Variable Independence and Resolution Paths for Quantified Boolean Formulas

Allen Van Gelder
http://www.cse.ucsc.edu/∼avg

University of California, Santa Cruz

Abstract. Variable independence in quantified boolean formulas (QBFs) informally means that the quantifier structure of the formula can be rearranged so that two variables reverse their outer-inner relationship without changing the value of the QBF. Samer and Szeider introduced the standard dependency scheme and the triangle dependency scheme to safely over-approximate the set of variable pairs for which an outer-inner reversal might be unsound (JAR 2009).
This paper introduces resolution paths and defines the resolution-path dependency relation. The resolution-path relation is shown to be the root (smallest) of a lattice of dependency relations that includes quadrangle dependencies, triangle dependencies, strict standard dependencies, and standard dependencies. Soundness is proved for resolution-path dependencies, thus proving soundness for all the descendants in the lattice.
It is shown that the biconnected components (BCCs) and block trees of a certain clause-literal graph provide the key to computing dependency pairs efficiently for quadrangle dependencies. Preliminary empirical results on the 568 QBFEVAL-10 benchmarks show that in the outermost two quantifier blocks quadrangle dependency relations are smaller than standard dependency relations by widely varying factors.

1 Introduction

Variable independence in quantified boolean formulas (QBFs) informally means that two variables that are adjacent in the quantifier structure can exchange places without changing the value of the QBF. The motivation for knowing such shifts are sound (i.e., cannot change the value of a closed QBF, which is *true* or *false*) is that QBF solvers have more flexibility in their choice of which variable to select for a solving operation. They are normally constrained to obey the quantifier order.

Samer and Szeider introduced dependency schemes to record dependency pairs (p, q) such that q is inner to p in the quantifier structure and any rearrangement that places q outer to p *might* be unsound. The absence of (p, q) ensures that there *is* some sound rearrangement that places q outer to p [6]. The idea is that the pairs in a dependency scheme can be computed with reasonable effort, and are a safe over-approximation of the exact relation that denotes unsound rearrangements of quantifier order. A smaller dependency scheme allows

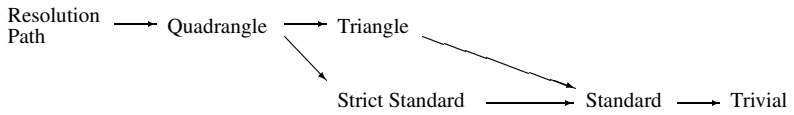

Fig. 1. Lattice of dependency relations.

more pairs to be treated as independent. They proposed two nontrivial schemes, the "standard" dependency scheme, which is easiest to compute, but coarse, and the "triangle" dependency scheme, which is more refined. Lonsing and Biere have reported favorable results on an implementation of the "standard" dependency scheme [5]. We are not aware of any implementation of triangle dependencies. Lonsing and Biere provide additional bibliography and discussion of other approaches for increasing solver flexibility.

This paper introduces ***resolution paths*** in Section 4 to define a dependency relation that is smaller than those proposed by Samer and Szeider. Resolution paths are certain paths in the resolution graph [7] associated with the quantifier-free part of the QBF. A hierarchy of new relations is introduced, called resolution-path dependencies (smallest), quadrangle dependencies, and strict standard dependencies. Quadrangle dependencies refine the triangle dependencies; strict standard dependencies refine standard dependencies. The resulting lattice is shown in Figure 1. Soundness is proved for resolution-path dependencies, thus proving soundness for all the descendants in the hierarchy. A slightly longer version of this paper contains some details omitted here, due to the page limit.[1]

The main obstacle is computing the dependency relation for anything more refined than standard dependencies or strict standard dependencies. Samer and Szeider sketched a polynomial-time algorithm, which enabled them to get interesting theoretical results involving triangle dependencies and back-door sets. It appears to be too inefficient for practical use on large QBF benchmarks and, to the best of our knowledge, it has not been implemented.

Samer and Szeider used a certain undirected graph, similar to what is called the clause-variable incidence graph in the literature, for their algorithm. This *clause-literal graph*, as we shall call it, is normally already represented in the data structures of a solver, as occurrence lists, and is practical to use for the standard dependency relation [5]. It is easy to see standard dependencies (and strict standard dependencies) are based on the connected components (CCs) of this graph. Strict standard dependencies, introduced in Definition 5.2, are essentially a cost-free improvement on standard dependencies, once this fact is recognized.

This paper shows in Section 6 that the ***biconnected components*** **(BCCs)** of the clause-literal graph provide the key to identifying dependency pairs for

[1] Please see `http://www.cse.ucsc.edu/~avg/QBFdeps/` for a more detailed version of this paper and a prototype program.

quadrangle dependencies, introduced in Definition 5.2. Like CCs, BCCs can be computed in time linear in the graph size. Based on the BCC structure, the clause-literal graph can be abstracted into a ***block tree***, so-called in the literature.

Quadrangle dependencies can be determined by paths in the block tree, which is normally much smaller than the clause-literal graph. Our algorithm could be modified to compute triangle dependencies, but this would cost the same as quadrangle dependencies, and produce less independence, so this modification has not been implemented. We avoid calling the quadrangle dependency relation a dependency *scheme* to avoid conflicting with the technical requirements stated by Samer and Szeider [6]

In a prototype `C++` implementation that builds dependency relations, computing BCCs was found to be as cheap as computing connected components (needed for any dependency relation), on the 568 QBFEVAL-10 benchmarks. Preliminary empirical results are given in Section 7, mainly consisting of statistics about the BCC structure and size of quadrangle dependency relations in these benchmarks.

