Quantifier Elimination by Dependency Sequents

Eugene Goldberg and Panagiotis Manolios
Northeastern University, USA, {eigold,pete}@ccs.neu.edu

***Abstract*—We consider the problem of existential quantifier elimination for Boolean CNF formulas. We present a new method for solving this problem called Derivation of Dependency-Sequents (DDS). A Dependency-sequent (D-sequent) is used to record that a set of quantified variables is redundant under a partial assignment. We introduce the *join* operation that produces new D-sequents from existing ones. We show that DDS is compositional, *i.e.*, if our input formula is a conjunction of independent formulas, DDS automatically recognizes and exploits this information. We introduce an algorithm based on DDS and present experimental results demonstrating its potential.**

I. Introduction

In this paper, we consider the problem of eliminating quantifiers from formulas of the form $\exists X[F]$ where F is a Boolean CNF formula and some variables of F may be free of quantifiers. We will refer to such formulas as $\exists$CNF. The **Quantifier Elimination (QE) problem**, is to find a quantifier-free CNF formula G such that $G \equiv \exists X[F]$.

Our interest in the QE problem is twofold. First, the QE problem occurs in numerous areas of hardware design and verification, *e.g.*, in symbolic model checking [9], [10], [22] when computing reachable states. Second, one can argue that progress in solving the QE problem should have a deep impact on SAT-solving [15]. In particular, as McMillan pointed out, even the basic operation of resolution is related to the QE problem [23]. The resolvent C of clauses C',C'' on a variable v is obtained by eliminating the quantifier from $\exists v[C' \wedge C'']$.

The success of resolution-based SAT-solvers [21], [24] has led to the hunt for efficient SAT-based algorithms for the QE problem [23], [18], [6], [13]. In this paper, we continue in this direction by introducing a resolution-based QE algorithm operating on CNF formulas. Such formulas are ubiquitous in hardware verification because a circuit N can be represented by a CNF formula whose size is linear in that of N and that has the same set of variables as N.

Our approach is based on the following observation. The QE problem is trivial if F does not depend on variables of X. In this case, dropping the quantifiers from $\exists X[F]$ produces an equivalent formula. If F depends on X, after adding to F a set of clauses implied by F, the variables of X may become redundant in $\exists X[F]$. That is the clauses of F depending on X can be dropped and the resulting CNF formula G is equivalent to the original formula $\exists X[F]$. The problem is that one needs to know *when the variables of X become redundant*.

Unfortunately, resolution is deficient in expressing redundancy of variables. Let $\boldsymbol{y}$ be an assignment to all non-quantified variables of $\exists X[F]$. Let $F_{\boldsymbol{y}}$ denote F under assignment $\boldsymbol{y}$. If $F_{\boldsymbol{y}}$ is unsatisfiable, then a clause C falsified by $\boldsymbol{y}$ can be derived by resolving clauses of F. After adding C to F, the variables of X are redundant in $\exists X[F_{\boldsymbol{y}}]$. In this case, resolution works. Assume, however, that $F_{\boldsymbol{y}}$ is *satisfiable*. Then, the variables of X are *also redundant* in $\exists X[F_{\boldsymbol{y}}]$ because $F_{\boldsymbol{y}}$ remains satisfiable after removing any set of clauses. But a resolution derivation cannot express this fact since no clause falsified by $\boldsymbol{y}$ is implied by F.

To address this problem, we introduce the notion of Dependency sequents (*D-sequents*). A D-sequent has the form $(\exists X[F], \boldsymbol{q}) \rightarrow Z$ where $\boldsymbol{q}$ is a partial assignment to variables of F and $Z \subseteq X$. This D-sequent states that in the subspace specified by $\boldsymbol{q}$, the variables of Z are redundant in $\exists X[F]$. That is in this subspace, after dropping clauses with variables of Z from F one gets a formula equivalent to $\exists X[F]$. In particular, D-sequent $(\exists X[F], \boldsymbol{y}) \rightarrow X$ holds, if formula $F_{\boldsymbol{y}}$ is satisfiable where $\boldsymbol{y}$ is an assignment to the non-quantified variables of $\exists X[F]$.

In this paper, we introduce a QE algorithm called *DDS* (Derivation of D-Sequents). In *DDS*, adding resolvent clauses to F is accompanied by computing D-sequents. The latter are used to *precisely identify the moment when the variables of X are redundant*. It occurs when the D-sequent $(\exists X[F], \emptyset) \rightarrow X$ is derived stating unconditional redundancy of X. Here F is a CNF formula that includes the initial clauses and some resolvent clauses. Then, a solution to the QE problem is obtained from F by dropping the clauses containing variables of X.

DDS produces new D-sequents from existing ones by an operation called *join*. Let $(\exists X[F], \boldsymbol{q}')\rightarrow Z$ and $(\exists X[F], \boldsymbol{q}'')\rightarrow Z$ be valid D-sequents where $\boldsymbol{q}'$ and $\boldsymbol{q}''$ have opposite assignments to exactly one variable v. Then a new valid D-sequent $(\exists X[F], \boldsymbol{q}) \rightarrow Z$ can be obtained by joining the D-sequents above, where $\boldsymbol{q}$ contains all assignments of $\boldsymbol{q}'$ and $\boldsymbol{q}''$ but those to v.

In this paper, we compare *DDS* with its counterparts both theoretically and experimentally. In particular, we show that *DDS* is *compositional* while algorithms based on enumeration of satisfying assignments [23], [19], [13], [6] are not. Compositionality here means that given an $\exists$CNF formula $\exists X[F_1 \wedge \cdots \wedge F_k]$ where formulas F_i depend on non-overlapping sets of variables, *DDS* breaks the QE problem into k independent subproblems. *DDS* is a branching algorithm and yet it remains compositional no matter how branching variables are chosen. Compositionality of *DDS* means that its performance can be *exponentially better* than that of enumeration-based QE algorithms. Since *DDS* is a branching algorithm it can process variables of different branches in different orders. This gives *DDS* a big edge over QE algorithms that eliminate quantified variables one by one using a global order [18], [15].

D-sequents are tightly related to boundary points [14]. A boundary point is a complete assignment to variables of F with certain properties. To make variables of $Z \subseteq X$ redundant in $\exists X[F]$ one needs to eliminate a particular set of boundary points. This elimination is performed by adding to F resolvent clauses that do not depend on variables of Z. Although, *DDS* does not compute boundary points *explicitly*, we introduce them in this paper for the following two reasons. First, boundary points provide the semantics of *DDS*. In particular, the proofs of many propositions we use in this paper are based on the notion of boundary points. Second, *DDS* avoids an explicit computation of boundary points by using a particular branching order: non-quantified variables of $\exists X[F]$ are assigned before quantified. However, there is no guarantee that such an order is always optimal and so to achieve the best performance one may need to interleave assignments to quantified and non-quantified variables. In this case, to reduce the number of new resolvent clauses to be added to F, one, in general, cannot avoid an explicit computation of boundary points [17].

The contribution of this paper is as follows. First, we relate the notion of variable redundancy with the elimination of boundary points. Second, we introduce the notion of D-sequents and the operation of joining D-sequents. Third, we describe *DDS*, a QE algorithm; we prove its correctness and evaluate it experimentally. Fourth, we show that *DDS* is compositional.

This paper is structured as follows. In Section II, we relate the notions of variable redundancy and boundary points. Section III explains the strategy of *DDS* in terms of boundary point elimination. D-sequents are introduced in Section IV. Sections V and VI describe *DDS* and discuss its compositionality. Section VII gives experimental results. Background is discussed in Section VIII, and conclusions are presented in Section IX.

II. Redundant Variables, Boundary Points and Quantifier Elimination

The main objective of this section is to introduce the notion of redundant variables and to relate it to the elimination of removable boundary points.

A. Redundant Variables and Quantifier Elimination

Definition 1: An $\exists$CNF formula is a quantified CNF formula of the form $\exists X[F]$ where F is a CNF formula, and X is a set of Boolean variables. If we do not explicitly specify whether we are referring to CNF or $\exists$CNF formulas, when we write "formula" we mean either a CNF or $\exists$CNF formula. Let $\boldsymbol{q}$ be an assignment, F be a CNF formula, and C be a clause. $\mathit{Vars}(\boldsymbol{q})$ denotes the variables assigned in $\boldsymbol{q}$; $\mathit{Vars}(F)$ denotes the set of variables of F; $\mathit{Vars}(C)$ denotes the variables of C; and $\mathit{Vars}(\exists X[F]) = \mathit{Vars}(F) \setminus X$.

