on complexity of equivalence checking of circuits with a common specification. The results of [2] were formulated in terms of correlation and filtering functions. In this report, we show that the introduction of filtering functions is not necessary and the theory of [2] can be formulated in terms of correlation functions only. This also applies to publications [3] and [4] in which the introduction of filtering functions can be avoided either. (Papers [2] and [4] are available online from the web page *http://eigold.tripod.com/papers.html*.)

In the report we follow the structure of paper [2] with the following exceptions. To reduce the repetition of text we omitted material presented in Sections 1, 5 and 6 of [2]. The major changes are made in section 4 which presents the main result. In particular, we added subsection 4.4 in which we explain why filtering functions can be dropped. Finally we added a section describing an algorithm for equivalence checking of circuits N_1, N_2 with a known common specification S (Section 5). This algorithm is identical to that of [4] with the following two exceptions. We formulate this algorithm in terms of correlation functions only (filtering functions are not used). We emphasize the fact that for this algorithm the specification S does not have to be represented *explicitly*. The algorithm only needs to know the partitioning of N_1 and N_2 into subcircuits that are implementations of blocks of S.

2. 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*** A. 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 $D(A)$ the ***domain*** of the variable A associated with a node of H. The value of $|D(A)|$ is called the ***multiplicity*** of A. If the multiplicity of every variable A 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 $D(A)=\{a_1,\ldots,a_t\}$ 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 $q{:}D(A)\rightarrow\{0,1\}^m$. Denote by $length(q(A))$ the number of bits in q that is the value of m. The value of $q(a_i)$, $a_i \in D(A)$ is called the ***code*** of a_i. Given an encoding q of length m of a variable A associated with a block of S, denote by $v(A)$ 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 A is a primary input (or output) variable of a specification, then $length(q(A))=log_2(|D(A)|)$

Assumption 4. If a_1 and a_2 are values of a variable A of a specification and $a_1 \neq a_2$, then $q(a_1) \neq q(a_2)$.

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

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 $v(\mathrm{C})$.
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 $I(z')=q(c)$. Here 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 $c=G(a,b)$.

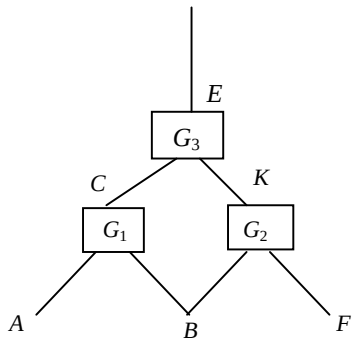
Remark 2. The reason why $Inp(I)$ may not include all the variables of $v(A)$ and/or $v(B)$ is that the function $G(A,B)$ may not distinguish some values of A or B. ($G(A,B)$ does not distinguish, say, values $a_1,a_2 \in D(A)$, if for any $b \in D(B)$, $G(a_1,b)=G(a_2,b)$.) So to implement $G(A,B)$ 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 $x \in v(R)$ is connected to the input of I_2 specified by the same variable of $v(R)$ if $x \in Inp(I_2)$.

In Fig. 1*a* a specification of three blocks is 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 length encoding 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.



(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	0	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

(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

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 Assumption 4 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 $Y=(x_1,..,x_k)$, $x_i \in D(X_i)$ 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.

Proof. The proposition can be proven by induction in topological levels of variables of the specification S. According to Remark 1, the proposition holds for the variables of topological level 0 (primary input variables of S). Let $C=G(A,B)$ be a block of S and $level(G)=n$, $n>0$. Let $I(G)$ be the implementation of G in N. By the induction hypothesis, values taken by the variables of $v(A)$ and $v(B)$ in N under the input assignment y should be $q(a)$ and $q(b)$ respectively. Here a and b are values of variables A and B under the input assignment Y. Then from Definition 3 it follows that the outputs of $I(G)$ take the values of $q(C)$ 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 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. Let y be an arbitrary assignment to input variables of N_1 and N_2. Since the encodings of primary input variables of S in N_1 and N_2 are the same, then the pre-image Y of y for N_1 and N_2 is the same. Let C be a primary output variable of S associated with a block G. From Proposition 1 it follows that the values taken by the implementations $I_1(G)$ and $I_2(G)$ of G in N_1 and N_2 are equal to $q_1(c)$ and $q_2(c)$ respectively. Here c is the value taken by the output of G under input assignment Y and q_1 and q_2 are encodings of the primary output variable C in N_1 and N_2. Since C has the same encoding in N_1 and N_2, then $q_1(c) = q_2(c)$. □

