Equivalence Checking By Logic Relaxation

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 set A that is a superset of new behaviors. That is A contains all new satisfying assignments introduced into the relaxed formula and does not contain an assignment satisfying the original formula. Set A is generated by a procedure called partial quantifier elimination. If all possible bad behaviors are in A, the original formula cannot have them and so is correct. 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*, which simplifies bug hunting. We give some 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 hence has a strong impact on design quality. Third, inutitively, there should exist robust methods of proving by induction the EC of circuits computing values in a "similar manner". Such methods can also be beneficial for EC of sequential circuits and even software.

1.2 EC by Logic Relaxation

Let $N(X,Y,z)$ be a single-output combinational circuit where X and Y are sets of input and internal variables and z specifies the output of N. We will say that a Boolean formula $F(X,Y,z)$ specifies N if every assignment satisfying F is a consistent assignment to variables of N and vice versa.We will assume that all formulas mentioned in the paper are Boolean formulas in Conjunctive Normal Form (CNF) unless otherwise stated. When we say that $\boldsymbol{p}$ is an assignment to a set of variables V we assume that $\boldsymbol{p}$ is a *complete* assignment unless otherwise stated. That is every variable of V is assigned a value in $\boldsymbol{p}$.

Let N' and N'' be single-output circuits to be checked for equivalence. 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. 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 X' and X'' of input variables but assume that only the input assignments satisfying EQ matter.)

Let $F_{N'}(X',Y',z')$ and $F_{N''}(X'',Y'',z'')$ be formulas specifying N' and N'' respectively. Let G denote formula $F_{N'} \wedge F_{N''} \wedge EQ$. Circuits N' and N'' are equivalent iff $G \wedge (z' \neq z'')$ is unsatisfiable. Denote by G^{rlx} the formula obtained by dropping the clauses of EQ from G i.e. $G^{rlx} = F_{N'} \wedge F_{N''}$. Formula G^{rlx} can be viewed as a *relaxation* of G. Obviously, every assignment satisfying G also satisfies G^{rlx} but the opposite is not true.

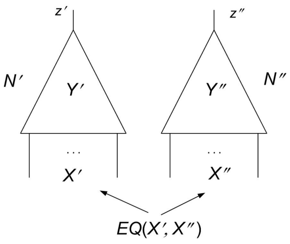

Fig. 1. Equivalence checking of N' and N''

EC by **Logic Relaxation (LoR)** presented in this paper is based on the following idea. Suppose that one manages to compute a superset of the set of new satisfying assignments of G^{rlx} that appeared due to discarding EQ. Suppose that this superset is specified in terms of output variables z', z'' by a formula $H(z',z'')$ defined as follows. If $z' = b', z'' = b''$ can be extended to an assignment satisfying G, then $H(b',b'') = 1$. If $z' = b', z'' = b''$ can be extended to an assignment satisfying G^{rlx} but not G, then $H(b',b'') = 0$. By examining the truth table of H, one can easily establish whether N' and N'' are equivalent. $H(0,1) = H(1,0) = 0$ implies that N' and N'' are equivalent: only assignments satisfying G^{rlx} but not G can have $z' = b', z'' = b''$,$b' \neq b''$.

On the other hand, if $H(b',b'') = 1$, then G^{rlx} and G are *equisatisfiable* under assignment $z' = b', z'' = b''$. So, if, say $H(0,1) = 1$, the satisfiability of G^{rlx} under assignment $z' = 0, z'' = 1$ means that G is also satisfiable under $z' = 0, z'' = 1$. Hence N' and N'' are inequivalent. Recall that formula G^{rlx} specifies circuits N' and N'' where input variables X' and X'' are independent of other. So checking the satisfiability of G^{rlx} under $z' = 0, z'' = 1$ means finding an input assignment $\boldsymbol{x'}$ for which N' evaluates to 0 and an input assignment $\boldsymbol{x''}$ for which N'' evaluates to 1. In other words, proving the satisfiability of G^{rlx} under $z' = 0, z'' = 1$ comes down to showing that N' is not constant 1 and N'' is not constant 0. An assignment $\boldsymbol{p}$ satisfying G^{rlx} does not, in general, satisfy G. So $\boldsymbol{p}$ is a *proof of inequivalence* of N' and N'' rather than a counterexample.

Formula H above is computed via constructing a sequence of so-called boundary formulas. A **boundary formula** specifies the difference (i.e. a "boundary") between G and G^{rlx} in terms of the variables of a cut. Formulas $EQ(X',X'')$ and $H(z',z'')$ are actually boundary formulas in terms of cuts $X' \cup X''$ and $\{z',z''\}$ respectively. Boundary formulas are computed by a technique called *partial quantifier elimination* (PQE) introduced in [11]. 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.

1.3 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. They can potentially handle any pair of circuits N', N'' that compute their output value in a similar way. By contrast, the current approaches (see e.g. [14, 15, 20]) employ very fragile induction proofs e.g. those that require existence of functionally equivalent internal points. Second, in EC by LoR, one can prove inequivalence of N' and N'' by showing the satisfiability of a relaxed formula when $z' \neq z''$. This can potentially boost bug hunting.

1.4 Contributions and structure of the paper

Our contributions is as follows. First, we present a new method of EC based on LoR that facilitates generation of inductive proofs. (Note that the success of EC by LoR srongly depends on the efficiency of computing boundary formulas by PQE-solvers. So the objective of this paper is to introduce a general method for efficient EC rather than describe an implementation of a full-fledged equivalence checker.) Second, we show that PQE, a "light" version of quantifier elimination, enables new powerful verification algorithms. Third, we give some experimental evidence in support of EC by LoR. Fourth, we relate interpolation to LoR beyond EC. In particular, we show how to generate short versions of long counterexamples by using boundary formulas to interpolate broken implications.

The structure of this paper is as follows. Section 2 discusses the challenge of proving EC by induction. In Section 3, we discuss proving (in)equivalence by LoR and relate it to partial quantifier elimination. Boundary formulas are discussed in Sections 4 and 5. Section 6 presents an algorithm of EC by LoR. Section 7 provides experimental evidence in favor of our approach. In Section 8, some background is given. Section 9 discusses taking LoR beyond EC. In particular, it relates LoR to interpolation. We make conclusions in Section 10.

