Quantifier Elimination by Dependency Sequents

Eugene Goldberg, Panagiotis Manolios

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

Abstract. We consider the problem of existential quantifier elimination for Boolean formulas in Conjunctive Normal Form (CNF). 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 a resolution-like operation called join that produces a new D-sequent from two existing D-sequents. We also show that DDS is compositional, *e.g.*, 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.

1 Introduction

In this paper, we consider the problem of eliminating existential quantifiers from Boolean CNF formulas. In the sequel, we omit the word "existential." Given a Boolean CNF formula $\exists X[F]$, the problem is to find a quantifier-free CNF formula G such that $G \equiv \exists X[F]$. We assume that the set of non-quantified variables $Vars(F) \setminus X$ is, in general, not empty. ($Vars(F)$ is the set of variables of F). So G specifies a Boolean function depending on non-quantified variables of F. We refer to this problem as the **QE problem**, where QE stands for Quantifier Elimination.

Our interest in the QE problem is twofold. First, the QE problem occurs in numerous areas of hardware/software design and verification, *e.g.*, in symbolic model checking [10,21] when computing reachable states. Second, one can argue that progress in solving the QE problem should have a deep impact on SAT-solving [13]. In particular, as McMillan pointed out, even the basic operation of resolution is related to the QE problem [20]. 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 [22,23] has led to the hunt for efficient SAT-based algorithms for the QE problem [20,17,7,12]. In this paper, we continue in this direction by introducing a resolution-based QE algorithm. 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 $\exists X[F]$ does not affect the meaning of the formula. If F depends on X, after adding to F a set of clauses implied by F, the variables of X may become redundant. If this happens, all the clauses of F depending on X can be dropped and the resulting 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 Y denote the set of non-quantified variables in $\exists X[F]$ i.e. $Y = \mathit{Vars}(F) \setminus X$. Let $\boldsymbol{y}$ be a complete assignment for Y and $F_{\boldsymbol{y}}$ denote F under assignment $\boldsymbol{y}$. 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 clauses. But a resolution derivation cannot express this fact because no clause falsified by $\boldsymbol{y}$ is implied by F.

To address the problem above, 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, the clauses containing variables of Z can be removed from F without changing the meaning of $\exists X[F]$. In particular, if the formula $F_{\boldsymbol{y}}$ is satisfiable, the D-sequent $(\exists X[F], \boldsymbol{y}) \rightarrow X$ holds. For the sake of simplicity, in the introduction, we drop the parameter of scope used in the definition of D-sequents given in Section 5.

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. Then, a solution G 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 operation *join*. Let $(\exists X[F], \boldsymbol{q_1}) \rightarrow Z$ and $(\exists X[F], \boldsymbol{q_2}) \rightarrow Z$ be D-sequents where $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ have opposite assignments to exactly one variable v. Then a new 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 $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ 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 [20,18,12,7] are not. Compositionality here means that given 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 [17,13].

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. *DDS* does not compute boundary points *explicitly*. Nevertheless, we introduce them in this paper because boundary

points provide the semantics of *DDS*. In particular, the notion of scoped variable redundancy we use in this paper can be formulated only in terms of boundary points.

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 introduce *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 2, we relate the notions of variable redundancy and boundary points. Section 3 explains the strategy of *DDS* in terms of boundary point elimination. Two simple cases of variable redundancy are described in Section 4 and D-sequents are introduced in Section 5. Sections 6 and 7 describe *DDS* and discuss its compositionality. Section 8 gives experimental results. Background is discussed in Section 9, and conclusions are presented in Section 10. In the appendix, we describe some details of the implementation of DDS we used in experiments and give proofs of propositions.

2 Redundant Variables, Boundary Points and Quantifier Elimination

The main objective of this section is to introduce the notion of redundant variables (Definition 5) and to relate it to the elimination of removable boundary points (Proposition 2).

2.1 Redundant Variables and Quantifier Elimination

In this paper, we consider a quantified CNF formula $\exists X[F]$ where $X \subseteq Vars(F)$ We will refer to such formulas as **∃CNF**. Let $\boldsymbol{q}$ be an assignment, F be a CNF formula, and C be a clause. $Vars(\boldsymbol{q})$ denotes the variables assigned in $\boldsymbol{q}$; $Vars(F)$ denotes the set of variables of F; $Vars(C)$ denotes the variables of C; and $Vars(\exists X[F]) = Vars(F) \setminus X$.

Definition 1. *Let C be a clause, F be a CNF formula, and $\boldsymbol{p}$ be an assignment such that $Vars(\boldsymbol{p}) \subseteq Vars(F)$. $C_{\boldsymbol{p}}$ is **true** if C is satisfied by $\boldsymbol{p}$; otherwise it is the clause obtained from C by removing all literals falsified by $\boldsymbol{p}$. $F_{\boldsymbol{p}}$ denotes the CNF formula obtained from F by replacing every clause C with $C_{\boldsymbol{p}}$ and then removing all the clauses that are true (i.e. satisfied by $\boldsymbol{p}$). If $Vars(F) \subseteq Vars(\boldsymbol{p})$, then $F_{\boldsymbol{p}}$ is semantically equivalent to a constant, and in the sequel, we will make use of this without explicit mention.*

Definition 2. *Let $\exists X[F]$ be an* ∃CNF*formula and $\boldsymbol{p}$ be an assignment such that $Vars(\boldsymbol{p}) \subseteq Vars(\exists X[F])$. Denote by $(\exists X[F])_{\boldsymbol{p}}$ the* ∃CNF*formula $\exists X[F_{\boldsymbol{p}}]$. If $Vars(\exists X[F]) \subseteq Vars(\boldsymbol{p})$, $Vars(\boldsymbol{p}) \cap X = \emptyset$, then $(\exists X[F])_{\boldsymbol{p}}$ is semantically equivalent to a constant, and in the sequel, we will make use of this without explicit mention.*

Definition 3. *The Quantifier Elimination (QE) problem for* ∃CNF *formula* $\exists X[F]$ *consists of finding a CNF formula* G *such that* $G \equiv \exists X[F]$. *This equivalence means that* $G_{\boldsymbol{p}} = (\exists X[F])_{\boldsymbol{p}}$ *holds for every complete assignment* $\boldsymbol{p}$ *to the variables of* $Vars(G) \cup Vars(\exists X[F])$.

Definition 4. *A clause* C *of* F *is called a* ***Z*-clause** *if* $Vars(C) \cap Z \neq \emptyset$. *Denote by* $\boldsymbol{F^Z}$ *the set of all* Z*-clauses of* F.

Definition 5. *The variables of* Z *are* ***redundant*** *in CNF formula* F *if* $F \equiv (F \setminus F^Z)$. *The variables of* Z *are* ***redundant*** *in* ∃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.*

2.2 Redundant Variables and Boundary Points

Definition 6. *Given assignment* $\boldsymbol{p}$ *and a formula* F, *we say that* $\boldsymbol{p}$ *is an* F***-point*** *(or a* ***point*** *of* F*) if* $Vars(F) \subseteq Vars(\boldsymbol{p})$.

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

Definition 7. *A point* $\boldsymbol{p}$ *of CNF formula* F *is called a* ***Z*-boundary point** *of* F *if a)* $Z \neq \emptyset$, *b)* $F_{\boldsymbol{p}} = false$; *c) every clause of* F *falsified by* $\boldsymbol{p}$ *is a* Z*-clause; d) the previous condition breaks for every proper subset of* Z.

Suppose that $\boldsymbol{p}$ is a Z-boundary point of F and F is satisfiable. If only Z variables can be flipped in $\boldsymbol{p}$, then it is at least $|Z|$ flips away from a satisfying assignment, hence the name "boundary."

Definition 8. *Given a CNF formula* F *and a* Z*-boundary point,* $\boldsymbol{p}$, *of* F*:*

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

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 the appendix of this paper.

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 Z-removable boundary points for all $Z \subseteq X$. By dropping all X-clauses of F, one produces a solution to the QE problem.

Below we introduce the notion of scoped redundancy of variables. We use the notion of scoped redundancy in the definition of dependency sequents (Section 5).

Definition 9. *Let Z be a set of variables redundant in $\exists X[F]$ where $Z \subseteq X$. We will say that the variables of Z are* ***redundant*** *in $\exists X[F]$* ***with scope*** *W where $W \supseteq Z$ if for any non-empty subset $V \subseteq Z$, the set of W-removable V-boundary points is empty. In other words, any V-boundary point of F where $V \subseteq Z$ can be turned into an assignment satisfying F by flipping only variables of W. We will say that the variables of Z are* ***locally redundant*** *in $\exists X[F]$ if the scope of their redundancy is equal to Z.*

Notice that if variables of Z are redundant in $\exists X[F]$ with scope W they are also redundant in $\exists X[F]$ in terms of Definition 5. The opposite is not true. Informally, W can be viewed as a measure of how hard it is to prove redundancy of Z. The larger W, the harder the proof. The notion of scoped redundancy is used in this paper instead of that of virtual redundancy[1] introduced in the previous version of this paper [16].

From now on, when we say that variables of Z are redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope W we will assume that $W \cap \mathit{Vars}(\boldsymbol{q}) = \emptyset$.

3 Boundary Points And Divide-And-Conquer Strategy

In this section, we provide the semantics of the QE algorithm *DDS* described in Section 6. *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 $\boldsymbol{q}$. This is done by using formula $F_{\boldsymbol{q}}$, a "local version" of F. Proposition 4 justifies proving redundancy of variables of X in $F_{\boldsymbol{q}}$ one by one.

Let $\boldsymbol{q}$ and $\boldsymbol{r}$ be assignments to a set of variables Z. Since $\boldsymbol{q}$ and $\boldsymbol{r}$ are sets of value assignments to individual variables of Z one can apply set operations to them. We will denote by $\boldsymbol{r} \subseteq \boldsymbol{q}$ the fact that $\boldsymbol{q}$ contains all the assignments $\boldsymbol{r}$. The assignment consisting of value assignments of $\boldsymbol{q}$ and $\boldsymbol{r}$ is represented as $\boldsymbol{q} \cup \boldsymbol{r}$.

---

[1] In [16], we used the notion of virtual redundancy to address the following problem. The fact that $\exists X[F_{\boldsymbol{s}}] \equiv \exists X[F_{\boldsymbol{s}} \setminus (F_{\boldsymbol{s}})^Z]$ does not imply that $\exists X[F_{\boldsymbol{q}}] \equiv \exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z]$ where $\boldsymbol{s} \subset \boldsymbol{q}$. That is redundancy of variables Z in subspace $\boldsymbol{s}$ specified by Definition 5 does not imply such redundancy in subspace $\boldsymbol{q}$ contained in subspace $\boldsymbol{s}$. The notion of virtual redundancy solves this paradox by **weakening** Definition 5. Namely, variables of Z are redundant in $\boldsymbol{q}$ even if $\exists X[F_{\boldsymbol{q}}] \not\equiv \exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z]$ but $\exists X[F_{\boldsymbol{s}}] \equiv \exists X[F_{\boldsymbol{s}} \setminus (F_{\boldsymbol{s}})^Z]$ for some $\boldsymbol{s}$ such that $\boldsymbol{s} \subset \boldsymbol{q}$. In this paper, we solve the problem above by using scoped redundancy i.e. by **strengthening** Definition 5. The trick is that we forbid to assign variables of scope W. Then (see Lemma 2 of the appendix), redundancy of Z with scope W in subspace $\boldsymbol{q}$ where $W \cap \mathit{Vars}(\boldsymbol{s}) = \emptyset$ implies redundancy of Z in any subspace $\boldsymbol{q}$ where $\boldsymbol{s} \subset \boldsymbol{q}$ if $W \cap \mathit{Vars}(\boldsymbol{q}) = \emptyset$.