Definition 2: Let H be a formula, C be a clause of H, and $\boldsymbol{q}$ be an assignment such that $\mathit{Vars}(\boldsymbol{q}) \subseteq \mathit{Vars}(H)$. Denote by $C_{\boldsymbol{q}}$ the formula that a) is equal to *true* if C is satisfied by $\boldsymbol{q}$; b) is the clause obtained from C by removing all literals falsified by $\boldsymbol{q}$ otherwise. Denote by $H_{\boldsymbol{q}}$ the formula obtained from H by a) replacing every clause C of H with $C_{\boldsymbol{q}}$ if the latter is not true; b) discarding every clause C of H if $C_{\boldsymbol{q}}$ is true. If $\mathit{Vars}(H) \subseteq \mathit{Vars}(\boldsymbol{q})$, then $H_{\boldsymbol{q}}$ is semantically equivalent to a constant, and in the sequel, we will make use of this without explicit mention.

Definition 3: Let G, H be formulas. We say that G, H are *equivalent*, written $G \equiv H$, if for all assignments, $\boldsymbol{q}$, such that $\mathit{Vars}(\boldsymbol{q}) \supseteq (\mathit{Vars}(G) \cup \mathit{Vars}(H))$, we have $G_{\boldsymbol{q}} = H_{\boldsymbol{q}}$. Notice that $G_{\boldsymbol{q}}$ and $H_{\boldsymbol{q}}$ have no free variables, so by $G_{\boldsymbol{q}} = H_{\boldsymbol{q}}$ we mean semantic equivalence.

Definition 4: The Quantifier Elimination (QE) problem for $\exists$CNF formula $\exists X[F]$ consists of finding a CNF formula G such that $G \equiv \exists X[F]$.

Definition 5: A clause C of F is called a $\boldsymbol{Z}$**-clause** if $\mathit{Vars}(C) \cap Z \neq \emptyset$. Denote by $\boldsymbol{F^Z}$ the set of all Z-clauses of F.

Definition 6: The variables of Z are **redundant** in CNF formula F if $F \equiv (F \setminus F^Z)$. The variables of Z are **redundant** in $\exists$CNF formula $\exists X[F]$ if $\exists X[F] \equiv \exists X[F \setminus F^Z]$. We note that since $F \setminus F^Z$ does not contain any Z variables, we could have written $\exists(X \setminus Z)[F \setminus F^Z]$. To simplify notation, we avoid explicitly using this optimization in the rest of the paper.

B. Redundant Variables and Boundary Points

Definition 7: Given assignment $\boldsymbol{p}$ and a formula F, we say that $\boldsymbol{p}$ is an F-**point** (or a **point** of F) if $\mathit{Vars}(F) \subseteq \mathit{Vars}(\boldsymbol{p})$.

In the sequel, by "assignment" we mean a possibly partial one. To refer to a *complete* assignment we will use the term "point".

Definition 8: A point $\boldsymbol{p}$ of CNF formula F is called a $\boldsymbol{Z}$-**boundary point** of F if a) $Z \neq \emptyset$ and b) $F_{\boldsymbol{p}} = \mathit{false}$ and c) every clause of F falsified by $\boldsymbol{p}$ is a Z-clause and d) the previous condition breaks for every proper subset of Z.

The term "boundary" is justified as follows. Let F be a satisfiable CNF formula with at least one clause. Then there always exists a $\{x\}$-boundary point of F, $x \in \mathit{Vars}(F)$ that is different from a satisfying assignment only in value of x.

Definition 9: Given a CNF formula F and a Z-boundary point, $\boldsymbol{p}$, of F:

- $\boldsymbol{p}$ is X-**removable** in F if 1) $Z \subseteq X \subseteq \mathit{Vars}(F)$; and 2) there is a clause C such that a) $F \Rightarrow C$; b) $C_{\boldsymbol{p}} = \mathit{false}$; and c) $\mathit{Vars}(C) \cap X = \emptyset$.
- $\boldsymbol{p}$ is **removable** in $\exists X[F]$ if $\boldsymbol{p}$ is X-removable in F.

In the above definition, notice that $\boldsymbol{p}$ is not a Z-boundary point of $F \wedge C$ because $\boldsymbol{p}$ falsifies C and $\mathit{Vars}(C) \cap Z = \emptyset$.

Proposition 1: A Z-boundary point $\boldsymbol{p}$ of F is removable in $\exists X[F]$, iff one cannot turn $\boldsymbol{p}$ into an assignment satisfying F by changing only the values of variables of X.

The proofs are given in [16].

Proposition 2: The variables of $Z \subseteq X$ are not redundant in $\exists X[F]$ iff there is an X-removable W-boundary point of F, $W \subseteq Z$.

Proposition 2 justifies the following strategy of solving the QE problem. Add to F a set G of clauses that a) are implied by F; b) eliminate all X-removable Z-boundary points for all $Z \subseteq X$. By dropping all X-clauses of F, one produces a solution to the QE problem.

III. Boundary Points And Divide-And-Conquer Strategy

In this section, we provide the semantics of the QE algorithm *DDS* described in Section V. *DDS* is a branching algorithm. Given an $\exists$CNF formula $\exists X[F]$, it branches on variables of F until proving redundancy of variables of X in the current subspace becomes trivial. Then *DDS* merges the results obtained in different branches to prove that the variables of X are redundant in the entire search space.

Below we give propositions justifying the divide-and-conquer strategy of *DDS*. Proposition 3 shows how to perform elimination of removable boundary points of F in the subspace specified by assignment $\boldsymbol{q}$. Proposition 4 justifies proving redundancy of variables of X in subspace $\boldsymbol{q}$ one by one. Finally, Subsection III-B describes two cases where proving variable redundancy is trivial.

A. Decomposing the Problem of Boundary Point Elimination

Definition 10: Let $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ be assignments. The expression $\boldsymbol{q_1} \leq \boldsymbol{q_2}$ denotes the fact that $\mathit{Vars}(\boldsymbol{q_1}) \subseteq \mathit{Vars}(\boldsymbol{q_2})$ and each variable of $\mathit{Vars}(\boldsymbol{q_1})$ has the same value in $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$.

Proposition 3: Let $\exists X[F]$ be an $\exists$CNF formula and $\boldsymbol{q}$ be an assignment to $\mathit{Vars}(F)$. Let $\boldsymbol{p}$ be a Z-boundary point of F where $\boldsymbol{q} \leq \boldsymbol{p}$ and $Z \subseteq X$. Then if $\boldsymbol{p}$ is removable in $\exists X[F]$ it is also removable in $\exists X[F_{\boldsymbol{q}}]$.

The opposite is not true: a boundary point may be X-removable in $F_{\boldsymbol{q}}$ and not X-removable in F. For instance, if $X = \mathit{Vars}(F)$, a Z-boundary point $\boldsymbol{p}$ of F is removable in $\exists X[F]$ for any $Z \subseteq X$ only by adding an empty clause to F. So if F is satisfiable, $\boldsymbol{p}$ is not removable. Yet $\boldsymbol{p}$ may be removable in $\exists X[F_{\boldsymbol{q}}]$ if $F_{\boldsymbol{q}}$ is unsatisfiable.

Definition 11: Let $\exists X[F]$ be an $\exists$CNF formula, $\boldsymbol{q}$ be an assignment to $\mathit{Vars}(F)$, and $Z \subseteq (X \setminus \mathit{Vars}(\boldsymbol{q}))$. Variables of Z are called **virtually redundant** in $\exists X[F_{\boldsymbol{q}}]$ if $\exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z] \equiv (\exists X[F])_{\boldsymbol{r}}$ where $\boldsymbol{r} \leq \boldsymbol{q}$ and $\mathit{Vars}(\boldsymbol{r}) = \mathit{Vars}(\boldsymbol{q}) \setminus X$.

Redundancy of variables of Z in $\exists X[F_{\boldsymbol{q}}]$ in terms of Definition 6 is a special case of virtual redundancy. To prove variables of Z redundant in $\exists X[F]$ in subspace $\boldsymbol{q}$, it is sufficient to show virtual redundancy of Z in $\exists X[F_{\boldsymbol{q}}]$. The reason is that one can ignore Z-boundary points that are removable in $\exists X[F_{\boldsymbol{q}}]$ and not removable in $\exists X[F]$. We introduce a new notion of redundancy of variables Z in $F_{\boldsymbol{q}}$ because the operation of joining D-sequents (Definition 16) preserves only virtual redundancy of Z. In the sequel, when we say that variables of Z are redundant in $\exists X[F_{\boldsymbol{q}}]$ we mean that they are **at least virtually redundant**.

