

Verification Of Partial Quantifier Elimination

Eugene Goldberg

eu.goldberg@gmail.com

Abstract. Quantifier elimination (QE) is an important problem that has numerous applications. Unfortunately, QE is computationally very hard. Earlier we introduced a generalization of QE called *partial* QE (or PQE for short). PQE allows to unquantify a *part* of the formula. The appeal of PQE is twofold. First, many important problems can be solved in terms of PQE. Second, PQE can be drastically faster than QE if only a small part of the formula gets unquantified. To make PQE practical, one needs an algorithm for verifying the solution produced by a PQE solver. In this paper, we describe a very simple SAT-based verifier called *VerPQE* and provide some experimental results.

1 Introduction

Earlier, we introduced a generalization of Quantifier Elimination (QE) called *partial* QE (or PQE for short) [1]. PQE allows to unquantify a *part* of the formula. So, QE is just a special case of PQE where the entire formula gets unquantified. The appeal of PQE is twofold. First, it can be much more efficient than QE if only a small part of the formula gets unquantified. Second, many known verification problems like SAT, equivalence checking, model checking and new problems like property generation can be solved in terms of PQE [1,2,3,4,5]. So, PQE can be used to design new efficient algorithms. To make PQE practical, one needs to verify the correctness of the solution provided by a PQE solver. Such verification is the focus of this paper.

We consider PQE on propositional formulas in conjunctive normal form (CNF)¹ with existential quantifiers. PQE is defined as follows. Let $F(X, Y)$ be a propositional CNF formula where X, Y are sets of variables. Let G be a subset of clauses of F . Given a formula $\exists X[F]$, find a quantifier-free formula $H(Y)$ such that $\exists X[F] \equiv H \wedge \exists X[F \setminus G]$. In contrast to QE, only the clauses of G are taken out of the scope of quantifiers here (hence the name partial QE). We will refer to H as a **solution** to PQE. As we mentioned above, PQE *generalizes* QE. The latter is just a special case of PQE where $G = F$ and the entire formula is unquantified.

To verify the solution H above one needs to check if $\exists X[F] \equiv H \wedge \exists X[F \setminus G]$ indeed holds. If derivation of H is done in some proof system, one can check the

¹ Every formula is a propositional CNF formula unless otherwise stated. Given a CNF formula F represented as the conjunction of clauses $C_1 \wedge \dots \wedge C_k$, we will also consider F as the *set* of clauses $\{C_1, \dots, C_k\}$.

correctness of H by verifying the proof (like it is done for SAT-solvers). Since, PQE is currently in its infancy and no well established proof system exists we use a more straightforward approach. Namely, we present a very simple SAT-based verification algorithm called *VerPQE* that does not require any knowledge of how the solution H is produced. A flaw of *VerPQE* is that, in general, it does not scale well. Nevertheless, *VerPQE* can be quite useful in two scenarios. First, *VerPQE* is efficient enough to handle PQE problems formed from random formulas of up to 70-80 variables. Such examples can be employed when debugging a PQE solver. Second, *VerPQE* can efficiently verify even large PQE problems for a particular class of formulas described in Subsection 4.3.

The paper is structured as follows. Basic definitions are given in Section 2. Section 3 formally describes how a solution to PQE can be verified. The verification algorithm called *VerPQE* is presented in Section 4. Section 5 gives experimental results. Some background is provided in Section 6 and conclusions are made in Section 7.

2 Basic Definitions

In this section, when we say “formula” without mentioning quantifiers, we mean “a quantifier-free formula”.

Definition 1. We assume that formulas have only Boolean variables. A **literal** of a variable v is either v or its negation. A **clause** is a disjunction of literals. A formula F is in conjunctive normal form (**CNF**) if $F = C_1 \wedge \dots \wedge C_k$ where C_1, \dots, C_k are clauses. We will also view F as a **set of clauses** $\{C_1, \dots, C_k\}$. We assume that **every formula is in CNF** unless otherwise stated.