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

Remark 1. Proposition 3 is not true in the opposite direction. That is, a boundary point may be X-removable in $F_{\boldsymbol{q}}$ and not X-removable in F. For instance, if $X = Vars(F)$, 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 in $\exists X[F]$. Yet $\boldsymbol{p}$ may be removable in $\exists X[F_{\boldsymbol{q}}]$ if $F_{\boldsymbol{q}}$ is unsatisfiable.

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 $\exists X[F_{\boldsymbol{q}}]$ with scope W where $Z \subseteq (X \setminus Vars(\boldsymbol{q}))$. Let a variable v of $X \setminus (Vars(\boldsymbol{q}) \cup Z)$ be locally redundant in $\exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z]$. Then the variables of $Z \cup \{v\}$ are redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope $W \cup \{v\}$.*

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

4 Two Simple Cases of Local Variable Redundancy

In this section, we describe two easily identifiable cases where variables are locally redundant (see Definition 9). These cases are specified by Propositions 5 and 6.

Definition 10. *Let C' and C'' be clauses having opposite literals of exactly one variable $v \in Vars(C') \cap Vars(C'')$. 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 11. *A variable x of a CNF formula F is called* ***blocked*** *if no two clauses of F are resolvable on x. A* ***monotone*** *variable x (literals of only one polarity of x are present in F) is a special case of a blocked variable.*

The notion of blocked variables is related to that of blocked clauses introduced in [19] (not to confuse with *blocking* clauses [20]). 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 $Vars(F)$. Let a variable v of $X \setminus Vars(\boldsymbol{q})$ be blocked in $F_{\boldsymbol{q}}$. Then v is locally 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 $Vars(F)$. Let $F_{\boldsymbol{q}}$ have an empty clause. Then the variables of $X \setminus Vars(\boldsymbol{q})$ are locally redundant in $\exists X[F_{\boldsymbol{q}}]$.*

5 Dependency Sequents (D-sequents)

In this section, we define D-sequents and introduce the operation of joining D-sequents. We also introduce the notion of composable D-sequents[2].

5.1 Definition of D-sequents

Definition 12. *Let $\exists X[F]$ be an* ∃CNF *formula. Let $\boldsymbol{q}$ be an assignment to $Vars(F)$ and Z be a subset of $X \setminus Vars(\boldsymbol{q})$. Let W be a set of variables such that $Z \subseteq W \subseteq (X \setminus Vars(\boldsymbol{q}))$. A dependency sequent (**D-sequent**) has the form $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$. It states that the variables of Z are redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope W.*

The definition above is different from those given in previous versions of this paper [15,16]. A brief discussion of this topic is given below[3].

Example 1. Consider an ∃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 locally redundant in $F_{\boldsymbol{q}}$ (Proposition 5). Hence, the D-sequent $(\exists X[F], \boldsymbol{q}, \{x\}) \rightarrow \{x\}$ holds.

According to Definition 12, a D-sequent holds with respect to a particular ∃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* ∃CNF *formula. Let $H = F \wedge G$ where $F \Rightarrow G$. Let $\boldsymbol{q}$ be an assignment to $Vars(F)$. Then if $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ holds, $(\exists X[H], \boldsymbol{q}, W) \rightarrow Z$ does too.*

The proposition below shows that it is safe to increase the scope of a D-sequent.

---

[2] As far as composability of D-sequents is concerned, we made two changes in comparison to paper [16]. First, we use term 'composable' instead of 'mergeable' and term 'compatible' instead of 'consistent'. Second, in [16] we put the discussion of composability of D-sequents into the appendix. In the current paper, we split this discussion between the main body of the paper and the appendix.

[3] In [15] we represented D-sequents in the following form $(F, q, X') \rightarrow X''$. In terms of the current paper, such a D-sequent says that the variables of X' are redundant in $\exists X[F_{\boldsymbol{q}}]$ and the variables of X'' are redundant in $\exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^{X'}]$. The flaw of this definition is that redundancy of variables of X'' is predicated on that of variables of some other set X'. To solve this problem, in [16], we changed the definition of a D-sequent representing it in the form $(\exists X[F], q) \rightarrow Z$. Such a D-sequent says that the variables of Z are redundant in $\exists X[F_{\boldsymbol{q}}]$. The drawback of such definition is that it ignores the fact that variables redundant in $\exists X[F_{\boldsymbol{q}}]$ may not be redundant in $\exists X[F_{\boldsymbol{s}}]$ where $q \subseteq s$ (see footnote 1). Definition 12 of this paper takes care of both problems above. First, redundancy of variables of Z is not predicated on that of some other set of variables. Second, by forbidding to make assignments to scope variables W we guarantee that variables redundant in $\exists X[F_{\boldsymbol{q}}]$ are redundant in $\exists X[F_{\boldsymbol{s}}]$ where $q \subseteq s$.

Proposition 8. *Let D-sequent* $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ *hold. Let* W' *be a superset of* W *where* $W' \cap Vars(\boldsymbol{q}) = \emptyset$. *Then* $(\exists X[F], \boldsymbol{q}, W') \rightarrow Z$ *holds as well.*

5.2 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.

Definition 13. *Let* $\boldsymbol{q'}$ *and* $\boldsymbol{q''}$ *be assignments in which exactly one variable* $v \in Vars(\boldsymbol{q'}) \cap Vars(\boldsymbol{q''})$ *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 9. *Let* $\exists X[F]$ *be an* $\exists$CNF *formula. Let D-sequents* $(\exists X[F], \boldsymbol{q'}, W') \rightarrow Z$ *and* $(\exists X[F], \boldsymbol{q''}, W'') \rightarrow Z$ *hold and* $(Vars(\boldsymbol{q'}) \cap W'') = (Vars(\boldsymbol{q''}) \cap W') = \emptyset$. *Let* $\boldsymbol{q'}$, $\boldsymbol{q''}$ *be resolvable on* $v \in Vars(F)$ *and* $\boldsymbol{q}$ *be the resolvent of* $\boldsymbol{q'}$ *and* $\boldsymbol{q''}$. *Then, the D-sequent* $(\exists X[F], \boldsymbol{q}, W' \cup W'') \rightarrow Z$ *holds too.*

Definition 14. *We will say that the D-sequent* $(\exists X[F], \boldsymbol{q}, W' \cup W'') \rightarrow Z$ *of Proposition 9 is produced by* ***joining D-sequents*** $(\exists X[F], \boldsymbol{q'}, W') \rightarrow Z$ *and* $(\exists X[F], \boldsymbol{q''}, W'') \rightarrow Z$ *at* v.

5.3 Composable D-sequents

In general, the fact that D-sequents $(\exists X[F], \boldsymbol{q}, W) \rightarrow \{v'\}$ and $(\exists X[F], \boldsymbol{q}, W) \rightarrow \{v''\}$ hold does not imply that $(\exists X[F], \boldsymbol{q}, W) \rightarrow \{v', v''\}$ does too. The reason is that derivation of D-sequent $(\exists X[F], \boldsymbol{q}, W) \rightarrow \{v', v''\}$ may involve recursive reasoning where $\{v'\}$-clauses are used to prove redundancy of variable v'' and vice versa. Proposition 10 below shows how to avoid recursive reasoning.

Definition 15. *Let* $\boldsymbol{q'}$ *and* $\boldsymbol{q''}$ *be assignments to a set of variables* Z. *We will say that* $\boldsymbol{q'}$ *and* $\boldsymbol{q''}$ *are* ***compatible*** *if every variable of* $Vars(\boldsymbol{q'}) \cap Vars(\boldsymbol{q''})$ *is assigned the same value in* $\boldsymbol{q'}$ *and* $\boldsymbol{q''}$.

Proposition 10. *Let* $\boldsymbol{s}$ *and* $\boldsymbol{q}$ *be assignments to variables of* F *where* $\boldsymbol{s} \subseteq \boldsymbol{q}$. *Let D-sequents* $(\exists X[F], \boldsymbol{s}, W) \rightarrow Z$ *and* $(\exists X[F \setminus F^Z], \boldsymbol{q}, \{v\}) \rightarrow \{v\}$ *hold where* $Vars(\boldsymbol{q}) \cap Z = Vars(\boldsymbol{q}) \cap W = \emptyset$. *Then D-sequent* $(\exists X[F], \boldsymbol{q}, W \cup \{v\}) \rightarrow Z \cup \{v\}$ *holds.*

Definition 16. *Let* S' *and* S'' *be D-sequents* $(\exists X[F], \boldsymbol{q'}, W) \rightarrow Z$ *and* $(\exists X[F], \boldsymbol{q''}, \{v\}) \rightarrow \{v\}$ *respectively where* $\boldsymbol{q'}$ *and* $\boldsymbol{q''}$ *are compatible assignments to* $Vars(F)$ *and* $v \notin Vars(\boldsymbol{q'}), Vars(\boldsymbol{q''}) \cap Z = \emptyset$, $Vars(\boldsymbol{q'}) \cap W = \emptyset$. *We will call* S' *and* S'' ***composable*** *if D-sequent* S *equal to* $(\exists X[F], \boldsymbol{q}, W \cup \{v\}) \rightarrow Z \cup \{v\}$ *holds where* $\boldsymbol{q} = \boldsymbol{q'} \cup \boldsymbol{q''}$. *From Proposition 10 it follows that if D-sequent* $(\exists X[F \setminus F^Z], \boldsymbol{q}, \{v\}) \rightarrow \{v\}$ *holds, then* S', S'' *are composable.*

6 Description of *DDS*

In this section, we describe a QE algorithm called *DDS* (Derivation of D-Sequents). *DDS* derives D-sequents $(\exists X[F], \boldsymbol{s}, W) \rightarrow \{x\}$ stating the redundancy of one variable of X. We will call D-sequent $(\exists X[F], \boldsymbol{s}, W) \rightarrow \{x\}$ **active** in the branch specified by assignment $\boldsymbol{q}$ if $\boldsymbol{s} \subseteq \boldsymbol{q}$ i.e. if this D-sequent provides a proof of redundancy of x in subspace $\boldsymbol{q}$. From now on, we will use a short notation of D-sequents writing $\boldsymbol{s} \rightarrow \{x\}$ instead of $(\exists X[F], \boldsymbol{s}, W) \rightarrow \{x\}$. We will assume that the parameter $\exists X[F]$ missing in $\boldsymbol{s} \rightarrow \{x\}$ is the *current* $\exists$CNF formula (with all resolvent clauses added to F so far). We will also assume that the missing parameter W is the set of variables that are currently redundant. One can omit $\exists X[F]$ from D-sequents because from Proposition 7 it follows that once D-sequent $(\exists X[F], \boldsymbol{s}, W) \rightarrow \{x\}$ is derived it holds after adding any set of resolvent clauses to F. The scope parameter W can be dropped because Proposition 8 entails that it is safe to increase the scope of a D-sequent. So one can just assume that all the D-sequents that are currently active have the same scope equal to the current set of redundant variables.

```
// Φ denotes ∃X[F], q is an assignment to Vars(F)
// Ω denotes a set of active D-sequents
DDS(Φ,q,Ω){
1   (Ω, ans, C) ← atomic_D_seqs(Φ, q, Ω);
2   if (ans = sat) return(Φ, Ω, sat);
3   if (ans = unsat) return(Φ, Ω, unsat, C);
4   v := pick_variable(F, q, Ω);
5   (Φ, Ω, ans_0, C_0) ←DDS(Φ,q ∪ {(v = 0)},Ω);
6   (Ω^sym, Ω^asym) ← split(F, Ω, v);
7   if (Ω^asym = ∅) return(Φ, Ω, ans_0, C_0);
8   Ω := Ω \ Ω^asym;
9   (Φ, Ω, ans_1, C_1) ←DDS(Φ,q ∪ {(v = 1)},Ω);
10  if ((ans_0 = unsat) and (ans_1 = unsat)){
11     C := resolve_clauses(C_0, C_1, v);
12     F := F ∧ C;
13     Ω := process_unsat_clause(Φ, C, Ω);
14     return(Φ, Ω, unsat, C);}
15  Ω := merge(Φ, q, v, Ω^asym, Ω);
16  return(Φ, Ω, sat);}
```

Fig. 1. *DDS* procedure

A description of *DDS* is given in Figure 1. *DDS* accepts an $\exists$CNF formula $\exists X[F]$ (denoted as Φ), an assignment $\boldsymbol{q}$ to $Vars(F)$ and a set Ω of active D-sequents stating redundancy of *some* variables of $X \setminus Vars(\boldsymbol{q})$ 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 $X \setminus Vars(\boldsymbol{q})$ 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 Φ, one needs to call *DDS* with $\boldsymbol{q} = \emptyset$, $\Omega = \emptyset$ and discard the X-clauses of the CNF formula F returned by *DDS*.