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 N is called ***a common specification*** (CS) of N_1 and N_2.

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.

3. Equivalence Checking as SAT

Since in this report 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*** [1] 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 $\sim x_1 \vee \sim x_2 \vee y$, $x_1 \vee \sim y$, $x_2 \vee \sim y$.

4. 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 the correlation function for encodings $q_1(C),q_2(C)$ of each variable C of S. The two main properties of correlation 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.
- The correlation function for encodings $q_1(C),q_2(C)$ of the variable C specifying the output of a block $G(A,B)$ can be computed "locally" from the correlation functions of encodings 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.

4.1 Class $M(p)$ and general resolution

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

The following Proposition 3 and also Propositions 4,5,6 proved in subsections 4.2, 4.3 are used in the proof of Proposition 7 that is the main result of this paper.

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

4.2 Correlation functions

Definition 16. Let S be a CS of circuits N_1 and N_2 and C be a variable of S. A function $Cf(v_1(C),v_2(C))$ 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 $Cf(z_1, z_2)=0$.

From now on we will say that $Cf(v_1(C),v_2(C))$ is the correlation function for the variable C meaning that it is the correlation function for encodings $q_1(C),q_2(C)$ of the variable C.

Remark 4. If C is a primary input variable of S, then $Cf(v_1(C),v_2(C)) \equiv Eq(v_1(C),v_2(C))$. The function $Eq(z_1, z_2)$ is equal to 1 iff $z_1=q_1(c)$, $z_2 = q_2(c)$, where $c \in D(C)$ and $q_1(c) = q_2(c)$. Otherwise , $Eq(v_1(C),v_2(C))$ is equal to 0. Indeed, as it follows from Remark 1, sets $v_1(C)$ and $v_2(C)$ have the same number of coding variables and 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 $q_1(C)=q_2(C)$. So every "permissible" assignment (x,y) to the variables of $v_1(C),v_2(C)$ can be represented as $(q_1(c),q_2(c))$, were $c \in D(C)$ and $q_1(c) = q_2(c)$.

Proposition 4. 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 3. 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 implies $Cf(v_1(C),v_2(C))$. Here P = *Correlation* $\wedge$ *Implementation* and, *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 *Correlation* to 1, then $x_1=q_1(a)$, $x_2=q_2(a)$ where $a \in D(A)$ and $y_1=q_1(b)$, $y_2=q_2(b)$, where $b \in D(B)$. So $h=(q_1(a),q_2(a), q_1(b),q_2(b), z_1, z_2)$. 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. □

4.3 Complexity of formulas from $M(p)$

Proposition 5. Let A,B,C be Boolean functions and $A \wedge B \rightarrow C$. Then for any function A' such that $A' \rightarrow A$, it is true that $A' \wedge B \rightarrow C$.

Proof. Let x be an assignment that sets $A' \wedge B$ to 1. Then $A'(x)=1$ and $B(x)=1$. Since $A' \rightarrow A$, then $A(x)=1$. Then $A(x) \wedge B(x) = 1$ and so $C(x)=1$.

Proposition 6. 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 in at most $3^{|supp(F)|}$ resolution steps one can derive a CNF formula H' that implies $H(X_2)$ such that $supp(H') \subseteq supp(H)$.