Definition 2. Let F be a formula. Then $\mathbf{Vars}(F)$ denotes the set of variables of F and $\mathbf{Vars}(\exists X[F])$ denotes $\mathbf{Vars}(F) \setminus X$.

Definition 3. Let V be a set of variables. An **assignment** \vec{q} to V is a mapping $V' \rightarrow \{0, 1\}$ where $V' \subseteq V$. We will denote the set of variables assigned in \vec{q} as $\mathbf{Vars}(\vec{q})$. We will refer to \vec{q} as a **full assignment** to V if $\mathbf{Vars}(\vec{q}) = V$. We will denote as $\vec{q} \subseteq \vec{r}$ the fact that a) $\mathbf{Vars}(\vec{q}) \subseteq \mathbf{Vars}(\vec{r})$ and b) every variable of $\mathbf{Vars}(\vec{q})$ has the same value in \vec{q} and \vec{r} .

Definition 4. A literal and a clause are said to be **satisfied** (respectively **falsified**) by an assignment \vec{q} if they evaluate to 1 (respectively 0) under \vec{q} .

Definition 5. Let C be a clause. Let H be a formula that may have quantifiers, and \vec{q} be an assignment to $\mathbf{Vars}(H)$. If C is satisfied by \vec{q} , then $C_{\vec{q}} \equiv \mathbf{1}$. Otherwise, $C_{\vec{q}}$ is the clause obtained from C by removing all literals falsified by \vec{q} . Denote by $H_{\vec{q}}$ the formula obtained from H by removing the clauses satisfied by \vec{q} and replacing every clause C unsatisfied by \vec{q} with $C_{\vec{q}}$.

Definition 6. Given a formula $\exists X[F(X, Y)]$, a clause C of F is called **quantified** if $\mathbf{Vars}(C) \cap X \neq \emptyset$.

Definition 7. Let G, H be formulas that may have existential quantifiers. We say that G, H are **equivalent**, written $\mathbf{G} \equiv \mathbf{H}$, if $G_{\vec{q}} = H_{\vec{q}}$ for all full assignments \vec{q} to $\text{Vars}(G) \cup \text{Vars}(H)$.

Definition 8. Let $F(X, Y)$ be a formula and $G \subseteq F$ and $G \neq \emptyset$. The clauses of G are said to be **redundant in $\exists X[F]$** if $\exists X[F] \equiv \exists X[F \setminus G]$.

Definition 9. Given a formula $\exists X[F(X, Y)]$ and G where $G \subseteq F$, the **Partial Quantifier Elimination (PQE)** problem is to find $H(Y)$ such that $\exists X[F] \equiv \mathbf{H} \wedge \exists X[F \setminus G]$. (So, PQE takes G out of the scope of quantifiers.) The formula H is called a **solution to PQE**. The case of PQE where $G = F$ is called **Quantifier Elimination (QE)**.

Remark 1. For the sake of simplicity, we will assume that every clause of formula G in Definition 9 is quantified.

Example 1. Consider the formula $F = C_1 \wedge C_2 \wedge C_3 \wedge C_4$ where $C_1 = \bar{x}_3 \vee x_4$, $C_2 = y_1 \vee x_3$, $C_3 = y_1 \vee \bar{x}_4$, $C_4 = y_2 \vee x_4$. Let Y denote $\{y_1, y_2\}$ and X denote $\{x_3, x_4\}$. Consider the PQE problem of taking C_1 out of $\exists X[F]$ i.e. finding $H(Y)$ such that $\exists X[F] \equiv H \wedge \exists X[F \setminus \{C_1\}]$. One can show that $\exists X[F] \equiv y_1 \wedge \exists X[F \setminus \{C_1\}]$. That is, $H = y_1$ is a solution to the PQE problem above.

Proposition 1. Let H be a solution to the PQE problem of Definition 9. That is $\exists X[F] \equiv H \wedge \exists X[F \setminus G]$. Then $F \Rightarrow H$ (i.e. F implies H).

