Form Methods Syst Des (2014) 45:111–143
DOI 10.1007/s10703-014-0214-z

Quantifier elimination by dependency sequents

Eugene Goldberg · Panagiotis Manolios

Published online: 14 August 2014

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

Keywords Quantifier elimination · Resolution · Model checking · SAT · Dependency sequents

1 Introduction

In this paper, we consider the problem of eliminating quantifiers from formulas of the form $\exists X[F]$ where F is a Boolean CNF formula and some variables of F may be free. We will refer to such formulas as $\exists$CNF. The **Quantifier Elimination (QE) problem**, is to find a quantifier-free CNF formula G such that $G \equiv \exists X[F]$. The equivalence "$\equiv$" is semantic. That is for every complete assignment $\boldsymbol{y}$ to the free variables of $\exists X[F]$, the logical value of $G_{\boldsymbol{y}}$ is equal to that of $\exists X[F_{\boldsymbol{y}}]$. Here $F_{\boldsymbol{y}}$ and $G_{\boldsymbol{y}}$ are formulas F and G under assignment $\boldsymbol{y}$.

Our interest in the QE problem is twofold. First, the QE problem occurs in numerous areas of hardware design and verification, *e.g.*, in symbolic model checking [9,10,24] when computing reachable states. So it is important to develop new QE algorithms. Second, studying QE can help better understand and advance the existing algorithms that do not use QE

E. Goldberg (✉) · P. Manolios
Northeastern University, Boston, MA, USA
e-mail: eu.goldberg@gmail.com; eigold@ccs.neu.edu

P. Manolios
e-mail: pete@ccs.neu.edu

explicitly. For example, many successful theorem proving methods such as interpolation [26] and IC3 [5] avoid QE and use SAT-based reasoning instead. However, one can argue that these methods can be viewed as solving specialized versions of the QE problem that can be dealt with efficiently. For example, finding an interpolant $I(Y)$ of formula $A(X, Y) \wedge B(Y, Z)$ comes down to solving a special case of QE where $I \equiv \exists X[A]$ needs to hold only in subspaces where $B \equiv 1$.

The success of resolution-based SAT-solvers [23,27] has led to the hunt for efficient SAT-based algorithms for the QE problem [6,14,20,25]. In this paper, we continue in this direction by introducing a resolution-based QE algorithm operating on CNF formulas. Such formulas are ubiquitous in hardware verification because a circuit N can be represented by a CNF formula whose size is linear in that of N and that has the same set of variables as N.

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

Let $\boldsymbol{y}$ be an assignment to all free variables of $\exists X[F]$. If $F_{\boldsymbol{y}}$ is unsatisfiable, then a clause C falsified by $\boldsymbol{y}$ can be derived by resolving clauses of F. After adding C to F, to obtain F', the variables of X are redundant in $\exists X[F'_{\boldsymbol{y}}]$. Detecting redundancy in this case is easy. Assume, however, that $F_{\boldsymbol{y}}$ is *satisfiable*; in the context of the QE problem, this is the *common* case. Then, the variables of X are *also redundant* in $\exists X[F_{\boldsymbol{y}}]$ because $F_{\boldsymbol{y}}$ remains satisfiable after removing any set of clauses. We want to identify such redundancies efficiently, without having to perform all possible resolutions.

To address this problem, we introduce the notion of dependency sequents (*D-sequents*)[1]. A D-sequent has the form $(\exists X[F], \boldsymbol{q}) \rightarrow Z$ where $\boldsymbol{q}$ is a partial assignment to variables of F and $Z \subseteq X$. This D-sequent states that in the subspace specified by $\boldsymbol{q}$, the variables of Z are redundant in $\exists X[F]$. That is, in this subspace, after dropping clauses with variables of Z from F one gets a formula equivalent to $\exists X[F]$. In particular, the D-sequent $(\exists X[F], \boldsymbol{y}) \rightarrow X$ holds, if formula $F_{\boldsymbol{y}}$ is satisfiable where $\boldsymbol{y}$ is an assignment to the non-quantified variables of $\exists X[F]$. For the sake of brevity, in the following exposition we use the same symbol F to denote the initial and the *current* CNF formula consisting of the initial clauses and *resolvents*. So symbol F used in the D-sequent above is the current CNF formula. We note that the formal definition of D-sequents, given in Sect. 5, is somewhat more involved than we have let on in the introduction, *e.g.*, it includes a scope parameter.

In this paper, we introduce a QE algorithm called derivation of D-sequents (*DDS*). In *DDS*, adding resolvent clauses to F is accompanied by computing D-sequents. The algorithm terminates when the D-sequent $(\exists X[F], \emptyset) \rightarrow X$ is derived, where F is the final CNF formula that includes the initial and resolvent clauses. Upon termination, the variables of X are unconditionally redundant and a solution to the QE problem is obtained by dropping the clauses containing variables of X from F.

[1] In this paper, we consider D-sequents presented at FMCAD-12 [17] that state redundancy of quantified *variables*. At FMCAD-13 [19], we introduced D-sequents stating the redundancy of *clauses* containing quantified variables. Formally, D-sequents based on variable redundancy are just a special case of those based on clause redundancy. However, there is an important difference that justifies a special attention to D-sequents based on variable redundancy. The set X of quantified variables in formula $\exists X[F]$ remains *the same* while the set of clauses containing variables of X may *grow exponentially* in $|X|$. If a QE-algorithm using D-sequents based on clause redundancy "gets lost", it may generate a large number of irrelevant clauses whose redundancy it will have to prove. QE-algorithms based on variable redundancy do not have this problem.

DDS produces new D-sequents from existing D-sequents using the *join* operation. Let $(\exists X[F], \boldsymbol{q}') \rightarrow Z$ and $(\exists X[F], \boldsymbol{q}'') \rightarrow Z$ be valid D-sequents where $\boldsymbol{q}'$ and $\boldsymbol{q}''$ have opposite assignments to exactly one variable v. Then a new, valid D-sequent $(\exists X[F], \boldsymbol{q}) \rightarrow Z$ can be obtained by joining the D-sequents above, where $\boldsymbol{q}$ contains all assignments of $\boldsymbol{q}'$ and $\boldsymbol{q}''$ except 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 [6,14,21,25] are not. Compositionality here means that given an $\exists$CNF formula $\exists X[F_1 \wedge \cdots \wedge F_k]$ where formulas F_i depend on non-overlapping sets of variables, *DDS* breaks the QE problem into k independent subproblems. *DDS* is a branching algorithm and yet it remains compositional no matter how branching variables are chosen. Compositionality of *DDS* means that its performance can be *exponentially better* than that of enumeration-based QE algorithms. Since *DDS* is a branching algorithm it can process variables of different branches in different orders. This gives *DDS* a big edge over QE algorithms that eliminate quantified variables one by one using a global order [16,20].

D-sequents are related to boundary points [15]. 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 are used to define the semantics of *DDS.*

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 scoped redundancy of variables that is a generalization of that of blocked and monotone variables. Third, we introduce the notion of D-sequents and the operation of joining D-sequents. Fourth, we describe *DDS*, a QE algorithm; we prove its correctness and evaluate it experimentally. Fifth, we show that *DDS* is compositional.

This paper is structured as follows. In Sect. 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 Sect. 4 and D-sequents are introduced in Sect. 5. A run of *DDS* on a simple formula is explained in Sect. 6. Sections 7 and 8 give a formal description of *DDS* and discuss its compositionality. Section 9 gives experimental results. Related work is discussed in Sect. 10, and conclusions are presented in Sect. 11.