2 Proving EC By Induction

Intuitively, when N' and N'' are structurally similar (i.e. they compute their value in a similar manner) there should exist a short inductive proof shown in Fig. 2. In this proof, for every set Cut_i forming a cut, only a small set H_i of short clauses relating points of N' and N'' that are in 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 the equivalence of z' and z'' are derived.

Give a formal definition of a Cut: Cut ⊆ Vars(N')∪ Vars(N")

The existing EC tools cannot build the inductive proofs above for the following reason. Finding such proofs requires *cut-wise* induction whereas these tools employ much more fragile induction schemes e.g. *subcircuit-wise* induction [2]. The latter is based on the following observation: the outputs of single-output subcircuits K' and K'' of N' and N'' are functionally equivalent if a) K' and

K'' are functionally equivalent to each other as stand-alone circuits and b) the corresponding input variables of K' and K'' are proved equivalent in N' and N''. Such induction works only when N' and N'' have many functionally equivalent internal points.

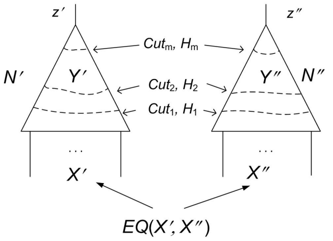

Fig. 2. An inductive proof of equivalence

Proving EC by cut-wise 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 and L_i denote the subcircuits consisting of the gates of N' and N'' located below and above i-th cut respectively (like subcircuits M and L of Fig. 3). A straightforward way to build an inductive proof is to make formula H_i specify the *range* of circuit M_i. 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}. However, a flaw of this approach is that a formula specifying the range of M_i can get prohibitively large.

Another approach is to build H_i as an *interpolant* [6, 19]. Let $A_i = EQ(X', X'') \wedge F_{M_i}$ and $B_i = F_{L_i} \wedge (z' \neq z'')$ where formulas F_{M_i} and F_{L_i} specify circuits M_i and L_i respectively. If N' and N'' are equivalent then $A_i \wedge B_i \equiv 0$. Let H_i be an interpolant for implication $A_i \rightarrow \overline{B_i}$ i.e. $A_i \rightarrow H_i$ and $H_i \rightarrow \overline{B_i}$. Then formula $EQ \wedge F_{M_i} \wedge F_{L_i} \wedge (z' \neq z'')$ reduces to $H_i \wedge F_{L_i} \wedge (z' \neq z'')$. After that a new interpolant can be built for implication $H_i \rightarrow \overline{B_i}$ to further simplify formula $H_i \wedge F_{L_i} \wedge (z' \neq z'')$ and so on. The appeal of this approach is that the existence of a short proof of $A_i \wedge B_i \equiv 0$ implies that of a small interpolant and hence small formula H_i. Unfortunately, finding a small interpolant is as hard as building a short proof. In particular, finding a small interpolant requires examining *the entire formula* $A_i \wedge B_i$ (although a poor quality interpolant can be derived from A_i alone by quantifier elimination).

A solution offered in EC by LoR is to use the boundary formulas mentioned in Subsection 1.2 as formulas H_i. In addition to simple semantics (as formulas specifying the difference between G and G^{rlx}) boundary formulas have three nice qualities. First, the size of a boundary formula depends on the structural similarity of circuits N' and N'' rather than their individual complexity. Second, 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 makes formula H_{i-1} *redundant* in some quantified formula. Third, unlike interpolant derivation, one can build a small boundary formula H_i from formula A_i *alone*.

3 Proving (In)equivalence By LoR. Its Relation To 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 clause in a quantified formula. In particular, solving the PQE problem above comes down to finding a set of clauses $A^*(V)$ implied by $A \wedge B$ that makes the clauses of A redundant in $A^* \wedge \exists W[A \wedge B]$. Then $\exists W[A \wedge B] \equiv A^* \wedge \exists W[A \wedge B] \equiv A^* \wedge \exists W[B]$.

Let $G(V)$ be a formula implied by B. Then $\exists W[A \wedge B] \equiv A^* \wedge G \wedge \exists W[B]$ implies that $\exists W[A \wedge B] \equiv A^* \wedge \exists W[B]$. 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 operates on two formulas (A and $A \wedge B$) and its efficiency depends on their *relative* complexity. This is important because no matter how high the individual complexity of these formulas is, their relative complexity can be quite manageable. In Section B of the appendix we briefly describe an algorithm for PQE and some relevant results presented in [9–11].

3.2 Proving equivalence/inequivalence by LoR

Proposition 1 shows how one proves equivalence/inequivalence of circuits by LoR[1]. Recall from Subsection 1.2 that $G = EQ \wedge F_{N'} \wedge F_{N''}$ and $G^{rlx} = F_{N'} \wedge F_{N''}$. 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 $H(z',z'')$ denote 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 $G \wedge (z' \neq z'')$ is equisatisfiable with $H \wedge G^{rlx} \wedge (z' \neq z'')$.*

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

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

implies that proving *inequivalence* of N' and N'' comes down to showing that formula G^{rlx} is satisfiable under assignment $(z' = b', z'' = b'')$ 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 only clauses of G^{rlx} are just "noise" (see Subsection 3.1). This observation is the basis of proving EC by induction described in Section 5.

4 Boundary Formulas

A key part of EC by LoR is to build a sequence of boundary formulas. In this section, we discuss them in more detail. Subsection 4.1 explains the semantics of boundary formulas. In Subsection 4.2, they are compared with interpolants in the context of EC. (The relation between interpolants and boundary formulas beyond EC is described in Section 9.)Subsection 4.3 discusses the size of boundary formulas.

boundary formulas and interpolants are compared in the context of EC. Section 9 discuss the relation in more detail.

4.1 Definition and some properties of boundary formulas

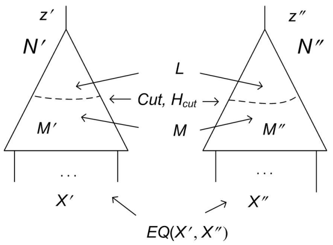

Fig. 3. Building boundary formula H_{cut}

Let M and L be subcircuits consisting of the gates of N', N'' located before and after a cut respectively as shown in Fig. 3. Formula G^{rlx} equal to $F_{N'} \wedge F_{N''}$ can also be represented as $F_M \wedge F_L$ where F_M and F_L specify subcircuits M and L respectively. As usual, G denotes $EQ(X', X'') \wedge F_{N'} \wedge F_{N''}$.

