On Resolution Proofs for Combinational Equivalence

Satrajit Chatterjee Alan Mishchenko
Robert Brayton

Department of EECS
U. C. Berkeley
{satrajit, alanmi, brayton}@eecs.berkeley.edu

Andreas Kuehlmann

Cadence Berkeley Labs
Cadence
kuehl@cadence.com

ABSTRACT

Modern combinational equivalence checking (CEC) engines are complicated programs which are difficult to verify. In this paper we show how a modern CEC engine can be modified to produce a *proof of equivalence* when it proves a miter unsatisfiable. If the CEC engine formulates the problem as a single SAT instance (call this naïve), one can use the resolution proof of unsatisfiability as a proof of equivalence. However, a modern CEC engine does not directly invoke a SAT solver for the whole miter, but instead uses a variety of techniques such as structural hashing, detection of intermediate functional equivalences, and circuit re-writing to first simplify the problem. We show that in spite of using these simplification techniques, a CEC engine can be modified to generate a *single* (extended) resolution proof for the whole miter just as in the naïve case. The benefit of having a single proof is that the proof verification program remains extremely simple, and *its* correctness is much easier to establish than that of the CEC engine.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: design aids—*verification*

General Terms

Algorithms, Theory, Verification

Keywords

resolution proofs, equivalence checking, transformation-based verification

1. RESOLUTION PROOFS

1.1 Example of a resolution proof

Consider the following unsatisfiable CNF SAT problem (the clauses are numbered for convenience):

DAC 2007, June 4–8, 2007, San Diego, California, USA.

1.	$(\overline{p} \vee a)$	2.	$(\overline{p} \vee b)$	3.	$(p \vee \overline{a} \vee \overline{b})$
4.	$(\overline{q} \vee a)$	5.	$(\overline{q} \vee b)$	6.	$(q \vee \overline{a} \vee \overline{b})$
7.	$(p \vee q \vee \overline{z})$	8.	$(p \vee \overline{q} \vee z)$	9.	$(\overline{p} \vee q \vee z)$
10.	$(\overline{p} \vee \overline{q} \vee \overline{z})$	11.	(z)		

Resolution [3] provides an easy way to check that this problem is indeed unsatisfiable. One takes two clauses that contain a variable, say x, in opposite polarity and constructs the *resolvent*. The resolvent is the disjunction of all the literals in the two clauses excluding x and $\overline{x}$. For example, resolving clauses 3 and 4 w.r.t. variable a produces the clause $(p \vee \overline{b} \vee \overline{q})$. Adding a resolvent to the current set of clauses does not alter satisfiability: The original set of clauses has a satisfying assignment only if the new set does.

Note that the empty clause () can only be obtained by resolving two contradictory unit clauses (x) and $(\overline{x})$. Thus, if a sequence of resolution steps leads to the empty clause, the original set of clauses is verified to be unsatisfiable. As an example, consider the following sequence of resolutions deriving the empty clause. It is a resolution proof of unsatisfiability of the above problem:

12.	$(p \vee \overline{b} \vee \overline{q})$	[from 3 and 4]
13.	$(p \vee \overline{q})$	[from 5 and 12]
14.	$(\overline{p} \vee q \vee \overline{a})$	[from 2 and 6]
15.	$(\overline{p} \vee q)$	[from 1 and 14]
16.	$(\overline{p} \vee \overline{q})$	[from 10 and 11]
17.	$(p \vee q)$	[from 7 and 11]
18.	$(\overline{q})$	[from 13 and 16]
19.	(q)	[from 15 and 17]
20.	()	[from 18 and 19]

1.2 Deriving a resolution proof

Davis and Putnam [3] provided the first complete method to generate a resolution proof for an unsatisfiable set of clauses. However, their method required exponential space, and was quickly superseded by the Davis-Putnam-Logemann-Loveland (DPLL) method [4] of systematic case-splitting on variables. However, DPLL does not directly generate a resolution proof. Since most modern SAT solvers are based on DPLL, they do not directly produce a resolution proof. DPLL can be seen as a resolution scheduling system, and Zhang and Malik [17] showed how a modern SAT solver (they used zChaff [14] as an example) can be modified to generate a resolution proof for unsatisfiable instances.

