arXiv:1511.01368v3 [cs.LO] 11 Jul 2016

Equivalence Checking By Logic Relaxation

Eugene Goldberg

eu.goldberg@gmail.com

Abstract. We introduce a new framework for Equivalence Checking (EC) of Boolean circuits based on a general technique called **Lo**gic **R**elaxation (LoR). The essence of LoR is to relax the formula to be solved and compute a superset S of the set of new behaviors. Namely, S contains all new satisfying assignments that appeared due to relaxation and does not contain assignments satisfying the original formula. Set S is generated by a procedure called partial quantifier elimination. If all possible bad behaviors are in S, the original formula cannot have them and so the property described by this formula holds. The appeal of EC by LoR is twofold. First, it facilitates generation of powerful *inductive proofs*. Second, proving inequivalence comes down to checking the presence of some bad behaviors in the *relaxed formula* i.e. in a simpler version of the original formula. We give experimental evidence that supports our approach.

1 Introduction

1.1 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.

1.2 Structural similarity of circuits

In this paper, we target EC of structurally similar circuits N' and N''. Providing a comprehensive definition of structural similarity is a tall order. Instead, below we give an example of circuits that can be viewed as structurally similar. Let v' be a variable of circuit N'. Let $S(v') = \{v''_{i_1}, \dots, v''_{i_k}\}$ be a set of variables of N'' that have the following property. The knowledge of values assigned to the variables of $S(v')$ in N'' under input $\boldsymbol{x}$ is sufficient to find out the value of v' in N' under the same input $\boldsymbol{x}$. (This means that v' is a function of variables of $S(v')$ modulo assignments to $S(v')$ that cannot be produced in N'' under any input.) Suppose that for every variable v' of N' there exists a *small* set $S(v')$ with the property above. Then one can consider circuits N' and N'' as **structurally**

similar. The smaller sets $S(v')$, the closer N' and N'' structurally. In particular, if N' and N'' are identical, for every variable v' there is a set $S(v')$ consisting only of one variable of N''. An example of structurally similar circuits where $S(v')$ consists of two variables is given in Subsection 7.2.

1.3 EC by logic relaxation

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 let N' and N'' have separate sets of input variables but assume that N' and N'' must be equivalent only for the input assignments satisfying $EQ(X',X'')$.)

EC by **Logic Relaxation (LoR)** presented in this paper is based on the following idea. Let $Out(N',N'')$ denote the set of outputs of N' and N'' when their input variables are constrained by $EQ(X',X'')$. For equivalent circuits N' and N'' that are not constants, $Out(N',N'')$ is equal to $\{(z'=0,z''=0),(z'=1,z''=1)\}$. Let $Out^{rlx}(N',N'')$ denote the set of outputs of N' and N'' when their inputs are not constrained by $EQ(X',X'')$. (Here *rlx* stands for "relaxed"). $Out^{rlx}(N',N'')$ is a superset of $Out(N',N'')$ that may contain an output $(z'=0,z''=1)$ and/or output $(z'=1,z''=0)$ even when N' and N'' are equivalent. Let D_{out} denote $Out^{rlx}(N',N'') \setminus Out(N',N'')$. That is D_{out} contains the outputs that can be produced *only* when inputs of N' and N'' are independent of each other.

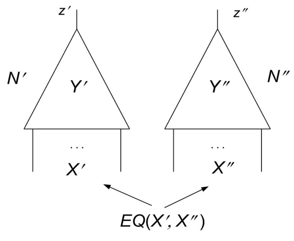

Fig. 1. Equivalence checking of N' and N''

Computing set D_{out} either solves the equivalence checking of N' and N'' or dramatically simplifies it. (This is important because, arguably, set D_{out} is much easier to find than $Out(N',N'')$.) Indeed, assume that D_{out} contains both $(z'=1,z''=0)$ and $(z'=0,z''=1)$. Then N' and N'' are *equivalent* because assignments where values of z' and z'' are different are present in $Out^{rlx}(N',N'')$ but not in $Out(N',N'')$. Now, assume that D_{out} does not contain, say, assignment $(z'=1,z''=0)$. This can only occur in the following two cases. First, N' cannot produce output 1 (i.e. N' is a constant 0) and/or N'' cannot produce 0 (i.e. N'' is a constant 1). Second, *both* $Out^{rlx}(N',N'')$ *and*

$Out(N', N'')$ contain assignment $(z'=1, z''=0)$ and hence N' and N'' are *inequivalent*. Separating these two cases comes down to checking if N' and N'' can evaluate to 1 and 0 respectively. If the latter is true, N' and N'' are inequivalent.

Set D_{out} is built in EC by LoR by computing a sequence of so-called **boundary formulas** $H_0, \dots, H_k$. Formula H_i depends only on the variables of a cut of N', N'' (see Figure 2) and *excludes* the assignments of $D_{Cut\,i} = Cut_i^{rlx}(N', N'') \setminus Cut_i(N', N'')$. (That is every assignment of $D_{Cut\,i}$ *falsifies* H_i). Here $Cut_i^{rlx}(N', N'')$ and $Cut_i(N', N'')$ are the sets of all cut assignments produced when inputs of N' and N'' are unconstrained or constrained by $EQ(X', X'')$ respectively. Formula H_0 is specified in terms of cut $X' \cup X''$ and is equal to $EQ(X', X'')$. Formula H_k is computed in terms of cut $\{z', z''\}$ and so specifies the required set D_{out} (as a set of assignments falsifying H_k). Boundary formulas are computed by a technique called *partial quantifier elimination* (PQE) introduced in [9]. In PQE, only a part of the formula is taken out of the scope of quantifiers. So PQE can be dramatically more efficient than complete quantifier elimination.

1.4 The appeal of EC by LoR

The appeal of EC by LoR is twofold. First, EC by LoR facilitates generation of very robust proofs by induction via construction of boundary formulas. By contrast, the current approaches (see e.g. [10,11,16]) employ fragile induction proofs e.g. those that require existence of functionally equivalent internal points. The size of boundary formulas depends on the similarity of N' and N'' rather than their individual complexity. This suggests that proofs of equivalence in EC by LoR can be generated efficiently.

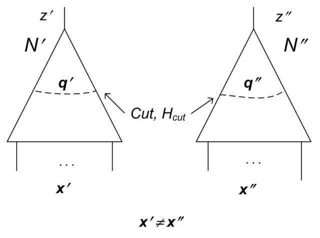

Fig. 2. Building boundary formula H_{cut}

Second, 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. We will assume that all formulas mentioned in this paper are Boolean formulas in Conjunctive Normal Form (CNF) unless otherwise stated.)

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. 2.) 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 independent of each other 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. 2) 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.

1.5 Contributions and structure of the paper

Our contributions are as follows. First, we present a new method of EC based on LoR meant for a very general class of structurally similar circuits. 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 relate EC by LoR to existing methods based on finding equivalent internal points. In particular, we show that a set of clauses relating points of an equivalence cut of N' and N'' is a boundary formula. So boundary formulas can be viewed as a machinery for generalization of the notion of an equivalence cut. Fourth, we give experimental evidence in support of EC by LoR. In particular, we employ an existing PQE algorithm whose performance can be drastically improved for solving non-trivial EC problems. Fifth, we show that interpolation is a special case of LoR (and interpolants are a special case of boundary formulas.) In particular, we demonstrate that by using LoR one can interpolate a "broken" implication. This extension of interpolation can be used for generation of short versions of counterexamples.

The structure of this paper is as follows. Section 2 discusses the challenge of proving EC by induction. In Section 3, we show the correctness of EC by LoR and relate the latter to partial quantifier elimination. Boundary formulas are discussed in Section 4. Section 5 presents an algorithm of EC by LoR. Section 6 describes how one can apply EC by LoR if the power of a PQE solver is not sufficient to compute boundary formulas precisely. Section 7 provides experimental evidence in favor of our approach. In Section 8, some background is given. We relate interpolation and LoR in Section 9 and make conclusions in Section 10. The appendix of the paper contains six sections with additional information.