Proof. Let K be a clause of H. From Proposition 3 it follows that in at most $3^{|supp(F)|}$ steps one can derive a clause K' that implies K. Since $K' \rightarrow K$, then $supp(K') \subseteq supp(K)$. So in at most $|H| * 3^{|supp(F)|}$ steps, where $|H|$ is the number of clauses in H, one can derive a CNF H' implying H such that $supp(H') \subseteq supp(H)$. (The fact that H' implies H follows from Proposition 5.) However, if

one does not produce the same resolvent twice, the total number of resolution steps when deriving H' cannot be more than $3^{|supp(F)|}$ (because it is the total number of clauses of $|supp(F)|$ variables).

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 4 it follows that one can deduce correlation 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 correlation functions for them are equal to $Eq(v_1(A),v_2(A))$ and $Eq(v_1(B),v_2(B))$ respectively (see Remark 4). The correlation function $Cf(v_1(C),v_2(C))$ is implied by $P = F(I_1(G)) \wedge F(I_2(G)) \wedge Eq(v_1(A),v_2(A)) \wedge Eq(v_1(B),v_2(B))$. So a function implying $Cf(v_1(C),v_2(C))$ can be derived from P by resolution. From Proposition 5 it follows that to apply Proposition 4, instead of the function $Cf(v_1(C),v_2(C))$, one can use any function implying it. After 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 the correlation function is 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 4 needed for computing $Cf(v_1(C),v_2(C))$ 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 from Proposition 6 it follows that in at most 3^{6p} steps one can deduce CNFs implying $Cf(v_1(C),v_2(C))$. Then the total number of resolution steps one needs to deduce functions implying the correlation functions for the 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 $|D(C)|=2^k$ (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 $Cf(v_1(C),v_2(C))$ that implies 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 $v_1(C)$ and $v_2(C)$ respectively that specify corresponding outputs of N_1 and N_2. Since S is a CS of N_1 and N_2, then $q_1(C)=q_2(C)$. So any assignment $q_1(c),q_2(c)$ to $v_1(C)$ and $v_2(C)$ that satisfies $Cf(v_1(C),v_2(C))$ 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 so clauses implying them can be deduced by the procedure described in the proof of Proposition 4. (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'' (or clauses implying them) and the clauses specifying the gate $g=XOR(f_1,f_2)$ 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 $n*3^{6p} + d'*n*p \le d*n*3^{6p}$ □

Remark 5. The essence of the resolution proof described in Proposition 7 is to compute correlation functions "inductively" moving from inputs to outputs. It is not hard to see that this computation is not restricted to general resolution. Indeed , all the terms of the expression $P=Cf(v_1(A), v_2(A)) \vee Cf(v_1(B), v_2(B)) \vee F(I_1(G)) \vee F(I_2(G))$ are just functions and so can be represented in any possible way (i.e. not only as CNF formulas). Besides, Proposition 7 can be proven in terms of existential quantification introduced by Definition 17. Indeed, from Proposition 8 below it follows that $Cf(v_1(C),v_2(C))$ (or a function implying it) can be obtained from P by existentially quantifying away all the variables except those of $v_1(C) \cup v_2(C)$. Existential quantification of a function can be done in many ways, for example, by using BDDs. So, summarizing, Proposition 7 can be formulated and proven in terms of functions and existential quantification i.e. independently of a proof system.

Definition 17. Let f be a Boolean function. We will say that function f^* is obtained from f by existentially quantifying away Boolean variable x if $f^* = f(\ldots,x=0,\ldots) \vee f(\ldots,x=1,\ldots)$.

Proposition 8. Let X_1 and X_2 be two disjoint sets of Boolean variables. Let $F(X_1,X_2)$ and $H(X_2)$ be two Boolean functions and F imply H. Let $F^*(X_1 \setminus \{x\}, X_2)$ be obtained from $F(X_1,X_2)$ by existentially quantifying away the variable x. Then $F^*(X_1\setminus\{x\},X_2)$ also implies $H(X_2)$.