2 Redundant variables, boundary points and quantifier elimination

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

2.1 Redundant variables and quantifier elimination

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

Definition 2 Let C be a clause, H be a formula, and $\boldsymbol{p}$ be an assignment. $C_{\boldsymbol{p}}$ is *true* if C is satisfied by $\boldsymbol{p}$; otherwise it is the clause obtained from C by removing all literals falsified by $\boldsymbol{p}$. $H_{\boldsymbol{p}}$ denotes the formula obtained from H by removing every clause $C \in H$ for which $C_{\boldsymbol{p}}$ = *true* and replacing C with $C_{\boldsymbol{p}}$ if $C_{\boldsymbol{p}} \neq$ *true*. If $\mathit{Vars}(H) \subseteq \mathit{Vars}(\boldsymbol{p})$, then $H_{\boldsymbol{p}}$ is semantically equivalent to a constant, and in the sequel, we will make use of this without explicit mention.

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

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

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

Definition 6 The variables of Z are **redundant** in CNF formula F if $F \equiv (F \setminus F^Z)$. The variables of Z are **redundant** in ∃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.

Note that $F \equiv (F \setminus F^Z)$ implies $\exists X[F] \equiv \exists X[F \setminus F^Z]$ but the opposite is not true.

2.2 Redundant variables and boundary points

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

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

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

Example 1 Let F be a CNF formula of four clauses: $C_1 = v_1 \vee v_2, C_2 = \overline{v}_1 \vee v_3, C_3 = v_1 \vee \overline{v}_4, C_4 = \overline{v}_3 \vee \overline{v}_5$. The set of clauses of F falsified by point $\boldsymbol{p} = (v_1 = 0, v_2 = 0, v_3 = 0, v_4 = 1, v_5 = 1)$ consists of C_1 and C_3. One can verify that $\boldsymbol{p}$ and the set $Z = \{v_1\}$ satisfy the four conditions of Definition 8, which makes $\boldsymbol{p}$ a $\{v_1\}$-boundary point. The set Z above is not unique. One can easily check that $\boldsymbol{p}$ is also a $\{v_2, v_4\}$-boundary point.

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

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

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

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

Example 2 Let us consider the $\{v_1\}$-boundary point $\boldsymbol{p} = (v_1 = 0, v_2 = 0, v_3 = 0, v_4 = 1, v_5 = 1)$ of Example 1. Let C denote the resolvent $v_2 \vee v_3$ of C_1 and C_2 on v_1. Note that the set $X = \{v_1\}$ and C satisfy the conditions a),b) and c) of Definition 9. So $\boldsymbol{p}$ is a $\{v_1\}$-removable $\{v_1\}$-boundary point. After adding C to F, $\boldsymbol{p}$ is not a $\{v_1\}$-boundary point any more. Notice, however, that if $X = \{v_1, v_2, v_4\}$, then $\boldsymbol{p}$ is not X-removable. Indeed, to satisfy conditions a),b),c) for X and $\boldsymbol{p}$ there should exist clause C implied by F that is falsified by $\boldsymbol{p}$ and does not contain the variables of X. The only clauses satisfying conditions b) and c) are clause $C = v_3 \vee \overline{v}_5$ and any clause subsuming C. But C is not implied by F and hence breaks condition a). The same applies to any clause subsuming C.

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.

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

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

Below we introduce the notion of scoped redundancy of variables that is used in the definition of dependency sequents (Sect. 5).

Definition 10 Let Z be a set of variables redundant in $\exists X[F]$ where $Z \subseteq X$. We will say that 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.

Example 3 Let $\exists X[F]$ be an $\exists$CNF formula where $F = C_1 \wedge C_2$, $C_1 = y \vee x_1$, $C_2 = y \vee \overline{x}_1 \vee x_2$ and $X = \{x_1, x_2\}$. Notice that F does not contain literal $\overline{x_2}$ and hence x_2 is monotone in F. From Proposition 5 below it follows that x_2 is redundant in $\exists X[F]$ with scope $\{x_2\}$. In other words, x_2 is locally redundant in $\exists X[F]$. After removing clause C_2 from F as containing x_2, variable x_1 becomes monotone and hence redundant in $\exists X[F \setminus \{C_2\}]$ with scope $\{x_1\}$. From Proposition 4 below it follows that x_1 is redundant in $\exists X[F]$ with scope $\{x_1, x_2\}$.

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 6 i.e. $\exists X[F] \equiv \exists X[F \setminus F^Z]$. The opposite is not true. The notion of scoped redundancy is used in this paper[2] instead of that of virtual redundancy introduced in [17]. As we show in Sect. 4, scoped redundancy can be viewed as a generalization of the notion of a blocked variable. From now on, when we say that variables of Z are redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope W we assume that $W \cap \mathit{Vars}(\boldsymbol{q}) = \emptyset$.

[2] In [17], 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 6 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 6. 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 6. 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 $W \cap \mathit{Vars}(\boldsymbol{q}) = \emptyset$.

3 Boundary points and divide-and-conquer strategy

In this section, we provide the semantics of the QE algorithm *DDS* described in Sect. 7. *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 local 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 $\boldsymbol{q} \cup \boldsymbol{r}$.

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 $\exists X[F_{\boldsymbol{q}}]$.*

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

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, say, a set of variables $\{v_1, v_2\}$ in $\exists X[F_{\boldsymbol{q}}]$ *incrementally*. This can be done by (a) proving redundancy of variable v_1 in $\exists X[F_{\boldsymbol{q}}]$, (b) removing all the $\{v_1\}$-clauses from $F_{\boldsymbol{q}}$, and (c) proving redundancy of v_2 in formula $\exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^{\{v_1\}}]$.

4 Two simple cases of local variable redundancy

In this section, we describe two easily identifiable cases where variables are locally redundant. These cases are specified by Propositions 5 and 6. We also show that scoped redundancy can be viewed as a generalization of the notion of a blocked variable.

Definition 11 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 12 A variable v of a CNF formula F is called **blocked** if no two clauses of F are resolvable on v. A variable v is called **monotone** if it is a pure literal variable [12] (i.e. literals of only one polarity of v are present in F). A monotone variable is a special case of a blocked variable.

Example 4 Let CNF formula $F(v_1, v_2, v_3)$ consist of clauses C_1, C_2, C_3 where $C_1 = v_1 \vee v_2$, $C_2 = \overline{v}_1 \vee v_3$, $C_3 = \overline{v}_1 \vee \overline{v}_2$. Since clauses C_1 and C_2 are resolvable on v_1 the latter is not blocked. On the contrary, variables v_2 and v_3 are blocked. Variable v_2 is blocked because C_1 and C_3, the only clauses with opposite literals of v_2, are not resolvable on v_2. In addition to being blocked, variable v_3 is also monotone since no clause of F contains literal $\overline{v}_3$.

The notion of blocked variables is related to that of blocked clauses introduced in [22] (not to confuse with *blocking* clauses [25]). A clause C of F is blocked at v if no clause C' of F is resolvable with C on v. Variable v is blocked in F if every $\{v\}$-clause of F is blocked at v.