The proofs of propositions are given in Appendix A.

3 Verification Of A Solution to PQE

Let $H(Y)$ be a solution found by a PQE solver when taking G out of $\exists X[F(X, Y)]$. That is $\exists X[F] \equiv H \wedge \exists X[F \setminus G]$ is supposed to hold. One can check if this is true (i.e. whether H is correct) using the proposition below.

Proposition 2. Formula $H(Y)$ is a solution to the PQE problem of taking G out of the scope of quantifiers in $\exists X[F(X, Y)]$ if and only if

- a) H is implied by F ;
- b) G is redundant in $H \wedge \exists X[F]$ i.e. $H \wedge \exists X[F] \equiv H \wedge \exists X[F \setminus G]$

Checking the first condition of Proposition 2 can be done by a SAT-solver. Namely, one just needs to check for every clause C of H if $F \wedge \bar{C}$ is unsatisfiable. (If so, then $F \Rightarrow C$.) Below, we describe how one can check the second condition of Proposition 2 in terms of boundary points.

Definition 10. Let F be a formula and G be a non-empty subset of clauses of F . A full assignment \vec{p} to $\text{Vars}(F)$ is called a **G -boundary point** of F if it falsifies G and satisfies $F \setminus G$.

The name “boundary point” is due to the fact that if the subset G is small, \vec{p} can sometimes be close to the boundary between assignments satisfying and falsifying F .

Definition 11. Let $F(X, Y)$ be a formula and G be a non-empty subset of F . Let (\vec{x}, \vec{y}) be a G -boundary point of F where \vec{x} and \vec{y} are full assignments to X and Y respectively. The G -boundary point (\vec{x}, \vec{y}) is called **Y-removable** (respectively **Y-unremovable**) if formula $F_{\vec{y}}$ is unsatisfiable (respectively satisfiable).

Recall that $F_{\vec{y}}$ describes the formula F in subspace \vec{y} . So the fact that $F_{\vec{y}}$ is unsatisfiable (or satisfiable) just means that F is unsatisfiable (respectively satisfiable) in subspace \vec{y} . We use the name “Y-removable boundary point” since such a boundary point can be eliminated by adding a clause implied by F that depends only on variables of Y . Indeed, suppose that (\vec{x}, \vec{y}) is a Y-removable G -boundary point. Then $F_{\vec{y}}$ is unsatisfiable and hence there is a clause $C(Y)$ falsified by \vec{y} and implied by F . Note that (\vec{x}, \vec{y}) is *not* a G -boundary point of $F \cup \{C\}$ because it *falsifies* the formula $(F \cup \{C\}) \setminus G$. So, adding C to F eliminates the G -boundary point (\vec{x}, \vec{y}) . On the contrary, a Y-unremovable boundary point (\vec{x}, \vec{y}) *cannot* be eliminated by adding a clause falsified by \vec{y} and implied by F .

Proposition 3. Let $F(X, Y)$ be a formula. Let G be a non-empty subset of clauses of F . The formula G is redundant in $\exists X[F]$ if and only if every G -boundary point of F (if any) is Y-unremovable.

So, to check the second condition of Proposition 2 one needs to show that every G -boundary point of $H \wedge F$ (if any) is Y-unremovable.

4 Description of *VerPQE*

```

VerPQE( $\exists X[F], G, H$ ) {
1  for every  $C \in H$  {
2     $\vec{p} := \text{Sat}(F \wedge \overline{C})$ 
3    if ( $\vec{p} \neq \text{nil}$ )
4      return(false)}
-----
5  for every  $C \in G$  {
6     $ok := \text{ChkRed}(\exists X[F \wedge H], C)$ 
7    if ( $ok = \text{false}$ ) return(false)
8     $F := F \setminus \{C\}$ 
9  return(true)

```

Fig. 1: *VerPQE*

In this section, we describe the algorithm for verification of PQE called *VerPQE*.