Proposition 4: Let $\exists X[F]$ be a CNF formula and $\boldsymbol{q}$ be an assignment to variables of F. Let the variables of Z be redundant in $\exists X[F_{\boldsymbol{q}}]$ where $Z \subseteq (X \setminus \mathit{Vars}(\boldsymbol{q}))$. Let a variable x of $X \setminus (\mathit{Vars}(\boldsymbol{q}) \cup Z)$ be redundant in $\exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z]$. Then the variables of $Z \cup \{x\}$ are redundant in $\exists X[F_{\boldsymbol{q}}]$.

Proposition 4 shows that one can make variables of $X \setminus \mathit{Vars}(\boldsymbol{q})$ redundant *incrementally*, if every $\{x\}$-clause is removed from $F_{\boldsymbol{q}}$ as soon as variable x is proved redundant.

B. Two Trivial Cases of Variable Redundancy

Definition 12: Let C' and C'' be clauses having opposite literals of exactly one variable $v \in \mathit{Vars}(C') \cap \mathit{Vars}(C'')$. The clause C consisting of all literals of C' and C'' but those of v is called the **resolvent** of C',C'' on v. Clause C is said to be obtained by **resolution** on v. Clauses C',C'' are called **resolvable** on v.

Definition 13: A variable x of a CNF formula F is called **blocked** if no two clauses of F are resolvable on x. A variable x is called **monotone** if it is a pure literal variable [11] (i.e. literals of only one polarity of x are present in F). A monotone variable is a special case of a blocked variable.

The notion of blocked variables is related to that of blocked clauses introduced in [20] (not to confuse with *blocking* clauses [23]). A clause C of F is blocked with respect to x if no clause C' of F is resolvable with C on x. Variable x is blocked in F if every $\{x\}$-clause of F is blocked with respect to x.

Proposition 5: Let $\exists X[F]$ be an $\exists$CNF formula and $\boldsymbol{q}$ be an assignment to $\mathit{Vars}(F)$. Let a variable x of $X \setminus \mathit{Vars}(\boldsymbol{q})$ be blocked in $F_{\boldsymbol{q}}$. Then x is redundant in $\exists X[F_{\boldsymbol{q}}]$.

Proposition 6: Let $\exists X[F]$ be an $\exists$CNF formula and $\boldsymbol{q}$ be an assignment to $\mathit{Vars}(F)$. Let $F_{\boldsymbol{q}}$ have an empty clause. Then the variables of $X \setminus \mathit{Vars}(\boldsymbol{q})$ are redundant in $\exists X[F_{\boldsymbol{q}}]$.

IV. Dependency Sequents (D-sequents)

In this section, we define D-sequents and introduce the operation of joining D-sequents.

A. Definition of D-sequents

Definition 14: Let $\exists X[F]$ be an $\exists$CNF formula. Let $\boldsymbol{q}$ be an assignment to $\mathit{Vars}(F)$ and Z be a subset of $X \setminus \mathit{Vars}(\boldsymbol{q})$. A dependency sequent (**D-sequent**) has the form $(\exists X[F], \boldsymbol{q}) \rightarrow Z$. It states that the variables of Z are redundant in $\exists X[F_{\boldsymbol{q}}]$.

Example 1: Consider an $\exists$CNF formula $\exists X[F]$ where $F = C_1 \wedge C_2$, $C_1 = x \vee y_1$ and $C_2 = \overline{x} \vee y_2$ and $X = \{x\}$. Let $\boldsymbol{q}$=$\{(y_1 = 1)\}$. Then $F_{\boldsymbol{q}}$ = C_2 because C_1 is satisfied. Notice that x is monotone and so redundant in $F_{\boldsymbol{q}}$ (Proposition 5). Hence, the D-sequent $(\exists X[F], \boldsymbol{q}) \rightarrow \{x\}$ holds.

According to Definition 14, a D-sequent holds with respect to a particular $\exists$CNF formula $\exists X[F]$. Proposition 7 shows that this D-sequent also holds after adding to F resolvent clauses.

Proposition 7: Let $\exists X[F]$ be an $\exists$CNF formula. Let $H = F \wedge G$ where $F \Rightarrow G$. Let $\boldsymbol{q}$ be an assignment to $\mathit{Vars}(F)$. Then if $(\exists X[F], \boldsymbol{q}) \rightarrow Z$ holds, $(\exists X[H], \boldsymbol{q}) \rightarrow Z$ does too.

B. Join Operation for D-sequents

In this subsection, we introduce the operation of joining D-sequents. The join operation produces a new D-sequent from two D-sequents derived earlier. The semantics of this operation in terms of elimination of boundary points is quite simple. Let A_1 and A_2 be subspaces from which all removable boundary

```
// $\xi$ denotes $\exists X[F]$, $\boldsymbol{q}$ is an assignment to $\mathit{Vars}(F)$
// $\Omega$ denotes a set of active D-sequents

$\mathit{DDS}(\xi,\boldsymbol{q},\Omega)\{$
1   $(\Omega, ans, C) \leftarrow atomic\_D\_seqs(\xi, \boldsymbol{q}, \Omega)$;
2   if ($ans$ = $sat$) return($\xi, \Omega, sat$);
3   if ($ans$ = $unsat$) return($\xi, \Omega, unsat, C$);
4   $v := pick\_variable(F, \boldsymbol{q}, \Omega)$;
5   $(\xi, \Omega, ans_0, C_0) \leftarrow \mathit{DDS}(\xi,\boldsymbol{q} \cup \{(v = 0)\},\Omega)$;
6   $(\Omega^{sym}, \Omega^{asym}) \leftarrow split(F, \Omega, v)$;
7   if ($\Omega^{asym} = \emptyset$) return($\xi, \Omega, ans_0, C_0$);
8   $\Omega := \Omega \setminus \Omega^{asym}$;
9   $(\xi, \Omega, ans_1, C_1) \leftarrow \mathit{DDS}(\xi,\boldsymbol{q} \cup \{(v = 1)\},\Omega)$;
10  if (($ans_0 = unsat$) and ($ans_1 = unsat$))$\{$
11      $C := resolve\_clauses(C_0, C_1, v)$;
12      $F := F \wedge C$;
13      $\Omega := process\_unsat\_clause(\xi, C, \Omega)$;
14      return($\xi, \Omega, unsat, C$);$\}$
15  $\Omega := merge(\xi, \boldsymbol{q}, v, \Omega^{asym}, \Omega)$;
16  return($\xi, \Omega, sat$);$\}$
```

Fig. 1. *DDS* procedure

```
$atomic\_D\_seqs(\xi, \boldsymbol{q}, \Omega)\{$
1   if ($\exists$ clause $C \in F$ falsif. by $\boldsymbol{q}$)$\{$
2       $\Omega := process\_unsat\_clause(\xi, C, \Omega)$;
3       return($\Omega, unsat, C$);$\}$
4   $\Omega := new\_redund\_vars(\xi,\boldsymbol{q},\Omega)$;
5   if ($all\_unassgn\_vars\_redund(\xi, \boldsymbol{q}, \Omega)$) return($\Omega, sat$);
6   return($\Omega, unknown$)$\}$;
```

Fig. 2. *atomic_D_seqs* procedure

points of F relevant to redundancy of $Z \subseteq X$ in $\exists X[F]$ have been eliminated. The join operation produces a new subspace A such that $A \subseteq A_1 \cup A_2$. We start with introducing resolution of assignments that is similar to that of clauses.

Definition 15: Let $\boldsymbol{q'}$ and $\boldsymbol{q''}$ be assignments in which exactly one variable $v \in \mathit{Vars}(\boldsymbol{q'}) \cap \mathit{Vars}(\boldsymbol{q''})$ is assigned different values. The assignment $\boldsymbol{q}$ consisting of all the assignments of $\boldsymbol{q'}$ and $\boldsymbol{q''}$ but those to v is called the **resolvent** of $\boldsymbol{q'}$,$\boldsymbol{q''}$ on v. Assignments $\boldsymbol{q'}$,$\boldsymbol{q''}$ are called **resolvable** on v.

Proposition 8: Let $\exists X[F]$ be an $\exists$CNF formula. Let D-sequents $(\exists X[F], \boldsymbol{q'}) \rightarrow Z$ and $(\exists X[F], \boldsymbol{q''}) \rightarrow Z$ hold. Let $\boldsymbol{q'}$, $\boldsymbol{q''}$ be resolvable on $v \in \mathit{Vars}(F)$ and $\boldsymbol{q}$ be the resolvent of $\boldsymbol{q'}$ and $\boldsymbol{q''}$. Then, the D-sequent $(\exists X[F], \boldsymbol{q}) \rightarrow Z$ holds too.