In order to follow the rest of the paper, it is not necessary to understand the details of Zhang and Malik's construction. We simply assume access to a SAT solver that can generate resolution proofs of unsatisfiable instances.

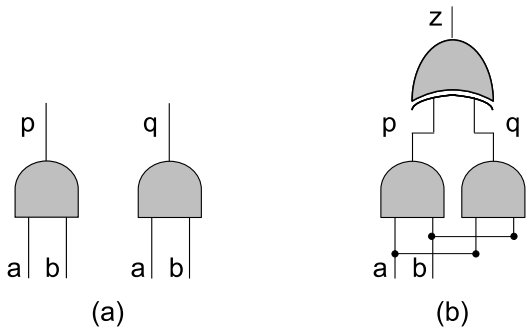

Figure 1: (a) Two circuits to be checked for equivalence. (b) The miter of the two circuits.

2. CEC PROOF VERIFICATION

2.1 Miter

In the combinational equivalence checking (CEC) problem, we are given two combinational circuits and the task is to verify that they are equivalent. This is usually done by constructing a *miter*. A miter is a circuit derived from the two circuits to be checked by connecting corresponding inputs together, adding 2-input XOR gates on top of corresponding outputs, and connecting the outputs of these XOR gates to a large OR gate. The miter has a single output which is the output of this OR gate. If the output of the miter can be shown to be constant 0, it is verified that the two circuits are equivalent.

For example, Figure 1(a) shows two circuits to be checked for equivalence. Figure 1(b) shows the corresponding miter (there is no OR gate at the top since the circuits have only one output each).

For the rest of this paper, we assume that the input to the CEC program is a miter and the task of the CEC program is to prove that the miter implements the constant 0 function.

2.2 Naïve CEC and Proof of Equivalence

It is well known that the combinational equivalence checking problem can be formulated as a SAT instance. Every net in the miter is assigned a variable and clauses corresponding to each gate of the miter are added to the CNF. In addition, the unit clause (z) is corresponding to the output z of the miter is added to the CNF. If the SAT instance is unsatisfiable, then the two circuits are equivalent.

For example, the CNF corresponding to the miter of Figure 1(b) is the same as the example in Section 1.1. Clauses 1-3 and 4-6 correspond to the two AND gates. Clauses 7-10 correspond to the XOR gate, and clause 11 is the unit clause corresponding to the output of the miter z.

Since a naïve equivalence checker directly invokes a SAT solver, the resolution proof generated by the SAT solver suffices as a proof of equivalence. Thus in our running example, the SAT solver would generate a resolution proof such as the one in Section 1.1 i.e. clauses 12-20.

2.3 Proof Verifier

The naïve CEC program writes out the original set of clauses followed by the sequence of resolution steps to a proof file. (Each resolution step is specified by the two clauses and the variable.) The proof verifier takes the proof file and the miter, and first ensures that the original clauses in the file only include those from the miter. Next, it performs the sequence of resolution steps in the proof file. Finally, it checks that the empty clause is derived at the end. Note that the proof verifier is very simple and hence it is easier to establish *its* correctness compared to that of the CEC program and the SAT solver.

2.4 Our Goal

In practice the naïve approach to CEC does not work because the resulting SAT instance is usually hard. Therefore, modern CEC programs (such as [13]) employ a variety of techniques to simplify the SAT problems. The three most important techniques are structural hashing [7, 8], identification of intermediate equivalent points [2, 7, 8, 10, 9], and circuit re-writing (i.e. light-weight logic synthesis) [1, 8, 12]. These techniques are powerful: in some cases, just structural hashing and re-writing suffice to prove the miter, and the SAT solver is not even invoked.

The use of these techniques complicates the proof verification. There is no single SAT problem corresponding to the whole miter: the CEC program may solve a sequence of smaller SAT problems. Furthermore, these SAT problems need not be on the same circuit since structural hashing and re-writing change the circuit structure. As a result, there is no monolithic resolution proof to be checked by the verifier.

So how does one verify the correctness of a modern CEC program? The approach we take in this paper is to modify a modern CEC checker to produce a *single* resolution proof for the miter – just as in the case of the naïve checker – though behind the scenes the program may use structural hashing, re-writing, etc. to prove the miter.