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*

a) $G \rightarrow H_{cut}$ holds
b) in the case $\boldsymbol{q}$ can be extended to an assignment satisfying G_{rlx} but not G, the value of $H_{cut}(\boldsymbol{q})$ is 0.

Note that Definition 1 does not specify the value of $H_{cut}(\boldsymbol{q})$ if $\boldsymbol{q}$ *cannot* be extended to an assignment satisfying G^{rlx} (and hence G). Formula $EQ(X', X'')$ relating input variables of N' and N'' and formula $H(z', z'')$ introduced by 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.*

Note that $\exists W[EQ \wedge F_M] \equiv H_{cut} \wedge \exists W[F_M]$ entails $EQ \wedge F_M \rightarrow H_{cut}$ whereas Definition 1 requires only $EQ \wedge F_M \wedge F_L \rightarrow H_{cut}$. So Proposition 2 gives only a sufficient condition for H_{cut} to be a boundary formula. Proposition 3 below extends Proposition 1 to an arbitrary boundary formula H_{cut}.

Can you possibly give a simple example why the converse is not true. I guess $EQ \wedge F_M \wedge F_L \Rightarrow H_{cut}$ in general does not imply $EQ \wedge F_M \Rightarrow H_{cut}$. Not sure though how condition (b) affects this.

Proposition 3. *Let H_{cut} be a boundary formula with respect to a cut. Then $G \wedge (z' \neq z'')$ is equisatisfiable with $H_{cut} \wedge G^{rlx} \wedge (z' \neq z'')$.*

The proposition below gives an example of formula H_{cut} for the case when circuits M' and M'' of Fig 3 are functionally equivalent.

Proposition 4. *Let Cut' and Cut'' be subsets of Cut specifying the outputs of circuits M' and M'' of Fig. 3 respectively. Let M' and M'' be functionally equivalent to each other. Then $H_{cut} = EQ(Cut', Cut'')$ is a boundary formula.*

Removed this:Let H_{cut} be equal to formula $EQ(Cut', Cut'')$ stating equivalence of corresponding output variables of M' and M''.

$EQ(Cut', Cut'')$ states equivalence of corresponding output variables of M' and M''. The equivalence of variables $v' \in Cut'$ and $v'' \in Cut''$ can be expressed by CNF formula $(\overline{v'} \vee v'') \wedge (v' \vee \overline{v''})$. Let $|Cut'| = |Cut''| = p$. Then formula H_{cut} of Proposition 4 consists of $2 * p$ clauses specifying equivalences between p pairs of equivalent variables of Cut' and Cut''. Every clause has only two literals. Note that this result does not depend on complexity of circuits M' and M''.

added this

Why is this? One still needs to show that M' and M" are equivalent which will depend on its complexity. Did you mean complexity of L?

4.2 Boundary formulas and interpolants in the context of EC

In this subsection, we discuss the difference between boundary formulas and interpolants [6, 19, 12] in the context of EC. Let R_{cut} be a formula depending only on the variables of a cut (see Fig. 3). Let N' and N'' be equivalent and hence $A \wedge B \equiv 0$ where $A = EQ(X', X'') \wedge F_M$ and $B = F_L \wedge (z' \neq z'')$. Assume that implications $A \rightarrow R_{cut}$ and $R_{cut} \rightarrow \overline{B}$ hold. Then R_{cut} is an interpolant for implication $A \rightarrow \overline{B}$ and $EQ \wedge F_M \wedge F_L \wedge (z' \neq z'')$ reduces to $R_{cut} \wedge F_L \wedge (z' \neq z'')$. The main difference between R_{cut} and a boundary formula H_{cut} is as follows. Adding R_{cut} makes redundant the clauses of *both* $EQ(X', X'')$ *and* F_M. On the other hand, Proposition 3 entails that adding H_{cut}, in general, makes *only* clauses of $EQ(X', X'')$ redundant. Intuitively, boundary formula H_{cut} should be easier to compute than interpolant R_{cut}.Consider the following example. Let subcircuits M' and M'' of Fig. 3 be functionally equivalent. From Proposition 4

You must explain atleast informally what is redundancy? Removing these clause does not affect the satisfiability of the formula

Say this after the example after the reader has built some intution.

it follows that formula $EQ(Cut', Cut'')$ specifying the equivalences of the corresponding output variables of M' and M'' is a boundary one. However, whether $EQ(Cut', Cut'')$ is an interpolant R_{cut} above *totally depends* on subcircuit L located above the cut. ~~One still has to do extra work to turn $EQ(Cut', Cut'')$ into an interpolant (by adding more clauses) or to prove that $EQ(Cut', Cut'')$ is already an interpolant.~~

This extra work is superfluous. Consider removing it.

4.3 Size of boundary formulas

In this subsection, we argue that for structurally similar circuits there should exist small boundary formulas. This claim is based on the two observations ~~below~~. The first observation is as follows. Let $\boldsymbol{q}$ be an assignment to the cut of Fig. 3. Assignment $\boldsymbol{q}$ can be represented as $(\boldsymbol{q'},\boldsymbol{q''})$ where $\boldsymbol{q'}$ and $\boldsymbol{q''}$ are ~~output assignments of circuits~~M' and M'' respectively. Definition 1 does not ~~specify~~constrain the value of $H_{cut}(\boldsymbol{q})$ if $\boldsymbol{q}$ cannot be extended to an assignment satisfying $F_{N'} \wedge F_{N''}$. So, if, for instance, output $\boldsymbol{q'}$ cannot be produced for any input of M', the value of $H_{cut}(\boldsymbol{q})$ can be arbitrary. This means that H_{cut} does not have to tell apart assignments to the cut variables that are in the range of M' and M'' from those that are not. 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.

assignments to output of circuits

I do not understand this explaination. An assignment q' is always possible for any input to M'. What you intend to say here is that it cannot be extended to satisfy condition (b) in definition 1.