6.1 The Big Picture

First, *DDS* looks for variables whose redundancy is trivial to prove (lines 1-3). If some variables of $X \setminus Vars(\boldsymbol{q})$ 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 $X \setminus Vars(\boldsymbol{q})$, it derives a D-sequent $\boldsymbol{s} \rightarrow \{x\}$ where $\boldsymbol{s} \subseteq \boldsymbol{q}$. As we show in the appendix (see Lemma 7) D-sequents derived by *DDS* are composable. Thus derivation of D-sequents for individual variables also means that a D-sequent $\boldsymbol{s^*} \rightarrow (X \setminus Vars(\boldsymbol{q}))$ holds where $\boldsymbol{s^*} \subseteq \boldsymbol{q}$. So, *DDS* terminates when the QE problem is solved for Φ in subspace $\boldsymbol{q}$. The composability of D-sequents is achieved by *DDS* by guaranteeing that

- for every path of the search tree leading to a leaf, variables are proved redundant in a particular order (but for different paths the order may be different);
- all the $\{v\}$-clauses are marked as redundant and ignored as long as variable v stays redundant.

So there is no path leading to a leaf of the search tree on which recursive reasoning is employed where $\{v'\}$-clauses are used to prove redundancy of v'' and vice versa.

6.2 Building Atomic D-sequents

atomic_D_seqs$(\Phi, \boldsymbol{q}, \Omega)\{$
1 if ($\exists$ clause $C \in F$ falsif. by $\boldsymbol{q}$)$\{$
2 $\quad \Omega :=$ *process_unsat_clause*(Φ, C, Ω);
3 $\quad$ return$(\Omega, unsat, C)$;$\}$
4 Ω:=*new_redund_vars*$(\Phi,\boldsymbol{q},\Omega)$;
5 if (*all_unassgn_vars_redund*$(\Phi, \boldsymbol{q}, \Omega)$) return$(\Omega, sat)$;
6 return$(\Omega, unknown)\}$;

Fig. 2. *atomic_D_seqs* procedure

Procedure *atomic_D_seqs* is called by *DDS* to compute D-sequents for trivial cases of variable redundancy listed in Section 4. 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 $x \in X \setminus Vars(\boldsymbol{q})$ for which Ω does not contain a D-sequent yet, procedure *process_unsat_clause* generates D-sequent $\boldsymbol{s} \rightarrow \{x\}$ and adds it to Ω. Here $\boldsymbol{s}$ is the shortest assignment falsifying C. Once Ω contains a D-sequent for every variable of $X \setminus Vars(\boldsymbol{q})$, *atomic_D_seqs* terminates returning the answer *unsat*, set Ω and clause C.

If no clause of F is falsified by $\boldsymbol{q}$, 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*, 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 $\boldsymbol{s} \rightarrow \{x\}$ is built as follows. 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 for the sake of clarity that it is always clause C'. The assignment $\boldsymbol{s}$ is a subset of $\boldsymbol{q}$ guaranteeing that every clause C' remains satisfied by $\boldsymbol{s}$ or redundant in $\exists X[F_{\boldsymbol{s}}]$ and so x remains blocked in $F_{\boldsymbol{s}}$. If C' is satisfied by $\boldsymbol{q}$, then $\boldsymbol{s}$ 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, $\boldsymbol{s}$ contains all the single-variable assignments of $\boldsymbol{s^*}$ where $\boldsymbol{s^*} \rightarrow \{x^*\}$ is the D-sequent of Ω stating redundancy of x^*.

6.3 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* we give in Figure 1, the left branch always explores assignment $v = 0$ but obviously $v = 1$ can be explored first too.

To simplify making the branching variable v redundant when merging results of the left and right branches, *DDS* first assigns values to variables of Y (more details are given in Subsection 6.5). 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.

6.4 Switching from Left to Right Branch

DDS prunes big chunks of the search space by not branching on redundant variables of X. One more powerful pruning technique of *DDS* discussed in this subsection is to reduce the size of right branches.

Let $\boldsymbol{s} \rightarrow \{x\}$ 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 $\boldsymbol{s}$ 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 $\boldsymbol{s} \rightarrow \{x\}$ is still active in subspace $\boldsymbol{q_1}$. *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 $\boldsymbol{s} \rightarrow \{x\}$ *symmetric* with respect to v, if $\boldsymbol{s}$ 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 [22].

6.5 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 6.2. 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.

```
merge(Φ, q, v, Ω^asym, Ω){
1  Ω := join_D_seqs(v, Ω^asym, Ω);
2  if (v ∈ X) Ω := Ω ∪ {atomic_D_seq_for_v(F, q, v, Ω)};
3  return(Ω);}
```

Fig. 3. *merge* procedure

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 6.4). 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. Procedure *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 is symmetric in v. D-sequent S_1 is replaced in Ω with S.

Let us consider the case[4] where S_1 is symmetric in v. If $F_{\boldsymbol{q_0}}$ was unsatisfiable, then S_1 remains in Ω untouched. Otherwise, *join_D_seqs* does the following. Let S_1 be equal to $\boldsymbol{s} \to \{x\}$. First, the right branch assignment $v = 1$ is added to $\boldsymbol{s}$, which makes S_1 asymmetric in v. Then S_1 is joined with S_0 at v to produce a new D-sequent S that is symmetric in v. S replaces S_1 in Ω. The reason one cannot simply keep S_1 in Ω untouched is as follows. As we mentioned above, the composability of D-sequents built by *DDS* is based on the assumption that for every path of the search tree, variables are proved redundant in a particular order. Using D-sequent S_1 in subspace $\boldsymbol{q}$ would violate this assumption and so would break the composability of D-sequents.

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 6.3, 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. So, *merge* builds an atomic D-sequent S stating the redundancy of v as described in Subsection 6.2 and adds it to Ω (line 2). Then *merge* terminates returning Ω.

6.6 Correctness of *DDS*

Let *DDS* be called on formula $\Phi = \exists X[F]$ 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 $\emptyset \to \{x\}$ is derived; e) the D-sequents derived by *DDS* are composable, which implies that the D-sequent $\emptyset \to X$ holds for the formula $\exists X[F]$ returned by *DDS*.

Proposition 11. *DDS is sound and complete.*

6.7 A Run of *DDS* on a Simple Formula

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, $\mathit{DDS}_{(y_1=1,y_2=0)}$ 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

[4] The description of this case given in [16] says that if S_1 is symmetric in v, it remains in Ω untouched. It is an error because, as we mentioned above, the set of D-sequents produced for subspace $\boldsymbol{q}$ may turn out to be uncomposable.

$DDS_{(\emptyset)}$. The work of *DDS* is shown in Figures 4, 5 used below to illustrate various aspects of *DDS*.

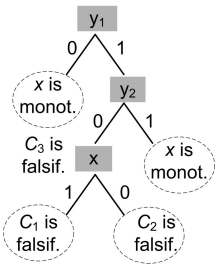

Fig. 4. Search tree built by *DDS*

Branching variables. Figure 4 shows a search tree built by *DDS*. Recall that *DDS* branches on variables of $Vars(F) \setminus X = \{y_1, y_2\}$ before those of X (see Subsection 6.3).

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 6.5, 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.

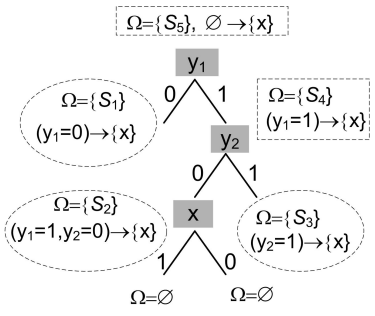

Fig. 5. Derivation of D-sequents

Generation of atomic D-sequents. Figure 5 describes derivation of D-sequents for the search tree of Figure 4. The atomic D-sequents are shown in dotted ovals. (Dotted boxes show D-sequents obtained by the join operation.) 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.

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 becomes inactive 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 $C_3 \equiv \exists X[C_1 \wedge C_2]$.

7 Compositionality of *DDS*

Let $F = F_1 \wedge \ldots \wedge F_k$ where $Vars(F_i) \cap Vars(F_j) = \emptyset$, $i \neq j$. 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 $G_1 \wedge \ldots \wedge G_k$.

Our interest in compositional QE algorithms is motivated as follows. First, a non-compositional algorithm has poor scalability. Second, even if the *original* formula F is not a conjunction of independent subformulas, such subformulas may appear in subspaces of the search space during branching. Notice that a QE algorithm that resolves out variables one by one as in the DP procedure [11] is compositional. (Clauses of F_i and F_j, $i \neq j$ cannot be resolved with each other). However, such an algorithm cannot take into account subtle properties of the formula and hence may have abysmal performance. Suppose, for example, that F does not have independent subformulas but such subformulas appear in subspaces $x = 0$ and $x = 1$ where $x \in X$. A compositional *branching* QE algorithm can make use of this fact in contrast to its counterpart eliminating quantified variables *globally* i.e. for all subspaces at once.

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 [8] 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 12 (compositionality of DDS). *Let T be the search tree built by DDS when solving the QE problem $\exists X[F_1 \wedge \ldots \wedge F_k]$ $Vars(F_i) \cap Vars(F_j) = \emptyset$, $i \neq j$. Let $X_i = X \cap Vars(F_i)$ and $Y_i = Vars(F_i) \setminus X$. The size of T in the number of nodes is bounded by $|Vars(F)| \cdot (\eta(X_1 \cup Y_1) + \ldots + \eta(X_k \cup Y_k))$ 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 12 is proved for a slightly modified version of *DDS* (see the appendix of this paper). 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 $\boldsymbol{s} \rightarrow x$ where $Vars(\boldsymbol{s}) \cup \{x\} \subseteq Vars(F_i)$. The only exception are D-sequents derived when the current assignment falsifies a clause of F. This exception is the reason why the compositionality of *DDS* is not ideal.