We want to emphasize that the proof verification procedure as outlined in Section 2.3 remains largely unchanged. (The only difference is that if re-writing is used, the generated proof is an *extended* resolution proof.) The verification procedure still takes the miter, and a single (extended) resolution proof and checks the validity of the proof w.r.t. the miter. It is the responsibility of the CEC program to generate a monolithic proof for the whole miter. In effect, the CEC program *constructs* a single proof from the partial proofs corresponding to the techniques it uses to establish equivalence.

3. PORTRAIT OF CEC AS SYNTHESIS

A modern CEC checker may be viewed as a logic synthesis engine that works on the miter to simplify it (eventually to the constant zero function). There are two main techniques used for this: (1) identification and removal of redundant logic, and (2) light-weight logic synthesis through re-writing.

3.1 Identifying and Removing Redundant Logic

At each step, the CEC program reduces the logic in the miter by identifying redundancies. These redundancies take the form of two nets in the circuit that are functionally equivalent. One of the nets (and the logic cone that is used only for that net) can be removed, and the other net can be used to drive the logic driven by the net that is removed. There are two ways in which these redundancies are identified:

1. **Structural Identification.** If there are two gates in

the miter that have the same function and same inputs, then they compute the same function, and their output nets are equivalent.

2. **Functional Identification.** Simulation might indicate that two nets always take on the same values. In that case a SAT problem may be formulated to assert the equivalence of the two nets. If the problem is unsatisfiable, then the two nets are equivalent.

Fanout Transfer. Once two nets are proved equivalent (through structural or functional identification), one net can be disconnected from the pins it drives, and the other net can be used to drive those pins.

Note that structural hashing [8] (without constant propagation) is a combination of structural identification and fanout transfer.

3.2 Re-writing

Logic synthesis using re-writing [1, 12] may be conceptually viewed as a four step process:

1. **Logic insertion.** A new single output logic cone is introduced into the circuit.
2. The output of this cone is "proved" equivalent to an existing node in the circuit (the node that is being re-written). In re-writing, usually this is done by structural hashing or by Boolean matching using truth-tables.
3. The fanouts of the existing node in the circuit are transferred to the output node of the newly introduced cone.
4. The other nodes in the new cone are structurally hashed with existing nodes in the circuit when possible.

The key ingredient of re-writing is Step 1. The remaining steps 2-4 are similar to those for identifying and removing redundant logic. Thus re-writing is seen to be a logic insertion step followed by functional identification, fanout transfer and structural hashing. Furthermore, the logic insertion step can be assumed to be the insertion of a single gate since a logic cone can be constructed by inserting a single gate at a time.

Constant Propagation. Constant propagation may be seen to be a special case of re-writing (where a single gate is replaced by a simpler gate possibly a buffer or a constant).

3.3 Catalog of Elementary Operations

From the above discussion we get a catalog of *elementary* synthesis operations that the CEC engine performs during verification to transform the circuit: (1) structural identification, (2) functional identification, (3) fanout transfer, and (4) logic insertion. We assume that the CEC program proves the miter by a sequence of these elementary operations.

4. GENERATING RESOLUTION PROOFS

4.1 Overview

The proof starts off with the set of clauses corresponding to the miter as described in Section 2.2. As the CEC program proceeds, it performs a sequence of elementary operations (as listed in Section 3.3) to modify the miter. For each elementary operation, a *proof fragment* that corresponds to the operation is generated and added to the proof.

At any point in time during this process we maintain the following invariant: The set of clauses corresponding to every gate in the miter is "present" in the resolution proof. This invariant ensures a correspondence between the miter and the clauses derived by resolution and is called the *correspondence invariant*.

Initially, the invariant holds trivially, since the resolution proof is started with the clauses corresponding to the miter. Of the four elementary operations listed in Section 3.3, fanout transfer and logic insertion are the only two that change the structure of the miter by introducing new gates or by changing existing gates. Therefore, when those operations are executed, we *derive* clauses for the new or modified gates by resolution. (These derivations are the proof fragments.)

Thus when the CEC uses structural identification to prove two nets n and m equivalent, it also generates a proof fragment to derive the clauses $(n \vee \overline{m})$ and $(\overline{n} \vee m)$ from the existing set of clauses in the proof. Later if the CEC program transfers the fanouts of m to n, it generates another proof fragment that uses the previously derived clauses $(n \vee \overline{m})$ and $(\overline{n} \vee m)$ to generate new clauses that correspond to the modified gates in the miter.