2 Proving Equivalence By Induction

Intuitively, for structurally similar circuits N' and N'', there should exist a short proof of equivalence shown in Fig. 3. In this proof, for every set Cut_i forming a cut, only a small set H_i of short clauses relating variables of Cut_i is generated. (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 $\{C_1,\dots,C_p\}$ interchangeably). The relations of i-th cut specified by H_i are derived using formulas H_j built earlier i.e. $j < i$. This goes on until clauses specifying $z' \equiv z''$ are derived. We will refer to the proof shown in Fig. 3 as a **proof by induction** (slightly abusing the term “induction”). A good scalability of the current EC tools is based on their ability to derive proofs by induction. However, they can find such proofs only when cut variables have a very tight relation (most commonly, an equivalence relation). This means that these tools can handle only a very narrow subclass of structurally similar circuits.

Proving EC by induction is a challenging task because one has to address the following **cut termination problem**. When does one stop generating a set of clauses H_i in terms of variables of Cut_i and switch to building formula H_{i+1} relating variables of Cut_{i+1}? Let M_i denote the subcircuit consisting of the gates of N' and N'' located below i-th cut (like subcircuit M of Fig. 4). A straightforward way to build an inductive proof is to make formula H_i specify the *range* of M_i i.e. the set of all output assignments that can be produced by M_i. (We will also refer to the range of circuit M_i as a **cut image** because it specifies all assignments that can appear on i-th cut.) Then formula H_{i+1} can be derived from formula H_i and the clauses specifying the gates located between Cut_i and Cut_{i+1}. A flaw of this approach is that a formula specifying the image of i-th cut can get prohibitively large.

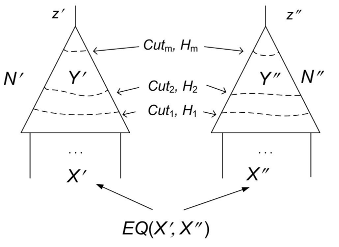

Fig. 3. An inductive proof of equivalence

A solution offered in EC by LoR is to use the boundary formulas introduced in Subsection 1.3 as formulas H_i. This solution has at least three nice qualities. First, boundary formulas have simple semantics. (H_i excludes the assignments of i-th cut that can be produced when inputs of N' and N'' are independent of each other but cannot be produced when inputs are constrained by $EQ(X', X'')$.) Second, the size of a boundary formula depends on the structural similarity of circuits N' and N'' rather than their individual complexity. In other words, a boundary formula computed for a cut is drastically simpler than a formula specifying the image of this cut in N' and N''. Third, formula H_i can be inductively derived from H_{i-1}, which gives an elegant solution to the cut termination problem. The construction of formula H_i ends (and that of H_{i+1} begins) when adding H_i to some quantified formula containing H_{i-1} makes the latter *redundant.*

3 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). Subsection 3.1 introduces PQE. In Subsection 3.2, we discuss proving equivalence/inequivalence in EC by LoR. Besides, we relate EC by LoR to PQE.

3.1 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 $\boldsymbol{A}$ 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 with respect to a quantified formula is *much more powerful* than that with respect to a quantifier-free one. For instance, if formula $F(V)$ is satisfiable, every clause of F is redundant in formula $\exists V[F]$. 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. Another way to contrast complete quantifier elimination with PQE is as follows. The former deals with a single formula and so, in a sense, has to cope with its *absolute* complexity. By contrast, PQE computes formula A^* that specifies the "difference" between formulas B and $A \wedge B$. So the efficiency of PQE depends on their *relative* complexity. This is important because no matter how high the individual complexity of B and $A \wedge B$ is, their relative complexity can be quite manageable. In Section B of the appendix we briefly describe an algorithm for PQE and recall some relevant results [7,8,9].

3.2 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 1.4 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 $H \wedge G^{rlx} \wedge (z' \not\equiv z'')$.*

[1] The proofs of propositions are given in Section A of the appendix.

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 $H(b', b'') = 1$. 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 3.1). This observation is the basis for our algorithm of generating EC proofs by induction.

4 Boundary Formulas

In this section, we discuss boundary formulas, a key notion of EC by LoR. Subsection 4.1 explains the semantics of boundary formulas. Subsection 4.2 discusses the size of boundary formulas. In Subsection 4.3, we describe how boundary formulas are built.

4.1 Definition and some properties of boundary formulas

Let M be the subcircuit consisting of the gates of N', N'' located before a cut as shown in Fig. 4. 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.

[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 boundary formulas of Definition 1. The only exception is Section 9 where, to compare LoR and interpolation, we use "property-driven" boundary formulas.

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). As we mentioned in the introduction, 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 $H(z',z'')$ 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.*

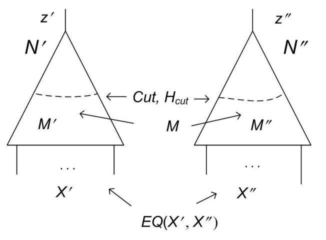

Fig. 4. Building boundary formula H_{cut}

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 $H_{cut} \wedge G^{rlx} \wedge (z' \not\equiv z'')$.*

The proposition below estimates the size of boundary formulas built for N' and N'' that satisfy the notion of structural similarity introduced in Subsection 1.2.

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

Proposition 4 demonstrates the existence of small boundary formulas for structurally similar circuits N',N''. Importantly, the size of these boundary formulas depend on *similarity* of N' and N'' rather than their *individual complexity.*

Corollary 2. *Let circuits M' and M'' of Fig. 4 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. This formula can be represented by $2{*}p$ two-literal clauses where $p = |Cut'| = |Cut''|$.*

Proposition 4 and the corollary above show that the machinery of boundary formulas allows one to extend the notion of an equivalence cut to the case where structurally similar circuits have no functionally equivalent internal variables.

4.2 Size of boundary formulas in general case

Proposition 4 above shows the existence of small boundary formulas for a particular notion of structural similarity. In this subsection, we make two observations that are applicable to a more general class of structurally similar circuits than the one outlined in Subsection 1.2.

The first observation is as follows. Let $\boldsymbol{q}$ be an assignment to the cut of Fig. 4. Assignment $\boldsymbol{q}$ can be represented as $(\boldsymbol{q'},\boldsymbol{q''})$ where $\boldsymbol{q'}$ and $\boldsymbol{q''}$ are assignments to output variables of M' and M'' respectively. Definition 1 does not constrain the value of $H_{cut}(\boldsymbol{q})$ if $\boldsymbol{q}$ cannot be extended to satisfy $F_{N'} \wedge F_{N''}$. So, if, for instance, output $\boldsymbol{q'}$ cannot be produced by M' for any input, the value of $H_{cut}(\boldsymbol{q})$ can be arbitrary. This means that H_{cut} does not have to tell apart cut assignments that *can* be produced by M' and M'' from those that *cannot*. In other words, H_{cut} does not depend on the *individual complexity* of M' and M''. Formula H_{cut} has only to differentiate cut assignments that can be produced *solely* when $\boldsymbol{x'} \neq \boldsymbol{x''}$ from those that can be produced when $\boldsymbol{x'}$=$\boldsymbol{x''}$. Here $\boldsymbol{x'}$ and $\boldsymbol{x''}$ are assignments to X' and X'' respectively.

The second observation is as follows. Intuitively, even a very broad definition of structural similarity of N' and N'' implies the existence of many short clauses relating cut variables that can be derived from $EQ(X',X'') \wedge F_M$. These clauses can be effectively used to eliminate the output assignments of M that can be produced only by inputs $(\boldsymbol{x'},\boldsymbol{x''})$ where $\boldsymbol{x'} \neq \boldsymbol{x''}$. Proposition 4 above substantiates this intuition in case the similarity of N' and N'' is defined as in Subsection 1.2.