The primary goal of this work to provide methods by which practical QBF solvers can soundly carry out a broader range of the operations they already perform. (Readers should be familiar with QBF solver operations to follow these paragraphs, or come back after reading Section 2.) The *universal reduction* operation is ubiquitous in QBF solvers. The standard requirement is that all existential literals must be independent of the universal literal u to be deleted in the *trivial* dependency relation. Theorem 4.9 shows that independence in the quadrangle relation is sufficient. Search-based QBF solvers make variable assignments as assumptions (the word "decision" is often used). Normally, an existential variable can be selected only if it is independent of *all* unassigned universal variables in the *trivial* dependency relation. Theorem 4.7 shows that independence in the quadrangle relation is sufficient.

2 Preliminaries

In general, *quantified boolean formulas* (QBFs) generalize propositional formulas by adding universal and existential quantification of boolean variables. See [3] for a thorough introduction and a review of any unfamiliar terminology. A *closed* QBF evaluates to either 0 (*false*) or 1 (*true*), as defined by induction on its principal operator.

1. $(\exists\, x\, \phi(x)) = 1$ iff $(\phi(0) = 1$ or $\phi(1) = 1)$.
2. $(\forall\, x\, \phi(x)) = 0$ iff $(\phi(0) = 0$ or $\phi(1) = 0)$.
3. Other operators have the same semantics as in propositional logic.

This definition emphasizes the connection of QBF to two-person games, in which player E (Existential) tries to set existential variables to make the QBF evaluate to 1, and player A (Universal) tries to set universal variables to make the QBF evaluate to 0 (see [4] for more details).

For this paper QBFs are in ***prenex conjunction normal form* (PCNF)**, i.e., $\Psi = \overrightarrow{Q}.\mathcal{F}$ consists of prenex $\overrightarrow{Q}$ and clause matrix $\mathcal{F}$. Clauses may be written enclosed in square brackets (e.g., $[p, q, \overline{r}]$). Literals are variables or negated variables, with overbar denoting negation. Usually, letters e and others near the beginning of the alphabet denote existential literals, while letters u and others near the end of the alphabet denote universal literals. Letters like p, q, r denote literals of unspecified quantifier type. The variable underlying a literal p is denoted by $|p|$ where necessary.

The quantifier prefix is partitioned into *quantifier blocks* of the same quantifier type. Each quantifier block has a unique *qdepth*, with the outermost block having qdepth = 1.

The proof system known as *Q-resolution* consists of two operations, *resolution* and *universal reduction*. Q-resolution is of central importance for QBFs because it is a sound and complete proof system [2]. Resolution is defined as usual, except that the clashing literal is always existential; universal reduction is special to QBF. Let α, β, and γ be possibly empty sets of literals.

$$\mathsf{res}_e(C_1, C_2) = \alpha \cup \beta \qquad \text{where } C_1 = [e, \alpha]\,,\ C_2 = [\overline{e}\,,\ \beta] \tag{1}$$

$$\mathsf{unrd}_u(C_3) = \gamma \qquad \text{where } C_3 = [C]_3 = [u, \gamma] \tag{2}$$

Resolvents must be non-tautologous for Q-resolution. $\mathsf{unrd}_u(C_3)$ is defined only if u is ***tailing*** for γ, which means that the quantifier depth (qdepth) of u is greater than that of any existential literal in γ.

A ***Q-derivation***, often denoted as π, is a directed acyclic graph (DAG) in which each node is either an input clause (a DAG leaf), or a proof operation (an internal node) with a specified clashing literal or reduction literal, and edge(s) to its operand(s). A *Q-refutation* is a Q-derivation of the empty clause.

An ***assignment*** is a partial function from variables to truth values, and is usually represented as the set of literals that it maps to *true*. Assignments are denoted by ρ, σ, τ, etc. Applications of an assignment σ to a logical expression are denoted by $q\lceil_\sigma$, $C\lceil_\sigma$, $\mathcal{F}\lceil_\sigma$, etc. If σ assigns variables that are quantified in Ψ, those quantifiers are deleted in $\Psi\lceil_\sigma$, and their variables receive the assignment specified by σ.

3 Regular Q-Resolution

In analogy with regular resolution in propositional calculus, we define Q-resolution to be regular if no variable is resolved upon more than once on any path in the proof DAG. We need the following property for analyzing resolution paths.

Theorem 3.1 Regular Q-resolution and regular tree-like Q-resolution are complete for QBF.

Proof: The proof for regular Q-resolution is the same as in the paper that showed Q-resolution is complete for QBF [2]. It is routine to transform a regular Q-resolution derivation into a regular tree-like Q-resolution derivation of the same clause, by splitting nodes as needed, working from the leaves (original clauses) up. ■

4 Resolution Paths

This section defines resolution paths and resolution-path dependencies, then states and proves the main results in Theorem 4.7 and subsequent theorems. Let a closed PCNF $\Psi = \overrightarrow{Q}.\mathcal{G}$ be given in which the quantifier block at qdepth $d+1$ is existential. Consider the *resolution graph* $G = (V, E)$ defined as follows [7]:

Definition 4.1 The *qdepth-limited resolution graph* $G = (V, E)$ at qdepth $d+1$ is the undirected graph in which:

1. V, the vertex set, consists of clauses in $\mathcal{G}$ containing some existential literal of qdepth at least $d+1$;
2. E, the undirected edge set, consists of edges between clauses C_i and C_j in V, where there is a unique literal q such that $q \in C_i$ and $\overline{q} \in C_j$, so that C_i and C_j have a non-tautologous resolvent. Further, q is required to be existential and its qdepth must be $d+1$ *or greater*. Each edge is annotated with the variable that qualifies it as an edge.

