Equivalence Checking By Logic Relaxation

Eugene Goldberg
email: eu.goldberg@gmail.com

***Abstract*—We introduce a new framework for Equivalence Checking (EC) of Boolean circuits based on a general technique called Logic Relaxation (LoR). LoR is meant for checking if a propositional formula G has only "good" satisfying assignments specified by a design property. The essence of LoR is to relax G into a formula G^{rlx} and compute a set S that contains all assignments that satisfy G^{rlx} but do not satisfy G. If all bad satisfying assignments are in S, formula G can have only good ones and the design property in question holds. Set S is built by a procedure called partial quantifier elimination.**

The appeal of EC by LoR is twofold. First, it facilitates generation of powerful *inductive proofs*. Second, proving inequivalence comes down to checking the existence of some assignments satisfying G^{rlx} i.e. a *simpler* version of the original formula. We give experimental evidence that supports our approach.

1. Introduction

A. Motivation

Our motivation for this work is threefold. First, **Equivalence Checking (EC)** is a crucial part of hardware verification. Second, more efficient EC enables more powerful logic synthesis transformations and so strongly impacts design quality. Third, intuitively, there should exist robust and efficient EC methods meant for combinational circuits computing values in a "similar manner". Once discovered, these methods can be extended to EC of sequential circuits and even software.

B. Proving equivalence by induction

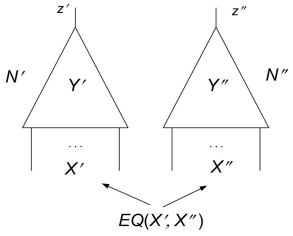

Fig. 1. Equivalence checking of N' and N''

Let $N'(X', Y', z')$ and $N''(X'', Y'', z'')$ be single-output circuits to be checked for equivalence. Here X' and Y' specify the sets of input and internal variables of N' respectively and z' specifies the output variable of N'. The same applies to X'', Y'', z'' of circuit N''. A traditional way to verify the equivalence of N' and N'' is to form a two-output circuit shown in Fig. 1 and check if $z' \neq z''$ for some input assignment ($\boldsymbol{x'}$,$\boldsymbol{x''}$) where $\boldsymbol{x'}$=$\boldsymbol{x''}$. Here $\boldsymbol{x'}$ and $\boldsymbol{x''}$ are assignments to variables of X' and X'' respectively. (By saying that $\boldsymbol{p}$ is an assignment to a set of variables V, we will assume that $\boldsymbol{p}$ is a *complete* assignment unless otherwise stated. That is every variable of V is assigned a value in $\boldsymbol{p}$.)

Formula $EQ(X', X'')$ relating inputs of N' and N'' in Fig. 1 evaluates to 1 for assignments $\boldsymbol{x'}$ and $\boldsymbol{x''}$ to X' and X'' iff $\boldsymbol{x'}$=$\boldsymbol{x''}$. Usually, N' and N'' are just assumed to share the same set of input variables. In this paper, for the sake of convenience, we separate input variables of N' and N'' but assume that N' and N'' must be equivalent only for input assignments satisfying $EQ(X', X'')$.

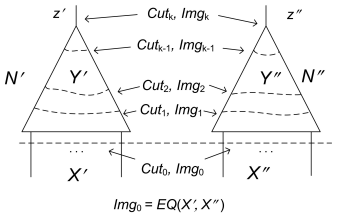

Fig. 2. An inductive proof of equivalence

A natural way to prove equivalence of N' and N'' is to build a sequence of cuts as shown in Fig. 2 and compute relations between cut points of N' and N'' [3], [11], [17]. A straightforward method of computing relations between cut points is to build formulas Img_i specifying **cut images**. The image of i-th cut is the set of all assignments to Cut_i that can be produced in N', N'' by input assignments satisfying $EQ(X', X'')$. Here $Cut_0 = X' \cup X''$, $Img_0 = EQ(X', X'')$ and $Cut_k = \{z', z''\}$. Circuits N' and N'' are equivalent iff $Img_k(z', z'') \rightarrow (z' \equiv z'')$. Formula Img_{i+1} can be derived from formula Img_i and formula specifying the gates located between i-th and $(i+1)$-th cuts. For that reason we will refer to the proofs employing a sequence of cuts as **proofs by induction**.

EC based on computing cut images is inefficient because the size of formulas Img_i is, in general, prohibitively large. In EC by logic relaxation, cut image formulas are replaced with formulas that, for structurally similar circuits, are dramatically simpler than the former.

C. EC by logic relaxation

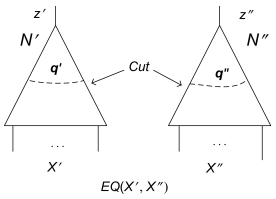

Fig. 3. A cut in N' and N''

Let Img_{cut} be a cut image formula built for the cut shown in Fig. 3. A cut assignment can be represented as ($\boldsymbol{q'}$,$\boldsymbol{q''}$) where $\boldsymbol{q'}$ and $\boldsymbol{q''}$ are assignments to cut variables of N' and N'' respectively. We will say that formula Img_{cut} **excludes** cut assignment ($\boldsymbol{q'}$,$\boldsymbol{q''}$) if the latter falsifies the former. The set of all cut assignments excluded by Img_{cut} can be represented as a union of sets $S^{cut}_{N'}$, $S^{cut}_{N''}$ and S_{rlx}. Assignment $(\boldsymbol{q'}, \boldsymbol{q''})$ is in

- set $S^{cut}_{N'}$ if no input $\boldsymbol{x'}$ of N' can produce $\boldsymbol{q'}$
- set $S^{cut}_{N''}$ if no input $\boldsymbol{x''}$ of N'' can produce $\boldsymbol{q''}$
- set S_{rlx} if there is an input ($\boldsymbol{x'}$,$\boldsymbol{x''}$), $\boldsymbol{x'} \neq \boldsymbol{x''}$ for which ($\boldsymbol{q'}$,$\boldsymbol{q''}$) is produced but the latter cannot be produced if inputs are constrained by $EQ(X', X'')$.

Informally, set S_{rlx} specifies the cut assignments that can be produced only when inputs are *relaxed* i.e. are not constrained

by formula $EQ(X', X'')$.

The essence of EC by Logic Relaxation (LoR) is to replace computation of cut image formulas with that of so-called boundary formulas. A **boundary formula** H_{cut} is implied by Img_{cut} and excludes only a small subset of cut assignments excluded by Img_{cut}. Namely, only exclusion of assignments of S_{rlx} is mandatory for H_{cut}. If $(\boldsymbol{q'}, \boldsymbol{q''}) \in S^{cut}_{N'} \cup S^{cut}_{N''}$, the value of H_{cut} can be arbitrary. This means that H_{cut} depends on the *relation* between N' and N'' (specified by S_{rlx}) rather than their *individual functionality* (specified by $S^{cut}_{N'}$ and $S^{cut}_{N''}$.) We call formula H_{cut} boundary because it describes the difference i.e. a "boundary" between original and relaxed EC problems.