4.3 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 $H_0,\dots,H_k$ computed with respect to cuts $Cut_0,\dots,Cut_k$ of N' and N'' (see Fig. 3). 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 located between the inputs of N' and N'' and cut Cut_i (as circuit M of Fig. 4). 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 $H_1,\dots,H_k$ 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} where $i > 1$ be a boundary formula such that $\exists W_{i-1}[H_0 \wedge F_{M_{i-1}}] \equiv H_{i-1} \wedge \exists W_{i-1}[F_{M_{i-1}}]$. 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. (So H_i is a boundary formula due to Proposition 2.)*

5 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 Figure 5. *EC_LoR* builds a sequence of boundary formulas $H_0, \dots, H_k$ as described in Subsection 4.3. 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 $G^{rlx} = F_{N'} \wedge F_{N''}$.

```
$EC\_LoR(N', N'')$ {
1  $(N', N'') := Bufferize(N', N'')$;
2  $Cut_0 = X' \cup X''$;
3  $Cut_k := \{z', z''\}$;
4  $Cut_1, .., Cut_{k-1} := BldCuts(N', N'')$;
5  $H_0 := EQ(X', X'')$;
– – – – – – – – – – – – – – – – –
6  $for(i := 1; i \leq k; i{+}{+})$ {
7     $H_i = 1$;
8     $F_{M_i} := SubForm(G^{rlx}, Cut_i)$;
9     $W_i := Vars(F_{M_i}) \setminus Vars(Cut_i)$;
10    while $(true)$ {
11       $C := Redund(H_i \wedge \exists W_i [H_{i-1} \wedge F_{M_i}])$;
12       if $(C = nil)$ break;
13       $H_i := H_i \wedge C$;}}
– – – – – – – – – – – – – – – – –
14 if $(H_k(0,1) = 1)$
15    if $(Sat(G^{rlx} \wedge \overline{z'} \wedge z''))$ return$(No)$;
16 if $(H_k(1,0) = 1)$
17    if $(Sat(G^{rlx} \wedge z' \wedge \overline{z''}))$ return$(No)$;
18 return$(Yes)$; }
```

Fig. 5. EC by LoR

EC_LoR consists of three parts separated by the dotted lines in Figure 5. *EC_LoR* starts the first part (lines 1-5) by calling procedure *Bufferize* that 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 Section C of the appendix.) Then *EC_LoR* sets the initial and final cuts to $X' \cup X''$ and $\{z', z''\}$ respectively and computes the intermediate cuts.

Boundary formulas H_i, $1 \leq i \leq k$ are computed in the second part (lines 6-13) that consists of a *for* loop. In the third part (lines 14-18), *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 18).

Formula H_i is computed as follows. First, H_i is set to constant 1. Then, *EC_LoR* extracts a subformula F_{M_i} of G^{rlx} that specifies the gates of N' and N'' located between the inputs and cut Cut_i. *EC_LoR* also computes the set W_i of quantified variables. The main work is done in a *while* loop (lines 10-13). First, *EC_LoR* calls procedure *Redund* that is essentially a PQE-solver. *Redund* checks if boundary formula H_{i-1} is redundant in $H_i \wedge \exists W_i[H_{i-1} \wedge F_{M_i}]$ (the cut termination condition). *Redund* stops as soon as it finds out that H_{i-1} is not redundant yet. It returns a clause C as the evidence that at least one clause must be added to H_i to make H_{i-1} redundant. If no clause is returned by *Redund*, then H_i is complete and *EC_LoR* ends the *while* loop and starts a new iteration of the *for* loop. Otherwise, *EC_LoR* adds C to H_i and starts a new iteration of the *while* loop.

6 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 $\exists W_i[H_{i-1} \wedge F_{M_i}]$ whose size 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 [9] 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 (see the discussion in Sections B and E of the appendix.) In this section, we describe two methods of adapting EC by LoR to a PQE-solver that is not efficient enough to compute boundary formulas *precisely.* (Both methods are illustrated experimentally in Section 7.)

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.* The downside of this is that condition b) of Definition 1 does not hold anymore and so EC by LoR becomes *incomplete.* Namely, if $H(b', b'') = 1$ where $b' \neq b''$, 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 $H(1, 0) = H(0, 1) = 0$, circuits N' and N'' are proved equivalent no matter how intermediate boundary formulas have been built. Importantly, checking cut termination conditions is a powerful way to structure the proof even when boundary formulas are computed approximately. That is, construction of H_i still ends when it makes H_{i-1} redundant in formula $H_i \wedge \exists W_i[H_{i-1} \wedge F_{M_i}]$. The only difference from computing H_i precisely is that formula F_{M_i} is simplified by discarding some clauses.

Another way to adapt EC by LoR to an insufficiently efficient PQE solver is as follows. Suppose that the power of a PQE solver is enough 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 1.4, finding such an assignment for β is easier than for α.

7 Experiments

In the experiments, we used the PQE solver published in [9] in 2014. We will refer to this solver as **PQE-14**. As we mentioned in Section 6, 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 (although the design of an efficient PQE-solver is discussed in Section B of the appendix). On the other hand, PQE-14 is efficient enough to make a few important points experimentally. In the experiments described in this section, we employed a new implementation of PQE-14.

The experiment of Subsection 7.1 compares computing cut image with building a boundary formula for this cut. (Recall that the image of a cut is the set of cut assignments that can be produced in N' and N'' under all possible inputs.) This experiment also contrasts complete quantifier elimination employed to compute cut image with PQE. In Subsection 7.2, 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 7.3, we give evidence that boundary formulas can be used to prove inequivalence more efficiently.

In the experiments, circuits N' and N'' to be checked for equivalence were derived from a circuit computing an output median bit of a k-bit multiplier. We will refer to this circuit as Mlp_k. 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 5).

7.1 Image computation versus building boundary formulas

Table 1. *Computing cut image and boundary formula. Time limit = 1 hour*

#bits	#quan. vars	#free vars	cut image (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**

In this subsection, we compared computation of a boundary formula H_{cut} and that of cut image. We used two identical copies of circuit Mlp_k 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''). Computing cut image comes down to performing quantifier elimination for formula $\exists W[EQ(X',X'') \wedge F_M]$. Here $W = X' \cup X''$ and formula F_M specifies the gates of the first topological level of N' and N''. Formula R_{cut} that is logically equivalent to $\exists W[EQ \wedge F_M]$ specifies the cut image. Computing a boundary formula comes down 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 1. Abbreviation QE stands for Quantifier Elimination. The value of k in Mlp_k 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 R_{cut} above we used our quantifier elimination program presented in [8]. Formula H_{cut} was generated by PQE-14. To make this comparison fair, formula H_{cut} was computed without applying any EC-specific heuristics (as opposed to computing boundary formulas in the experiments of Subsection 7.2). When computing image formula R_{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 1 shows, formulas H_{cut} are much smaller than R_{cut} and take much less time to compute.

7.2 An example of equivalence checking by *EC_LoR*

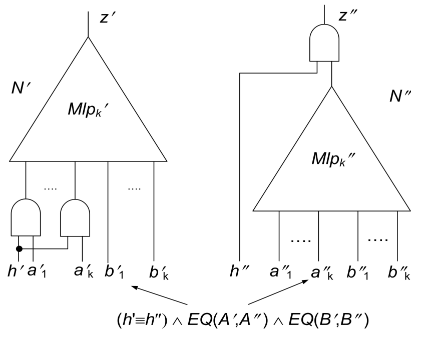

Fig. 6. Equivalence checking of N' and N'' derived from Mlp_k

In this subsection, we run an implementation of *EC_LoR* introduced in Section 5 on circuits N' and N'' shown in Fig. 6. (The idea of this EC example was suggested by Vigyan Singhal [19].) These circuits are derived from Mlp_k by adding one extra input h. Both circuits produce the same output as Mlp_k 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 1.2. 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 $S(v')$ introduced in Subsection 1.2 consisting of only two variables of N''.). In particular, if v' is an internal variable of Mlp'_k, then v'' is the corresponding variable of Mlp''_k. 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_k we used in the experiments). The objective of the experiment below is to show that *EC_LoR* can check for equivalence structurally similar circuits that have no functionally equivalent internal points.