A ***resolution path of depth*** $d+1$ is a path in G such that no two *consecutive* edges are annotated with the same variable. (Nonconsecutive edges with the same variable label are permitted and variable labels with qdepths greater than $d+1$ are permitted.) □

Definition 4.2 We say that a literal p *presses on* an existential literal q in the graph G defined in Definition 4.1 if there is a resolution path of depth $d+1$ connecting a vertex that contains $\overline{p}$ with a vertex that contains q without using an edge annotated with $|q|$. Similarly, p *presses on* $\overline{q}$ if there is a resolution path of depth $d+1$ connecting a vertex that contains $\overline{p}$ with a vertex that contains $\overline{q}$ without using an edge annotated with $|q|$. □

One may think of "presses on" as a weak implication chain: if all the clauses involved are binary, it actually is an implication chain. An example is discussed later in Example 5.4 and Figure 3 after some other graph structures have been introduced. The intuition is that if literal p presses on literal q, then making p *true* makes it more likely that q will need to be *true* to make a satisfying assignment. Theorem 4.7 shows that transposing the variable order in the quantifier prefix is sound, even though many combinations of pressing are present. Only certain combinations are dangerous.

We say that a sequence S' is a ***subsequence*** of a sequence S if every element in S' is also in S, in the same order as S, but not necessarily contiguous in S.

The next theorem shows that Q-resolution cannot bring together variables unless there is a "presses on" relationship in the original clauses. This suggests that resolution paths are the natural form of connection for variable dependencies.

Theorem 4.3 Let $\Psi = \overrightarrow{Q}.\mathcal{G}$ be a closed PCNF. Let π be a regular tree-like Q-resolution derivation from Ψ. For all literals p and for all *existential* literals f, if there is a clause (input or derived) in π that contains both p and f, then the order of sibling subtrees of π may be swapped if necessary so that a resolution

path from a clause with p to a clause with f appears as a subsequence of the leaves of π (not necessarily contiguous, but in order).

Proof: The proof is by induction on the subtree structure of π. The base case is that p and f are together in a clause of $\mathcal{G}$, say D_1, which is a leaf of π. Then D_1 constitutes a resolution path from p to f.

For any non-leaf subtree, say π_1, assume the theorem holds for all proper subtrees of π_1. That is, assume for all literals q and for all *existential* literals e, if there is a clause in a proper subtree of π_1, say π_2, that contains both q and e, then the subtrees of π_2 may be swapped so that a resolution path from a clause with q to a clause with e appears as a subsequence of the leaves of π_2.

Suppose that clause D_1, the root clause of π_1 contains both p and f. If p and f appear in a clause in a proper subtree of π_1, then the inductive hypothesis states that the needed resolution path can be obtained, so assume p and f do *not* appear together in any proper subtree of π_1.

Arrange the two principal subtrees of π_1 so that p is in the root clause of the left subtree and f is in the root clause of the right subtree (p and/or f might be in both subtrees). Let the clashing literal be g at the root of π_1. That is, g appears in the left operand and $\overline{g}$ appears in the right operand of the resolution whose resolvent is D_1.

By the inductive hypothesis, the left subtree has a resolution path P_L from a clause with p to a clause with g as a subsequence of its leaves. Also, the right subtree has a resolution path P_R from a clause with $\overline{g}$ to a clause with f as a subsequence of its leaves. Concatenate P_L and P_R (with the edge being labeled $|g|$) to give a resolution path from a clause with p to a clause with f. Since $|g|$ was a clashing literal at D_1, above the two subtrees, by regularity of the derivation, $|g|$ cannot appear as an edge label in either P_L or P_R, so the concatenation cannot have consecutive edges labeled with $|g|$. ■

We now consider when transposing adjacent quantified variables of different quantifier types in the quantifier prefix does not change the value of the QBF.

Definition 4.4 Let a closed PCNF $\Psi = \overrightarrow{Q}.\mathcal{G}$ be given in which the universal literal u is at qdepth d and the existential literal e is at some qdepth greater than d. The pair (u, e) satisfies the ***resolution-path independence criterion*** if (at least) one of the following conditions hold in the depth-limited graph G defined in Definition 4.1:

(A) u *does not* press on $\overline{e}$ and $\overline{u}$ *does not* press on $\overline{e}$; or
(B) $\overline{u}$ *does not* press on e and $\overline{u}$ *does not* press on $\overline{e}$.

If u and e are variables, the pair (u, e) satisfies the ***resolution-path independence criterion for variables*** if any of (u, e) or $(u, \overline{e})$ or $(\overline{u}, e)$ or $(\overline{u}, \overline{e})$ satisfies the resolution-path independence criterion for literals. □

Definition 4.5 Let universal u and existential e be variables, as in Definition 4.4. We say the pair (u, e) is a ***resolution-path dependency tuple*** if and only if (at least) one of the following conditions holds in G:

(C) u presses on e and $\overline{u}$ presses on $\overline{e}$; or

(D) $\overline{u}$ presses on e and u presses on $\overline{e}$.

Lemma 4.6 states that either this definition or Definition 4.4, but not both, applies for pairs (u, e) of the correct types and qdepths. □

Lemma 4.6 If u and e are universal and existential variables, respectively, then $(u,\ e)$ satisfies the resolution-path independence criterion for variables if and only if e does *not* have a resolution-path dependency upon u.