Computing a boundary formula H_{cut} for the cut $\{z', z''\}$ either immediately solves EC of N' and N'' or requires a few simple SAT-checks to finish it. Suppose $H_{cut}(z', z'')$ evaluates to 0 for assignment $\boldsymbol{z_1} = (z' = 0, z'' = 1)$ and assignment $\boldsymbol{z_2} = (z' = 1, z'' = 0)$. Then $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$ cannot be produced when inputs are constrained by $EQ(X', X'')$ because $Img_{cut} \rightarrow H_{cut}$ entails $\overline{H}_{cut} \rightarrow \overline{Img}_{cut}$. So N' and N'' are *equivalent*. If $H_{cut}(z', z'')$ evaluates to 1, say, for $\boldsymbol{z_1}$ above, one needs to check if $(z' = 0, z'' = 1)$ can be produced when inputs are relaxed. This comes down to checking that N' is not constant 1 and N'' is not constant 0. If this is the case, i.e. N' and N'' can produce outputs 0 and 1 respectively, then assignment $\boldsymbol{z_1}$ can also be produced when inputs are constrained by $EQ(X', X'')$. (Otherwise, H_{cut} would evaluate to 0 under $\boldsymbol{z_1}$.) So N' and N'' are *inequivalent*.

D. The appeal of EC by LoR

The appeal of EC by LoR is threefold. First, boundary formulas are much smaller and easier to compute than cut image formulas. Generation of Img_{cut} requires solving the quantifier elimination problem whereas boundary formula H_{cut} can be found by partial quantifier elimination (PQE). In PQE, only a part of the formula is taken out of the scope of quantifiers. So PQE can be much more efficient than complete quantifier elimination.

Second, similarly to cut image formulas, boundary formulas can be computed by induction for a sequence of cuts starting with cut $X' \cup X''$ and ending with cut $\{z', z''\}$. So EC by LoR facilitates generation of inductive proofs. These proofs do not require the existence of particular relations like equivalence between internal points of N' and N''. So they are much more robust than inductive proofs generated in existing approaches (see, for example, [11], [12], [17]).

Third, the machinery of boundary formulas facilitates proving inequivalence. Let $F_{N'}(X', Y', z')$ and $F_{N''}(X'', Y'', z'')$ be formulas specifying N' and N'' respectively. We will say that a Boolean formula F_N specifies circuit N if every assignment satisfying F_N is a consistent assignment to variables of N and vice versa. (An assignment to variables of N is called *consistent* if, for every gate g of N, the value assigned to the output of g is implied by the values assigned to its input variables.) We will assume that all formulas mentioned in this paper are Boolean formulas in Conjunctive Normal Form (CNF) unless otherwise stated.

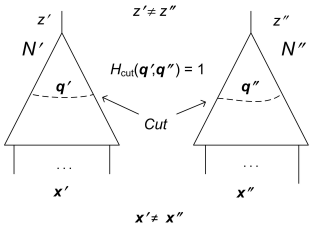

Fig. 4. Using a boundary formula for bug hunting

Circuits N' and N'' are inequivalent iff formula $EQ(X', X'') \wedge F_{N'} \wedge F_{N''} \wedge (z' \not\equiv z'')$ is satisfiable. Denote this formula as α. As we show in this paper, α is equisatisfiable with formula β equal to $H_{cut} \wedge F_{N'} \wedge F_{N''} \wedge (z' \not\equiv z'')$. Here H_{cut} is a boundary formula computed with respect to a cut (see Fig. 4.) In general, formula β is easier to satisfy than α for the following reason. Let $\boldsymbol{p}$ be an assignment satisfying formula β . Let $\boldsymbol{x'}$ and $\boldsymbol{x''}$ be the assignments to variables of X' and X'' respectively specified by $\boldsymbol{p}$. Since variables of X' and X'' are not constrained by $EQ(X', X'')$ in formula β, in general, $\boldsymbol{x'} \neq \boldsymbol{x''}$ and so $\boldsymbol{p}$ does not satisfy α. Hence, neither $\boldsymbol{x'}$ nor $\boldsymbol{x''}$ are a counterexample. They are just inputs producing cut assignments $\boldsymbol{q'}$ and $\boldsymbol{q''}$ (see Fig. 4) such that a) $H_{cut}(\boldsymbol{q'}, \boldsymbol{q''}) = 1$ and b) N' and N'' produce different outputs under cut assignment ($\boldsymbol{q'}$,$\boldsymbol{q''}$). To turn $\boldsymbol{p}$ into an assignment satisfying α one has to do *extra work*. Namely, one has to find assignments $\boldsymbol{x'}$ and $\boldsymbol{x''}$ to X' and X'' that are *equal to each other* and under which N' and N'' produce cut assignments $\boldsymbol{q'}$ and $\boldsymbol{q''}$ above. Then $\boldsymbol{x'}$ and $\boldsymbol{x''}$ specify a counterexample. So the equisatisfiability of α and β allows one to prove N' and N'' inequivalent (by showing that β is satisfiable) without providing a counterexample.

E. Contributions and structure of the paper

Our contributions are as follows. First, we present a generic method of EC based on LoR. This method is formulated in terms of a new technique called PQE that is a "light" version of quantifier elimination. Showing the potential of PQE for building new verification algorithms is our second contribution. Third, we provide a theoretical proof that boundary formulas computed in EC by LoR are small for a broad class of structurally similar circuits. Fourth, we give experimental evidence in support of EC by LoR.

The structure of this paper is as follows. In Section 2, we show the correctness of EC by LoR and relate the latter to PQE. Boundary formulas are discussed in Section 3. Section 4 presents an algorithm of EC by LoR. Section 5 describes how one can apply EC by LoR if the power of a PQE solver is not sufficient to compute boundary formulas precisely. Section 6 provides experimental evidence in favor of our approach. In Section 7, some background is given. We make conclusions in Section 8.

2. Equivalence Checking By LoR And PQE

In this section, we prove the correctness of Equivalence Checking (EC) by Logic Relaxation (LoR) and relate the latter to Partial Quantifier Elimination (PQE).

