Partial Quantifier Elimination

Eugene Goldberg, Panagiotis Manolios

Northeastern University, USA {eigold,pete}@ccs.neu.edu

Abstract. We consider the problem of Partial Quantifier Elimination (PQE). Given formula $\exists X[F(X,Y) \wedge G(X,Y)]$, where F, G are in conjunctive normal form, the PQE problem is to find a formula $F^*(Y)$ such that $F^* \wedge \exists X[G] \equiv \exists X[F \wedge G]$. We solve the PQE problem by generating and adding to F clauses over the free variables that make the clauses of F with quantified variables *redundant* in $\exists X[F \wedge G]$. The traditional Quantifier Elimination problem (QE) can be viewed as a degenerate case of PQE where G is empty so *all* clauses of the input formula with quantified variables need to be made redundant. The importance of PQE is threefold. First, in non-degenerate cases, PQE can be solved more efficiently than QE. Second, many problems are more naturally formulated in terms of PQE rather than QE. Third, an efficient PQE-algorithm will enable new methods of model checking and SAT-solving. We describe a PQE algorithm based on the machinery of dependency sequents and give experimental results showing the promise of PQE.

1 Introduction

The elimination of existential quantifiers is an important problem arising in many practical applications. We will refer to this problem as the Quantifier Elimination problem, or QE. Given a formula $\exists X[G]$ where G is a propositional formula, the **QE problem** is to find a quantifier free formula G^* such that $G^* \equiv \exists X[G]$. In this paper, we assume that all propositional formulas are represented in conjunctive normal form (CNF).

Unfortunately, the efficiency of current QE algorithms still leaves much to be desired. This is why many successful theorem proving methods such as interpolation and IC3 avoid QE and use SAT-based approaches instead. The lack of efficient QE solvers can be addressed by looking for variations of QE that are easier to solve. In this paper, we consider such a variation called Partial QE (PQE). Given formula $\exists X[F(X,Y) \wedge G(X,Y)]$, the **PQE problem** is to find a quantifier free formula $F^*(Y)$ such that $F^* \wedge \exists X[G] \equiv \exists X[F \wedge G]$. We will say that F^* is obtained by **taking F out of the scope of the quantifiers**. QE can be viewed as a degenerate case of PQE where G is empty and so the entire formula is taken out of the scope of quantifiers. In the following exposition, when contrasting PQE and QE we mean non-degenerate instances of PQE.

An important advantage of PQE over QE is that the former is "structurally sound". A prototypical QE problem is to compute the range of a circuit. Let formula $G(X,Y,Z)$ specify a combinational circuit N where X, Y, Z are sets of input, internal and output variables respectively. Then a formula $G^*(Z)$ such that

$G^* \equiv \exists W[G]$ where $W = X \cup Y$ specifies the range of N. The very definition of QE forces one to build formula G^* by *destroying* the structure of G inherited from circuit N. A prototypical PQE problem [8] is to compute reduction of range of N when one excludes the inputs of N falsifying a formula $F(X)$. Let $F^*(Z)$ be a formula such that $F^* \wedge \exists W[G] \equiv \exists W[F \wedge G]$. One can view the assignments falsifying F^* as the outputs excluded from the range of N after the inputs falsifying F are removed from consideration. So F^* describes the reduction in the range of N caused by constraining its inputs by F. Note that computation of formula F^* leaves formula G *intact*. Moreover, an intelligent PQE-solver will *exploit* the structure of G to find F^* more efficiently.

Besides our interest in PQE as being "structurally sound", our motivation for studying PQE is twofold. First, in addition to traditional QE applications, PQE brings in many new ones. In Subsection 2.1, we show that PQE can be used to compute states reachable *only* from a specified set of states, which enables a new class of model checkers. Subsection 2.3 gives an example of using PQE for SAT-solving. Second, in some cases, even if the original problem is formulated in terms of QE, it can sometimes be reduced to PQE. In Subsection 2.2, we show that this is the case for pre-image computation in backward model checking.

The relation between efficiency of solving PQE and QE can be better understood in terms of clause redundancy [7]. The PQE problem of taking F out of the scope of quantifiers in $\exists X[F \wedge G]$ reduces to finding a set of clauses F^* that makes all X-clauses of F redundant in $\exists X[F \wedge G]$. (An ***X*-clause** is a clause that contains a variable from X.) Then every clause of F can be either dropped as redundant or removed from the scope of the quantifiers as it contains only free variables.

One can view the process of building F^* as follows. X-clauses of F are made redundant in $\exists X[F \wedge G]$ by adding to F resolvent clauses derived from $F \wedge G$. Notice that no clause obtained by resolving *only* clauses of G needs to be made redundant. Adding resolvents to F goes on until all X-clauses of the current formula F are redundant. At this point, the X-clauses of F can be dropped and the remaining clauses of F form F^*. If F is much smaller than G, the process of solving PQE looks like wave propagation where F is the original "perturbation" and G is the "media" where this wave propagates. Such propagation can be efficient even if G is large. By contrast, when solving the QE problem for $\exists X[F \wedge G]$ one needs to make redundant the X-clauses of both F and G and *all* resolvent X-clauses including the ones obtained by resolving only clauses of G.

In this paper, we describe a PQE-algorithm called *DS-PQE* that is based on the machinery of D-Sequents [6, 7]. One needs this machinery for PQE for the same reason as for QE [6]. Every clause of $F^*(Y)$ can be obtained by resolving clauses of $F \wedge G$. However, the number of clauses that are implied by $F \wedge G$ and depend only on Y is, in general, exponential in $|Y|$. So it is crucial to identify the moment when the set of clauses derived so far that depend only on Y is sufficient to make the X-clauses of F redundant in $\exists X[F \wedge G]$. The machinery of D-sequents is used by *DS-PQE* to perform such identification.

The following exposition is structured as follows. In Section 2, we discuss some problems that can benefit from an efficient PQE-algorithm. A run of *DS-PQE* on

a simple formula is described in Section 3. Sections 4 and 5 give basic definitions and recall the notion of D-Sequents. In Section 6, algorithm *DS-PQE* is described. Some background is given in Section 7. In Section 8, we experimentally compare *DS-PQE* with our QE algorithm from [7] in the context of model checking. We make conclusions in Section 9.

2 Some Applications of PQE

In this section, we describe some applications where using an efficient PQE solver can be very beneficial. More applications of PQE can be found in [8–10]. Section 2.1 shows that PQE can be used to compute the set of states reachable *only* from a specified set of states, which enables a new type of model checkers [8]. In Subsection 2.2, we describe application of PQE to the traditional method of backward model checking. Application of PQE to SAT-solving is presented in Subsection 2.3.

2.1 Enabling a new type of model checkers

The basic operation of forward model checking is to compute the set of states reachable from another set of states or to find its over-approximation. In this subsection, we show that one can use PQE to compute the set of states reachable *only* from a specified set of states. This enables a new type of model checkers [8] that iteratively eliminate traces of reachable states. This elimination goes on until a counterexample is found or the set of possible behaviors is reduced to one trace consisting of good states. These new model checkers have a great potential in verification of sequential circuits e.g. they can be used to find very deep bugs.

Let $T(S, S')$ be a transition relation where S and S' specify the current and next state variables respectively. We will refer to complete assignments $\boldsymbol{s}$ and $\boldsymbol{s'}$ to variables S and S' as present and next states respectively. Let $C^{\boldsymbol{s}}$ be the longest clause of variables of S that is falsified by $\boldsymbol{s}$. The set of states reachable from $\boldsymbol{s}$ in one transition is specified by formula $R^{\boldsymbol{s}}(S')$ logically equivalent to $\exists S[\overline{C^{\boldsymbol{s}}} \wedge T]$. That is for every state $\boldsymbol{s'}$ *satisfying* $R^{\boldsymbol{s}}$ there is a transition from state $\boldsymbol{s}$.