Definition 16: We will say that the D-sequent $(\exists X[F], \boldsymbol{q}) \rightarrow Z$ of Proposition 8 is produced by **joining D-sequents** $(\exists X[F], \boldsymbol{q'}) \rightarrow Z$ and $(\exists X[F], \boldsymbol{q''}) \rightarrow Z$ at v.

V. Description of *DDS*

In this section, we describe a QE algorithm called *DDS* (Derivation of D-Sequents). *DDS* derives D-sequents $(\exists X[F], \boldsymbol{q}) \rightarrow \{x\}$ stating the redundancy of one variable of X. From now on, we will use a short notation of D-sequents writing $\boldsymbol{q} \rightarrow \{x\}$ instead of $(\exists X[F], \boldsymbol{q}) \rightarrow \{x\}$. We will assume that the parameter $\exists X[F]$ missing in $\boldsymbol{q} \rightarrow \{x\}$ is the *current* $\exists$CNF formula (with all resolvent clauses added to F so far). One can omit $\exists X[F]$ from D-sequents because from Proposition 7 it follows that once D-sequent $(\exists X[F], \boldsymbol{q}) \rightarrow \{x\}$ is derived it holds after adding to F any set of resolvent clauses. We will call D-sequent $\boldsymbol{r} \rightarrow \{x\}$ **active** in the branch specified by assignment $\boldsymbol{q}$ if $\boldsymbol{r} \leq \boldsymbol{q}$ i.e. if this D-sequent provides a proof of redundancy of x in subspace $\boldsymbol{q}$.

A description of *DDS* is given in Figure 1. *DDS* accepts an $\exists$CNF formula $\exists X[F]$ (denoted as ξ), an assignment $\boldsymbol{q}$ to $\mathit{Vars}(F)$ and a set Ω of active D-sequents stating redundancy of *some* variables of $X \setminus \mathit{Vars}(\boldsymbol{q})$ in $\exists X[F_{\boldsymbol{q}}]$. *DDS* returns a modified formula $\exists X[F]$ (where resolvent clauses have been added to F) and a set Ω of active D-sequents stating redundancy of *every* variable of $X \setminus \mathit{Vars}(\boldsymbol{q})$ in $\exists X[F_{\boldsymbol{q}}]$. *DDS* also returns the answer *sat* if $F_{\boldsymbol{q}}$ is satisfiable. If $F_{\boldsymbol{q}}$ is unsatisfiable, *DDS* returns the answer *unsat* and a clause of F falsified by $\boldsymbol{q}$. To build a CNF formula equivalent to ξ, one needs to call *DDS* with $\boldsymbol{q} = \emptyset$, $\Omega = \emptyset$ and discard the X-clauses of the CNF formula F returned by *DDS*.

A. The Big Picture

First, *DDS* looks for variables whose redundancy is trivial to prove (lines 1-3). If some variables of $X \setminus \mathit{Vars}(\boldsymbol{q})$ are not proved redundant yet, *DDS* picks a branching variable v (line 4). Then it extends $\boldsymbol{q}$ by assignment $(v = 0)$ and recursively calls itself (line 5) starting the left branch of v. Once the left branch is finished, *DDS* extends $\boldsymbol{q}$ by $(v = 1)$ and explores the right branch (line 9). The results of the left and right branches are then merged (lines 10-16).

DDS terminates when for every variable x of $X \setminus \mathit{Vars}(\boldsymbol{q})$ it derives a D-sequent $\boldsymbol{g} \rightarrow \{x\}$ where $\boldsymbol{g} \leq \boldsymbol{q}$. According to Proposition 4, derivation of such D-sequents means that the D-sequent $\boldsymbol{q} \rightarrow X \setminus \mathit{Vars}(\boldsymbol{q})$ holds. Proposition 4 is applicable here because once a variable x of $X \setminus \mathit{Vars}(\boldsymbol{q})$ is proved redundant in $\exists X[F_{\boldsymbol{q}}]$, every $\{x\}$-clause of $F_{\boldsymbol{q}}$ is *marked* as redundant. (A redundant clause is ignored by *DDS* until it is unmarked as non-redundant.) So, *DDS* terminates when the QE problem is solved for ξ in subspace $\boldsymbol{q}$.

B. Building Atomic D-sequents

Procedure *atomic_D_seqs* is called by *DDS* to compute D-sequents for trivial cases of variable redundancy listed in Subsection III-B. We refer to such D-sequents as **atomic**. Procedure *atomic_D_seqs* returns an updated set of active D-sequents Ω and answer *sat*, *unsat*, or *unknown* depending on whether F is satisfiable, unsatisfiable or its satisfiability is not known yet. If F is unsatisfiable, *atomic_D_seqs* also returns a clause C of F falsified by the current assignment $\boldsymbol{q}$.

Lines 1-3 of Figure 2 show what is done when F contains a clause C falsified by $\boldsymbol{q}$. In this case, every unassigned variable of F becomes redundant (Proposition 6). So, for every variable of $x \in X \setminus \mathit{Vars}(\boldsymbol{q})$ for which Ω does not contain a D-sequent yet, procedure *process_unsat_clause* generates D-sequent $\boldsymbol{g} \rightarrow \{x\}$ and adds it to Ω. Here $\boldsymbol{g}$ is the shortest assignment falsifying C. Once Ω contains a D-sequent for every variable of $X \setminus \mathit{Vars}(\boldsymbol{q})$, *atomic_D_seqs* terminates returning the answer *unsat*, set Ω and clause C.

Suppose no clause of F is falsified by $\boldsymbol{q}$. Then for every variable x of $X \setminus \mathit{Vars}(\boldsymbol{q})$ that does not have a D-sequent in Ω and that is blocked, a D-sequent is built as explained below. This D-sequent is then added to Ω (line 4). If every variable of $X \setminus \mathit{Vars}(\boldsymbol{q})$ has a D-sequent in Ω, then $F_{\boldsymbol{q}}$ is satisfiable.

(If $F_{\boldsymbol{q}}$ is *unsatisfiable*, the variables of $X \setminus \mathit{Vars}(\boldsymbol{q})$ can be made redundant *only* by adding a clause falsified by $\boldsymbol{q}$.) So, *atomic_D_seqs* returns the answer *sat* and set Ω (line 5).

Given a blocked variable $x \in X \setminus \mathit{Vars}(\boldsymbol{q})$ of $F_{\boldsymbol{q}}$, a D-sequent $\boldsymbol{g} \rightarrow \{x\}$ is built as follows. Recall that an assignment $\boldsymbol{q}$ is a set of single-variable assignments. The fact that x is blocked in $F_{\boldsymbol{q}}$ means that for any pair of clauses C',C'' resolvable on x, C' or C'' is either satisfied by $\boldsymbol{q}$ or redundant (as containing a variable proved redundant in $\exists X[F_{\boldsymbol{q}}]$ earlier). Assume that it is clause C'. The assignment $\boldsymbol{g}$ is a subset of $\boldsymbol{q}$ guaranteeing that C' remains satisfied by $\boldsymbol{g}$ or redundant in $\exists X[F_{\boldsymbol{g}}]$ and so x remains blocked in $F_{\boldsymbol{g}}$. If C' is satisfied by $\boldsymbol{q}$, then $\boldsymbol{g}$ contains a single-variable assignment of $\boldsymbol{q}$ satisfying C'. If C' is not satisfied by $\boldsymbol{q}$ but contains a variable x^* proved redundant earlier, $\boldsymbol{g}$ contains all the single-variable assignments of $\boldsymbol{g}^*$ where $\boldsymbol{g}^* \rightarrow \{x^*\}$ is the D-sequent of Ω stating redundancy of x^*.

Searching for blocked variables of F for every call of *DDS* may be too expensive. Some simple techniques can be used to reduce the complexity of this search but a discussion of such techniques is beyond the scope of this paper. In the implementation of *DDS* we used in experiments, no optimization techniques were applied when searching for blocked variables.