A. Complete and partial quantifier elimination

In this paper, by a quantified formula we mean one with *existential* quantifiers. Given a quantified formula

$\exists W[A(V,W)]$, the problem of *quantifier elimination* is to find a quantifier-free formula $A^*(V)$ such that $A^* \equiv \exists W[A]$. Given a quantified formula $\exists W[A(V,W) \wedge B(V,W)]$, the problem of **Partial Quantifier Elimination** (**PQE**) is to find a quantifier-free formula $A^*(V)$ such that $\exists W[A \wedge B] \equiv A^* \wedge \exists W[B]$. Note that formula B remains quantified (hence the name *partial* quantifier elimination). We will say that formula A^* is obtained by **taking A out of the scope of quantifiers** in $\exists W[A \wedge B]$. Importantly, there is a strong relation between PQE and the notion of *redundancy* of a subformula in a quantified formula. In particular, solving the PQE problem above comes down to finding $A^*(V)$ implied by $A \wedge B$ that makes A redundant in $A^* \wedge \exists W[A \wedge B]$. Indeed, in this case, $\exists W[A \wedge B] \equiv A^* \wedge \exists W[A \wedge B] \equiv A^* \wedge \exists W[B]$.

Importantly, redundancy in a quantified formula is *much more powerful* than that in a quantifier-free formula [9]. For instance, if formula $F(V)$ is satisfiable, every clause of F is redundant in formula $\exists V[F]$. (A **clause** is a disjunction of literals. We will use the notions of a CNF formula $C_1 \wedge .. \wedge C_p$ and the set of clauses $\{C_1, \ldots, C_p\}$ interchangeably.) On the other hand, a clause C is redundant in a *quantifier-free* formula F *only* if C is implied by $F \setminus \{C\}$.

Let $G(V)$ be a formula implied by B. Then $\exists W[A \wedge B] \equiv A^* \wedge G \wedge \exists W[B]$ entails $\exists W[A \wedge B] \equiv A^* \wedge \exists W[B]$. In other words, clauses implied by the formula that remains quantified are *noise* and can be removed from a solution to the PQE problem. So when building A^* by resolution it is sufficient to use only the resolvents that are descendants of clauses of A. For that reason, in the case formula A is much smaller than B, PQE can be dramatically faster than complete quantifier elimination. Describing how PQE is solved is beyond the scope of this paper. A brief discussion of a PQE algorithm and recall of the necessary background is given in the technical report [7] presenting a complete version of this paper. The relevant results are described in [8], [9], [10] in more detail.

B. Proving equivalence/inequivalence by LoR

Proposition 1 below shows how one proves[1] equivalence/inequivalence of circuits by LoR. Let formula G denote $EQ \wedge F_{N'} \wedge F_{N''}$ and formula G^{rlx} denote $F_{N'} \wedge F_{N''}$. Recall from Subsection 1-D that $F_{N'}(X', Y', z')$ and $F_{N''}(X'', Y'', z'')$ specify circuits N' and N'' respectively. Formula $EQ(\boldsymbol{x'}, \boldsymbol{x''})$ evaluates to 1 iff $\boldsymbol{x'}$=$\boldsymbol{x''}$ where $\boldsymbol{x'}$ and $\boldsymbol{x''}$ are assignments to variables of X' and X'' respectively.

Proposition 1: Let $H(z', z'')$ be a formula such that $\exists W[EQ \wedge G^{rlx}] \equiv H \wedge \exists W[G^{rlx}]$ where $W = X' \cup X'' \cup Y' \cup Y''$. Then formula $G \wedge (z' \not\equiv z'')$ is equisatisfiable with $H \wedge G^{rlx} \wedge (z' \not\equiv z'')$.

Note that finding formula $H(z', z'')$ of Proposition 1 reduces to taking formula EQ out of the scope of quantifiers i.e. to solving the PQE problem. Proposition 1 implies that proving *inequivalence* of N' and N'' comes down to showing that formula G^{rlx} is satisfiable under assignment $(z' = b', z'' = b'')$ (where $b', b'' \in \{0,1\}$) such that $b' \neq b''$ and $H(b', b'') = 1$.

[1]The proofs of propositions are given in [7].

Recall that the input variables of N' and N'' are independent of each other in formula G^{rlx}. Hence the only situation where G^{rlx} is unsatisfiable under $(z' = b', z'' = b'')$ is when N' is constant $\overline{b'}$ and/or N'' is constant $\overline{b''}$. So the corollary below holds.

Corollary 1: If neither N' nor N'' are constants, they are equivalent iff $H(1,0) = H(0,1) = 0$.

Reducing EC to an instance of PQE also provides valuable information when proving *equivalence* of N' and N''. Formula G^{rlx} remains quantified in $\exists W[EQ \wedge G^{rlx}] \equiv H \wedge \exists W[G^{rlx}]$. This means that to obtain formula H, it suffices to generate only resolvents that are descendants of clauses of EQ. The clauses obtained by resolving solely clauses of G^{rlx} are just "noise" (see Subsection 2-A). This observation is the basis of our algorithm for generating proofs of equivalence by induction.

3. Boundary Formulas

In this section, we discuss boundary formulas, a key notion of EC by LoR. Subsection 3-A explains the semantics of boundary formulas. Subsection 3-B discusses the size of boundary formulas. In Subsection 3-C, we describe how boundary formulas are built.

A. Definition and some properties of boundary formulas

Let M be the subcircuit consisting of the gates of N', N'' located below a cut as shown in Fig. 5. As usual, G denotes $EQ(X', X'') \wedge F_{N'} \wedge F_{N''}$ and G^{rlx} does $F_{N'} \wedge F_{N''}$.

Definition 1: Let formula H_{cut} depend only on variables of a cut. Let $\boldsymbol{q}$ be an assignment to the variables of this cut. Formula H_{cut} is called **boundary** if[2]

a) $G \rightarrow H_{cut}$ holds and
b) for every $\boldsymbol{q}$ that can be extended to satisfy G^{rlx} but cannot be extended to satisfy G, the value of $H_{cut}(\boldsymbol{q})$ is 0.

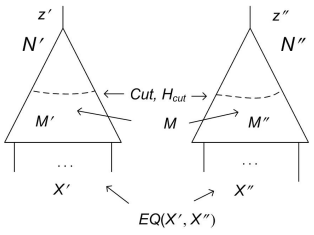

Fig. 5. Building boundary formula H_{cut}

A cut assignment $\boldsymbol{q}$ can be represented as ($\boldsymbol{q'}$,$\boldsymbol{q''}$) where $\boldsymbol{q'}$ and $\boldsymbol{q''}$ are assignments to cut variables of N' and N'' respectively. Note that Definition 1 does not specify the value of $H_{cut}(\boldsymbol{q})$ if $\boldsymbol{q}$ *cannot* be extended to satisfy G^{rlx} (and hence G). This means that H_{cut} does not have to exclude ($\boldsymbol{q'}$,$\boldsymbol{q''}$) if, say, no input $\boldsymbol{x'}$ of N' produces $\boldsymbol{q'}$. This means that H_{cut} *does not depend* on the individual complexity of N' and N''.