Define informally the range of a circuit

The second observation is that structurally similarity of N' and N'' implies the existence of many short clauses relating internal points of M' and M'' that can be derived from formula $EQ(X', X'') \wedge F_M$. 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 $\boldsymbol{x'} \neq \boldsymbol{x''}$.

The intuition above is substantiated by Proposition 4. It shows that if M' and M'' are functionally equivalent, there is a ~~very~~ small boundary formula H_{cut} whose size does not depend on individual complexity of M' and M''.

only depends on the cut variables and

5 Computing Boundary Formulas ~~By Induction~~

In this section, we consider computing boundary formulas by cut-wise induction. In Subsection 5.1, we present a method for precise computation of boundary formulas.~~Computing boundary formulas approximately is discussed in Subsection 5.2.~~

Consider rewording this and explain why you are considering this relaxation to “approximate” boundary formulas.

5.1 Precise computation of boundary formulas

The main part of EC by LoR is to compute an output boundary formula $H(z', z'')$. In this subsection, we show how formula H can be computed inductively via a sequence of cuts $Cut_0, \ldots, Cut_m$ in circuits N' and N'' (see Fig. 2). We will assume that $Cut_0 = X' \cup X''$ and $Cut_m = \{z', z''\}$ and cuts do not overlap i.e. $Cut_i \cap Cut_j = \emptyset$ 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. 3). 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. Then H_i is built to satisfy $\exists W_i[H_{i-1} \wedge F_{M_i}] \equiv H_i \wedge \exists W_i[F_{M_i}]$. So H_i makes 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 $H_1, \dots, H_m$ are indeed boundary formulas follows from the proposition below.

This follows from Proposition 2

Proposition 5. *Let $i > 0$, and W_i ~~consist~~ be set of variables of F_{M_i} minus those of Cut_i. Let H_{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 and so H_i is a boundary formula.*

5.2 Computing boundary formulas approximately

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. ~~Arguably, this problem can be successfully solved by improving the PQE-algorithm employed for constructing H_i. (For instance, the algorithm of [11] currently lacks re-using learned information. Fixing this problem should lead to a major performance boost like clause re-using speeds up SAT-solving. A more detailed discussion of this topic is given in Section E of the appendix.)~~In this subsection, we describe an alternative solution to the problem, which is to compute boundary formulas *approximately*. A straightforward way to simplify building boundary formula H_i is to use only a subset $F^*_{M_i}$ of clauses of F_{M_i}. (For instance, one can use formula $F^*_{M_i}$ that consists only of clauses of F_{M_i} specifying the gates between Cut_{i-p} and Cut_i, $p < i$.) The downside of this is that the second condition of Definition 1 may not hold and so EC by LoR becomes *incomplete*. Namely, if $H(b', b'') = 1$ for some assignment $z' = b', z'' = b''$, $b' \neq b''$, the fact that G^{rlx} is satisfiable under $z' = b', z'' = b''$ does not mean that N' and N'' are inequivalent.

“whose” refers to H_i? because size of F_{M_i} also grows. Can you elaborate on this more. A benefit would be you can motivate the relaxation/approximation of the problem

Thus it is imperative that the PQE algorithm is efficient. Though significant improvements have been reported in the literature [], one can argue against the capacity of state-of-the-art PQE algorithms. In Section **??**, we discuss aspect in more detail.

Move the simplification method after you have explained how approximate computation of boundary formulas be used to infer equivalence, the primary problem.

On the other hand, there are at least two reasons why even EC by LoR with approximate computation of boundary formulas ~~can be a great step~~forward with respect to existing tools. The first reason is that approximate boundary formulas can be successfully used to prove *equivalence* of N' and N''. If $H(0,1) = H(1,0) = 0$, circuits N' and N'' are equivalent even if boundary formulas were computed approximately. Importantly, to derive H, EC by LoR can still use cut-wise induction and so go far beyond the capacity of existing fragile induction schemes. This would be the place to explain how to compute approximate boundary formulas.

is a significant improvement in comparison to existing tools for EC.

The second reason is as follows. Suppose $H(b', b'') = 1$ where $b' \neq b''$. To check if N' and N'' can produce outputs b' and b'' (and hence are inequivalent) one can run a general-purpose SAT-solver on formula $G \wedge H_1 \wedge .. \wedge H_m \wedge (z' = b') \wedge (z'' = b'')$. The latter can be much simpler to solve than $G \wedge (z' = b') \wedge (z'' = b'')$ because boundary formulas consist of valuable clauses that are very hard to generate by heuristics of a general-purpose SAT-solver.

Consider: Furthermore, this method is arguably better than general-purpose SAT-solver in proving inequivalence of N’ and N”

6 Algorithm of EC by LoR

In this section, we introduce an algorithm of EC by LoR called *EC_LoR* that is applied to two single-output circuits N' and N''. The pseudo-code of *EC_LoR* is given in Figure 4. *EC_LoR* builds a sequence of boundary formulas $H_0, \dots, H_m$ where H_0 equals $EQ(X', X'')$ and $H_m(z', z'')$ is an output boundary formula. Then, according to Proposition 1, *EC_LoR* checks the satisfiability of formula $H_m \wedge G^{rlx} \wedge (z' \neq z'')$ where $G^{rlx} = F_{N'} \wedge F_{N''}$. In Figure 4, *EC_LoR* computes formulas $H_1, \dots, H_m$ as described in Subsection 5.1 (i.e. precisely).

```
$EC\_LoR(N', N'')$\{
1  $(N', N'') := Bufferize(N', N'')$;
2  $Cut_0 = X' \cup X''$;
3  $Cut_m := \{z', z''\}$;
4  $Cut_1, .., Cut_{m-1} := BldCuts(N', N'')$;
5  $H_0 := EQ(X', X'')$;
− − − − − − − − − − − − − − − − −
6  $for(i := 1; i \leq m; i{+}{+})$ \{
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 ($H_m(0,1) = 1$)
15     if ($Sat(G^{rlx} \wedge \overline{z'} \wedge z'')$) return($No$);
16 if ($H_m(1,0) = 1$)
17     if ($Sat(G^{rlx} \wedge z' \wedge \overline{z''}$)) return($No$);
18 return($Yes$); \}
```

Fig. 4. EC by LoR

EC_LoR consists of three parts separated by the dotted lines in Figure 4. *EC_LoR* starts the first part (lines 1-5) by calling procedure *Bufferize* that eliminates non-local connections of N' and N''. Suppose that the output of a gate of topological level j feeds a gate of a topological level k where $k > j + 1$. (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.) Then this connection is called non-local. 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 ($d > 1$) with a chain of $d-1$ 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. The first part is concluded by setting formula H_0 to $EQ(X', X'')$.