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 $\exists X[F_{\boldsymbol{q}}]$.*

Note that a blocked variable v can be viewed as a special case of a variable redundant in $\exists X[F_{\boldsymbol{q}}]$ with scope W where $\{v\} \subseteq W$. Indeed, let v be redundant with scope W. This means that for every $\{v\}$-boundary $\boldsymbol{p}$, one can find an assignment satisfying $F_{\boldsymbol{q}}$ by flipping only variables of W. Notice that redundancy of v in $\exists X[F_{\boldsymbol{q}}]$ with scope W is possible even if $F_{\boldsymbol{q}}$ has clauses that can be resolved on v. The only exception is when $W = \{v\}$. In this case, a $\{v\}$-boundary point turns into an assignment satisfying $F_{\boldsymbol{q}}$ by flipping the value of v. This means that $F_{\boldsymbol{q}}$ cannot have clauses that can be resolved on variable v and so, v is blocked in $F_{\boldsymbol{q}}$.

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 describe the operation of joining D-sequents. We also introduce the notion of composable D-sequents.

5.1 Definition of D-sequents

Definition 13 Let $\exists X[F]$ be an $\exists$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.

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

According to Definition 13, a D-sequent holds with respect to a particular $\exists$CNF formula $\exists X[F]$. Proposition 7 shows that this D-sequent also holds after adding to F resolvent clauses.

Proposition 7 *Let $\exists X[F]$ be an $\exists$CNF formula. Let $H = F \wedge G$ where $F \Rightarrow G$. Let $\boldsymbol{q}$ be an assignment to $Vars(F)$. 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.

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

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

Definition 15 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 circular reasoning where $\{v'\}$-clauses are used to prove redundancy of v'' and vice versa. Proposition 10 below shows how to avoid circular reasoning.

Definition 16 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 $\mathit{Vars}(\boldsymbol{q}') \cap \mathit{Vars}(\boldsymbol{q}'')$ 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 $\mathit{Vars}(\boldsymbol{q}) \cap Z = \mathit{Vars}(\boldsymbol{q}) \cap W = \emptyset$. Then D-sequent $(\exists X[F], \boldsymbol{q}, W \cup \{v\}) \rightarrow Z \cup \{v\}$ holds.*

Definition 17 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 $\mathit{Vars}(F)$ and $v \notin \mathit{Vars}(\boldsymbol{q}')$,$\mathit{Vars}(\boldsymbol{q}'') \cap Z = \emptyset$, $\mathit{Vars}(\boldsymbol{q}'') \cap W = \emptyset$. 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 A run of *DDS* on a simple formula

In this section, we describe a run of *DDS* on a simple formula. A detailed explanation of *DDS* is given in Sect. 7.

6.1 Problem formulation

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 $X = \{x\}$. The problem is to find formula $G(y_1, y_2)$ such that $G \equiv \exists X[F]$.

6.2 Short notion for D-sequents

In this section, we will use a short notation of D-sequents. In this notation, one writes $\boldsymbol{s} \rightarrow Z$ instead $(\exists X[F], \boldsymbol{s}, W) \rightarrow Z$ omitting parameters $\exists X[F]$ and W. (See explanation in Sect. 7.)

6.3 Active D-sequents

DDS derives D-sequents $\boldsymbol{s} \rightarrow \{x\}$ stating the redundancy of variable x. We will call D-sequent $\boldsymbol{s} \rightarrow \{x\}$ **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 $\boldsymbol{q}$.

6.4 The big picture

As far as the example at hand is concerned, the goal of *DDS* is to derive D-sequent $\emptyset \rightarrow \{x\}$ stating unconditional redundancy of variable x. As we mentioned before, *DDS* is a branching algorithm. It proves redundancy of x in subspaces and then merges the results of different branches. The work of *DDS* is shown in Figs. 1 and 2. To make variable x redundant, *DDS* has to generate a new clause (see below). When x is proved redundant, the $\{x\}$-clauses of F are marked as redundant. When *DDS* enters the subspace where the current D-sequent stating redundancy of x is not active anymore, the $\{x\}$-clauses of F (that are currently unsatisfied) are unmarked. This indicates that x is not proved redundant in the new subspace yet.

Fig. 1 Search tree

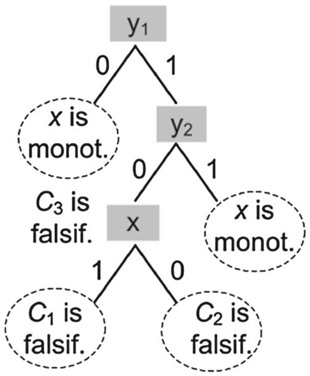
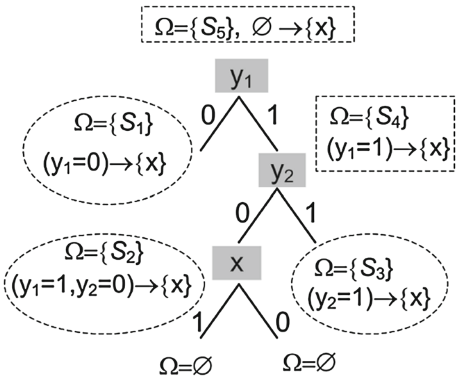


Fig. 2 Derivation of D-sequents



6.5 Search tree

Figure 1 shows the search tree built by *DDS*. To identify a node of the search tree we will use the assignment $\boldsymbol{q}$ leading to this node. The root node is specified by assignment $\boldsymbol{q} = \emptyset$. *DDS* branches on variables of $Vars(F) \setminus X = \{y_1, y_2\}$ before those of X (see Sect. 7.3). The search tree has four leaf nodes shown in dotted ovals. *DDS* backtracks as soon as variable x is assigned or proved redundant. For example, x is proved redundant at node $(y_1 = 0)$ and assigned at node $(y_1 = 1, y_2 = 0, x = 1)$.

6.6 Generation of a new clause

DDS generates a new clause at node $(y_1 = 1, y_2 = 0)$ after branching on x. At node $(y_1 = 1, y_2 = 0, x = 1)$, clause C_1 gets falsified and *DDS* immediately backtracks. Similarly, *DDS* falsifies clause C_2 at node $(y_1 = 1, y_2 = 0, x = 0)$. As described in Sect. 7.5, 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.

6.7 Generation of atomic D-sequents

Figure 2 describes derivation of D-sequents. Ω specifies the set of D-sequents that are active at the corresponding node. *DDS* has at most one active D-sequent for every quantified variable that is currently unassigned. Since in our example, only one variable is quantified, Ω is either empty or contains one active D-sequent. The dotted boxes show D-sequents obtained by the join operation. The D-sequents in the dotted ovals record trivial cases of redundancy discussed in Sect. 4. Such D-sequents are called atomic. For instance, *DDS* generates D-sequent S_1 equal to $(y_1 = 0) \rightarrow \{x\}$ at node $(y_1 = 0)$. 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* at node $(y_1 = 1, y_2 = 0)$. As we mentioned above, at node $(y_1 = 1, y_2 = 0)$, *DDS* 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.

6.8 Switching from left to right branch

Let us consider switching between branches at the root node where y_1 is picked for branching. The set of D-sequents 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* computes the set of D-sequents that become inactive in the right branch. Since S_1 contains assignment $y_1 = 0$ as the condition of redundancy of x, this D-sequent becomes inactive in the right branch. Hence *DDS* removes S_1 from Ω. So, before *DDS* starts exploring branch $y_1 = 1$, variable x becomes non-redundant and clause $C_2 = y_2 \vee x$ is unmarked because it does not contain a redundant variable.

6.9 Branch merging

Consider how branch merging is performed by *DDS* at node $(y_1 = 1)$. In the left branch $y_2 = 0$, the set Ω=$\{S_2\}$ is computed where S_2 is $(y_1 = 1, y_2 = 0) \rightarrow \{x\}$. In the right branch

$y_2 = 1$, the set Ω={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 $F_{(y_1=1)}$.