Formula $EQ(X', X'')$ and formula $H(z', z'')$ of Proposition 1 are actually boundary formulas with respect to cuts $X' \cup X''$ and $\{z', z''\}$ respectively. We will refer to $H(z', z'')$

[2]Since formula $(z' \not\equiv z'')$ constraining the outputs of N' and N'' is not a part of formulas G^{rlx} and G, a boundary formula of Definition 1 is not "property driven". This can be fixed by making a boundary formula specify the difference between $G^{rlx} \wedge (z' \not\equiv z'')$ and $G \wedge (z' \not\equiv z'')$ rather than between G^{rlx} and G. In this paper, we explore only boundary formulas of Definition 1 leaving property-driven ones for future research.

as an **output boundary formula**. Proposition 2 below reduces building H_{cut} to PQE.

Proposition 2: Let H_{cut} be a formula depending only on variables of a cut. Let H_{cut} satisfy $\exists W[EQ \wedge F_M] \equiv H_{cut} \wedge \exists W[F_M]$. Here W is the set of variables of F_M minus those of the cut. Then H_{cut} is a boundary formula.

Proposition 3 below extends Proposition 1 to an arbitrary boundary formula.

Proposition 3: Let H_{cut} be a boundary formula with respect to a cut. Then $G \wedge (z' \not\equiv z'')$ is equisatisfiable with $H_{cut} \wedge G^{rlx} \wedge (z' \not\equiv z'')$.

B. Size of boundary formulas

The proposition below estimates the size of a boundary formula computed for a cut if every cut variable of N' can be expressed as a function of cut variables of N''. If the number of arguments in the functions relating cut points of N' and N'' is small, these circuits can be viewed as structurally similar.

Proposition 4: Let circuits M' and M'' consist of the gates of N' and N'' located below a cut as shown in Fig. 5. Let Cut', Cut'' specify the outputs of M' and M'' respectively. Assume that for every variable v'_i of Cut' there is a set $S(v'_i) = \{v''_{i_1}, \ldots, v''_{i_p}\}$ of variables of Cut'' that have the following property. Knowing the values of variables of $S(v'_i)$ produced in N'' under input $\boldsymbol{x}$ one can determine the value of v'_i of N' under the same input $\boldsymbol{x}$. We assume here that $S(v'_i)$ has this property for every possible input $\boldsymbol{x}$. Let $Max(S(v'_i))$ be the size of the largest $S(v'_i)$ over variables of Cut'. Then there is a boundary formula H_{cut} where every clause has at most $Max(S(v'_i)) + 1$ literals.

Proposition 4 gives an example of boundary formulas whose complexity is exponential in the value of $Max(S(v'_i)) + 1$ rather than the cut size. This means that these boundary formulas depend only on *similarity* of N' and N'' and do not depend on how complex N' and N'' are.

Corollary 2: Let circuits M' and M'' of Fig. 5 be functionally equivalent. Then for every variable $v' \in Cut'$ there is a set $S(v') = \{v''\}$ where v'' is the variable of Cut'' that is functionally equivalent to v'. In this case, formula $EQ(Cut', Cut'')$ stating equivalence of corresponding output variables of M' and M'' is a boundary formula for the cut in question. Note that $v' \equiv v''$ can be represented as a CNF formula $(v' \vee \overline{v''}) \wedge (\overline{v'} \vee v'')$. So $EQ(Cut', Cut'')$ can be represented by $2{*}p$ two-literal clauses where $p = |Cut'| = |Cut''|$.

C. Computing Boundary Formulas

The key part of EC by LoR is to compute an output boundary formula $H(z', z'')$. In this subsection, we show how to build formula H *inductively* by constructing a sequence of boundary formulas $H_0, \ldots, H_k$ computed with respect to cuts $Cut_0, \ldots, Cut_k$ of N' and N'' (see Fig. 2). We assume that $Cut_0 = X' \cup X''$ and $Cut_k = \{z', z''\}$ (i.e. $H = H_k$) and $Cut_i \cap Cut_j = \emptyset$ if $i \neq j$.

Boundary formula H_0 is set to $EQ(X', X'')$ whereas formula H_i, $i > 0$ is computed from H_{i-1} as follows. Let M_i be the circuit consisting of the gates of N' and N'' located

```
EC_LoR(N', N''){
1  (N', N'') := Bufferize(N', N'');
2  Cut_0 = X' ∪ X'';
3  Cut_1, .., Cut_{k-1} := BldIntermCuts(N', N'');
4  Cut_k := {z', z''};
5  H_0 := EQ(X', X'');
- - - - - - - - - - - - - - - - - -
6  for(i := 1; i ≤ k; i++) {
7     F_{M_i} := SubForm(G^{rlx}, Cut_i);
8     W_i := Vars(F_{M_i}) \ Vars(Cut_i);
9     H_i := PQE(∃W_i[H_{i-1} ∧ F_{M_i}]); }
- - - - - - - - - - - - - - - - - -
10 if (H_k(0, 1) = 1)
11    if (Sat(G^{rlx} ∧ ¬z' ∧ z'')) return(No);
12 if (H_k(1, 0) = 1)
13    if (Sat(G^{rlx} ∧ z' ∧ ¬z'')) return(No);
14 return(Yes); }
```

Fig. 6. EC by LoR

below i-th cut. Let F_{M_i} be the subformula of G^{rlx} specifying M_i. Let W_i consist of all the variables of F_{M_i} minus those of Cut_i. Formula H_i is built to satisfy $\exists W_i[H_{i-1} \wedge F_{M_i}] \equiv H_i \wedge \exists W_i[F_{M_i}]$ and so make the previous boundary formula H_{i-1} *redundant* in $H_i \wedge \exists W_i[H_{i-1} \wedge F_{M_i}]$. The fact that $H_1, \ldots, H_k$ are indeed boundary formulas follows from Proposition 5.

Proposition 5: Let W_i where $i > 0$ be the set of variables of F_{M_i} minus those of Cut_i. Let H_{i-1}, $i > 1$ satisfy $\exists W_{i-1}[H_0 \wedge F_{M_{i-1}}] \equiv H_{i-1} \wedge \exists W_{i-1}[F_{M_{i-1}}]$. (So H_{i-1} is a boundary formula due to Proposition 2.) Let $\exists W_i[H_{i-1} \wedge F_{M_i}] \equiv H_i \wedge \exists W_i[F_{M_i}]$ hold. Then $\exists W_i[H_0 \wedge F_{M_i}] \equiv H_i \wedge \exists W_i[F_{M_i}]$ holds and so, H_i is a boundary formula too.