Boundary formulas H_i, $1 \leq i \leq m$ 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 $H_m(z', z'')$ computed in the second part to decide whether N', N'' are equivalent. If $H_m(b', b'') = 1$ 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 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 computation takes place 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 $\exists W_i[H_{i-1} \wedge F_{M_i} \wedge H_i]$ (the cut termination condition). *Redund* stops as soon as it finds out that H_{i-1} is not redundant yet. *Redund* 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.

7 ~~Some~~ Experimental Evidence In Favor Of EC By LoR

~~Although this is a theoretical paper, in this section we give some experimental support of our approach~~. ~~In experiments,~~ We used the implementation of the PQE-algorithm introduced in [11]. It is not clear if PQE-14 is good enough to be used in a full-fledged equivalence checker. Most likely, some improvements need to be made first (see Section E for a more detailed discussion). On the other hand, PQE-14 is sufficiently efficient to make a few important points. In all experiments, we considered checking self-equivalence of a circuit. Namely, we checked for equivalence two identical copies N' and N'' of a circuit computing a median output of an n-bit multiplier.

In previous sections, we have established a framework for effectively reasoning about EC using LoR. In this section, we give preliminary experimental results to illustrate the benefits of the methodology

Discuss this at the end after you have presented the result with PQE-14

The experiment of Subsection 7.1 shows that computing the range of a cut is much more expensive than building a boundary formula for this cut. This also contrasts complete quantifier elimination (used to compute cut range) with PQE. In Subsection 7.2, we experiment with EC by SAT-solving. Our objective here is to give ~~an~~ example of EC formulas where SAT-solving by conflict-driven learning fails to find a short proof. This suggests that finding small interpolants for those formulas is unlikely. In Subsection 7.3, we experiment with *EC_LoR* described in Section 6. One of the reasons for robustness of induction proofs generated in EC by LoR is in checking the cut termination condition (i.e. whether adding clauses of boundary formula H_i makes those of H_{j-1} redundant). ~~We show~~that the complexity of checking the cut termination condition by PQE-14 when boundary formulas are computed approximately is ~~quite~~ acceptable for a class of formulas.

This is again to early to discuss the experiment. Probably it will be better to clearly lay out on what is the objective of the experiments and on what parameters are you evaluating the experiments.

Consider: Our objective is to compare EC by LoR and EC by SAT-solving for formulaes where conflict-driven learning fails to find a short proof.

The experiments show that

Here is where you can add: We further discuss the limitations of the current algorithm and possible improvements to it.

7.1 Range computation versus building boundary formulas

In this subsection, we compare computation of a boundary formula H_{cut} and that of a cut range. As a cut of circuits N', N'' we picked the set of variables specifying the outputs of gates of the first topological level (i.e. those fed by input variables of N' and N''). We assume here that non-local connections in N' and N'' were replaced by chains of buffers as described in Section 6. Computing cut range comes down to performing quantifier elimination for formula $\exists W[EQ \wedge F_M]$. Here $W = X' \cup X''$ and formula F_M specifies the gates of N', N'' of the first topological level. Namely, one needs to build a formula R_{cut} (specifying the cut range) such that $R_{cut} \equiv \exists W[EQ \wedge F_M]$. Computing a boundary formula H_{cut} comes down to solving the PQE problem i.e. finding a formula H_{cut} such that $\exists W[EQ \wedge F_M] \equiv H_{cut} \wedge \exists W[F_M]$.

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

#bits	#quan. vars	#free vars	cut range (QE)		bound. form. (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**

The results of the experiment are given in Table 1. (Abbreviation QE stands for Quantifier Elimination.) The size of the multiplier from which circuits N' and N'' were generated is given in the first column. The next two columns show the number of quantified and free variables in $\exists W[EQ \wedge F_M]$. To compute formula R_{cut} above we used the quantifier elimination program described in [10] kindly provided to us by the authors. 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 by *EC_LoR* in the experiments of Subsection 7.3). When computing range R_{cut} and boundary formula H_{cut} we recorded the size of the result (in the number of clauses) and the run time. As Table 1 shows, formula H_{cut} is much smaller than R_{cut} and takes much less time to compute.

7.2 EC by SAT-solving

The results of SAT-solving of EC formulas are shown in Table 2. We used the well-known SAT-solver Minisat [7, 24] (version 2.0) and Lingeling [23], a winner of the SAT-14 competition. In the experiment, we tested the satisfiability of formula $EQ(X', X'') \wedge F_{N'} \wedge F_{N''} \wedge (z' \neq z'')$ where $F_{N'}$,$F_{N''}$ specify N' and N'' respectively. The first column of Table 2 shows the size of the multiplier from which the circuit computing a median bit was extracted. The next two columns give the size of the formula. The last two columns give the run time in seconds. Table 2 shows that run times grow very fast with n. Either SAT-solver failed to solve formula G for $n = 16$ in 6 hours.

Table 2. *Checking self-equivalence of multipliers by SAT. Time limit = 6 hours*

#bits	#vars	#clauses	minisat (s.)	lingeling (s.)
10	705	2,665	11	1.6
11	863	3,281	67	15
12	1,037	3,961	176	42
13	1,227	4,705	1,285	71
14	1,433	5,513	5,085	1,560
15	1,655	6,385	18,113	5,127
16	1,893	7,321	$*$	$*$

Table 2 suggests that for circuits with a complex topology, a general-purpose SAT-solver cannot find a short proof of equivalence. (There is always a resolution proof of self-equivalence that is *linear* in circuit size [8]). The reason is that the only short equivalence proofs for such circuits are those by induction and such proofs are impossible to find by heuristics of a general-purpose SAT-solver. This implies that finding a small interpolant for formulas describing EC of such circuits is also problematic. Note that self-equivalence formulas can be easily solved by a SAT-solver that employs some local reasoning (like trying to find easy-to-prove equivalence relations).