4.1 High-level view of *VerPQE*

A high-level view of *VerPQE* is given in Fig. 1. *VerPQE* accepts formula $\exists X[F]$, a subset $G \in F$ of clauses to take out of the scope of quantifiers and a solution H to this PQE problem. That is $\exists X[F]$ is supposed to be logically equivalent to $H \wedge \exists X[F \setminus G]$. *VerPQE* returns *true* if this equivalence holds and so, H is a correct solution. Otherwise, *VerPQE* returns *false*.

VerPQE consists of two parts separated by a solid line. In the first part (lines 1-4), *VerPQE* just checks if H is implied by F . This is done by checking

for every clause $C \in H$ if $F \wedge \overline{C}$ is satisfiable. If so, C is not implied by F and the solution H is incorrect. Hence, *VerPQE* returns *false* (line 4). In the second part (lines 5-9), for every clause $C \in G$, the algorithm checks if C is redundant in $\exists X[F \wedge H]$ by calling the function *ChkRed* (line 6). Namely, *ChkRed* checks if $\exists X[F \wedge H] \equiv \exists X[(F \setminus \{C\}) \wedge H]$. If so, C is removed from F (line 8). Otherwise, C is not redundant in $\exists X[F \wedge H]$ and *VerPQE* returns *false* (line 7). If all clauses of G can be removed from $\exists X[F \wedge H]$, then H is a correct solution and *VerPQE* returns *true*.

4.2 Description of *ChkRed*

```

ChkRed( $\exists X[F \wedge H], C$ ) {
1   $Y := \text{Vars}(F) \setminus X$ 
2   $Plg := \emptyset$ 
3  while (true) {
4     $F' := (F \setminus \{C\}) \wedge H$ 
5     $(\vec{x}, \vec{y}) := \text{Sat}(Plg \wedge F' \wedge \overline{C})$ 
6    if  $((\vec{x}, \vec{y}) = \text{nil})$ 
7      return(true)
8     $\vec{x}^* := \text{Sat}(F_{\vec{y}} \wedge H_{\vec{y}})$ 
9    if  $(\vec{x}^* = \text{nil})$ 
10     return(false)
11    $D := \text{PlugCls}(\vec{y}, \vec{x}^*, F, H)$ 
12    $Plg := Plg \cup \{D\}$  } }
```

Fig. 2: *ChkRed*

The pseudocode of *ChkRed* is shown in Fig. 2. *ChkRed* accepts the formula $\exists X[F(X, Y) \wedge H(Y)]$ and a quantified clause C to be checked for redundancy. *ChkRed* returns *true* if C is redundant in $\exists X[F \wedge H]$. Otherwise, it returns *false*. To verify redundancy of C , *ChkRed* checks if $F \wedge H$ has a Y -removable C -boundary point. If not, C is redundant. Otherwise, C is not redundant.

ChkRed starts with computing the set Y of unquantified variables (line 1). Then it initializes the set of “plugging” clauses (see below). The main work is done in the while loop (lines 3-12). *ChkRed* starts with checking if formula $F \wedge H$ has a C -boundary point (lines 4-5) i.e. checking if there is an assignment (\vec{x}, \vec{y}) satisfying $Plg \wedge (F \setminus \{C\}) \wedge H \wedge \overline{C}$. The formula Plg is used here to exclude the C -boundary points examined in the previous iterations of the loop. If no (\vec{x}, \vec{y}) exists, the clause C is redundant and *ChkRed* returns *true* (line 7).

If the assignment (\vec{x}, \vec{y}) above exists, *ChkRed* checks if formula $F_{\vec{y}} \wedge H_{\vec{y}}$ is satisfiable i.e. whether $F \wedge H$ is satisfiable in subspace \vec{y} (lines 8-10). If not, the C -boundary point (\vec{x}, \vec{y}) is Y -removable. This means that C is not redundant in $\exists X[F \wedge H]$ and *ChkRed* returns *false* (line 10). Otherwise, (\vec{x}, \vec{y}) is a Y -unremovable boundary point and *ChkRed* calls the function *PlugCls* to build a plugging clause $D(Y)$. The latter is falsified by \vec{y} and so excludes re-examining C -boundary points in the subspace \vec{y} . After that, *ChkRed* adds D to the formula Plg and starts a new iteration of the loop.