Let formula $H(S)$ specify a set of states. Let $\boldsymbol{s}$ be one of states specified by H i.e. $H(\boldsymbol{s}) = 1$. Now we show how to compute the *subset* of states specified by $R^{\boldsymbol{s}}$ that consists of states reachable in one transition *only* from $\boldsymbol{s}$. More precisely, we want to exclude from $R^{\boldsymbol{s}}$ every state that is reachable in one transition from a state $\boldsymbol{r}$ satisfying H and different from $\boldsymbol{s}$. Let $Q^{\boldsymbol{s}}(S')$ be a CNF formula such that $Q^{\boldsymbol{s}} \wedge \exists S[H \wedge T] \equiv \exists S[C^{\boldsymbol{s}} \wedge H \wedge T]$. It is not hard to show that $Q^{\boldsymbol{s}}(\boldsymbol{s'}) = 0$ iff a) $\boldsymbol{s'}$ is reachable in one transition only from state $\boldsymbol{s}$ or b) $\boldsymbol{s'}$ is not reachable from a state specified by H in one transition. The states of item b) are just "noise" i.e. $Q^{\boldsymbol{s}}$ remains a solution to the PQE problem above even if it is not falsified by any of such states. So, complete assignments *falsifying* $Q^{\boldsymbol{s}}$ specify the states reachable only from $\boldsymbol{s}$ modulo some noise. The set of states reachable only from $\boldsymbol{s}$ can be very *small* even when the set of states reachable from $\boldsymbol{s}$ is *huge*. This important fact is what enables the new type of model checkers mentioned above.

2.2 Computing pre-image in backward model checking

Let formula $F(S')$ specify a set of next-states and $H(S)$ specify the pre-image of $F(S')$. That is, a present state $\boldsymbol{s}$ satisfies H iff there exists a next state $\boldsymbol{s'}$ such that $F(\boldsymbol{s'}) \wedge T(\boldsymbol{s}, \boldsymbol{s'}) = 1$. Here T is a transition relation.

Finding H comes down to building a formula logically equivalent to $\exists S'[F \wedge T]$ i.e. reduces to QE. However, one can construct the pre-image of F by PQE as follows. Let F^* be a formula such that $F^* \wedge \exists S'[T] \equiv \exists S'[F \wedge T]$ *i.e.*, F^* is a solution to the PQE problem. Notice that $\exists S'[T] \equiv 1$ and hence can be dropped. Indeed, for every present state $\boldsymbol{s}$ there always exists some next state $\boldsymbol{s'}$ such that $T(\boldsymbol{s}, \boldsymbol{s'}) = 1$. So $F^* \equiv \exists S'[F \wedge T]$ and therefore F^* specifies the pre-image of F. In other words, here QE reduces to PQE.

2.3 SAT-solving by PQE

In this subsection, we give a method of using PQE for SAT-solving. Other methods of applying PQE to SAT-solving can be found in [9, 10].

Testing the satisfiability of a CNF formula $G(X)$ is equivalent to checking if formula $\exists X[G]$ is true. The latter problem can be viewed as a special instance of QE where *all* variables are quantified. In this case, every non-empty clause of G is an X-clause and needs to be proved redundant to solve the QE problem. If the clauses of G are proved redundant in $\exists X[G]$ without derivation of an empty clause, then G is satisfiable. Otherwise, it is unsatisfiable.

Let $\boldsymbol{x}$ be a complete assignment to variables of X that falsifies G. Let F be the set of clauses of G that are falsified by $\boldsymbol{x}$. Formula $\exists X[G]$ can be represented as $\exists X[F \wedge G']$ where $G' = G \setminus F$. Let us consider the PQE problem of finding formula F^* such that $F^* \wedge \exists X[G'] \equiv \exists X[F \wedge G']$. Since $\exists X[F \wedge G']$ has no free variables, F^* is a constant. If $F^* \equiv 1$, then the clauses of F are redundant in $\exists X[F \wedge G']$. In other words, G is satisfiable iff G' is. Since $\boldsymbol{x}$ satisfies G', the original formula G is *satisfiable* as well. If $F^* \equiv 0$, then, to take F out of the scope of quantifiers, one needs to derive an empty clause from $F \wedge G'$ i.e. from G. In this case, G is obviously *unsatisfiable*. So, to check the satisfiability of G, PQE needs to prove *only* redundancy of clauses of F as opposed to proving redundancy of *all* clauses of G in QE.

The PQE algorithm we present in this paper is not powerful enough to compete with SAT-solvers yet. (One of the problems here is that D-sequents are not re-used. A brief discussion of this topic is given in Section 8). However, this may change soon.

3 Example

In this section, we describe a run of a PQE algorithm called *DS-PQE* that is described in Section 6. *DS-PQE* is based on the machinery of Dependency sequents. The latter will be formally defined in Section 5. Recall that an X-clause is a clause that contains at least one variable from a set X of Boolean variables.

Let $F = C_1 \wedge C_2$ where $C_1 = y \vee x_1$, $C_2 = \overline{y} \vee x_3$ Let $G = C_3 \wedge C_4 \wedge C_5 \wedge C_6$ where $C_3 = \overline{x}_1 \vee x_2$, $C_4 = \overline{x}_1 \vee \overline{x}_2$, $C_5 = \overline{x}_3 \vee x_4$, $C_6 = y \vee \overline{x}_4$. Let $X = \{x_1, x_2, x_3, x_4\}$ be the set of variables quantified in formula $\exists X[F \wedge G]$. So y is the only free variable of $\exists X[F \wedge G]$.

Problem formulation. Suppose one needs to solve the PQE problem of taking F out of the scope of the quantifiers in $\exists X[F \wedge G]$. That is one needs to find $F^*(y)$ such that $F^* \wedge \exists X[G] \equiv \exists X[F \wedge G]$. Below, we describe a run of *DS-PQE* when solving this problem.

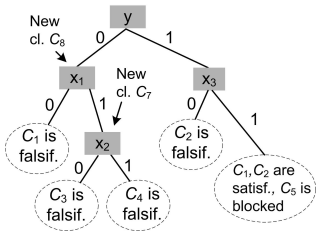

Fig. 1. The search tree built by *DS-PQE*

Search tree. DS-PQE is a branching algorithm. It first proves redundancy of X-clauses of F in subspaces and then merges results of different branches. When *DS-PQE* returns to the root of the search tree, all the X-clauses of F are proved redundant in $\exists X[F \wedge G]$. The search tree built by *DS-PQE* is given in Figure 1. It also shows the nodes where new clauses C_7 and C_8 were derived. *DS-PQE* assigns free variables *before* quantified. So, variable y is assigned first. At every node of the search tree specified by assignment $\boldsymbol{q}$, *DS-PQE* maintains a set of clauses denoted as $PR(\boldsymbol{q})$. Here PR stands for "clauses to Prove Redundant". We will refer to a clause of $PR(\boldsymbol{q})$ as a **PR-clause**. Adding a clause to $PR(\boldsymbol{q})$ is an *obligation* to prove redundancy of this clause in subspace $\boldsymbol{q}$. $PR(\boldsymbol{q})$ includes all X-clauses of F plus some X-clauses of G. The latter are proved redundant to make proving redundancy of X-clauses of F easier. Sets $PR(\boldsymbol{q})$ are shown in Figure 3. For every non-leaf node of the search tree, two sets of PR-clauses are shown. The set on the left side (respectively right side) of node $\boldsymbol{q}$ gives $PR(\boldsymbol{q})$ when visiting node $\boldsymbol{q}$ for the first time (respectively when backtracking to the right branch of node $\boldsymbol{q}$).