Eventually, the CEC program succeeds in proving the miter output – call it z – equivalent to the constant 0 net by functional identification, or by re-writing (recall that constant propagation is a special case of re-writing). The proof fragment corresponding to these operations will generate the unit clause $(\overline{z})$. Resolving this unit clause with the clause (z) which is part of the initial set of clauses leads to the empty clause and completes the proof.

In the rest of this section we look at the nature of proof fragments generated for each of the elementary operations listed in Section 3.3.

4.2 Structural Identification

In structural identification, two nets n and m are established to be equivalent based on the fact that they are driven by the same type of gate, with identical inputs. The goal is to generate a proof fragment to derive the clauses $(n \vee \overline{m})$ and $(\overline{n} \vee m)$ from the existing set of clauses in the proof. This can be efficiently implemented by having a template of the proof fragment that is instantiated with the actual variables corresponding to the variables assigned to the nets. The template depends on the type of gate.

Suppose $n = \text{AND}(x, y)$, and $m = \text{AND}(x, y)$. From the correspondence invariant, we know that the clauses corresponding to the two AND gates already exist in the resolution proof. Let those clauses be numbered as follows:

c_1: $(\overline{n} \vee x)$ c_2: $(\overline{n} \vee y)$ c_3: $(n \vee \overline{x} \vee \overline{y})$
d_1: $(\overline{m} \vee x)$ d_2: $(\overline{m} \vee y)$ d_3: $(m \vee \overline{x} \vee \overline{y})$

The proof fragment generated would be:

t_1: $(n \vee \overline{y} \vee \overline{m})$ [from c_3 and d_1]
t_2: $(n \vee \overline{m})$ [from t_1 and d_2]
t_3: $(\overline{n} \vee m \vee \overline{x})$ [from c_2 and d_3]
t_4: $(\overline{n} \vee m)$ [from c_1 and t_3]

Thus the required clauses $(n \vee \overline{m})$ and $(\overline{n} \vee m)$ are derived from previously derived clauses. For a different pair, say n' = $\text{AND}(x', y')$ and $m' = \text{AND}(x', y')$, we can use the *same* fragment replacing x by x', y by y', n by n' and m by m'. The fragment thus serves as a *template* that can be used to

identify any two structurally equivalent AND nodes in the miter.

Note that if the miter was an And-Inverter Graph, then the only template needed would be the one above for an AND gate. If other types of gates (such as XOR, MUX, etc.) are allowed, they would require their own specific templates. (For complex gates or complicated structural identifications, these templates can be pre-computed by formulating a small equivalence checking problem and obtaining the resolution proof from a SAT solver.)

Example. In the example of Figure 1(a), the two AND gates are structurally identical. Therefore we can instantiate the above template with $x = a$, $y = b$, $n = p$ and $m = q$. Clauses $c_1 - c_3$ and $d_1 - d_3$ correspond to clauses $1 - 3$ and $4 - 6$ in Section 1.1. With that we can obtain the following derivations for $(p \vee \overline{q})$ and $(\overline{p} \vee q)$:

t_1: $(p \vee \overline{b} \vee \overline{q})$ [from 3 and 4]
t_2: $(p \vee \overline{q})$ [from t_1 and 5]
t_3: $(\overline{p} \vee q \vee \overline{a})$ [from 2 and 6]
t_4: $(\overline{p} \vee q)$ [from 1 and t_3]

4.3 Functional Identification

In functional identification, once again the goal is to provide a proof fragment that derives the clauses $(n \vee \overline{m})$ and $(\overline{n} \vee m)$, for two nets n and m that are known to be equivalent. However, in this case the situation is more complicated since the equivalence is established by constructing two SAT instances:

$$1. \quad \mathcal{C} \cdot (n) \cdot (\overline{m}) \qquad\qquad 2. \quad \mathcal{C} \cdot (\overline{n}) \cdot (m)$$

where $\mathcal{C}$ is the set of clauses corresponding to the gates in the cone of logic driving n and m. If both instances are unsatisfiable, then n is equivalent to m.

(Note that if the nets are constant zero or constant one, one problem is trivially unsatisfiable. In what follows we do not consider this case, but it is a straightforward extension.)