Proof: Apply DeMorgan's laws and distributive laws to the definitions. ■

We are now ready to state the main theoretical results of the paper. We use ***transpose*** in its standard sense to mean interchange of two adjacent elements in a sequence.

Theorem 4.7 Let a closed PCNF $\Psi = \overrightarrow{Q}.\mathcal{G}$ be given in which the universal literal u is at qdepth d and is adjacent in the quantifier prefix to the existential literal e at qdepth $d+1$. Let (u, e) satisfy the resolution-path independence criterion for literals (Definition 4.4). Then transposing $|u|$ and $|e|$ in the quantifier prefix does not change the value of Ψ.

Proof: It suffices to show that transposing u to a later position does not cause Ψ to change in value from 1 to 0. We show this holds for all assignments σ to all variables outer to u in Ψ. That is, let $\overrightarrow{Q}_{rem}$ be the suffix of $\overrightarrow{Q}$ beginning immediately after $\forall u\,\exists e$, and define

$$\Phi = \forall u\,\exists e\,\overrightarrow{Q}_{rem}.\mathcal{F}, \quad \text{where } \mathcal{F} = \mathcal{G}\lceil_{\sigma} \tag{3}$$

$$\Phi' = \exists e\,\forall u\,\overrightarrow{Q}_{rem}.\mathcal{F}. \tag{4}$$

Note that if the hypotheses (A) and (B) in Definition 4.4 hold for Ψ, then they also hold for Φ. Throughout this proof "A" and "B" refer to these conditions. Suppose Φ' evaluates to 0. By Theorem 3.1 there is a regular tree-like Q-refutation π' of Φ'. Note that π' has no redundant clauses; they all contribute to the refutation. Let us attempt to use π' as a starter for π, which we want to be a Q-refutation of Φ. For notation, any primed symbol (such as D') in Φ' or π' represents the corresponding unprimed symbol (such as D) in Φ or π.

What operation of π' can be incorrect for π? The only possibilities are a universal reduction involving a clause containing literals on both $|u|$ and $|e|$. In π', $|u|$ is tailing w.r.t. $|e|$, whereas in π it is not.

The key observation is that a regular tree-like Q-refutation derivation from Φ' cannot produce certain clauses containing literals on both $|u|$ and $|e|$, due to Theorem 4.3. Any resolution path in Φ from u or $\overline{u}$ to e or $\overline{e}$ that is implied by applying Theorem 4.3 to π' cannot contain edges labeled with $|e|$, by regularity. So such a path is also a resolution path after the transposition of u and e in the quantifier prefix. Such a resolution path in Φ' or Φ is also a resolution path at the corresponding quantifier depth (i.e., $d+1$) in Ψ. The theorem hypothesis that Definition 4.4 holds, together with Lemma 4.6, prohibits certain resolution paths that would imply that Definition 4.5 holds.

As stated, the only cases where the operation in π' might not be imitated in π are where the operation is a universal reduction on u or $\overline{u}$ in a clause D'. Let D in π correspond to D' in π'. Without loss of generality we assume that

all universals *other than* u or $\overline{u}$ have already been reduced out of D'. There are several cases to examine, to show that the problematic operations in π' can always be transformed into correct operations in π that achieve a Q-refutation of Φ. It will follow that transposing u and e does not change the evaluation of Ψ.

If D' contains u, in π' let the clause $D'_2 = \mathsf{unrd}_u(D')$. D'_2 must contain e or $\overline{e}$ or the same reduction can apply to D.

If D' contains u and D'_2 contains e, we cannot have case (B), so consider case (A). The reduced clause D'_2 must resolve on e with some clause, say C', that contains $\overline{e}$. But C' cannot contain $\overline{u}$. Let π resolve D with C, giving D_2. D_2 must be non-tautologous and now u can be reduced out, constructing a Q-refutation of Φ.

If D' contains u and D'_2 contains $\overline{e}$, neither case (A) nor case (B) is possible.

If D' contains $\overline{u}$, in π' let the clause $D'_3 = \mathsf{unrd}_{\overline{u}}(D')$. D'_3 must contain e or $\overline{e}$ or the same reduction can apply to D.

If D' contains $\overline{u}$ and D'_3 contains e, D'_3 must resolve with some clause, say C'_3, that contains $\overline{e}$. C'_3 cannot contain u in either case (A) or (B). Let π resolve D with C_3, giving D_3. D_3 must be non-tautologous and now $\overline{u}$ can be reduced out, constructing a Q-refutation of Φ.

If D' contains $\overline{u}$ and D'_3 contains $\overline{e}$, we cannot have case (A), so consider case (B). The reduced clause D'_3 must resolve with some clause, say C'_4, that contains e. But C'_4 cannot contain u. Let π resolve D with C_4, giving D_4. D_4 must be non-tautologous and now $\overline{u}$ can be reduced out, constructing a Q-refutation of Φ. ■

Corollary 4.8 If e is an existential pure literal in the matrix of a closed QBF Ψ, then e may be placed outermost in the quantifier prefix without changing the value of Ψ. If u is a universal pure literal in a closed QBF Ψ, then u may be placed innermost in the quantifier prefix without changing the value of Ψ. ■

Next we consider cases in which u and e are separated by more than one qdepth. Although it might not be sound to revise the quantifier prefix, we still might be able to perform universal reduction and other operations soundly.