Cuts $Cut_0, \ldots, Cut_m$ 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. 5. We will refer to this version as *EC_LoR**. The main change was that boundary formulas were computed in *EC_LoR** *approximately.* That is when checking if formula H_j was redundant in $H_j \wedge \exists W_i[H_{j-1} \wedge F_{M_i}]$ (line 11 of Fig. 5) only a subset of clauses of F_{M_i} was used to make the check simpler. Nevertheless, *EC_LoR** was able to compute an output boundary formula $H(z', z'')$ proving that N' and N'' were equivalent. One more difference between *EC_LoR* and *EC_LoR** was as follows. *EC_LoR* runs a cut termination check *every time* formula H_i is updated (in the *while* loop of Fig. 5, lines 10-13). In *EC_LoR**, the number of cut termination checks was reduced. Namely, derivation of clauses of H_i was modified so that *EC_LoR** did not run a cut termination check if some cut variable was not present in clauses of H_i yet. The intuition here was that in that case H_i was still under-constrained. *EC_LoR** is described in Section D of the appendix in more detail.

Table 2. *EC of N' and N'' derived from Mlp_k. 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

In Table 2, we compare *EC_LoR** with *ABC* [20]. The first column gives the value of k of Mlp_k 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. (Circuits N' and N'' were fed into *ABC* as circuits in the BLIF format.) Here $X = \{h, a_1, \ldots, a_k, b_1, \ldots, b_k\}$ denotes the set of input variables. 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 2 show that equivalence checking of N' and N'' derived from Mlp_k 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** is taken by PQE-14 when checking cut termination conditions. So, PQE-14 is also the reason why the run time of *EC_LoR** grows quickly with the size of Mlp_k. Using a more efficient PQE-solver should reduce such a strong dependency of the performance of *EC_LoR** on the value of k.

7.3 Using boundary formulas for proving inequivalence

In the experiment of this subsection, we checked for equivalence a correct and a buggy version of Mlp_{16} as circuits N' and N'' 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 experiment 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.

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. 4). 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 $EQ(X', X'')$ is redundant in $H_i \wedge \exists W[EQ(X', X'') \wedge F_{M'_i} \wedge F_{M''_i}]$.) In experiments we used 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 $i = 4$.

Table 3. *Sat-solving of formulas α and β by Minisat. Time limit = 600 s.*

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

We generated 100 buggy versions of Mlp_{16}. Table 3 contains results of checking the satisfiability of 100 formulas α and β by Minisat 2.0 [6,21]. Similar results were observed for the other SAT-solvers we tried. The first column of Table 3 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. 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.

8 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 [14,17,6,2]. 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 [13]. 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 [10,11,16]. However, in some approaches the derived relations are implications [12] 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.

9 Logic Relaxation And Interpolation

In this section, we compare LoR and interpolation. In Subsection 9.1, we give a more general formulation of LoR in terms of arbitrary CNF formulas. In Subsection 9.2, we show that interpolation is a special case of LoR and interpolants are a special case of boundary formulas. We also explain how one can use LoR to interpolate a"broken" implication. This extension of interpolation can be used for generation of short versions of counterexamples. Finally, in Subsection 9.3, we contrast interpolants with boundary formulas employed in EC by LoR.

So far we have considered a boundary formula specifying the difference in assignments satisfying formulas G^{rlx} and G equal to $F_{N'} \wedge F_{N''}$ and $EQ(X', X'') \wedge F_{N'} \wedge F_{N''}$ respectively. In the footnote of Section 4, we mentioned that one can also consider "property driven" boundary formulas. Such formulas specify the difference in assignments satisfying $G^{rlx} \wedge (z' \not\equiv z'')$ and $G \wedge (z' \not\equiv z'')$ rather than G^{rlx} and G. In this section, to simplify explanation, we use "property driven" boundary formulas. They describe the difference in assignments satisfying a relaxed formula and an original formula that is supposed to be unsatisfiable.

9.1 Generalizing LoR to arbitrary formulas

Let $S(X, Z)$ be a formula whose satisfiability one needs to check. Here X and Z are non-overlapping sets of Boolean variables. In the context of formal verification, one can think of S as obtained by conjoining formulas $G(X, Z)$ and $\overline{Good(Z)}$. Here $G(X, Z)$ specifies the consistent design behaviors, X and Z being sets of "internal" and "external" variables. Formula *Good* specifies design behaviors that preserve a required property defined in terms of external variables.

Let formula S be represented as $S^{rlx}(X, Z) \wedge E(X, Z)$. Formula S^{rlx} can be viewed as a relaxation of S that is easier to satisfy. Let $H(Z)$ be a formula obtained by taking E out of the scope of quantifiers in $\exists X[E \wedge S^{rlx}]$ i.e. $\exists X[E \wedge S^{rlx}] \equiv H \wedge \exists X[S^{rlx}]$. Then S is *equisatisfiable* to $H \wedge S^{rlx}$ (see Proposition 6 of the appendix). Checking the satisfiability of $H \wedge S^{rlx}$ reduces to testing the satisfiability of S^{rlx} under assignments to Z for which H evaluates to 1. So, if formula S is "sufficiently" relaxed in S^{rlx} and Z is much smaller than X, solving formula $H \wedge S^{rlx}$ can be drastically simply than S.

One can view H as a *boundary formula* specifying the difference in assignments satisfying S and S^{rlx}. In particular, formula H satisfies the properties of Definition 1 (see Proposition 7 of the appendix). That is a) $S \rightarrow H$ and b) $H(\boldsymbol{z}) = 0$ for every assignment $\boldsymbol{z}$ to Z that can be extended to satisfy S^{rlx} but not S.

9.2 Interpolation as a special case of LoR

Let formula S denote $A(X,Y) \wedge B(Y,Z)$ where X,Y,Z are non-overlapping sets of variables. Let a relaxed formula S^{rlx} be obtained from S by dropping the clauses of A i.e. $S^{rlx} = B$. Let $\exists W[A \wedge B] \equiv H \wedge \exists W[B]$ hold for a formula $H(Y)$ where $W = X \cup Z$. Then, H is a boundary formula in terms of Y for relaxation S_{rlx}. That is from Proposition 7 it follows that a) $S \rightarrow H$ and b) $H(\boldsymbol{y}) = 0$ for every assignment $\boldsymbol{y}$ to Y that can be extended to satisfy S^{rlx} but not S.

Let $A \wedge B \equiv 0$ and $A \rightarrow H$ hold (the latter being a stronger version of $S \rightarrow H$). Then H is an interpolant [5,18,15] for implication $A \rightarrow \overline{B}$ (see Proposition 8 of the appendix). So an interpolant is a *special case* of a boundary formula.

Suppose that $A \rightarrow H$ and $A \wedge B \not\equiv 0$ (and hence $A \not\rightarrow \overline{B}$). Then $H \wedge B \not\equiv 0$ and H can be viewed as an interpolant for the *broken implication* $A \not\rightarrow \overline{B}$. When $A \rightarrow \overline{B}$ holds, $H \rightarrow \overline{B}$ gives a more abstract version of the former. Similarly, if $A \not\rightarrow \overline{B}$, then $H \not\rightarrow \overline{B}$ is a more abstract version of the former. Interpolants of broken implications can be used to generate *short versions of counterexamples.* A counterexample breaking $H \rightarrow \overline{B}$ can be extended to one breaking $A \rightarrow \overline{B}$ (see Proposition 9 of the appendix). So a counterexample for $H \rightarrow \overline{B}$ is a short version of that for $A \rightarrow \overline{B}$.

9.3 Interpolation and LoR in the context of equivalence checking

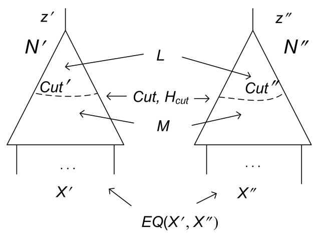

Fig. 7. Replacing/Separating boundary formula H_{cut}

In this subsection, we discuss the difference between boundary formulas and interpolants in the context of EC. Let formulas F_M and F_L specify the gates located below and above a cut as shown in Fig. 7. Then checking the equivalence of N' and N'' comes down to testing the satisfiability of formula S equal to $EQ(X',X'') \wedge F_M \wedge F_L \wedge (z' \not\equiv z'')$.

Below, we contrast two types of relaxation of formula S called *replacing* and *separating* relaxation. The former corresponds to *interpolation* while the latter is the relaxation we studied in the previous sections. A *replacing* relaxation of S is to drop the clauses of $EQ \wedge F_M$.