At first, it may seem like obtaining the resolution proof in this case should be simple, since the SAT solver can be asked to generate the resolution proof of unsatisfiability. But this does not work. Consider the resolution proof for unsatisfiability of (1). It derives the empty clause, starting from the clauses $\mathcal{C}(n)(\overline{m})$. However, we are interested in generating the clause $(\overline{n} \vee m)$ starting from only $\mathcal{C}$. (i.e. in the monolithic resolution proof that we are constructing, we cannot derive n or $\overline{m}$).

The problem, then, is one of compositionality. Functional identification relies of solving some related SAT problems on the side, and the proofs obtained for the unsatisfiability of those problems need to be *integrated* into the monolithic proof for the miter.

Let $\mathcal{C}$ be a set of clauses in CNF form. Furthermore, suppose that each of the CNFs $\mathcal{C}$, $\mathcal{C} \cdot (n)$, and $\mathcal{C} \cdot (\overline{m})$ is satisfiable, but $\mathcal{C} \cdot (n) \cdot (\overline{m})$ is not.

Consider the set of clauses $\mathcal{C}'$ obtained from $\mathcal{C}$ by *unit propagation* i.e. resolution w.r.t. the unit clauses (n) and $(\overline{m})$ i.e. $\mathcal{C}'$ is obtained from $\mathcal{C}$ by

1. removing all clauses which contain n or $\overline{m}$, and
2. removing literals $\overline{n}$ or m from clauses that contain them

Note that clauses of $\mathcal{C}$ that do not contain variables n and m are present unchanged in $\mathcal{C}'$. Also, $\mathcal{C}'$ does not contain the variables n or m.

Clause Pre-image. Every clause c in $\mathcal{C}'$ comes from a unique clause in $\mathcal{C}$ called its *pre-image*, denoted by pre(c).

LEMMA 1. *If $\mathcal{C} \cdot (n) \cdot (\overline{m})$ is unsatisfiable, so is $\mathcal{C}'$.*

PROOF. Suppose not. Then a satisfying assignment of $\mathcal{C}'$ extended with $n = 1$ and $m = 0$ can be seen to be a satisfying assignment of $\mathcal{C} \cdot (n) \cdot (\overline{m})$. This is a contradiction. □

Let R' be a resolution proof of unsatisfiability of $\mathcal{C}'$, i.e. R' is a sequence of resolution steps starting from the clauses in $\mathcal{C}'$ and deriving the empty clause ().

Proof Lifting. Since every clause in $\mathcal{C}'$ has a unique pre-image in $\mathcal{C}$, and every variable in $\mathcal{C}'$ is present in $\mathcal{C}$, the resolution steps in R' can be performed on corresponding pre-image clauses in $\mathcal{C}$ yielding a valid resolution derivation R. R is called the *lifting* of R'. For each resolvent clause in R', there is a corresponding resolvent in R.

THEOREM 1. *Given $\mathcal{C}$, $\mathcal{C} \cdot (n)$, and $\mathcal{C} \cdot (\overline{m})$ satisfiable, and $\mathcal{C} \cdot (n) \cdot (\overline{m})$ not. Let $\mathcal{C}'$ be derived from $\mathcal{C}$ by unit propagation of n and $\overline{m}$. Let R' be the resolution proof of unsatisfiability of $\mathcal{C}'$. The lifting R of R' derives the clause $(\overline{n} \vee m)$.*

PROOF. (Sketch.)

1. A resolution proof of unsatisfiability (i.e. a sequence of resolution steps) can be viewed as a binary tree where the leaves are the original clauses in the SAT problem, the root is the empty clause and intermediate nodes are resolvents.

2. Claim: In the resolution proof R', there exist leaves c_1 and c_2 (possibly the same) such that $\overline{n}$ is a literal of pre(c_1) and m is a literal of pre(c_2).