6.10 Termination

When *DDS* 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. After dropping the X-clauses C_1, C_2 in $\exists X[F]$, one obtains formula $\exists X[G]$ where $G(y_1, y_2) = C_3$. Since G does not depend on x, the quantifier can be dropped. So $G \equiv \exists X[C_1 \wedge C_2]$, which makes G a solution to the QE problem.

7 Description of *DDS*

In this section, we describe a QE algorithm called *DDS* (derivation of D-Sequents). We will continue using the short notation of D-sequents we introduced in Sect. 6. Namely we will write $\boldsymbol{s} \rightarrow \{x\}$ 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* ∃CNF *formula* (with all resolvent clauses added to F so far) and 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.

A description of *DDS* is given in Fig. 3. *DDS* accepts an ∃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), a set Ω of active D-sequents stating redundancy of *every* variable of $X \setminus$ *Vars*$(\boldsymbol{q})$ in $\exists X[F_{\boldsymbol{q}}]$ and a clause C. If $F_{\boldsymbol{q}}$ is unsatisfiable, then C is a

```
// $\Phi$ denotes $\exists X[F]$, $\boldsymbol{q}$ is an assignment to $Vars(F)$
// $\Omega$ denotes a set of active D-sequents
// If $DDS$ returns clause $nil$ (respectively a non-$nil$ clause),
// $F_{\boldsymbol{q}}$ is satisfiable (respectively unsatisfiable)

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

Fig. 3 *DDS* procedure

clause of F falsified by $\boldsymbol{q}$. Otherwise, C is equal to *nil* meaning that no clause implied by F is falsified by $\boldsymbol{q}$.

To build a quantifier-free 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*.

7.1 The big picture

First, *DDS* looks for variables whose redundancy is trivial to prove (lines 1-2). If some variables of $X \setminus \mathit{Vars}(\boldsymbol{q})$ are not proved redundant yet, *DDS* picks a branching variable v (line 3). Then it extends $\boldsymbol{q}$ by assignment $(v = 0)$ and recursively calls itself (line 4) 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 8). The results of the left and right branches are then merged (lines 9-15).

DDS terminates when for every variable x of $X \setminus \mathit{Vars}(\boldsymbol{q})$ it derives a D-sequent $\boldsymbol{s} \rightarrow \{x\}$ where $\boldsymbol{s} \subseteq \boldsymbol{q}$. As we show in the appendix (Lemma 7) D-sequents derived by *DDS* are composable. So derivation of D-sequents for individual variables also means that a D-sequent $\boldsymbol{s^*} \rightarrow (X \setminus \mathit{Vars}(\boldsymbol{q}))$ holds where $\boldsymbol{s^*} \subseteq \boldsymbol{q}$. Thus, *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 circular reasoning is employed where $\{v'\}$-clauses are used to prove redundancy of variable v'' and vice versa.

7.2 Building atomic D-sequents

Procedure *atomic_D_seqs* is called by *DDS* to compute D-sequents for trivial cases of variable redundancy listed in Sect. 4. We refer to such D-sequents as **atomic**. Procedure *atomic_D_seqs* returns an updated set of active D-sequents Ω, answer *solved* or *unsolved* (depending on whether the satisfiability/unsatisfiability of F has been established) and a clause C. If there is a clause of F falsified by $\boldsymbol{q}$, then C is this clause. Otherwise, C is *nil*.

Lines 1–3 of Fig. 4 show what is done when F contains a clause C falsified by $\boldsymbol{q}$. In this case, every unassigned variable of F becomes redundant (Proposition 6). So, for every variable of $x \in X \setminus \mathit{Vars}(\boldsymbol{q})$ for which Ω does not contain a D-sequent yet, procedure *process_unsat_clause* generates D-sequent $\boldsymbol{s} \rightarrow \{x\}$ and adds it to Ω. Here $\boldsymbol{s}$ is the shortest assignment falsifying C. Once Ω contains a D-sequent for every variable of $X \setminus \mathit{Vars}(\boldsymbol{q})$, *atomic_D_seqs* terminates returning set Ω, answer *solved* and clause C.