The simplest way to build D is to form the longest clause falsified by \vec{y} . One can try to make D shorter to exclude a greater subspace from future considerations. Suppose there is $\vec{y}^* \subset \vec{y}$ such that the assignment \vec{x}^* satisfying $F_{\vec{y}} \wedge H_{\vec{y}}$ (found in line 8) still satisfies $F_{\vec{y}^*} \wedge H_{\vec{y}^*}$. This means that every C -boundary point of the larger subspace \vec{y}^* is Y -unremovable too. So, one can add to Plg a shorter plugging clause D falsified by \vec{y}^* rather than \vec{y} .

4.3 Scalability issues

As we mentioned earlier, *VerPQE* consists of two parts. The first part of *VerPQE* checks if every clause of the solution H is implied by F . In the second part, for every clause $C \in G$, the function *ChkRed* checks if C is redundant in $\exists X[F \wedge H]$. (Recall that G is the subset of clauses that one must take out of $\exists X[F]$.) The first part reduces to $|H|$ calls to a SAT-solver. So, it is as scalable as SAT-solving (unless H blows up as the size of the PQE problem grows). The second part of *VerPQE* scales much poorer. The reason is that this part requires enumeration of Y -unremovable C -boundary points and the number of such points is typically grows exponentially. Besides, the plugging clauses produced by *ChkRed* are long. So, adding a plugging clause cannot exclude a big chunk of C -boundary points at once. So, the size of formulas that can be efficiently handled by *VerPQE* is limited by 70-80 variables.

There is however **an important case** where *VerPQE* can efficiently verify large formulas. This is the case where $F \wedge H$ does not have any Y -unremovable G -boundary points. (Since F and $F \wedge H$ have the same Y -unremovable G -boundary points, this means that F has no such boundary points either.) Then for every clause $C \in G$, the function *ChkRed* immediately finds out that the formula $(F \setminus \{C\}) \wedge H \wedge \overline{C}$ is unsatisfiable. So, the verification of solution H reduces to $|H| + |G|$ SAT-checks.

5 Experimental Results

In this section, we experimentally evaluate our implementation of *VerPQE*. In this implementation, we used Minisat [6] as an internal SAT-solver. The source of *VerPQE* and some examples can be downloaded from [7]. We conducted three experiments in which we verified solutions obtained by the PQE algorithm called *EG-PQE⁺* [5]. In the experiments we solved the PQE problem of taking a clause C out of formula $\exists X[F(X, Y)]$ i.e. finding a formula $H(Y)$ such that $\exists X[F] \equiv H \wedge \exists X[F \setminus \{C\}]$. In Subsections 5.1 and 5.2 we consider large formulas appearing in the process of “property generation”. Namely, these formulas were constructed when generating properties of circuits from the HWMCC-13 set as described in [5]. In Subsection 5.3, we consider small random formulas. In the experiments we used a computer with Intel[®] Core[™] i5-10500 CPU @ 3.10GHz.

5.1 Formulas where all boundary points are removable

In this subsection, we consider the PQE problems of taking C out of $\exists X[F(X, Y)]$ where all C -boundary points of F are Y -removable. In [5], we generated 3,736 of such formulas. In Table 1, we give a sample of 7 formulas. The first column of the table gives the name of the circuit of the HWMCC-13 set used to generate the PQE problem. (The real names of circuits *exmp1*, *exmp2* and *examp3* in the HWMCC-13 set are *mentorbm1*, *bob12m08m*, and *bob12m03m* respectively.)