However, this works only for circuits that are almost identical and so have a lot of equivalent variables.

7.3 EC by LoR: a low price for robustness

This subsection describes an experiment with EC_LoR introduced in Section 6. The objective of this experiment is as follows. Robustness of EC by LoR is achieved by building boundary formulas. This makes even the case of checking self-equivalence a non-trivial problem. The reason is that one needs to call a PQE-solver to check if H_i is finished (*the cut termination condition*). So we want to check if the robustness of EC by LoR comes at a reasonable price.

Cuts $Cut_0, \dots, Cut_m$ used by EC_LoR were generated according to topological levels. That is Cut_i consisted of the variables specifying the output of gates of i-th topological level. Since EC_LoR employs bufferization to eliminate non-local connections (see Section 6), $Cut_i \cap Cut_j = \emptyset$ if $i \neq j$. In the experiment, we used two versions of EC_LoR that we refer to as EC_LoR_1 and EC_LoR_2. Both versions were slightly different from the one described in Fig. 4. The main change was that boundary formulas were computed *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. 4) only a subset of clauses of F_{M_i} was used to make the check simpler. Nevertheless, both versions were able to compute an output boundary formula $H(z', z'')$ proving that N' and N'' were equivalent. The difference between EC_LoR_1 and EC_LoR_2 was as follows. EC_LoR_1 ran a cut termination check every time formula H_i was updated (see the *while* loop of Fig. 4, lines 10-13). In EC_LoR_2 , the number of cut termination checks was reduced. Namely, derivation of clauses of H_i was modified so that EC_LoR_2 did not run a cut termination check if some cut variable was not present in clauses of H_i. The intuition here was that in that case H_i was still under-constrained. More details are given in Section D of the appendix.

Table 3. *Checking self-equivalence of multipliers by EC_LoR. Time limit 1 hour*

#bits	#vars	#clauses	#cuts	EC_LoR_1 (s.)	EC_LoR_2 (s.)
10	2,766	6,744	36	8.6	0.6
12	4,632	11,100	44	27	1.3
14	7,162	16,912	52	73	2.5
16	10,452	24,372	60	168	4.5
24	33,132	74,532	90	2,492	27
32	75,652	166,420	124	$*$	98

The results of EC_LoR_1 and EC_LoR_2 are shown in Table 3. The first column gives the size of the multipliers. The next two columns show the size of the formulas describing self-equivalence. The fourth column gives the number of cuts (i.e. topological levels) in circuits N' and N'' used by EC_LoR_1 and EC_LoR_2. The results of Table 3 show that checking cut termination conditions too often is expensive even when computing boundary formulas approximately. (This conclusion obviously applies only to the current PQE algorithms. Their performance can get much better in the near future.) However, if one makes an effort to reduce the number of cut termination checks, the overhead is not too high. For instance, EC_LoR_2 proved self-equivalence of the circuit computing a median bit of a 16-bit multiplier in 4.5 seconds.

8 Some Background

The EC methods can be roughly classified into two groups. Methods of the first group do not assume that circuits N',N'' to be checked for equivalence are structurally similar. Checking if N' and N'' have identical BDDs [5] 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 [18, 21, 7, 3, 13]. 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',N'' that do not affect their range [17]. 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 [14, 15, 20]. However, in some approaches the derived relations are implications (between internal points of N' and N'') [16] or equivalences modulo observability [4]. 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. On the other hand, it is tempting to look for EC methods that can handle a much more general notion of structural similarity. Namely, two circuits are considered structurally similar if there is there is a short induction proof of their equivalence. We are unaware of an approach that could handle such a broad notion of structural similarity.

9 Taking Logic Relaxation Beyond Equivalence Checking

LoR is a general technique that can be applied beyond EC. As an example of that, in Subsection 9.1, we relate LoR to interpolation. First we show that an interpolant can be viewed as a special case of a boundary formula. Then we use boundary formulas to introduce interpolants of broken implications. Such interpolants can be very helpful in generating short versions of counterexamples. In Subsection 9.2, we make a point that a relaxed formula should not be viewed as an abstraction of the original formula.

9.1 Boundary formulas and interpolation

Let formula G denote $A(X, Y) \wedge B(Y, Z)$ where X, Y, Z are non-overlapping sets of variables. Let a relaxed formula G^{rlx} be obtained from G by dropping the clauses of A i.e. $G^{rlx} = B$. Let $\exists W[A \wedge B] \equiv H \wedge \exists W[B]$ hold for a formula $H(Y)$ where $W = X \cup Z$. Similarly to the proof of Proposition 2, one can show that H is a boundary formula for relaxation G_{rlx} in terms of Y (see Definition 1). That is a) $G \rightarrow H$ and b) $H(\boldsymbol{y}) = 0$ for every assignment $\boldsymbol{y}$ to Y that can be extended to an assignment satisfying G^{rlx} but not G.

Note that $A \wedge B \equiv 0$ implies $H \wedge B \equiv 0$. Hence, if, in addition to $G \rightarrow H$, $A \rightarrow H$ holds, H is an interpolant for implication $A \rightarrow \overline{B}$ (see Proposition 6 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 7 of the appendix). So a counterexample for $H \rightarrow \overline{B}$ is a short version of that for $A \rightarrow \overline{B}$.

9.2 Relaxation $\neq$ abstraction

In this paper, we considered a case where formula G^{rlx} is obtained from G by dropping some clauses and so $G \rightarrow G^{rlx}$ holds. So one may decide that G^{rlx} is an abstraction of G. In reality, one does not need $G \rightarrow G^{rlx}$ to apply LoR. Informally, G^{rlx} has to meet the three conditions below. First, every assignment satisfying G must have at least one corresponding assignment satisfying G^{rlx}. Second, finding a satisfying assignment for G^{rlx} should be easier than for G. Third, one should be able to apply PQE to compute the difference between G and G^{rlx} in terms of a (small) set of variables. In [22], we describe an instance of LoR where a set V of new variables is introduced in G^{rlx} and $G \rightarrow \exists V[E \wedge G^{rlx}]$ holds where E is some formula. On the one hand, the implication above guarantees that every assignment satisfying G is represented in G^{rlx}. On the other hand, $G \rightarrow G^{rlx}$, in general, does not hold.