```
atomic_D_seqs($\Phi$, $\boldsymbol{q}$, $\Omega$){
1  if ($\exists$ clause $C \in F$ falsif. by $\boldsymbol{q}$){
2      $\Omega := process\_unsat\_clause(\Phi, C, \Omega)$;
3      return($\Omega$, $solved$, $C$);}
4  $\Omega := new\_redund\_vars(\Phi, \boldsymbol{q}, \Omega)$;
5  if ($all\_unassgn\_vars\_redund(\Phi, \boldsymbol{q}, \Omega)$) return($\Omega$, $solved$, $nil$);
6  return($\Omega$, $unsolved$, $nil$);}
```

Fig. 4 *Atomic_D_seqs* procedure

Suppose no clause of F is falsified by $\boldsymbol{q}$. Then for every variable x of $X \setminus \mathit{Vars}(\boldsymbol{q})$ that does not have a D-sequent in Ω and that is blocked, a D-sequent is built as explained below. This D-sequent is then added to Ω (line 4). If every variable of $X \setminus \mathit{Vars}(\boldsymbol{q})$ has a D-sequent in Ω, then $F_{\boldsymbol{q}}$ is satisfiable. (If $F_{\boldsymbol{q}}$ is *unsatisfiable*, the variables of $X \setminus \mathit{Vars}(\boldsymbol{q})$ can be made redundant *only* by adding a clause falsified by $\boldsymbol{q}$). So, *atomic_D_seqs* returns set Ω, answer *solved* and clause *nil* (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^*.

A straightforward search for blocked variables of F for every call of *DDS* may be too expensive. To reduce the cost of search for blocked variables one can use an approach similar to that of watched literals [27]. Let x be a variable of X. To guarantee that x is not blocked in the current subspace one can maintain a pair of "watched" clauses C', C'' that are resolvable on x and are neither satisfied nor proved redundant yet. As soon as C' or C'' is satisfied or proved redundant one needs to replace the failed watched clause. If such replacement is not possible, then x is currently blocked. So variable x is processed only when watched clauses responsible for x change their state. In the implementation of *DDS* we used in experiments, no optimization techniques were applied when searching for blocked variables.

7.3 Selection of a branching variable

Let *DDS* be called with assignment $\boldsymbol{q}$ and set of D-sequents Ω and X_{red} be the set of variables of X whose D-sequents are in Ω. Let $Y = \mathit{Vars}(F) \setminus X$. *DDS* branches only on a subset of free (*i.e.*, unassigned) variables of X and Y. A free variable $x \in X \setminus \mathit{Vars}(\boldsymbol{q})$ is picked for branching only if $x \notin X_{red}$.

Although Boolean Constraint Propagation (BCP) is not shown explicitly in Fig. 3, it is included into the *pick_variable* procedure as follows: a) preference is given to branching on variables of unit clauses of $F_{\boldsymbol{q}}$ (if any); b) if v is a variable of a unit clause of C of $F_{\boldsymbol{q}}$ and v is picked for branching, then the value falsifying C is assigned first to cause immediate termination of this branch. In the description of *DDS* of Fig. 3, the left branch always explores assignment $v = 0$. But, obviously, v can be first assigned value 1.

To simplify making the branching variable v redundant when merging results of the left and right branches (see Sect. 7.5), *DDS first assigns values to variables of* Y. This means that *pick_variable* never selects a variable $x \in X$ for branching, if there is an unassigned variable of Y. In particular, BCP does not assign values to variables of X if there are unassigned variables of Y.

7.4 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.

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

Fig. 5 *Merge* procedure

Let $\boldsymbol{s} \rightarrow \{x\}$ be a D-sequent of the set Ω computed by *DDS* in the left branch $v = 0$ (line 4 of Fig. 3). 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 the subspace specified by $\boldsymbol{q_1}$. *DDS* splits the set Ω into subsets Ω^{sym} and Ω^{asym} of D-sequents symmetric and asymmetric with respect to variable v (line 5). 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 $X_{red} \setminus Vars(\boldsymbol{q})$ whose redundancy is stated by D-sequents of Ω^{sym} and Ω^{asym} respectively. Before exploring the right branch (line 8), 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 $Vars(C) \cap X^{sym} \neq \emptyset$.

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 6). Pruning the right branch when X^{asym} is empty is similar to non-chronological backtracking well known in SAT-solving [23].

7.5 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 10–13 of Fig. 3. 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 $x \in X \setminus Vars(\boldsymbol{q})$ 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 Sect. 7.2. If $v \notin Vars(C_1)$, then *resolve_clauses* (line 10) returns C_1 itself since C_1 is falsified by $\boldsymbol{q}$ and no new clause is added to F. (The situation $v \notin Vars(C_0)$ is impossible because *DDS* does not branch after a clause is falsified).

If at least one branch returns answer *sat*, then *DDS* calls procedure *merge* described in Fig. 5. First, *merge* takes care of the variables of X^{asym} (see Sect. 7.4). 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[3] 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

[3] The description of this case given in [17] 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-sequent produced for subspace $\boldsymbol{q}$ may turn out to be uncomposable.

$\boldsymbol{s} \rightarrow \{x\}$. First, the right branch assignment $v = 1$ is added to $\boldsymbol{s}$. 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 Sect. 7.3, 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 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, $F_{\boldsymbol{q}}$ can only contain unit clauses depending on v. Moreover, these unit clauses cannot contain literals of both polarities of v because *merge* is called only when either branch $v = 0$ or $v = 1$ is satisfied. Therefore, v is monotone. An atomic D-sequent S stating the redundancy of v is built as described in Sect. 7.2 and added to Ω (line 2). Then *merge* terminates returning Ω.

7.6 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 $\emptyset \rightarrow \{x\}$ is derived; e) the D-sequents derived by *DDS* are composable, which implies that the D-sequent $\emptyset \rightarrow X$ holds for the formula $\exists X[F]$ returned by *DDS*.

Proposition 11 *DDS is sound and complete.*

8 Compositionality of *DDS*

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

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

A QE algorithm based on enumeration of satisfying assignments is not compositional. The reason is that the set of assignments satisfying F is a Cartesian product of those satisfying

$F_i, i = 1, \ldots, k$. So if, for example, all F_i are identical (modulo variable renaming), the complexity of an enumeration based QE algorithm is *exponential* in k. A QE algorithm based on BDDs [7] is compositional only for variable orderings where variables of F_i and F_j, $i \neq j$ do not interleave.

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

Proposition 12 (Compositionality of DDS) *Let T be the search tree built by DDS when solving the QE problem $\exists X[F_1 \wedge \ldots \wedge F_k]$ above. Let $X_i = X \cap \mathit{Vars}(F_i)$ and $Y_i = \mathit{Vars}(F_i) \setminus X$. The size of T in the number of nodes is bounded by $|\mathit{Vars}(F)| \cdot (\eta(X_1 \cup Y_1) + \ldots + \eta(X_k \cup Y_k))$ where $\eta(X_i \cup Y_i) = 2 \cdot 3^{|X_i \cup Y_i|} \cdot (|X_i| + 1)$, $i = 1, \ldots, k$ no matter how decision branching variables are chosen.*

Proposition 12 is proved in the appendix for a slightly modified version of *DDS*. Notice that the compositionality of *DDS* is not ideal. For example, if all subformulas F_i are identical, *DDS* is *quadratic* in k as opposed to being linear. Informally, *DDS* is compositional because D-sequents it derives have the form $\boldsymbol{s} \rightarrow \{x\}$ where $\mathit{Vars}(\boldsymbol{s}) \cup \{x\} \subseteq \mathit{Vars}(F_i)$. The only exception are D-sequents derived when the current assignment falsifies a clause of F. This exception is the reason why *DDS* is quadratic in k.

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