Table 1: *VerPQE* on formulas where all boundary points are removable

name of circ.	clauses of F	variables of F	size of set Y	size of H	<i>EG-PQE</i> ⁺ run time (s)	<i>VerPQE</i> run time (s)
exmp1	64,365	26,998	4,376	1	0.2	0.03
6s207	73,457	30,540	3,012	6	0.2	0.04
exmp2	84,009	32,147	1,994	1	0.1	0.1
exmp3	94,523	41,354	5,174	825	11	0.4
6s249	226,666	78,289	1,111	1	0.4	0.1
6s428	231,506	92,274	3,790	118	2.8	0.2
6s311	259,086	87,974	519	80	2.0	0.1

taken by *EG-PQE*⁺ and *VerPQE* (in seconds) to finish the PQE problem and verify the solution. As we mentioned in Subsection 4.3, if all C -boundary points of F are Y -removable the same applies to formula $F \wedge H$. So, *VerPQE* should be very efficient even for large formulas. Table 1 substantiates this intuition.

5.2 Formulas with unremovable boundary points

Table 2: *VerPQE* on formulas with unremovable boundary points. The time limit is 600 sec.

name of circ.	clauses of F	variables of F	size of set Y	size of H	<i>EG-PQE</i> ⁺ run time (s)	<i>VerPQE</i> run time (s)
6s209	25,086	14,868	5,759	5	0.1	>600
6s413	29,321	14,063	4,343	18	0.2	>600
6s276	35,810	17,631	3,201	11	0.1	>600
6s176	39,704	15,754	1,566	0	0.9	>600
6s207	73,457	30,540	3,012	20	0.5	>600
6s110	83,396	34,165	807	6	0.2	0.1
6s275	109,328	49,130	3,196	2	0.1	>600

Table 2 shows that *VerPQE* failed to verify 6 out of 7 solutions in the time limit of 600 sec. whereas the corresponding problems were easily solved by *EG-PQE*⁺. (The reason is that *EG-PQE*⁺ uses a more powerful technique of proving redundancy of C than plugging unremovable boundary points as *VerPQE* does.) So, solutions H obtained for large formulas $\exists X[F]$ where F has a lot of unremovable boundary points cannot be efficiently verified by *VerPQE*.

5.3 Random formulas

In this subsection, we continue consider formulas with Y -unremovable C -boundary points. Only, in contrast to the previous subsection, here we consider small random formulas. In this experiment we verified solutions obtained for formulas whose number of variables ranged from 70 to 85. To get more reliable data, for each size we generated 100 random PQE problems and computed the average result. For each example, the formula F had 20% of two-literal and 80% of three-literal clauses.

The second and third columns give the number of clauses and variables of formula F . The fourth column shows the size of the set Y i.e. the number of unquantified variables in $\exists X[F(X, Y)]$. The next column gives the number of clauses in the solution H found by *EG-PQE*⁺. The last two columns show the time

Here we consider the same PQE problems as in the previous subsection. The only difference is that the formula F contains C -boundary points that are Y -unremovable. In [5], we generated 3,094 of such formulas. In Table 2, we give a sample of 7 formulas. The name and meaning of each column is the same as in Table 1.

Table 3: *VerPQE* on random formulas

num- ber of prob.	cla- uses of F	vari- ables of F	size of set Y	size of H	$EG-PQE^+$ run time (s)	$VerPQE$ run time (s)
100	140	70	35	28	0.01	1.0
100	150	75	37	41	0.01	4.7
100	160	80	40	69	0.03	11.5
100	170	85	42	63	0.03	98.3

The results of this experiment are shown in Table 3. Let us explain the meaning of each column of this table using its first line. The first column indicates that we generated 100 PQE problems of the same size shown in the next three columns.

That is for all 100 problems corresponding to the first line of Table 3 the number of clauses, variables and the size of the set Y was 140, 70 and 35 respectively. The last three columns of the first line show the *average* results over 100 examples. For instance, the first column of the three says that the average size of the solution H found by $EG-PQE^+$ was 28 clauses. Table 3 shows that the performance of $VerPQE$ drastically drops as the number of variables grows due to the exponential blow-up of the set of Y -unremovable C -boundary points.