That is $S^{rlx} = F_L \wedge (z' \not\equiv z'')$. Let H^r_{cut} be a boundary formula computed for replacing relaxation. (Superscript r stands for "replacing".) That is $\exists W[EQ \wedge F_M \wedge F_L \wedge (z' \not\equiv z'')] \equiv H^r_{cut} \wedge \exists W[F_L \wedge (z' \not\equiv z'')]$ where W consists of all the variables of S but cut variables. Note that H^r_{cut} *replaces* all clauses depending on variables corresponding to gates below the cut, hence the name replacing relaxation. Let A denote formula $EQ \wedge F_M$ and B denote formula $F_L \wedge (z' \not\equiv z'')$. From Proposition 8 it follows that if $A \rightarrow H^r_{cut}$ and $A \wedge B \equiv 0$ then H^r_{cut} is an *interpolant* of implication $A \rightarrow \overline{B}$. So an interpolant can be viewed as a boundary formula for replacing relaxation.

A *separating* relaxation of S is to drop the clauses of EQ. As we mentioned above, this kind of relaxation has been the focus of the previous sections. Let H^s_{cut} denote a boundary formula for separating relaxation. (Superscript s stands for "separating".) Formula H^s_{cut} satisfies $\exists W[EQ \wedge F_M \wedge F_L \wedge (z' \not\equiv z'')] \equiv H^s_{cut} \wedge \exists W[F_M \wedge F_L \wedge (z' \not\equiv z'')]$. Note that adding formula H^s_{cut} *separates* input variables X' and X'' of N' and N'' by making formula $EQ(X', X'')$ redundant, hence the name separating relaxation. We will refer to H^r_{cut} and H^s_{cut} as replacing and separating boundary formulas respectively.

Let us assume for the sake of simplicity that a replacing boundary formula H^r_{cut} is an interpolant i.e. it is implied by $EQ \wedge F_M$. We will also assume that a separating boundary formula H^s_{cut} satisfies the condition of Proposition 2 and hence is implied by $EQ \wedge F_M$ as well. An obvious difference between H^r_{cut} and H^s_{cut} is as follows. Adding H^s_{cut} to formula $\exists W[EQ \wedge F_M \wedge F_L \wedge (z' \not\equiv z'')]$ makes redundant only a *subset* of clauses that is made redundant after adding H^r_{cut}. The fact that adding H^r_{cut} has to make redundant both clauses of EQ and F_M creates the following problem with using interpolants for equivalence checking. On the one hand, since H^r_{cut} is implied by $EQ \wedge F_M$, the former can be obtained by resolving clauses of the latter i.e. without looking at the part of N' and N'' above the cut. On the other hand, proving that H^r_{cut} is indeed an interpolant, in general, requires checking that $H^r_{cut} \wedge F_L \wedge (z' \not\equiv z'') \equiv 0$ and hence needs the knowledge of the part of N' and N'' above the cut.

Informally, the problem above means that one cannot build a small interpolant H^r_{cut} using only clauses of $EQ \wedge F_M$. By contrast, one can construct a small separating boundary formula without any knowledge of formula F_L. Let us consider the following simple example. Suppose that the cut of Fig. 7 is an equivalence cut. That is for for every cut point of N' there is a functionally equivalent cut point of N'' and vice versa. From Corollary 2 it follows that formula $EQ(Cut', Cut'')$ is a separating boundary formula. (Here Cut' and Cut'' specify the cut points of N' and $''$ respectively.) This fact can be established from formula $EQ \wedge F_M$ *alone*. However, whether $EQ(Cut', Cut'')$ is an interpolant of implication $A \rightarrow \overline{B}$ (where $A = EQ(X', X'') \wedge F_M$ and $B = F_L \wedge (z' \not\equiv z'')$) *totally depends on formula* F_L i.e. on the part of N' and N'' above the cut.

10 Conclusions

We introduced a new framework for Equivalence Checking (EC) based on Lo-gic 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 Harsh Raju Chamarthi for reading the first version of this paper. My special thanks go 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. W. Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic*, 22(3):269–285, 1957.
6. N. Eén and N. Sörensson. An extensible sat-solver. In *SAT*, pages 502–518, Santa Margherita Ligure, Italy, 2003.
7. E. Goldberg and P. Manolios. Quantifier elimination by dependency sequents. In *FMCAD-12*, pages 34–44, 2012.
8. E. Goldberg and P. Manolios. Quantifier elimination via clause redundancy. In *FMCAD-13*, pages 85–92, 2013.
9. E. Goldberg and P. Manolios. Partial quantifier elimination. In *Proc. of HVC-14*, pages 148–164. Springer-Verlag, 2014.
10. A. Kuehlmann and F. Krohm. Equivalence Checking Using Cuts And Heaps. *DAC*, pages 263–268, 1997.
11. 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.
12. W. Kunz. Hannibal: An efficient tool for logic verification based on recursive learning. In *ICCAD-93*, pages 538–543, 1993.
13. H. Kwak, I. MoonJames, H. Kukula, and T. Shiple. Combinational equivalence checking through function transformation. In *ICCAD-02*, pages 526–533, 2002.
14. J. Marques-Silva and K. Sakallah. Grasp – a new search algorithm for satisfiability. In *ICCAD-96*, pages 220–227, 1996.
15. K. L. Mcmillan. Interpolation and sat-based model checking. In *CAV-03*, pages 1–13. Springer, 2003.

16. A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een. Improvements to combinational equivalence checking. In *ICCAD-06*, pages 836–843, 2006.
17. 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.
18. P. Pudlak. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, 1997.
19. V. Singhal. Private communication.
20. *ABC*. http://www.eecs.berkeley.edu/ alanmi/abc/.
21. Minisat2.0. http://minisat.se/MiniSat.html.

Appendix

A Proofs Of Propositions

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 $H \wedge G^{rlx} \wedge (z' \not\equiv z'')$.*

Proof. A proof of this proposition follows from Propositions 2 and 3 below. Proposition 2 entails that $H(z',z'')$ is a boundary formula. From Proposition 3 it follows that $G \wedge (z' \not\equiv z'')$ is equisatisfiable with $H \wedge G^{rlx} \wedge (z' \not\equiv z'')$.

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.*

Proof. $\exists W[EQ \wedge F_M] \equiv H_{cut} \wedge \exists W[F_M]$ entails $EQ \wedge F_M \rightarrow H_{cut}$. Let L be the subcircuit consisting of the gates of N' and N'' located above the cut. Let F_L be a formula specifying L. Since $G = EQ \wedge F_M \wedge F_L$, then $G \rightarrow H_{cut}$ and so condition a) of Definition 1 is met. Let us prove that condition b) is met as well. Let $\boldsymbol{q}$ be a cut assignment that can be extended to satisfy G^{rlx} but not G. This means that $\boldsymbol{q}$ cannot be extended to an assignment $\boldsymbol{p}$ satisfying $EQ \wedge F_M$ either. (Otherwise, one could easily extend $\boldsymbol{p}$ to an assignment satisfying $EQ \wedge F_M \wedge F_L$ and hence G by using the values of an execution trace computed for circuit L. This trace describes computation of output values of L when its input variables i.e. the cut variables are assigned as in $\boldsymbol{q}$.) So $\exists W[EQ \wedge F_M]$=0 under assignment $\boldsymbol{q}$. This means that $H_{cut} \wedge \exists W[F_M] = 0$ under assignment $\boldsymbol{q}$. Taking into account that $\boldsymbol{q}$ can be extended to an assignment satisfying G^{rlx} and hence F_M, one has to conclude that $H_{cut}(\boldsymbol{q}) = 0$.

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

Proof. Let us show that the satisfiability of the left formula i.e. $G \wedge (z' \not\equiv z'')$ implies that of the right formula i.e. $H_{cut} \wedge G^{rlx} \wedge (z' \not\equiv z'')$ and vice versa.

Left sat. $\rightarrow$ Right sat. Let $\boldsymbol{p}$ be an assignment satisfying $G \wedge (z' \not\equiv z'')$. From Definition 1 it follows that G implies H_{cut} and so H_{cut} is satisfied by $\boldsymbol{p}$.