   PROOF. Suppose there is no such c_1. This means no leaf of R' is a result of unit propagation due to n. Thus R' can serve as a proof of unsatisfiability of $\mathcal{C} \cdot (\overline{m})$. This is a contradiction since $\mathcal{C} \cdot (\overline{m})$ is assumed to be satisfiable. Similarly for c_2. □

3. Claim: The root of R must be a clause containing $\overline{n}$ and m.

   PROOF. In R, there is no resolution w.r.t. the variables n or m, since every resolution in R corresponds to a resolution in R', and the leaves of R' do not contain the variables n or m. From Step 2, we know that some leaves of R contain the literals $\overline{n}$ and m, and so these literals must be present in the root. □

4. Claim: The root of R cannot contain any literals other than $\overline{n}$ and m.

   PROOF. By contradiction. Suppose the root of R contains some other literal x. That means there exists a path from the root to a leaf clause containing x along which there is no resolution w.r.t. to the variable x. Look at the corresponding path in R'. Since there is no resolution w.r.t. x along that path in R', the root clause of R' must also contain x which is a contradiction since the root clause of R' is empty. □

From 3 and 4, it is clear that the root clause of R contains only $\overline{n}$ and m and hence must be $(\overline{n} \vee m)$. $\square$

Thus, by presenting the problem $\mathcal{C}'$ (instead of $\mathcal{C} \cdot (n) \cdot (\overline{m})$) to the SAT solver, we can lift the resolution proof generated by the solver to derive the clause $(\overline{n} \vee m)$. Similarly, the clause $(n \vee \overline{m})$ can also be derived.

We note that it may not even be necessary to explicitly construct the problem $\mathcal{C}'$. When the SAT solver is used in incremental mode (as is commonly the case), the clauses (n) and $(\overline{m})$ are added in incremental mode as top-level assignments, and the solver performs unit propagation immediately. In this incremental setting, Zhang and Malik's method may be modified to directly produce the resolution proof of $(\overline{n} \vee m)$.

Example. Consider the miter in Figure 1(b). Although, structural identification would easily show that p and q are equivalent, it also provides a simple example for functional identification. The clauses 1-6 of Section 1.1 constitute the set $\mathcal{C}$. Consider the following set of clauses $\mathcal{C}'$ obtained from $\mathcal{C}$ by unit propagation of p and $\overline{q}$:

id:	clause	pre-image
$1'$:	(a)	[1: $(\overline{p} \vee a)$]
$2'$:	(b)	[2: $(\overline{p} \vee b)$]
$6'$:	$(\overline{a} \vee \overline{b})$	[6: $(q \vee \overline{a} \vee \overline{b})$]

The resolution proof of unsatisfiability of $\mathcal{C}'$ is:

r_1: $(\overline{b})$ [from $1'$ and $6'$]
r_2: $()$ [from $2'$ and r_1]

Applying lifting (i.e. performing the same sequence of resolutions on the pre-images of the clauses) gives the following fragment:

l_1: $(\overline{p} \vee q \vee \overline{b})$ [from 1 and 6]
l_2: $(\overline{p} \vee q)$ [from 2 and l_1]

Thus the required clause $(\overline{p} \vee q)$ is derived from the clauses in the original resolution proof. Similarly by considering $\mathcal{C}$ and unit propagation of $\overline{p}$ and q, we can derive $(p \vee \overline{q})$.

4.4 Fanout Transfer

When the fanouts of a net m are transfered to another net n, every gate in the miter with m as an input is modified to use n as input. To maintain the correspondence invariant, new clauses have to be derived (using resolution) from the current clauses to reflect the change in the miter. These clauses are derived as follows. First, for this fanout transfer to be sound, the following clauses must already have been derived:

f_1: $(\overline{n} \vee m)$ f_2: $(n \vee \overline{m})$

(Clauses f_1 and f_2 assert the equivalence of nets m and n, and they would have been derived as a result of structural or functional identification.)

Second, each clause in the proof that has the literal m, is resolved with f_2 to get a corresponding clause with the literal n. Similarly, each clause in the proof that has the literal $\overline{m}$ is resolved with f_1 to obtain a corresponding clause with the literal $\overline{n}$.

Example. We continue with the example of Figure 1, and the clauses 1-10 as listed in Section 1.1. Suppose we have derived the following clauses using structural or functional identification.

f_1: $(\overline{p} \vee q)$ f_2: $(p \vee \overline{q})$

When we transfer the fanouts of q to p (only one fanout in this case), we obtain the following resolvents:

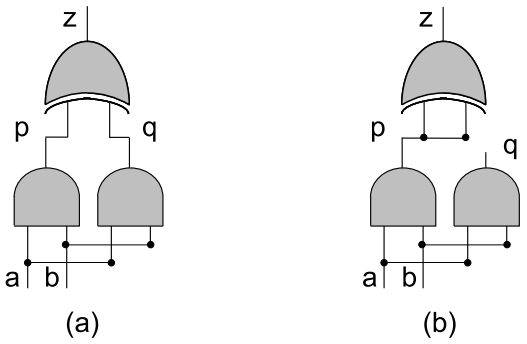

Figure 2: (a) Miter where p and q are known to be equivalent. (b) The miter obtained by transferring the fanout of q to p.