Theorem 4.9 Let a closed PCNF $\Psi = \overrightarrow{Q}.\mathcal{G}$ be given in which the universal literal u is at qdepth d and the existential literals $e_1, \ldots, e_k$ are at qdepths greater than d. Let $C_0 = [\alpha, u, e_1, \ldots, e_k]$ be clause in $\mathcal{G}$, where α (possibly empty) consists of existential literals with qdepths less than d and universal literals. For each $i \in \{1, \ldots, k\}$, let $(|u|, |e_i|)$ satisfy the resolution-path independence criterion for variables (Definition 4.4). Then deleting u from C_0 does not change the truth value of Ψ. That is, universal reduction on u in C_0 is sound.

Proof: The proof idea is similar to Theorem 4.7, but is more involved because Theorem 4.3 needs to be invoked on multiple subtrees. It suffices to show that deletion of u from C_0 does not cause Ψ to change from 1 to 0. We show this holds for all assignments σ to all variables outer to u in Ψ. That is, let $\overrightarrow{Q}_{rem}$ be the suffix of $\overrightarrow{Q}$ beginning immediately after $\forall u$, and define

$$\Phi = \forall u \, \overrightarrow{Q}_{rem}.\mathcal{F}, \quad \text{where } \mathcal{F} = \mathcal{G}\lceil_\sigma \tag{5}$$

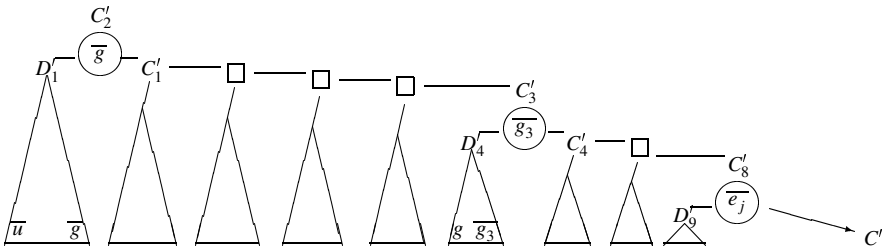

Fig. 2. Refutation π' exhibiting resolution path from $\overline{u}$ to $\overline{e_j}$ for proof of Theorem 4.9. Circles contain clashing literals of resolutions that derive clauses immediately above them.

$$\Phi' = \forall u \, \overrightarrow{Q}_{rem}. \mathcal{F}', \tag{6}$$

where $\mathcal{F}'$ is obtained from $\mathcal{F}$ by replacing clause $C = C_0 \lceil_\sigma$ by $C' = C - \{u\}$. For notation, any primed symbol (such as D') in Φ' or π' represents the corresponding unprimed symbol (such as D) in Φ or π.

Suppose Φ' evaluates to 0. By Theorem 3.1 Φ' has a regular tree-like Q-refutation, say π', which we use as a starter for π. The only operation in π' that might be incorrect for π is a resolution involving a clause C_1 in π, where $u \in C_1$, u has been reduced out of C'_1 in π', and the extra u causes the resolvent to be tautologous in π. Thus C_1 and C'_1 contain at least one of the literals e_1, ..., e_k. Also C_1 and C'_1 resolve with some clause $D_1 = D'_1$ that contains $\overline{u}$. We show this leads to a contradiction.

Figure 2 shows the proof ideas. Let the resolvent of C'_1 and D'_1 in π' be C'_2 and let the clashing literal in D'_1 be $\overline{g}$. By Theorem 4.3 there is a resolution path from $\overline{u}$ to $\overline{g}$ using (some of) the leaves of the subtree rooted at D'_1.

C provides a resolution-path from u to e_i in Φ, for each $i \in \{1, \ldots, k\}$ so to establish the contradiction, it suffices to show that there is a resolution path from $\overline{u}$ to $\overline{e_j}$, for *some* $j \in \{1, \ldots, k\}$. If g is equal to any of e_1, ..., e_k, we are done, so assume not.

Swap the order of sibling subtrees in π' as necessary to place C' on the rightmost branch, called the right spline. Find the *lowest* clause on this spline containing g. Call this clause C'_3 and call its left child D'_4. D'_4 contains g and the clashing literal used to derive C'_3, say $\overline{g_3}$. If D'_4 contains $\overline{e_j}$ for any $j \in \{1, \ldots, k\}$ rearrange its subtrees to exhibit a resolution path from g to $\overline{e_j}$ and we are done. Otherwise, rearrange its subtrees to exhibit a resolution path from g to $\overline{g_3}$, as suggested in the figure. Append this to the path from $\overline{u}$ to $\overline{g}$ (from the subtree deriving D'_1), giving a resolution path from $\overline{u}$ to $\overline{g_3}$.

Continue extending the path in this manner down the right spline. That is, let C'_5 be the *lowest* clause on this spline containing g_3 and let its left child be D'_6, etc. The figure does not show these details. Eventually, the left child of a spline clause contains some $\overline{e_j}$, shown as C'_8 in the figure. (This must occur at some point because the first resolution above C' must use some $\overline{e_j}$ as the

clashing literal.) When $\overline{e_j}$ is reached, a resolution path from $\overline{u}$ to $\overline{e_j}$ has been constructed, using the subtree that derives D'_9 for the last segment. ■

5 Clause-Literal Graphs

Let a closed QBF Ψ be given in which the quantifier block at qdepth $d+1$ is existential. We define qdepth-limited clause-literal graphs as follows:

Definition 5.1 The ***qdepth-limited clause-literal graph*** denoted as $G = ((V_0, V_1, V_2), E)$ at qdepth $d+1$ is the undirected tripartite graph in which: The vertex set V_0 consists of clauses containing some existential literal of qdepth at least $d+1$; The vertex set V_1 consists of existential positive literals of qdepth at least $d+1$ that occur in some clause in V_0. The vertex set V_2 consists of existential negative literals of qdepth at least $d+1$ that occur in some clause in V_0. The undirected edge set E consists of $(e_i, \overline{e_i})$, where $e_i \in V_1$, (e_i, C_j), where $e_i \in V_1$ and $C_j \in V_0$ and $e_i \in C_j$, and $(\overline{e_i}, C_j)$, where $\overline{e_i} \in V_2$ and $C_j \in V_0$ and $\overline{e_i} \in C_j$. See examples in Figure 3. □

Several dependency relations can be specified in terms of paths in the depth-limited clause-literal graph G. Simple paths and simple cycles in G are defined as usual for undirected graphs.

Definition 5.2 Let u be a universal literal at qdepth d and let e be an existential literal at qdepth $d+1$. A dependency pair $(|u|, |e|)$ means $|e|$ depends on $|u|$.

1. *Standard* dependencies are based on connected components.
   `stdDepA`$(|u|, |e|)$ holds if any path in G connects a clause with universal literal u or $\overline{u}$ to a clause with existential literal e or $\overline{e}$.
2. *Strict standard* dependencies are based on connected components of G.
   `ssDepA`$(|u|, |e|)$ holds if some path in G connects a clause with universal literal u to a clause with existential literal e or $\overline{e}$, and some path in G connects a clause with $\overline{u}$ to a clause with e or $\overline{e}$.
3. *Quadrangle* dependencies are based on biconnected components and articulation points of G, because they involve paths that avoid a certain literal. (Definitions are reviewed at the beginning of Section 6.) Articulation points are the only vertices that *cannot* be avoided. `quadDepA`$(|u|, |e|)$ holds if; **(A)** Some path in G connects a clause with universal literal u to a clause with existential literal e and avoids vertex $\overline{e}$; *and* **(B)** some path in G connects a clause with universal literal $\overline{u}$ to a clause with existential literal $\overline{e}$ and avoids vertex e.
   Note that u and e can independently be chosen as positive or negative literals to satisfy the above conditions (A) and (B). The name "quadrangle" is chosen because all four literals on $|u|$ and $|e|$ are involved in the requirement.
4. *Triangle* dependencies are a relaxation of Quadrangle dependencies, also based on biconnected components and articulation points of G. Specifically, `triDepA`$(|u|, |e|)$ holds under the same conditions as `quadDepA`$(|u|, |e|)$, except in condition (B) the path may start at a clause with either u or $\overline{u}$.

Table 1. QBFs for Example 5.4.

Ψ_1	$\forall u$	$\exists e$	$\forall t$	$\exists d$
C_1	u		t	$\overline{d}$
C_2	$\overline{u}$		$\overline{t}$	$\overline{d}$
C_3		e	t	d
C_4		$\overline{e}$	t	$\overline{d}$
C_5		e	$\overline{t}$	d
C_6		$\overline{e}$	$\overline{t}$	$\overline{d}$

Ψ_2	$\forall u$	$\exists e$	$\forall t$	$\exists d$
C_1	u		t	$\overline{d}$
C_2	$\overline{u}$		$\overline{t}$	$\overline{d}$
C_3		e	t	d
C_4		$\overline{e}$	t	$\overline{d}$
C_7		$\overline{e}$	$\overline{t}$	d
C_6		$\overline{e}$	$\overline{t}$	$\overline{d}$

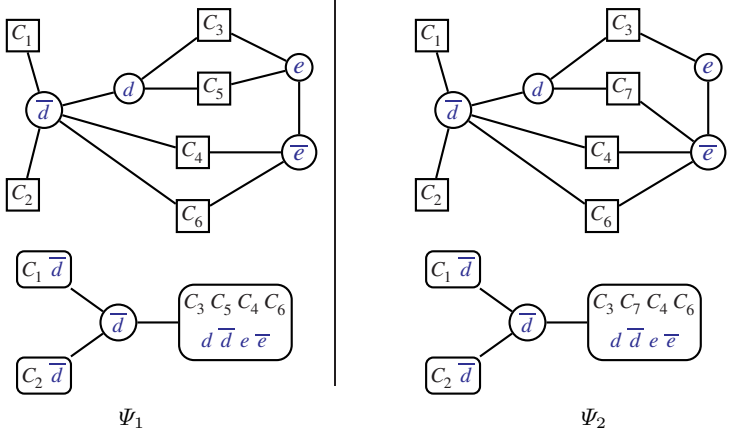

Fig. 3. (Above) Clause-literal graphs for Example 5.4. (Below) BCC-based block trees.

5. Paths for *resolution-path* dependencies, denoted by `rpDepA`$(|u|, |e|)$, are further restricted from those for quadrangle dependencies. Restrictions on paths are as follows: **(C)** If a path arrives at a literal node from a clause node, its next step must be to the complement literal. **(D)** If a path arrives at a literal node from its complement literal node, its next step must be to a clause node.
   If a path goes from C_1 to literal q, then to C_2, then both C_1 and C_2 contain q. This path is allowed for triangle and quadrangle dependencies, but not for resolution-path dependencies. □

Curiously, strict standard dependencies relax quadrangle dependencies in the opposite way from triangle dependencies. The motivation for strict standard dependencies is that they seem to be more efficient to compute than quadrangle dependencies, as discussed later.