Since G^{rlx} is a subformula of G, assignment $\boldsymbol{p}$ satisfies G^{rlx} as well. Hence $\boldsymbol{p}$ satisfies $H_{cut} \wedge G^{rlx} \wedge (z' \not\equiv z'')$.

Right sat. $\rightarrow$ Left sat. Let $\boldsymbol{p}$ be an assignment satisfying $H_{cut} \wedge G^{rlx} \wedge (z' \not\equiv z'')$. Let $\boldsymbol{q}$ be the subset of $\boldsymbol{p}$ consisting of the assignments to the cut variables. Since $H_{cut}(\boldsymbol{q})$=1, Definition 1 entails that $\boldsymbol{q}$ can be extended to an assignment $\boldsymbol{p^*}$ satisfying formula G. Since the variables assigned in $\boldsymbol{q}$ form a cut of circuits N' and N'', the consistent assignments to the variables of N' and N'' located above the cut are identical in $\boldsymbol{p}$ and $\boldsymbol{p^*}$. This means that $\boldsymbol{p^*}$ satisfies $(z' \not\equiv z'')$ and hence formula $G \wedge (z' \not\equiv z'')$.

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

Proof. Let $\boldsymbol{q}$ be an assignment to the cut variables that can be extended to satisfy formula G^{rlx} but not formula G. To prove the proposition at hand, one needs to show that there is a clause C consisting of cut variables such that

- C is implied by formula G
- $C(\boldsymbol{q}) = 0$
- C consists of at most $Max(S(v'))$ literals

(Using clauses satisfying the three conditions above one can build a required boundary formula H_{cut}.)

Let $\boldsymbol{p}$ be an assignment satisfying formula G^{rlx} that is obtained by extending $\boldsymbol{q}$. Let $\boldsymbol{x'}$ and $\boldsymbol{x''}$ be the assignments of $\boldsymbol{p}$ to variables of X' and X'' respectively. Note that $\boldsymbol{x'} \neq \boldsymbol{x''}$ (otherwise $\boldsymbol{p}$ would satisfy formula G as well). Cut assignment $\boldsymbol{q}$ can be represented as $(\boldsymbol{q'}, \boldsymbol{q''})$ where $\boldsymbol{q'}$ and $\boldsymbol{q''}$ are assignments of $\boldsymbol{q}$ to Cut' and Cut'' respectively. Assignment $\boldsymbol{q'}$ (respectively $\boldsymbol{q''}$) is produced by circuit M' (respectively M'') under input $\boldsymbol{x'}$ (respectively $\boldsymbol{x''}$).

Let v' be a variable of Cut'. The value of v' is uniquely specified by assignment $\boldsymbol{q''}$ to $S(v')$. So the value of every variable of Cut' is specified by assignment $\boldsymbol{q''}$ to Cut''. Denote by $\boldsymbol{s'}$ the assignment to Cut' specified by $\boldsymbol{q''}$. Let us show that $\boldsymbol{s'} \neq \boldsymbol{q'}$. Assume the contrary i.e. $\boldsymbol{s'} = \boldsymbol{q'}$ and show that then one can extend $\boldsymbol{q}$ to an assignment $\boldsymbol{p^*}$ satisfying formula G and so we have a contradiction. Assignment $\boldsymbol{p^*}$ is constructed as follows. The variables below the cut are assigned in $\boldsymbol{p^*}$ as in the execution trace obtained by applying $\boldsymbol{x''}$ to M' and M''. Note that by assumption, applying input $\boldsymbol{x''}$ to M' will produce cut assignment $\boldsymbol{s'}$ equal to $\boldsymbol{q'}$. The variables above the cut are assigned in $\boldsymbol{p^*}$ as in $\boldsymbol{p}$. Since $\boldsymbol{p}$ satisfies G^{rlx} and X' and X'' are assigned the same input $\boldsymbol{x''}$ in $\boldsymbol{p^*}$, the latter satisfies G. Besides, the cut assignment specified by $\boldsymbol{p^*}$ is $\boldsymbol{q}$ i.e. the same as the one specified by $\boldsymbol{p}$.

Since $\boldsymbol{q'} \neq \boldsymbol{s'}$, there is a variable v' of Cut' that is assigned in $\boldsymbol{q'}$ inconsistently with the assignment of $\boldsymbol{q''}$ to the variables of $S(v')$. Let C'' be the clause of variables of $S(v')$ falsified by $\boldsymbol{q''}$. Let $l(v')$ be the literal of v' falsified by $\boldsymbol{q'}$. Then clause $l(v') \vee C''$ is falsified by $\boldsymbol{q}$. The fact that assignment $\boldsymbol{q'}$ to $S(v')$ determines the value of v' means that clause $l(v') \vee C''$ is implied by formula $F_{M'} \wedge F_{M''}$. Hence $l(v') \vee C''$ is implied by G. Finally, the number of literals in $l(v') \vee C''$ is $|S(v')| + 1$. So clause $l(v') \vee C''$ satisfies the three conditions above.

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} where $i > 1$ be a boundary formula such that $\exists W_{i-1}[H_0 \wedge F_{M_{i-1}}] \equiv H_{i-1} \wedge \exists W_{i-1}[F_{M_{i-1}}]$. 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. (So H_i is a boundary formula due to Proposition 2.)*

Proof. Let ϕ denote formula $\exists W_i[H_0 \wedge F_{M_i}]$. Let $F_{i-1,i}$ be the set of clauses equal to $F_{M_i} \setminus F_{M_{i-1}}$. Formula ϕ can be represented as $\exists W_{i-1} \exists W_{i-i,i}[H_0 \wedge F_{M_{i-1}} \wedge F_{i-1,i}]$ where $W_{i-1,i} = W_i \setminus W_{i-1}$. Taking into account that formula $F_{i-1,i}$ does not depend on variables of W_{i-1}, one can rewrite formula ϕ as $\exists W_{i-1,i}[F_{i-1,i} \wedge \exists W_{i-1}[H_0 \wedge F_{M_{i-1}}]]$. Using the assumption imposed on H_{i-1} by the proposition at hand, one can transform formula ϕ into $\exists W_{i-1,i}[F_{i-1,i} \wedge H_{i-1} \wedge \exists W_{i-1}[F_{M_{i-1}}]]$. After putting $F_{i-1,i}$ and H_{i-1} back under the scope of quantifiers, ϕ becomes equal to $\exists W_{i-1,i}[\exists W_{i-1}[H_{i-1} \wedge F_{M_{i-1}} \wedge F_{i-1,i}]]$ and hence to $\exists W_i[H_{i-1} \wedge F_{M_i}]$. Since $\exists W_i[H_{i-1} \wedge F_{M_i}] \equiv H_i \wedge \exists W_i[F_{M_i}]$ holds we get that the original formula ϕ equal to $\exists W_i[H_0 \wedge F_{M_i}]$ is logically equivalent to $H_i \wedge \exists W_i[F_{M_i}]$.

Proposition 6. *Let $S(X,Z)$, $S^{rlx}(X,Z)$, $E(X,Z)$ and $H(Z)$ be Boolean formulas where X,Z are non-overlapping sets of variables. Let $S = E \wedge S^{rlx}$ and $\exists X[E \wedge S^{rlx}] \equiv H \wedge \exists X[S^{rlx}]$ hold. Then S is equisatisfiable with $H \wedge S^{rlx}$.*

Proof. By assumptions of the proposition, $\exists X[S] \equiv \exists X[H \wedge S^{rlx}]$. So if formula S is satisfiable, there is an assignment $\boldsymbol{z}$ to the variables of Z for which $\exists X[S]$ evaluates to 1. Since formula $\exists X[H \wedge S^{rlx}]$ also evaluates to 1 for $\boldsymbol{z}$, formula $H \wedge S^{rlx}$ is satisfiable too. Similarly, one can show that the satisfiability of $\exists X[H \wedge S^{rlx}]$ means that that S is satisfiable too.