Proof. Denote by X'_1 the set $X_1\setminus\{x\}$. Let $(z, \mathbf{z}_1', \mathbf{z}_2)$ be a boolean vector representing an assignment to the variables of $X_1 \cup X_2$. Here z is a Boolean value assigned to the variable x and $\mathbf{z}_1'$, $\mathbf{z}_2$ are Boolean vectors representing assignments to the variables of X'_1 and X_2 respectively. Suppose that $F^*(\mathbf{z}_1', \mathbf{z}_2) = 1$. According to Definition 17, $F^*(X'_1,X_2) = F(0, X'_1,X_2) \vee F(1, X'_1,X_2)$ and so either $F(0, \mathbf{z}_1', \mathbf{z}_2)$ or $F(0, \mathbf{z}_1', \mathbf{z}_2)$ has to be equal to 1. Since $F(x,X'_1,X_2)$ implies $H(X_2)$ then $H(\mathbf{z}_2) = 1$. So from $F^*=1$ it follows that $H=1$. Hence F^* implies H.

Remark 6. In Proposition 7 we give a worst case estimate for the complexity of correlation function computation. 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 complexity of equivalence checking of circuits N_1,N_2 with a CS S is linear in the number of blocks in S.

Remark 7. In this report, for the sake of clarity, we assumed that every block of a specification has two inputs and one output (Assumption 1). However, one can easily extend Proposition 7 to the case of a specification S where a block may have an aribtrary (but finite) number of inputs. (We still assume that every gate of circuits N_1 and N_2 implementing S have two inputs and one output). Indeed, let G be a block of S with n inputs and let $C, A_1,.., A_n$ be variables associated with its output

and n inputs respecitvely. Then one can prove (in the same manner as in Proposition 4) that correlation function $Cf(v_1(C),v_2(C))$ is implied by the expression $P = Correlation \wedge Implementation$. Here $Implementation= F(I_1(G)) \vee F(I_2(G))$ is the same as in Proposition 4 and $Correlation = Cf(v_1(A_1), v_2(A_1)) \wedge \ldots \wedge Cf(v_1(A_n), v_2(A_n))$. So to compute the correlation function for the output of an n-input block one needs to compute n correlation functions corresponding to n input variables. Other than that, the proof of Proposition 7 does not change.

4.4 A few words about filtering functions

In subsection 4.3, we reproduced the result of [2] without introducing filtering functions. To make things even more clear, in this subsection we give an informal explanation of why filtering functions can be dropped. We also explain under what circumstances filtering functions might come useful.

Here is the definition of filtering functions from [2].

Definition 18. Let N be an implementation of a specification S. Let C be a variable of S associated with the output of a block G. A function Ff is called ***a filtering function*** if:

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

Let N_1 and N_2 be circuits with a CS S. Let $v_1(C)$ and $v_2(C)$ be the coding variables of the variable C of S corresponding to the implementations $I_1(G)$ and $I_2(G)$ in N_1 and N_2 respectively. From Definition 18 of filtering functions and Definition 16 of correlation functions it follows that

$Ff(v_1(C)) \wedge Ff(v_2(C)) \wedge Cf(v_1(C),v_2(C)) = Cf(v_1(C),v_2(C))$.

On the other hand, in Proposition 7 of [2] (used to prove the main result i.e. Proposition 8) filtering functions $Ff(v_1(C)) \wedge Ff(v_2(C))$ appear only in conjunction with the correlation function $Cf(v_1(C),v_2(C))$. So filtering functions can be removed from the proof.

The reason why filtering functions appeared in [2] was that originally the definition of correlation functions used in the manuscript was as follows.

Definition 19. 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 obtaining N_1 and N_2) if :

- $supp(Cf) \subseteq v_1(C) \cup v_2(C)$.
- $Cf(z_1, z_2)=0$ 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 $c,c^* \in D(C)$ and $c \neq c^*$.