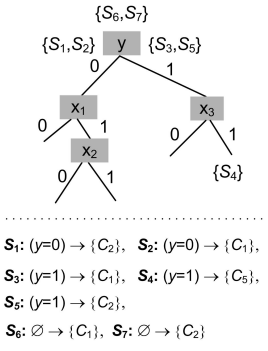

Fig. 2. Derived D-sequents

Using D-sequents. The main concern of *DS-PQE* is to prove redundancy of PR-clauses. Branching is used to reach subspaces where proving redundancy is easy. The redundancy of a PR-clause C is expressed by a Dependency Sequent (D-sequent). In short notation, a D-sequent is a record $\boldsymbol{s} \to \{C\}$ saying that clause C is redundant in formula $\exists X[F \wedge G]$ in any subspace where assignment $\boldsymbol{s}$ is made. We will refer to $\boldsymbol{s}$ as the **conditional part** of the D-sequent. The D-sequents $S_1, \ldots, S_7$ derived by *DS-PQE* are shown in Figure 2. They are numbered in the order they were generated. So-called atomic D-sequents record trivial cases of redundancy. More complex D-sequents are derived by a resolution-like operation called *join*. When *DS-PQE* returns to the root, it derives D-sequents stating the unconditional redundancy of the X-clauses of F.

Merging results of different branches. Let v be the current branching variable and $v = 0$ be the first branch explored by *DS-PQE*. After completing this branch, *DS-PQE* proves redundancy of all clauses that currently have the PR-status. (The only exception is the case when a PR-clause gets falsified in branch $v = 0$. We discuss this exception below.) Then *DS-PQE* explores branch $v = 1$ and derives D-sequents stating redundancy of clauses in this branch. Before backtracking from node v, *DS-PQE* uses operation *join* to produce D-sequents whose conditional part does not depend on v. For example, in branch $y = 0$, D-sequent S_1 equal to $(y = 0) \rightarrow \{C_2\}$ was derived. In branch $y = 1$, D-sequent S_5 equal to $(y = 1) \rightarrow \{C_2\}$ was derived. By joining S_1 and S_5 at variable y, D-sequent S_7 equal to $\emptyset \rightarrow \{C_2\}$ was produced where the conditional part did not depend on y.

Derivation of new clauses. Proving redundancy of PR-clauses in subspace $y = 0$ required derivation of clauses $C_7 = \overline{x}_1$ and $C_8 = y$. For instance, clause C_7 was generated at node $(y = 0, x_1 = 1)$ by resolving C_3 and C_4. Clause C_7 was *temporarily* added to F to make PR-clauses C_3 and C_4 redundant at the node above. However, C_7 was removed from formula F after derivation of clause C_8 because the former is subsumed by the latter in subspace $y = 0$. This is similar to conflict clause generation in SAT-solvers where the intermediate resolvents are discarded.

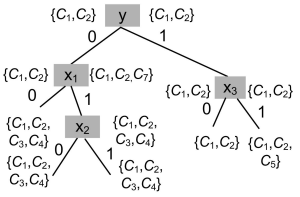

Fig. 3. Dynamics of the $PR(\boldsymbol{q})$ set

Derivation of atomic D-sequents. $S_1, \ldots, S_5$ are the atomic D-sequents derived by *DS-PQE*. They record trivial cases of redundancy. (Due to the simplicity of this example, the conditional part of all atomic D-sequents has only assignment to y *i.e.*, the free variable. In general, however, the conditional part of a D-sequent also contains assignments to quantified variables.) There are three kinds of atomic D-sequents. *D-sequents of the first kind* state redundancy of clauses satisfied in a subspace. For instance, D-sequent S_1 states redundancy of clause C_2 satisfied by assignment $y = 0$. *D-sequents of the second kind* record the fact that a clause is redundant because some other clause is falsified in the current subspace. For instance, D-sequent S_2 states that C_1 is redundant because clause $C_8 = y$ is falsified in subspace $y = 0$. *D-sequents of the third kind* record the fact that a clause is redundant in a subspace because it is blocked [15] at a variable v. That is this clause cannot be resolved on v. For example, D-sequent S_4 states redundancy of C_5 that cannot be resolved on x_4 in subspace $(y = 1, x_3 = 1)$. Clause C_5 is resolvable on x_4 only with C_6 but C_6 is satisfied by assignment $y = 1$. Atomic D-sequents are further discussed in Subsection 6.3.

Computation of the set of PR-clauses. The original set of PR-clauses is equal to the the initial set of X-clauses of F. Denote this set as PR_{init}. In our example, $PR_{init} = \{C_1, C_2\}$. There are two situations where $PR(\boldsymbol{q})$ is extended. The first situation occurs when a parent clause of a new resolvent is in $PR(\boldsymbol{q})$ and this resolvent is an X-clause. Then this resolvent is added to $PR(\boldsymbol{q})$. An example of that is clause $C_7 = \overline{x_1}$ obtained by resolving PR-clauses C_3 and C_4.

The second situation occurs when a PR-clause becomes unit. Suppose a PR-clause C is unit at node $\boldsymbol{q}$ and v is the unassigned variable of C where $v \in X$. *DS-PQE* first makes the assignment falsifying C. Suppose that this is assignment $v = 0$. Note that all PR-clauses but C itself are obviously redundant at node $\boldsymbol{q} \cup (v = 0)$. *DS-PQE* backtracks and explores the branch $v = 1$ where clause C is satisfied. At this point *DS-PQE* extends the set $PR(\boldsymbol{q} \cup (v = 1))$ by adding *every* clause of $F \wedge G$ that a) has literal $\overline{v}$; b) is not satisfied; c) is not already in $PR(\boldsymbol{q})$. The extension of the set of PR-clauses in the second situation is done to guarantee that clause C will be proved redundant when backtracking off the node $\boldsymbol{q}$. Depending on whether formula $F \wedge G$ is satisfiable or unsatisfiable in branch $v = 1$, the second situation splits into two cases considered below.

The first case is that formula $F \wedge G$ is *unsatisfiable* in branch $v = 1$. Then extension of the set of PR-clauses above guarantees that a clause falsified by $\boldsymbol{q} \cup (v = 1)$ will be derived to make the new PR-clauses redundant. Most importantly, this clause will be resolved with C on v to produce a clause rendering C redundant in subspace $\boldsymbol{q}$. In our example, the first case occurs at node $y = 0$ where PR-clause C_1 becomes unit. *DS-PQE* falsifies C_1 in branch $x_1 = 0$, backtracks and explores branch $x_1 = 1$. In this branch, clauses C_3, C_4 of G are made PR-clauses. This branch is unsatisfiable. Making C_3,C_4 PR-clauses forces *DS-PQE* to derive $C_7 = \overline{x_1}$ that makes C_3, C_4 redundant. But the real goal of obtaining C_7 is to resolve it with C_1 to produce clause $C_8 = y$ that makes C_1 redundant.

The second case is that formula $F \wedge G$ is *satisfiable* in branch $v = 1$. Making the clauses with literal $\overline{v}$ PR-clauses forces *DS-PQE* to prove their redundancy. So when backtracking to node $\boldsymbol{q}$, clause C will be blocked at variable v and hence redundant. In our example, the second case occurs at node $y = 1$ where clause C_2 becomes unit. Clause C_2 gets falsified in branch $x_3 = 0$. Then *DS-PQE* backtracks and explores branch $x_3 = 1$. In this branch, C_5 of G becomes a new PR-clause as containing literal $\overline{x}_3$. This branch is satisfiable and C_5 is proved redundant without adding new clauses. Due to redundancy of C_5, clause C_2 gets blocked at node $y = 1$ and hence redundant.