Proposition 7. *Let formula $S(X,Z)$ be represented as $E(X,Z) \wedge S^{rlx}(X,Z)$ where X,Z are non-overlapping sets of Boolean variables. Let $\exists X[E \wedge S^{rlx}] \equiv H \wedge \exists X[S^{rlx}]$ hold for a formula $H(Z)$. Then H is a boundary formula in terms of Z for relaxation S^{rlx} (see Definition 1). That is*

a) $S \rightarrow H$ and
b) for every assignment $\boldsymbol{z}$ to Z that can be extended to satisfy S^{rlx} but not S, the value of $H(\boldsymbol{z})$ is 0.

Proof. $\exists X[E \wedge S^{rlx}] \equiv H \wedge \exists X[S^{rlx}]$ entails $E \wedge S^{rlx} \rightarrow H$. So condition a) is met. Let us show that condition b) holds as well. Let $\boldsymbol{z}$ be an assignment to Z that can be extended to satisfy S^{rlx} but not S. This means that $\exists X[E \wedge S^{rlx}]$ and $\exists X[S^{rlx}]$ evaluate to 0 and 1 respectively under assignment $\boldsymbol{z}$. Hence $H(\boldsymbol{z})$ has to be equal to 0 to preserve $\exists X[E \wedge S^{rlx}] \equiv H \wedge \exists X[S^{rlx}]$.

Proposition 8. *Let $A(X,Y)$ and $B(Y,Z)$ be formulas where X,Y,Z are non-overlapping sets of variables. Let $A \wedge B \equiv 0$. Formula $H(Y)$ is an interpolant of implication $A \rightarrow \overline{B}$ iff $A \rightarrow H$ and $\exists W[A \wedge B] \equiv H \wedge \exists W[B]$ where $W = X \cup Z$.*

Proof. If part. Suppose that $A \rightarrow H$ holds and $\exists W[A \wedge B] \equiv H \wedge \exists W[B]$. Since $A \rightarrow \overline{B}$ holds, then $A \wedge B \equiv 0$ and so $H \wedge B \equiv 0$. Hence $H \rightarrow \overline{B}$ and H is an interpolant of implication $A \rightarrow \overline{B}$.

Only if part. Suppose that H is an interpolant and so $A \rightarrow H$ and $H \rightarrow \overline{B}$ hold. Assume that $\exists W[A \wedge B] \not\equiv H \wedge \exists W[B]$. Since $H \rightarrow \overline{B}$ and hence $H \wedge B \equiv 0$, this means that $A \wedge B \not\equiv 0$. So we have a contradiction.

Proposition 9. *Let $A(X,Y) \wedge B(Y,Z) \not\equiv 0$ where X,Y,Z are non-overlapping sets of variables. Let $H(Y)$ be a formula such that $\exists W[A \wedge B] \equiv H \wedge \exists W[B]$ where $W = X \cup Z$. Let $\boldsymbol{y}$ and $\boldsymbol{z}$ be assignments to Y and Z respectively such that $(\boldsymbol{y},\boldsymbol{z})$ satisfies $H \wedge B$. Then $(\boldsymbol{y},\boldsymbol{z})$ can be extended to an assignment satisfying $A \wedge B$.*

Proof. The fact that $\exists W[A \wedge B] \equiv H \wedge \exists W[B]$ holds and $H \wedge B$ is satisfied by $(\boldsymbol{y},\boldsymbol{z})$ means that $\boldsymbol{y}$ can be extended to an assignment $(\boldsymbol{x^*},\boldsymbol{y},\boldsymbol{z^*})$ satisfying $A \wedge B$. Then assignment $(\boldsymbol{x^*},\boldsymbol{y},\boldsymbol{z})$ satisfies $A \wedge B$ as well. Indeed, $(\boldsymbol{x^*},\boldsymbol{y})$ satisfies A and $(\boldsymbol{y},\boldsymbol{z})$ does B.

B Algorithm For Partial Quantifier Elimination

In this section, we discuss Partial Quantifier Elimination (PQE) in more detail. In Subsection B.1, we give a high-level description of a PQE-solver. This PQE-solver is based on the machinery of Dependency sequents (D-sequents) that we recall in Subsection B.2.

B.1 A PQE solver

In this subsection, we describe our algorithm for PQE introduced in [9] in 2014. We will use the same name for this algorithm as in Section 7, i.e. PQE-14. Let $A(V,W), B(V,W)$ be Boolean formulas where V,W are non-overlapping sets of variables. As we mentioned in Subsection 3.1, the PQE problem is to find formula $A^*(V)$ such that $\exists W[A \wedge B] \equiv A^* \wedge \exists W[B]$. We will refer to a clause containing a variable of W as a ***W*-clause**. PQE-14 is based on the three ideas below.

First, finding formula A^* comes down to generation of clauses depending only on variables of V that make the W-clauses of A redundant in $A^* \wedge \exists W[A \wedge B]$. *Second*, the clauses of A^* can be derived by resolving clauses of $A \wedge B$. The intermediate resolvents that are W-clauses need to be proved redundant along with the original W-clauses of A. However, since formula B remains quantified, there is no need to prove redundancy of W-clauses of B or W-clauses obtained by resolving only clauses of B.

Third, since proving redundancy of a clause is a hard problem it makes sense to partition this problem into simpler subproblems. To this end, PQE-14 employs branching. After proving redundancy of required clauses in subspaces, the results of branches are merged. The advantage of branching is that for every W-clause C one can always reach a subspace where C can be trivially proved redundant. Namely, C is trivially redundant in the current subspace if a) C is satisfied in the current branch; b) C is implied by some other clause; c) there is an unassigned variable y of C where $y \in W$, such that C cannot be resolved on y with other clauses that are not satisfied or proved redundant yet.

B.2 Dependency sequents

PQE-14 branches on variables of $V \cup W$ until the W-clauses that are descendants of W-clauses of A are proved redundant in the current subspace. To keep track of conditions under which a W-clause becomes redundant in a subspace, PQE-14 uses the machinery of Dependency sequents (D-sequents) developed in [7,8]. A **D-sequent** is a record of the form $(\exists W[A \wedge B], \boldsymbol{q}) \rightarrow \{C\}$. It states that clause C is redundant in formula $\exists W[A \wedge B]$ in subspace $\boldsymbol{q}$. Here $\boldsymbol{q}$ is an assignment to variables of $V \wedge W$ and A is the *current* formula that consists of the initial clauses of A and the resolvent clauses. When a W-clause C is proved redundant in a subspace, this fact is recorded as a D-sequent. If $\boldsymbol{q} = \emptyset$, the D-sequent is called *unconditional*. Derivation of such a D-sequent means that clause C is redundant in the current formula $\exists W[A \wedge B]$ in the entire space.

The objective of PQE-14 is to derive unconditional D-sequents for all W-clauses of A and their descendants that are W-clauses. A new D-sequent can be obtained from two parent D-sequents by a resolution-like operation on a variable y. This operation is called *join*. When PQE-14 merges the results of branching on variable y it joins D-sequents obtained in branches $y = 0$ and $y = 1$ at variable y. So the resulting D-sequents do not depend on y. If formula $A \wedge B$ is unsatisfiable in both branches, a new clause C is added to formula A. Clause C is obtained by resolving a clause falsified in subspace $y = 0$ with a clause falsified in subspace $y = 1$ on y. Adding C makes all W-clauses redundant in the current subspace. By the time PQE-14 backtracks to the root of the search tree, it has derived unconditional D-sequents for all W-clauses of the current formula A.

Algorithms based on D-sequents (including PQE solving) is work in progress. So they still lack some important techniques like D-sequent re-using. In the current algorithms based on D-sequents, the parent D-sequents are discarded as soon as they produce a new D-sequent by the join operation. Although D-sequent re-using promises to be as powerful as re-using learned clauses in SAT-solving, it requires more sophisticated bookkeeping and so is not implemented yet [9].

C Generation Of Cuts That Do Not Overlap

An important part of *EC_LoR* described in Section 5 is to build non-overlapping cuts. These cuts are used to generate a sequence of boundary formulas converging to an output boundary formula. As we mentioned there, the presence of non-local connections makes it hard to find cuts that do not overlap. In this section, we consider this issue in more detail. First, we give the necessary definitions and describe the problem. Then we explain how one can get rid of non-local connections by buffer insertion.