8 Experimental Results

We compared *DDS* with a QE algorithm based on enumeration of satisfying assignments [7] (courtesy of Andy King). We will refer to this QE algorithm as *EnumSA*. We also compared *DDS* with the QE algorithm of [13] that we will call *QE-GBL*. 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.

Table 1. *Experiments with model checking formulas. The time limit is 1min*

model checking 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

We used *QE-GBL* for two reasons. First, *DDS* can be viewed as a branching version of *QE-GBL*. So it is interesting to check if branching is beneficial for QE algorithms. Second, one can consider *QE-GBL* as an algorithm similar to that of [17]. 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 [17] since it was not implemented as a stand-alone tool.

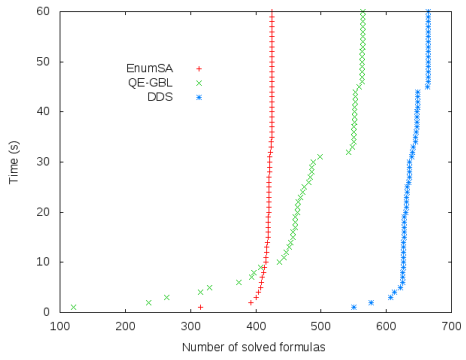

Fig. 6. Forward model checking (1 iteration)

Our implementation of *QE-GBL* was quite efficient. In particular, we employed Picosat [5] for finding boundary points. On the other hand, in experiments, we used a very simple, proof-of-the-concept implementation of *DDS* . More details about this implementation can be found in the appendix of this paper.

In the first two experiments (Table 1), we used the 758 model checking benchmarks of HWMCC'10 competition [26]. In the first experiment (the first line of Table 1) 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 1) 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).

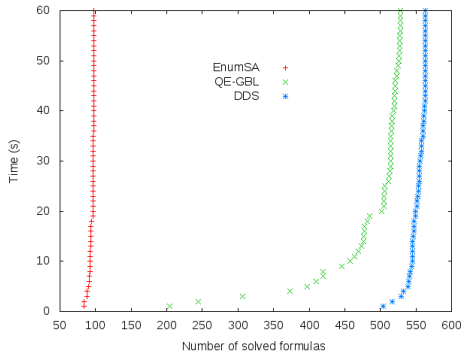

Fig. 7. Backward model checking (1 iteration)

Table 1 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 1 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 constrains 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 [5] to remove redundant literals and clauses of G with the time limit of 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 2. *Applying QE algorithms to conjunction of independent formulas. The time limit is 1 hour*

#copies	(#vars, #clauses)	$\|Y\|$	*EnumSA* (s.)	*DDS* rand (s.)	*DDS* (s.)
5	(20,30)	10	0	0.01	0.01
10	(40,60)	20	10.46	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 [4]. 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 2), 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 $N_1, \ldots, N_k$ 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 2 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.

9 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 $\exists$CNFformula $\exists X[F]$. This extension is crucial for describing the semantics of D-sequents.

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. For the lack of space, we omit references to papers on QE algorithms that use BDDs [8,9].) The members of the first category employ various techniques to eliminate quantified variables of the formula one by one in some order [25,6,2,17,1]. For example, in [17], quantified variables are eliminated by interpolation. All these solvers face the same problem: there may not exist a good *single* order for variable elimination, which, may lead to exponential growth of the size of intermediate formulas. In Subsection 7, we already gave an example of this problem. Here is one more. Let $\boldsymbol{q}$ be an assignment to variables of F. If formula $F_{\boldsymbol{q}}$ has unit clauses, the variables of such clauses can be eliminated by

unit resolution, *i.e.*, BCP. In a sense, unit resolution eliminates variables of $F_{\boldsymbol{q}}$ in a natural order. However, natural orders in formulas $F_{\boldsymbol{q'}}$ and $F_{\boldsymbol{q''}}$ of different branches $\boldsymbol{q'}$ and $\boldsymbol{q''}$ may be incompatible.

The solvers of the second category are based on enumeration of satisfying or unsatisfying assignments [20,18,12,7,24]. 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.

10 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).

11 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. Een, "Symbolic reachability analysis based on SAT-solvers", in *Proc. TACAS-2000*, pp. 411-425.
2. A.Ayari and D.Basin,"QUBOS: Deciding quantified Boolean logic using propositional satisfiability solvers", in *Proc. FMCAD-2002*,pp.187-201.
3. L.Bachmair and H. Ganzinger. "Resolution theorem proving", *Handbook of automated reasoning*, A.Robinson, A.Voronkov, Eds., Chap. 2, vol. 1, pp. 19-99, Elsevier Sci. Publ., 2001.
4. A. Biere, F. Lonsing, M. Seidl. Blocked Clause Elimination for QBF, in *Proc. CADE-2011*, LNCS, vol. 6803, pp. 101-115.
5. A.Biere,"PicoSAT essentials", *JSAT*, vol.4, no.2-4, pp.75-97, 2008.
6. A.Biere, "Resolve and expand", in *Proc. SAT-2004*, pp. 238-246.
7. J.Brauer, A. King, and J. Kriener, "Existential quantification as incremental SAT", in *Proc. CAV-2011*, pp. 191-207.

8. R.Bryant,"Graph-based algorithms for Boolean function manipulation", *IEEE Trans. on Computers*, vol.C-35, no.8, pp.677-691, 1986.
9. P. Chauhan, E. M. Clarke, S. Jha, J.H. Kukula, H. Veith, D. Wang, "Using Combinatorial Optimization Methods for Quantification Scheduling", in *Proc. CHARME 2001*, pp. 293-309.
10. E.Clarke, O. Grumberg, and D. Peled. *Model checking*, MIT Press, 2000.
11. M.Davis, and H.Putnam, "A Computing procedure for quantification theory", *J. ACM*, vol.7, no.3, pp.201-215, July, 1960.
12. M.Ganai, A.Gupta, and P. Ashar, "Efficient SAT-based unbounded symbolic model checking using circuit cofactoring", in *Proc. ICCAD-2004*, pp.510-517.
13. E.Goldberg, and P.Manolios, "SAT-solving Based on Boundary Point Elimination", in *Proc. HVC-2010*, LNCS vol.6504, pp.93-111.
14. E.Goldberg,"Boundary points and resolution", in *Proc. SAT-2009*, LNCS vol.5584, pp.147-160.
15. E.Goldberg, P.Manolios, "*Quantifier Elimination by Dependency Sequents arXiv:1201.5653v1 [cs.LO]*".
16. E.Goldberg, P.Manolios, "*Quantifier Elimination by Dependency Sequents arXiv:1201.5653v3 [cs.LO]*".
17. J. R. Jiang. "Quantifier Elimination via Functional Composition", in *Proc. CAV-2009*, pp. 383-397.
18. H.Jin, and F.Somenzi,"Prime clauses for fast enumeration of satisfying assignments to Boolean circuits", in *Proc. DAC-2005*, pp. 750-753.
19. O. Kullmann, "*New Methods for 3-SAT Decision and Worst-case Analysis*", Theor. Comput. Sci., vol. 223, no. 1-2, 1999, pp. 1-72.
20. K.McMillan,"Applying SAT methods in unbounded symbolic model checking", in *Proc. CAV-2002*, pp.250-264.
21. K.McMillan, *Symbolic model checking*, Kluwer Academic Publishers, 1993.
22. J. Marques-Silva and K. Sakallah, "GRASP: A mew search algorithm for satisfiability", in *Proc. ICCAD-1996*, pp. 220-227.
23. M.Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S.Malik, "*Chaff: Engineering an Efficient SAT-solver*", DAC 2001, pp. 530-535.
24. D.Plaisted, A.Biere, and Y.Zhu, "A satisfiability procedure for quantified Boolean formulae", in *Discrete Appl. Math.*,vol.130,no.2,pp.291-328.
25. P. Williams, A. Biere, E. Clarke, and A. Gupta, "Combining decision diagrams and SAT procedures for efficient symbolic model checking", in *Proc. CAV-2000*,pp.124-138.
26. HWMCC-2010 benchmarks, http://fmv.jku.at/hwmcc10/benchmarks.html

Appendix

The appendix is structured as follows. In the first section, we give some details of the implementation of *DDS* we used in experiments. In the following sections we provide proofs[5] of the propositions listed in the paper. We also give proofs of lemmas that are used in the proofs of propositions. The numbering of propositions in the appendix is the same as in the main body of the paper.

[5] The proofs of this paper are similar to those of [16]. We changed only the parts affected by using the notion of scoped redundancy of variables (see Section 3).

Some Implementation Details

In this section, we describe some features of the implementation of *DDS* we used in experiments. We will refer to this implementation as DDS_{impl}.

- In Figure 1, *DDS* is described in terms of recursive calls. It is more convenient, to consider DDS_{impl} as building a search tree. Let n be the node of the search tree built by DDS_{impl} at which a variable v of $Vars(F)$ is assigned. Then the depth $Depth(n)$ of n is equal to the recursion depth at which variable v is assigned by *DDS*.
- In DDS_{impl}, we followed the common practice of using stack for implementing branching algorithms. When a new node n of the search tree is created, all the relevant information about n is pushed on the stack. When backtracking from node n, all the information about n is popped off the stack.
- To make the code of DDS_{impl} easy to modify, we have not implemented optimization techniques like using watched literals to speed up BCP, special representation of two-literal clauses and so on.
- In Figure 1, a D-sequent depending on an assignment to the branching variable is discarded when the current *DDS* call terminates. On the other hand, keeping such D-sequents may be very beneficial. The reason is that after getting broken, a D-sequent S stating redundancy of $x \in X$ may become active again in a different part of the search space. S can be used in that part of the space to avoid branching on x. This is similar to reusing conflict clauses to avoid entering the parts of the search space already proved unsatisfiable. Nevertheless, to keep DDS_{impl} as simple as possible, D-sequent reusing has not been implemented.
- In Figure 1, if both branches are unsatisfiable, *DDS* adds the resolvent C of clauses C_0 and C_1 falsified in left and right branches respectively. Recall that C is falsified by the current assignment $\boldsymbol{q}$. Let $Depth(C)$ describe the maximum recursion depth at which an assignment of $\boldsymbol{q}$ falsifying a literal of C is made. In DDS_{impl}, clause C is not added to F if another clause C' falsified by $\boldsymbol{q}$ can be derived later such that $Depth(C') < Depth(C)$. This is similar to the conflict clause generation procedure of a SAT-solver. In such a procedure, all intermediate resolvents produced in the course of generation of a conflict clause are discarded.

  The condition above means that DDS_{impl} keeps a resolvent clause C only if it is empty or if in the node of the search tree located at depth $Depth(C)$

  - the left branch is currently explored or
  - the right branch is currently explored and formula F was *satisfiable* in the left branch.

  In terms of a conflict clause generation procedure, DDS_{impl} backtracks to the closest *decision* assignment of the current path of the search tree or to the root of the tree if the current path does not have any decision assignments.

Propositions of Section 2: Redundant Variables, Boundary Points and Quantifier Elimination

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.*

Proof: If part. Assume the contrary. That is $\boldsymbol{p}$ is not removable while no satisfying assignment can be obtained from $\boldsymbol{p}$ by changing only assignments to variables of X. Let $Y = \mathit{Vars}(F) \setminus X$ and C be a clause consisting only of variables of Y and falsified by $\boldsymbol{p}$. Since $\boldsymbol{p}$ is not removable, clause C is not implied by F. This means that there is an assignment $\boldsymbol{s}$ that falsifies C and satisfies F. By construction, $\boldsymbol{s}$ and $\boldsymbol{p}$ have identical assignments to variables of Y. Thus, $\boldsymbol{s}$ can be obtained from $\boldsymbol{p}$ by changing only values of variables of X. Contradiction.