Importantly, the extension of the set $PR(\boldsymbol{q})$ in the first and second situations above is *temporary*. Suppose that a clause C is added to $PR(\boldsymbol{q})$ as a result of the first situation. That is at least one of the parents of C is a PR-clause. Then C preserves its PR-status as long as its parents (see Subsection 6.7 for more details). In the second situation, the clauses that became PR-clauses at node $\boldsymbol{q}$ lose their PR-status when *DS-PQE* backtracks off this node.

Forming a solution to the PQE problem. The D-sequents derived by *DS-PQE* at a node of the search tree are *composable*. This means that the clauses that are redundant individually are also redundant together. For example, on returning to the root node, D-sequents S_6 and S_7 equal to $\emptyset \rightarrow \{C_1\}$ and $\emptyset \rightarrow \{C_2\}$ respectively are derived. The composability of S_6 and S_7 means that D-sequent $\emptyset \rightarrow \{C_1, C_2\}$ holds as well. The only new clause added to F is $C_8 = y$ (clause C_7 was added temporarily). After dropping the X-clauses C_1, C_2 from F as proved redundant one concludes that $y \wedge \exists X[G] \equiv \exists X[F \wedge G]$ and $F^* = y$ is a solution to the PQE problem.

4 Basic Definitions

In this section, we give relevant definitions.

Definition 1. *An* $\exists$***CNF formula*** *is a formula of the form* $\exists X[F]$ *where* F *is a Boolean CNF formula, and* X *is a set of Boolean variables. Let* $\boldsymbol{q}$ *be an assignment,* F *be a CNF formula, and* C *be a clause.* $Vars(\boldsymbol{q})$ *denotes the variables assigned in* $\boldsymbol{q}$*;* $Vars(F)$ *denotes the set of variables of* F*;* $Vars(C)$ *denotes the variables of* C*; and* $Vars(\exists X[F]) = Vars(F) \setminus X$*.*

We consider *true* and *false* as a special kind of clauses.

Definition 2. *Let* C *be a clause,* H *be a CNF formula, and* $\boldsymbol{q}$ *be an assignment such that* $Vars(\boldsymbol{q}) \subseteq Vars(H)$*. Denote by* $C_{\boldsymbol{q}}$ *the clause equal to true if* C *is satisfied by* $\boldsymbol{q}$*; otherwise* $C_{\boldsymbol{q}}$ *is the clause obtained from* C *by removing all literals falsified by* $\boldsymbol{q}$*.* $H_{\boldsymbol{q}}$ *denotes the formula obtained from* H *by replacing every clause* C *of* H *with* $C_{\boldsymbol{q}}$*. In this paper, we assume that clause* $C_{\boldsymbol{q}}$ *equal to true remains in* $H_{\boldsymbol{q}}$*. We treat such a clause as redundant in* $H_{\boldsymbol{q}}$*.*

Let $\exists X[H]$ be an $\exists$CNF and $\boldsymbol{y}$ be an assignment to $Vars(H) \setminus X$. Note that in this case, $(\exists X[H])_{\boldsymbol{y}} = \exists X[H_{\boldsymbol{y}}]$.

Definition 3. *Let* S, Q *be* $\exists$CNF *formulas. We say that* S, Q *are* ***equivalent****, written* $S \equiv Q$*, if for all assignments,* $\boldsymbol{y}$*, such that* $Vars(\boldsymbol{y}) \supseteq (Vars(S) \cup Vars(Q))$*, we have* $S_{\boldsymbol{y}} = Q_{\boldsymbol{y}}$*. Notice that* $S_{\boldsymbol{y}}$ *and* $Q_{\boldsymbol{y}}$ *have no free variables, so by* $S_{\boldsymbol{y}} = Q_{\boldsymbol{y}}$ *we mean semantic equivalence.*

Definition 4. *The* ***Quantifier Elimination (QE) problem*** *for* $\exists$CNF *formula* $\exists X[H]$ *is to find a CNF formula* H^* *such that* $H^* \equiv \exists X[H]$*. The* ***Partial QE (PQE) problem*** *for* $\exists$CNF *formula* $\exists X[F \wedge G]$ *is to find a CNF formula* F^* *such that* $F^* \wedge \exists X[G] \equiv \exists X[F \wedge G]$*.*

Definition 5. *Let* X *be a set of Boolean variables,* H *be a CNF formula and* R *be a subset of* X*-clauses of* H*. The clauses of* R *are* ***redundant*** *in CNF formula* H *if* $H \equiv (H \setminus R)$*. The clauses of* R *are* ***redundant*** *in* $\exists$CNF *formula* $\exists X[H]$ *if* $\exists X[H] \equiv \exists X[H \setminus R]$*. Note that* $H \equiv (H \setminus R)$ *implies* $\exists X[H] \equiv \exists X[H \setminus R]$ *but the opposite is not true.*

The notion of clause redundancy in a *quantified* formula is very powerful. For example, if formula $H(X)$ is satisfiable, *every* clause of H is redundant in $\exists X[H]$.

5 Dependency Sequents

In this section, we recall clause Dependency sequents (D-sequents) introduced in [7], operation *join* and the notion of composability. Informally, the join operation extends resolution-like reasoning to subspaces where formula is satisfiable. For example, in Definition 7, formula H can be satisfiable in subspaces $\boldsymbol{s'}$ and $\boldsymbol{s''}$. In this paper, we will refer to clause D-sequents of [7] as just D-sequents.

Definition 6. *Let $\exists X[H]$ be an* $\exists$CNF *formula. Let $\boldsymbol{s}$ be an assignment to $Vars(H)$ and R be a subset of X-clauses of H. A dependency sequent (**D-sequent**) has the form $(\exists X[H], \boldsymbol{s}) \rightarrow R$. It states that the clauses of $R_{\boldsymbol{s}}$ are redundant in $\exists X[H_{\boldsymbol{s}}]$. Alternatively, we will say that the clauses of R are redundant in $\exists X[H]$ in subspace $\boldsymbol{s}$ (and in any other subspace $\boldsymbol{q}$ such that $\boldsymbol{s} \subseteq \boldsymbol{q}$).*

We will say that a D-sequent $(\exists X[H], \boldsymbol{s}) \rightarrow R$ **holds**, to tell apart a correct D-sequent where clauses of R are indeed redundant in $\exists X[H]$ in subspace $\boldsymbol{s}$ from a record $(\exists X[H], \boldsymbol{s}) \rightarrow R$ relating an arbitrary $\boldsymbol{s}$ with some set R of X-clauses.

Definition 7. *Let $\exists X[H]$ be an* $\exists$CNF *formula. Let D-sequents $(\exists X[H], \boldsymbol{s'}) \rightarrow R$ and $(\exists X[H], \boldsymbol{s''}) \rightarrow R$ hold. We will refer to them as parent D-sequents. Let $\boldsymbol{s'}$, $\boldsymbol{s''}$ have precisely one variable $v \in Vars(\boldsymbol{s'}) \cap Vars(\boldsymbol{s''})$ that is assigned differently in $\boldsymbol{s'}$ and $\boldsymbol{s''}$. Let $\boldsymbol{s}$ be the assignment equal to $\boldsymbol{s'} \cup \boldsymbol{s''}$ minus assignments to variable v. We will say that D-sequent $(\exists X[H], \boldsymbol{s}) \rightarrow R$ is obtained by **joining** the parent D-sequents at v. The fact that the parent D-sequents hold implies that the D-sequent obtained by joining them at v holds too [7].*

Definition 8. *Let $(\exists X[H], \boldsymbol{s'}) \rightarrow R'$ and $(\exists X[H], \boldsymbol{s''}) \rightarrow R''$ be two D-sequents such that every assignment to variables of $Vars(\boldsymbol{s'}) \cap Vars(\boldsymbol{s''})$ is the same in $\boldsymbol{s'}$ and $\boldsymbol{s''}$. We will call these D-sequents **composable** if the D-sequent $(\exists X[H], \boldsymbol{s'} \cup \boldsymbol{s''}) \rightarrow R' \cup R''$ holds.*

6 Algorithm

In this section, we describe a PQE algorithm called ***DS-PQE*** where DS stands for Dependency Sequents. *DS-PQE* algorithm is the result of a substantial modification of our QE algorithm *DCDS* described in [7]. The new features of *DS-PQE* are summarized in Subsection 6.7.