It is not hard to see that Definition 19, only partially defines Cf. Definition 19 can be viewed is a relaxation of Definition 16, meaning that the correlation function specified by the latter is an implementation of the correlation function specified by the former. If correlation functions are specified by Definition 19, then to prove Proposition 7 one needs filtering functions. (Because now $Ff(v_1(C)) \wedge Ff(v_2(C)) \wedge Cf(v_1(C),v_2(C))$, in general, is not equivalent to $Cf(v_1(C),v_2(C))$.) Later, the definition of correlation functions was changed to the one used in [2] which made filtering functions redundant.

Definition 16 is preferable from a "theoretical" point of view because it reduces the number of objects employed in our theory. However, in practical implementations of the algorithm of equivalence checking described in Section 5, the use of filtering and correlation functions specified by Definition 18 and Definition 19 respectively, instead of correlation functions specified by Definition 16, may make sense. The reason is that Definition 16 mixes up two unary and one binary relation specified over the set of output assignments of subcircuits $I_1(G)$ and $I_2(G)$. On the other hand, in practical applications one may want to compute them separately. (The unary relations are specified by the filtering functions $Ff(v_1(C))$ and $Ff(v_2(C))$ that single out output assignments of $I_1(G)$ and $I_2(G)$ that are codes of C. The binary relation is given by the correlation function $Cf(v_1(C),v_2(C))$ specified by Definition 19.)

5. Algorithm of Equivalence Checking with a Known Specification

In this section we describe an algorithm for equivalence checking of circuits with a known specification. This algorithm is identical to the one introduced in [4] with a few exceptions. First, we formulate this algorithm in terms of correlation functions only (omitting filtering functions). Second, we emphasize the fact that this equivalence checking procedure needs only an *implicit* representation of a CS of circuits N_1, N_2. This representation is given as a partitioning of N_1, N_2 into subcircuits.

In Section 4 we considered equivalence checking in general resolution that is a non-deterministic proof system. This means that this proof is guided by an oracle that points to the next pair of clauses to be resolved. Now we summarize the results of Section 4 in a deterministic procedure of equivalence checking of circuits N_1 and N_2 with a CS S of granularity p. The idea is that if a CS S of N_1 and N_2 is known, then S itself can be viewed as an oracle. This oracle is powerful enough to make equivalence checking of N_1 and N_2 efficient. (However, if S is unknown it is unlikely that there is an efficient algorithm for equivalence checking of N_1 and N_2 even if there exists a CS of N_1,N_2 of small granularity.)

For the sake of simplicity, we will assume that all the primary input and output variables of S are binary. (A more general case implied by Assumption 2 and Assumption 3 is not much different but makes explanation more wordy.) We will also assume that N_1 and N_2 have only one primary output. Besides, we give the description of the algorithm that is independent of the proof system (see Remark 5).

Note that in the proof of Proposition 7 we never used an explicit representation of blocks of S. We only needed to know how gates of N_1 and N_2 are assigned to subcircuits that are implementations of blocks of S. So in the algorithm description shown in Figure 2, a k-block specification S of N_1,N_2 is represented *implicitly* as a partitioning of these two circuits into k subcircuits $N_1^1,..,N_1^k$, $N_2^1,..,N_2^k$. We assume that N_1^i and N_2^i are implementations of the same block of specification S. We also

assume that subcircuits are numbered in the topological order of blocks in S. That is if $i > j$, then the topological level of the block implemented by N_1^i is greater or equal to the topological level of the block implemented by N_1^j.

```
    /* ---------------------------
       Part(N1)= {N1^1,..,N1^k},
       Part(N2)= {N2^1,..,N2^k}
    ----------------------------*/
check_for_equivalence(N1, N2, Part(N1),Part(N2))
{
 /* check that specification is correct "topolgically" */
 if (check_partitionins(Part(N1),Part(N2)) == 'incorrect')
     return('unsolved');

 /* compute correlation functions */
 for (i=1; i <= k ; i++)
    {Correlation = comp_inp_corr_func(N1^i,N2^i);
     Cf(N1^i, N2^i) =  comp_out_corr_func(N1^i,N2^i, Correlation);
    }

/* check the correlation function of the last pair of subcircuits */
  if (Cf(N1^k, N2^k)  implies  equivalence_ function)
        return('equivalent');
  else
        return('unsolved');

}
```