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.

References

1. H.R. Andersen and H. Hulgaard. Boolean expression diagrams. Inf. Comput., 179(2):194–212, 2002.
2. C. Berman and L. Trevillyan. Functional comparison of logic designs for vlsi chips. In ICCAD, pages 456–459, 1989.
3. A. Biere. Picosat essentials. JSAT, 4(2-4):75–97, 2008.
4. D. Brand. Verification of large synthesized designs. In ICCAD-93, pages 534–537, 1993.

5. R. Bryant. Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers, C-35(8):677–691, August 1986.
6. 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.
7. N. Eén and N. Sörensson. An extensible sat-solver. In SAT, pages 502–518, Santa Margherita Ligure, Italy, 2003.
8. E. Goldberg. Boundary points and resolution. In Proc. of SAT, pages 147–160. Springer-Verlag, 2009.
9. E. Goldberg and P. Manolios. Quantifier elimination by dependency sequents. In FMCAD-12, pages 34–44, 2012.
10. E. Goldberg and P. Manolios. Quantifier elimination via clause redundancy. In FMCAD-13, pages 85–92, 2013.
11. E. Goldberg and P. Manolios. Partial quantifier elimination. In Proc. of HVC-14, pages 148–164. Springer-Verlag, 2014.
12. P. Jancík, J. Kofron, S.F. Rollini, and N. Sharygina. On interpolants and variable assignments. In FMCAD-14, pages 123–130, 2014.
13. H. Jin and F. Somenzi. Circus: A hybrid satisfiability solver. In SAT-2004, 2004.
14. A. Kuehlmann and F. Krohm. Equivalence Checking Using Cuts And Heaps. DAC, pages 263–268, 1997.
15. 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.
16. W. Kunz. Hannibal: An efficient tool for logic verification based on recursive learning. In ICCAD-93, pages 538–543, 1993.
17. H. Kwak, I. MoonJames, H. Kukula, and T. Shiple. Combinational equivalence checking through function transformation. In ICCAD-02, pages 526–533, 2002.
18. J. Marques-Silva and K. Sakallah. Grasp – a new search algorithm for satisfiability. In ICCAD-96, pages 220–227, 1996.
19. K. L. Mcmillan. Interpolation and sat-based model checking. In CAV-03, pages 1–13. Springer, 2003.
20. A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een. Improvements to combinational equivalence checking. In ICCAD-06, pages 836–843, 2006.
21. 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.
22. Title and authors are omitted. A technical report covering the content of this paper.
23. Lingeling. http://fmv.jku.at/lingeling/.
24. Minisat2.0. http://minisat.se/MiniSat.html.

Appendix

A Proofs Of Propositions

Proposition 1. *Let $H(z',z'')$ denote 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 $G \wedge (z' \neq z'')$ is equisatisfiable with $H \wedge G^{rlx} \wedge (z' \neq z'')$.*

Proof. A proof of this proposition follows from Propositions 2 and 3 below. Proposition 2 implies that $H(z',z'')$ is a boundary formula. From Proposition 3 it follows that $G \wedge (z' \neq z'')$ is equisatisfiable with $H \wedge G^{rlx} \wedge (z' \neq z'')$.

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 $EQ \wedge F_M \rightarrow H_{cut}$. 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 $G \wedge (z' \neq z'')$ is equisatisfiable with $H_{cut} \wedge G^{rlx} \wedge (z' \neq z'')$.*

Proof. Let us show that the satisfiability of the left formula i.e. $G \wedge (z' \neq z'')$ implies that of the right formula i.e. $H_{cut} \wedge G^{rlx} \wedge (z' \neq z'')$ and vice versa.

Left sat. → Right sat. Let $\boldsymbol{p}$ be an assignment satisfying $G \wedge (z' \neq z'')$. 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 $H_{cut} \wedge G^{rlx} \wedge (z' \neq z'')$.

Right sat. → Left sat. Let $\boldsymbol{p}$ be an assignment satisfying $H_{cut} \wedge G^{rlx} \wedge (z' \neq z'')$. 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 $(z' \neq z'')$ and hence formula $G \wedge (z' \neq z'')$.

Proposition 4. *Let Cut' and Cut'' be subsets of Cut specifying the outputs of circuits M' and M'' of Fig. 3 respectively. Let M' and M'' be functionally equivalent to each other. Let H_{cut} be equal to formula $EQ(Cut', Cut'')$ stating equivalence of corresponding output variables of M' and M''. Then H_{cut} is a boundary formula.*

Proof. From Definition 1 it follows that one needs to prove that

a) $G \rightarrow H_{cut}$
b) If a cut assignment $\boldsymbol{q}$ can be extended to an assignment satisfying G^{rlx} but not G, then $H_{cut}(\boldsymbol{q})= 0$

Condition a) holds because M' and M'' are functionally equivalent and hence formula $EQ(Cut', Cut'')$ is implied by $EQ(X', X'') \wedge F_M$. Let us show that condition b) is true as well. Let $\boldsymbol{q}$ be a cut assignment that can be extended to an assignment satisfying G^{rlx} but not G. Assignment $\boldsymbol{q}$ can be represented as $(\boldsymbol{q'}$,$\boldsymbol{q''})$ where $\boldsymbol{q'}$ and $\boldsymbol{q''}$ are assignments to Cut' and Cut'' respectively. Let us show that $\boldsymbol{q'} \neq \boldsymbol{q''}$ and so $H_{cut}(\boldsymbol{q}) = EQ(\boldsymbol{q'}, \boldsymbol{q''}) = 0$.