Theorem 4.7 implies the following:

Corollary 5.3 With the preceding notation: **(1)** If the universal variable u at qdepth d has *no* tuple $(u, e) \in$ `quadDepA` such that the qdepth of e is less than $d + 2k$, where $k > 0$, then u can be placed at qdepth $d + 2k$ in the quantifier prefix without changing the value of Ψ. **(2)** If existential variable e at qdepth

$d+1$ has *no* tuple $(u, e) \in$ `quadDepA` such that the qdepth of u is greater than $d-2k$, where $k > 0$, then e can be placed at qdepth $d+1-2k$ in the quantifier prefix without changing the value of Ψ. ■

Example 5.4 This example illustrates resolution-path dependencies, quadrangle dependencies, and their differences, with reference to various graph structures. Consider the closed QBFs Ψ_1 and Ψ_2, given in chart form in Table 1. In the following, the notation "$C_1(u)$" abbreviates the phrase "C_1, which contains the literal u," etc., and does not represent any operation on C_1.

In both formulas a quadrangle dependency `quadDepA`$(|u|, |e|)$ is established by the paths $C_1(u) \stackrel{|d|}{\rightarrow} C_4(\overline{e})$ and $C_2(\overline{u}) \stackrel{|d|}{\rightarrow} C_3(e)$. However, the first path is not a resolution path because d does not occur with opposite signs in C_1 and C_4. Indeed, in Ψ_1 neither u nor $\overline{u}$ presses on $\overline{e}$ by any resolution path, recalling that the universal t cannot be used for connection. Therefore e is independent of u based on `rpDepA`. It follows that u and e may be exchanged in the quantifier prefix without decreasing the value of Ψ_1 (and such a swap can never increase the value). Following this exchange, it is easy to see that u may be exchanged with t, then with d, and universally reduced out of all clauses.

Observe that Ψ_2 is the same as Ψ_1 except that it has C_7 instead of C_5. There is no obvious difference in the chart appearance, but now $C_1(u) \stackrel{|d|}{\rightarrow} C_7(\overline{e})$ is a resolution path and `rpDepA`$(|u|, |e|)$ holds in Ψ_2, so transposing u and e in the quantifier prefix is unsafe by this criterion.

The role of the block trees is explained in Section 6, in connection with biconnected components and articulation points of the clause-literal graph. The definitions are reviewed at the beginning of that section. Here we just note that the circular node is an articulation point and the rounded rectangular nodes are biconnected components. □

6 Finding Dependency-Related Paths

Now we turn to the issue of computing `quadDepA`. Biconnected components play a central role. After reviewing the standard theory, this section describes how the specific information needed for quadrangle dependencies is extracted.

Recall that a subgraph, say B, of an undirected graph G is biconnected if and only if removing any one vertex and all edges incident upon that vertex does not disconnect the remaining subgraph. A ***biconnected component*** **(BCC)** of any undirected graph G is a *maximal* biconnected subgraph of G.

Each edge of G is in exactly one BCC. Also, two BCCs have at most one vertex in common. A vertex that is in more than one BCC is called an ***articulation point*** **(AP)**. Removal of an articulation point increases the number of connected components in G.

The BCCs and APs of the depth-limited clause-literal graph G can be found in time linear in its size. The code in [1, Fig. 7.26] avoids putting edges redundantly into the BCCs.

As a by-product, the BCC algorithm can determine simple connected components (CCs). An additional by-product of this algorithm is the creation of an acyclic undirected bipartite graph associated with each CC, called the ***block tree***, in which the BCCs are collapsed to single vertices and are separated by the APs (see Figure 3). All universal literals incident upon each BCC can be collected, as well.

We continue with the terminology of Definition 5.1 for G, d, u, e, etc. It is easy to determine if there is a path in G between some clause containing u or $\overline{u}$ and a literal e in V_1: just check if one of those clauses is in the same CC as e. Since e and $\overline{e}$ are always in the same CC, the same clauses can reach $\overline{e}$. However, the triangle and quadrangle dependency relations require paths to e and $\overline{e}$ that avoid the complement literal. If neither e nor $\overline{e}$ is an AP of G, both of these paths must exist. In this case, the relevant universal literals for $|e|$ are just those that occur in some clause in the same CC as e. These sets of universal literals can be collected once, during the BCC algorithm.

Now suppose e or $\overline{e}$ or both are APs of G. The relevant universal literals for e can be found by starting a graph search of the block tree containing $|e|$, from e, and avoiding a visit of $\overline{e}$. The relevant universal literals for $\overline{e}$ can be found by starting a graph search of the block tree containing $|e|$, from $\overline{e}$, and avoiding a visit of e. As each BCC is visited, any universal literals at qdepth d can be collected. It appears that adapting this approach to compute triangle dependencies instead of quadrangle dependencies will not save much time. Details are omitted for lack of space, but are straightforward.

At this time, the question of whether resolution-path dependencies can be computed in polynomial time is open. We conjecture that it is possible, but the requirement that two consecutive edge labels in the resolution graph cannot be the same makes it difficult.

7 Empirical Data

A prototype program was implemented in `C++` with the Standard Template Library to gauge the amount of variable independence that might be found by various dependency relations.[2] The program computes dependency-related quantities on QBF benchmarks. It was run on the 568 QBFEVAL-10 benchmarks. Two benchmarks had no universal variables, so the tables include data on 566 benchmarks. The platform was a 2.6 GHz 64-bit processor with 16 GB of RAM, Linux OS.