Notice that a QE algorithm that resolves out variables one by one as in the DP procedure [13] is also compositional. (If $\mathit{Vars}(F_i) \cap \mathit{Vars}(F_j) = \emptyset$, then clauses of F_i and F_j cannot be resolved with each other). However, although such an algorithm may perform well on some classes of formulas, it is not very promising overall. This is due to the necessity to eliminate a variable in one big step, which may lead to generation of a very large number of new resolvent clauses. On the contrary, being a branching algorithm, *DDS* is very opportunistic and eliminates the same variable differently in different subspaces trying to reduce the number of new resolvents to be added (if any). The lack of flexibility in variable elimination is exactly the cause of the poor scalability of the DP procedure in SAT-solving. There is no reason to believe that DP-like procedures will scale better for the harder problem of quantifier elimination.

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

9 Experimental results

In this section, we describes results of experiments with an implementation of *DDS*. In Sect. 9.1, the implementation of *DDS* tested in experiments is described in more detail. In Sect. 9.2, some problems arising in model checking are used to compare *DDS* with other QE based algorithms. Section 9.3 showcases the compositionality of *DDS*. Finally, Sect. 9.4 compares backward model checking based on DDS and BDDs.

9.1 Some implementation details

In this section, we describe some features of the implementation of *DDS* we used in experiments.

- In Fig. 3, *DDS* is described in terms of recursive calls. It is more convenient to consider our implementation of *DDS* as building a search tree. Let n be the node of the search tree built by *DDS* 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 the algorithm described in Fig. 3.
- When implementing *DDS* 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 our implementation of *DDS* easy to modify, we did not use optimization techniques like employing watched literals to speed up BCP, special representation of two-literal clauses and so on. We do not think, however, that the presence of these techniques is crucial for *DDS*. There are good reasons to believe that D-sequent re-using discussed below will have much more dramatic effect on performance of *DDS* than any of the optimization techniques above.
- In Fig. 3, 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 lead to dramatic performance improvements. The reason is that a D-sequent S stating redundancy of $x \in X$ that is currently inactive 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. However, re-using D-sequents is not as simple as re-using clauses. In particular, our research showed that re-using D-sequents indiscriminately may lead to circular reasoning. This problem is easily solved by imposing some restrictions on the order in which D-sequents are re-used. Nevertheless, finding the best way to implement such restrictions needs further research.
- In Fig. 3, 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 our implementation of *DDS*, 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 our implementation of *DDS* keeps a resolvent clause C only if it is an empty clause of F or if in the node of the search tree located at depth $Depth(C)$

Table 1 Experiments with model checking formulas

Model checking mode	*EnumSA*		*C2D*		*QE-GBL*		*DDS*	
	Solved (%)	Time (s)	Solved (%)	Time (s)	Solved (%)	Time (s)	Solved (%)	Time (s)
Forward	425(56)	466	538 (71)	10,314	561 (74)	4,865	664 (88)	1, 530
Backward	97(12)	143	333 (44)	5,172	522 (68)	2,744	563 (74)	554

The time limit is 1 min

- 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, our implementation 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.

9.2 Comparison of various QE algorithms

In this subsection, we compare *DDS* with three other QE algorithms. The first QE algorithm of the three is based on enumeration of satisfying assignments [6] (courtesy of Andy King). We will refer to it as *EnumSA*.

The second QE algorithm that we compared with *DDS* is called *C2D* [30]. It is based on the idea of compiling CNF to deterministic decomposable negation normal forms [11].

The third algorithm we used in the experiments of this section is based on explicit elimination of boundary points [16]. We will refer to this algorithm as *QE-GBL*. Here *GBL* stands for *global*. Given a formula $\exists X[F]$, *QE-GBL* eliminates variables of X globally, one by one, as in the DP procedure. However, when resolving out a variable $x \in X$, *QE-GBL* adds a new resolvent to F *only if* it eliminates an $\{x\}$-removable $\{x\}$-boundary point of F. Variable x is redundant in $\exists x[F]$ if all $\{x\}$-removable $\{x\}$-boundary points of F are eliminated. *QE-GBL* does not generate so many redundant clauses as DP, but still has the flaw of eliminating variables globally.

We used *QE-GBL* for two reasons. First, *DDS* can be viewed as a branching version of *QE-GBL*. In Sect. 8, we argued that branching gives *DDS* more flexibility in variable elimination in comparison to procedures eliminating variables globally. So we wanted to confirm that *DDS* indeed benefited from branching. Second, one can consider *QE-GBL* as an algorithm similar to that of [20]. 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 [20] since it was not implemented as a stand-alone tool. Our implementation of *QE-GBL* was quite efficient. In particular, we employed Picosat [4] for finding boundary points.

In this subsection, we describe two experiments with the 758 model checking benchmarks of HWMCC'10 competition [31]. In the first experiment (the first line of Table 1) we used *EnumSA*, *C2D*, *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 $\boldsymbol{p}$

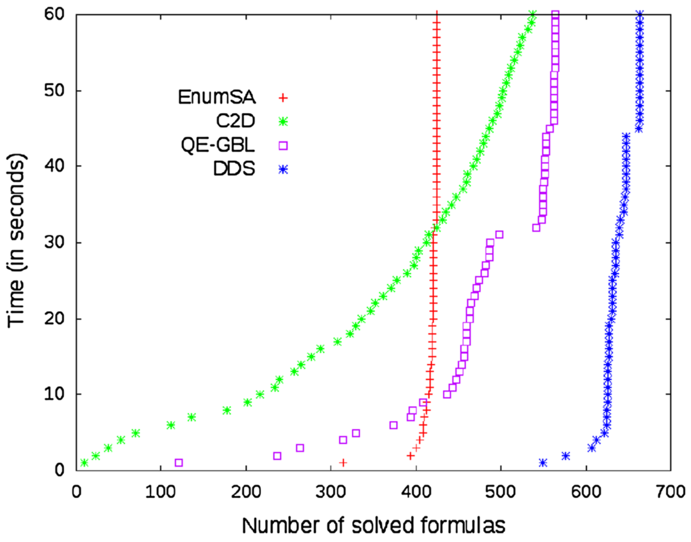


Fig. 6 Forward model checking (1 iteration)

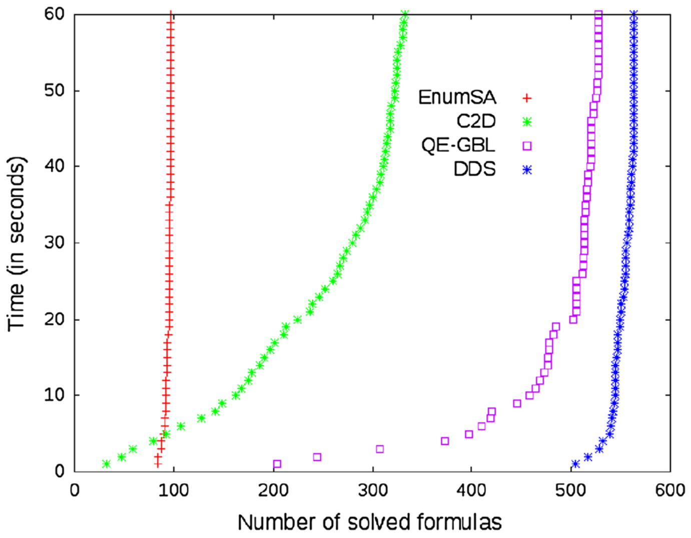


Fig. 7 Backward model checking (1 iteration)

to *Vars*(F), the property in question fails 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).

Table 1 shows the comparison of the four 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-min time limit, *DDS* solved more formulas than *EnumSA*, *C2D* and *QE-GBL* in forward and backward model checking. Figures 6 and 7 give the number of formulas of Table 1 solved by the four programs in t seconds, $0 \leq t \leq 60$. These figures show the superiority of DDS over *EnumSA*, *C2D* and *QE-GBL* on the set of formulas we used.

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

9.3 Compositionality of DDS

In this subsection, we describe an experiment showcasing the compositionality of *DDS*. We also confirm that *EnumSA* is not compositional (see Sect. 8).

In this experiment, *DDS* and *EnumSA* computed the output assignments produced by a combinational circuit N composed of small *identical* circuits $N_1, \dots, N_k$ with independent sets of variables. In this case, one needs to eliminate quantifiers from $\exists X[F]$ where $F = F_1 \wedge \dots \wedge F_k$. 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 h. Notice that *DDS* was able to quickly solve all the formulas even with the random choice of branching variables.

9.4 Backward model checking based on DDS and BDDs

In this subsection, we compare backward model checkers based on DDS and BDDs [7]. Our implementation of a model checker based on DDS was straightforward: *DDS* was used to compute backward images until an initial state or a fixed point were reached. We will refer to the model checker based on *DDS* as *MC-DDS*.

Table 2 Compositionality of QE algorithms

#Copies	#Vars, #clauses	$\lvert Y \rvert$	*EnumSA* (s)	*DDS* rand (s)	*DDS* (s)
5	20, 30	10	0	0.01	0.01
10	40, 60	20	10.5	0.01	0.01
15	60, 90	30	>1 h	0.01	0.01
500	2,000, 3,000	1, 000	>1 h	1.95	0.04

Time limit = 1 h

Table 3 Results on examples solved by *MC-DDS*

Mod. checker	MC-BDD	MC-DDS
#Solved	193	247
#Timeouts	54	0
Time for solved by both (s)	9,080	11,293

The time limit is 2,000 s

In experiments, we used the BDD-based model checker incorporated into the latest version of a tool called PdTrav [32] (courtesy of Gianpiero Cabodi). We ran PdTrav in the backward model checking mode with ternary simulation turned off (as a non-BDD optimization). The other non-BDD optimizations, e.g. computation of the cone of influence, remained active because there was no way to switch them off. Since *DDS* maintains a single search tree, we also forced PdTrav to represent the transition relation by a monolithic BDD. (A D-sequent based QE algorithm *does not have to* build a single search tree e.g. it can employ restarts. However, using restarts requires storing and re-using D-sequents). We will refer to PdTrav with the options above as *MC-BDD*.