DS-PQE derives D-sequents $(\exists X[F \wedge G], \boldsymbol{s}) \rightarrow \{C\}$ stating the redundancy of PR-clause C in any subspace $\boldsymbol{q}$ such that $\boldsymbol{s} \subseteq \boldsymbol{q}$. From now on, we will use a short notation of D-sequents writing $\boldsymbol{s} \rightarrow \{C\}$ instead of $(\exists X[F \wedge G], \boldsymbol{s}) \rightarrow \{C\}$. We will assume that the parameter $\exists X[F \wedge G]$ missing in $\boldsymbol{s} \rightarrow \{C\}$ is the *current* $\exists$CNF formula (with all resolvents added to F). One can omit $\exists X[F \wedge G]$ from D-sequents because $(\exists X[F \wedge G], \boldsymbol{s}) \rightarrow \{C\}$ holds no matter how many resolvent clauses are added to F [7]. We will call D-sequent $\boldsymbol{s} \rightarrow \{C\}$ **active** in subspace $\boldsymbol{q}$ if $\boldsymbol{s} \subseteq \boldsymbol{q}$. The fact that $\boldsymbol{s} \rightarrow \{C\}$ is active in subspace $\boldsymbol{q}$ means that C is redundant in $\exists X[F \wedge G]$ in subspace $\boldsymbol{q}$.

6.1 Input and output of *DS-PQE*

Recall that a PR-clause is an X-clause of $F \wedge G$ whose redundancy needs to be proved in subspace $\boldsymbol{q}$ (see Section 3). *DS-PQE* shown in Figure 4 accepts an $\exists$CNF formula $\exists X[F \wedge G]$ (denoted as Φ), an assignment $\boldsymbol{q}$ to $Vars(F)$, the set of PR-clauses (denoted as W) and a set Ω of D-sequents active in subspace $\boldsymbol{q}$ stating redundancy of *some* PR-clauses in $\exists X[F \wedge G]$ in subspace $\boldsymbol{q}$.

// $\boldsymbol{q}$ is an assignment to $Vars(F \wedge G)$
// Ω is a set of active D-sequents
// Φ denotes $\exists X[F \wedge G]$
// W denotes $PR(\boldsymbol{q})$
// If *ds_pqe* returns *nil* (or a clause),
// $(F \wedge G)_{\boldsymbol{q}}$ is sat. (respect. unsat.)

ds_pqe(Φ,W,$\boldsymbol{q}$,Ω){
1 if ($\exists C \in F \cup G$ is falsif. by $\boldsymbol{q}$) {
2 $\Omega := atomic\_Dseqs1(\Omega, \boldsymbol{q}, C)$;
3 return(Φ, Ω, C);}
4 $\Omega := atomic\_Dseqs2(\Phi, \boldsymbol{q}, \Omega)$;
5* if ($every\_PR\_clause\_redund(W, \Omega)$)
6* return(Φ, Ω, nil);
- - - - - - - - - - - -
7 $v := pick\_variable(F \wedge G, \boldsymbol{q}, \Omega)$;
8 $\boldsymbol{q_b} := \boldsymbol{q} \cup \{(v = b)\}$;
9* $(\Phi, \Omega, C_b) :=ds\_pqe(\Phi,W, \boldsymbol{q_b},\Omega)$;
10 $\Omega^- := InactiveDseqs(F, \Omega, v)$;
11 if ($\Omega^- = \emptyset$) return(Φ, Ω, C_b);
12 $\Omega := \Omega \setminus \Omega^-$;
13* if ($impl\_assgn(v, \bar{b})$)
14* $W' := newPRclauses(W, F \wedge G, \bar{b})$;
15* else $W' := \emptyset$;
16 $\boldsymbol{q_{\bar{b}}} := \boldsymbol{q} \cup \{(v = \bar{b})\}$; $W'' := W \cup W'$;
17* $(\Phi, \Omega, C_{\bar{b}}) := ds\_pqe(\Phi,W'',\boldsymbol{q_{\bar{b}}},\Omega)$;
- - - - - - - - - - - -
18 if (($C_b \neq nil$) and ($C_{\bar{b}} \neq nil$)){
19 $C := resolve\_clauses(C_b, C_{\bar{b}}, v)$;
20 $F := F \wedge C$;
21 $\Omega := atomic\_Dseqs1(\Omega, \boldsymbol{q}, C)$;
22* if (($C_b \in W$) or ($C_{\bar{b}} \in W$))
23* if ($X\_clause(C)$)
24* $W := W \cup \{C\}$;
25 return(Φ, Ω, C);}
26 $\Omega := merge(\Phi, \boldsymbol{q}, v, \Omega^-, \Omega, C_b, C_{\bar{b}})$;
27 return(Φ, Ω, nil);}

Fig. 4. *DS-PQE* procedure

Similarly to Section 3, we will assume that the resolvent clauses are added to formula F while formula G remains unchanged. *DS-PQE* returns a formula $\exists X[F \wedge G]$ modified by resolvent clauses added to F (if any), a set Ω of D-sequents active in subspace $\boldsymbol{q}$ that state redundancy of *all* PR-clauses in $\exists X[F \wedge G]$ in subspace $\boldsymbol{q}$ and a clause C or *nil*. If $(F \wedge G)_{\boldsymbol{q}}$ is unsatisfiable, C is a clause of $F \wedge G$ falsified by $\boldsymbol{q}$. Otherwise, *DS-PQE* returns *nil* meaning that no clause implied by $F \wedge G$ is falsified by $\boldsymbol{q}$.

The active D-sequents derived by *DS-PQE* are composable. That is if $\boldsymbol{s_1} \rightarrow \{C_1\}$,..., $\boldsymbol{s_k} \rightarrow \{C_k\}$ are the active D-sequents of subspace $\boldsymbol{q}$, then the D-sequent $\boldsymbol{s^*} \rightarrow \{C_1, \ldots, C_k\}$ holds where $\boldsymbol{s^*} = \boldsymbol{s_1} \cup \ldots \cup \boldsymbol{s_k}$ and $\boldsymbol{s^*} \subseteq \boldsymbol{q}$. Like *DCDS* , *DS-PQE* achieves composability of D-sequents by proving redundancy of PR-clauses in a particular order (that can be different for different paths). This guarantees that no circular reasoning is possible and hence the D-sequents derived at a node of the search tree are composable.

A solution to the PQE problem in subspace $\boldsymbol{q}$ is obtained by discarding the PR-clauses of subspace $\boldsymbol{q}$ (specified by W) from the CNF formula F returned by *DS-PQE*. To solve the original problem of taking F out of the scope of the quantifiers in $\exists X[F \wedge G]$, one needs to call *DS-PQE* with $\boldsymbol{q} = \emptyset$, $\Omega = \emptyset, W = PR_{init}$. Recall that PR_{init} is the set of X-clauses of the original formula F.