C. Selection of a Branching Variable

Let $\boldsymbol{q}$ be the assignment *DDS* is called with and X_{red} be the set of variables of X whose D-sequents are in the current set Ω. Let $Y = \mathit{Vars}(F) \setminus X$. *DDS* branches only on a subset of free (*i.e.*, unassigned) variables of X and Y. Namely, a variable $x \in X \setminus \mathit{Vars}(\boldsymbol{q})$ is picked for branching only if $x \notin X_{red}$. A variable $y \in Y \setminus \mathit{Vars}(\boldsymbol{q})$ is picked for branching only if it is not detached. A variable y of $Y \setminus \mathit{Vars}(\boldsymbol{q})$ is called **detached** in $F_{\boldsymbol{q}}$, if every $\{y\}$-clause C of $F_{\boldsymbol{q}}$ that has at least one variable of X is redundant (because C contains a variable of X_{red}).

Although Boolean Constraint Propagation (BCP) is not shown explicitly in Figure 1, 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 of C of $F_{\boldsymbol{q}}$ and v is picked for branching, then the value falsifying C is assigned first to cause immediate termination of this branch. In the description of *DDS* of Figure 1, the left branch always explores assignment $v = 0$. But, obviously, v can be first assigned value 1.

To simplify making the branching variable v redundant when merging results of the left and right branches (see Subsection V-E), *DDS first assigns values to variables of Y*. This means that *pick_variable* never selects a variable $x \in X$ for branching, if there is a free non-detached variable of Y. In particular, BCP does not assign values to variables of X if a non-detached variable of Y is still unassigned.

D. Switching from Left to Right Branch

DDS prunes big chunks of the search space by not branching on redundant variables of X or detached variables of Y.

```
merge($\xi, \boldsymbol{q}, v, \Omega^{asym}, \Omega$){
1  $\Omega := join\_D\_seqs(v, \Omega^{asym}, \Omega)$;
2  if ($v \in X$) $\Omega := \Omega \cup \{atomic\_D\_seq\_for\_v(F, \boldsymbol{q}, v, \Omega)\}$;
3  return($\Omega$);}
```

Fig. 3. *merge* procedure

One more powerful pruning technique of *DDS* discussed in this subsection is to reduce the size of right branches.

Let $\boldsymbol{g} \rightarrow \{x\}$ be a D-sequent of the set Ω computed by *DDS* in the left branch $v = 0$ (line 5 of Figure 1). Notice that if $\boldsymbol{g}$ has no assignment (v=0), variable x remains redundant in $\exists X[F_{\boldsymbol{q_1}}]$ where $\boldsymbol{q_1} = \boldsymbol{q} \cup \{(v = 1)\}$. This is because $\boldsymbol{g} \rightarrow \{x\}$ is still active in the subspace specified by $\boldsymbol{q_1}$. *DDS* splits the set Ω into subsets Ω^{sym} and Ω^{asym} of D-sequents symmetric and asymmetric with respect to variable v (line 6). We call a D-sequent $\boldsymbol{g} \rightarrow \{x\}$ *symmetric* with respect to v, if $\boldsymbol{g}$ does not contain an assignment to v and *asymmetric* otherwise.

Denote by X^{sym} and X^{asym} the variables of $X_{red} \setminus \mathit{Vars}(\boldsymbol{q})$ whose redundancy is stated by D-sequents of Ω^{sym} and Ω^{asym} respectively. Before exploring the right branch (line 9), the variables of X^{asym} become non-redundant again. Every clause C of $F_{\boldsymbol{q}}$ with a variable of X^{asym} is unmarked as currently non-redundant unless $\mathit{Vars}(C) \cap X^{sym} \neq \emptyset$.

Reducing the set of free variables of the right branch to X^{asym} allows to prune big parts of the search space. In particular, if X^{asym} is empty there is no need to explore the right branch. In this case, *DDS* just returns the results of the left branch (line 7). Pruning the right branch when X^{asym} is empty is similar to non-chronological backtracking well known in SAT-solving [21].

E. Branch Merging

Let $\boldsymbol{q_0} = \boldsymbol{q} \cup \{(v = 0)\}$ and $\boldsymbol{q_1} = \boldsymbol{q} \cup \{(v = 1)\}$. The goal of branch merging is to extend the redundancy of all unassigned variables of X proved in $\exists X[F_{\boldsymbol{q_0}}]$ and $\exists X[F_{\boldsymbol{q_1}}]$ to formula $\exists X[F_{\boldsymbol{q}}]$. If both $F_{\boldsymbol{q_0}}$ and $F_{\boldsymbol{q_1}}$ turned out to be unsatisfiable, this is done as described in lines 11-14 of Figure 1. In this case, the unsatisfied clauses C_0 and C_1 of $F_{\boldsymbol{q_0}}$ and $F_{\boldsymbol{q_1}}$ returned in the left and right branches respectively are resolved on v. The resolvent C is added to F. Since F contains a clause C that is falsified by $\boldsymbol{q}$, for every variable $x \in X \setminus \mathit{Vars}(\boldsymbol{q})$ whose D-sequent is not in Ω, *DDS* derives an atomic D-sequent and adds it to Ω. This is performed by procedure *process_unsat_clause* described in Subsection V-B. If, say, $v \notin \mathit{Vars}(C_1)$, then *resolve_clauses* (line 11) returns C_1 itself since C_1 is falsified by $\boldsymbol{q}$ and no new clause is added to F.

If at least one branch returns answer *sat*, then *DDS* calls procedure *merge* described in Figure 3. First, *merge* takes care of the variables of X^{asym} (see Subsection V-D). Note that redundancy of variables of X^{asym} is already proved in both branches. If a D-sequent of a variable from X^{asym} returned in the *right* branch is asymmetric in v, then *join_D_seqs* (line 1) replaces it with a D-sequent symmetric in v as follows.

Let $x \in X^{asym}$ and S_0 and S_1 be the D-sequents stating the redundancy of x derived in the left and right branches respectively. Then *join_D_seqs* joins S_0 and S_1 at v producing a new D-sequent S. The latter also states the redundancy of x

but does not depend on v. D-sequent S_1 is replaced in Ω with S. If S_1 itself does not depend on v, no new D-sequent is produced. S_1 remains in Ω as the active D-sequent for variable x in $F_{\boldsymbol{q}}$.

Finally, if the branching variable v is in X, *DDS* derives a D-sequent stating the redundancy of v. Notice that v is not currently redundant in $\exists X[F_{\boldsymbol{q}}]$ because *DDS* does not branch on redundant variables. As we mentioned in Subsection V-C, the variables of $Y = \mathit{Vars}(F) \setminus X$ are assigned in *DDS* before those of X. This means that before v was selected for branching, all free non-detached variables of Y had been assigned. Besides, every variable of $X \setminus \mathit{Vars}(\boldsymbol{q})$ but v has just been proved redundant in $\exists X[F_{\boldsymbol{q}}]$. So, $F_{\boldsymbol{q}}$ may have only two types of non-redundant clauses: a) clauses having only detached variables of Y; b) unit clauses depending on v. Moreover, these unit clauses cannot contain literals of both polarities of v because *merge* is called only when either branch $v = 0$ or $v = 1$ is satisfied. Therefore, v is monotone. An atomic D-sequent S stating the redundancy of v is built as described in Subsection V-B and added to Ω (line 2). Then *merge* terminates returning Ω.

F. Correctness of *DDS* and Example

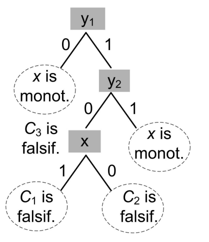

Fig. 4. Search tree built by *DDS*

Let *DDS* be called on formula $\xi = \exists X[F]$ with $\boldsymbol{q} = \emptyset$ and $\Omega = \emptyset$. Informally, *DDS* is correct because a) the atomic D-sequents built by *DDS* are correct; b) joining D-sequents produces a correct D-sequent; c) every clause added to formula F is produced by resolution and so is implied by F; d) by the time *DDS* backtracks to the root of the search tree, for every variable $x \in X$, D-sequent $\emptyset \rightarrow \{x\}$ is derived. Due to Proposition 4, this implies that the D-sequent $\emptyset \rightarrow X$ holds for the formula $\exists X[F]$ returned by *DDS*.

Proposition 9: *DDS* is sound and complete.

As we mentioned earlier, the proofs of the propositions given in this paper are provided in [16].

Example 2. Let $\exists X[F]$ be an $\exists$CNF formula where $F = C_1 \wedge C_2$, $C_1 = \overline{y}_1 \vee \overline{x}$, $C_2 = y_2 \vee x$ and $X = \{x\}$. To identify a particular *DDS* call we will use the corresponding assignment $\boldsymbol{q}$. For example, $DDS_{(y_1=1,y_2=0)}$ means that the assignments $y_1 = 1$ and $y_2 = 0$ were made at recursion depths 0 and 1 respectively. So the current recursion depth is 2. Originally, assignment $\boldsymbol{q}$ is empty so the initial call is $DDS_{(\emptyset)}$. The work of *DDS* is shown in Figures 4 and 5 used below to illustrate various aspects of *DDS*.