4. Algorithm of EC by LoR

In this section, we introduce an algorithm called EC_LoR that checks for equivalence two single-output circuits N' and N''. The pseudo-code of EC_LoR is given in Fig. 6. EC_LoR builds a sequence of boundary formulas $H_0, \ldots, H_k$ as described in Subsection 3-C. Here H_0 equals $EQ(X', X'')$ and $H_k(z', z'')$ is an output boundary formula. Then, according to Proposition 1, EC_LoR checks the satisfiability of formula $H_k \wedge G^{rlx} \wedge (z' \not\equiv z'')$ where $G^{rlx} = F_{N'} \wedge F_{N''}$.

EC_LoR consists of three parts separated by the dotted lines in Figure 6. EC_LoR starts the first part (lines 1-5) by calling procedure *Bufferize*. This procedure eliminates non-local connections of N' and N'' i.e. those that span more than two consecutive topological levels. (The *topological level* of a gate g of a circuit K is the longest path from an input of K to g measured in the number of gates on this path.) The presence of non-local connections makes it hard to find cuts that do not overlap. To avoid this problem, procedure *Bufferize* replaces every non-local connection spanning d topological levels ($d > 2$) with a chain of $d - 2$ buffers. (A more detailed discussion of this topic is given in [7].) Then EC_LoR sets the initial cut to $X' \cup X''$, computes the intermediate cuts (line 3), sets the final cut to $\{z', z''\}$ and formula H_0 to $EQ(X', X'')$.

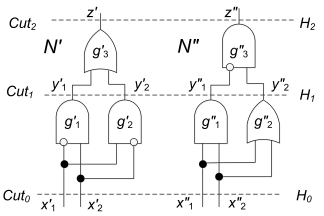

Fig. 7. An example of EC by LoR

Boundary formulas H_i, $1 \leq i \leq k$ are computed in the second part (lines 6-9) that consists of a *for* loop. In the third part (lines 10-14), *EC_LoR* uses the output boundary formula $H_k(z', z'')$ computed in the second part to decide whether N', N'' are equivalent. If $H_k(b', b'') = 1$ where $b' \neq b''$ and G^{rlx} is satisfiable under $z' = b', z'' = b''$, then N', N'' are inequivalent. Otherwise, they are equivalent (line 14).

Boundary formulas are computed in the *for* loop as follows. Formula H_i, $i > 0$ is obtained by taking H_{i-1} out of the scope of quantifiers in $\exists W_i[H_{i-1} \wedge F_{M_i}]$ (line 9) i.e. $\exists W_i[H_{i-1} \wedge F_{M_i}] \equiv H_i \wedge \exists W_i[F_{M_i}]$. Here F_{M_i} is the formula specifying the gates of N' and N'' below i-th cut and W_i consists of all the variables of F_{M_i} but cut variables.

Example 1: Let us explain the operation of *EC_LoR* by the example of Fig. 7 showing two different implementations of function $XOR(x_1, x_2)$. *EC_LoR* starts by executing the first part specified by lines 1-5 of Fig. 6. Since circuits N' and N'' do not have no-local connections, no buffers are inserted. *EC_LoR* sets the initial cut Cut_0 to $\{X', X''\}$ where $X' = \{x'_1, x'_2\}$, $X'' = \{x''_1, x''_2\}$, generates an intermediate cut $Cut_1 = \{y'_1, y'_2, y''_1, y''_2\}$ and the final cut $Cut_2 = \{z', z''\}$. *EC_LoR* concludes the first part by setting H_0 to $EQ(X', X'')$ i.e. to $(x'_1 \equiv x''_1) \wedge (x'_2 \equiv x''_2)$.

In the second part (lines 6-9 of Fig. 6), *EC_LoR* computes boundary formulas for Cut_1 and Cut_2. A boundary formula for Cut_1 is obtained by taking H_0 out of the scope of quantifiers in $\exists W_1[H_0 \wedge F_{M_1}]$ i.e. by finding formula H_1 such that $\exists W_1[H_0 \wedge F_{M_1}] \equiv H_1 \wedge \exists W_1[F_{M_1}]$. Here F_{M_1} specifies the gates located below cut Cut_1 and so $F_{M_1} = F_{g'_1} \wedge F_{g'_2} \wedge F_{g''_1} \wedge F_{g''_2}$ where F_g specifies gate g. For instance, $F_{g'_1} = (x'_1 \vee \overline{x'}_2 \vee y''_1) \wedge (\overline{x'}_1 \vee \overline{y'}_1) \wedge (x'_2 \vee \overline{y'}_1)$. Set W_1 consists of all variables of F_{M_1} minus the variables of Cut_1 i.e. $W_1 = X' \cup X''$. Formula H_1 obtained by a PQE-solver implementing the algorithm of [10] consists of five clauses: $C_1 = y'_1 \vee y'_2 \vee y''_1 \vee \overline{y''}_2$, $C_2 = \overline{y'}_1 \vee \overline{y''}_1$, $C_3 = \overline{y'}_1 \vee y''_2$, $C_4 = \overline{y'}_2 \vee y''_2$, $C_5 = \overline{y'}_2 \vee \overline{y''}_1$.

A boundary formula for cut Cut_2 is obtained by taking H_1 out of the scope of quantifiers in $\exists W_2[H_1 \wedge F_{M_2}]$. Here F_{M_2} specifies the gates located below cut Cut_2, so $F_{M_2} = F_{M_1} \wedge F_{g'_3} \wedge F_{g''_3}$. Set W_2 consists of the variables of F_{M_2} minus those of Cut_2, so $W_2 = W_1 \cup \{y'_1, y'_2, y''_1, y''_2\}$. Formula H_2 obtained by the PQE-solver mentioned above is equal to $(z' \vee \overline{z''}) \wedge (\overline{z'} \vee z'')$. This means that $H(0,1) = H(1,0) = 0$. So after executing its last part (lines 10-14 of Fig. 6), *EC_LoR* reports that N' and N'' are equivalent.

Let us take a closer look at formula H_1. On one hand, as a boundary formula, H_1 excludes every assignment to Cut_1 that can be produced by applying an input ($\boldsymbol{x'}$,$\boldsymbol{x''}$) where $\boldsymbol{x'} \neq \boldsymbol{x''}$ but cannot be produced if input assignments are constrained by $EQ(X', X'')$. For instance, input $(x'_1 = 0, x'_2 = 0, x''_1 = 0, x''_2 = 1)$ produces cut assignment $y'_1 = 0, y'_2 = 0, y''_1 = 0, y''_2 = 1)$ that cannot be produced by an input assignment ($\boldsymbol{x'}$,$\boldsymbol{x''}$) where $\boldsymbol{x'}$=$\boldsymbol{x''}$. This cut assignment falsifies clause $C_1 = y'_1 \vee y'_2 \vee y''_1 \vee \overline{y''}_2$ of H_1. On the other hand, H_1 is simpler that formula Img_1 that excludes *every* assignment to Cut_1 that cannot be produced by an input ($\boldsymbol{x'}$,$\boldsymbol{x''}$) where $\boldsymbol{x'}$=$\boldsymbol{x''}$. Formula Img_1 is logically equivalent to $\exists W_1[H_0 \wedge F_{M_1}]$ i.e. it is obtained from the latter by *complete* quantifier elimination. By applying our program of [9], we obtain formula Img_1 equal to $H_1 \wedge C_6 \wedge C_7$ where $C_6 = \overline{y'_1} \vee \overline{y'_2}$, $C_7 = \overline{y''}_1 \wedge y''_2$. Note that C_6, C_7 do not relate variables of N' and N''. Instead, they exclude some cut assignments that cannot be produced in N' or N''. For instance, clause C_6 excludes cut assignment $(y'_1 = 1, y'_2 = 1)$ that cannot be produced in N'.