Figure 2. Pseudocode of equivalence checking algorithm

The pseudocode of our algorithm for equivalence checking is given in Figure 2. The procedure *check_partitions* checks that specification S represented by $Part(N_1)$,$Part(N_2)$ is correct topologically. Namely, it checks that if outputs of subcircuit N_1^i are (not) connected to inputs of subcircuit N_1^j, then outputs of subcircuit N_2^i should (not) be connected to inputs of N_2^j. If this is not true, the *check_partitions* procedure returns result '*incorrect*'.

In the main loop we compute the correlation functions $Cf(N_1^i,N_2^i)$ in topological order. (Note that in Figure 2 we denote correlation function differently to emphasize the fact that specification S is represented implicitly. Here $Cf(N_1^i,N_2^i)$ denotes what we previously denoted as $Cf(v_1(C),v_2(C))$ where C is the variable associated with the output of the block of S implemented by N_1^i and N_2^i.).

Before computing $Cf(N_1^i,N_2^i)$ the procedure *comp_inp_corr_func* forms the expression *Correlation*. This expression is a conjunction of

- the correlation functions corresponding to subcircuits whose outputs are connected to inputs of N_1^i and N_2^i.
- the correlation functions corresponding to the primary inputs of N_1 and N_2 (if any) that are in the fanin of N_1^i or N_2^i.

If, for example, inputs of N_1^i are connected only to outputs of subcircuits N_1^j and N_1^m (and so inputs of N_2^i are connected only to outputs of N_2^j and N_2^m), then $Correlation = Cf(N_1^j,N_2^j) \wedge Cf(N_1^m,N_2^m)$. On the other hand, if an input x_1 of N_1^i is a primary input of N_1 (and so the corresponding input x_2 of N_2 is a primary input of N_2), then in the conjunction of terms specifying *Correlation* there is term $Eq(x_1,x_2)$ describing the equivalence of x_1 and x_2.

The function *comp_out_corr_func* computes the correlation function $Cf(N_1^i,N_2^i)$ by existentially quantifying the function $P=Implementation \wedge Correlation$. The function $Implementation = F(N_1^i) \wedge F(N_2^i)$ describes consistent assignments to the variables of N_1^i and N_2^i. The function $Cf(N_1^i,N_2^i)$ is obtained from P by existentially quantifying away all the variables of N_1^i and N_2^i except the ones corresponding to outputs of N_1^i and N_2^i.

Finally, the algorithm checks if the correlation function of subcircuits N_1^k and N_2^k (whose primary outputs are primary outputs of N_1 and N_2) implies the equivalence function. If yes, then N_1 and N_2 are equivalent. Otherwise, the algorithm returns the '*unsolved*' answer.

The complexity of the algorithm shown in Figure 2 is the same as in general resolution i.e. $d*n*3^{6p}$ where d is a constant. That is for the class of formulas $M(p)$ with the fixed value of p, the complexity of this algorithm is linear in circuit size.

6. Conclusions

In this report, we prove the results of paper [2] without using the notion of filtering functions. This allows us to simplify the formulation of our theory for equivalence checking of circuits with a common specification. Besides, we give a modified description of the algorithm for equivalence checking of circuits with a known specification. In this description we use only correlation functions (omitting fitering functions) and emphasize the fact that CS is represented implicitly.

7. References

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

[2] E.Goldberg, Y.Novikov. *Equivalence Checking of Dissimilar Circuits.* International Workshop on Logic and Synthesis, May 28-30,2003,USA.

[3] E.Goldberg, *What Sat-solvers can and cannot do.* pp.1-43. in Advanced Formal Verification, edited by Rolf Drechsler,2004, Kluwer Academic Publishers.

[4] E.Goldberg, Y.Novikov. *On complexity of equivalence checking*. Technical Report, CDNL-TR-2003-08026, August, 2003.