Branching variables. Figure 4 shows a search tree built by *DDS*. Recall that *DDS* branches on variables of $\mathit{Vars}(F) \setminus X = \{y_1, y_2\}$ before those of X (see Subsection V-C).

Leaves. The search tree of Figure 4 has four leaf nodes shown in dotted ovals. In each leaf node, variable x is either assigned or proved redundant. For example, x is proved redundant by $DDS_{(y_1=0)}$ and assigned by $DDS_{(y_1=1,y_2=0,x=1)}$.

Generation of new clauses. $DDS_{(y_1=1,y_2=0)}$ generates a new clause after branching on x. $DDS_{(y_1=1,y_2=0,x=1)}$ returns C_1 as a clause of F that is empty in $F_{(y_1=1,y_2=0,x=1)}$. Similarly, $DDS_{(y_1=1,y_2=0,x=0)}$ returns C_2 because it is empty in $F_{(y_1=1,y_2=0,x=0)}$. As described in Subsection V-E, in this case, *DDS* resolves clauses C_1 and C_2 on the branching variable x. The resolvent $C_3 = \overline{y}_1 \vee y_2$ is added to F.

Generation of atomic D-sequents. Figure 5 describes derivation of D-sequents. The dotted boxes show D-sequents obtained by the join operation. The atomic D-sequents are shown in dotted ovals. For instance, $DDS_{(y_1=0)}$ generates D-sequent S_1 equal to $(y_1\!=\!0) \rightarrow \{x\}$. S_1 holds because $F_{(y_1=0)}$=$y_2 \vee x$ and so x is a blocked (monotone) variable of $F_{(y_1=0)}$. The atomic D-sequent S_2 is derived by $DDS_{(y_1=1,y_2=0)}$. As we mentioned above, $DDS_{(y_1=1,y_2=0)}$ adds clause $C_3 = \overline{y}_1 \vee y_2$ to F. This clause is empty in $F_{(y_1=1,y_2=0)}$. So D-sequent S_2 equal to $(y_1 = 1, y_2 = 0) \rightarrow \{x\}$ is generated where $(y_1\!=\!1, y_2\!=\!0)$ is the shortest assignment falsifying C_3.

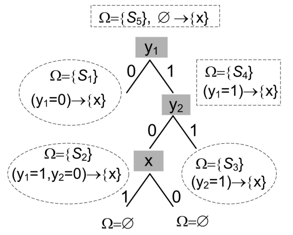

Fig. 5. Derivation of D-sequents

Switching from left to right branch. Let us consider switching between branches by $DDS_{(\emptyset)}$ where y_1 is picked for branching. The set of D-sequents $\Omega_{(\emptyset)}$ returned by the left branch equals $\{S_1\}$ where S_1 is equal to $(y_1 = 0) \rightarrow \{x\}$. The only clause $y_2 \vee x$ of $F_{(y_1=0)}$ is marked as redundant because it contains x that is currently redundant. Before starting the right branch $y_1 = 1$, $DDS_{(\emptyset)}$ splits $\Omega_{(\emptyset)}$ into subsets $\Omega^{sym}_{(\emptyset)}$ and $\Omega^{asym}_{(\emptyset)}$ of D-sequents respectively symmetric and asymmetric in y_1. Since the only D-sequent of $\Omega_{(\emptyset)}$ depends on y_1, then $\Omega^{asym}_{(\emptyset)}$=$\Omega_{(\emptyset)}$ and $\Omega^{sym}_{(\emptyset)}$=$\emptyset$. $DDS_{(\emptyset)}$ removes D-sequent S_1 from Ω because S_1 is not active if $y_1 = 1$. So, before $DDS_{(y_1=1)}$ is called, variable x becomes non-redundant and clause $C_2 = y_2 \vee x$ is unmarked as currently non-redundant.

Branch merging. Consider how branch merging is performed by $DDS_{(y_1=1)}$. In the left branch $y_2 = 0$, the set $\Omega_{(y_1=1)}$=$\{S_2\}$ is computed where S_2 is $(y_1 = 1, y_2 = 0) \rightarrow \{x\}$. Since S_2 depends on y_2, then $\Omega^{asym}_{(y_1=1)}$=$\Omega_{(y_1=1)}$. In the right branch $y_2 = 1$, the set $\Omega_{(y_1=1)}$=$\{S_3\}$ is computed where S_3 is $(y_2 = 1) \rightarrow \{x\}$. By joining S_2 and S_3 at y_2, D-sequent S_4 is derived that equals $(y_1 = 1) \rightarrow \{x\}$. S_4 states redundancy of x in $F_{(y_1=1)}$.

Termination. When $DDS_{(\emptyset)}$ terminates, $F = C_1 \wedge C_2 \wedge C_3$ where $C_3 = \overline{y}_1 \vee y_2$ and D-sequent $\emptyset \rightarrow \{x\}$ is derived. By dropping C_1, C_2 as X-clauses one obtains $C_3 \equiv \exists X[C_1 \wedge C_2]$.

VI. Compositionality of *DDS*

We will call a CNF formula F **compositional** if $F = F_1 \wedge \ldots \wedge F_k$ where $\mathit{Vars}(F_i) \cap \mathit{Vars}(F_j) = \emptyset$, $i \neq j$. We will say that an algorithm solves the QE problem specified by $\exists X[F]$ **compositionally** if it breaks this problem down into k independent subproblems of finding G_i equivalent to $\exists X[F_i]$. A formula G equivalent to $\exists X[F]$ is then built as $G_1 \wedge \ldots \wedge G_k$.

There are at least two reasons to look for compositional QE algorithms. First, even if the *original* formula F is not compositional, a formula $F_{\boldsymbol{q}}$ obtained from F by making assignment $\boldsymbol{q}$ may be compositional. Second, a practical formula F typically can be represented as $F_1(X_1, Y_1) \wedge \ldots \wedge F_k(X_k, Y_k)$ where X_i are internal variables of F_i and Y_i are communication variables i.e. ones shared by subformulas F_i. One can view F_i as describing a "design block" with external variables Y_i. The size of Y_i is usually much smaller than that of X_i. The latter fact is, arguably, what one means by saying that F has structure. One can view compositional formulas as a degenerate case where $|Y_i| = 0, i = 1, \ldots, k$ and so F_i do not "talk" to each other. Intuitively, an algorithm that does not scale well even if $|Y_i| = 0$ will not do well when $|Y_i| > 0$.

A QE algorithm based on enumeration of satisfying assignments is not compositional. The reason is that the set of assignments satisfying F is a Cartesian product of those satisfying F_i,$i = 1, \ldots, k$. So if, for example, all F_i are identical, the complexity of an enumeration based QE algorithm is *exponential* in k. A QE algorithm based on BDDs [7] is compositional only for variable orderings where variables of F_i and F_j, $i \neq j$ do not interleave.

Now we show the compositionality of *DDS*. By a *decision branching variable* mentioned in the proposition below, we mean that this variable was not present in a unit clause of the current formula when it was selected for branching.

Proposition 10 (compositionality of DDS): Let T be the search tree built by *DDS* when solving the QE problem $\exists X[F_1 \wedge \ldots \wedge F_k]$ above. Let $X_i = X \cap \mathit{Vars}(F_i)$ and $Y_i = \mathit{Vars}(F_i) \setminus X$. The size of T in the number of nodes is bounded by $|\mathit{Vars}(F)| \cdot (\eta(X_1 \cup Y_1) + \ldots + \eta(X_k \cup Y_k))$ where $\eta(X_i \cup Y_i) = 2 \cdot 3^{|X_i \cup Y_i|} \cdot (|X_i| + 1), i = 1, \ldots, k$ no matter how decision branching variables are chosen.

Proposition 10 is proved in [16] for a slightly modified version of *DDS*. Notice that the compositionality of *DDS* is not ideal. For example, if all subformulas F_i are identical, *DDS* is *quadratic* in k as opposed to being linear. Informally, *DDS* is compositional because D-sequents it derives have the form $\boldsymbol{g} \rightarrow \{x\}$ where $\mathit{Vars}(\boldsymbol{g}) \cup \{x\} \subseteq \mathit{Vars}(F_i)$. The only exception are D-sequents derived when the current assignment falsifies a clause of F. This exception is the reason why *DDS* is quadratic in k.