Let M be a multi-output circuit. The *length* of a path from an output of a gate to an input of another gate is measured by the number of gates on this path.The *topological level* of a gate g is the longest path from an input of M to g. We treat the inputs of M as special gates that are not fed by other gates. We will denote the topological level of gate g as $\mathit{TopLvl}(g)$. It can be computed recursively as follows. If g is an input, then $\mathit{TopLvl}(g) = 0$. Otherwise, $\mathit{TopLvl}(g)$ is equal to the maximum topological level among the gates feeding g plus 1.

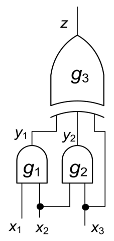

Fig. 8. A circuit

We will call gates g_i and g_j *topologically independent* if there is no path from an input to an output of M going through both these gates. For instance, gates g_1 and g_2 in Fig. 8 are topologically independent. We will call a set S of gates a **cut**, if every path from an input to an output of M goes through a gate of S. A cut S is *minimal*, if for every gate $g \in S$, set $S \setminus \{g\}$ is not a cut. *EC_LoR* employs only minimal cuts. In this section, we use the notion of a gate and the variable specifying its output interchangeably. For example, the topological level of a variable v specifying the output of gate g (denoted as $\mathit{TopLvl}(v)$) is equal to $\mathit{TopLvl}(g)$.

If gate g_i of M feeds gate g_j and $\mathit{TopLvl}(g_j) > \mathit{TopLvl}(g_i) + 1$, then g_i and g_j are said to have a **non-local connection**. Non-local connections make topologically dependent gates appear on the same cut. Consider the circuit of Fig 8. The input gate x_3 feeds gates g_2 and g_3. Since $\mathit{TopLvl}(x_3)$=0 and $\mathit{TopLvl}(g_3)$=2, the connection between x_3 and g_3 is non-local. This leads to appearance of cut $\{y_1, y_2, x_3\}$ where variables y_2 and x_3 are topologically dependent. If gate g_i feeds gate g_j and this connection is non-local, gate g_i appears in every cut that separates g_i and g_j and does not include g_j. So the presence of a large number of non-local connections leads to the heavy overlapping of cuts.

There are a few techniques for dealing with non-local connections of N' and N'' in the context of EC by LoR. The simplest one is to insert buffers. A buffer is a single-input and single-output gate that copies its input to the output. Let g_i and g_j be gates of N' such that a) g_i feeds g_j and b) $\mathit{TopLvl}(g_j) > \mathit{TopLvl}(g_i)+1$. By inserting $\mathit{TopLvl}(g_j) - \mathit{TopLvl}(g_i) - 1$ buffers between g_i to g_j, this non-local connection is replaced with $\mathit{TopLvl}(g_j) - \mathit{TopLvl}(g_i)$ local connections.

D Version of *EC_LoR* Used In Experiments

In the experiments of Subsection 7.2, we used a version of *EC_LoR* that was modified in comparison to the description given in Fig. 5. We will refer to this version as *EC_LoR**. In this section, we describe *EC_LoR** in more detail.

Boundary formula H_i was computed in *EC_LoR** as follows. If there was a variable specifying the output of a cut gate g' that was not present in a clause of

H_i, *EC_LoR** called the procedure below. That procedure generated short clauses relating the output variable of g' and those of its "relatives" from N''. This way, *EC_LoR** avoided running a cut termination check before every variable of i-th cut was present in a clause of H_i.

Clauses of H_i constraining variable of g' were generated as follows. First, *EC_LoR** identified the relatives $g''_1,\dots,g''_m$ of gate g' in N''. A gate g''_j was considered a relative of g' if there was a clause of formula H_{i-1} relating input variables of g' and g''_j. Finally, a set of clauses A^* relating the output variable of gate g' and those of its relatives was generated and added to formula H_i. The clauses of A^* were obtained by taking formula A out of the scope of quantifiers in $\exists W[A \wedge B]$. Here A is the set of clauses of formulas H_{i-1} containing the input variables of gate g' and its relatives. Formula B contains the clauses specifying gate g' and its relatives. Set W consists of the variables of $A \wedge B$ minus output variables of g' and its relatives.

Another modification of *EC_LoR** was that boundary formulas were computed approximately. In line 11 of Fig 5, formula F_{M_i} specifying the gates located between inputs of N' and N'' and i-th cut is used to compute a new clause of H_i. In *EC_LoR** only the subset of clauses of F_{M_i} specifying the gates located between $(i-1)$-th and i-th cuts was used when computing H_i.

E Computing Boundary Formulas Efficiently

Computation of a boundary formula is based on PQE solving. In turn, a PQE-solver is based on derivation of D-sequents (see Subsection B.2 of the appendix). As we showed in Subsection 4.3, boundary formula H_i is obtained by taking H_{i-1} out of the scope of quantifiers in formula $\exists W_i[H_{i-1} \wedge F_{M_i}]$. Here F_{M_i} specifies the gates located between inputs of circuits N', N'' and i-th cut and W_i is the set of variables of F_{M_i} minus those of the i-th cut. Since the size of formula F_{M_i} grows with i, a PQE-solver that computes H_i *precisely* must have high scalability. PQE-14 (see Section B) does not scale well yet because it does not re-use D-sequents. In this section, we argue that once D-sequent re-using is implemented, efficient computation of boundary formulas becomes quite possible.

Consider the scalability problem in more detail. Formula H_i is obtained by generating a set of clauses that make the clauses of H_{i-1} redundant. Let $C \in H_{i-1}$ be a clause whose redundancy one needs to prove. PQE-14 is a branching algorithm. Clause C is trivially redundant in every subspace where C is satisfied. Proving redundancy is non-trivial only in the subspace where C is falsified. To prove C redundant in such a subspace, PQE-14 uses the machinery of local proofs of redundancy described below. (For the sake of simplicity we did not mention this aspect of PQE-14 in Section B.)

Suppose clause C above contains the positive literal of variable v. Suppose, in the current branch, literal v is unassigned and all the other literals of C are falsified. So C is currently a unit clause. Then PQE-14 marks all the clauses containing literal $\overline{v}$ that can be resolved with C as ones that have to be proved redundant in branch $v = 1$ i.e. *locally.* This is done even for clauses of F_{M_i} (that do not have to be proved redundant *globally* because F_{M_i} remains quantified). The obligation to prove redundancy of clauses with literal $\overline{v}$ is made to prove redundancy of C in the branch where $v = 0$ and C is falsified. This obligation is canceled immediately after PQE-14 backtracks to the node w of the search tree that precedes node v. When exploring branch $v = 1$ one of the two alternatives occurs. If formula is UNSAT in this branch, a new clause is generated that subsumes C in node w. Adding this clause to $\exists W_i[H_{i-1} \wedge F_{M_i}]$ makes C redundant in node w. Otherwise, clauses with literal $\overline{v}$ are proved redundant in node w and so C is redundant in node w because it cannot be resolved on v in the current subspace. (This also means that C is redundant in branch $v = 1$ where C is falsified.)

To prove that a clause B with literal $\overline{v}$ is redundant in branch $v = 1$ one may need to make obligations to prove redundancy of some other clauses that can be resolved with B on one of its variables and so on. So proving redundancy of one clause C makes PQE-14 prove local redundancy of many clauses. Currently PQE-14 discards a D-sequent as soon as it is joined at a branching variable of the search tree (with some other D-sequent). Moreover, the D-sequent of a clause that one needs to prove only locally is discarded after the obligation to prove redundancy of this clause is canceled. This cripples the scalability of PQE-14 because one has to reproduce D-sequents seen before over and over again. As the size of formula F_{M_i} grows, more and more clauses need to be proved redundant locally and the size of the search tree blows up.

Re-using D-sequents should lead to drastic reduction of the search tree size for two reasons. First, when proving redundancy of clause C one can immediately discard every clause whose D-sequent states the redundancy of this clause in the current subspace. Second, one can re-use D-sequents of clauses of F_{M_j}, $j < i$ that were derived when building formula H_j. Informally, D-sequent re-using should boost the performance of a PQE algorithm like re-using learned clauses boosts that of a SAT-solver.