6.2 The big picture

DS-PQE consists of three parts separated in Figure 4 by the dotted lines. In the first part (lines 1-6), *DS-PQE* builds atomic D-sequents recording trivial cases of redundancy of PR-clauses. If all the PR-clauses are proved redundant in $\exists X[F \wedge G]$ in subspace $\boldsymbol{q}$, *DS-PQE* terminates at node $\boldsymbol{q}$.

If some PR-clauses are not proved redundant yet, *DS-PQE* enters the second part of the code (lines 7-17). First, *DS-PQE* picks a branching variable v (line 7).

Then it recursively calls itself (line 9) starting the left branch of v by adding to $\boldsymbol{q}$ assignment $v = b$, $b \in \{0, 1\}$. Once the left branch is finished, *DS-PQE* explores the right branch $v = \overline{b}$ (line 17).

In the third part, *DS-PQE* merges the left and right branches (lines 18-27). This merging results in proving all PR-clauses redundant in $\exists X[F \wedge G]$ in subspace $\boldsymbol{q}$. For every PR-clause C proved redundant in subspace $\boldsymbol{q}$, the set Ω contains precisely one active D-sequent $\boldsymbol{s} \rightarrow \{C\}$ where $\boldsymbol{s} \subseteq \boldsymbol{q}$. As soon as C is proved redundant, it is marked and ignored until *DS-PQE* enters a subspace $\boldsymbol{q'}$ where $\boldsymbol{s} \not\subseteq \boldsymbol{q'}$ *i.e.*, a subspace where D-sequent $\boldsymbol{s} \rightarrow \{C\}$ becomes inactive. Then clause C gets unmarked signaling that *DS-PQE* does not have a proof of redundancy of C in subspace $\boldsymbol{q'}$ yet[1].

6.3 Building atomic D-sequents

Procedures *atomic_Dseqs1* and *atomic_Dseqs2* are called by *DS-PQE* to compute D-sequents for trivial cases of clause redundancy listed in Section 3. We refer to such D-sequents as *atomic*. Procedure *atomic_Dseqs1* is called when a clause C of $F \wedge G$ is falsified by $\boldsymbol{q}$. For every PR-clause $C'_{\boldsymbol{q}}$ of $F_{\boldsymbol{q}}$ that has no active D-sequent yet, *atomic_Dseq1* generates a D-sequent $\boldsymbol{s} \rightarrow \{C'\}$. Here $\boldsymbol{s}$ is the shortest assignment falsifying C.

If no clause of $F \wedge G$ is falsified by $\boldsymbol{q}$, procedure *atomic_Dseqs2* is called. It builds D-sequents for PR-clauses that became satisfied or blocked in $F_{\boldsymbol{q}}$. Let C be a clause satisfied by $\boldsymbol{q}$. Then D-sequent $\boldsymbol{s} \rightarrow \{C\}$ is generated where $\boldsymbol{s} = (w{=}b)$, $b \in \{0, 1\}$ is the assignment to a variable w satisfying C.

Let clause C be blocked [15] in $F_{\boldsymbol{q}}$ at variable $w \in X$. Let K be the set of clauses of $F \wedge G$ that can be resolved with C on w. The fact that C is blocked in $F_{\boldsymbol{q}}$ means that every clause of K is either satisfied by $\boldsymbol{q}$ or is proved redundant in subspace $\boldsymbol{q}$. In this case, *atomic_Dseqs2* generates a D-sequent $\boldsymbol{s} \rightarrow \{C\}$ where $\boldsymbol{s}$ is constructed as follows. If $C' \in K$ is satisfied by $\boldsymbol{q}$, then $\boldsymbol{s}$ contains the assignment to a variable of $\mathit{Vars}(\boldsymbol{q})$ that satisfies C'. If $C' \in K$ is proved redundant in subspace $\boldsymbol{q}$ and $\boldsymbol{r} \rightarrow \{C'\}$ is the active D-sequent for C', then $\boldsymbol{s}$ contains $\boldsymbol{r}$.

[1] This footnote is added to the paper after publication at HVC-14. The description of the algorithm given in Figure 4 is missing a case. (The implementation tested in experiments was correct but the pseudo-code of the algorithm given in Figure 4 lacks a few lines addressing the case in question.) The missing part of the algorithm addresses the situation where 1) formula $F \wedge G$ is satisfiable in the left branch ($v = b$) and unsatisfiable in the right branch ($v = \overline{b}$) and 2) assignment $v = \overline{b}$ is not derived from a clause that is currently unit in $F \wedge G$. In this case, *DS-PQE* adds clause $C_{\overline{b}}$ derived in the right branch to F , removes D-sequents derived in branches $v = b$ and $v = \overline{b}$, and unassigns variable v (i.e. backtracks).

After backtracking, assignment $v = \overline{b}$ can be derived from clause $C_{\overline{b}}$, and hence branching on v will be handled by lines 13-14 of Figure 4. The early backtracking is done to facilitate proving redundancy of new clause $C_{\overline{b}}$. In more detail, this situation is described in Section 6 of "E.Goldberg and P.Manolios, Partial quantifier elimination, arXiv:1407.4835v2 [cs.LO], 2017". (The latter is the second version of technical report [9].)

6.4 Selection of a branching variable

Let $\boldsymbol{q}$ be the assignment *DS-PQE* is called with. Let $Y = \mathit{Vars}(F) \setminus X$. *DS-PQE* branches on unassigned variables of X and Y. Importantly, an unassigned variable $x \in X \setminus \mathit{Vars}(\boldsymbol{q})$ is picked for branching *only* if a PR-clause contains x and is not proved redundant yet.

Although Boolean Constraint Propagation (BCP) is not shown explicitly in Figure 4, it is included into the *pick_variable* procedure as follows: a) preference is given to branching on variables of unit clauses of $F_{\boldsymbol{q}}$ (if any); b) if v is a variable of a unit clause $C_{\boldsymbol{q}}$ of $F_{\boldsymbol{q}}$ and v is picked for branching, then the value falsifying $C_{\boldsymbol{q}}$ is assigned first to cause immediate termination of this branch.

To simplify merging results of the left and right branches, *DS-PQE* first assigns values to variables of Y (see Subsection 6.6). This means that *pick_variable* never selects a variable $x \in X$ for branching, if there is an unassigned variable of Y. In particular, BCP does not assign values to variables of X if a variable of Y is still unassigned.

6.5 Switching from left to right branch

Let $\boldsymbol{s} \to \{C\}$ be a D-sequent of the set Ω computed by *DS-PQE* in the left branch $v = b$ (line 9 of Figure 4). We will call this D-sequent **symmetric in $\boldsymbol{v}$**, if v is not assigned in $\boldsymbol{s}$. Otherwise, this D-sequent is called **asymmetric in $\boldsymbol{v}$**. Notice that if $\boldsymbol{s}$ is symmetric in v, then D-sequent $\boldsymbol{s} \to \{C\}$ is active in the right branch $v = \bar{b}$ and so C is redundant in $\exists X[F \wedge G]$ in subspace $\boldsymbol{q} \cup \{(v = \bar{b})\}$. Denote by Ω^{-} the subset of active D-sequents that are asymmetric in v. It is computed in line 10. Before exploring the right branch (line 17), the PR-clauses of $F \wedge G$ whose redundancy is stated by D-sequents of Ω^{-} become non-redundant again.