Importantly, the compositionality of *DDS* is achieved not by using some ad hoc techniques but is simply a result of applying the machinery of D-sequents. This provides some evidence that *DDS* can be successfully applied to non-compositional formulas of the form $F_1(X_1, Y_1) \wedge \ldots \wedge F_k(X_k, Y_k)$ where $|Y_i| > 0$ and $|Y_i| \ll |X_i|$, $i = 1, \ldots, k$.

Notice that a QE algorithm that resolves out variables one by one as in the DP procedure [12] is also compositional. (If $\mathit{Vars}(F_i) \cap \mathit{Vars}(F_j) = \emptyset$, then clauses of F_i and F_j cannot be resolved with each other). However, although such an algorithm may perform well on some classes of formulas, it is not very promising overall. This is due to the necessity to eliminate a variable in one big step, which may lead to generation of a very large number of new resolvent clauses. On the contrary, being a branching algorithm, *DDS* is very opportunistic and eliminates the same variable differently in different subspaces trying to reduce the number of new resolvents to be added (if any). The lack of flexibility in variable elimination is exactly the cause of the poor scalability of the DP procedure in SAT-solving. There is no reason to believe that DP-like procedures will scale better for the harder problem of quantifier elimination.

As we mentioned above, QE algorithms based on BDDs are compositional only for particular variable orders. This limitation coupled with the necessity for a BDD to maintain one global variable order may cripple the performance of BDD based algorithms even on very simple formulas. Suppose, for instance, that H and G are compositional CNF formulas where $H = H_1 \wedge \ldots \wedge H_k$ and $G = G_1 \wedge \ldots \wedge G_m$. Suppose that variables of subformulas of H and G overlap with each other so that every variable order for which a BDD of G is small renders a large BDD for H and vice versa. Let F be a CNF formula equivalent to $(w \vee H) \wedge (\overline{w} \vee G)$ where $w \notin \mathit{Vars}(H) \cup \mathit{Vars}(G)$. (A CNF formula for, say, $w \vee H$ is trivially obtained by adding literal w to every clause of H.) Notice that F is compositional in branches $w = 0$ and $w = 1$ since $F_{w=0} = H$ and $F_{w=1} = G$. However, a BDD based QE algorithm cannot benefit from this fact because the same variable order has to be used in either branch and no order is good for both H and G. Notice, that *DDS* will not have any problem in handling formula F because *DDS* is compositional for any choice of decision variables in branches $w = 0$ and $w = 1$.

VII. Experimental Results

The objective of experiments was to compare *DDS* with other SAT-based QE algorithms. We are planning to make a comparison of *DDS* with BDD-based algorithms in the near future. In our experiments, we used a QE algorithm based on enumeration of satisfying assignments [6] (courtesy of Andy King). We will refer to this QE algorithm as *EnumSA*. We also compared *DDS* with the QE algorithm of [15] that we will call *QE-GBL*. Here *GBL* stands for *global*. Given a formula $\exists X[F]$, *QE-GBL* eliminates variables of X globally, one by one, as in the DP procedure. However, when resolving out a variable $x \in X$, *QE-GBL* adds a new resolvent to F *only if* it eliminates an $\{x\}$-removable $\{x\}$-boundary point of F. Variable x is redundant in $\exists x[F]$ if all $\{x\}$-removable $\{x\}$-boundary points of F are eliminated. *QE-GBL* does not generate so many redundant clauses as DP, but still has the flaw of eliminating variables globally.

We used *QE-GBL* for two reasons. First, *DDS* can be viewed as a branching version of *QE-GBL*. In Section VI, we argued that branching gives *DDS* more flexibility in variable elimination in comparison to procedures eliminating variables globally. So we wanted to confirm that *DDS* indeed benefited from branching. Second, one can consider *QE-GBL* as an algorithm similar to that of [18]. The latter solves $\exists x[F(x,Y)]$ by looking for a Boolean function $H(Y)$ such that $F(H(Y),Y) \equiv \exists x[F(x,Y)]$. We used *QE-GBL* to get an idea about the performance of the algorithm of [18] since it was not implemented as a stand-alone tool.Our implementation of *QE-GBL* was quite efficient. In particular, we employed Picosat [4] for finding boundary points.

TABLE I
Experiments with model checking formulas. The time limit is 1min

model che-king mode	*EnumSA*		*QE-GBL*		*DDS*	
	solved (%)	time (s.)	solved (%)	time (s.)	solved (%)	time (s.)
forward	425 (56%)	466	561 (74%)	4,865	664 (87%)	1,530
backward	97 (12%)	143	522 (68%)	2,744	563 (74%)	554

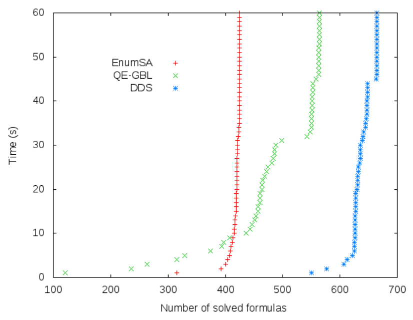

Fig. 6. Forward model checking (1 iteration)

In the first two experiments (Table I), we used the 758 model checking benchmarks of HWMCC'10 competition [27]. In the first experiment (the first line of Table I) we used *EnumSA*, *QE-GBL* and DDS to compute the set of states S^1_{reach} reachable in the first transition. In this case, CNF formula F describes the transition relation and the initial state. CNF formula G equivalent to $\exists X[F]$ specifies S^1_{reach}.

In the second experiment, (the second line of Table I) we used the same benchmarks to compute the set of "bad" states in backward model checking. In this case, F specifies the output function and the property in question. If F evaluates to 1 for some assignment $\boldsymbol{p}$ to $Vars(F)$, this property is broken and the state given by the state bits of $\boldsymbol{p}$ is bad. Formula G equivalent to $\exists X[F]$ specifies the set of all bad states (that may or may not be reachable from the initial state).

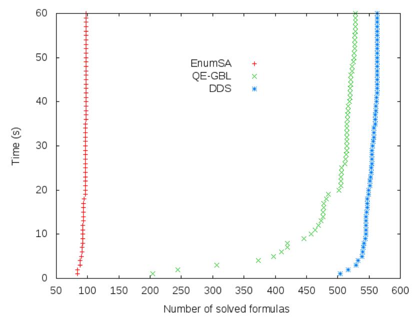

Fig. 7. Backward model checking (1 iteration)

Table I shows the comparison of the three programs with respect to the number of formulas solved, percentage of this number to the total number (758) and time taken for the *solved* problems. With 1-minute time limit, DDS solved more formulas than *EnumSA* and *QE-GBL* in forward and backward model checking. Figures 6 and 7 give the number of formulas of Table I solved by the three programs in t seconds, $0 \leq t \leq 60$. These figures show the superiority of DDS over *QE-GBL* and *EnumSA* on the set of formulas we used. The poor performance of *EnumSA* on backward model checking formulas is due to lack of constraints on next state variables. In the presence of such constraints, *EnumSA* performs much better (see below).

The size of the 1,227 formulas solved by DDS peaked at 98,105 variables, the medium size being 2,247 variables. The largest number of non-quantified (*i.e.*, state) variables was 7,880 and 541 formulas had more than 100 state variables. The size of resulting formula G peaked at 32,769 clauses, 361 resulting formulas had more than 100 clauses. We used Picosat [4] to remove redundant literals and clauses of G. Namely, for every clause C of G we checked if G was equivalent to $G \setminus \{C\}$. If so, C was removed from G. Otherwise, we tested every literal l of C if removal l from C changed the function of G. If not, l was removed from C. The total runtime for the optimization of G by Picosat was limited by 4 seconds. Overall, the resulting formulas built by DDS were smaller than those of *EnumSA* and *QE-GBL*. For instance, out of 1069 formulas solved by both DDS and *QE-GBL*, the size of G built by DDS was smaller (respectively equal or larger) in 267 (respectively 798 and 4) cases.

TABLE II
Compositionality of QE algorithms. Time limit=1hour

#co-pies	#vars, #clauses	$\|Y\|$	*EnumSA* (s.)	DDS rand (s.)	DDS (s.)
5	20,30	10	0	0.01	0.01
10	40,60	20	10.5	0.01	0.01
15	60,90	30	>1hour	0.01	0.01
500	2000,3000	1000	>1hour	1.95	0.04

In the experiments above, we did not use formula preprocessing even though it could have been beneficial. For instance, the forward model checking formulas had a lot of unit clauses encoding the initial state. The backward model checking formulas had many blocked (*i.e.*, redundant) clauses [5]. The reason is that when the *original* set of bad states is computed, the next state variables are not constrained yet. However, when we compared the three programs on preprocessed formulas we obtained similar results: DDS outperformed *EnumSA* and *QE-GBL*. In particular, we generated 189 backward model checking formulas specifying bad states after a number of iterations. The idea was to get formulas were preprocessing simplifications performing initial BCP and elimination of blocked clauses failed. With 1-minute time limit, DDS, *QE-GBL* and *EnumSA* solved 185, 163 and 149 formulas out of 189 respectively. Notice that *EnumSA* performed much better here than in the initial iteration.