Only if part. Assume the contrary. That is $\boldsymbol{p}$ is removable but one can obtain an assignment $\boldsymbol{s}$ satisfying F from $\boldsymbol{p}$ by changing only values of variables of X. Since $\boldsymbol{p}$ is removable, there is a clause C that is implied by F and falsified by $\boldsymbol{p}$ and that depends only of variables of Y. Since $\boldsymbol{s}$ and $\boldsymbol{p}$ have identical assignments to variables of Y, point $\boldsymbol{s}$ falsifies C. However, since $\boldsymbol{s}$ satisfies F, this means that C is not implied by F. Contradiction □

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, $W \subseteq Z$.*

Proof: Let H denote $F \setminus F^Z$ and Y denote $\mathit{Vars}(F) \setminus X$. Given a point $\boldsymbol{p}$, let $(\boldsymbol{x},\boldsymbol{y})$ specify the assignments of $\boldsymbol{p}$ to the variables of X and Y respectively.
If part. Assume the contrary, *i.e.*, there is an X-removable W-boundary point $\boldsymbol{p}$=$(\boldsymbol{x},\boldsymbol{y})$ of F where $W \subseteq Z$ but the variables of Z are redundant and hence $\exists X[F] \equiv \exists X[H]$. Since $\boldsymbol{p}$ is a boundary point, $F(\boldsymbol{p}) = 0$. Since $\boldsymbol{p}$ is removable, $(\exists X[F])_{\boldsymbol{y}} = 0$. On the other hand, since $\boldsymbol{p}$ falsifies only W-clauses of F it satisfies H. Hence $(\exists X[H])_{\boldsymbol{y}} = 1$ and so $(\exists X[F])_{\boldsymbol{y}} \neq (\exists X[H])_{\boldsymbol{y}}$. Contradiction.

Only if part. Assume the contrary, *i.e.*, the variables of Z are not redundant (and hence $\exists X[F] \not\equiv \exists X[H]$) and there does not exist an X-removable W-boundary point of F, $W \subseteq Z$. Let $\boldsymbol{y}$ be an assignment to Y such that $(\exists X[F])_{\boldsymbol{y}} \neq (\exists X[H])_{\boldsymbol{y}}$. One has to consider the following two cases.

- $(\exists X[F])_{\boldsymbol{y}} = 1$ and $(\exists X[H])_{\boldsymbol{y}} = 0$. Then there exists an assignment $\boldsymbol{x}$ to X such that $(\boldsymbol{x},\boldsymbol{y})$ satisfies F. Since every clause of H is in F, formula H is also satisfied by $\boldsymbol{p}$. Contradiction.
- $(\exists X[F])_{\boldsymbol{y}} = 0$ and $(\exists X[H])_{\boldsymbol{y}} = 1$. Then there exists an assignment $\boldsymbol{x}$ to variables of X such that $(\boldsymbol{x},\boldsymbol{y})$ satisfies H. Since $F_{\boldsymbol{y}} \equiv 0$, point $(\boldsymbol{x},\boldsymbol{y})$ falsifies F. Since $H(\boldsymbol{p}) = 1$ and every clause of F that is not in H is an Z-clause, $(\boldsymbol{x},\boldsymbol{y})$ is a W-boundary point of F where $W \subseteq Z$. Since $F_{\boldsymbol{y}} \equiv 0$, $(\boldsymbol{x},\boldsymbol{y})$ is an X-removable W-boundary point of F. Contradiction □

Propositions of Section 3: Boundary Points And Divide-And-Conquer Strategy

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

Proof: Let Y denote $Vars(F) \setminus X$. Assume the contrary. That is $\boldsymbol{p}$ is removable in $\exists X[F]$ but is not removable in $\exists X[F_{\boldsymbol{q}}]$. The fact that $\boldsymbol{p}$ is removable in $\exists X[F]$ means that there is a clause C implied by F and falsified by $\boldsymbol{p}$ that consists only of variables of Y. Since $\boldsymbol{p}$ is not removable in $\exists X[F_{\boldsymbol{q}}]$, from Proposition 1 it follows that an assignment $\boldsymbol{s}$ satisfying $F_{\boldsymbol{q}}$ can be obtained from $\boldsymbol{p}$ by changing only values of variables of $X \setminus Vars(\boldsymbol{q})$. By construction, $\boldsymbol{p}$ and $\boldsymbol{s}$ have identical assignments to variables of Y. So $\boldsymbol{s}$ has to falsify C. On the other hand, by construction, $\boldsymbol{q} \subseteq \boldsymbol{s}$. So, the fact that $\boldsymbol{s}$ satisfies $F_{\boldsymbol{q}}$ implies that $\boldsymbol{s}$ satisfies F too. Since $\boldsymbol{s}$ falsifies C and satisfies F the former cannot be implied by the latter. Contradiction □

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 $\exists X[F_{\boldsymbol{q}}]$ with scope W where $Z \subseteq (X \setminus Vars(\boldsymbol{q}))$. Let a variable v of $X \setminus (Vars(\boldsymbol{q}) \cup Z)$ be locally redundant in $\exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z]$. Then the variables of $Z \cup \{v\}$ are redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope $W \cup \{v\}$.*

Proof: Assume the contrary, that is the variables of $Z \cup \{v\}$ are not redundant with scope $W \cup \{v\}$. Then from Definition 9 it follows that $F_{\boldsymbol{q}}$ has a Z'-boundary point $\boldsymbol{p}$ where $Z' \subseteq Z \cup \{v\}$, $\boldsymbol{q} \subseteq \boldsymbol{p}$ that is $(W \cup \{v\})$-removable in $F_{\boldsymbol{q}}$. Let us consider the two possible cases:

- $v \notin Z'$ (and so $Z' \subseteq Z$). Since $\boldsymbol{p}$ is $(W \cup \{v\})$-removable in $F_{\boldsymbol{q}}$, it is also W-removable in $F_{\boldsymbol{q}}$. Hence, the variables of Z are not redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope W. Contradiction.
- $v \in Z'$ (and so $Z' \not\subseteq Z$). Then $\boldsymbol{p}$ is a $\{v\}$-boundary point of $F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z$. Indeed, there has to be a clause C of $F_{\boldsymbol{q}}$ falsified by $\boldsymbol{p}$ that contains variable v. Otherwise, condition d) of the definition of a boundary point is broken because v can be removed from Z' (see Definition 7) .

  Let P denote the set of all points obtained from $\boldsymbol{p}$ by flipping values of variables of $W \cup \{v\}$. Let us consider the following two possibilities.
  - Every point of P falsifies $F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z$. This means that the point $\boldsymbol{p}$ is a $\{v\}$-removable $\{v\}$- boundary point of $F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z$. So v is not locally redundant in $\exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z]$. Contradiction.
  - A point $\boldsymbol{d}$ of P satisfies $F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z$. Let us consider the following two cases.
    - $\boldsymbol{d}$ satisfies $F_{\boldsymbol{q}}$. This contradicts the fact that $\boldsymbol{p}$ is a $(W \cup \{v\})$-removable Z'-boundary point of $F_{\boldsymbol{q}}$. (By flipping variables of $W \cup \{v\}$ one can obtain a point satisfying $F_{\boldsymbol{q}}$.)

- $\boldsymbol{d}$ falsifies some clauses of $F_{\boldsymbol{q}}$. Since $F_{\boldsymbol{q}}$ and $F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z$ are different only in Z-clauses, $\boldsymbol{d}$ is a Z''-boundary point of $F_{\boldsymbol{q}}$ where $Z'' \subseteq Z$. By construction, $\boldsymbol{p}$ and $\boldsymbol{d}$ are different only in values of variables from $W \cup \{v\}$. So, the fact that $\boldsymbol{p}$ is a $(W \cup \{v\})$-removable Z'-boundary point of $F_{\boldsymbol{q}}$ implies that $\boldsymbol{d}$ is a W-removable Z''-boundary point of $F_{\boldsymbol{q}}$. So the variables of Z are not redundant in $F_{\boldsymbol{q}}$ with scope W. Contradiction □

Propositions of Section 4: Two Simple Cases of Local Variable Redundancy

Lemma 1. *Let $\boldsymbol{p}$ be a $\{v\}$-boundary point of CNF formula $G(Z)$ where $v \in Z$. Let $\boldsymbol{p'}$ be obtained from $\boldsymbol{p}$ by flipping the value of v. Then $\boldsymbol{p'}$ either satisfies G or it is also a $\{v\}$-boundary point of G.*

Proof: Assume the contrary, *i.e.*, $\boldsymbol{p'}$ falsifies a clause C of G that does not have a literal of v. (And so $\boldsymbol{p'}$ is neither a satisfying assignment nor a $\{v\}$-boundary point of G.) Since $\boldsymbol{p}$ is different from $\boldsymbol{p'}$ only in the value of v, it also falsifies C. Then $\boldsymbol{p}$ is not a $\{v\}$-boundary point of G. Contradiction □

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

Proof: Assume the contrary i.e. v is not locally redundant in $\exists X[F_{\boldsymbol{q}}]$. Then there is a v-removable $\{v\}$-boundary point $\boldsymbol{p}$ of $F_{\boldsymbol{q}}$. Note that the clauses of $F_{\boldsymbol{q}}$ falsified by $\boldsymbol{p}$ have the same literal $l(v)$ of variable v. Let $\boldsymbol{p'}$ be the point obtained from $\boldsymbol{p}$ by flipping the value of v. According to Lemma 1, one needs to consider only the following two cases.

- $\boldsymbol{p'}$ satisfies $F_{\boldsymbol{q}}$. Since $\boldsymbol{p'}$ is obtained from $\boldsymbol{p}$ by changing only variable v, $\boldsymbol{p}$ is not $\{v\}$-removable in $F_{\boldsymbol{q}}$. Contradiction.
- $\boldsymbol{p'}$ falsifies only the clauses of $F_{\boldsymbol{q}}$ with literal $\overline{l(v)}$. (Point $\boldsymbol{p'}$ cannot falsify a clause with literal $l(v)$.) Then there is a pair of clauses C and C' of $F_{\boldsymbol{q}}$ falsified by $\boldsymbol{p}$ and $\boldsymbol{p'}$ respectively that have opposite literals only of variable v. Hence v is not a blocked variable of $F_{\boldsymbol{q}}$. Contradiction □

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

Proof: Let X' denote the set $X \setminus Vars(\boldsymbol{q})$. Assume the contrary i.e. the variables of X' are not locally redundant in $\exists X[F_{\boldsymbol{q}}]$. Then there is an X'-removable Z-boundary point where $Z \subseteq X'$. However, the set of Z-boundary points of $F_{\boldsymbol{q}}$ is empty. Indeed, on the one hand, $F_{\boldsymbol{q}}$ contains an empty clause C that is falsified by any point. On the other hand, according to Definition 7, if $\boldsymbol{p}$ is a Z-boundary point, then Z is a non-empty set that has to contain at least one variable of every clause falsified by $\boldsymbol{p}$, in particular, a variable of clause C □

Propositions of Section 5: Dependency Sequents (D-sequents)

Proposition 7. *Let* $\exists X[F]$ *be an* $\exists$CNF *formula. Let* $H = F \wedge G$ *where* F *implies* G. *Let* $\boldsymbol{q}$ *be an assignment to* $Vars(F)$. *Then if* $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ *holds, the D-sequent* $(\exists X[H], \boldsymbol{q}, W) \rightarrow Z$ *does too.*

Proof: Assume the contrary, *i.e.*, $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ holds but $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ does not. According to Definition 12, this means that variables of Z are not redundant in $\exists X[H_{\boldsymbol{q}}]$ with scope W. That is, there is a W-removable Z'-boundary point $\boldsymbol{p}$ of $H_{\boldsymbol{q}}$ where $Z' \subseteq Z$. The fact that the variables of Z are redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope W means that $\boldsymbol{p}$ is not a W-removable Z''-boundary point of $F_{\boldsymbol{q}}$ where $Z'' \subseteq Z$. This can happen for the following three reasons.