6.6 Branch merging

Let $\boldsymbol{q_b} = \boldsymbol{q} \cup \{(v = b)\}$ and $\boldsymbol{q_{\bar{b}}} = \boldsymbol{q} \cup \{(v = \bar{b})\}$. The goal of branch merging is to use solutions of the PQE problem in subspaces $\boldsymbol{q_b}$ and $\boldsymbol{q_{\bar{b}}}$ to produce a solution to the PQE problem in subspace $\boldsymbol{q}$. If both $F_{\boldsymbol{q_b}}$ and $F_{\boldsymbol{q_{\bar{b}}}}$ are unsatisfiable, this is done as described in lines 19-25 of Figure 4. Let $C_b, C_{\bar{b}}$ be clauses returned in the left and right branches respectively. Then, the empty clauses $(C_b)_{\boldsymbol{q_b}}$ and $(C_{\bar{b}})_{\boldsymbol{q_{\bar{b}}}}$ are solutions to the PQE in subspaces $\boldsymbol{q_b}$ and $\boldsymbol{q_{\bar{b}}}$. The empty clause $C_{\boldsymbol{q}}$ where C is the resolvent of C_b and $C_{\bar{b}}$ added to F (line 20) is a solution to the PQE problem in subspace $\boldsymbol{q}$. After C is added, *atomic_Dseqs1* completes Ω by generation of atomic D-sequents built due to presence of a clause falsified by $\boldsymbol{q}$.

Suppose that $F_{\boldsymbol{q_b}}$ and/or $F_{\boldsymbol{q_{\bar{b}}}}$ is satisfiable. In this case, to finish solving the QE problem in subspace $\boldsymbol{q}$, one needs to make sure that every PR-clause is proved redundant in $F_{\boldsymbol{q}}$. This means that every PR-clause should have a D-sequent active in subspace $\boldsymbol{q}$ and hence symmetric in the branching variable v. This work is done by procedure *merge* (line 26) that consists of three steps.

In the first step, *merge* takes care of D-sequents of "old" PR-clauses that is the clauses that were present in F at the time the value of v was flipped from b

to $\overline{b}$. For every such PR-clause, a D-sequent was derived in the left branch $v = b$. Let S_b be a D-sequent from Ω^- (that is asymmetric in v) that states redundancy of clause C in the left branch. Let $S_{\overline{b}}$ be the D-sequent stating redundancy of C in the right branch. These D-sequents are joined at variable v to produce a new D-sequent stating redundancy of C in subspace $\boldsymbol{q}$.

In the second step, *merge* processes new PR-clauses that is PR-clauses generated in the right branch $v = \overline{b}$. No D-sequents were derived for such clauses in the branch $v = b$. Let S be a D-sequent $\boldsymbol{s} \rightarrow \{C\}$ derived in the right branch $v = \overline{b}$ where clause C was generated. If S is symmetric in v, it simply remains in Ω untouched. Otherwise, S is updated by removing the assignment to v from $\boldsymbol{s}$.

In the third step, if, say, clause C_b mentioned above is not equal to *nil*, a D-sequent is generated for C_b if it is a PR-clause. It can be shown [7] that due the fact that free variables are assigned before quantified (see Subsection 6.4), clause C_b is always blocked at the branching variable v. So, an atomic D-sequent is built for C_b as described in Subsection 6.3.

6.7 New features of *DS-PQE* with respect to *DCDS*

In this subsection, we focus on the part of *DS-PQE* that is different from *DCDS*. The lines of this part are marked with an asterisk inf Figure 4.

The main difference between *DS-PQE* and *DCDS* is that at every node $\boldsymbol{q}$ of the search tree, *DS-PQE* maintains a set $PR(\boldsymbol{q})$ of PR-clauses. $PR(\boldsymbol{q})$ contains all the X-clauses of F and some X-clauses of G (if any). *DS-PQE* terminates its work at node $\boldsymbol{q}$ when all the current PR-clauses are proved redundant (lines 5-6). In contrast to *DS-PQE*, *DCDS* terminates at node $\boldsymbol{q}$, when *all* X-clauses are proved redundant. Line 9 is marked because *DS-PQE* uses an additional parameter W when recursively calling itself to start the left branch of node $\boldsymbol{q}$. Here W specifies the set of PR-clauses to prove redundant in the left branch.

Lines 13-15 show how $PR(\boldsymbol{q})$ is extended. As we discussed in Section 3, this extension takes place when assignment $v = \overline{b}$ satisfies a unit PR-clause C. In this case, the set W' of new PR-clauses is computed. It consists of all the X-clauses that a) contain the literal of v falsified by assignment $v = \overline{b}$; b) are not PR-clauses and c) are not satisfied. As we explained in Section 3, this is done to facilitate proving redundancy of clause C at node $\boldsymbol{q}$. The set W' is added to W before the right branch is explored (lines 16-17). Notice that the clauses of W' have PR-status only in the subtree rooted at node $\boldsymbol{q}$. Upon return to node $\boldsymbol{q}$ from the right branch, the clauses of W' lose their PR-status.

As we mentioned in Section 3, one more source of new PR-clauses are resolvents (lines 22-24). Let $v = b$ and $v = \overline{b}$ be unsatisfiable branches and C_b and $C_{\overline{b}}$ be the clauses returned by *DS-PQE* . If C_b or $C_{\overline{b}}$ is currently a PR-clause, and the resolvent C is an X-clause, then C becomes a new PR-clause. One can think of a PR-clause as supplied with a tag indicating the level up to which this clause preserves its PR-status. If only one of the clauses C_b and $C_{\overline{b}}$ is a PR-clause, then C inherits the tag of this clause. If both parents have the PR-status, the resolvent inherits the tag of the parent clause that preserves its PR-status longer.

6.8 Correctness of *DS-PQE*

The correctness of *DS-PQE* is proved similarly to that of *DCDS* [7]. *DS-PQE* is complete because it examines a finite search tree. Here is an informal explanation of why *DS-PQE* is sound. First, the clauses added to F are produced by resolution and so are correct in the sense they are implied by $F \wedge G$. Second, the atomic D-sequents built by *DS-PQE* are correct. Third, new D-sequents produced by operation *join* are correct. Fourth, the D-sequents of individual clauses are composable.

So when *DS-PQE* returns to the root node of the search tree, it derives the correct D-sequent $(\exists X[F \wedge G], \emptyset) \rightarrow F^X$. Here F^X denotes the set of all X-clauses of F. By removing the X-clauses from F one obtains formula F^* such that $\exists X[F^* \wedge G] \equiv \exists X[F \wedge G]$. Since F^* does not depend on variables of X it can be taken out of the scope of quantifiers.

7 Background

QE has been studied by many researchers, due to its important role in verification *e.g.*, in model checking. QE methods are typically based on BDDs [2, 3] or SAT [16, 11, 17, 13, 5, 12, 14]. At the same time, we do not know of research where the PQE problem was solved or even formulated. Of course, identification and removal of redundant clauses is often used in preprocessing procedures of QBF-algorithms and SAT-solvers [4, 1]. However, these procedures typically exploit only situations where clause redundancies are obvious.

One of the most important differences of PQE from QE is that a PQE-algorithm has to have a significant degree of "structure-awareness". This is because PQE is essentially based on the notion of redundancy of a subset of clauses in a quantified formula. So it is not clear, for example, if a BDD-based algorithm would benefit from replacing QE with PQE. This also applies to many SAT-based algorithms of QE. For instance, in [6] we presented a QE algorithm called DDS that was arguably more structure aware than its SAT-based predecessors. DDS is based on the notion of D-sequents defined in terms of redundancy of variables rather than clauses. DDS makes quantified variables redundant in subspaces and merges the results of different branches. Despite its structure-awareness, it is hard to adjust DDS to solving PQE: in PQE, one, in general, does not eliminate quantified variables (only some clauses with quantified variables are eliminated).