t_1: $(p \vee \overline{z})$ [from f_2 and 7]
t_2: $(p \vee \overline{p} \vee z)$ [from f_1 and 8]
t_3: $(\overline{p} \vee p \vee z)$ [from f_2 and 9]
t_4: $(\overline{p} \vee \overline{z})$ [from f_1 and 10]

Note that some redundant clauses may be obtained as a result of this step.

4.5 Logic Insertion

In logic insertion, a new gate is added to the miter. In practice, this is the simplest of the four operations, since only clauses for the new gate are introduced in the resolution proof. (No actual resolution steps are needed.)

However, conceptually, logic insertion requires upgrading our proof system. We need to use *extended resolution* [16] instead of resolution in order to accommodate the new clauses that are introduced on account of the new gate.

In an extended resolution proof, in addition to clauses derived using resolution, one can *introduce* new free variables. Thus given a set of clauses, say $\mathcal{C}$ over variables x_1, x_2, .. x_n, one can introduce a new clause of the type $(t \equiv f(x_1, x_2, .. x_n))$ where f is some function of x_1, x_2, .. x_n and t is a new variable. Soundness is easy to check: $\mathcal{C}$ is satisfiable iff $\mathcal{C} \cdot (t \equiv f(x_1, x_2, ..x_n))$ is. (Since t does not occur in $\mathcal{C}$, a satisfying assignment of $\mathcal{C}$ can be extended by assigning to t the value of $f(x_1, x_2, .. x_n)$ under the assignment. The other direction is trivial.)

We note that the actual clause introduced into the resolution proof would not be of the form $(t \equiv f(x_1, x_2, .. x_n))$, since it is not in CNF. Instead a set of CNF clauses corresponding to this clause would be inserted.

Example. Consider the introduction of a logic cone comprising a single AND gate on inputs a and b. Let t be a variable that doesn't occur in the proof so far. Instead of introducing the clause $(t \equiv (a \wedge b))$, one would introduce the three CNF clauses $(\overline{t} \vee a)$, $(\overline{t} \vee b)$, and $(t \vee \overline{a} \vee \overline{b})$.

Thus the verifier is modified to accept a set of clauses of the form:

e_1: $(\overline{t} \vee a)$ [clause 1 of $t \equiv \text{AND}(a, b)$]
e_2: $(\overline{t} \vee b)$ [clause 2 of $t \equiv \text{AND}(a, b)$]
e_3: $(t \vee \overline{a} \vee \overline{b})$ [clause 3 of $t \equiv \text{AND}(a, b)$]

The verifier checks that the variable t is a fresh variable (i.e. is not mentioned previously in the proof), and that the correct set of 3 clauses is introduced. For every type of gate that can be introduced by logic insertion, one modifies

the verifier to accept the corresponding set of clauses. If the CEC program uses an And-Inverter Graph, only AND gates can be introduced, and so the verifier needs only the above modification. Note that this modification does not significantly add to the complexity of the verifier.

Example. Consider the example of Figure 2(b). During re-writing we may realize that we can replace the XOR gate with the constant zero gate. Thus a new gate (with no inputs) whose output is 0 is introduced into the miter. Let the output of the gate be u. The corresponding clause is $(\overline{u})$ is introduced using extended resolution.

To complete the example, functional identification (or even structural identification with a suitable template) would generate a proof of the equivalence of z and u, and derive the clauses $(\overline{z} \vee u)$ and $(z \vee \overline{u})$. Next, fanout transfer would derive the clause (u) from clauses (z) (clause 11 in Section 1.1) and $(\overline{z} \vee u)$. The empty clause would be derived from (u) and $(\overline{u})$ thus completing the proof.

5. RELATED WORK

To our knowledge, there is no prior work in the literature on generating proofs of equivalence in state-of-the-art equivalence checkers that use structural hashing, intermediate functional equivalences, re-writing, etc.