5. Computing Boundary Formulas By Current PQE Solvers

To obtain boundary formula H_i, one needs to take H_{i-1} out of the scope of quantifiers in formula $\exists W_i[H_{i-1} \wedge F_{M_i}]$. The size of the latter grows with i due to formula F_{M_i}. So a PQE solver that computes H_i must have good scalability. On the other hand, the algorithm of [10] does not scale well yet. The main problem here is that learned information is not re-used in contrast to SAT-solvers effectively re-using learned clauses. Fixing this problem requires some time because bookkeeping of a PQE algorithm is more complex than that of a SAT-solver. (In more detail, this topic is discussed in [7].) In this section, we describe two methods of adapting EC by LoR to a PQE-solver that is not efficient enough to compute every boundary formula *precisely*. Both methods are illustrated experimentally in Section 6.

One way to reduce the complexity of computing H_i is to use only a subset of F_{M_i}. For instance, one can discard the clauses of F_{M_i} specifying the gates located between cuts Cut_0 and Cut_p, $0 < p < i$. In this case, boundary formula H_i is computed *approximately*. A downside of this is that condition b) of Definition 1 does not hold anymore and so EC by LoR becomes *incomplete*. Namely, if $H_k(b', b'') = 1$ where $b' \neq b''$ and H_k is an output boundary formula, the fact that G^{rlx} is satisfiable under $z' = b', z'' = b''$ does not mean that N' and N'' are inequivalent. Nevertheless, even EC by LoR with approximate computation of boundary formulas can be a powerful tool for proving N' and N'' *equivalent* for the following reason. If $H_k(1,0) = H_k(0,1) = 0$, circuits N' and N'' are proved equivalent even if intermediate formulas H_i are built approximately. Importantly, computing boundary formulas *inductively* still provides a powerful way to *structure* a proof of equivalence. Formula H_i (i.e. a "sufficient" set of clauses relating variables of i-th cut) is still obtained by taking H_{i-1} out of the scope of quantifiers in $\exists W_i[H_{i-1} \wedge F_{M_i}]$. Only now formula F_{M_i} is simplified by discarding some clauses.

Another way to adapt EC by LoR to a PQE solver that is not efficient enough to compute *every* boundary formula precisely is as follows. Suppose that the power of a PQE solver

TABLE I
Computing cut image and boundary formulas. Time limit = 1 hour

#bits	#quan. vars	#free vars	cut image formula (QE)		boundary formula (PQE)	
			result size	(s.)	result size	(s.)
8	32	84	3,142	4.0	**242**	**0.1**
9	36	104	4,937	13	**273**	**0.2**
10	40	126	7,243	51	**407**	**0.3**
11	44	150	9,272	147	**532**	**0.5**
12	48	176	14,731	497	**576**	**0.6**
13	52	206	19,261	1,299	**674**	**0.9**
14	56	234	*	*	**971**	**1.5**
15	60	266	*	*	**1,218**	**2.0**
16	64	300	*	*	**1,411**	**3.0**

is sufficient to build *one* intermediate boundary formula H_i precisely. From Proposition 3 it follows that formula α equal to $G \wedge (z' \not\equiv z'')$ is equisatisfiable with formula β equal to $H_{cut} \wedge G^{rlx} \wedge (z' \not\equiv z'')$. So, to show that N' and N'' are inequivalent it is sufficient to find an assignment satisfying β. As we argued in Subsection 1-D, finding such an assignment for β is easier than for α.

6. Experiments

In the experiments, we employed the PQE algorithm published in [10] in 2014. We will refer to this algorithm as **PQE-14**. As we mentioned in Section 5, PQE-14 does not scale well yet. So building a full-fledged equivalence checker based on EC_LoR would mean simultaneously designing a new EC algorithm and a new PQE solver. The latter is beyond the scope of our paper. On the other hand, PQE-14 is efficient enough to make a few important points experimentally. In the experiments described in this section, we used a new implementation of PQE-14 [6].

The experiment of Subsection 6-A compares computing a cut image formula and a boundary formula. Recall that a cut image formula is satisfied by a cut assignment iff the latter can be produced in N' and N'' by some input satisfying $EQ(X', X'')$. This experiment also contrasts complete quantifier elimination (employed to compute a cut image formula) with PQE. In Subsection 6-B, we apply EC_LoR to a non-trivial instance of equivalence checking that is hard for *ABC*, a high-quality synthesis and verification tool [20]. In Subsection 6-C, we show that computing boundary formulas is beneficial for proving inequivalence.

In the experiments, circuits N' and N'' to be checked for equivalence were derived from a circuit computing a median output bit of an s-bit multiplier. We will refer to this circuit as Mlp_s. Our motivation here is as follows. In many cases, the equivalence of circuits with simple topology and low fanout values can be efficiently checked by a general-purpose SAT-solver. This is not true for circuits involving multipliers. In all experiments, circuits N' and N'' were bufferized to get rid of long connections (see Section 4).

A. Image computation versus building boundary formulas

In the experiment of this subsection, we compared computation of a boundary formula H_{cut} and a cut image formula Img_{cut}. We used two identical copies of circuit Mlp_s as circuits N' and N''. As a cut of N', N'' we picked the set of variables of the first topological level (every variable of this level specifies the output of a gate fed by input variables of N' or N''). Formula Img_{cut} is logically equivalent to $\exists W[EQ(X', X'') \wedge F_M]$ where $W = X' \cup X''$ and formula F_M specifies the gates of the first topological level of N' and N''. So, computing Img_{cut} comes down to solving the quantifier elimination problem. Computing a boundary formula reduces to finding H_{cut} such that $\exists W[EQ \wedge F_M] \equiv H_{cut} \wedge \exists W[F_M]$ i.e. solving the PQE problem.