Interestingly, there is no trivial algorithm for solving PQE like solving QE by resolving out quantified variables one by one. For example, one cannot solve PQE by simply resolving out X-clauses of formula F in $\exists X[F \wedge G]$ because this can lead to looping [9].

8 Experimental Results

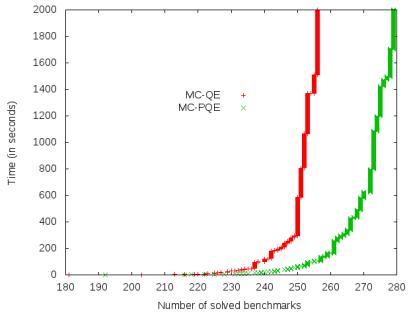

Fig. 5. Performance of model checkers on 282 examples solved by *MC-QE* or *MC-PQE*

Since we are not aware of another tool performing PQE, in the experiments we focused on contrasting PQE and QE. Namely, we compared *DS-PQE* with our QE algorithm called *DCDS* [7]. The fact that *DS-PQE* and *DCDS* are close in terms of implementation techniques is beneficial: any difference in performance should be attributed to difference in algorithms rather than implementations.

In the experiments, we used *DS-PQE* and *DCDS* for backward model checking. It is important to emphasize that, in the long run, we plan to use PQE in new types of model checkers like the ones mentioned in Section 2. However, since these model checkers are not available yet we experimented with *DS-PQE* in the context of a traditional model checker. We will refer to the two algorithms for backward model checking based on *DS-PQE* and *DCDS* as *MC-PQE* and *MC-QE* respectively. The difference between *MC-PQE* and *MC-QE* is as follows. Let $F(S')$ and $T(S,S')$ specify a set of next-states and transition relation respectively. The basic operation here is to find the pre-image $H(S)$ of F where $H \equiv \exists S'[F \wedge T]$. So H is a solution to the QE problem. As we showed in Subsection 2.2, one can also find H just by taking F out of the scope of the quantifiers in formula $\exists S'[F \wedge T]$. *MC-QE* computes H by making redundant *all* S'-clauses of $F \wedge T$ while *MC-PQE* finds H by making redundant *only* the S'-clauses of F.

Table 1. *Model checking results on some concrete examples*

benchmark	#latches	#gates	#iterations	bug	MC-QE (s.)	MC-PQE (s.)
bj08amba3g62	32	9,825	4	no	241	**38**
kenflashp03	51	3,738	2	no	**33**	104
pdtvishuffman2	55	831	6	yes	>2,000	**296**
pdtvisvsar05	82	2,097	4	no	1,368	**7.7**
pdtvisvsa16a01	188	6,162	2	no	>2,000	**17**
texaspimainp12	239	7,987	4	no	807	**580**
texasparsesysp1	312	11,860	10	yes	39	**25**
pj2002	1,175	15,384	3	no	254	**47**
mentorbm1and	4,344	31,684	2	no	**1.4**	1.7

The current implementations of *DCDS* and *DS-PQE* lack D-sequent re-using: the parent D-sequents are discarded after a join operation. We believe that re-using D-sequents should boost performance like clause recording in SAT-solving. However, when working on a new version of *DCDS* we found out that re-using D-sequents indiscriminately may lead to circular reasoning. We have solved this problem theoretically and resumed our work on the new version of *DCDS*. However, here we report the results of implementations that do not re-use D-sequents.

We compared *MC-PQE* and *MC-QE* on the 758 benchmarks of HWMCC-10 competition [18]. With the time limit of 2,000s, *MC-QE* and *MC-PQE* solved 258 and 279 benchmarks respectively. On the set of 253 benchmarks solved by both model checkers, *MC-PQE* was about 2 times faster (the total time is 4,652s versus 8,528s). However, on the set of 282 benchmarks solved by at least one model checker *MC-PQE* was about 6 times faster (10,652s versus 60,528s). Here we charged 2,000s, *i.e.*, the time limit, for every unsolved benchmark.

Figure 5 gives the performance of *MC-QE* and *MC-PQE* on the 282 benchmarks solved by at least one model checker in terms of the number of problems finished in a given amount of time. Model checking results on some concrete benchmarks are given in Table 1. The column *iterations* show the number of backward images computed by the algorithms before finding a bug or reaching a fixed point.

In [7], we compared *MC-QE* with a BDD-based model checker (MC-BDD). This comparison showed that although MC-BDD solved more benchmarks than MC-QE, there were 65 benchmarks solved by MC-QE that MC-BDD failed to solve. In addition to these 65 benchmarks, *MC-PQE* solved 7 more benchmarks that MC-BDD failed to solve (and that were not solved by *MC-QE* either).

Acknowledgment

This research was supported in part by DARPA under AFRL Cooperative Agreement No. FA8750-10-2-0233 and by NSF grants CCF-1117184 and CCF-1319580.

9 Conclusion

We introduced the Partial Quantifier Elimination problem (PQE), a generalization of the Quantifier Elimination problem (QE). We presented a PQE-algorithm based on the machinery of D-sequents and gave experimental results showing that PQE can be more efficient than QE. An efficient PQE-solver will enable new methods of solving old problems like model checking and SAT. In addition, many verification problems can be formulated and solved in terms of PQE rather than QE, a topic ripe for further exploration.

References

1. A.Biere, F.Lonsing, and M.Seidl. Blocked clause elimination for qbf. CADE-11, pages 101–115, 2011.
2. R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
3. P. Chauhan, E. M. Clarke, S. Jha, J.H. Kukula, H. Veith, and D. Wang. Using combinatorial optimization methods for quantification scheduling. CHARME-01, pages 293–309, 2001.
4. N. Eén and A. Biere. Effective preprocessing in sat through variable and clause elimination. In *SAT*, pages 61–75, 2005.
5. E.Goldberg and P. Manolios. Sat-solving based on boundary point elimination. HVC-10, pages 93–111, 2010.

6. E.Goldberg and P.Manolios. Quantifier elimination by dependency sequents. In *FMCAD-12*, pages 34–44, 2012.
7. E.Goldberg and P.Manolios. Quantifier elimination via clause redundancy. In *FMCAD-13*, pages 85–92, 2013.
8. E.Goldberg and P.Manolios. Bug hunting by computing range reduction. Technical Report arXiv:1408.7039 [cs.LO], 2014.
9. E.Goldberg and P.Manolios. Partial quantifier elimination. Technical Report arXiv:1407.4835 [cs.LO], 2014.
10. E.Goldberg and P.Manolios. Software for quantifier elimination in propositional logic. In *ICMS-2014,Seoul, South Korea, August 5-9*, pages 291–294, 2014.
11. H.Jin and F.Somenzi. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. DAC-05, pages 750–753, 2005.
12. J.Brauer, A.King, and J.Kriener. Existential quantification as incremental sat. CAV-11, pages 191–207, 2011.
13. J.R.Jiang. Quantifier elimination via functional composition. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV-09, pages 383–397, 2009.
14. W. Klieber, M. Janota, J.Marques-Silva, and E. M. Clarke. Solving qbf with free variables. In *CP*, pages 415–431, 2013.
15. O. Kullmann. New methods for 3-sat decision and worst-case analysis. *Theor. Comput. Sci.*, 223(1-2):1–72, 1999.
16. K. McMillan. Applying sat methods in unbounded symbolic model checking. In *Proc. of CAV-02*, pages 250–264. Springer-Verlag, 2002.
17. M.K.Ganai, A.Gupta, and P.Ashar. Efficient sat-based unbounded symbolic model checking using circuit cofactoring. ICCAD-04, pages 510–517, 2004.
18. HWMCC-2010 benchmarks, http://fmv.jku.at/hwmcc10/benchmarks.html.