6 Some Background

In this section, we give some background on boundary points. The notion of a boundary point with respect to a variable was introduced in [8]. (At the time it was called an *essential* point). Given a formula $F(X)$, a boundary point with respect to a variable $x \in X$ is a full assignment \vec{p} to X such that each clause falsified by \vec{p} contains x . Later we showed a relation between a resolution proof and boundary points [9]. Namely, it was shown that if F is unsatisfiable and contains a boundary point with respect to a variable x , any resolution proof that F is unsatisfiable has to contain a resolution on x . In [10], we presented an algorithm that performs SAT-solving via boundary point elimination.

In [11,12], we introduced the notion of a boundary point with respect to a subset of variables rather than a single variable. Using this notion we formulated a QE algorithm that builds a solution by eliminating removable boundary points. In [5], we formulated two PQE algorithms called $EG-PQE$ and $EG-PQE^+$. The algorithm $EG-PQE$ is quite similar to $VerPQE$ and implicitly employs the notion of a boundary point we introduced here i.e. the notion formulated with respect to a subset of *clauses* rather than variables. In this report, when describing $VerPQE$ we use this notion of a boundary point *explicitly*.

7 Conclusions

We present an algorithm called $VerPQE$ for verifying a solution to Partial Quantifier Elimination (PQE). The advantage of $VerPQE$ is that it does not need to know how this solution was obtained (e.g. if a particular proof system was employed). So, $VerPQE$ can be used to debug an *any* PQE algorithm. A flaw of $VerPQE$ is that its performance strongly depends on the presence of so-called unremovable boundary points of the formula at hand. If this formula has no such points,

VerPQE can efficiently verify solutions to very large PQE problems. Otherwise, its performance is, in general, limited to small problems of 70-80 variables.

References

1. E. Goldberg and P. Manolios, “Partial quantifier elimination,” in *Proc. of HVC-14*. Springer-Verlag, 2014, pp. 148–164.
2. —, “Software for quantifier elimination in propositional logic,” in *ICMS-2014, Seoul, South Korea, August 5-9, 2014*, pp. 291–294.
3. E. Goldberg, “Equivalence checking by logic relaxation,” in *FMCAD-16*, 2016, pp. 49–56.
4. —, “Property checking without inductive invariant generation,” Tech. Rep. arXiv:1602.05829 [cs.LO], 2016.
5. —, “Partial quantifier elimination and property generation,” Tech. Rep. arXiv:2303.13811 [cs.LO], 2023.
6. N. Eén and N. Sörensson, “An extensible sat-solver,” in *SAT*, Santa Margherita Ligure, Italy, 2003, pp. 502–518.
7. The source of *VerPQE*, <http://eigold.tripod.com/software/ver-pqe.1.0.tar.gz>.
8. E. Goldberg, M. Prasad, and R. Brayton, “Using problem symmetry in search based satisfiability algorithms,” in *DATE '02*, Paris, France, 2002, pp. 134–141.
9. E. Goldberg, “Boundary points and resolution,” in *Proc. of SAT*. Springer-Verlag, 2009, pp. 147–160.
10. E. Goldberg and P. Manolios, “Sat-solving based on boundary point elimination,” in *Proc. of HVC-10*. Springer-Verlag, 2011, pp. 93–111.
11. —, “Removal of quantifiers by elimination of boundary points,” Tech. Rep. arXiv:1204.1746 [cs.LO], 2012.
12. —, “Quantifier elimination by dependency sequents,” *Formal Methods in System Design*, vol. 45, no. 2, pp. 111–143, 2014.