There is, however, some interesting work on generating extended resolution proofs from BDDs [15, 6]. The techniques described in those papers could be used in conjunction with the ones described herein to provide an unified proof format for "multi-engine" CEC programs that use both SAT solvers and BDDs.

We note here that Sinz and Biere [15, Section 1] point out that extended resolution so far has mainly been of theoretical interest, since extended resolution proofs are difficult to find. However, like them, we find extended resolution to be a convenient way of expressing proofs found through other means (in our case through re-writing). Thus, the techniques presented in this paper may be seen as another way of generating (short) extended resolution proofs for propositional problems.

6. CONCLUSIONS

We have shown that it is possible to generate a *single* (extended) resolution proof for the unsatisfiability of the miter, even though the CEC engine internally uses techniques such as structural hashing, detection of intermediate functional equivalences, and circuit re-writing. The advantage of generating a single resolution proof is that it is very easy to check, and is independent of the methods used by the CEC engine.

A key challenge in the practical implementation of these ideas relates to the size of the resolution proofs that are generated. Goldberg and Nobikov present a method to obtain a more compact representation of the resolution proof generated by a SAT solver at the expense of increasing the verifier complexity [5]. It would be interesting to explore similar tradeoffs between the size of the proof and the complexity of the verifier in the context of the method presented here.

Some of the techniques described in this paper may be useful for a correct-by-construction logic synthesis flow in which the synthesis tool directly generates a resolution proof of equivalence of the original circuit and the synthesized circuit. This would eliminate the need for an equivalence checker. A more challenging project would be to extend these ideas to the sequential synthesis and verification domains since inductive proofs of sequential equivalence can be expressed using resolution. Furthermore, resolution proofs generated by this method could have other applications such as generating interpolants for use in sequential equivalence checking [11].

7. ACKNOWLEDGMENTS

We thank Kaushik Ravindran for his comments on an early version of this paper. This work was partly supported by SRC (contract #1361.001) and the California Micro Program with our industrial sponsors Altera, Intel, Magma, and Synplicity.

8. REFERENCES

[1] P. Bjesse and A. Boralv, "DAG-aware circuit compression for formal verification", Proc. ICCAD 2004, pp. 42-49.

[2] D. Brand, "Verification of large synthesized designs," in Proc. ICCAD 1993, pp. 534-537.

[3] M. Davis and H. Putnam, "A computing procedure for quantification theory," Journal of the ACM, vol. 7, pp. 201-215, 1960.

[4] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," Comm. ACM, vol. 5, pp. 394-397, 1962.

[5] E. Goldberg and Y. Novikov, "Verification of Proofs of Unsatisfiability for CNF Formulas," In Proc. DATE 2003, pp. 886-891.

[6] T. Jussila, C. Sinz, and A. Biere, "Extended resolution proofs for symbolic SAT solving with quantification," In Proc. 9th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT '06), Lecture Notes in Computer Science (LNCS), vol. 4121, Springer 2006.

[7] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," Proc. DAC 1997, pp. 263-268.

[8] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification," IEEE Trans. CAD, Vol. 21(12), 2002, pp. 1377-1394.

[9] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," Proc. ICCAD 2004, pp. 50-57.

[10] W. Kunz, "HANNIBAL: An efficient tool for logic verification based on recursive learning," in Proc. ICCAD 1993, pp. 538-543.

[11] K. McMillan, "Interpolation and SAT-based Model Checking," in Proc. CAV 2003, pp. 1-13.

[12] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," Proc. DAC 2006, pp. 532-536.

[13] A. Mishchenko, S. Chatterjee, R. Brayton and N. Eén, "Improvements to combinational equivalence checking," Proc. ICCAD 2006, pp. 836-843.

[14] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver," Proc. DAC 2001, pp. 530-535.

[15] C. Sinz and A. Biere, "Extended resolution proofs for conjoining BDDs," In Proc. 1st Intl. Computer Science Symp. in Russia (CSR 2006), St. Petersburg, Russia, Lecture Notes in Computer Science (LNCS), vol. 3967, Springer 2006.

[16] G. Tseitin, "On the complexity of derivation in propositional calculus," In Studies in Constructive Mathematics and Mathematical Logic, 1970.

[17] L. Zhang and S. Malik, "Validating SAT solvers using an independent resolution-based checker: practical implementations and other applications," Proc. DATE 2003, pp. 880-885.