- $\boldsymbol{p}$ satisfies $F_{\boldsymbol{q}}$. Then it also satisfies $H_{\boldsymbol{q}}$ and hence cannot be a boundary point of $H_{\boldsymbol{q}}$. Contradiction.
- $\boldsymbol{p}$ is not a Z''-boundary point of $F_{\boldsymbol{q}}$ where $Z'' \subseteq Z$. That is $\boldsymbol{p}$ falsifies a clause C of $F_{\boldsymbol{q}}$ that does not contain a variable of Z. Since $H_{\boldsymbol{q}}$ also contains C, point $\boldsymbol{p}$ cannot be a Z'-boundary point of $H_{\boldsymbol{q}}$ where $Z' \subseteq Z$. Contradiction.
- $\boldsymbol{p}$ is a Z''-boundary point of $F_{\boldsymbol{q}}$ where $Z'' \subseteq Z$ but it is not W-removable in $F_{\boldsymbol{q}}$. This means that one can obtain a point $\boldsymbol{s}$ satisfying $F_{\boldsymbol{q}}$ by flipping values of variables of W in $\boldsymbol{p}$. Since $\boldsymbol{s}$ also satisfies $H_{\boldsymbol{q}}$, one has to conclude that $\boldsymbol{p}$ is not a W-removable point of $H_{\boldsymbol{q}}$. Contradiction □

Proposition 8. *Let D-sequent* $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ *hold. Let* W' *be a superset of* W *where* $W' \cap Vars(\boldsymbol{q}) = \emptyset$. *Then* $(\exists X[F], \boldsymbol{q}, W') \rightarrow Z$ *holds as well.*

Proof: Assume that $(\exists X[F], \boldsymbol{q}, W') \rightarrow Z$ does not hold. Then there is a V-boundary point $\boldsymbol{p}$ of $F_{\boldsymbol{q}}$ where $V \subseteq Z$ that is W'-removable in $F_{\boldsymbol{q}}$. Since $W \subseteq W'$, point $\boldsymbol{p}$ is also W-removable. This means that $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ does not hold. Contradiction □

Proposition 9. *Let* $\exists X[F]$ *be an* $\exists$CNF *formula. Let D-sequents* $(\exists X[F], \boldsymbol{q'}, W')$ $\rightarrow Z$ *and* $(\exists X[F], \boldsymbol{q''}, W'') \rightarrow Z$ *hold and* $(Vars(\boldsymbol{q'}) \cap W'') = (Vars(\boldsymbol{q''}) \cap W') = \emptyset$. *Let* $\boldsymbol{q'}$, $\boldsymbol{q''}$ *be resolvable on* $v \in Vars(F)$ *and* $\boldsymbol{q}$ *be the resolvent of* $\boldsymbol{q'}$ *and* $\boldsymbol{q''}$. *Then, the D-sequent* $(\exists X[F], \boldsymbol{q}, W' \cup W'') \rightarrow Z$ *holds too.*

Proof: Assume the contrary, that is D-sequent $(\exists X[F], \boldsymbol{q}, W' \cup W'') \rightarrow Z$ does not hold and so the variables of Z are not redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope $W' \cup W''$. Then there is a Z^*-boundary point $\boldsymbol{p}$ where $Z^* \subseteq Z$ and $\boldsymbol{q} \subseteq \boldsymbol{p}$ that is $(W' \cup W'')$-removable in $F_{\boldsymbol{q}}$. By definition of $\boldsymbol{q}$, the fact that $\boldsymbol{q} \subseteq \boldsymbol{p}$ implies that $\boldsymbol{q'} \subseteq \boldsymbol{p}$ or $\boldsymbol{q''} \subseteq \boldsymbol{p}$. Assume, for instance, that $\boldsymbol{q'} \subseteq \boldsymbol{p}$. The fact that $\boldsymbol{p}$ is a Z^*-boundary point of $F_{\boldsymbol{q}}$ implies that $\boldsymbol{p}$ is also a Z^*-boundary point of $F_{\boldsymbol{q'}}$. Since $\boldsymbol{p}$ is $(W' \cup W'')$-removable in $F_{\boldsymbol{q}}$ it is also W'-removable in $F_{\boldsymbol{q'}}$. So the variables of Z are not redundant in $F_{\boldsymbol{q'}}$ with scope W' and D-sequent $(\exists X[F], \boldsymbol{q'}, W') \rightarrow Z$ does not hold. Contradiction □

Lemma 2. *Let D-sequent* $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ *hold and* $\boldsymbol{r}$ *be an assignment such that* $\boldsymbol{q} \subseteq \boldsymbol{r}$ *and* $Vars(\boldsymbol{r}) \cap W = \emptyset$. *Then D-sequent* $(\exists X[F], \boldsymbol{r}, W) \rightarrow Z$ *holds too.*

Proof: Assume the contrary i.e. the variables of Z are not redundant in $F_{\boldsymbol{r}}$ with scope W. Then there is a Z'-boundary point $\boldsymbol{p}$ where $Z' \subseteq Z$ that is W-removable in $F_{\boldsymbol{r}}$. Note that $\boldsymbol{p}$ is also a Z'-boundary point of $F_{\boldsymbol{q}}$ and it is also W-removable in $F_{\boldsymbol{q}}$. This implies that the variables of Z are not redundant in $F_{\boldsymbol{q}}$ with scope W. Contradiction.

Proposition 10. *Let* $\boldsymbol{s}$ *and* $\boldsymbol{q}$ *be assignments to variables of* F *where* $\boldsymbol{s} \subseteq \boldsymbol{q}$. *Let D-sequents* $(\exists X[F], \boldsymbol{s}, W) \rightarrow Z$ *and* $(\exists X[F \setminus F^Z], \boldsymbol{q}, \{v\}) \rightarrow \{v\}$ *hold where* $Vars(\boldsymbol{q}) \cap Z = Vars(\boldsymbol{q}) \cap W = \emptyset$. *Then D-sequent* $(\exists X[F], \boldsymbol{q}, W \cup \{v\}) \rightarrow Z \cup \{v\}$ *holds.*

Proof: From Lemma 2 it follows that $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ holds. Proposition 4 implies that the variables of $Z \cup \{v\}$ are redundant in $F_{\boldsymbol{q}}$ with scope $W \cup \{v\}$. Hence D-sequent $(\exists X[F], \boldsymbol{q}, W \cup \{v\}) \rightarrow Z \cup \{v\}$ holds.

Proposition of Section 6: Description of *DDS*

The objective of this Section is to prove the correctness of *DDS* (Proposition 11). To reach this objective, we need to introduce a few new definitions and prove several lemmas.

Definition 17. *Let* $\exists X[F]$ *be an* $\exists$CNF*formula,* $\boldsymbol{q}$ *be an assignment to* $Vars(F)$ *and* $Z \subseteq (X \setminus Vars(\boldsymbol{q}))$. *We will call D-sequent* $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ ***single-variable*** *if* $|Z|=1$.

Definition 18. *D-sequents* $(\exists X[F], \boldsymbol{q'}, W') \rightarrow \{v'\}$ *and* $(\exists X[F], \boldsymbol{q''}, W'') \rightarrow \{v''\}$ *are called* ***compatible*** *if*

- $\boldsymbol{q'}$ *and* $\boldsymbol{q''}$ *are compatible*
- $(Vars(\boldsymbol{q'}) \cup Vars(\boldsymbol{q''})) \cap (W' \cup W'' \cup \{v'\} \cup \{v''\} = \emptyset$

Definition 19. *Let* Ω *be a set of single-variable D-sequents for an* $\exists$CNF *formula* $\exists X[F]$. *We will say that* Ω *is* ***a set of compatible D-sequents*** *if every pair of D-sequents of* Ω *is compatible.*

Definition 20. *Let* Ω *be a set of compatible D-sequents for an* $\exists$CNF *formula* $\exists X[F]$. *Denote by* $\boldsymbol{a^{\Omega}}$ *the assignment that is the union of all* $\boldsymbol{s}$ *occurring in D-sequents* $(\exists X[F], \boldsymbol{s}, W) \rightarrow W$ *of* Ω. *We will call* $\boldsymbol{a^{\Omega}}$ *the* ***axis*** *of* Ω. *Denote by* $\boldsymbol{W^{\Omega}}$ *the union of the scopes* W *of the D-sequents of* Ω.

Definition 21. *Let* Ω *be a set of compatible D-sequents for an* $\exists$CNF *formula* $\exists X[F]$. *Denote by* $\boldsymbol{X^{\Omega}}$ *the set of all variables of* X *whose redundancy is stated by D-sequents of* Ω. *In the following write-up we assume that* $|\boldsymbol{X^{\Omega}}| = |\boldsymbol{\Omega}|$. *That is for every variable* v *of* X^{Ω}, *set* Ω *contains exactly one D-sequent stating the redundancy of* v.

Definition 22. *Let Ω be a set of compatible D-sequents for an* $\exists$CNF *formula* $\exists X[F]$. *We will call D-sequent* $(\exists X[F], \boldsymbol{a^{\Omega}}, W^{\Omega}) \rightarrow X^{\Omega}$ ***the composite D-sequent*** *for Ω. We will call set Ω* ***composable*** *if the composite D-sequent of Ω holds for* $\exists X[F]$.

Lemma 3. *Let v be the branching variable picked by DDS after making assignment $\boldsymbol{q}$. Assume for the sake of clarity that $v = 0$ and $v = 1$ are assignments of left and right branches respectively. Denote by Ω_0 and Ω_1 the sets of D-sequents derived in branches $v = 0$ and $v = 1$ respectively. Denote by Ω the set of D-sequents produced by procedure join_D_seqs of Figure 3. Let Ψ,Ψ_0,Ψ_1 be subsets of $\Omega, \Omega_0, \Omega_1$ and X^{Ψ}=X^{Ψ_0}=X^{Ψ_1}. Let the composite D-sequents of Ψ_0 and Ψ_1 hold. Then the composite D-sequent of Ψ holds too.*

Proof: Assume the contrary i.e. $(\exists X[F], \boldsymbol{a^{\Psi}}, W^{\Psi}) \rightarrow X^{\Psi}$ does not hold. Then there is a Z-boundary point $\boldsymbol{p}$ of $F_{\boldsymbol{a^{\Psi}}}$ where $Z \subseteq X^{\Psi}$ that is W^{Ψ}-removable. Let v be a variable of X^{Ψ}. Denote by $\boldsymbol{q_0}$ and $\boldsymbol{q_1}$ the points $\boldsymbol{q} \cup \{(v = 0)\}$ and $\boldsymbol{q} \cup \{(v = 1)\}$ respectively. Let $(\exists X[F], \boldsymbol{s_0}, W_0) \rightarrow \{v\}$, $(\exists X[F], \boldsymbol{s_1}, W_1) \rightarrow \{v\}$, $(\exists X[F], \boldsymbol{s}, W) \rightarrow \{v\}$ be the D-sequents derived in subspaces $\boldsymbol{q_0}$, $\boldsymbol{q_1}$ and $\boldsymbol{q}$ respectively. We can have two situations here. First, all three D-sequents are equal to each other because the D-sequent of subspace $\boldsymbol{q_0}$ is symmetric in v. In this case, W=W_0=W_1. Second, the D-sequent of subspace $\boldsymbol{q}$ is obtained by joining the D-sequents of subspaces $\boldsymbol{q_0}$ and $\boldsymbol{q_1}$ at variable v. In this case, $W = W_0 \cup W_1$. In either case $W_0 \subseteq W$ and $W_1 \subseteq W$ hold. Hence $W^{\Psi_0} \subseteq W^{\Psi}$ and $W^{\Psi_1} \subseteq W^{\Psi}$.