We ran *MC-DDS* and *MC-BDD* on the 758 benchmarks of the HWMCC-10 competition. With the time limit of 2,000 seconds, *MC-BDD* and *MC-DDS* solved 374 and 247 benchmarks respectively. This is not surprising taking into account the maturity of current BDD algorithms and their re-using of learned information via subgraph hashing. An important fact however is that *MC-DDS* solved many problems that *MC-BDD* could not.

Table 3 shows the results of both model checkers on the 247 benchmarks solved by *MC-DDS* (i.e. *favored* by *DDS*). The second line of this table gives the number of benchmarks solved under 2,000s. The third line shows how many examples out of 247 were not solved in the time limit. The last line gives the total time in seconds for the benchmarks solved by both model checkers. Table 3 shows that a large number of problems solved by *MC-DDS* were

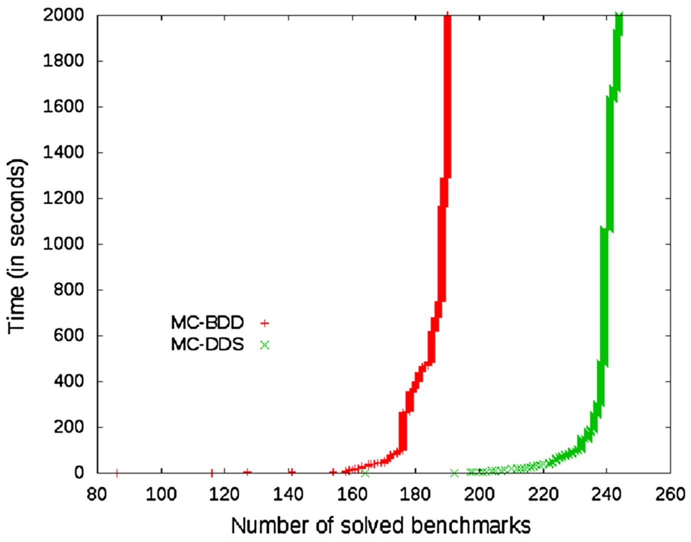


Fig. 8 Performance of model checkers on 247 examples solved by *MC-DDS*

Table 4 Comparison on some concrete benchmarks solved by *MC-DDS*

Benchmark	#Latches	#Gates	#Iterations	Bug	MC-BDD (s)	MC-DDS (s)
pdtvistwoall0	33	1, 713	2	No	Timeout	**41**
pdtvisvending02	34	958	5	No	**0.7**	1,635
bj08amba4g5	36	13, 637	4	No	Timeout	**113**
pdtvisbakery3	48	7, 514	2	Yes	**1.3**	12
texasifetch1p5	57	663	21	Yes	**1.5**	368
visprodcellp01	78	2, 885	5	No	19	**7.9**
pdtpmss1269b	99	811	3	No	Timeout	**64**
texasparsesysp1	312	12, 173	10	Yes	**0.7**	231
bobmiterbm1or	381	3, 720	1	Yes	0.7	**0.1**
bobsynthand	3, 015	15, 397	2	No	1.2	**0.6**
mentorbm1and	4, 344	31, 684	2	No	Timeout	**1.8**

The time limit is 2,000 s
Bold value shows the fastest time out of the results of the two last columns

hard for *MC-BDD*. Figure 8 gives the performance of both model checkers on the benchmarks solved by *MC-DDS* in terms of the number of problems finished in a given amount of time.

Results of the model checkers on some concrete benchmarks solved by *MC-DDS* are given in Table 4. The column *iterations* shows the number of backward images computed by the algorithms before finding a bug or reaching a fixed point.

10 Related work

The relation between a resolution proof and the process of elimination of boundary points was described in [15]. In terms of the present paper, [15] dealt only with a special kind of Z-boundary points of formula F where $|Z| = 1$. In the present paper, we consider the case where Z is an arbitrary subset of the set of quantified variables X of an $\exists$CNF formula $\exists X[F]$. This extension is crucial for describing the semantics of D-sequents. The notion of D-sequents was introduced in [18]. There, we formulated a QE algorithm that branched only on quantified variables of $\exists X[F]$. This algorithm is more complex than *DDS* because it computed boundary points explicitly.

As far as quantifier elimination is concerned, QE algorithms and QBF solvers can be partitioned into two categories. (Although, in contrast to a QE algorithm, a QBF-solver is a decision procedure, they both employ methods of quantifier elimination. Since this paper is focused on SAT-based solvers, we omit references to papers on QE algorithms that use BDDs [7,8].) The members of the first category employ various techniques to eliminate quantified variables of the formula one by one in some order [1–3,20,29]. For example, in [20], quantified variables are eliminated by interpolation [26]. All these solvers face the problem that we already discussed in Sect. 8. The necessity to eliminate a variable in one big step deprives the algorithm of flexibility and, in general, leads to generation of prohibitively large sets of clauses.

The solvers of the second category are based on enumeration of satisfying or unsatisfying assignments [6,14,21,25,28]. 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.

11 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).

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

Appendix

The appendix contains proofs of the propositions listed in the paper. We also give proofs of lemmas used in the proofs of propositions.

Propositions of Sect. 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 and we have a 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 and we have a 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 $(\exists X[F])_{\boldsymbol{y}} \neq (\exists X[H])_{\boldsymbol{y}}$, which leads to a 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 $\boldsymbol{p}$. So we have a 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 a 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, which leads to a contradiction. □

Propositions of Sect. 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* $\mathit{Vars}(F)$. *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 $\mathit{Vars}(F) \setminus X$. 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 $X \setminus \mathit{Vars}(\boldsymbol{q})$. 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, C is not implied by F and we have a 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* $Z \subseteq (X \setminus \mathit{Vars}(\boldsymbol{q}))$. *Let a variable* v *of* $X \setminus (\mathit{Vars}(\boldsymbol{q}) \cup Z)$ *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 10 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 and we have a 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 8) .
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 $\boldsymbol{p}$ is a $\{v\}$-removable $\{v\}$-boundary point of $F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z$. So v is not locally redundant in $\exists X[F_{\boldsymbol{q}} \setminus (F_{\boldsymbol{q}})^Z]$ and we have a 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, which leads to a contradiction. □

Propositions of Sect. 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. (That is $\boldsymbol{p}'$ 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 and we have a contradiction. □

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 $\exists X[F_{\boldsymbol{q}}]$.*

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 the value of v, $\boldsymbol{p}$ is not $\{v\}$-removable in $F_{\boldsymbol{q}}$. So we have a 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 of variable v. Hence v is not a blocked variable of $F_{\boldsymbol{q}}$ and we have a contradiction. □

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 $\exists X[F_{\boldsymbol{q}}]$.*

Proof Let $X' = X \setminus Vars(\boldsymbol{q})$. 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, which leads to a contradiction. 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 8, 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 Sect. 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 $(\exists X[H], \boldsymbol{q}, W) \rightarrow Z$ does not. According to Definition 13, 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 $H_{\boldsymbol{q}}$. So we have a 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 $Z' \subseteq Z$. So we have a contradiction again.
- $\boldsymbol{p}$ 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 $H_{\boldsymbol{q}}$. Thus we have a 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, which leads to a 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. So we have a 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* $\mathit{Vars}(\boldsymbol{r}) \cap W = \emptyset$*. 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. So we have a 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* $\mathit{Vars}(\boldsymbol{q}) \cap Z = \emptyset$ *and* $\mathit{Vars}(\boldsymbol{q}) \cap W = \emptyset$*. 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 Sect. 7: description of *DDS*

In this section, we prove the correctness of *DDS* (Proposition 11). First we introduce a few new definitions and prove a few lemmas.

Definition 18 Let $\exists X[F]$ be an ∃CNF formula, $\boldsymbol{q}$ be an assignment to $\mathit{Vars}(F)$ and $Z \subseteq (X \setminus \mathit{Vars}(\boldsymbol{q}))$. We will call $(\exists X[F], \boldsymbol{q}, W) \rightarrow Z$ a **single-variable D-sequent** if $|Z|$=1.

Definition 19 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