The results of the experiment are given in Table I. Abbreviation QE stands for Quantifier Elimination. The value of s in Mlp_s is shown in the first column. The next two columns give the number of quantified and free variables in $\exists W[EQ \wedge F_M]$. To compute formula Img_{cut} we used our quantifier elimination program presented in [9]. Formula H_{cut} was generated by PQE-14. When computing image formula Img_{cut} and boundary formula H_{cut} we recorded the size of the result (as the number of clauses) and the run time in seconds. As Table I shows, formulas H_{cut} are much smaller than Img_{cut} and take much less time to compute.

B. Proving equivalence by LoR

In this subsection, we ran an implementation of EC_LoR introduced in Section 4 on circuits N' and N'' shown in Fig. 8. (The idea of this EC example was suggested by Vigyan Singhal [19].) These circuits were derived from Mlp_s by adding one extra input h. Either circuit produces the same output as Mlp_s when $h = 1$ and output 0 if $h = 0$. So N' and N'' are logically equivalent. Note that the value of every internal variable of N' depends on h whereas this is not the case for N''. So N' and N'' have no functionally equivalent internal variables. On the other hand, N' and N'' satisfy the notion of structural similarity introduced in Subsection 3-B to prove Proposition 4. Namely, the value of every internal variable v' of N' is specified by that of h'' and some variable v'' of N''. (So, in this case, for every internal variable v' of N' there is a set $S(v')$ defined in Proposition 4 consisting of only two variables of N''.) In particular, if v' is an internal variable of Mlp'_s, then v'' is the corresponding variable of Mlp''_s. Indeed, if $h'' = 1$, then v' takes the same value as v''. If $h'' = 0$, then v' is a constant (in the implementation of Mlp_s we used in the experiments). The objective of the experiment below was to show that EC_LoR can check for equivalence structurally similar circuits that have no functionally equivalent internal points.

Cuts $Cut_0, \ldots, Cut_k$ used by EC_LoR were generated according to topological levels. That is every variable of Cut_i specified the output of a gate of i-th topological level. Since N' and N'' were bufferized, $Cut_i \cap Cut_j = \emptyset$ if $i \neq j$. The version of EC_LoR we used in the experiment was slightly different from the one described in Fig. 6. We will refer to this version as EC_LoR^*. (A detailed description of EC_LoR^* is given in [7]). The main change was that boundary formulas were computed in EC_LoR^* *approximately*. That is formula

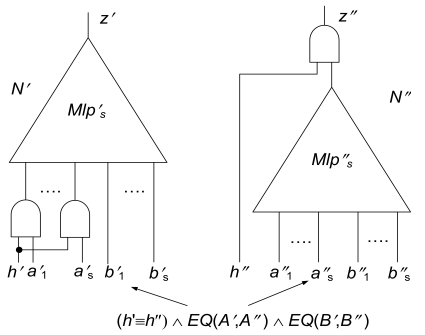

Fig. 8. Equivalence checking of N' and N'' derived from Mlp_s

TABLE II
EC of N' and N'' derived from Mlp_s. Time limit = 6 hours

#bits	#vars	#clauses	#cuts	EC_LoR^* (s.)	*ABC* (s.)
10	2,844	6,907	37	**4.5**	10
11	3,708	8,932	41	**7.1**	38
12	4,726	11,297	45	**11**	142
13	5,910	14,026	49	**16**	757
14	7,272	17,143	53	**25**	3,667
15	8,824	20,672	57	**40**	11,237
16	10,578	24,637	61	**70**	$>$ 21,600

H_i was obtained by taking H_{i-1} out of the scope of quantifiers in formula $\exists W_i[H_{i-1} \wedge F_{M_i}]$ where only a subset of clauses of F_{M_i} was used. Nevertheless, EC_LoR^* was able to compute an output boundary formula $H_k(z', z'')$ that implied $(z' \equiv z'')$ thus proving that N' and N'' were equivalent.

One more difference between EC_LoR and EC_LoR^* was that the latter built formula H_i by solving a sequence of small PQE problems rather than one large PQE problem (line 9 of Fig. 6). Each PQE problem of this sequence was meant to find clauses relating the output of a gate g' of N' of Cut_i to its "siblings" of N'' that are in Cut_i. A gate g'' of N'' was considered a sibling of g' if inputs of g' and g'' were related by a clause of H_{i-1}. After solving the sequence of small PQE problems above, EC_LoR^* checked a *cut termination condition*. That is EC_LoR^* verified that $\exists W_i[H_{i-1} \wedge F_{M_i}] \equiv H_i \wedge \exists W_i[F_{M_i}]$ and so the set of clauses accumulated in H_i was indeed a boundary formula for i-th cut.

In Table II, we compare EC_LoR^* with *ABC* [20]. The first column gives the value of s of Mlp_s used in N' and N''. The next two columns show the size of formulas $EQ(X', X'') \wedge F_{N'} \wedge F_{N''} \wedge (z' \not\equiv z'')$ specifying equivalence checking of N' and N'' to which EC_LoR^* was applied. (N' and N'' were fed into *ABC* as circuits in the BLIF format.) Here $X = \{h, a_1, \ldots, a_s, b_1, \ldots, b_s\}$ denotes the set of input variables of circuits N' and N''. The fourth column shows the number of topological levels in circuits N' and N'' and so the number of cuts used by EC_LoR^*. The last two columns give the run time of EC_LoR^* and *ABC*.

The results of Table II show that equivalence checking of N' and N'' derived from Mlp_s was hard for *ABC*. On the other hand, EC_LoR^* managed to solve all instances in a reasonable time. Most of the run time of EC_LoR^* was taken by PQE-14 when checking cut termination conditions mentioned above. So, PQE-14 was also the reason why the run time of EC_LoR^* grew quickly with the size of Mlp_s. The performance of EC_LoR^* with a more efficient PQE-solver should have a weaker dependency on the value of s.

C. Using boundary formulas for proving inequivalence

In the experiment of this subsection, we checked for equivalence circuits N' and N'' that were correct and buggy versions of Mlp_{16} respectively. Since EC_LoR^* described in the previous subsection computes boundary formulas approximately, one cannot directly apply it to prove inequivalence of N' and N''. In this subsection, we show that the precise computation of even *one* boundary formula corresponding to an intermediate cut can be quite useful for proving inequivalence. Let α and β denote formulas $EQ(X', X'') \wedge F_{N'} \wedge F_{N''} \wedge (z' \equiv z'')$ and $H_i \wedge F_{N'} \wedge F_{N''} \wedge (z' \equiv z'')$ respectively. Here H_i is a boundary formula precisely computed for the cut of N' and N'' consisting of the gates with topological level equal to i. According to Proposition 3, α and β are equisatisfiable. Proving N' and N'' inequivalent comes down to showing that β is satisfiable. Intuitively, checking the satisfiability of β the easier, the larger the value of i and so the closer the cut to the outputs of N' and N''. In the experiment below, we show that computing boundary formula H_i makes proving inequivalence of N' and N'' easier even for a cut with a small value of i.