By construction, $\boldsymbol{q_0} \subseteq \boldsymbol{p}$ or $\boldsymbol{q_1} \subseteq \boldsymbol{p}$. Assume for the sake of clarity that $\boldsymbol{q_0} \subseteq \boldsymbol{p}$ holds. Then point $\boldsymbol{p}$ is a Z-boundary point of $F_{\boldsymbol{a^{\Psi_0}}}$ where $Z \subseteq X^{\Psi_0}$ that is W^{Ψ_0}-removable. Hence, the composite D-sequent $(\exists X[F], \boldsymbol{a^{\Psi_0}}, W^{\Psi_0}) \rightarrow X^{\Psi_0}$ does not hold. Contradiction $\square$

Lemma 4. *Let D-sequent* $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ *hold. Let V be a subset of Z. Then D-sequent* $(\exists X[F], \boldsymbol{q}, W) \rightarrow V$ *holds too.*

Proof: Assume that $(\exists X[F], \boldsymbol{q}, W) \rightarrow V$ does not hold. Then there is a V'-boundary point $\boldsymbol{p}$ where $V' \subseteq V$ that is W-removable in $F_{\boldsymbol{q}}$. Since $V' \subseteq Z$ this means that Z is not redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope W. Contradiction.

Lemma 5. *Let Ω be a compatible set of D-sequents for an* $\exists$CNF*formula* $\exists X[F]$. *Let $\boldsymbol{q}$ be an assignment to variables of $Vars(F)$ such that $\boldsymbol{a^{\Omega}} \subseteq \boldsymbol{q}$ where $\boldsymbol{a^{\Omega}}$is the axis of Ω. Let $v \in X \setminus (Vars(\boldsymbol{q}) \cup X^{\Omega})$ be a blocked variable of $F_{\boldsymbol{q}}$. Let $\boldsymbol{s}$ be an assignment defined as follows. For every pair of clauses A, B of F that can be resolved on variable v, $\boldsymbol{s}$ contains either*

1. *an assignment satisfying A or B or*
2. *all the assignments of $\boldsymbol{r}$ such that*
   - *a D-sequent* $(\exists X[F], \boldsymbol{r}, W') \rightarrow \{v'\}$ *is in Ω and*
   - *A or B contains variable v'*

Denote by Ψ the subset of Ω comprising of all D-sequents $(\exists X[F], \boldsymbol{r}) \rightarrow \{w\}$ *that were used in the second condition above. Let the composite D-sequent* $(\exists X[F], \boldsymbol{a^{\Psi}}, W^{\Psi}) \rightarrow X^{\Psi}$ *hold. Then a D-sequent* $(\exists X[F], \boldsymbol{s}, W^{\Psi} \cup \{v\}) \rightarrow \{v\}$ *holds.*

Proof: Notice that variable v is blocked in the formula $F_{\boldsymbol{s}} \backslash (F_{\boldsymbol{s}})^{X^{\Psi}}$. Then Proposition 5 entails that v is redundant in $F_{\boldsymbol{s}} \backslash (F_{\boldsymbol{s}})^{X^{\Psi}}$. Since, by construction, $\boldsymbol{a}^{\boldsymbol{\Psi}} \subseteq \boldsymbol{s}$, then Lemma 2 implies that D-sequent $(\exists X[F], \boldsymbol{s}, W^{\Psi}) \rightarrow X^{\Psi}$ holds. Then from Proposition 4 it follows that the D-sequent $(\exists X[F], \boldsymbol{s}, W^{\Psi} \cup \{v\}) \rightarrow X^{\Psi} \cup \{v\}$ holds. Then Lemma 4 entails that the D-sequent $(\exists X[F], \boldsymbol{s}, W^{\Psi} \cup \{v\}) \rightarrow \{v\}$ holds $\square$

Lemma 6. *Let $\exists X[F]$ be an $\exists$CNF. Let C be a clause of F falsified by an assignment $\boldsymbol{q}$. Let v be a variable of $X \setminus Vars(\boldsymbol{q})$. Then D-sequent $(\exists X[F], \boldsymbol{s}, \{v\}) \rightarrow \{v\}$ holds where $\boldsymbol{s}$ is the shortest assignment falsifying C.*

Proof: The proof is similar to that of Proposition 6.

Lemma 7. *Any subset of active D-sequents derived by DDS is composable.*

Proof: Let us first give an informal argument. As we mentioned in Subsection 5.3, D-sequents $(\exists X[F], \boldsymbol{q'}, W') \rightarrow \{v'\}$ and $(\exists X[F], \boldsymbol{q''}, W') \rightarrow \{v''\}$ may be uncomposable if recursive reasoning is involved. That is $\{v'\}$-clauses are used to prove redundancy of variable v'' and vice versa. *DDS* avoids recursive reasoning by keeping the $\{v\}$-clauses removed from $\exists X[F]$ as long as a D-sequent for variable v remains active. Thus, if, for instance, $\{v'\}$-clauses are used to prove redundancy of variable v'', the $\{v''\}$-clauses are removed from F and cannot be used to prove redundancy of variable v'. In other words, for every path of the search tree, variables v' and v'' are proved redundant in a particular order (but this order may be different for different paths).

Let Ψ be a set of active D-sequents. To show composability of D-sequents from Ψ one needs to consider the following three cases.

1. All D-sequents of Ψ are atomic. Assume for the sake of simplicity that $\Psi = \{S', S''\}$ where S' and S'' are equal to $(\exists X[F], \boldsymbol{q'}, W') \rightarrow \{v'\}$ and $(\exists X[F], \boldsymbol{q''}, W') \rightarrow \{v''\}$ respectively. One can have two different cases here.
   - S' and S'' are independent of each other. That is there is no clause C of F that has variables v' and v'' and is not blocked at v' or v''. In this case, one can easily show that the D-sequent $(\exists X[F], \boldsymbol{q'} \cup \boldsymbol{q''}, W' \cup W'') \rightarrow \{v', v''\}$ holds.
   - S' and S'' are interdependent. This can happen only if v' and v'' are blocked. Atomic D-sequents derived due to the presence of a clause falsified by $\boldsymbol{q}$ (see Lemma 6) are independent of each other or D-sequents of blocked variables. Suppose the fact that v' is blocked is used to prove that v'' is blocked as well. Then Lemma 5 entails that $\boldsymbol{q'} \subseteq \boldsymbol{q''}$ and $W' \subseteq W''$ and that D-sequent $(\exists X[F], \boldsymbol{q''}, W'') \rightarrow Z$ holds where $\{v', v''\} \subseteq Z$. Then the composability of S' and S'' simply follows from Lemma 4.
2. The set Ψ is obtained from set Ψ_0 and Ψ_1 when merging branches $v = 0$ and $v = 1$. Then Lemma 3 entails that if Ψ_0 and Ψ_1 are composable, then Ψ is composable as well.
3. Ψ is a mix of atomic and non-atomic D-sequents. Assume for the sake of simplicity that $\Psi = \{S', S''\}$ where S' and S'' are equal to $(\exists X[F], \boldsymbol{q'}, W') \rightarrow \{v'\}$ and $(\exists X[F], \boldsymbol{q''}, W') \rightarrow \{v''\}$ respectively. Assume that S' is a result of join operations while S'' is atomic. Let $S'_1, \ldots, S'_k$ be the set of atomic

D-sequents that are ancestors of S'. Here $S'_i = (\exists X[F], \boldsymbol{q'_i}, W'_i) \rightarrow \{v'\}$. Let $S''_1, \ldots, S''_k$ be the set of D-sequents obtained from S'' where $S''_i = (\exists X[F], \boldsymbol{q'_i} \cup \boldsymbol{q''}, W'') \rightarrow \{v''\}$. Due to Lemma 2, each D-sequent S''_i holds. Since S'_i, S''_i are atomic this case is covered by item 1 above and so they are composable. Then the D-sequents obtained by composition of S'_i, S''_i can be joined producing correct D-sequents (due to correctness of operation join). Eventually, a correct D-sequent that is the composite of S' and S'' will be derived □

Proposition 11. *DDS is sound and complete.*

Proof: First, we show that *DDS* is *complete*. *DDS* builds a binary search tree and visits every node of this tree at most three times (when starting the left branch, when backtracking to start the right branch, when backtracking from the right branch). So *DDS* is complete.

Now we prove that *DDS* is *sound*. *DDS* terminates in two cases. First, it terminates when an empty clause is derived, which means that F is unsatisfiable. In this case, the formula G returned by *DDS* consists only of an empty clause. This result is correct because this clause is built by resolving clauses of F and resolution is sound. Second, *DDS* terminates after building a sequence of D-sequents $(\exists X[F], \emptyset, X_{i_1}) \rightarrow \{x_{i_1}\}$, ... ,$(\exists X[F], \emptyset, X_{i_k}) \rightarrow \{x_{i_k}\}$. Here $x_{i_1}, ..., x_{i_k}$ are the variables forming X and $\{x_{i_m}\} \subseteq X_{i_m} \subseteq X$, $m = 1, \ldots, k$. We need to show that these D-sequents are correct and composable. The latter means that the D-sequent $(\exists X[F], \emptyset, X) \rightarrow X$ holds, which means that the variables of X are redundant in the current formula $\exists X[F]$.

Let us carry out the proof by induction in the number of steps of *DDS*. The algorithm has two kinds of steps. A step of the first kind is to add a new atomic D-sequent to an existing set Ω of active D-sequents. A step of the second kind is to produce a new set of D-sequents Ω from the sets of D-sequents Ω_0 and Ω_1 obtained in branches $v = 0$ and $v = 1$.

Let $\boldsymbol{q^k}$ be the assignment made by Ω after steps $1, \ldots, k$. Let Ω^k be the set of D-sequents maintained by *DDS* that are active in subspace $\boldsymbol{q^k}$. (We assume here that every D-sequent is discarded after it takes part in a join operation. So for one redundant variable Ω contains only one active D-sequent.)

The induction hypothesis is as follows. The fact that D-sequents of Ω^k are individually correct and every subset of Ω^k is composable implies that the D-sequents of Ω^{k+1} are correct and every subset of Ω^{k+1} is composable.

The base step, k=1. We need to consider the following two situations.

- The first atomic D-sequent S is derived. In this case, its correctness follows Lemmas 5, 6. Since Ω^1 consists only of one D-sequent, every subset of Ω^1 is obviously composable.
- The first step consists of merging empty sets of D-sequents Ω^1_0 and Ω^1_1 derived in branches $v = 0$ and $v = 1$. In this case, Ω is empty. So the claims that every D-sequent of Ω is correct and all subsets are composable are vacuously true.