- $(\mathit{Vars}(\boldsymbol{q'}) \cup \mathit{Vars}(\boldsymbol{q''})) \cap (W' \cup W'' \cup \{v'\} \cup \{v''\}) = \emptyset$

Definition 20 Let Ω be a set of single-variable D-sequents for an ∃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 21 Let Ω be a set of compatible D-sequents for an ∃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 22 Let Ω be a set of compatible D-sequents for an ∃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 23 Let Ω be a set of compatible D-sequents for an ∃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 Fig. 5. Let* Ψ,Ψ_0,Ψ_1 *be subsets of* Ω, Ω_0, Ω_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. So we have a contradiction. □

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. So we have a 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 formula $F_{\boldsymbol{s}} \setminus (F_{\boldsymbol{s}})^{X^{\Psi}}$. Then Proposition 5 entails that v is redundant in $F_{\boldsymbol{s}} \setminus (F_{\boldsymbol{s}})^{X^{\Psi}}$. Since, by construction, $\boldsymbol{a^{\Psi}} \subseteq \boldsymbol{s}$, 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 □

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 Sect. 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 proving redundancy of both v' and v'' involves circular reasoning where $\{v'\}$-clauses are used to prove redundancy of variable v'' and vice versa. *DDS* avoids circular 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 S' and S'' are D-sequents derived when 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 after the right branch is finished). 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 $(\exists X[F], \emptyset, X_{i_1}) \rightarrow \{x_{i_1}\}, \ldots, (\exists X[F], \emptyset, X_{i_k}) \rightarrow \{x_{i_k}\}$. Here $x_{i_1}, \ldots, x_{i_k}$ 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 and so the variables of X are redundant in the formula $\exists X[F]$ returned by *DDS*.

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 *DDS* 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_0^1 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 Sect. 8: compositionality of *DDS*

Definition 24 We will refer to D-sequents derived due to appearance of an empty clause in formula $F_{\boldsymbol{q}}$ (see Sect. 7.2) 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 number of nodes in T 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 $Vars(F) \setminus X$.

Let P be a path of T and $n(v)$ be a node of T 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 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 of 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 variable of $X \setminus \{x\}$ 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, the current formula, *i.e.*, formula $F_{\boldsymbol{q}} \setminus F_{\boldsymbol{q}}^{X'}$ can only contain unit clauses that depend 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 and we have 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. Thus we have a contradiciton again.

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 $F_i, i = 1, \dots, k$ is bounded by $|X_1| \cdot 3^{|X_1 \cup Y_1|} + \dots + |X_k| \cdot 3^{|X_k \cup Y_k|}$. 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 that either satisfy $\{x\}$-clauses of F or are the reason for redundancy of $\{x\}$-clauses. 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 15, 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 □

References

1. Abdulla P, Bjesse P, Eén N (2000) symbolic reachability analysis based on sAT-solvers. In: Proceedings of the 6th international conference on tools and algorithms for construction and analysis of systems, TACAs'00, pp 411–425
2. Ayari A, Basin D (2002) Qubos: deciding quantified boolean logic using propositional satisfiability solvers. In: Proceedings of 4th international conference on formal methods in computer-aided design, vol 2517 of LNCS, FMCAD'02, pp 187–201
3. Biere A (2004) Resolve and expand. In: Procedings of the seventh international conference on theory and applications of satisfiability testing, SAT'04, pp 59–70
4. Biere A (2008) Picosat essentials. J Satisf Boolean Model Comput 4(2–4):75–97
5. Bradley AR (2011) Sat-based model checking without unrolling. In: Proceedings of the 12th international conference on verification, model checking, and abstract interpretation, VMCAI'11, pp 70–87
6. Brauer J, King A, Kriener J (2011) Existential quantification as incremental sat. In: Proceedings of the 23rd international conference on computer aided verification, CAV'11, Springer-Verlag, pp 191–207
7. Bryant R (1986) Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput C–35(8):677–691

8. Chauhan P, Clarke E, Jha S, Kukula J, Veith H, Wang D (2001) Using combinatorial optimization methods for quantification scheduling. In: Proceedings of the 11th IFIP WG 10.5 advanced research working conference on correct hardware design and verification methods, CHARME '01, pp 293–309
9. Clarke E, Emerson A (1982) Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Proceedings of logic of programs, Workshop, pp 52–71
10. Clarke E, Grumberg O, Peled D (1999) Model checking. MIT Press, Cambridge
11. Darwiche A (2001) Decomposable negation normal form. J ACM 48:608
12. Davis M, Logemann G, Loveland D (1962) A machine program for theorem proving. Commun ACM 5(7):394–397
13. Davis M, Putnam H (1960) A computing procedure for quantification theory. J ACM 7(3):201–215
14. Ganai M, Gupta A, Ashar P (2004) Efficient sat-based unbounded symbolic model checking using circuit cofactoring. In: Proceedings of the 2004 IEEE/ACM international conference on computer-aided design, ICCAD'04, pp 510–517
15. Goldberg E (2009) Boundary points and resolution. In: Proceedings of theory and applications of satisfiability testing, 12th international conference, SAT'09, pp 147–160
16. Goldberg E, Manolios P (2011) Sat-solving based on boundary point elimination. In: Proceedings of 6th international Haifa Verification Conference, pp 93–111
17. Goldberg E, Manolios P (2012) Quantifier elimination by dependency sequents. In: Proceedings of formal methods in computer-aided design, FMCAD'12, pp 34–44
18. Goldberg E, Manolios P (2012) Removal of quantifiers by elimination of boundary points. Technical Report. arXiv:1204.1746v2 [cs.LO], Northeastern University
19. Goldberg E, Manolios P (2013) Quantifier elimination via clause redudnancy. In: Proceedings of formal methods in computer-aided design, FMCAD'13, pp 85–92
20. Jiang R (2009) Quantifier elimination via functional composition. In: Proceedings of the 21st international conference on computer aided verification, CAV'09, pp 383–397
21. Jin H, Somenzi F (2005) Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In: Proceedings of the 42nd annual design automation conference, DAC'05, pp 750–753
22. Kullmann O (1999) New methods for 3-sat decision and worst-case analysis. Theor Comput Sci 223(1–2):1–72
23. Marques-Silva J, Sakallah K (1996) Grasp: a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM international conference on computer-aided design, ICCAD'96, pp 220–227
24. McMillan K (1993) Symbolic model checking. Kluwer Academic Publishers, Norwell
25. McMillan K (2002) Applying sat methods in unbounded symbolic model checking. In: Proceedings of the 14th international conference on computer aided verification, CAV'02, pp 250–264
26. McMillan K (2003) Interpolation and sat-based model checking. In: Proceedings of computer aided verification, 15th international conference, CAV'03, Springer, pp 1–13
27. Moskewicz M, Madigan C, Zhao Y, Zhang L, Malik S (2001) Chaff: engineering an efficient sat solver. In: Proceedings of the 38th annual design automation conference, DAC'01, pp 530–535
28. Plaisted D, Biere A, Zhu Y (2003) A satisfiability procedure for quantified Boolean formulae. Discret Appl Math 130(2):291–328
29. Williams P, Biere A, Clarke E, Gupta A (2000) Combining decision diagrams and sat procedures for efficient symbolic model checking. In: Proceedings of the 12th international conference on computer aided verification, CAV'00, pp 124–138
30. C2D. http://reasoning.cs.ucla.edu/c2d. Accessed 19 May 2013
31. HWMCC-2010 benchmarks. http://www.fmv.jku.at/hwmcc10/benchmarks.html. Accessed 3 April 2013
32. http://www.fmgroup.polito.it/index.php/download/. Accessed 5 May 2013