The third experiment (Table II), clearly shows the compositionality of DDS in comparison to *EnumSA*. In this experiment, both programs computed the output assignments produced by a combinational circuit N composed of small *identical* circuits $N_1,\dots,N_k$ with independent sets of variables. In this case, one needs to eliminate quantifiers from

$\exists X[F]$ where $F = F_1 \wedge \ldots \wedge F_k$. CNF formula F_i specifies N_i and $Vars(F_i) \setminus X$ and $Vars(F_i) \cap X$ are the sets of output and non-output variables of N_i respectively. So a CNF formula equivalent to $\exists X[F]$ specifies the output assignments of N.

The first column of Table II shows k (the number of copies of N_i). The next two columns give the size of CNF formula F and the number of outputs in circuit N. The last three columns show the run time of *EnumSA* and two versions of DDS. In the first version, the choice of branching variables was random. In the second version, this choice was guided by the compositional structure of N. While DDS solved all the formulas easily, *EnumSA* could not finish the formulas F with $k \geq 15$ in 1 hour. Notice that DDS was able to quickly solve all the formulas even with the random choice of branching variables.

VIII. Background

The relation between a resolution proof and the process of elimination of boundary points was discussed in [14]. In terms of the present paper, [14] dealt only with a special kind of Z-boundary points of formula F where $|Z| = 1$. In the present paper, we consider the case where Z is an arbitrary subset of the set of quantified variables X of an $\exists$CNF formula $\exists X[F]$. This extension is crucial for describing the semantics of D-sequents.

The notion of D-sequents was introduced in [17]. There, we formulated a QE algorithm that branched only on quantified variables of $\exists X[F]$. This algorithm is more complex than DDS because it has to compute boundary points explicitly. At the same time, as we mentioned in the introduction, the limitation on variable order used by DDS (see Subsection V-C) is artificial. In general, to achieve the best results one has to interleave assignments to quantified and non-quantified variables. Then to reduce the number of resolvent clauses to be added one needs to compute boundary points explicitly.

As far as quantifier elimination is concerned, QE algorithms and QBF solvers can be partitioned into two categories. (Although, in contrast to a QE algorithm, a QBF-solver is a decision procedure, they both employ methods of quantifier elimination. Since this paper is focused on SAT-based solvers, we omit references to papers on QE algorithms that use BDDs [7], [8].) The members of the first category employ various techniques to eliminate quantified variables of the formula one by one in some order [26], [3], [2], [18], [1]. For example, in [18], quantified variables are eliminated by interpolation. All these solvers face the problem that we already discussed in Section VI. The necessity to eliminate a variable in one big step deprives the algorithm of flexibility and, in general, leads to generation of prohibitively large sets of clauses.

The solvers of the second category are based on enumeration of satisfying or unsatisfying assignments [23], [19], [13], [6], [25]. Since such assignments are, in general, "global" objects, it is hard for such solvers to follow the fine structure of the formula, *e.g.*, such solvers are not compositional. In a sense, DDS tries to take the best of both worlds. It branches and so can use different variable orders in different branches as the solvers of the second category. At the same time, in every branch, DDS eliminates quantified variables individually as the solvers of the first category, which makes it easier to follow the formula structure.

IX. Conclusion

We introduced Derivation of Dependency-sequents (DDS), a new method for eliminating quantifiers from a formula $\exists X[F]$ where F is a CNF formula. The essence of DDS is to add resolvent clauses to F to make the variables of X redundant. The process of making variables redundant is described by dependency sequents (D-sequents) specifying conditions under which variables of X are redundant. In contrast to methods based on the enumeration of satisfying assignments, DDS is compositional. Our experiments with a proof-of-the-concept implementation show the promise of DDS. Our future work will focus on studying various ways to improve the performance of DDS, including lifting the constraint that non-quantified variables are assigned before quantified variables and reusing D-sequents instead of discarding them after one join operation (as SAT-solvers reuse conflict clauses).

Acknowledgment

This work was funded in part by NSF grant CCF-1117184 and SRC contract 2008-TJ-1852.

References

[1] P. Abdulla, P. Bjesse, and N. Eén, "Symbolic reachability analysis based on SAT-solvers," in *Proc. of TACAS*, 2000, pp. 411–425.

[2] A. Ayari and D. Basin, "Qubos: Deciding quantified boolean logic using propositional satisfiability solvers," in *FMCAD*, 2002, pp. 187–201.

[3] A. Biere, "Resolve and expand," in *Proc. of SAT-04*, 2005, pp. 59–70.

[4] ——, "Picosat essentials," *JSAT*, vol. 4, no. 2-4, pp. 75–97, 2008.

[5] A. Biere, F. Lonsing, and M. Seidl, "Blocked clause elimination for qbf," in *Proc. of CADE*, 2011, pp. 101–115.

[6] J. Brauer, A. King, and J. Kriener, "Existential quantificationnn as incremental sat," in *Proc. of CAV-11*. Springer-Verlag, July 2011, pp. 191–207.

[7] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, August 1986.

[8] P. Chauhan, E. Clarke., S. Jha, J. Kukula, H. Veith, and D. Wang, "Using combinatorial optimization methods for quantification scheduling," in *Proc. of CHARME*, 2001, pp. 293–309.

[9] E. Clarke and A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in *Logic of Programs, Workshop*, 1982, pp. 52–71.

[10] E. Clarke, O. Grumberg, and D. Peled, *Model checking*. Cambridge, MA, USA: MIT Press, 1999.

[11] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, July 1962.

[12] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM*, vol. 7, no. 3, pp. 201–215, July 1960.

[13] M. Ganai, A. Gupta, and P. Ashar, "Efficient sat-based unbounded symbolic model checking using circuit cofactoring," in *Proc. of ICCAD*, 2004, pp. 510–517.

[14] E. Goldberg, "Boundary points and resolution," in *Proc. of SAT*. Springer-Verlag, 2009, pp. 147–160.

[15] E. Goldberg and P. Manolios, "Sat-solving based on boundary point elimination," in *Proc. of HVC-10*. Springer-Verlag, 2011, pp. 93–111.

[16] ——, "Quantifier elimination by dependency sequents," Northeastern University, Tech. Rep. arXiv:1201.5653v3 [cs.LO], 2012. [Online]. Available: http://arxiv.org/pdf/1201.5653v3

[17] ——, “Removal of quantifiers by elimination of boundary points,” Northeastern University, Tech. Rep. arXiv:1204.1746v2 [cs.LO], 2012. [Online]. Available: http://arxiv.org/pdf/1204.1746v2

[18] R. Jiang, “Quantifier elimination via functional composition,” in *Proc. of CAV ’09*. Springer, 2009, pp. 383–397.

[19] H. Jin and F. Somenzi, “Prime clauses for fast enumeration of satisfying assignments to boolean circuits,” in *Proc. of DAC*, 2005, pp. 750–753.

[20] O. Kullmann, “New methods for 3-sat decision and worst-case analysis,” *Theor. Comput. Sci.*, vol. 223, no. 1-2, pp. 1–72, Jul. 1999.

[21] J. Marques-Silva and K. Sakallah, “Grasp—a new search algorithm for satisfiability,” in *ICCAD-96*, Washington, DC, USA, 1996, pp. 220–227.

[22] K. McMillan, *Symbolic Model Checking*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[23] ——, “Applying sat methods in unbounded symbolic model checking,” in *Proc. of CAV-02*. Springer-Verlag, 2002, pp. 250–264.

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

[25] D. Plaisted, A. Biere, and Y. Zhu, “A satisfiability procedure for quantified boolean formulae,” *Discrete Appl. Math.*, vol. 130, no. 2, pp. 291–328, Aug. 2003.

[26] P. Williams, A. Biere, E. Clarke, and A. Gupta, “Combining decision diagrams and sat procedures for efficient symbolic model checking,” in *Proc. of CAV*, 2000, pp. 124–138.

[27] HWMCC-2010 benchmarks, http://fmv.jku.at/hwmcc10/benchmarks.html.