Assume the contrary i.e. $\boldsymbol{q'}$=$\boldsymbol{q''}$. Let $\boldsymbol{p} = (\boldsymbol{x'}, \boldsymbol{x''}, \boldsymbol{y'}, \boldsymbol{y''}, z', z'')$ be an assignment satisfying G_{rlx} obtained by extending $\boldsymbol{q}$. Assignment $\boldsymbol{p}$ specifies the execution trace produced when inputs $\boldsymbol{x'}$,$\boldsymbol{x''}$ are applied to N' and N'' respectively. Let $\boldsymbol{p^*}$ be the assignment specifying the execution trace when inputs $\boldsymbol{x'}$,$\boldsymbol{x'}$ are applied to N' and N''. Since M' and M'' are functionally equivalent, such input assignment will produce cut assignment $(\boldsymbol{q'}$,$\boldsymbol{q'})$ i.e. the same as assignment as $\boldsymbol{q}$. (Recall that $\boldsymbol{q}$=$(\boldsymbol{q'}$,$\boldsymbol{q''})$ and by assumption, $\boldsymbol{q'}$=$\boldsymbol{q''}$.) So assignment $\boldsymbol{p^*}$ can also be obtained by extending $\boldsymbol{q}$. Since assignments to X' and X'' are identical in $\boldsymbol{p^*}$, the latter satisfies G and we have a contradiction.

Proposition 5. *Let W_i consist of the variables of F_{M_i} minus those of Cut_i. Let H_{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 and so H_i is a boundary formula.*

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 $A(X, Y)$ 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 $W = X \cup Y$.*

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 7. *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 $A \wedge B$. Since ($\boldsymbol{x}$,$\boldsymbol{y}$) satisfies A and ($\boldsymbol{y}$,$\boldsymbol{z}$) does B, then ($\boldsymbol{x}$,$\boldsymbol{y}$,$\boldsymbol{z}$) satisfies $A \wedge B$.

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 an algorithm for PQE introduced in [11] in 2014. We will refer to this algorithm as PQE-14. Let $A(V,W), B(V,W)$ be Boolean formulas and V and 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, PQE-14 uses 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 [9, 10]. 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 $\exists W[A \wedge B]$.

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.

PQE solving is work in progress and so is still lacking some important techniques like D-sequent re-using. In current algorithms, 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 [11].

C Generation Of Cuts That Do Not Overlap

An important part of *EC_LoR* described in Section 6 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 $M(X, Y, Z)$ 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 to an output of M going through 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 *TopLvl*(g). It can be computed recursively as follows. If g is an input, then *TopLvl*$(g) = 0$. Otherwise, *TopLvl*(g) is equal to the maximum topological level among the gates feeding g plus 1.

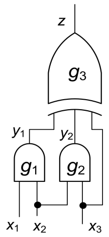

Fig. 5. A circuit

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. 5 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 $g \in S$ it is true that $S \setminus \{g\}$ 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 *TopLvl*(v)) is equal to *TopLvl*(g).

If gate g_i of M feeds gate g_j and *TopLvl*$(g_j) >$ *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 5. The input gate x_3 feeds gates g_2 and g_3. Since *TopLvl*(x_3)=0 and *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) *TopLvl*$(g_j) >$ *TopLvl*(g_i)+1. By inserting *TopLvl*$(g_j) -$ *TopLvl*$(g_i) - 1$ buffers between g_i to g_j, this non-local connection is replaced with *TopLvl*$(g_j) -$ *TopLvl*(g_i) local connections.

D *EC_LoR*$_1$ and *EC_LoR*$_2$ Used In Experiments

In the experiments of Subsection 7.3, we used two versions of *EC_LoR* that were modified in comparison to the description given in Fig. 4. We will refer to these versions as *EC_LoR*$_1$ and *EC_LoR*$_2$. In this section, we describe *EC_LoR*$_1$ and *EC_LoR*$_2$ in more detail. The heuristics we describe below seem to be excessive taking into account that we used *EC_LoR*$_1$ and *EC_LoR*$_2$ only for checking self-equivalence. Our motivation here was to give heuristics that a) work in the trivial case of self-equivalence; b) may come useful later when working on EC of circuits that have few or no functionally equivalent internal points.

Boundary formula H_i was computed in *EC_LoR*$_1$ as follows. The code of Fig 4 just adds to H_i clause C returned as evidence that computation of H_i is not completed yet (line 11). *EC_LoR*$_1$ used clause C just as a pointer to the part of Cut_i that was still under-constrained. Then *EC_LoR*$_1$ generated a set of (shorter) clauses as follows. First, EC_LoR_1 picked a gate g' of N' whose output variable was in clause C. Then using the input variables of g' and boundary formula H_{i-1} derived earlier EC_LoR_1 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 (instead of clause C). 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_1 was that boundary formulas were computed approximately. In line 11 of Fig 4, 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_1 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.

Boundary formula H_i was computed in EC_LoR_2 as follows. If there was a variable specifying the output of a cut gate g' that was still unconstrained in H_i, EC_LoR_2 called the procedure above that generated short clauses relating the output variable of g' and those of its relatives from N''. This way, EC_LoR_2 avoided running a cut termination check before every variable of i-th cut was constrained in H_i. Similarly to EC_LoR_1, boundary formulas were computed in EC_LoR_2 approximately using the same subset of clauses of F_{M_i} as EC_LoR_1 did.

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 5.1, boundary formula H_i is obtained by taking H_{i-1} out of the scope of quantifiers in formula $\exists W_i[H_{i-1} \wedge F_{M_i}]$ whose size grows with 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. PQE-14 (see Section B) does not re-use D-sequents, which makes the precise computation of boundary formulas for large values of i infeasible. In this section, we argue that once D-sequent re-using is implemented, efficient computation of boundary formulas becomes quite possible.

Consider the problem above 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 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 n 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 n. Adding this clause to $\exists W_i[H_{i-1} \wedge F_{M_i}]$ makes C redundant in node n. Otherwise, clauses with literal $\overline{v}$ are proved redundant and C is redundant in node n because it cannot be resolved on v in the current subspace.

To prove that a clause A 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 A on one of its variables. 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 needed to prove only locally is discarded after an obligation to prove redundancy of this clause is canceled. This leads to very poor scalability of PQE-14 because one has to reproduce the same D-sequent in many nodes of the search tree. 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 will lead to drastic reduction of 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. Note, however, that re-using D-sequents is not as simple as re-using clauses in SAT-solving because the former requires more sophisticated bookkeeping [11].