TABLE III
Sat-solving of formulas α and β by Minisat. Time limit = 600 s.

formula type	#solved	total time (s.)	median time (s.)
α	95	$>$ 3,490	4.2
β	**100**	**1,030**	**1.0**

Bugs were introduced into circuit N'' *above* the cut (so N' and N'' were identical *below* the cut). Let M_i' and M_i'' denote the subcircuits of N' and N'' consisting of the gates located below the cut (like circuits M' and M'' in Fig. 5). Since M_i' and M_i'' are identical they are also functionally equivalent. Then Corollary 2 entails that formula H_i equal to $EQ(Cut_i', Cut_i'')$ is boundary. Here Cut_i' and Cut_i'' specify the output variables of M_i' and M_i'' respectively. Derivation of $EQ(Cut_i', Cut_i'')$ for identical circuits M_i' and M_i'' is trivial. However, *proving* that H_i equal to $EQ(Cut_i', Cut_i'')$ is indeed a boundary formula is *non-trivial* even for identical circuits. (According to Proposition 2, this requires showing that $\exists W_i[EQ(X', X'') \wedge F_{M_i}] \equiv H_i \wedge \exists W_i[F_{M_i}]$ where F_{M_i} specifies the gates of M_i' and M_i'' and W_i consists of all the variables of F_{M_i} but the cut variables.) In the experiment, we used the cut with $i = 3$ i.e. the gates located below the cut had topological level less or equal to 3. Proving that $EQ(Cut_i', Cut_i'')$ is a boundary formula takes a fraction of a second for $i = 3$ but requires much more time for $i = 4$.

We generated 100 buggy versions of Mlp_{16}. Table III contains results of checking the satisfiability of 100 formulas α and β by Minisat 2.0 [5], [21]. Similar results were observed for the other SAT-solvers we tried. The first column of Table III shows the type of formulas (α or β). The second column gives

the number of formulas solved in the time limit of 600 s. The third column shows the total run time on all formulas. We charged 600 s. to every formula α that was not solved within the time limit. The run times of solving formulas β include the time required to build H_3. The fourth column gives the median time. The results of this experiment show that proving satisfiability of β is noticeably easier than that of α. As we mentioned above, using formula β for proving inequivalence of N' and N'' should be much more beneficial if formula H_i is computed for a cut with a greater value of i. However, this will require a more powerful PQE solver than PQE-14.

7. Some Background

The EC methods can be roughly classified into two groups. Methods of the first group do not assume that circuits N' and N'' to be checked for equivalence are structurally similar. Checking if N' and N'' have identical BDDs [4] is an example of a method of this group. Another method of the first group is to reduce EC to SAT and run a general-purpose SAT-solver [15], [18], [5], [2]. A major flaw of these methods is that they do not scale well with the circuit size.

Methods of the second group try to exploit the structural similarity of N', N''. This can be done, for instance, by making transformations that produce isomorphic subcircuits in N' and N'' [1] or make simplifications of N' and N'' that do not affect their range [14]. The most common approach used by the methods of this group is to generate an inductive proof by computing simple relations between internal points of N', N''. Usually, these relations are equivalences [11], [12], [17]. However, in some approaches the derived relations are implications [13] or equivalences modulo observability [3]. The main flaw of the methods of the second group is that they are very "fragile". That is they work only if the equivalence of N' and N'' can be proved by derivation of relations of a very small class.

The machinery of boundary formulas introduced in this paper can be related to interpolation [16]. As far as propositional logic is concerned, interpolation and an interpolant are a special case of logic relaxation and a boundary formula respectively [7].

8. Conclusions

We introduced a new framework for Equivalence Checking (EC) based on Logic Relaxation (LoR). The appeal of applying LoR to EC is twofold. First, EC by LoR provides a powerful method for generating proofs of equivalence by induction. Second, LoR gives a framework for proving inequivalence without generating a counterexample. The idea of LoR is quite general and can be applied beyond EC. LoR is enabled by a technique called partial quantifier elimination and the performance of the former strongly depends on that of the latter. So building efficient algorithms of partial quantifier elimination is of great importance.

Acknowledgment

I would like to thank the anonymous reviewers of this paper for their helpful feedback. I am thankful to Harsh Raju Chamarthi for reading the first version of the manuscript. I am especially grateful to Mitesh Jain who has read several versions of this paper and made detailed and valuable comments. This research was supported in part by NSF grants CCF-1117184 and CCF-1319580.

References

[1] H.R. Andersen and H. Hulgaard. Boolean expression diagrams. *Inf. Comput.*, 179(2):194–212, 2002.

[2] A. Biere. Picosat essentials. *JSAT*, 4(2-4):75–97, 2008.

[3] D. Brand. Verification of large synthesized designs. In *ICCAD-93*, pages 534–537, 1993.

[4] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[5] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT*, pages 502–518, Santa Margherita Ligure, Italy, 2003.

[6] E. Goldberg. On efficient methods for partial quantifier elimination. To be published.

[7] E. Goldberg. Equivalence checking by logic relaxation. Technical Report arXiv:1511.01368 [cs.LO], 2015.

[8] E. Goldberg and P. Manolios. Quantifier elimination by dependency sequents. In *FMCAD-12*, pages 34–44, 2012.

[9] E. Goldberg and P. Manolios. Quantifier elimination via clause redundancy. In *FMCAD-13*, pages 85–92, 2013.

[10] E. Goldberg and P. Manolios. Partial quantifier elimination. In *Proc. of HVC-14*, pages 148–164. Springer-Verlag, 2014.

[11] A. Kuehlmann and F. Krohm. Equivalence Checking Using Cuts And Heaps. *DAC*, pages 263–268, 1997.

[12] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Trans. CAD*, 21:1377–1394, 2002.

[13] W. Kunz. Hannibal: An efficient tool for logic verification based on recursive learning. In *ICCAD-93*, pages 538–543, 1993.

[14] H. Kwak, I. MoonJames, H. Kukula, and T. Shiple. Combinational equivalence checking through function transformation. In *ICCAD-02*, pages 526–533, 2002.

[15] J. Marques-Silva and K. Sakallah. Grasp – a new search algorithm for satisfiability. In *ICCAD-96*, pages 220–227, 1996.

[16] K. L. Mcmillan. Interpolation and sat-based model checking. In *CAV-03*, pages 1–13. Springer, 2003.

[17] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een. Improvements to combinational equivalence checking. In *ICCAD-06*, pages 836–843, 2006.

[18] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient sat solver. In *DAC-01*, pages 530–535, New York, NY, USA, 2001.

[19] V. Singhal. Private communication.

[20] ABC. http://www.eecs.berkeley.edu/ alanmi/abc/.

[21] Minisat2.0. http://minisat.se/MiniSat.html.