Appendix

A Proofs Of Propositions

Proposition 1. *Let H be a solution to the PQE problem of Definition 9. That is $\exists X[F] \equiv H \wedge \exists X[F \setminus G]$. Then $F \Rightarrow H$ (i.e. F implies H).*

Proof. By conjoining both sides of the equality with H one concludes that $H \wedge \exists X[F] \equiv H \wedge \exists X[F \setminus G]$, which entails $H \wedge \exists X[F] \equiv \exists X[F]$. Then $\exists X[F] \Rightarrow H$ and thus $F \Rightarrow H$.

Proposition 2. *Formula $H(Y)$ is a solution to the PQE problem of taking G out of the scope of quantifiers in $\exists X[F(X, Y)]$ if and only if*

- a) H is implied by F ;
- b) G is redundant in $H \wedge \exists X[F]$ i.e. $H \wedge \exists X[F] \equiv H \wedge \exists X[F \setminus G]$

Proof. The if part. Given the two conditions above, one needs to prove that $\exists X[F] \equiv H \wedge \exists X[F \setminus G]$. Assume the contrary i.e. $\exists X[F] \not\equiv H \wedge \exists X[F \setminus G]$. Consider the two possible cases. The first case is that there exists a full assignment \vec{y} to Y such that F is satisfiable in subspace \vec{y} whereas $H \wedge (F \setminus G)$ is unsatisfiable in this subspace. Since $F \setminus G$ is satisfiable in the subspace \vec{y} , H is unsatisfiable in this subspace. So, F does not imply H and we have a contradiction.

The second case is that F is unsatisfiable in the subspace \vec{y} whereas $H \wedge (F \setminus G)$ is satisfiable there. Then $H \wedge F$ is unsatisfiable in subspace \vec{y} too. So, $H \wedge \exists X[F] \not\equiv H \wedge \exists X[F \setminus G]$ in subspace \vec{y} and hence G is not redundant in $H \wedge \exists X[F]$. So, we have a contradiction again.

The only if part. Given $\exists X[F] \equiv H \wedge \exists X[F \setminus G]$, one needs to prove the two conditions above. The first condition (that H is implied by F) follows from Proposition 1. Now assume that the second condition (that G is redundant in $H \wedge \exists X[F]$) does not hold. That is $H \wedge \exists X[F] \not\equiv H \wedge \exists X[F \setminus G]$. Note that if $H \wedge F$ is satisfiable in a subspace \vec{y} , then $H \wedge (F \setminus G)$ is satisfiable too. So, the only case to consider here is that $H \wedge F$ is unsatisfiable in a subspace \vec{y} whereas $H \wedge (F \setminus G)$ is satisfiable there. This means that F is unsatisfiable in the subspace \vec{y} . Then $\exists X[F] \not\equiv H \wedge \exists X[F \setminus G]$ in this subspace and we have a contradiction.

Proposition 3. *Let $F(X, Y)$ be a formula. Let G be a non-empty subset of clauses of G . The formula G is redundant in $\exists X[F]$ if and only if every G -boundary point of F (if any) is Y -unremovable.*

Proof. The if part. Given that every G -boundary point of F is Y -unremovable, one needs to show that G is redundant in $\exists X[F]$ i.e. $\exists X[F] \equiv \exists X[F \setminus G]$. Assume that this is not true. Then there is a full assignment \vec{y} to Y such that F is unsatisfiable in subspace \vec{y} whereas $F \setminus G$ is satisfiable there. This means that there is an assignment (\vec{x}, \vec{y}) falsifying F and satisfying $F \setminus G$. Since this assignment falsifies G , it is a G -boundary point. This boundary point is Y -removable, because F is unsatisfiable in subspace \vec{y} . So, we have a contradiction.

The only if part. Given that G is redundant in $\exists X[F]$, one needs to show that every G -boundary point of F is Y -unremovable. Assume the contrary i.e. there is a Y -removable G -boundary point of F . This means that there is an assignment (\vec{x}, \vec{y}) falsifying G and satisfying $F \setminus G$ such that F is unsatisfiable in subspace \vec{y} . Then $\exists X[F] \not\equiv \exists X[F \setminus G]$ in subspace \vec{y} and so, we have a contradiction.