The computation was limited to the outermost universal block and the adjacent enclosed existential block. The number of "trivial dependencies" is simply the product of the sizes of these two blocks. The primary purpose of the program is to find out the relative sizes of the relations for standard dependencies, strict standard dependencies, and quadrangle dependencies. Only the outermost block pair is analyzed because this provides a direct comparison between standard

[2] Please see `http://www.cse.ucsc.edu/~avg/QBFdeps/` for the prototype program.

Table 2. Eight largest QBFEVAL-10 benchmarks.

			Fraction of Trivial		
	Trivial			Strict	
Benchmark	(000,000)	CCs	Standard	Standard	Quadrangle
s3330_d10_u-shuffled	627	1	1	1	0.000076
s3330_d4_s-shuffled	68	1	1	1	0.000069
s499_d15_s-shuffled	15	1	1	1	0.000125
s510_d12_s-shuffled	16	1	1	1	0.000028
s510_d31_s-shuffled	122	1	1	1	0.000082
szymanski-24-s-shuffled	1293	1	1	1	0.001944
vonNeumann-rip...-13-c-	278	1	0.999999	0.999999	0.992812
vonNeumann-rip...-15-c-	627	1	0.999999	0.999999	0.993727

Table 3. Dependency fractions as unweighted ratios.

		Average		Avg. Fraction of Trivial		
Benchmark	Num. in	Trivial	Avg.		Strict	
Group	Group	(000)	CCs	Standard	Standard	Quadrangle
Eight Largest	8	219363	1	1.0000	1.0000	0.2486
Str.Std. Helped	239	158	34.6	0.3722	0.3718	0.1928
Str.Std. No Help	319	359	9.8	1	1	0.7278

dependencies and quadrangle dependencies. Including multiple blocks would obscure the size relationships because standard dependencies use transitive closure when multiple blocks are involved, while quadrangle dependencies do not.

The benchmarks were partitioned into several groups to try to make the statistics more informative. Table 2 shows data for the eight largest benchmarks, as measured by the number of trivial dependencies. For six of these benchmarks, the Quadrangle relation is 3-5 orders of magnitude smaller than the Trivial, while the Strict Standard gives no reduction. On two others, no relation gives reduction.

Table 3 shows the eight largest as a group, and separate the remaining benchmarks according to whether Strict Standard Dependencies gave any reduction at all. Quadrangle dependencies give substantial additional reductions, beyond standard and strict standard dependencies. Although Strict Standard gave very little improvements in this test, they are essentially free, once the overhead of Standard has been incurred.

A serious question is whether the time needed to compute Quadrangle Dependencies pays back in more efficient solving. Experience with `depqbf` indicates tentatively that Standard Dependencies pay back in the long run [5]. For the 566 runs to get these statistics, the three longest runs took 75628, 2354, and 1561 seconds. The average of the remaining 563 runs was 9.40 seconds. Only finding the Strict Standard dependencies and the BCCs averaged 0.50 seconds on all 566 instances.

Concerning the three longest runs, two of these instances have never been solved by any solver, so in a sense, nothing has been lost. However, the third

instance, `szymanski-24-s-shuffled`, is not considered exceptionally difficult. It took 75628 seconds to find the quadrangle dependencies, yet finding the BBCs took only three seconds, and computing the Standard Dependencies took only four additional seconds. We do not have an explanation for this outlier behavior.

8 Conclusion

This paper analyzes several new dependency relations for QBF solving, and shows they form a hierarchy, together with the standard and triangle relations proposed by Samer and Szeider. The root of the hierarchy and strongest for detecting variable *independence* is the resolution-path dependency relation. Its soundness is proved; soundness of supersets (more restrictive relations) is a corollary. Whether the resolution-path relation has an efficient implementation is an open question, so quadrangle dependencies, the next relation down in the lattice (Figure 1), were studied in more detail. Computational methods for quadrangle dependencies are described, using the theory of biconnected components, and a prototype was implemented to gauge the sizes of BCCs and related structures in benchmarks.

Future work includes a trial implementation of quadrangle dependencies in a QBF solver, but the publicly available solvers we looked at are not good candidates for such a retrofit by anyone except one of the original programmers, in most cases because the source code is *not* public. The few with public source code tend to lack documentation and contain numerous short-cuts to improve solver speed. Also, there are numerous ways to use dependencies, so one implementation experience will not be definitive.

Acknowledgment We thank Florian Lonsing and Armin Biere for many helpful email discussions. We thank the anonymous reviewers for helpful comments.

References

1. Baase, S., Van Gelder, A.: Computer Algorithms: Introduction to Design and Analysis. Addison-Wesley, 3rd edn. (2000)
2. Kleine Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified boolean formulas. Information and Computation 117, 12–18 (1995)
3. Kleine Büning, H., Lettmann, T.: Propositional Logic: Deduction and Algorithms. Cambridge University Press (1999)
4. Klieber, W., Sapra, S., Gao, S., Clarke, E.: A non-prenex, non-clausal QBF solver with game-state learning. In: Proc. SAT, LNCS (2010)
5. Lonsing, F., Biere, A.: Integrating dependency schemes in search-based QBF solvers. In: Proc. SAT. pp. 158–171. Springer (2010)
6. Samer, M., Szeider, S.: Backdoor sets of quantified boolean formulas. J. Automated Reasoning 42, 77–97 (2009)
7. Yates, R.A., Raphael, B., Hart, T.P.: Resolution graphs. Artificial Intelligence 1, 257–289 (1970)