The induction step. We need to consider the following two situations.

- The set Ω^{k+1} is produced by adding an atomic D-sequent S to Ω^k. The correctness of S follows from Lemmas 5, 6. Notice that to apply Lemma 5 we need to use the induction hypothesis. The fact that every subset of D-sequents of $\Omega^k \cup \{S\}$ is composable can be proved using the reasoning of Lemma 7. (Notice that we cannot directly apply Lemma 7 because this lemma itself needs to be proved by induction. In the sketch of a proof of Lemma 7, we just gave reasoning one can use to perform such a proof.)
- The set Ω^{k+1} is produced by merging sets of D-sequents Ω_0^k and Ω_1^k derived in branches $v = 0$ and $v = 1$. The correctness of individual D-sequents of Ω^{k+1} follows from the induction hypothesis and the correctness of operation join (Proposition 9). Lemma 3 and the induction hypothesis entail that every subset of D-sequents of Ω^{k+1} is composable.

Proposition of Section 7: Compositionality of *DDS*

Definition 23. *We will refer to D-sequents derived due to appearance of an empty clause in formula $F_{\boldsymbol{q}}$ (see Subsection 6.2) as **clause D-sequents**.*

Proposition 12 (compositionality of DDS). *Let T be the search tree built by DDS when solving the QE problem $\exists X[F_1 \wedge \ldots \wedge F_k]$, $Vars(F_i) \cap Vars(F_j) = \emptyset$, $i \neq j$. Let $X_i = X \cap Vars(F_i)$ and $Y_i = Vars(F_i) \setminus X$. The size of T in the number of nodes is bounded by $|Vars(F)| \cdot (\eta(X_1 \cup Y_1) + \ldots + \eta(X_k \cup Y_k))$ 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.*

Proof: Denote by Y the set of variables $Vars(F) \setminus X$.

We prove this proposition for a slightly modified version of *DDS*. In the version of *DDS* shown in Figure 1, the D-sequents depending on the branching variable are discarded. The modification is to keep all derived D-sequents. This means that there is a set Π where all derived D-sequents are stored. We assume that *DDS* does not derive the same D-sequent twice. That is if Π contains a D-sequent S equal to $(\exists X[F], \boldsymbol{q}, \{x\}) \rightarrow \{x\}$, then the modified *DDS* declares $\{x\}$ redundant as soon as S becomes active instead of deriving it again.

Let P be a path of T and $n(v)$ be a node of T that is on P. Here v is the branching variable selected in the node n by *DDS*. We will call $n(v)$ a **BCP node**, if the variable v was selected due to its presence in a unit clause of $F_{\boldsymbol{q}}$. We will call P an **essential path**, if for every BCP node $n(v)$ lying on P (if any) the latter corresponds to the *right branch* of n. That is the variable v is currently assigned the value *satisfying* the unit clause C of $F_{\boldsymbol{q}}$ due to which v was picked. Recall that the first value assigned to v by *DDS* falsifies C.

Let d denote the total number of nodes lying on essential paths. Notice that the number of all nodes of T is bounded by $2 \cdot d$. The reason is that a non-essential path contains a BCP node $n(v)$ where v is assigned the value falsifying the unit clause due to which v was selected. So the last node of this path is the left child of node $n(v)$. Thus the number of nodes lying only on non-essential

paths is bounded by the number of BCP nodes of T. Since every BCP node lies on an essential path, the total number of nodes of T is bounded by $2 \cdot d$.

Denote by N_{ess_paths} the total number of essential paths of T. Denote by N_{res_cl} the total number of resolvent clauses generated by *DDS*. Denote by N_{D_seqs} the total number of D-sequents generated by *DDS* with the exception of clause D-sequents.

We do the rest of the proof in two steps. First we show that $N_{ess\_paths} \leq N_{res\_cl} + N_{D\_seqs}$. Since a path of T cannot contain more than $|X \cup Y|$ nodes, this means that the total number of nodes of T is bounded by $2 \cdot |X \cup Y| \cdot (N_{res\_cl} + N_{D\_seqs})$. In the second step, we show that $2 \cdot (N_{res\_cl} + N_{D\_seqs}) \leq \eta(X_1 \cup Y_1) + \ldots + \eta(X_k \cup Y_k)$ where $\eta(X_i \cup Y_i) = 2 \cdot 3^{|X_i \cup Y_i|} \cdot (|X_i| + 1), i = 1, \ldots, k$.

FIRST STEP: To prove that $N_{ess\_paths} \leq N_{res\_cl} + N_{D\_seqs}$ we show that every essential path of T corresponds to a new resolvent clause or a new D-sequent generated by *DDS* that is not a clause D-sequent. Let P be an essential path of T. Let $v \in X \cup Y$ be the first variable of P picked by *DDS* for branching. The very fact that v was selected means that some of the variables of X were not proved redundant in $\exists X[F]$ yet. Let us assume the contrary, that is *DDS* is able to finish P without generating a new clause or a new D-sequent that is not a clause D-sequent. This only possible if *DDS* can assign all free non-redundant variables of X without running into a conflict (in which case a new clause is generated) or producing a new blocked variable (in which case a new non-clause D-sequent is generated).

Let $x \in X$ be the last variable assigned by *DDS* on path P. That is every other variable of X is either assigned or proved redundant before making an assignment to x. Let $\boldsymbol{q}$ be the set of assignments on path P made by *DDS* before reaching the node $n(x)$, and X' be the set of all redundant variables of X in $F_{\boldsymbol{q}}$. Since variables of Y are assigned before those of X, all non-detached variables of Y are assigned. Then the current formula, *i.e.*, formula $F_{\boldsymbol{q}} \setminus F_{\boldsymbol{q}}^{X'}$ has only two kinds of clauses:

- clauses depending only on detached variables of Y or
- unit clauses that depend only on variable x.

The two possibilities for the unit clauses depending on x are as follows.

- $F_{\boldsymbol{q}} \setminus F_{\boldsymbol{q}}^{X'}$ contains both clauses x and $\overline{x}$. Then, *DDS* generates a new clause. Contradiction.
- $F_{\boldsymbol{q}} \setminus F_{\boldsymbol{q}}^{X'}$ does not contain either x or $\overline{x}$ or both. Then x is blocked and *DDS* generates a new non-clause D-sequent. Contradiction.

SECOND STEP: Notice that no clause produced by resolution can share variables of two different subformulas F_i and F_j. This means that for every clause C produced by *DDS*, $Vars(C) \subseteq (X_i \cup Y_i)$ for some i. The total number of clauses depending on variables of $X_i \cup Y_i$ is $3^{|X_i \cup Y_i|}$. So $N_{res\_cl} \leq 3^{|X_1 \cup Y_1|} + \ldots + 3^{|X_k \cup Y_k|}$.

Now we show that $N_{D\_seqs} \leq |X_1| \cdot 3^{|X_1 \cup Y_1|} + \ldots + |X_k| \cdot 3^{|X_k \cup Y_k|}$ and hence $2 \cdot (N_{res\_cl} + N_{D\_seqs}) \leq \eta(X_1 \cup Y_1) + \ldots + \eta(X_k \cup Y_k)$. The idea is to prove that every non-clause D-sequent generated by *DDS* is **limited to $\boldsymbol{F_i}$**, *i.e.*, has the form $(\exists X[F], \boldsymbol{s}, W) \rightarrow \{x\}$ where $Vars(\boldsymbol{s}) \subseteq X_i \cup Y_i$, $W \subseteq X_i$ and $x \in X_i$. Recall

that due to Proposition 7, D-sequent $(\exists X[F], \boldsymbol{s}, W) \rightarrow \{x\}$ is invariant to adding resolvent clauses to F. For that reason, we will ignore the parameter $\exists X[F]$ when counting the number of D-sequents limited to F_i. Besides, due to Proposition 8, one can always increase the scope of a D-sequent. For that reason, when counting D-sequents, we will also ignore the parameter W. Then the total number of D-sequents limited to F_i is equal to $|X_i| \cdot 3^{|X_i \cup Y_i|}$. So the total number of D-sequents limited to F_i, $i = 1, \ldots, k$ is bounded by $|X_1| \cdot 3^{|X_1 \cup Y_1|} + \ldots + |X_k| \cdot 3^{|X_k \cup Y_k|}$. The factor $|X_i|$ is the number of variables appearing on the right side of a D-sequent limited to F_i. The factor $3^{|X_i \cup Y_i|}$ specifies the total number of all possible assignments $\boldsymbol{s}$.

Now we prove that every non-clause D-sequent derived by *DDS* is limited to a formula F_i. We carry out this proof by induction. Our base statement is that D-sequents of an empty set are limited to F_i. It is vacuously true. Assume that the non-clause D-sequents generated so far are limited to F_i and then show that this holds for the next non-clause D-sequent S. Let S be a D-sequent $(\exists X[F], \boldsymbol{s}, W) \rightarrow \{x\}$ generated for a blocked variable $x \in X_i$. Such a D-sequent is built as described in Lemma 5. Then $\boldsymbol{s}$ consists of assignments satisfying $\{x\}$-clauses of F or being the reason for their redundancy. Since clauses of different subformulas cannot be resolved with each other, every $\{x\}$-clause of F can only have variables of F_i where $x \in \mathit{Vars}(F_i)$. By the induction hypothesis every non-clause D-sequent is limited to some subformula. On the other hand, *DDS* looks for blocked variables when $F_{\boldsymbol{q}}$ has no empty clause. So, at the time S is derived, no variable of $F_{\boldsymbol{q}}$ can be redundant due to a clause D-sequent. This means that if a variable x^* of an $\{x\}$-clause of F is redundant due to D-sequent $(\exists X[F], \boldsymbol{s^*}, W^*) \rightarrow \{x^*\}$ then $\mathit{Vars}(\boldsymbol{s^*}) \subseteq \mathit{Vars}(F_i)$. So $\mathit{Vars}(\boldsymbol{s}) \subseteq \mathit{Vars}(F_i)$.

Now consider the case when S is obtained by joining two D-sequents S', S''. Let us consider the following three possibilities

- Neither S' nor S'' is a clause D-sequent. Then according to the induction hypothesis they should be limited to F_i. (They cannot be limited to different subformulas because then they cannot be joined due to absence of a common variable.) Then due to Definition 14, the D-sequent produced by joining S' and S'' is also limited to F_i.
- Either S' or S'' is a clause D-sequent. Let us assume for the sake of clarity that this is the D-sequent S'. This means that S' has the form $(\exists X[F], \boldsymbol{s}, \{x\}) \rightarrow \{x\}$ where $\boldsymbol{s}$ is the minimum set of assignments falsifying a clause C of F and $x \in X \setminus \mathit{Vars}(\boldsymbol{s})$. Since for any resolvent C of F, $\mathit{Vars}(C) \subseteq \mathit{Vars}(F_i)$, then $\mathit{Vars}(\boldsymbol{s}) \subseteq \mathit{Vars}(F_i)$. By the induction hypothesis, S'' is limited to F_j. Since S' and S'' have at least one common variable (at which they are joined), j has to be equal to i. So $x \in X_i$. Then joining S' with S'' produces a D-sequent that is also limited to F_i.
- Both S' and S'' are clause D-sequents. We do not care about this situation because by joining S' and S'' one obtains a clause D-sequent $\square$